NTNU
Norwegian University of
Science and Technology

Master's thesis

2020

Stine Gustavsen

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Natural Sciences
Department of Physics

Stine Gustavsen

Development and evaluation of a radiochromic film dosimetry program

Application on stereotactic columna radiotherapy

August 2020

NTNU
Norwegian University of
Science and Technology

Development and evaluation of a radiochromic film dosimetry program

Application on stereotactic columna radiotherapy

Stine Gustavsen

Applied Physics and Mathematics - Biophysics and medical technology
Submission date: August 2020
Supervisor: Jomar Frengen
Co-supervisor: Signe Danielsen

Norwegian University of Science and Technology
Department of Physics

Preface

The work during this master thesis have been conducted at St. Olavs Hospital during spring 2020, under the guidance of medical physicist Jomar Frengen and head of Department, education and research, at the cancer clinic at St. Olavs Hospital and associate at the Department of Physics Signe Danielsen. During the timeline of this work the worldwide pandemic, Covid-19, struck forcing all experimental work to rest. Because of this the basis for my master thesis had to change, going from an experimental assignment to focusing more on creating a software as well as the literature of the field. I would like to give a big thank you to both Jomar Frengen and Signe Danielsen for their substantial help and support during this work, and for being very helpful in adjusting the basis for my master thesis to the situation. In addition, I would like to thank product designer Lucas Cueni for helpful tips on the design of the software Fidora, and Therese Have Gustavsen for helping me proofreading.

Abstract

As the field of radiotherapy is constantly evolving new techniques are developed. One such method is stereotactic radiotherapy, which is the delivery of high, precise doses. One of the challenges during the work with stereotactic treatment planning is the dosimetry. As the most common dosimeters have relatively large spatial extent, their resolution is too low to give good measurements of the steep gradients typical for stereotactic radiotherapy. The GafChromic EBT3 film is a dosimeter offering a 2D, continuous readout, and has proved as a reliable dosimeter for use in stereotactic radiotherapy. As such, there has been a need for an analysing tool to process the measurements done by the film. In this work such an analysing tool was developed, named Fidora. In Fidora the user can perform background corrections and calibrations as well as investigating profiles and dose volume histograms.

There are two parts to this work, the first being the development of Fidora with all its functionalities. The second part has been to study the MLC model in the treatment planning system RayStation and four different stereotactic treatment plans using Fidora. The last part of the work has been done as a proof of concept to analyse how well Fidora performs. It has been found that Fidora is a reliable analysing tool, proven to be able to discover deviation in measured dose compared to planned dose plans as well as comparing different treatment plans.

Sammendrag

Stråleterapifaget er stadig under utvikling og med det utvikles det nye teknikker. En slik metode er stereotaktisk strålingsbehandling, hvor det er typisk med høye, presise doser. En av utfordringene med behandlingsplanlegging innen stereotaksi er dosimetrien. Ettersom de mest brukte dosimetrene har relativt store målevolum, som resulterer i lav oppløsning, gir de en for dårlig måling av de bratte gradientene som er typiske i stereotaksi. GafChromic EBT3 film er et dosimeter som tilbyr 2D, kontinuerlig målinger, og har bevist å være et pålitelig dosimeter for bruk innen stereotaktisk strålebehandling. Med det har det kommet et behov for et analyseverktøy for å prosessere målingene gjort med filmen. I dette arbeidet har et slikt analyseringsverktøy blitt utviklet og fått navn Fidora. Med Fidora har brukeren mulighet til å utføre bakgrunnskorrigering og kalibrering i tillegg til å studere profiler og dose volum histogrammer.

Det er to deler av dette arbeidet, hvor den første delen er utviklingen av Fidora og alle dens funksjonaliteter. Den andre delen har vært å studere MLC modellen i behandlingsplanleggingssystemet RayStation og fire stereotaktiske behandlingsplaner ved hjelp av Fidora. Den siste delen av arbeidet har fungert som et ”bevis av konsept” for å analysere hvor godt Fidora presterer. Resultatene har vist at Fidora er et pålitelig analyseringsverktøy, og har bevist at det fungerer godt i å oppdage avvik i målte doser sammenlignet med planlagte doseplaner og også til sammenligning av forskjellige behandlingsplaner.

Table of Contents

Preface **1**

Summary **i**

Summary **i**

Table of Contents **v**

List of Tables **viii**

List of Figures **xiii**

Abbreviations **xiv**

1 Introduction **1**

2 Background **3**
- 2.1 Interaction of radiation 3
  - 2.1.1 Ionizing radiation 3
  - 2.1.2 KERMA 8
  - 2.1.3 Dosimetry 8
- 2.2 Introduction to radiotherapy 11
  - 2.2.1 Radiobiology 11
  - 2.2.2 The five r's of radiotherapy 12
  - 2.2.3 Fractionation 14
- 2.3 Dose measurements 14
  - 2.3.1 Cavity theroy 15
  - 2.3.2 Ionization chamber 16
  - 2.3.3 Radiochromic film 16
- 2.4 Radiation treatment techniques 18
  - 2.4.1 Standardized volumes used in radiotherapy 18
  - 2.4.2 Planning CT 19

2.4.3 Linear accelerator 19
2.4.4 Multi-leaf collimators 21
2.4.5 Dose calculation algorithms 23
2.4.6 Treatment planning 24
2.4.7 Stereotactic radiotherapy 27
2.4.8 Dosimetry and dose measurements in SRT 28
2.5 Python 30
2.5.1 Algorithms 30

3 Materials and Method **33**
3.1 The basis of Fidora 33
3.1.1 General specification made by Fidora 34
3.1.2 How Fidora reads a scanned image 39
3.1.3 How Fidora reads a doseplan 39
3.2 Fuctionalities of Fidora 39
3.2.1 CoMet 39
3.2.2 Dose response 42
3.2.3 Profiles 46
3.2.4 Dose volume histogram (DVH) 49
3.3 Stereotactic radiotherapy 51
3.3.1 MLC model in Raystation 51
3.3.2 Stereotactic treatment plans 53

4 Results **57**
4.1 Fidora 57
4.1.1 CoMet 58
4.1.2 Dose Response 62
4.1.3 Profiles 66
4.1.4 Dose volume histogram 69
4.2 MLC model in RayStation 71
4.3 Stereotactic treatment plans 80
4.3.1 Treatment plan V1 80
4.3.2 Treatment plan V2 83
4.3.3 Treatment plan V3 85
4.3.4 Treatment plan V4 88

5 Discussion **91**
5.1 Fidora 91
5.1.1 CoMet 91
5.1.2 Dose Response 92
5.1.3 Profiles 92
5.1.4 DVH 93
5.1.5 MLC models in Raystation 93
5.1.6 Stereotactic treatment 95
5.1.7 Further work in Fidora 98

6 Conclusion **101**

Bibliography **101**

Appendix **107**

List of Tables

3.1 Specifications during treatment planning when using Fidora. 34
3.2 Specifications during irradiation when using Fidora. 36
3.3 Scanning specifications used in Espon scan app. 38
3.4 Specifications for treatment plans V1-V8. Arc rotation direction is in all cases counterclockwise. 54

4.1 Standard deviation at each measured point when the difference map is averaged over all six dose levels. The standard deviations are given for the profile across the scanner lateral direction as well as the scanners scanning direction. 60
4.2 Total uncertainty related to the fitting of the dose response curves. These numbers will depend on how many calibration points are used. 66
4.3 Directed Hausdorff distance showing the largest difference in dose along each curve. 66
4.4 Dose measured in the centre of the x-profile as the film has been filtered using a 5-pixel median filter. The centre is measured as an average over 5mm around centre point. 73
4.5 Dose measured in the centre of the x-profile as the film has not been filtered. The centre is measured as an average over 5mm around centre point. 73
4.6 Dose measured in the centre of the x-profile as the film has been filtered using a 15-pixel median filter. The centre is measured as an average over 5mm around centre point. 73
4.7 Left and right penumbra measured in x-profile as the film has been filtered using a 5-pixel median filter. The penumbra is measured between 20% and 80% of max dose. 74
4.8 Left and right penumbra measured in x-profile as the film has not been filtered. The penumbra is measured between 20% and 80% of max dose. . 74
4.9 Left and right penumbra measured in x-profile as the film has been filtered using a 15-pixel median filter. The penumbra is measured between 20% and 80% of max dose. 74

4.10 Measurements of the field size in both film and dose plan. The field size is defined by the limits of 50% of centre dose in x-profile. Here the film has been filtered using a 5-pixel median filter. 75
4.11 Dose measured in the centre of the y-profile as the film has been filtered using a 5-pixel median filter. The centre is measured as an average over 5mm around centre point. 76
4.12 Dose measured in the centre of the y-profile as the film has not been filtered. The centre is measured as an average over 5mm around centre point. 76
4.13 Dose measured in the centre of the y-profile as the film has been filtered using a 15-pixel median filter. The centre is measured as an average over 5mm around centre point. 76
4.14 Left and right penumbra measured in y-profile as the film has been filtered using a 5-pixel median filter. The penumbra is measured between 20% and 80% of max dose. 77
4.15 Left and right penumbra measured in y-profile as the film has not been filtered. The penumbra is measured between 20% and 80% of max dose. . 77
4.16 Left and right penumbra measured in y-profile as the film has been filtered using a 15-pixel median filter. The penumbra is measured between 20% and 80% of max dose. 77
4.17 Measurements of the field size in both film and dose plan. The field size is defined by the limits of 50% of centre dose in y-profile. Here the film has been filtered using a 5-pixel median filter. 78
4.18 Measurements of the penumbra (defined between 20% and 80% of max dose) and the length of the peak. The profile is drawn across the side of the leaf in the MLC. 79
4.19 Measurements of the penumbra (defined between 20% and 80% of max dose) and the length of the peak. The profile is drawn across the tip of the leaf in the MLC 79
4.20 Measurements from profiles and dose volume histogram for treatment plan V1 83
4.21 Measurements from profiles and dose volume histogram for treatment plan V2 85
4.22 Measurements from profiles and dose volume histogram for treatment plan V3 88
4.23 Measurements from profiles and dose volume histogram for treatment plan V4 90

List of Figures

2.1 Illustration of photoelectric effect. 4
2.2 Illustration of Compton scattering. 5
2.3 An illustration of fluence in a radiation beam. 7
2.4 Curve showing the relationship between collision KERMA and dose. . . . 9
2.5 An illustration of a beam profile taken at different depths in a phantom. . 10
2.6 The cell cycle showing phase order and the typical relative duration. . . . 12
2.7 An illustration of a ionization chamber. 16
2.8 An illustration of the layers in GacChromic EBT3. 17
2.9 An illustration of the volume definitions in radiation therapy. 18
2.10 An illustration of the main components of the linac. 20
2.11 An illustration of the beam limiting device 20
2.12 An illustration of the isocenter in the treatment room. 21
2.13 An illustration of the MLC leaves. 22
2.14 An illustration of the tongue and groove in MLC leaves. 22
2.15 The concept of calculating absorbed dose using TERMA and dose kernels. 23
2.16 An illustration of geometrical offset between field being defined by the MLC leaf tip and side. 25
2.17 The leaf-tip width of an MLC leaf. 26
2.18 An illustration of a dose volume histogram. 27
2.19 An illustration of the changes in output resulting from overlapping penumbra. 28

3.1 Flow chart of Fidora showing its main functionalities. 33
3.2 An illustration of the patient coordinate system. 34
3.3 An example of the dose matrix orientation in the patient coordinate system. 35
3.4 An illustration of the different patient positions. 35
3.5 Examples of how the film can be placed according to the dose matrix. The film can be placed in any slice. The green illustrates the film, and the arrows shows the principal axes. 36
3.6 An illustration of the three film orientations, sagittal, axial and coronal. . 36

3.7 An example of the markings made on a film to find isocenter. The black marks (shown by the red arrows) on the film marks the directional lines going towards isocenter (shown by the blue dot). 37
3.8 An illustration of the scanning direction and orientation. 38
3.9 An image of the setup during scanning of film pieces meant for background correction. 40
3.10 Flowchart showing the main steps in CoMet. 41
3.11 Setup en experiment verifying the correction. 42
3.12 Flowchart showing the main steps in the tab Dose response. 43
3.13 An example of how to find the center of the scanner. 44
3.14 Scanner setup when doing a calibration. 45
3.15 An illustration of the effects of a flattening filter. Red line is filter free and blue is the beam profile when using a flattening filter. The yellow dashed line indicates the profile corresponding to the size of the film pieces used in the experiment. 46
3.16 Example of the profiles and table with information. 47
3.17 Flow chart showing the main steps in the tab Profiles. 48
3.18 Flow chart showing the main steps in the tab DVH 50
3.19 Setup in MLC model experiment 51
3.20 An illustration of how the MLC were used to shape the partially blocked 10cm$x$10cm field. 52
3.21 An illustration of how the MLC were used to shape the partially blocked 10cm$x$10cm field. The red lines indicate where the profiles were taken. . 52
3.22 Phantom showing the different volumes of interest. 53
3.23 Image from the setup in the linac. 55
3.24 Setup during scanning of the film. 55
3.25 Illustration of how the profiles were drawin in the treatment plans. The red lines indicate the positions of the profiles. In V1-V4 all three profiles (a, b and c) were plotted, while in treatment plans V5-V8 only a and c were plotted. 56

4.1 Initial page when opening the software Fidora. The tabs on the left hold the different functionalities included in Fidora. 58
4.2 Profile of the difference in read pixel value in relation to the centre value. The measurements have been taken across the scanners lateral direction and are plotted with error bars for each dose level. 58
4.3 Profile of the difference in read pixel value relation to the centre value. The measurements have been taken across the scanners scanning direction and are plotted with error bars for each dose level. 59
4.4 Profile of the difference in read pixel value in relation to the centre value as an average over all six dose levels. The measurements have been taken across the scanners lateral direction and are plotted with error bars at each measuring point. 59

4.5 Profile of the difference in read pixel value in relation to the centre value as an average over all six dose levels. The measurements have been taken across the scanners scanning direction and are plotted with error bars at each measuring point. 60
4.6 Profiles from a corrected image and a non-corrected image shown in green and red, respectively. The profiles are taken in the scanner's lateral direction across the centre of the scanner surface. 61
4.7 Screenshot of the first tab in Fidora - CoMet. In this module the user is able to perform background corrections on a scanned image of the film. . 61
4.8 Screenshot of the first tab in Fidora, CoMet, after a successful run. The corrected image here is used as an example. 62
4.9 Screenshot of the second tab in Fidora - Dose response. In this module the user are able to perform a calibration as well as investigating the dose response curve, which will be plotted on the screen. 63
4.10 Screenshot of the window where the user uploads images of scanned films. The left image shows how the window looks at the beginning and the right image shows how it look when files have been uploaded. 63
4.11 Screenshot of how the tab Dose Response looks like after enough measurements have been uploaded and the dose response has been fitted. . . . 64
4.12 Screenshot of the window where the user can save the performed calibration for later use in Fidora. 64
4.13 Plot showing the dose response curves both when calibration is done in Fidora and manual calibration, with fitting uncertainty ± 0.07Gy and ± 0.02Gy respectively. 65
4.14 Plot showing the dose response curves both when calibration is done in Fidora and manual calibration using a flattening filter free beam, with fitting uncertainty ± 0.06Gy and ± 0.02Gy respectively. 65
4.15 Plot showing the dose response curves created in Fidora for one flattening filter free beam and one beam using the filter, with fitting uncertainty ± 0.06Gy and ± 0.07Gy respectively. 66
4.16 Screenshot of the fourth tab in Fidora - Profiles. In this module the user can upload scanned images of film to study horizontal, vertical or manually drawn profiles and compare the results in film and dose plan. 67
4.17 Screenshot of the daughter window where user marks isocenter/reference point and ROI. This is a pop-up window showing as the film has been uploaded. 68
4.18 Screenshot of the Profiles tab after film and treatment plan has been uploaded and profiles drawn. Here the user can make adjustments to the placement of the film to make up for positioning errors. 69
4.19 Screenshot of the fifth tab in Fidora - DVH. In this module the user can upload scanned images of film to study the dose volume histogram and compare the results in film and dose plan. 70
4.20 Screenshot of the fifth tab in Fidora - DVH to illustrate how it looks after a successful run. 71

4.21 X-profiles from 1cm$x$1cm, 2cm$x$2cm, 3cm$x$3cm, 5cm$x$5cm and 10cm$x$10cm fields. The film is represented with the red curve, while the dose plan is represented with a blue curve. 72

4.22 Y-profiles from 1cm$x$1cm, 2cm$x$2cm, 3cm$x$3cm, 5cm$x$5cm and 10cm$x$10cm fields. The film is represented with the red curve, while the dose plan is represented with a blue curve. 75

4.23 Horizontal profile of a 10cm$x$10cm field with one quadrant blocked out by the MLC leaves. The penumbra is due to blocking from the MLC side. The drawing at the right illustrates where the profile was taken. 78

4.24 Vertical profile of a 10cm$x$10cm field with one quadrant blocked out by the MLC leaves. The penumbra is due to blocking from the MLC tip. The drawing at the right illustrates where the profile was taken. 79

4.25 Phantom showing the different volumes of interest. 80

4.26 Profiles from treatment plan V1. Their number correspond to the numbered lines in the last image. The green, vertical lines in profile (b) illustrates the placement of the spinal cord, while the horizontal, green line indicates dose 18Gy which comes from one of the parameters in the optimization of the treatment plan. 81

4.27 Profiles from treatment plan V5 for comparison at the high dose areas with treatment plan V1 shown in Figure 4.26. Plots (1) and (2) corresponds to plots (a) and (c) in Figure 4.26 respectively. 81

4.28 Dose volume histogram of treatment plan V1 for both film and dose plan. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.26. 82

4.29 Dose volume histogram of treatment plan V1 for both film and dose plan showing only the curve for spinal cord and spinal cord PRV. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.26. 82

4.30 Profiles from treatment plan V2. Their number correspond to the numbered lines in the last image. The green, vertical lines in profile (b) illustrates the placement of the spinal cord, while the horizontal, green line indicates dose 18Gy which comes from one of the parameters in the optimization of the treatment plan. 83

4.31 Profiles from treatment plan V6 for comparison at the high dose areas with treatment plan V2 shown in Figure 4.30. Plots (1) and (2) corresponds to plots (a) and (c) in Figure 4.30 respectively. 84

4.32 Dose volume histogram of treatment plan V2 for both film and dose plan. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.30. 84

4.33 Dose volume histogram of treatment plan V2 for both film and dose plan showing only the curve for spinal cord and spinal cord PRV. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.30. 85

4.34 Profiles from treatment plan V3. Their number correspond to the numbered lines in the last image. The green, vertical lines in profile (b) illustrates the placement of the spinal cord, while the horizontal, green line indicates dose 18Gy which comes from one of the parameters in the optimization of the treatment plan. 86
4.35 Profiles from treatment plan V7 for comparison at the high dose areas with treatment plan V3 shown in Figure 4.34. Plots (1) and (2) corresponds to plots (a) and (c) in Figure 4.34 respectively. 86
4.36 Dose volume histogram of treatment plan V3 for both film and dose plan. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.34. 87
4.37 Dose volume histogram of treatment plan V3 for both film and dose plan showing only the curve for spinal cord and spinal cord PRV. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.34. 87
4.38 Profiles from treatment plan V4. Their number correspond to the numbered lines in the last image. The green, vertical lines in profile (b) illustrates the placement of the spinal cord, while the horizontal, green line indicates dose 18Gy which comes from one of the parameters in the optimization of the treatment plan. 88
4.39 Profiles from treatment plan V8 for comparison at the high dose areas with treatment plan V4 shown in Figure 4.38. Plots (1) and (2) corresponds to plots (a) and (c) in Figure 4.38 respectively. 89
4.40 Dose volume histogram of treatment plan V4 for both film and dose plan. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.38. 89
4.41 Dose volume histogram of treatment plan V4 for both film and dose plan showing only the curve for spinal cord and spinal cord PRV. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.38. 90

Abbreviations

Symbol	=	definition
IMRT	=	Intensity-modulated radiotherapy
VMAT	=	Volumetric Modulated Arc Therapy
MLC	=	Multi-leaf collimators
ROC	=	Radius of curvature
CC	=	Collapsed cone
MC	=	Monte Carlo
SABR	=	Stereotactic ablative radiotherapy
CT	=	Computer Tomography
ICRU	=	International Commission on Radiation Units and Measurments
CTV	=	Clinical target volume
PTV	=	Planning target volume
GTV	=	Gross tumor volume
OAR	=	Organ at risk
IAEA	=	International Atomic Energy Agency
FWHM	=	Full width at half maximum
KERMA	=	Kinetic energy released per unit mass
CEMA	=	Converted energy per unit mass
LET	=	Linear energy transfer
TCP	=	Tumor control probability
NTCP	=	Normal tissue complication probability
CCC	=	Collapsed cone classic
CCE	=	Collapsed cone enhances
MC	=	Monte Carlo
DVH	=	Dose volume histogram

Chapter 1

Introduction

Lung cancer, together with other cancer types like tumors near the spinal cord, in the brain or lymph nodes are difficult to treat with conventional radiotherapy treatment as these types of cancers are usually small and near organs of risk (Benedict et al. (2010)). Because of this a treatment technique called stereotactic radiotherapy was developed. In this kind of treatment, the tumor is irradiated using a small, but intense, radiation field, usually from many different angles. This results in a high dose delivered to the tumor while the surrounding tissue only receives a small amount of radiation, which gives better tumor control and at the same time limits the late effects of radiation. In stereotactic radiotherapy the accuracy of the delivery is very important, as small deviations can have great consequences for the patient and in the worst case kill cells in organs of risk. As such, careful validation of stereotactic treatment plans are necessary and proper dosimeters must be used in the dosimetry. One of the characteristics of stereotactic radiotherapy is heterogeneous dose distributions with steep gradients. Therefore, the dosimetry must be performed using dosimeters with resolution high enough to read the steep gradient over small distances. Commonly used dosimeters such as ionization chambers and diodes suffer from the lack of resolution as they have a fairly large spatial extent. At the beginning of 2000 radiochromic film made its entry as a dosimeter offering a continuous 2D measurement of radiation fields. Unfortunately, the second generation of the film proved to have a low accuracy which resulted in a lack of interest in radiochromic film as a dosimeter. However, in 2011 a third generation was released having shown a much greater accuracy (van Battum (2018)) and the interest of using radiochromic film has been rising, especially in stereotactic radiotherapy. In this work the GafChromic EBT3 film has been used as a dosimeter when studying stereotactic treatment plans.

Because GafChromic film is not a normally used dosimeter at the radiation clinic at St. Olavs Hospital there was a need to develop analysing tools. Therefor the work of this master thesis was to develop a software, named Fidora, with different functionalities aimed at film as a dosimeter. The programme is written in collaboration with another master student, Ane Vigre Håland. The programming language chosen for this task was Python as this is an open source language and relatively easy manageable independent on the programming background of the user. The aim of Fidora was to develop an open source alternative to other licensed and limited versions available for purchase. In that way it is possible to do further developments and alternations if needed and add more functionality. As it is an open source software anyone can clone the program and adapt it to their needs, but to do alterations to the original software a request to do so is needed. During this work Fidora will be developed to perform calibrations of the film, measuring and comparing profiles in both film and dose plan and studying dose volume histograms measured by the film. In addition, since radiochromic film must be scanned and the chosen table scanner, Epson v750 pro, suffers from a non-uniform readout over the scanner surface, Fidora will offer the possibility to perform background corrections.

As a proof of concept, several stereotactic treatment plans will be analysed using Fidora. As mentioned above, steep gradients are typical in stereotactic treatments and since these gradients will be defined by the jaws and multi-leaf collimator in the linear accelerator, the modelling of such features in the treatment planning systems are critical. Therefore, an experiment to validate the modelling of these in RayStation, the treatment planning system used at St. Olavs Hospital, will be studied.

Chapter 2

Background

2.1 Interaction of radiation

The x-rays were discovered in 1895 by Wilhelm Conrad Röntgen and used in medical practise. It has later been widely used in various medical disciplines and the need for protection when working with x-rays has become important. To be able to predict biological effects and reproduce in clinical cases the concept of dose has been introduced. There are many physical quantities created to describe the dose of radiation and beams of radiation. In the field of radiotherapy photons and electrons are most used, and in Norway today they are used exclusively even though proton treatment is making an entrance in the coming years. Where others are not mentioned the reference used in this section is Podgorsak et al. (2005) and Bourland (2016).

2.1.1 Ionizing radiation

When foreseeing how photons and electrons interact when entering matter, one assumes a stochastic behaviour and the estimate must always be thought of as a probability estimate. Even though this is true for both photons and electrons the behaviour of each of them is very different. While photons interact only a very few times and are attenuated exponentially, electrons interact a large number of times and deposits its energy over its entire track until it stops.

Photons

When describing photon interaction with matter one talks about cross section, σ, and attenuation coefficient, μ, both being target matter specific. Cross section refers to the cross-sectional area of the target and the attenuation coefficient describes the probability for interaction to take place when traversing the specific matter. Then the radiation intensity at depth x is given as

$$I(x) = I_0 e^{-x\mu}, \tag{2.1}$$

where I_0 is the initial intensity. The interaction between photons and matter can be categorized into three mechanisms, photoelectric effect, scattering and pair production, each associated with a specific cross section. In radiotherapy the target is DNA and therefore the interactions are described with atoms, meaning that cross section refers to atomic cross section. If the energy of the incoming photon is larger than the binding energy of one or more of the electrons in the atom one way for the radiation to interact is through photoelectric effect. In this case the incoming photon interacts with the atom and is fully absorbed while ejecting one electron from the inner orbital in the atom, see Figure 2.1.

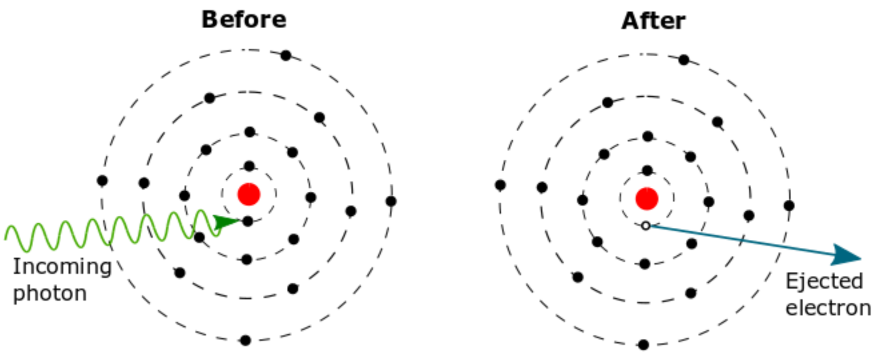


Figure 2.1: Illustration of photoelectric effect.

As the electron is being ejected a vacancy in the inner orbital is left, needed to be filled by one of the electrons in the outer shells. As an outer electron fills that vacancy it will move to a lower energy state and by that release an energy amount equal to the energy difference in the two orbitals. One of two things can come from such an energy release, either the energy is used to eject another electron, called an auger electron, or a photon is released. The released photon is in that case called a characteristic x-ray as its energy will equal the energy difference between two atom orbitals, which is characteristic for each material. For photoelectric effect in the clinical energy range one has found that the cross section per atom is a function of the atomic number Z and the energy of the incoming photon. A normal approximation is

$$\sigma(Z, h\nu) \propto \frac{Z^4}{(h\nu)^3}. \tag{2.2}$$

Another way the photon can interact with the target is through Compton scattering. Then the electron interacts with outer electrons in the atom, where the electron is considered to be free. Both the photon and electron are treated as particles during this interaction. As the incoming photons interact with the electron it gives up a small portion of its energy to the electron and as a result the photon is deflected an angle θ and continues with less energy, while the electron is recoiled through an angle ϕ, see Figure 2.2.

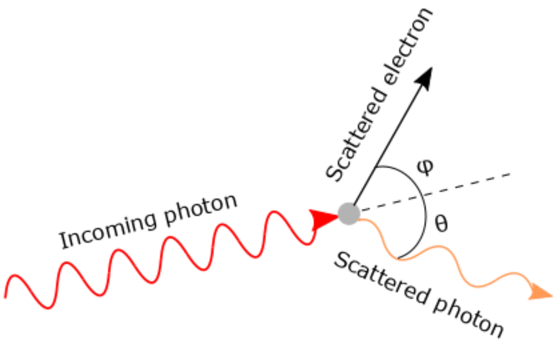


Figure 2.2: Illustration of Compton scattering.

At which angle the electron is recoiled will depend on the energy of the incoming photon, where higher energy gives a larger angle and highest value gives backscattering. The cross section related to Compton scattering has a slight dependence on the energy of the incoming photon, where an increase in energy will decrease the cross section. The probability of Compton scattering is independent on the atomic number but depends on the electron density. Since the electron density is approximately the same in almost all material the cross section related to Compton scattering is almost constant. For energies used in external radiotherapy, Compton scattering is the most prominent interaction mechanism. The last mechanism, pair production, is when the photon is transformed into a positron-electron pair and make up only a small part of the interactions in radiotherapy. For pair production to be possible the energy needs to be above an energy threshold of about 1.02MeV. The pair production cross section is dependent on the atomic number as $\sigma \propto Z^2$, so heavier atoms gives a higher probability for pair production to happen.

As a photon transverse in matter it can interact several times, e.g. a photon which is initially scattered can be scattered several times or end up with photoelectric absorption. To get the macroscopic overview the attenuation coefficient is used, which is related to the cross section. It is a measure of the probability of a beam of radiation being attenuated when penetrating a matter and is specific for specific materials. A large attenuation coefficient means that as the beam of radiation transverse the matter is will be quickly attenuated.

When the primary photons enter a medium an energy transfer happens where the process can be modelled by the energy transfer coefficient given in the following formula,

$$\mu_{tr} = \mu \frac{E_{tr}}{h\nu} = \mu \frac{E_{tr}}{E_0}, \tag{2.3}$$

where hν is the initial energy of the incoming photon, E$_{tr}$ is the energy transferred and μ is the linear attenuation coefficient. This can also be written as the energy absorption coefficient

$$\mu_{en} = \mu_{tr} \cdot (1 - g), \tag{2.4}$$

where g is the fraction of the transferred energy lost in radiation interactions. The result of the energy transfer is production of light charged particles, e.g. secondary electrons, which travels further into the medium and interacts. The most common choice is to use photons in external radiotherapy, and then it is the secondary electrons which delivers the dose.

Electrons

Since electrons generally deposits its energy by interaction through collisions when transversing the matter cross sections and attenuation coefficient is not relevant for electrons, but instead electron range and material stopping power is used. Electron range refers to how far the electron travels before it has deposited all its energy and material stopping power is the specific matters ability to stop the electron. As charged particles passes through a material it will interact along its track and lose its energy little by little before coming to rest. A way to measure this in a material is to study the stopping power of the material. The stopping power is given as the ratio of the loss of kinetic energy ,dE, and path travelled, dx,

$$S = \frac{dE}{dx}. \tag{2.5}$$

Stopping power regularly is referred to as mass stopping power which is defined as S/ρ. The main interaction mechanism for electrons when transversing a matter is through collision with other electrons, which in turn will leave the atom it belonged to ionized. The calculation of the mass stopping power related to this mechanism was first solved by Bethe and later extended by Sternheimer, and is given as

$$\frac{S}{\rho} = \frac{2\pi r_0^2 N_e \mu_0}{\beta^2}\left(ln\frac{T^2(T+2\mu_0)}{2\mu I^2} + \frac{T^2/8 - (2T+\mu_0)\mu_0 ln2}{(T+\mu_0)^2} + 1 - \beta^2 - \delta\right), \tag{2.6}$$

where r_0 is the electron radius, $N_e = N_A(Z/A_r)$, $\mu_0 = m_0c^2$, T is the kinetic energy, $\beta = v^2/c^2$, v is the speed of the electron, δ is a density correction term and I is the mean excitation energy. From Equation 2.6 it is evident that the mass stopping power will be nearly inversely proportional to the energy of the incoming electron. As I is dependent on the atomic number Z in such a way that an increase in the atomic number gives a higher I, that means that for heavier atoms this results in a smaller mass stopping power. Although N_e depends on Z, the ratio Z/A$_r$ will for all materials except hydrogen be about 0.5 as the nucleus almost always hold as many protons as neutrons. This results in a very small over all dependence on atomic number.

The other way the electron can interact with matter is through Coulomb force by passing the nucleus in near proximity. The electron will then be decelerated and pass its energy as electromagnetic radiation, this is called Bremsstrahlung. The mass stopping power can in this case be calculated as the following

$$\frac{S}{\rho} = \frac{1}{137}\left(\frac{e^2}{\mu_0}\right)^2 \frac{N_A}{A_r} Z^2(T+\mu_0)B, \tag{2.7}$$

where B is a function $B = B(hv/T)$ which only has a small dependence on Z and T and are usually used as an average $\overline{B} = 16/3$. It is the seen that the mass stopping power will in this case have a strong dependence on the atomic number $\frac{S}{\rho} \propto Z^2$.

Radiation beam

A radiation beam is quantified by the number of particles and their energy. To find the number of particles in the beam in point P in space a small sphere with diameter d is drawn around. Then the number of charged particles entering that sphere is counted and divided by the cross section of the sphere, dA = d$\cdot\pi$, which gives the fluence Φ,

$$\Phi = \frac{dN}{dA}. \tag{2.8}$$

The same concept can be repeated with energy fluence, Ψ, where instead of number of particles the radiant energy, dE, is used,

$$\Psi = \frac{dE}{dA} = E\Phi. \tag{2.9}$$

As the particles interact with the medium, they start depositing energy. The sum of all the energy deposits in a given volume is the imparted energy, $R_{in} - R_{out}$, which can be measured using radiation detectors.

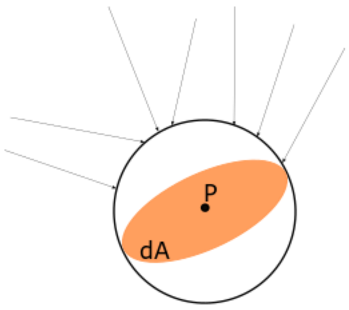


Figure 2.3: An illustration of fluence in a radiation beam.

The radiation beam consists of uncorrelated primary photons, which through interactions produce secondary electrons that are uncorrelated to each other, and they again produce further uncorrelated generations of electrons able to deposit energy. Even though there is no correlation in each generation, there is a correlation between a given primary photon, its secondary electron and further generations produced by that electron. The series of energy deposits created by the primary photon is called an (energy impartation) event. The energy imparted by an event is the sum of all the correlated energy deposits. The event is of a stochastic nature, as this is true for all the energy deposits. Energy imparted can be calculated using the following formula,

$$\epsilon = \sum_{i=1}^{N} \sum_{j=1}^{n_i} \epsilon_j,$$

where N is the number of events, n$_i$ the number of energy deposited at event i and ϵ_j is energy imparted at j.

2.1.2 KERMA

Kinetic energy released per unit mass (KERMA), $K = \frac{dE_{tr}}{dm}$, quantifies the average amount of transferred energy, dE_{tr}, in a small volume, dm,to electrons liberated by the photon interaction. It does not account for what happens after the energy transfer. KERMA is measured in Gray (Gy) which is defined as 1Gy=1J/kg. KERMA can be split into two categories, either originating from collision interactions K$_{col}$ or radiation interactions K$_{rad}$, where K = K$_{col}$ + K$_{rad}$. Further one can define

$$K_{col} = K \cdot (1 - g), \tag{2.10}$$

where g is the fraction of the transferred energy lost in radiation interactions. The total KERMA in a point is given as $K = \frac{dE_{tr}}{dm}$, and from Equation 2.3 it is found that the total KERMA is related to fluence and energy fluence by the following formula,

$$K = \Phi E \frac{\mu_{tr}}{\rho} = \Psi \frac{\mu_{tr}}{\rho}, \tag{2.11}$$

where ρ is the mass density. From Equation 2.4, 2.10 and 2.11 it is found that

$$K_{col} = \Psi \cdot \frac{\mu_{en}}{\rho} \tag{2.12}$$

Since KERMA is an average it is not of a stochastic nature.

2.1.3 Dosimetry

For ionizing radiation stemming from uncharged particles then the particles first need to transfer its kinetic energy to charged particles, resulting in KERMA. Then the charged particles can deposit their energy along their tracks which finally results in absorbed dose. Ionizing radiation stemming from charged particles can directly start its deposits of energy along its track resulting in absorbed dose. Absorbed dose D relates to the imparted energy ϵ,

$$D = \frac{d\epsilon}{dm}, \tag{2.13}$$

and is measured using Gy (=J/kg). Since photons mostly escape the small volume of interest the radiation KERMA is of little interest, and the dose is usually related to the collision KERMA secondary electrons. This means that mainly one sees absorbed dose as being dependent on the deposition of energy by the charged particles, secondary electrons, and not the photons directly. But since the secondary electrons have a finite range not all the energy will be deposited inside the volume in which they are created, which means that K$_{col}$ $\neq$D. This of course depends on the size of the volume, but is true related to dose as dose is defined in an infinitesimal volume.

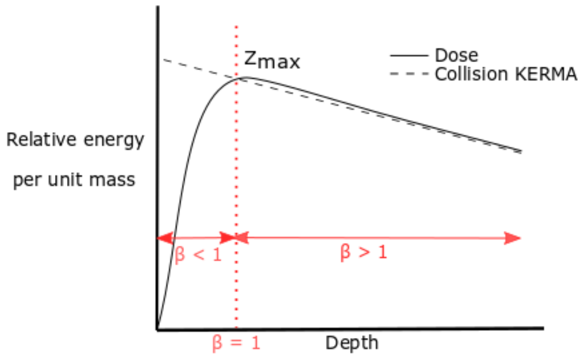


Figure 2.4: Curve showing the relationship between collision KERMA and dose.

Figure 2.4 shows the relationship between collision KERMA and absorbed dose. The ratio β is given by $\beta = D/K_{col}$. Figure 2.4 is what is called a depth-dose curve, assuming the photon fluence is constant and that the beam is attenuated when entering the material. The first part of the curve shows a build-up which reflects the range of the secondary electrons. The electrons travels and deposit energy a short path from where they were liberated, meaning that the build-up is due to upstream electrons from areas close to the surface. At a certain depth, Z$_max$, a maximum is reached where more secondary electrons (or further generations) come to rest than new ones being induces.

Charged particle equilibrium

Charged particle equilibrium (CPE) is the state of constant ionization in a volume dV. In this state the same amount of charged particles is being liberating and then leaving dV as the amount of charged particles that have been liberated elsewhere and then enters dV. From Figure 2.4 the case $\beta = 1$ show when a true charge particle equilibrium exists. The build-up region for absorbed dose is related to $\beta < 1$, and the state after true equilibrium is called transient charged particle equilibrium (TCPE). When a true CPE is achieved the absorbed dose in the material is the same as collision KERMA, given by

$$D = K_{col} = \Phi E \frac{\mu_{tr}}{\rho} = \Psi \frac{\mu_{tr}}{\rho}. \tag{2.14}$$

To assure achieval of CPE there is two necessary conditions that must be fulfilled. The first saying that the medium in dV must be homogeneous in both atomic composition and mass density, and the second condition states that the radiation field must be homogeneous in dV. Once a beam of radiation hits a material the field will no longer be a clean field of either photons or electrons. Once entering the material, the field will be a mixture of primary photons, scattered photons, Bremsstrahlung radiation, secondary electrons, and further generations of electrons. Only when there is true CPE can an accurate description of the radiation field after the beam has entered the material be made using experiments. If there is not equilibrium of charged particles then the field must be described using numerical methods, e.g. Monte Carlo simulation. In reality, a true CPE is impossible and therefore TCPE is used instead.

Beam profile

To measure and study the radiation beam one looks at a profile, beam profile, perpendicular to the central axis of the beam at a certain depth, see Figure 2.5. This gives information about the variation of the beam across one direction. There are typically three parts of the profile which is defined and investigated. The central region is usually defined as the part of the profile where the dose is above 80%. Depending on how the beam is built this can be of different shapes but is traditionally aimed at being as flat as possible. The penumbra defines the rapid fall-off at the edges of the profile. The ICRU has recommended defining penumbra between 80% and 20% of the dose. The last region is the umbra region which is defined as the part of the profile where the dose is minimal, under 20%.

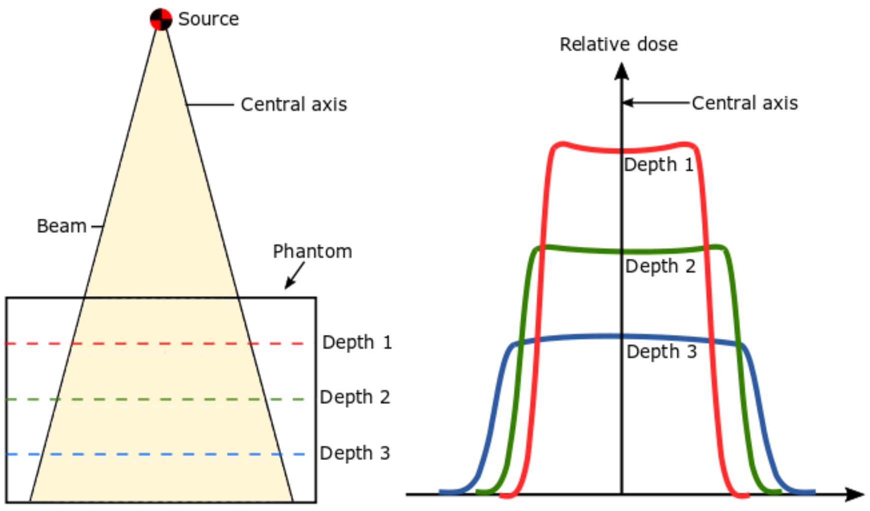


Figure 2.5: An illustration of a beam profile taken at different depths in a phantom.

Using the central region of the beam profile it is possible to study how the dose varies over the same depth. Ideally this region would be flat and the penumbra very small, giving a very precise definition of the field size. However, this is usually not the case and should be investigated when doing quality controls of a radiotherapy treatment. The penumbra can be split into two components, the geometrical penumbra and the transmission penumbra. The geometrical penumbra is a consequence of the fact that the radiation source is not a point source, and therefore will have a fall-off at the edges. As such, the geometrical penumbra will depend on the source size. The transmission penumbra is a result of the beam passing through the different field shaping mechanisms in the linear accelerator, see Section 2.4.3, and is dependent on the energy and field size. The final penumbra will be a sum of these two components and is what is seen in the profile. In conventional radiotherapy where the field sizes are relatively large, the size of the radiation source is of little influence, and it is common the define the field size using the 50% dose level. As the field size is decreased the influence of the geometrical part of the penumbra is increased, and the definition of the field size gets more complicated. In this subsection the reference used is Podgorsak et al. (2005).

2.2 Introduction to radiotherapy

With cancer being the second leading cause of death in the world today (World Health Orginasation (2018)) it is not surprising that a lot of effort and resources are put into cancer research. Of special interest is the knowledge of risk factors and developing effective treatments, there among radiation therapy. Cell growth in a tissue with normal cells is controlled with a balance between the signal that suppresses cell division and the signal that promotes it. Also, a signal telling the cell to undergo apoptosis when needed, e.g. when the cell is damaged, controls the cell growth. These signals originate from the activities of genes in the cell. When mutations happen in these genes, which is what creates cancer cells, the signals will change and cell growth gets out of control. As a result, the cancer cells have a rapid proliferation and get less dependent on signals from surrounding cells. If the cancer at a specific place in the body is allowed to grow, then at a later stage it will metastasize to other parts of the body. When a cell divides only a small fraction of the resulting cells will have the ability to undergo enough divisions to create a colony, these are called clonogenic cells. That means that only a small fraction of the cells contributes to the out-of-control cell growth and it is these cells that is the main target when treating cancer. To be able to cure cancer all the clonogenic cells need to be killed to prevent the remaining cancer cells from starting to grow and spread. There are several ways to treat cancer today, depending on the nature of the case. The most used methods are chemotherapy, surgery and radiotherapy, and often a combination of these are used. Where others are not mentioned the reference used in this section is Mayles et al. (2007) and Podgorsak et al. (2005).

2.2.1 Radiobiology

Radiobiology is a medical science that study the effects of ionizing radiation to living organisms. It explains how ionizing radiation damage the cells and the potential consequences it may have.

Cell cycle

All cells in the human, except for sperm and egg, are somatic cells. When they undergo cell division, mitosis, the resulting products are two identical daughter cells. Identical in this case means genetically identical. The mitosis is separated into four phases, G1, S, G2 and M (see Figure 2.6).

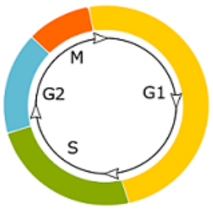


Figure 2.6: The cell cycle showing phase order and the typical relative duration.

How sensitive the cell is to radiation has proven to be varying with phase, where it is most radio-resistant in late S-phase and most radio-sensitive in G2 and M. When the cell is in S-phase the synthesis of the DNA occurs, and it may be thought that the cell is in an environment where repair is easier and the presence of a DNA template can be used to do the repair. During G2 and M the chromosomes are lined up upon the spindle and it can therefore be thought that repair is more difficult and a damage in this phase, e.g. from radiation, would be fatal.

Biological damage

Biological damage to tissue when exposed to radiation mostly occurs due to damage of the DNA, but it can also be a result of damage to other parts of the cell. In relation to cancer treatment the biological damage one hopes to achieve is damage resulting in reproductive death in clonogenic cells. Biological damage to the cell caused by radiation can be a result from either a direct or indirect action. In the direct case the damage is done by the incoming particle directly, while for the indirect case the damage is done indirectly through the formation of free radicals which in turn creates the biological damage. Free radicals are molecules with an unpaired valence electron and are therefore highly reactive and breaks chemical bonds in the target which again leads to biological damage. If the biological damage results in reproductive death as wanted, then the cell will die after some time when it undergoes cell division or some cycles later. Radiation induced biological damage can be put into three categories, lethal, sublethal and potentially lethal damage. Lethal damage is irreversible and will lead to cell death, while sublethal damage can be repaired and potentially lethal damage can be repaired under certain conditions.

2.2.2 The five r's of radiotherapy

The main aim in radiotherapy is to get a high tumor control probability while minimizing the damage done to normal tissue surrounding the tumor. There are several ways to optimize the treatment and there are in particular five factors of special interest which can be used to manipulate the outcome of a radiation treatment. These are called the five r's of radiotherapy.

Reoxygenation

It has been found that the presence of oxygen influences the biological effect of ionizing radiation. This effect is called the oxygen effect. The more oxygen is presence the higher is the biological effect of radiation, but with a saturation for high levels. The effect is also larger for low LET radiation compared to high LET. The explanation of the oxygen effect regarded as the most satisfactory is the oxygen fixation hypothesis created in the late 1950s (Ewing (1998)). The main aspects of this hypothesis are how free radicals created by the indirectly ionizing radiation damage the DNA, any molecular oxygen presence will fixate the DNA and make the damage permanent. This means that the biological damage is a lethal damage and the cell will die. Typical in a tumor is hypoxic cells, which means cells that have had little or no access to oxygen. This is usually due to the poorly constructed blood vessels constructed in the tumor and closing of existing blood vessels for some time. Then the oxygen consumption by the metabolically overactive tumor cells near the vessels will surpass what is supplied and the cells further out will not get access to oxygen. Normally normal tissue is well oxygenated which result in tumor cells being more radioresistant than normal cells in relation to the oxygen effect. If one is able to kill the oxygenated cells then the hypoxic cells will gain access to oxygen and over time be reoxygenated and less resistant to radiation.

Repopulation

As time pass all surviving cells after irradiation will undergo cell division and over time repopulation will happen in both tumor and normal tissue. Usually tumor cells have a higher proliferation than normal cells, which results in a faster repopulation in tumors.

Redistribution

As the cell goes through the different phases in the cell cycle the cell will be radiosensitive to a varying degree. It is most sensitive in G2 and M phase, and least sensitive in late S-phase. When being exposed to radiation the cells in the tumor and normal tissue will be at different phases in their cycle and it is reasonable that a larger fraction of cells in their radiosensitive phase will die compared to cells in their radioresistant phase. After irradiation, the surviving cells will continue their cell cycle and enter other phases, changing their sensitivity to radiation.

Repair

Cells that are not lethally damaged will start their process of repair after irradiation. Cells are created with different pathways of repair, but as tumor cells develop they normally suppresses these pathways and have a smaller potential for repair.

Radiosensitivity

Apart from the factors already mentioned there is an inherent radiosensitivity to each tissue type. Some cancer cells are more radioresistant than the average and will be harder to treat using radiotherapy.

2.2.3 Fractionation

A much used technique in radiotherapy is fractionation. With fractionation the dose prescribed is split into smaller fractions given over several deliveries. There are different fractionation regimens suited to different tumor types. As the dose is given in fractions with time passing between them this gives the cells the opportunity to react according to the 5 r's of radiotherapy. As time is allowed to pass reoxygenation will make sure that more of the tumor cells, earlier being hypoxic, will be radiosensitive coming the following fractions. At the same time the normal tissue which repair more than tumor cells will get time to do that. Because of this the normal tissue will have a better chance at avoiding complications without too much loss of the tumor control when using a fractionation regime. The time between fractions will also let the cells earlier in a radioresistant cell cycle phase to move out of their radioresistant phase and into their radiosensitive phase. Since this is true for both the tumor cells and the normal tissue cells the fact at which speed this happens has to be accounted for. There are drugs developed to synchronize the tumor cells and try to hit them when they are all in their radiosensitive phase. Repopulation will of course work against the aim of radiotherapy and needs to be considered when time between fractions are decided. In addition it has been seen that sometimes cells that have been exposed to radiation will accelerate their repopulation rate, e.g. in some head and neck cancers, and it is in these cases necessary to accelerate the treatment as well.

The five r's of radiotherapy give a sound reasoning to use fractionation and this has become a widely used strategy to gain tumor control. Since the prescribed dose is split into smaller fractions the dose region used is smaller than it would have been without fractionation, usually under 4Gy. This means that any detector used to measure dose should be sensitive in that range.

2.3 Dose measurements

To make sure that the treatment used actually deliver the prescribed dose in the way it was planned an important step in treatment development is to perform dose measurements. Dosimetry is a science that provides a physical parameter to predict biological effects following radiation therapy. There are many dosimetric quantities that can be used to determine biological effects. For photons, the most common ones are fluence, KERMA, charged particle equilibrium and absorbed dose. A dosimeter is a device that can measure one of these quantities. There are several properties a dosimeter should have to be able to be considered a good dosimeter. Firstly, the measurements should be easy to reproduce and have good accuracy and precision. Secondly, there should be a known response to energy and dose, and no saturation as the dose increases. Thirdly, there must be no directional dependence, and at the same time a sufficient spatial resolution. Finally, the dosimeter should be insensitive to other influences such as temperature etc. (Attix (2008)).

A fundamental problem when using a dosimeter is the presence of the detector. Since a dosimeter extends in space it introduces a perturbation of the fluence and creates a state which is not equal to the situation when the dose is not being measured. That means that it is necessary to create a conversion such that the measured dose can be mapped to a value of the absorbed dose without the presence of a detector, which is followed by an uncertainty.

2.3.1 Cavity theroy

To find the absorbed dose, D$_{med}$, at a point P in a medium it is necessary to introduce a detector at that point to measure a dose, D$_{det}$. The sensitive volume on the detector, called a cavity, is normally made of a different material than the material one want to measure dose in, therefore D$_{med}$ $\neq$D$_{det}$. To solve this problem the cavity theory was introduced aiming to study the modification in dose and establish a relationship between measured and absorbed dose. Ideally the cavity should be as small as possible. Since dose related to the photons depends among other things on the energy fluence, that means that the detector must be larger than what is possible in radiotherapy. Because of this one instead measures the fluence from the secondary electrons generated by the interactions of photons. In general, the measured dose within the entire cavity can be calculated by the following formula

$$D_m = \int_{V_{det}} \int_{E=0}^{E_{max}} \Phi \frac{S_{det}}{\rho} dEdr, \tag{2.15}$$

where V$_{det}$ is the volume over the entire cavity, and $\Phi = \Phi(E, r)$ and S$_{det}$=S$_{det}(E)$. This means that the measured dose is a measure of stopping power and electron fluence over the cavity which is a result of photon interactions. A cavity can be defined in one of three category, small, medium or large when comparing to the range of a secondary charged particle induced inside the cavity. The small cavity is of special interest and are defined as having dimensions small enough for the charged particle range to go far past it.

Bragg-Gray cavity theory

The Bragg-Gray cavity theory is a theory developed to provide a solution to the fundamental cavity problem where the cavity is considered as small. There are two Bragg-Gray theory conditions which must be fulfilled. Firstly, when comparing to the range of the charged particles incident on the cavity, the size of the cavity must be small. This is to ensure that the cavity does not influence the fluence of the particles. However, this condition can only be valid if charged particle equilibrium or transient charged particle equilibrium is achieved. Secondly, only those electrons crossing the cavity contribute to the absorbed dose. The dose in the cavity is given by the following equation, given that no energy from the crossing particles are deposited inside the cavity.

$$D = \int_{E_K=0}^{E_{K0}} \Phi(E_K) \frac{S(E_K)}{\rho} dE_k = \Phi \cdot \frac{S}{\rho}, \tag{2.16}$$

where E_{K0} is the initial energy of the secondary electrons produced by photons, E_K it the kinetic energy of the particles, Φ is the fluence and $\frac{S(K_0)}{\rho}$ is the mass stopping power. From Equation 2.16 the relationship between the absorbed dose and the measured absorbed dose in the dosimeter is given by

$$D = \frac{\Phi \cdot \left(\frac{S}{\rho}\right)_{med} \cdot D_{det}}{\Phi \cdot \left(\frac{S}{\rho}\right)_{det}} = \frac{D_{det}\left(\frac{S}{\rho}\right)_{med}}{\left(\frac{S}{\rho}\right)_{det}}, \tag{2.17}$$

where the subscripts med and det stands for material in which one wants to measure dose and the detector, respectively.

2.3.2 Ionization chamber

The ionization chamber is a widely used dosimeter due to its linear dose response, stability, beam quality response independence and more (Low et al. (2011)). As other dosimeters it can be identified as a cavity, and for ionization chambers the cavity is filled with air. When being irradiated ions will be formed inside the cavity. Because of an electric field in the dosimeter the ions will be drawn to either the central electrode or the chamber walls depending on its charge. Then an absolute measure of the absorbed dose can be calculated from the charge accumulated on the electrode. To make sure there is a true electric charge equilibrium inside the ionization chamber a certain size of the active volume is required. Usually the size of the chamber is around 6mm in diameter (Low et al. (2011)) which means that the resolution of the measurement is being compromised. The detector measures a signal that is proportional to the absorbed dose over its active volume. Since the detector is not a point measure, but extends in room, and the signal will be varying over the volume, the measured absorbed dose is an average.

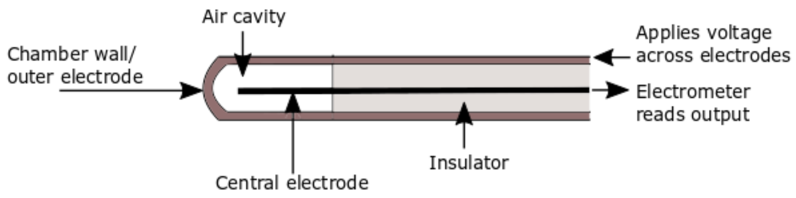


Figure 2.7: An illustration of a ionization chamber.

2.3.3 Radiochromic film

Radiochromic film is a dosimeter that measure the absorbed dose. It was first introduced in the 1960s but had some challenges with the lack of sensitivity. In the beginning of 2000 Ashland released GafChromic EBT film which was tested and accepted as a good tool in quality control (Saur and Frengen (2008)). Unfortunately, the second generation proved to be less accurate and consequently film dosimetry has not been much used. However, in 2011 a third generation, EBT3, was released, which has shown a higher potential (Sorriaux et al. (2013) and Håland and Gustavsen (2019)). Some of the advantages with radiochromic film as a dosimeter are that in the clinically relevant energy range (megavoltage) the radiochromic film is water equivalent, which means there is only a small influence on the charged particle fluence in the material. Also, in the megavoltage beam range there is only a small energy dependence. Other advantages are that radiochromic film can be immersed into water, it is light insensitive and self-developing (Parwaie et al. (2018)). These factor makes the radiochromic film easier to work with, and there is no need for chemical processing. However, the main argument for using radiochromic film is the fact that it gives a continuous readout over two dimensions, only limited by the resolution of the scanner used. This gives the opportunity to study areas of large dose gradients and complex treatment plans. In addition, radiochromic film offers more flexibility as it is possible to place the thin film piece however suits the situation.

GafChromic EBT3

Radiochromic film is a chemical dosimeter that uses the optical characteristics, optical density, of a dye to map the dose distribution. The optical density of a material describes its ability to absorb the light as it passes through it. That means that a material with a high optical density will transmit less light than a material with low optical density. GafChromic EBT3 is composed of two polyester layers covering the active layer in the middle. The active layer is crystals filled with a monomer (diacetylene: Lithium pentacosa-10,12-diynoate (Ashland (2020))), that react upon irradiation by polymerization, forming polymer chains. This results in loss of transparency in the film, meaning a higher optical density. This is the characteristic that is being measured when relating to absorbed dose. To read this increase in optical density a flat-bed scanner can be used. The scanner will transmit light through the film piece and read how much light was not absorbed. As the dose increase less light will be transmitted through the film and the readout in the scanner decreases. The relationship between the optical density and the dose was earlier modelled using polynomial functions, but as pointed out by Micke et al. (2011), this is not an especially good fit. Building on that it was found that a 'reciprocal linear vs dose' works better and is what is recommended by the producer of the GafChromic film EBT3 (Mathot et al. (2014)). The relationship is modelled by the following formula,

$$\text{pixel value} = a + \frac{b}{D - c}, \tag{2.18}$$

where the pixel value is what is read from the scanner (reflecting the optical density), D is the dose and a, b and c are constants that needs to be fitted by performing a calibration. The dynamic range of the GafChromic EBT3 film is between 0.1Gy and 20Gy, but optimum dose range lies between 0.2Gy and 10Gy (Ashland (2020)). This means that it is well suited for applications in standard radiotherapy regimens in which fractionation dose is around 2Gy. The particles in the active layer in the GafChromic EBT3 film tends to align along the short side of the film, resulting in anisotropic light scattering through the film. This means that the pixel value being read will be somewhat different depending on the scanning direction. The polymer chain in the active layer absorbs light in typical bands at wavelengths of 636 and 585 (van Battum (2018)). The absorbance maximum is at the wavelength corresponding to the red light. Therefore, one typically only look at the red color channel when reading out intensity in the images scanned of irradiated radiochromic film. This is in accordance with studies on the GafChromic EBT3 (Håland and Gustavsen (2019)), where it was seen that the red color channel was more sensitive to irradiation than green and blue, and therefore gives better dose resolution.

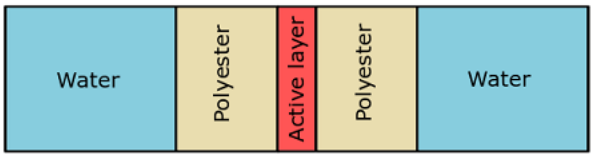


Figure 2.8: An illustration of the layers in GacChromic EBT3.

In earlier studies using the GafChromic film, (Saur and Frengen (2008)) and (Håland and Gustavsen (2019)), it has been seen that a flat-bed scanner has a non-uniform readout in the lateral direction of the scanner. The producer of the GafChromic EBT3, Ashland, mentions the finite anisotropic light source in a flat-bed scanner as the reason for this variation (Ashland (2020)). Another study (van Battum (2018)) found that cross talk, optical path and polarization, all influencing the optical density, are the properties responsible for the variations. Nevertheless, a non-uniform readout is confirmed using a flat-bed scanner and as follows there is a need to do a correction for this when using film dosimetry.

2.4 Radiation treatment techniques

When a patient is diagnosed with cancer one of the treatment options is radiotherapy. This is usually chosen when the cancer was caught early and there is no spreading, or with a palliative intention. At later stages it can be used in combination with other treatment techniques. To start the process of radiotherapy treatment a planning Computer Tomography (CT) is taken. This is used to delineate the tumor and other volumes of interest, which is used during treatment planning, dose measurements and finally patient positioning during treatment. Once the CT is taken the treatment planning starts, which is done as a collaboration between a radiation oncologists, radiation therapist and medical physicists. When the plan is created it is possible to perform a quality check by doing dose measurements using the dose plan, although this is not usually done. Before the delivery of each fraction a cone beam-CT is performed to assure right position of the tumor and other volumes of interest.

2.4.1 Standardized volumes used in radiotherapy

In conventional radiation therapy the gross tumor volume (GTV) is defined as what can be seen, imaged or palpated. The clinical target volume (CTV) is the GTV with an extra margin to cover sub-clinical disease that have spread. This is an important volume because it must be adequately treated to achieve a cure. Planning target volume (PTV) is the volume that includes CTV, internal movements and an extra margin to make up for uncertainties related to external circumstances. In addition the critical organs surrounding the volumes to be irradiated must be delineated and are defined in organs at risk (OAR).

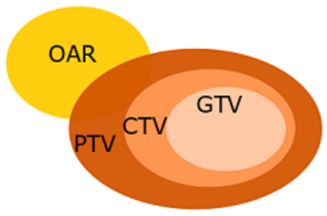


Figure 2.9: An illustration of the volume definitions in radiation therapy.

2.4.2 Planning CT

Before developing a treatment plan a planning CT is performed which is the basis to differentiate the volumes inside the patient. When the CT image is supplied the radiation oncologists can delineate the GTV, OAR and CTV, where PTV will follow from the medical physicists' conditions. To best be able to plan out the dose delivery it is important that the CT during planning is taken while the patient is in treatment position. The appearance of the tumor in the planning CT, and by that the delineation of the treatment volumes, will be directly influenced by parameters such as slice thickness, motion during scan, gap thickness and so on. Because of this these parameters must be controlled and considered when processing the results. It is recommended that the slice thickness in a planning CT should be between 1 and 3 mm (Winer-Muram et al. (2003)), which means that the data obtained by a CT will be stored as matrices with resolution limited by the slice thickness.

A CT works by producing a narrow beam of x-rays which is rotated around a patient. At the same time there is a detector opposite to the x-ray source which detects the x-rays that have not been attenuated when going through the patient. By one rotation one obtains a slice across the patient, and this can be done several times to obtain a 3D image. As mentioned, the output in the CT image is a result of the detected x-ray not being attenuated. This means that a CT image gives direct information about the attenuation of the tissue, and when the delineation is done on the CT-image this gives information about the attenuation in each of the standardized volumes. When calculation dose information about the attenuation coefficient is necessary, see Section 2.1.

2.4.3 Linear accelerator

A linear accelerator is a tool used to generate ionizing radiation, which can be used in radiotherapy treatment. Using a modulator, electron gun and RF power source electrons are released and accelerated through a waveguide. The modulator provides high voltage pulses which leads to a propagating electromagnetic field inside the waveguide. When the free electrons are in coherence with the microwaves from the RF pulse they can be accelerated. The waveguide is built in such a way that as the electrons, together with the microwaves, travel down the waveguide their wavelength is increased. As the frequency of the electrons are kept constant, this will lead to an increase in speed according to the equation $c = \lambda \cdot f$, where c, λ and f are speed, wavelength and frequency respectively. To assure an accurate delivery of the ionizing beam, there are focusing and steering performed through the waveguide. At the end of the waveguide there is a bending system, consisting of either one magnet deflecting the beam 270° or three magnets deflecting a total of 112.5°. Now the electrons can either produce photons through Bremsstrahlung or they can be used directly by letting them hit a scattering foil.

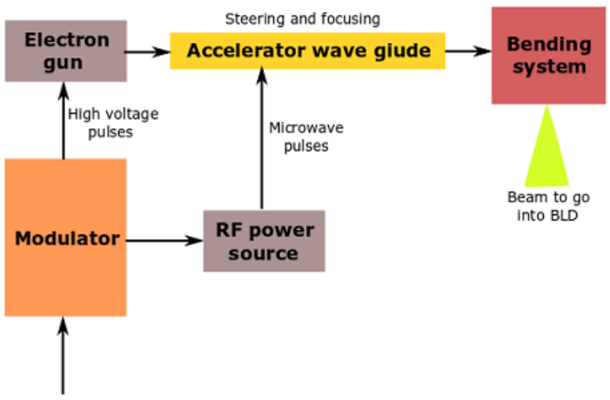


Figure 2.10: An illustration of the main components of the linac.

To make the beam clinical usable it must be shaped using different collimators and filters. The first aperture is the primary collimator, which for electrons or low energy x-rays can be an open hole, while for beams made of higher energy photons it is used as a filter. Next follows a flattening filter. Since the bremsstrahlung photons are more forward peaked, the fluence profile will be cone shaped. Therefor the flattening filter has been an important component to decrease this effect, and make sure to deliver a flat profile. Although it is most often still in use today, it is no longer necessary as the modern planning and delivery techniques has becomes so advanced. After the secondary filter two ion chambers are installed. The first, which is called the primary dosimetry channel, is there to monitor the dose rate and integral dose. When the full dose is given the monitor shuts off the beam. The second ion chamber, which is called the backup dosimetry channel, is there in case the first chamber fails. If an electron beam is used, an electron applicator is used to sharply define the field at the target. If the electrons have been transformed to photons one can use multi-leaf collimators (MLC) to define the field. Also, wedges and shutter can be used to modify the output profile. (Wangler (2008))

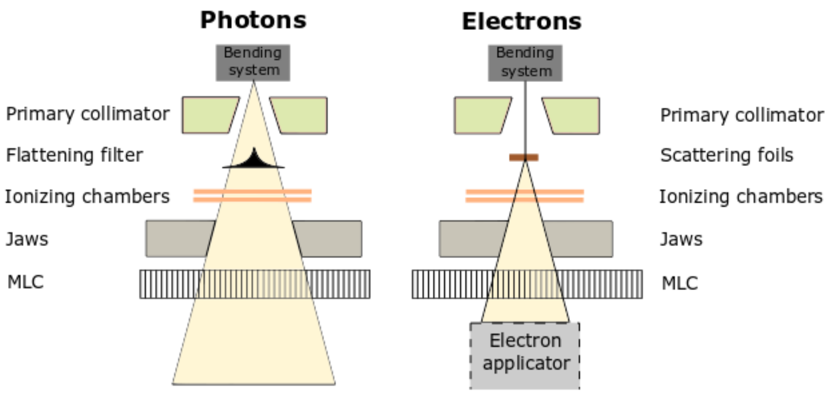


Figure 2.11: An illustration of the beam limiting device

One important consideration during radiation therapy is the right placement of the patient relative to the beam. To do this as best as possible a constant point in the treatment room is created to be used as reference during placement of the patient. That point, called the isocenter, is the point at which the gantry and couch rotates around, meaning that the beam will always pass through the isocenter.

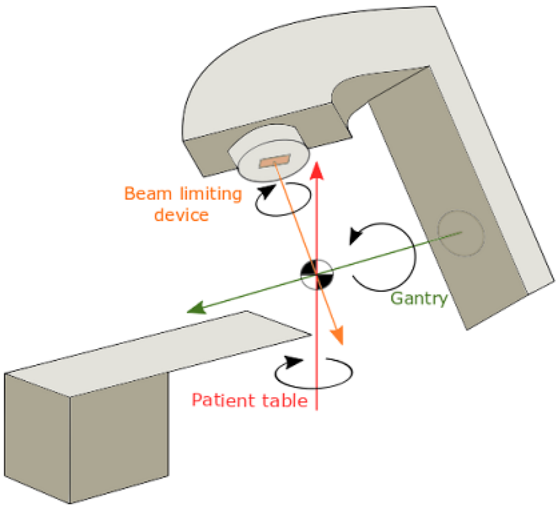


Figure 2.12: An illustration of the isocenter in the treatment room.

2.4.4 Multi-leaf collimators

When the concept of the multi-leaf collimators (MLC) was first introduced by Proimos (1960) and Trump et al. (1961) it was not much used. It was intended to replace the Cerrobend block, which is solid, molded field shaping devices with little flexibility, meaning its main purpose was to shield normal tissue. It was not until later that is was used in dosimetry to create complex conformal beams. The MLC defines the beam field used in the linear accelerator for each treatment. With many interdigitating leaves individually moving, continuously or stepwise, at a high speed it is possible to form a complex beam to fit the treatment case. The gain of this flexibility has made the delivering of dose much more laborious, and more machine power is needed to run. The MLC on the system used at St. Olavs Hospital, Elekta Agility, is 0.5 cm wide at isocenter, around 7cm long and made of wolfram. According to Elekta (2020), the producer of the MLC system this gives a transmission of $< 2\%$.

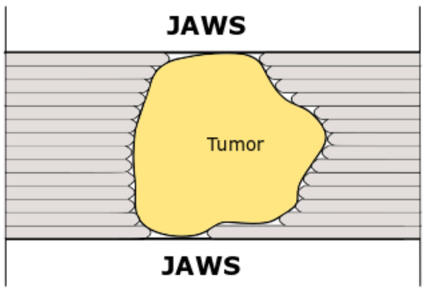


Figure 2.13: An illustration of the MLC leaves.

By changing the positions of the MLC leaves it is possible to create countless number of fields. To monitor the movements of the leaves in real time an optical video-camera system is used. The system is placed inside the gantry head and its accuracy is therefore unaffected by the rotation of the gantry head as the system and the MLC leaves rotates the same. The MLC leaves are placed parallel from two sides that meets at isocenter. The movement of the leaves are linearly and so to minimize the penumbra and keep it constant the tip of each leaf is rounded with a given radius of curvature (ROC). As an effect of this the edge of the radiation field is not cut off as modelled by a light source projection. But this has been studied and found not to affect the sharpness of the beam fall-off significantly (Jordan and Williams (1994)). Also, the rounded edges introduce a path of leakage when the MLC leaves are fully closed.

Tongue and groove effect

To allow for effective movements of the leaves there must be a small gap between them. This creates a path of leakage between the leaves. To reduce this leakage the MLC leaves have a tongue and groove design, see Figure 2.14. For a leaf that is 1 cm in isocenter it has been shown that the actual blocking from that single leaf will be 1.1cm at isocenter because of the tongue and groove design (Liu et al. (2008)). In addition the fall-off at the leaf edge will not be straight, but manipulated depending on how the individual leaves are formed together, see Figure 2.14. This will create a larger penumbra over the leaf flank. For the dose plan system to be able to create a realistic and errorless plan it is important that this characteristic of the MLC leaves is well modelled in the system.

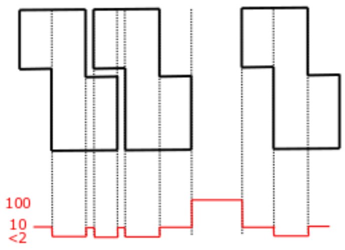


Figure 2.14: An illustration of the tongue and groove in MLC leaves.

2.4.5 Dose calculation algorithms

One important condition to use radiation as a treatment technique is the ability to model the delivered dose in different situations originating from different radiation beams. Several such models have been developed and implemented into hospital systems. There are two conditions that should be met when accepting a dose calculation algorithm. The first is that the result holds a high accuracy, and the second condition is that different results should be easy to compare so that the planner can have confidence in the model used. The main problem when developing a model is inefficient algorithms that take up too much time when using a standard computer with normal processing capacity. Most models follow a fluence-to-dose standard, and is built on simulation of the radiation source in the linear accelerator and how the fluence exists there, total energy released per unit mass (TERMA) and dose kernels (Brady et al. (2006)). TERMA is similar to KERMA, but unlike KERMA it includes energy losses due to Coherent scattering, which is scattering due to interaction with low energy photons. A dose kernel describes the amount of dose per photon at each point relative to the interaction point. The dose inside a voxel will be a result not only by the interactions from the primary radiation beam, but also of interaction of scattered photons and secondary electrons originating form interactions in all other voxels. This makes the modelling much more complicated and time consuming, and because of this much effort has been put into developing good models of dose distribution from a photon interaction. These models are models of the dose kernels. The dose can then be simulated by combining TERMA and the dose kernels, by first finding the beam attenuation and applying a dose kernel to each voxel and then using convolution with TERMA to find the dose, see Figure 2.15.

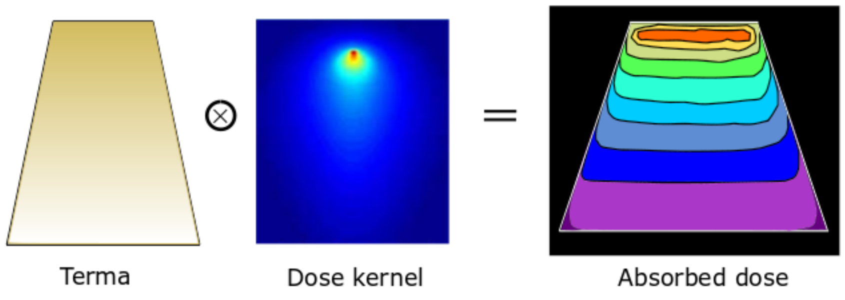


Figure 2.15: The concept of calculating absorbed dose using TERMA and dose kernels.

Because the density in the human body can be varying, e.g. the lung, which is much less dense than the surrounding tissue, the dose kernel should be modelled differently in different tissues. In low density tissue the kernel will be stretched compared to tissue with higher density. There are several ways to implement the dose kernel model but perhaps the most common one when modelling radiation using photons is the collapse cone model (Brady et al. (2006)).

Collapsed cone model

The collapsed cone (CCC) model was developed and described by Ahnesjö int 1989 (Ahnesjö (1989)). It uses the superposition method using point kernel and convolution as described above. By using collapsing poly-energetic point kernels are modelled using particles transported in a straight line with an exponential attenuation (Ahnesjö and Aspradakis (1999)). The advantages of using CCC is that it can be used in tissue with different densities, as CCC will scale the kernel accordingly. Also, this model describes the energy transport in a lateral direction, which was not done by earlier models. At St. Olavs Hospital the dose calculation algorithm when using a photon beam is an enhanced version of the CCC, namely the collapsed cone enhanced (CCE) model. The difference from the new version compared to the old classic model is that the enhanced model considers the fact that there is room between the MLC leaves.

2.4.6 Treatment planning

In this section the reference used is Podgorsak et al. (2005) unless other is stated. Originally treatment planning systems were created to make sure that the fields chosen in a treatment were correct and that the volume being irradiated was indeed the target volume. As the treatment techniques have developed and become more complex the role of the planning system is also changing and becoming more complicated. Among other things the treatment systems today can be used to do quality checks on the field geometry, identifying volumes of risk, patient positioning and generation of treatment plans with the opportunity of comparing.

Before starting to plan the delivery of the treatment all slices of the planning CT must be processed. In every slice all volume of interest and the outer contour of the patient must be delineated. It is important for the precision of the delivery that the patient is placed in the same position during the planning CT that it will be placed during delivery of the treatment. Therefore, this must be planned before doing the planning CT. When uploaded to a treatment planning system the data collected during the planning CT will make up a 3D model of the patient and all the volumes, which will be used to simulate the delivered dose. From the data in the CT slices the treatment system is able to find the attenuation and thereafter simulate dose in each volume depending on the radiation beam its being exposed to. The treatment system is built on physical properties and simulate a radiation source and properties of all the components in the beam limiting device (see Figure 2.11). Using this and the dose calculation algorithms, e.g. collapsed cone model, for dose calculation the treatment planning system can simulate delivered dose given different compositions of radiation beams. There are different approaches when starting to plan treatment delivery depending on the complexity of the case. In simple cases, e.g. in palliative cases, the field is not so complicated, and an old-fashion approach is often used where the field is built up of only a few beams (parallel or opposed) having a flat beam profile. These plans are created manually in the treatment systems. For more complex cases, e.g. conformal treatment planning, more information about the patient geometry and composition is needed and more sophisticated methods are used, e.g. inverse treatment planning.

Inverse treatment planning

As technology becomes more evolved this opens the opportunity to create more sophisticated treatment plans. Examples of such methods are intensity modulated radiation therapy (IMRT) and volumetric modulated arc therapy (VMAT). IMRT is a method where the dose is delivered using many beams from different gantry angles. Usually the gantry will be at 5, 7 or 9 different gantry angles delivering around 10 segments at each angle. By segments is meant different beams being given after each other, where each segment is different from the other by the shape of the MLC. This creates an intensity modulated field built up by the fluence from each segment. VMAT is a method where the gantry moves through an arc in one continuous movement whilst the MLC are changing formation. As these kind of treatment methods are so complex and is composed of many parts, the planning is more requiring and cannot be done manually. Instead a method of inverse planning was introduced, where the concept of inverse is used because the algorithm starts with a condition of the delivered dose and uses conditions set by the planner to calculate how to achieve this. This algorithm is built on an objective function, which is an expression of the comparison between the requested dose and the simulated dose. The planner will have to input some conditions and constraints, e.g. dose given to specific volumes, and the algorithm will the try to minimize this objective function using an iterative process. The fundamental problem using this approach is to make sure that the found minimum is actually the global minimum and not a local. This is solved by using a gradient decent. Once the system comes up with a treatment plan the planner can try to adjust their conditions and constraints and see if the plan can be further optimized.

MLC in treatment planning systems

The treatment planning systems uses models to simulate the different component involved in dose delivery. One of these components are the MLC, which is modelled using four parameters. The first parameter used when simulating MLC is leaf-tip offset. When calibrating the MLC according to a planned field a light field is used, and since light fields diverge and the MLC leaf-tip is rounded this gives an offset between the field being defined by the leaf tip and by the leaf side, see Figure 2.16.

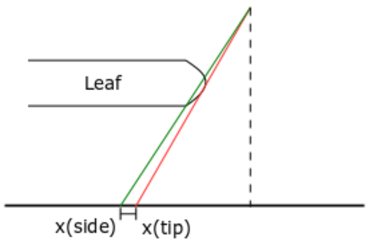


Figure 2.16: An illustration of geometrical offset between field being defined by the MLC leaf tip and side.

The second parameter being used when modelling MLC in treatment planning systems is the leaf-tip width, see Figure 2.17. In the treatment planning system used during this work the tip width used is one-half the thickness. When the parameter is set like this that means that the radiation transmission leakage in the leaf-tip will be the square root of the average leaf transmission factor and is a source of uncertainty. The leaf-tip offset together with the leaf-tip width are the two parameters that will define the radiation field edge and penumbra.

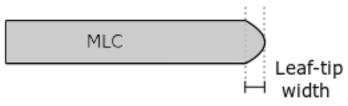


Figure 2.17: The leaf-tip width of an MLC leaf.

The third parameter used to model MLC in treatment planning systems are the leaf radiation transmission factor, which is composed of two elements, the leakage through the leaf body and the leakage through the gap between each leaf. Since the average leakage through the body of the leaves are stated by the producers this is a simple matter of defining. But the leakage that originates from the gap between each leaf will depend on the formation of the MLC and dose being delivered. Therefore, this component of the parameter must be modelled. The mostly used solution is to simply use the average leakage as an estimate, and this is also done in the treatment planning system used in this work. The last parameter used to model MLC is the tongue and groove width. This width is modelled using half width of the normal leaf width as is done with the leaf tip. Therefor the radiation transmission factor through the tongue ang groove regions will also here be the square root of the average transmission factor. Also, when adjacent leaves about the effect of the tongue and groove will not be taken into account by the treatment planning system. In this subsection the reference used was Chen et al. (2015).

Dose volume histogram

As a treatment plan has been developed, one way to study the and compare different plans is using dose volume histograms. As the treatment planning systems give information about the dose distribution as a 3D-matrix, as well as information about the defined volumes inside the dose plan, it is possible to look at the dose distribution in each of the volumes. A dose volume histogram is a plot with volume given as a percentage at the y-axis and dose at the x-axis. The curves then show how much of the volume obtains a certain level of dose during irradiation. By looking at the lowest percentage one can obtain information about the maximum dose delivered to the volume of interest, which often are of interest when optimizing the treatment plan in inverse treatment planning. Figure 2.18 illustrates an example of how a dose volume histogram can look like. In this subsection the reference used was Podgorsak et al. (2005).

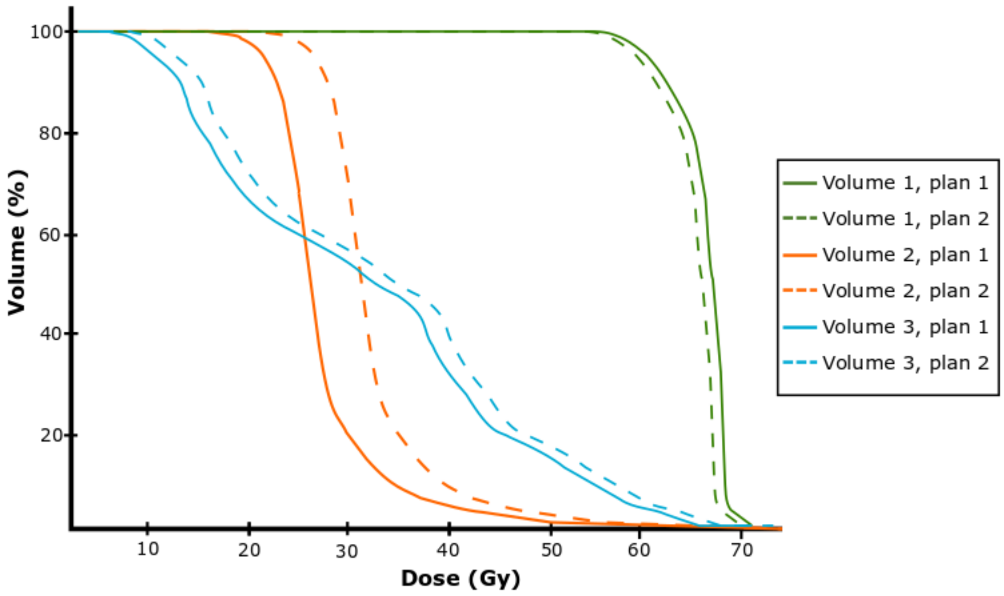


Figure 2.18: An illustration of a dose volume histogram.

2.4.7 Stereotactic radiotherapy

Stereotactic radiotherapy is an up and coming procedure that have proved effective in control of early stage primary and metastatic cancers. It is used to treat cancers located throughout the abdominopelvic and thoracic cavities, brain and at spinal and paraspinal sites. What separates Stereotactic radiotherapy (SRT) from conventional therapy is the use of high doses given in a few fractions to a small volume, where doses are high relative to conventional radiotherapy. This results in a high biological effect, and to spare the normal tissue as much as possible it is therefore critical that the dose is delivered with a steep dose gradient away from the target. Because of this it is important to have confidence in the accuracy of the whole treatment system, both the planning part and the delivery of the dose. Also, a requirement for the practice of SRT is precise delineation of the specific anatomically volumes. Many publications have affirmed the usefulness of SRT in the treatment of both benign and malignant lesions. In clinical studies SRT compare favourably to surgery with minimal adverse effects both for primary and metastatic diseases (Benedict et al. (2010)).

The small fields used in stereotactic treatments are made in the linear accelerator using either flattened or unflattened high-energy photon beams and shaped using jaws, MLC and/or cones. The most common is to use flattening free filter, as this has shown to decrease treatment time and give a better overall result (Dang et al. (2017)). In general there are three conditions to decide if a field can be considered as a small field in radiotherapy, a) lateral charged particle equilibrium does not hold on the beam axis, b) the collimating devices partially block the beam, and c) at the depth of measurement the cross section of the beam is smaller or equal to the size of the detector (Palmans et al. (2018)). The actual size that defines a small field is not an exact number but will be inside a range generally under 4cmx4cm. The size will depend on the range of the charged particles giving the dose, which again depends on the energy of the primary photons or electrons. For a treatment using 6MV an approximate definition of a small field is 1cmx1cm (Andreo et al. (2017)).

The radiation beam stemming from the linac originates from a source of a finite size. In conventional therapy radiation from the entire source will be seen at a detector, while this changes when using small fields, as the forming of a small field will cover parts of the source. This means that the outcome is smaller than what is the case when the whole source is used. When the field is small enough the only parts of the radiation beam that will contribute on forming the field is the geometrical penumbra. This causes an overlap of the penumbra on each side, as seen in Figure 2.19. Since the traditional way to define a radiation field size is using the full width at half maximum (FWHM) method this will in turn result in larger field sizes than intended. (Das et al. (2008)).

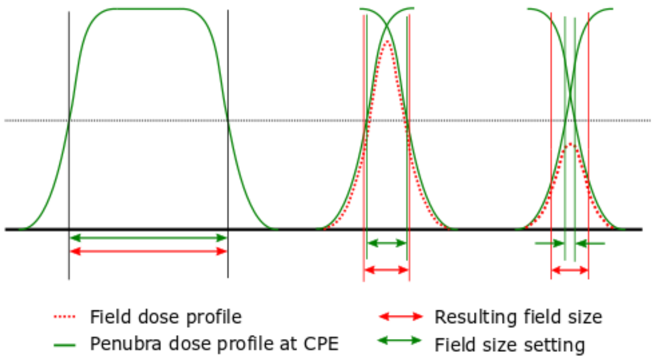


Figure 2.19: An illustration of the changes in output resulting from overlapping penumbra.

2.4.8 Dosimetry and dose measurements in SRT

As small static megavoltage photon fields are a relatively new method of delivering radiation therapy the standard publications and reports used in dosimetry did for some time not cover the topic. The International Commission on Radiation Units and Measurements (ICRU) have officially defined target volumes in specific reports. Report 50, 62 and 83 are good for small volume stereotactic treatments, but for some types of tissues it has been found necessary to give specific recommendation on image modality used when performing the delineation of volumes. These tissues include lung, liver, head and neck and prostate when using stereotactic radiation treatment. In the old reports the standard for prescribing dose was also covered, but for conventional field sizes. In reports 50 and 62 the prescription of dose was based on using a reference point and prescribing the dose to that point (Wilke et al. (2019)). However, as the treatment plans got more complex and the target volumes no longer was supposed to receive a homogenous dose distribution the method using a reference point was not as good. In report 83 they introduced a new method based on using multiple dose-volume histograms. A dose-volume histogram relates the dose given to a volume by the percentage of the volume is give what percentage of the dose. Still this gave a prescription of a relatively homogenous dose to the volume. With stereotactic treatment the dose profiles are in general inhomogeneous, and therefore this method is not ideal. In SRT the method most used for dose prescription to the target covers isodose lines below 100% (Wilke et al. (2019)). With an increasing need for a formally recommendation of dose prescription in stereotactic treatment a new report was made covering SRT, report 91 on prescribing, recording, and reporting of stereotactic treatments with small photon beams.

Before the new reports came with specific information about dosimetry, prescribing, recording, and reporting of stereotactic treatments the way to plan and deliver treatment in SRT essentially followed the same steps as in conventional treatments. However, since the new reports have become available the image modality used to obtain information defining both the volumes and attenuation have been specified and should be followed. The type of image modality and parameters used still depends on the nature of the case. When it comes to prescribing, recording, and reporting this should be implemented in the treatment systems and used when planning a SRT treatment. But even with the new reports specifically developed to better the simulation for SRT there is still some uncertainties, especially related to the MLC leaves and how they are simulated. As mentioned in Section 2.4.6 the approximation of the radiation transmission through the leaf tip width introduces an uncertainty. Since a precise definition of the field edge is very important in SRT this uncertainty can be a problem and is not very well studied for the specific combination of treatment planning system and linear accelerator used at St. Olavs Hospital. As the leaf-tip width will define the radiation field edge and penumbra, this is a very important parameter in SRT treatment, and should be studied more by investigating the penumbra over the leaf tip and flank. Also, because the fields are very small in SRT the beam profiles should be investigated to make sure that the treatment planning system actually delivers what is intended. As high gradients should be expected a good dosimeter to perform these measurements are radiochromic film.

In November 2017, the International Atomic Energy Agency (IAEA) released a new standard, IAEA TRS-483, that addresses the reference and relative dosimetry for SRT with nominal accelerating potential up to 10MV (Palmans et al. (2018)). TRS-483 sets a standard for measuring absolute and relative doses in small fields, whereas standard formalism before this has been limited to larger beam fields. Studies have shown that there is large discrepancies between the ratios of measured values using different types of detector but same beam quality and the actual ratio of absorbed dose to water (Das et al. (2008), Sauer and Wilbert (2007) and Sánchez-Doblado et al. (2007)). The field size and detector type is responsible for these differences, which can be very large if the active volume of the detector is big compared to the field size, as is the case in SRT. As a detector is placed in a conventional radiation beam the field is large enough for the charged particle to hold laterally, while the same is not true for small radiations fields. The differences in readings done in small fields (0.5 x 0.5 cm^2) and larger fields (10 x 10 cm^2) can amount to several tens of per cent (Sánchez-Doblado et al. (2007)). This is shown for ionization chambers, diodes and diamond detectors with an active volume between 1 and 3 mm in diameter. The loss of lateral charged particle equilibrium together with the fact that small fields have large dose gradients will make the uncertainty related to corrections for volume averaging in dose measurements large. Also, the perturbations in the fluence due to the presence of the detector will become large for the same reasons. Since most dosimeters has a large volume relative to the small field size, it becomes apparent that dosimetry in small field radiation therapy is complex and has room for improvement. The new standard, IAEA TRS-483 aims to cover these problems by setting recommendations for reference dosimetry of the radiation machines and the determination of field output factors (Palmans et al. (2018)). One dosimeter has a growing interest in small field treatment, namely radiochromic film. As film dosimetry offers an absolute and continuous measurement of absorbed dose, it is ideal to study the large dose gradient without introducing any large uncertainty due to volume averaging (Wen et al. (2016)) as is done in other dosimeters, e.g. ionization chamber (see Section 2.3.2). Also, the resolution given by other dosimeters, e.g. ionization chambers, is usually too low to correctly present the large dose gradient, whereas when using the film one gets a continuous measurement only limited by the resolution in the scanner. In addition, as mentioned in Section 2.3.3 the radiochromic film will when used in a clinically relevant beam range be water equivalent, which means that there will be little perturbation of the charged particle fluence due to the presence of the film and therefore this will not contribute much to the uncertainty. Another advantage with using radiochromic film in SRT is the fact that the film as a dosimeter does not depend on energy, which is necessary as different beam energies (up to 10MV) are used with different cases in SRT (Palmans et al. (2018)).

2.5 Python

Using film dosimetry introduces the need to perform image processing on the scanned images. To do this the producers of the GafChromic EBT3 film has created a software with this intention. However, since this is a patented and licensed program that means that one must pay to use it and are only able to use their functionalities without the possibility to do any changes. Since the interest in radiochromic film often lies in studying new methods and smaller areas of more complicated plans the user would gain ease of use by being able to do adjustments in the image processing software. Because of this an open source software written in a mainstream programming language, such as Python, could be a good alternative.

2.5.1 Algorithms

Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm is the standard choice for performing curve fitting for unconstrained problems. By expressing the function as a sum of squares of nonlinear functions it locates the minimum using an iterative technique (Polykarpou and Kyriakides (2016)). It is widely used in optimizing computer programming in many languages and is implemented as standard in many of Pythons libraries, e.g. SciPy.

Bresenham’s line algortihm

In computer science a common problem is drawing lines through random points in a dataset. Since datasets are commonly represented as a 2D or 3D matrix there is usually no way of drawing completely straight lines across two random points. Bresenham’s line algorithm is a well-known method of finding the shortest path between two points in a dataset, always moving through existing indices in the matrix. It is built on a incremental error method and moves across the dataset point by point always choosing the direction giving the smallest error to the actual slope of the ideal line (Kennedy (2012)).

Hausdorff distance

There are several ways of comparing two datasets to see how close they are to each other. One way is through the Hausdorff distance, which measures the longest distance between any points in the two datasets. To compare datasets with matching indices, one can use directed Hausdorff distance, where the distance is measured between corresponding points. As a result, the directed Hausdorff distance gives the distance where the two datasets are furthest apart (Zhao et al. (2005)).

Chapter 3

Materials and Method

3.1 The basis of Fidora

As mentioned in Section 2.5 an open source software written in Python (vs. 3.7.4) could stand as a good tool to perform image processing and analysis for film dosimetry. Therefore, a program called Fidora was developed in this work. Fidora got its name from Film Dosimetry in Radiotherapy, and even though it was developed with a special focus on irradiation on the breast and stereotactic treatment it is written as a general tool. Therefore, most of its futures are non-specific and general with the possibility to do changes and developments in the scripts. As an analysing tool Fidora holds different functionalities to investigate different properties of the dose plans. During this work, the functionalities for background correction of scanned film, map dose, calibration, profiles and dose volume histograms were developed. Figure 3.1 shows a simple overview of the functionalities. Since Fidora is meant as an open source software, the though is that Fidora can and will be altered as the need for new functionalities arises.

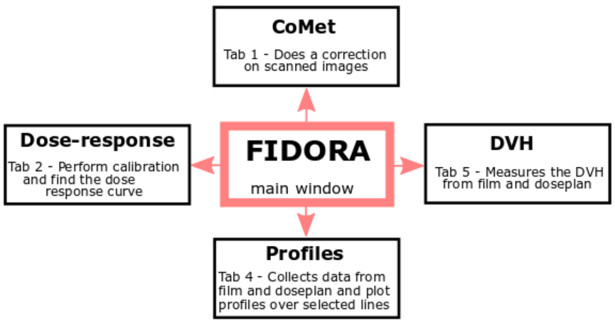

CoMet
Tab 1 - Does a correction on scanned images
Dose-response
Tab 2 - Perform calibration and find the dose response curve
FIDORA
main window
DVH
Tab 5 - Measures the DVH from film and doseplan
Profiles
Tab 4 - Collects data from film and doseplan and plot profiles over selected lines

Figure 3.1: Flow chart of Fidora showing its main functionalities.

3.1.1 General specification made by Fidora

Even though Fidora is written as a tool that can import dosimetric data from any treatment planning system there are some initial specifications created on treatment planning, irradiation and scanning which must be followed.

Specifications during treatment planning

Table 3.1 shows which parameters in the treatment planning system that is specified by Fidora. If this is not followed Fidora needs to be reprogrammed.

Specifications during treatment planning	**Valid choices**
Slice thickness (given from planning CT)	1, 2 or 3 mm
Dose matrix resolution	same as slice thickness
Dose matrix orientation in relation to patient coordinate system (see Figure 3.2)	[1,0,0,0,1,0], [1,0,0,0,0,1], [0,1,0,1,0,0], [0,1,0,0,0,1], [0,0,1,1,0,0] or [0,0,1,0,1,0]
Patient orientation (see Figure 3.4)	HFS, HFP, FFS, FFP, HFDR, HFDL, FFDR, FFDL

Table 3.1: Specifications during treatment planning when using Fidora.

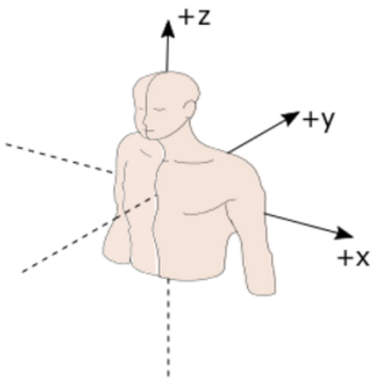


Figure 3.2: An illustration of the patient coordinate system.

As mentioned in Section 2.4.2 the slice thickness in the planning CT is recommended to be between 1 and 3 mm, which is why Fidora is limited to using slice thickness and resolution 1, 2 and 3 mm. The dose matrix orientation is defined by an array where the three first entries hold the cosine of the angle of the first row relative to the patients x, y and z axis (see Figure 3.2) and the three last entries holds the cosine of the angle for the first column. Meaning that if the dose matrix orientation is given as [1,0,0,0,0,1] that means that the first row in the dose matrix is parallel to the x-axis and perpendicular to the y- and z-axis, while the first column is parallel to the z-axis and perpendicular to the other two, see Figure 3.3.

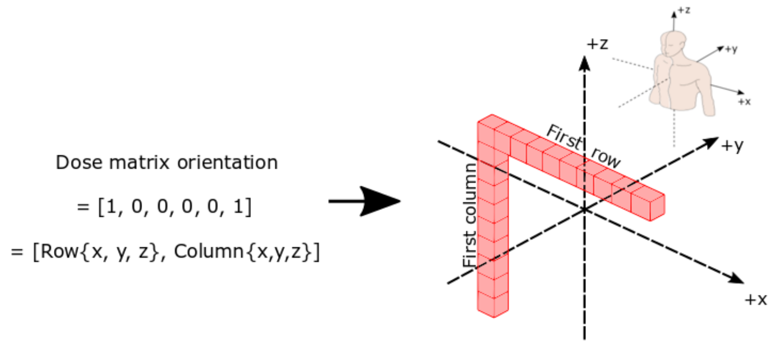


Figure 3.3: An example of the dose matrix orientation in the patient coordinate system.

The patient orientation describes how the patient is positioned at the table in the linear accelerator and the different positions is shown in Figure 3.4. The patient coordinate system is a standard way of orienting the dose matrix in DICOM files and is always defined with the given patient directions as in Figure 3.2, meaning that it is independent on the patient orientation in the treatment room.

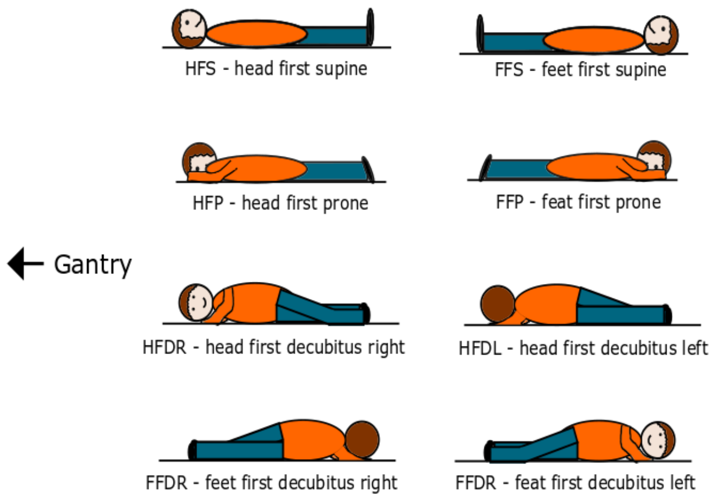

HFS - head first supine
FFS - feet first supine
HFP - head first prone
FFP - feat first prone
Gantry
HFDR - head first decubitus right
HFDL - head first decubitus left
FFDR - feet first decubitus right
FFDR - feat first decubitus left

Figure 3.4: An illustration of the different patient positions.

However, to be able to match the position of the film in the dose matrix the patient orientation is needed so as to know how the patient coordinate system is oriented relative to the spatial coordinate system in the treatment room.

Specifications during irradiation

One of the key parts in most of Fidoras functionalities is to align the position of the film with the area of interest in the dose matrix from the treatment planning system. To do so it is important that the film is irradiated with parameters specified in Table 3.2.

Specifications during irradiation	**Valid choices**
Film orientation	Sagittal, Coronal or Axial
Position reference	Isocenter or reference point in phantom
Beam energy	6MV

Table 3.2: Specifications during irradiation when using Fidora.

When creating a treatment plan the final dose plan will be stored as a 3D-matrix, meaning that all its data will be stored according to three principal axes, which gives the dose plan a limitation on resolution. In addition, a datapoint retrieval not perpendicular to one of the axes will further reduce the resolution. Because of this it was decided that the radiochromic film, when using Fidora, should always be placed perpendicular to one axes and parallel to the other two during irradiation, see Figure 3.5. Also, to lower the uncertainty of positioning the film in the phantom it was decided that the film would only be placed in one of three planes defined in a patient, sagittal, axial and coronal, see Figure 3.6.

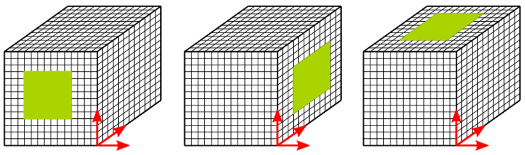


Figure 3.5: Examples of how the film can be placed according to the dose matrix. The film can be placed in any slice. The green illustrates the film, and the arrows shows the principal axes.

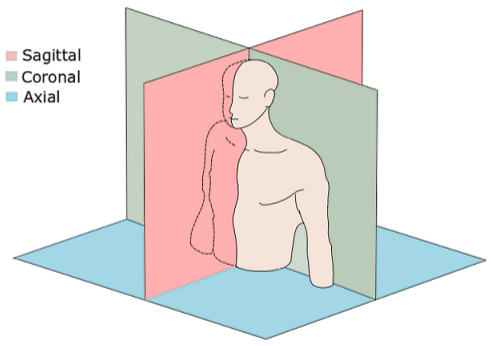


Figure 3.6: An illustration of the three film orientations, sagittal, axial and coronal.

Before irradiation the film has to be placed at the position in the phantom which one wants to measure. When this is done a mark on the film either on the lines leading to isocenter or a reference point should be made, see Figure 3.7. If neither of these points are on the film sheet a temporary point on the film must be made and a note on the distance from that point to isocenter/reference point in lateral, vertical and longitudinal directions has be made for later orientation in the dose plan. Fidora uses the information about dose matrix orientation, patient orientation and film orientation to orient the film according to the dose matrix. Then by using the reference position to the film, slice thickness and dose matrix resolution Fidora align the scanned film to the correct position in the dose matrix.

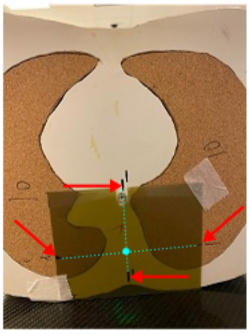


Figure 3.7: An example of the markings made on a film to find isocenter. The black marks (shown by the red arrows) on the film marks the directional lines going towards isocenter (shown by the blue dot).

Specifications during scanning

In the workflow using GafChromic EBT3 film a central part is scanning of the film after irradiation. Fidora assumes that a flat-bed scanner, Epson v750 Pro, was used. The scanner must be used doing a transmission scan, meaning that it is the light being transmitted through the film that is being measured. The resolution of the scanner is limited by how many photodiode detectors the scanner contains. A bit simplified, the scanner can be viewed as a 2D grid of photodiodes, each detecting signal intensity and storing the information in three channels (red, green and blue). This principle is the basis for all image processing and data representation and is the reason why results will be represented in terms of either pixel values or dose. Pixel values are absolute measurements and allows for different results to be compared.

The specifications listed in Table 3.3 shows the Fidora specifications regarding the scanning of the film. These specifics are based on recommendations from the producers of GafChromic EBT3 film. By following these specifications one assures that the scanner do not initiate any automatic adjustments to the image, resulting in non-absolute pixel values, and also that the data loss is minimal giving a 48-bit storing capacity and using TIFF format. In addition, the resolution is specified to 127 dots per inch (5 pixel/mm) which assures a resolution high enough for comparison with dose plans, which is given with a resolution of 1, 2 or 3 pixel/mm.

Choice in Epson scan app	**Setting**
Mode	Professional
Document type	Trasmission
Film type	Positive film
Image type	48 bit
Resolution	127 dpi
File type	Multi-TIFF
Scanner corrections	No corrections

Table 3.3: Scanning specifications used in Espon scan app.

As described in Section 2.3.3 the particles in the active layer of the film has a tendency to align along the short side of the film. It is therefore important to keep a constant orientation of the film when scanning. Fidora assume that landscape orientation is always used during scanning, see Figure 3.8.

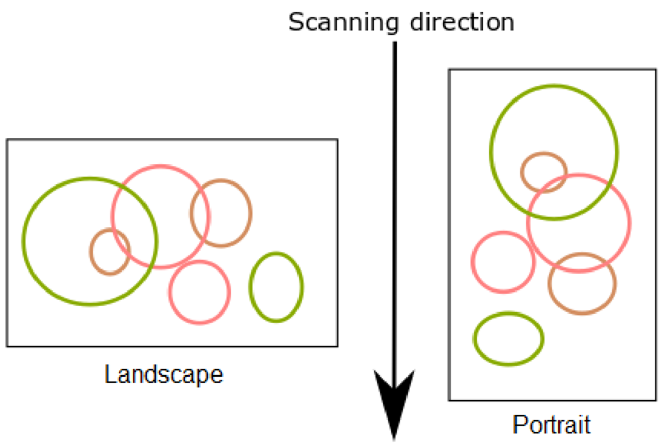


Figure 3.8: An illustration of the scanning direction and orientation.

From earlier work (Håland and Gustavsen (2019)) it was discovered that the flat-bed scanner needs to undergo a warm-up procedure to produce a stable readout and at least 3-5 warm-up rounds (without the film inside) should be used. Also, if the scanner is inactive for longer than 4 minutes, the warm-up procedure needs to be repeated. In addition, to avoid any blackening of the film because of overheating in the scanner all scans must be done at least 1 minute apart. When performing a scan, the scanner uses its calibration area placed at the top of the scanner as a reference to perform a self-calibration of the machine. It is therefore very important that this area is not covered by film. To avoid varying optical paths during scanning a glass plate is recommended to fix the film on the scanner surface. To avoid band-artefacts this glass plate must be cleaned before every use and placed to cover the entire calibration area.

3.1.2 How Fidora reads a scanned image

The first step in all Fidoras functionalities is to process the image scanned of an irradiated film. As the film is being scanned, using 127 dpi, every 1mm will be mapped by 5 pixels which creates a 1270x1016x3 matrix with datapoints for used scanner. The third dimension is the color channels, red, green and blue. When processing the images in Fidora these color channels have been separated into three 1270x1016 matrices to simplify the process and give the opportunity to collect data from each of the color channels. Directly after the reading of the image and before any processing of data Fidora applies a median filter with a 5-pixel width on the datasets. This is done to avoid any 'salt and pepper'-artefacts. Also, because the scanning is done using transmission instead of reflection that means that the image is flipped left to right relative to how you see it. To avoid any mistakes when using the dataset to compare to other data, e.g. dose plan, the image is flipped back as it is being read in Python.

3.1.3 How Fidora reads a doseplan

There are three files of the treatment plan that Fidora can read, RT Dose file, RT plan file and the structure file. These three files make up the basis for the total treatment plan and describe how the irradiation were planned and should be delivered. More files can be part of the total treatment plan, e.g. one dose file for each part of VMAT, but only RT Dose (which will hold the total dose plan from all parts of the delivery), RT Plan and the structure file are needed in Fidora. In all Fidoras functionalities using a dose plan the RT Dose file must be uploaded. This is the file holding the dose matrix, resolution and orientation. In addition, where Fidora must perform positioning of the film into the dose matrix the RT Plan file also has to be uploaded, holding information about positioning of the dose matrix relative to the treatment room. To be able to use anatomically or manually drawn structures the structure file also needs to be uploaded. This would be relevant in cases where one wishes to study the dose distribution in volumes of interest.

3.2 Fuctionalities of Fidora

In this section the main functionalities of Fidora will be described in detail. During this work, each functionality is placed in separate tabs and work independently of each other, although some of them are built on similar functions. See Figure 3.1 for an overview of the functionalities of Fidora developed during this work.

3.2.1 CoMet

The first tab holds the further developed version of a software programmed in an earlier project, CoMet. CoMet stands for Correction Method and is a program where the user can upload their scanned image of GafChromic film, and the program will perform a correction. As described in Section 2.3.3, because there has been found that a flat-bed scanner has a non-uniform readout in the lateral direction of the scanner there is a need to make corrections for this. As such a correction matrix has been developed, which is used in Fidora on every uploaded image.

Creating the correction matrix

The correction method in CoMet is based on an absolute subtraction method, using matrices in Python. An area of 10cm$x$10cm of the scanner surface were studied to find the variations along both directions of the scanner. Since a linear accelerator is not perfect, it will not deliver a totally uniform dose across its beam field. The most precise area of the beam field is the centre and because of this the 10cm$x$10cm scanner area must be measured using small film pieces each irradiated at the centre of the beam field. To cover the area a 5x5-matrix was used, resulting in 25 measuring points over the 10cm x 10cm area with a separation of 2cm in each direction, see Figure 3.9.

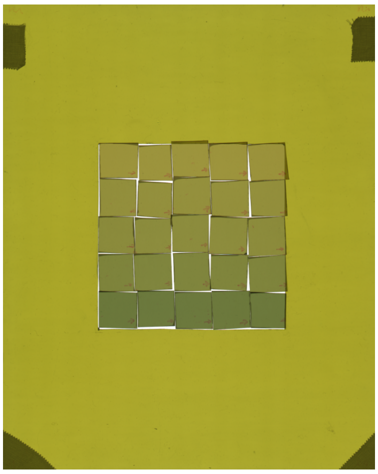


Figure 3.9: An image of the setup during scanning of film pieces meant for background correction.

Film pieces of 2cm x 2cm were cut out and irradiated with doses 0, 25, 50, 100, 200 and 400cGy using a field size of 10cm x 10cm. Every piece was then scanned in all 25 positions, and an area of 3mm x 3mm (15x15 pixels) in the centre was read and averaged using Python. This was repeated for each dose level, resulting in a 5x5-map for each color channel representing the surface of the scanner. Since the centre of the scanner surface is assumed to be correct this was chosen to be the reference point. The correction matrices in each color channel for each dose were then found by calculating the difference in each point in reference to the centre. Because the scanning itself will contribute to increasing the optical density of the film, it was decided that each film piece should only be scanned once at each position. Employing cubic interpolation and extrapolation the difference matrices were reshaped to fit the entire scanning area and the resulting matrices were written and saved in *.txt files, which is necessary attachments included in Fidora.

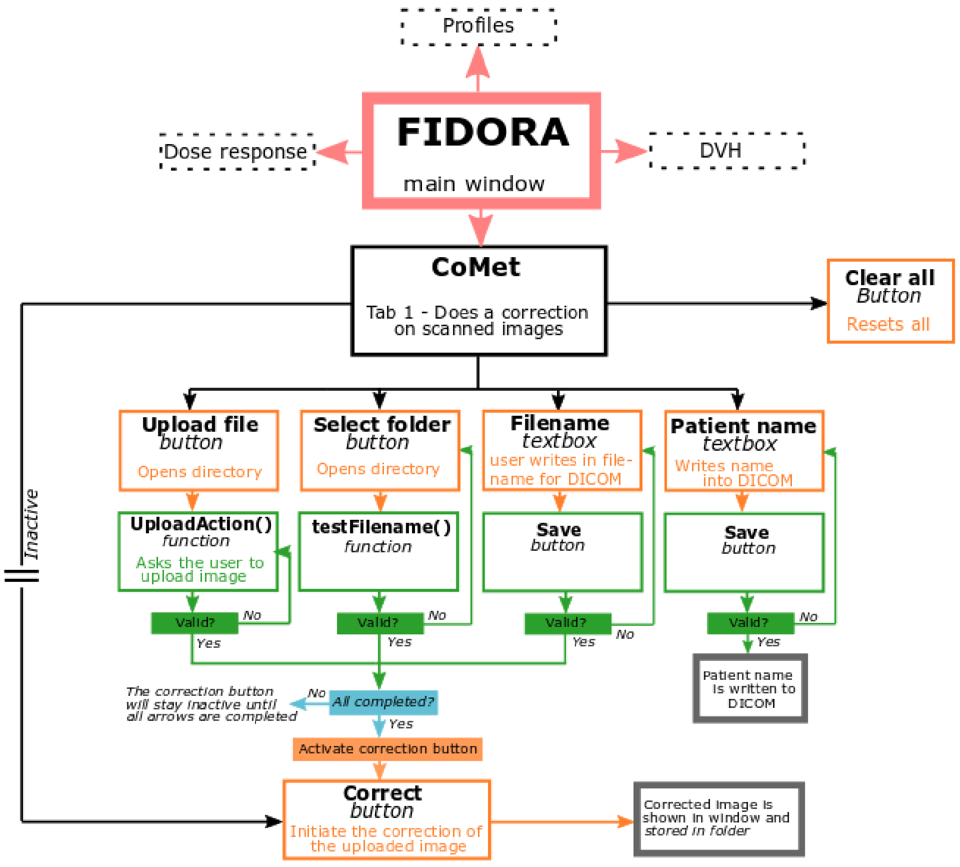

Profiles
FIDORA main window
Dose response
DVH
CoMet
Tab 1 - Does a correction on scanned images
Clear all Button
Resets all
Inactive
Upload file button
Opens directory
Select folder button
Opens directory
Filename textbox
user writes in file-name for DICOM
Patient name textbox
Writes name into DICOM
UploadAction() function
Asks the user to upload image
testFilename() function
Save button
Save button
Valid?
No
Yes
Patient name is written to DICOM
The correction button will stay inactive until all arrows are completed
All completed?
Activate correction button
Correct button
Initiate the correction of the uploaded image
Corrected image is shown in window and stored in folder

Figure 3.10: Flowchart showing the main steps in CoMet.

These *.txt files will be read every time Fidora is opened and stored as a 1270x1016 correction matrix for each color channel. As stated in Section 3.1, when the user uploads an image it will be read in Fidora as three 1270x1016 matrices. Fidora will then use the correction matrices and element by element perform an absolute subtraction on the uploaded matrices. Fidora is written so this will happen automatically every time the user uploads a scanned image. In the CoMet tab the user can perform this correction and the three resulting datasets, representing each color channel, are merged together to form a full RGB image and will be plotted in the program as well as saved in a DICOM. The uploaded image must have been scanned according to the scanning specifics in Table 3.3. The corrected image is stored in a DICOM file, as to not get any data loss and to use a standard which is known in the radiation clinic.

Verification of the correction matrix

To verify the need and the consequence of the correction matrix a comparison of a corrected profile vs. a non-corrected profile across the lateral scanning direction was made. 4 film pieces were cut into 10cm$x$10cm squares and irradiated to doses 0, 50, 100 and 200cGy using a flat beam of size 10cm$x$10cm. After being scanned each image was created into a difference map using the centre datapoint to subtract from all other points. Assuming the radiation beam was in fact flat, each element then holds the difference due to the scanner surface position. A profile was drawn at the same position in each dataset, giving 4 profiles, which were averaged. The image processing was repeated but with a correction before creating the difference map. The setup can be seen in Figure 3.11.

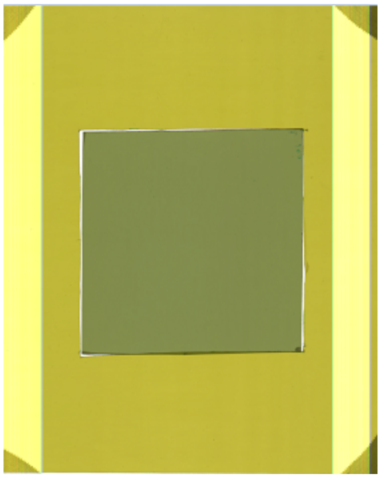


Figure 3.11: Setup en experiment verifying the correction.

3.2.2 Dose response

The second tab in Fidora, Dose response, is created to perform a calibration of the radiochromic film and further investigate the dose response curve of the film. To be able to map the pixel value read when scanning a radiochromic film to a dose it is necessary to create a method built on algorithms modelling the properties of the film. As stated in Section 2.3.3 the model used is a 'reciprocal linear vs dose' plot. Because of the nature of the equation (Equation 2.18) at least three measurements are needed to perform a fitting. That means that the user has to upload scanned films belonging to at least three different dose levels before Fidora will fit the dose response, and the more data points being uploaded the better the fit will be. Figure 3.12 shows a flow chart of the tab Dose Response.

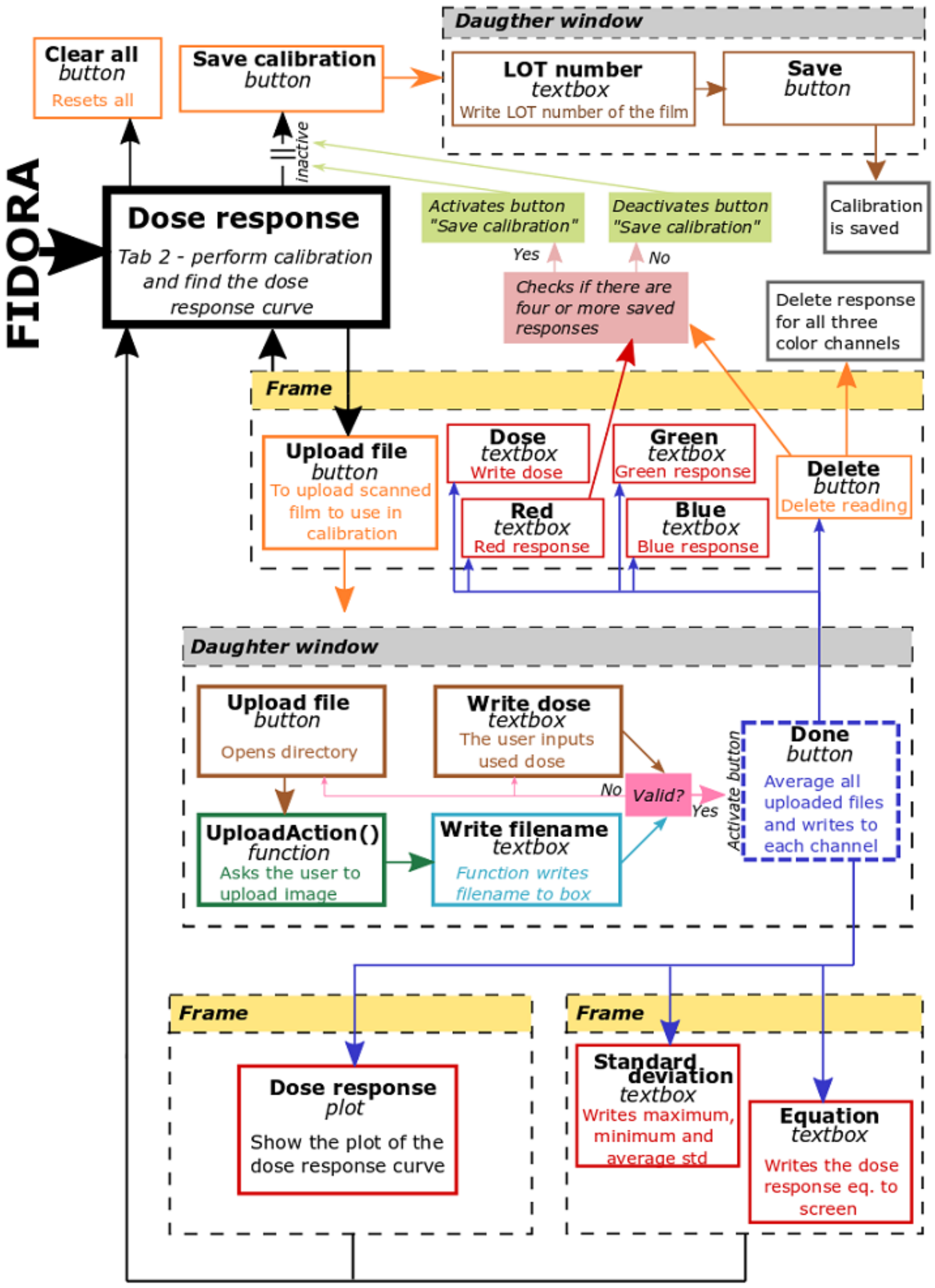

FIDORA
Clear all button
Resets all
Save calibration button
Daugther window
LOT number textbox
Write LOT number of the film
Save button
Calibration is saved
inactive
Dose response
Tab 2 - perform calibration and find the dose response curve
Activates button "Save calibration"
Deactivates button "Save calibration"
Yes
No
Checks if there are four or more saved responses
Delete response for all three color channels
Frame
Upload file button
To upload scanned film to use in calibration
Dose textbox
Write dose
Green textbox
Green response
Red textbox
Red response
Blue textbox
Blue response
Delete button
Delete reading
Daughter window
Upload file button
Opens directory
Write dose textbox
The user inputs used dose
No
Valid?
Yes
Activate button
Done button
Average all uploaded files and writes to each channel
UploadAction() function
Asks the user to upload image
Write filename textbox
Function writes filename to box
Frame
Dose response plot
Show the plot of the dose response curve
Frame
Standard deviation textbox
Writes maximum, minimum and average std
Equation textbox
Writes the dose response eq. to screen

Figure 3.12: Flowchart showing the main steps in the tab Dose response.

The following procedure explains how a calibration done with Fidora should be carried out. First, to avoid the uncertainties related to the beam non-uniformity in the linear accelerator and non-uniform scanning in the lateral direction the calibration should be done with each film piece being irradiated in the centre of the beam field and scanned at the scanner surface centre. In specified scanner used with Fidora, Epson v750 Pro, the geometrical centre of the scanner surface is not the actual centre of the scanned image, meaning that the correct centre must be identified. To do this one option is to scan a mm-paper placed at the corner of the scanner surface and use the readout to find the distance to the actual centre of the scanned image, see Figure 3.13.

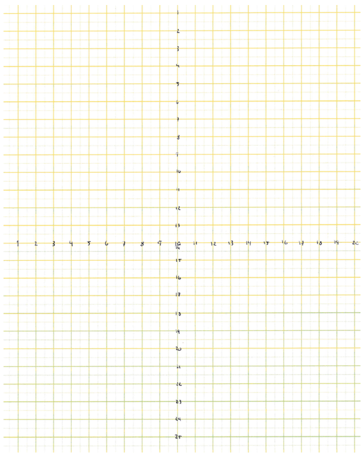


Figure 3.13: An example of how to find the center of the scanner.

When this is done a mask should be cut out to fit the scanner surface such that the centre is easily accessible for later use. That means that relatively small film pieces can be used, e.g. 2cm$x$2cm. Because of the nature of the 'reciprocal linear vs dose'-model the dose levels used for calibration should arise from a geometrical series, e.g. 0, 2, 4, 6, 8 etc. To avoid film-to-film uncertainties several film pieces should be irradiated for each dose level. During irradiation, the film must be placed perpendicular to the radiation beam, and the beam must be of a larger size than the cut film piece. After irradiation, the film pieces are to be kept in an opaque envelope for at least 12 hours to avoid any post-irradiation effects (Ashland (2020)). When the film pieces are scanned it is assumed this happens according to the specifications in Table 3.1.1, and Fidora will reject any files not following these. In the tab Dose response in Fidora the files can then be uploaded (see Figure 3.12), either as groups of files each belonging to different dose levels, or one by one. It is recommended to scan each film piece several times to minimize the scan-to-scan uncertainty. Each file being uploaded will be read according to Section 3.1 giving three 1270x1016 matrices (red, green and blue channels). From these the centre is read as an average over an area of $5mmx5mm$, corresponding to 25x25 pixels in the matrices. As all files have been uploaded Fidora average all measured pixel values from every scan belonging to the same dose level. When enough datapoints are collected to perform a fitting Fidora will do the fitting automatically, and for every new datapoint added it will be further fitted. In addition, a plot showing the dose response curve is created and maximum, minimum and average standard deviation in the datapoints as well as the fitting uncertainty are printed to the screen. To perform the fitting a Python function called $curve\_fit$ from the library $scipy.optimize$ was used. The choice of method for the curve fitting was Levenberg-Marquardt algorithm that solves a least squares problem (see Section 2.3.3). This was chosen because the fitting was without constraints.

Testing the calibration done by Fidora

Since Fidora assumes that the calibration is done from datapoints in the centre of the scanner. That means that Fidora will always read the same positions in the images and could be subject to an uncertainty related to positioning of the film in the scanner, as the film piece could be placed shifted related to the centre. To test this a calibration was done in Fidora and compared to a manually performed calibration. By not using a general approach, but instead study each scanned image for the correct beam centre, the calibration will constitute a much more extensive workload, but will minimize the uncertainty related to positioning of the film in scanner. Film pieces of size 2cm$x$2cm were cut and irradiated to dose 0, 1, 3, 10, 33, 100, 333, 1000 and 2000cGy. All films were scanned according to the scanning specifications given in Table 3.3. The scanning setup of the experiment can be seen in Figure 3.14. First the images were run through Dose response in Fidora, which gives the dose response curve and the fitting parameters a, b and c in Equation 2.18. In addition, a manual calibration in Python were performed by reading and searching every scanned image for the correct position of the centre of the beam field. This was done by measuring all horizontal and vertical profiles across the film and stepwise moving towards the global maximum in each direction starting from one corner. Then, as the centre was found, an average over a 5mm$x$5mm area at that point was calculated. All scans from the same dose level were averaged and the data stored as a dataset, holding dose level and associated measured pixel value. Using the same Python function, $curve\_fit$, as Fidora the dataset was fitted to Equation 2.18. The two dose response curves were plotted against each other, and compared both in fitting uncertainty and directed Hausdorff distance.

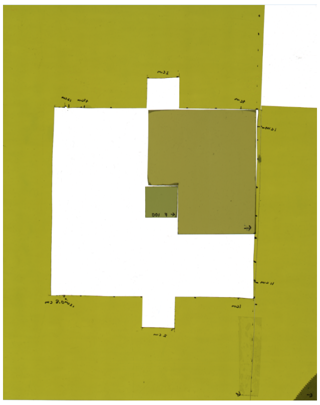


Figure 3.14: Scanner setup when doing a calibration.

In stereotactic treatment it is common to use a beam without the flattening filter. Since a flattening filter free beam is non uniform (see Figure 3.15) it is more sensitive to positional errors during both irradiation and scanning. Because of this a calibration done using a flattening filter free beam was also performed in the same matter as the experiment explained above, to investigate how Fidora performs and if there is a need to do separate calibrations in cases where filter free beams are used.

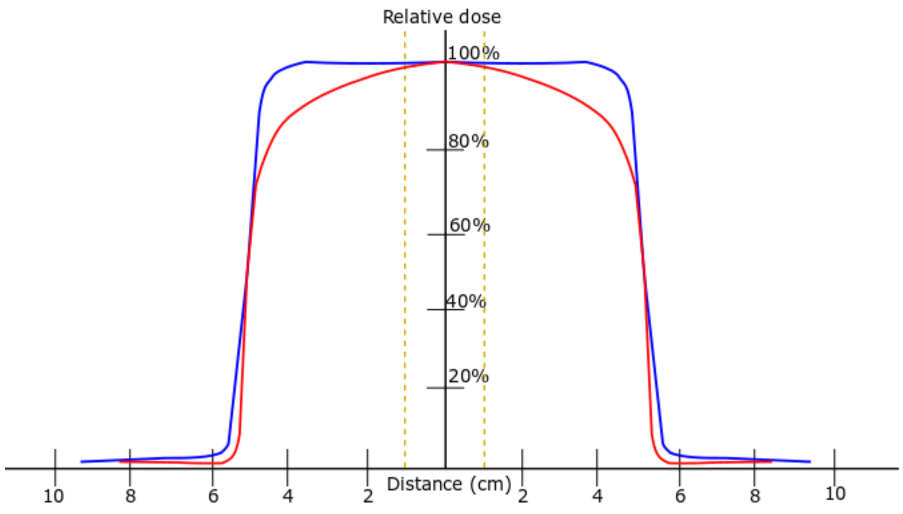
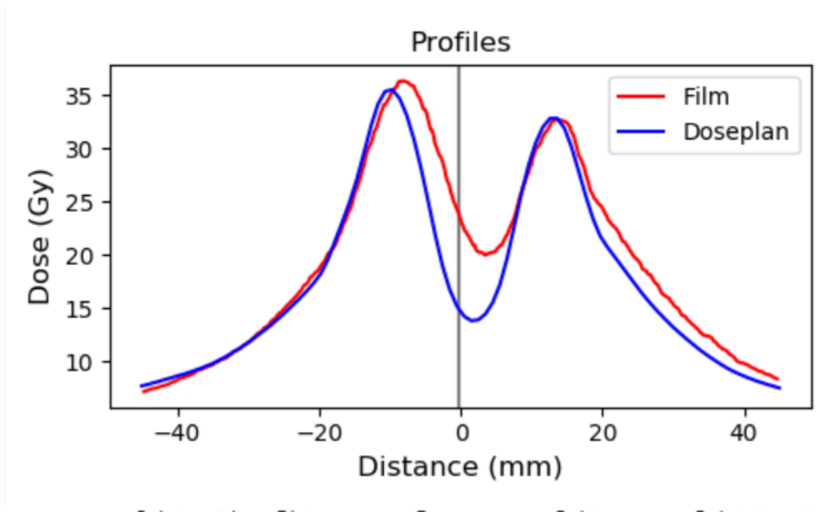


Figure 3.15: An illustration of the effects of a flattening filter. Red line is filter free and blue is the beam profile when using a flattening filter. The yellow dashed line indicates the profile corresponding to the size of the film pieces used in the experiment.

3.2.3 Profiles

Profiles, the fourth tab in Fidora, is created to study how the measured dose profiles in the radiochromic film compares to the planned dose distribution. It results in two profiles plotted together, one profile belonging to the film and one belonging to the dose plan. Such a comparison would only be valid if the profiles are drawn at the exact same position in the dose plan as in the film. Therefore, a critical part of this functionality is to correctly match the position of the film inside the dose matrix uploaded through the RT Dose DICOM file from the treatment planning system. By strictly following the specifications described in Section 3.1.1 the film will be placed correctly. However, there will always be some uncertainty related to positioning the film inside of the phantom during irradiation and when marking the reference point/isocenter in Fidora. To make up for this Fidora gives the user the opportunity to adjust the placements of the region of interest in the scanned film. To create the profiles the user needs to upload the scanned image of the film, RT Dose file and a RT Plan file. The RT Plan file is used for positioning, while the scanned film and RT Dose file is used to find the profiles. Fidora is written with three options when plotting the profiles, it can be calculated over a horizontal line, a vertical line or a manually drawn line. The map the pixel values read in the scanned image Fidora uses a saved calibration from the same film LOT. Together with the plotted profiles a table of information about the plot is printed to the window as the computer mouse hover over the plot. The table gives information about the match between the two profiles as a percentage at given point, the doses at the given point in plot, how the dose at each profile compare to maximum dose in ROI and how the dose in each profile compare to the target dose (see Figure 3.16).



	Point match	Distance	Dose	Rel. to max	Rel. to target
Film:	64.07	-0.24	23.72	65.29	55.48
Doseplan:	64.07	-0.24	15.2	42.81	35.55

Figure 3.16: Example of the profiles and table with information.

Once the profiles are matched Fidora gives the opportunity to export the plot to the computer. In addition to the plot an image of the ROI in both the film and dose plan will be shown in the window so the user are able to visually check that the positioning of the film in the dose plan is as expected. In the case where the user chooses to manually draw the profiles, an algorithm called Bresenham's line algorithm is used (described in Section 2.5). Such an algorithm has to be used because the dataset representing the images is a matrix, meaning that a random line drawn from one point to another will in most cases not created a continuous line through the coordinates but needs to be approximated. Figure 3.17 show a flow chart of the functions in Profiles.

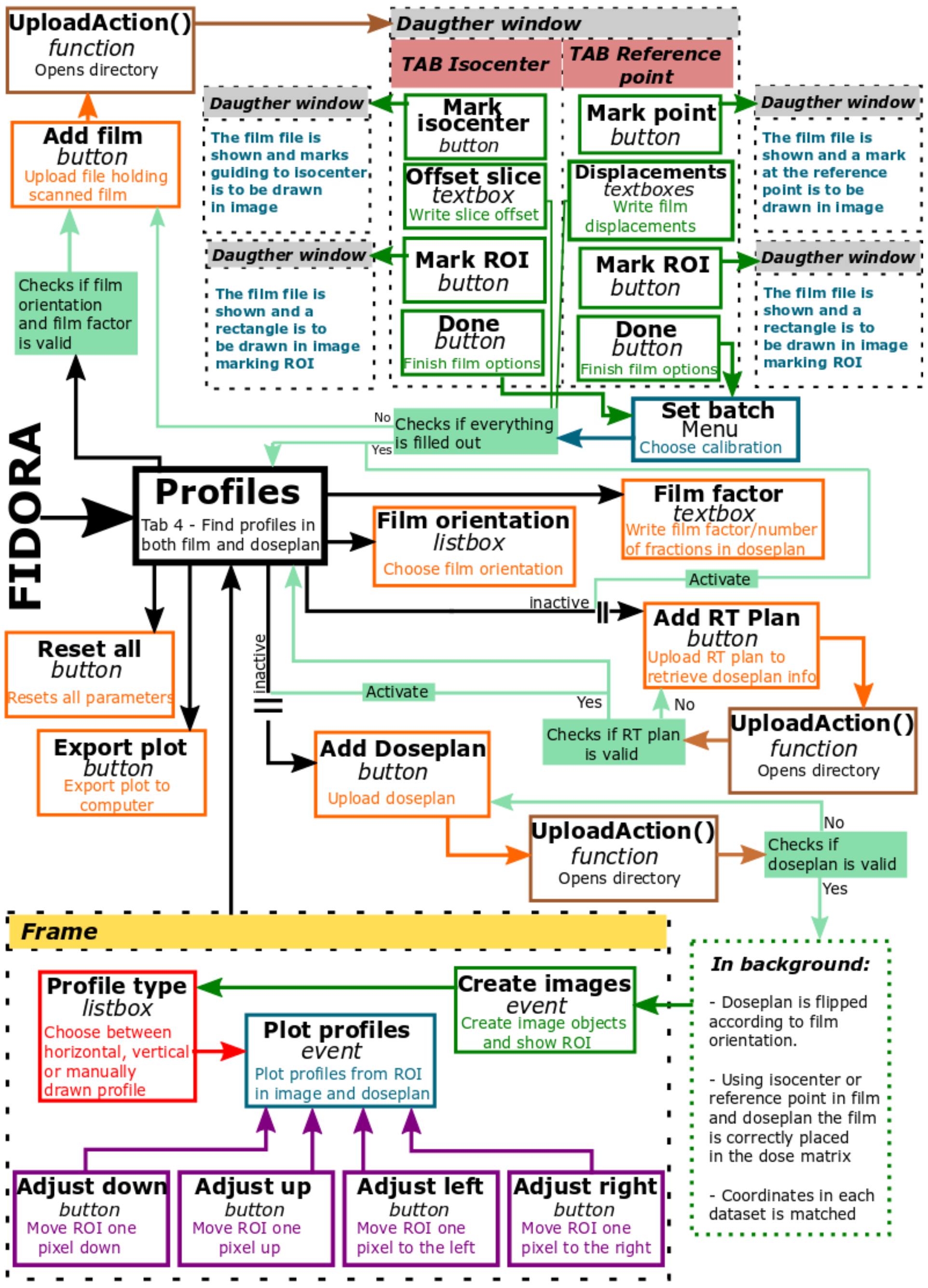

UploadAction() function Opens directory
Daugther window
TAB Isocenter
TAB Reference point
Daugther window
The film file is shown and marks guiding to isocenter is to be drawn in image
Mark isocenter button
Mark point button
Daugther window
The film file is shown and a mark at the reference point is to be drawn in image
Add film button Upload file holding scanned film
Offset slice textbox Write slice offset
Displacements textboxes Write film displacements
Daugther window
The film file is shown and a rectangle is to be drawn in image marking ROI
Mark ROI button
Mark ROI button
Daugther window
The film file is shown and a rectangle is to be drawn in image marking ROI
Checks if film orientation and film factor is valid
Done button Finish film options
Done button Finish film options
No
Checks if everything is filled out
Yes
Set batch Menu Choose calibration
FIDORA
Profiles
Tab 4 - Find profiles in both film and doseplan
Film factor textbox Write film factor/number of fractions in doseplan
Film orientation listbox Choose film orientation
Activate
inactive
Add RT Plan button Upload RT plan to retrieve doseplan info
Reset all button Resets all parameters
inactive
Activate
Yes
No
Checks if RT plan is valid
UploadAction() function Opens directory
Export plot button Export plot to computer
Add Doseplan button Upload doseplan
No
UploadAction() function Opens directory
Checks if doseplan is valid
Yes
Frame
In background:
- Doseplan is flipped according to film orientation.
- Using isocenter or reference point in film and doseplan the film is correctly placed in the dose matrix
- Coordinates in each dataset is matched
Profile type listbox Choose between horizontal, vertical or manually drawn profile
Create images event Create image objects and show ROI
Plot profiles event Plot profiles from ROI in image and doseplan
Adjust down button Move ROI one pixel down
Adjust up button Move ROI one pixel up
Adjust left button Move ROI one pixel to the left
Adjust right button Move ROI one pixel to the right

Figure 3.17: Flow chart showing the main steps in the tab Profiles.

3.2.4 Dose volume histogram (DVH)

The fifth tab in Fidora is where the user can create dose volume histograms. This functionality is similar to the tab Profiles, in the way that is matches the position of the film onto the dose plan in the same matter. In this tab the user must upload an addition file, the structure file, so Fidora can find and recognize different structures in the defined region of interest. From the structure file Fidora gets all available contours and every coordinate from the whole dose plan given as the boundaries of each contour. Upon performing the position matching of the film and dose plan Fidora maps all coordinates given for each contour over to the coordinate system for which both the film and dose plan is given (see Figure 3.5). Fidora then retrieves all coordinates defining in which slice the contour boudary belongs and eliminate all boundary contours which is not in the same slice as the film. The structure file does not hold a complete set of all coordinates in the boundary of the contours, meaning that the coordinates retrieved in the slice will not make a closed path. In addition, because the film has a much better resolution compared to the dose plan, 0.2mm/pixel compared to 1mm/pixel, 2mm/pixel or 3mm/pixel the mapping of the coordinates to the dataset from the film also creates an unclosed path. To close the paths in both the dose plan dataset and in the film dataset Fidora uses a function called Path() from the library $matplotlib.path$. The function takes an unclosed path as parameter and close it by using straight lines and returning a polygon. Then Fidora uses another function in the same library called contains_points() and get in return every point in the region of interest which is contained in that, now closed, contour. Using these two functions Fidora places all the contours and calculates the dose in every given point inside each contour. Then an array holding the dose and what percentage of the volume has reached the corresponding dose in each element is calculated. This array is then plotted for each contour. On the screen the user will get dose volume histograms for every possible contour from the structure file, with the possibility to hide any one of the contours in the plot. Together with the plot an image showing the region of interest in the dose plan is drawn to the screen. Once the user has selected the volumes of interest it is possible to export the plot to save on the local computer for future study. Figure 3.18 show a flow chart of the workflow in the tab DVH.

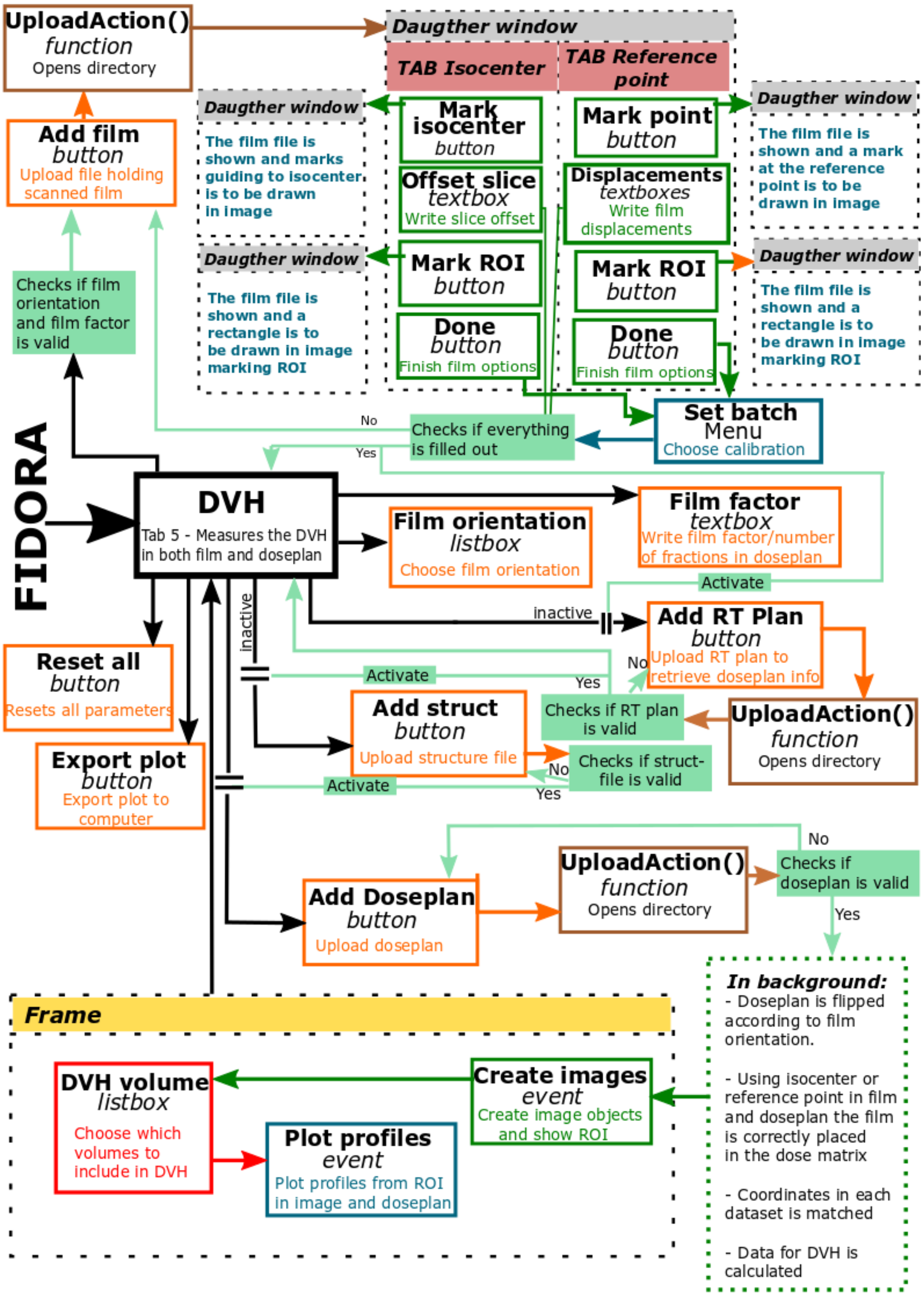

UploadAction() function Opens directory
Add film button Upload file holding scanned film
Checks if film orientation and film factor is valid
FIDORA
Daugther window
TAB Isocenter
TAB Reference point
Daugther window
The film file is shown and marks guiding to isocenter is to be drawn in image
Mark isocenter button
Offset slice textbox Write slice offset
Mark point button
Displacements textboxes Write film displacements
Daugther window
The film file is shown and a mark at the reference point is to be drawn in image
Daugther window
The film file is shown and a rectangle is to be drawn in image marking ROI
Mark ROI button
Done button Finish film options
Mark ROI button
Done button Finish film options
Daugther window
The film file is shown and a rectangle is to be drawn in image marking ROI
Set batch Menu Choose calibration
No
Yes
Checks if everything is filled out
DVH
Tab 5 - Measures the DVH in both film and doseplan
Film orientation listbox Choose film orientation
Film factor textbox Write film factor/number of fractions in doseplan
Activate
inactive
inactive
Add RT Plan button Upload RT plan to retrieve doseplan info
No
Yes
Activate
Reset all button Resets all parameters
Add struct button Upload structure file
Checks if RT plan is valid
UploadAction() function Opens directory
Export plot button Export plot to computer
No
Checks if struct-file is valid
Activate
Yes
No
Checks if doseplan is valid
Yes
UploadAction() function Opens directory
Add Doseplan button Upload doseplan
In background:
- Doseplan is flipped according to film orientation.
- Using isocenter or reference point in film and doseplan the film is correctly placed in the dose matrix
- Coordinates in each dataset is matched
- Data for DVH is calculated
Frame
DVH volume listbox Choose which volumes to include in DVH
Create images event Create image objects and show ROI
Plot profiles event Plot profiles from ROI in image and doseplan

Figure 3.18: Flow chart showing the main steps in the tab DVH

3.3 Stereotactic radiotherapy

As described in Section 2.4.8 radiochromic film, such as GafChromic EBT3, is a good dosimeter when studying stereotactic treatment in radiotherapy. When developing Fidora, the possible use in stereotactic treatment was in mind, and as such there is a need to investigate how Fidora handles stereotactic cases.

3.3.1 MLC model in Raystation

When doing stereotactic radiotherapy the beam sizes are usually relatively small compared to conventional radiotherapy. In addition, the dose gradients are meant to be steep, meaning that the beam shaping is important in stereotactic treatment. The largest uncertainty in this matter is the MLC, and how they are modelled in the treatment planning system, see Section 2.4.6. Especially how the MLC is modelled at their tip and side will be important in regards to shaping the beam field. To see how well the treatment planning system fits the actual measured dose distribution a series of measurements were performed. First a calibration was performed in Fidora using dose levels 0, 10, 20, 40, 80, 160 and 320cGy. Then five film pieces of sizes 5cm$x$5cm, 6cm$x$6cm, 7cm$x$7cm, 9cm$x$9cm and 12cm$x$12cm were cut out and irradiated according to the specifications in Section 3.1.1 with dose 2Gy using beam field sizes of 1cm$x$1cm, 2cm$x$2cm, 3cm$x$3cm, 5cm$x$5cm, 10cm$x$10cm respectively. The profiles were drawn across the centre of each field, one x-profile across the collimator jaws and one y-profile across the MLC-tip, in Fidora. See Figure 3.19 for setup.

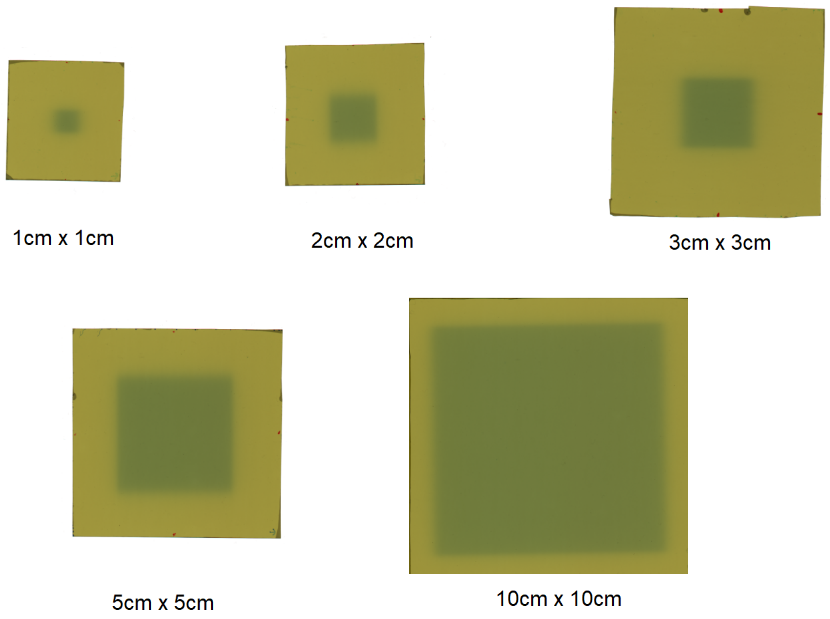


Figure 3.19: Setup in MLC model experiment

One film piece of size 12cm$x$12cm was also cut out and irradiated with dose 2Gy using a beam field size of 10cm$x$10cm with the left upper quadrant blocked out using the MLC leaves, see Figure 3.20 for setup. Then a x-profile across the collimator jaws was drawn 2.5 cm from the top giving a penumbra defined by the side of the MLC leaves. A y-profile across the MLC leaf-tip was also drawn 2.5cm from the left giving a penumbra defined by the tip of the MLC leaves. Both profiles were found using Fidora. See Figure 3.21 for how the profiles were taken. When irradiating all film pieces, the film was placed at a depth of 10 cm in a solid water phantom.

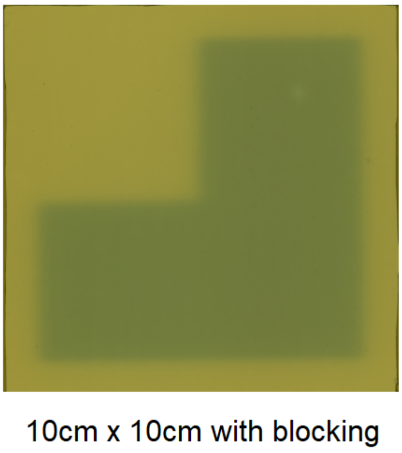


Figure 3.20: An illustration of how the MLC were used to shape the partially blocked 10cm$x$10cm field.

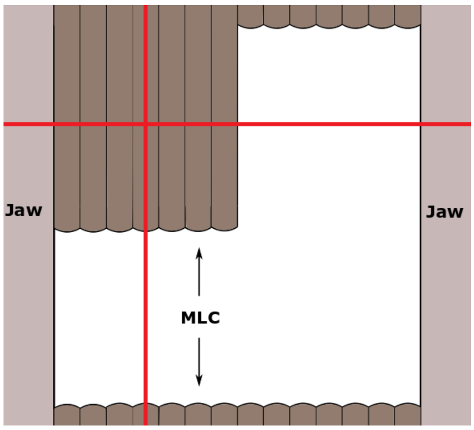


Figure 3.21: An illustration of how the MLC were used to shape the partially blocked 10cm$x$10cm field. The red lines indicate where the profiles were taken.

3.3.2 Stereotactic treatment plans

To see how the GafChromic EBT3 film works with Fidora compared to the treatment plan four treatment plans were created,

- Treatment plan V1
- Treatment plan V2
- Treatment plan V3
- Treatment plan V4

The phantom used is shown in Figure 3.22, together with all the delineated volumes of interest.

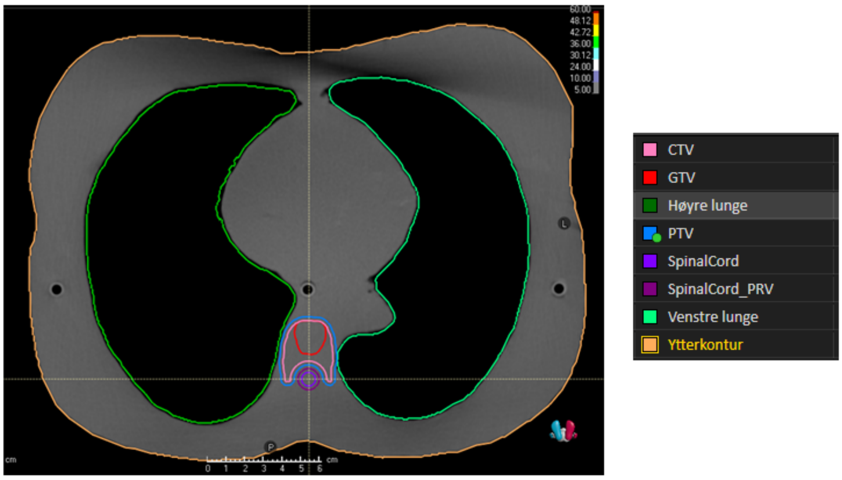


Figure 3.22: Phantom showing the different volumes of interest.

All the treatment plans are planned with three fractions delivered using a beam energy of 6MV and a criterion saying that 99% of GTV should be covered by 95% isodose. V1-V4 are clinically relevant treatment plans and are optimized with the following parameters,

- Standarization 12.5Gy x 3 fractions,
- GTV > 35.6Gy,
- CTV > 30Gy,
- PTV > 25.1Gy,
- PTV$_{max} < 52.2$Gy,
- SpinalCord PRV < 18Gy at 0.35 cm^3 and
- SpinalCord PRV < 21.9Gy at 0.01cm^3.

As the prescribed dose in V1-V4 is up to around 12.5Gy for each fraction, and the radiochromic film has an optimum region up to 10 Gy, four additional, similar plans were made with a lower prescribed dose. Each of them technically the same as the four plans mentioned above but re-planned so that the prescribed dose was scaled down to within the functional region of the film. the plans, named V5, V6, V7 and V8, are scaled versions of V1, V2, V3 and V4 respectively. The standardization dose in V5-V8 is 5Gy x 3 fractions. These plans are not as clinically relevant but will be used to see if the measured high dose regions are in compliance with the dose plans. In all plans the isocenter was placed in the centre of the spinal cord (see Figure 3.22), and the optimization was done with a sliding window with a maximum of 2cm MLC movement for each gantry angle.

	Collimator angle	Patient table angle	Number	Degrees of arcs on arcs	Flattening filter
V1	5°	5°	1	178°-356°	Yes
V2	5°	5°	1	178°-356°	No
V3	5°	5°	2	178°-90° and 270°-182°	No
V4	90°	5°	1	178°-356°	No
V5	5°	5°	1	178°-356°	Yes
V6	5°	5°	1	178°-356°	No
V7	5°	5°	2	178°-90° and 270°-182°	No
V8	90°	5°	1	178°-356°	No

Table 3.4: Specifications for treatment plans V1-V8. Arc rotation direction is in all cases counter-clockwise.

In V3 two arcs were used in an attempt to lower the dose to organs of risk by avoiding radiation being delivered to the front of the phantom. In V4 the collimator rotated at 90 degrees to let the jaws block out the spinal cord instead of the MLC-leaves. Here the jaws are locked at the edge of the region called SpinalCord PRV (see Figure 3.22), which is defined as the spinal cord + a 2mm margin.

When performing the irradiation the phantom was placed on the patient table in treatment position Head First Supine (see Figure 3.4) and isocenter was placed in a marked reference point. Then the table was moved -0.2cm, -17.53cm and -2.53cm in x, y and z directions (see Figure 3.2). These values were given by the treatment plan as a reference from the marked reference point to placed isocenter. Then eight film pieces of size 10cm$x$10cm were cut out and irradiated in turn. Each film piece was only given one out of three fractions from the dose plan. When placed in the correct slice of the phantom a small piece of the top of the film piece had to be cut out to make the piece fit inside the phantom, see Figure 3.23. Also, markings showing the isocenter on the film were made for later use in Fidora.

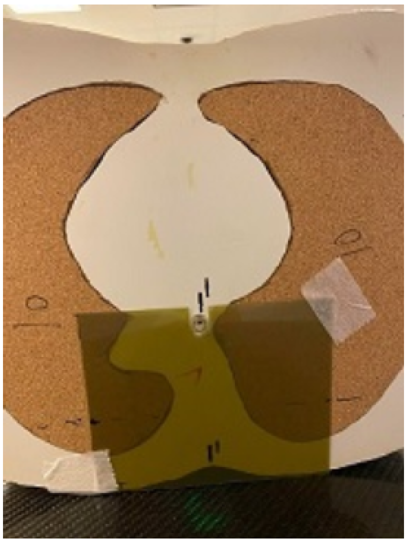


Figure 3.23: Image from the setup in the linac.

As the film pieces were scanned all instructions in Section 3.1.1 were followed, using a mask made of radiochromic film to best place the film at the centre of the scanner surface. See Figure 3.24 for the setup during scanning.

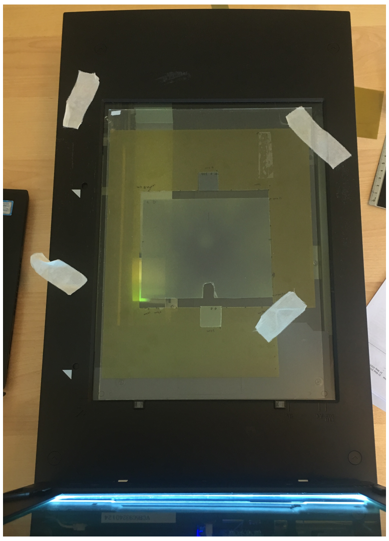


Figure 3.24: Setup during scanning of the film.

To analyse the results in Fidora the dose plan and film was compared by drawing profiles, as seen in Figure 3.25 and looking at the dose volume histograms. Since each film only had received one fraction, whereas three were planned, the film had to be upscaled 3 times in Fidora.

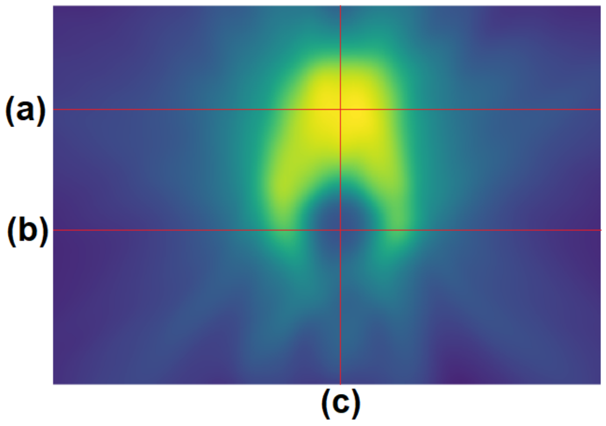


Figure 3.25: Illustration of how the profiles were drawin in the treatment plans. The red lines indicate the positions of the profiles. In V1-V4 all three profiles (a, b and c) were plotted, while in treatment plans V5-V8 only a and c were plotted.

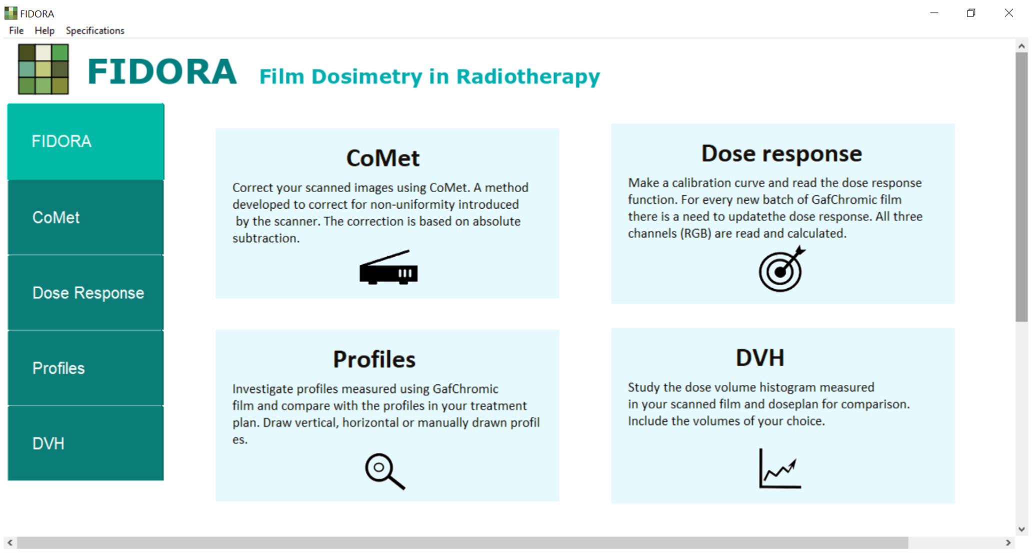
Chapter 4

Results

As radiochromic film is a lesser used dosimeter at the radiation clinic at St. Olavs Hospital there was a need to introduce this dosimeter as a quality assurance instrument, which was the work in a former project done by this thesis author along with another student Ane Vigre Håland (Håland and Gustavsen (2019)). With the quality and usefulness of the film established in that project the need for a flexible analysing tool raised. The main results of this work is the developed analysing tool named Fidora. As such, this chapter will start by going through each module in Fidora and see how each of them turned out, in addition to the results of validation of background correction and calibration procedure in the case of CoMet and Dose Response respectively. Then, as a proof of concept, Fidora has been used to analyse the modelling of the MLC in the treatment planning system and study four different stereotactic treatment plans.

4.1 Fidora

Fidora is an analysing tool that performs the needed corrections on scanned film and calculate the calibrations and dose response curves needed to compare film to dose plans. In addition, Fidora uses profiles and dose volume histograms to analyse the outcome read from film. Fidora was developed with a special focus on irradiation on the breast and stereotactic treatment, but the functionalities included are general and can be used on other cases as well. Figure 4.1 shows the front page of Fidora, where the tabs on the left side holds the functionality menu.



FIDORA
File Help Specifications

FIDORA Film Dosimetry in Radiotherapy

FIDORA
CoMet
Dose Response
Profiles
DVH

CoMet
Correct your scanned images using CoMet. A method developed to correct for non-uniformity introduced by the scanner. The correction is based on absolute subtraction.

Dose response
Make a calibration curve and read the dose response function. For every new batch of GafChromic film there is a need to updatethe dose response. All three channels (RGB) are read and calculated.

Profiles
Investigate profiles measured using GafChromic film and compare with the profiles in your treatment plan. Draw vertical, horizontal or manually drawn profil es.

DVH
Study the dose volume histogram measured in your scanned film and doseplan for comparison. Include the volumes of your choice.

Figure 4.1: Initial page when opening the software Fidora. The tabs on the left hold the different functionalities included in Fidora.

4.1.1 CoMet

Figure 4.2 and 4.3 shows the profiles in the difference maps in the lateral and the scanning directions respectively. It was confirmed that there is a deviation in each direction, but no systematic relationship between the deviation in intensity and dose was found. As a result the correction matrices for all six dose levels were averaged and Fidora only operates with one correction matrix for each color channels.

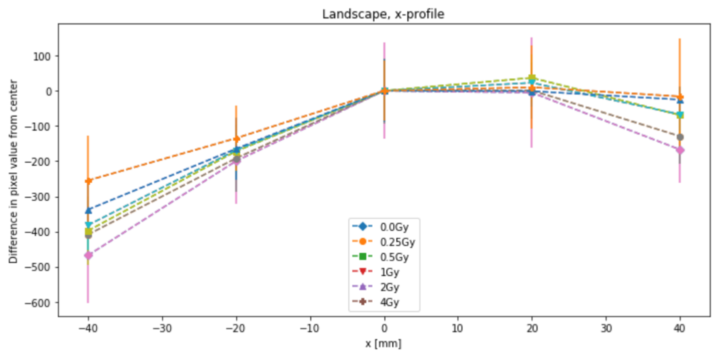


Figure 4.2: Profile of the difference in read pixel value in relation to the centre value. The measurements have been taken across the scanners lateral direction and are plotted with error bars for each dose level.

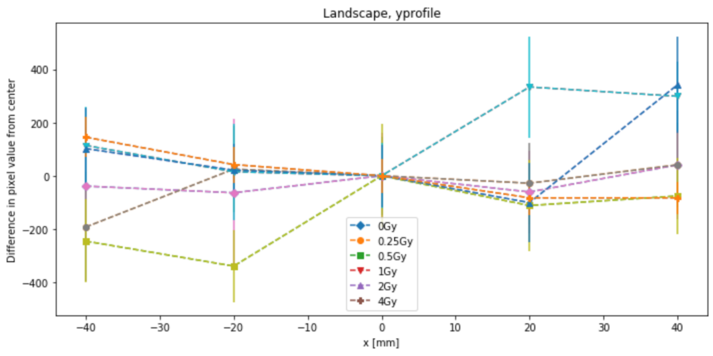


Figure 4.3: Profile of the difference in read pixel value relation to the centre value. The measurements have been taken across the scanners scanning direction and are plotted with error bars for each dose level.

Figure 4.4 and 4.5 shows the profiles in the difference maps in the lateral and the scanning directions respectively. Table 4.1 gives the standard deviation for each measured point in the difference maps as they are averaged over all dose levels.

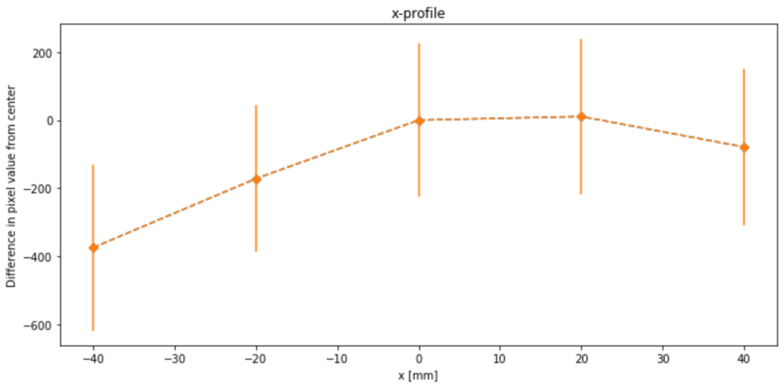


Figure 4.4: Profile of the difference in read pixel value in relation to the centre value as an average over all six dose levels. The measurements have been taken across the scanners lateral direction and are plotted with error bars at each measuring point.

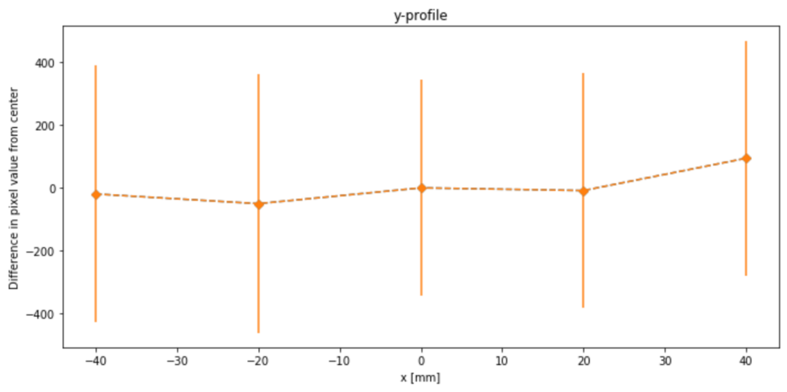


Figure 4.5: Profile of the difference in read pixel value in relation to the centre value as an average over all six dose levels. The measurements have been taken across the scanners scanning direction and are plotted with error bars at each measuring point.

Distance relative to scanner center	**SD in x-profile (pixel value)**	**SD in y-profile (pixel value)**
-40 mm	± 244	± 407
-20 mm	± 215	± 410
0 mm	± 226	± 344
20 mm	± 229	± 373
40 mm	± 229	± 374

Table 4.1: Standard deviation at each measured point when the difference map is averaged over all six dose levels. The standard deviations are given for the profile across the scanner lateral direction as well as the scanners scanning direction.

As a verification of the need and the consequence of using a correction matrix horizontal profiles from non-corrected images were compared to horizontal profiles from corrected images after subtracting the centre element. The profiles were measured across the centre of the scanner surface. The results can be seen in Figure 4.6. It shows that the non-corrected image has a larger error in measured dose than the non-corrected image. The error increases with the distance from the centre of the scanner surface.

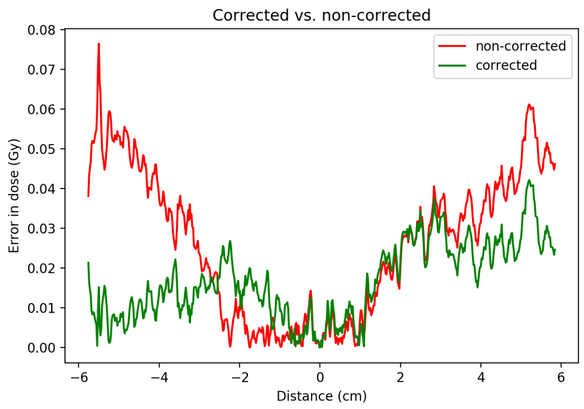
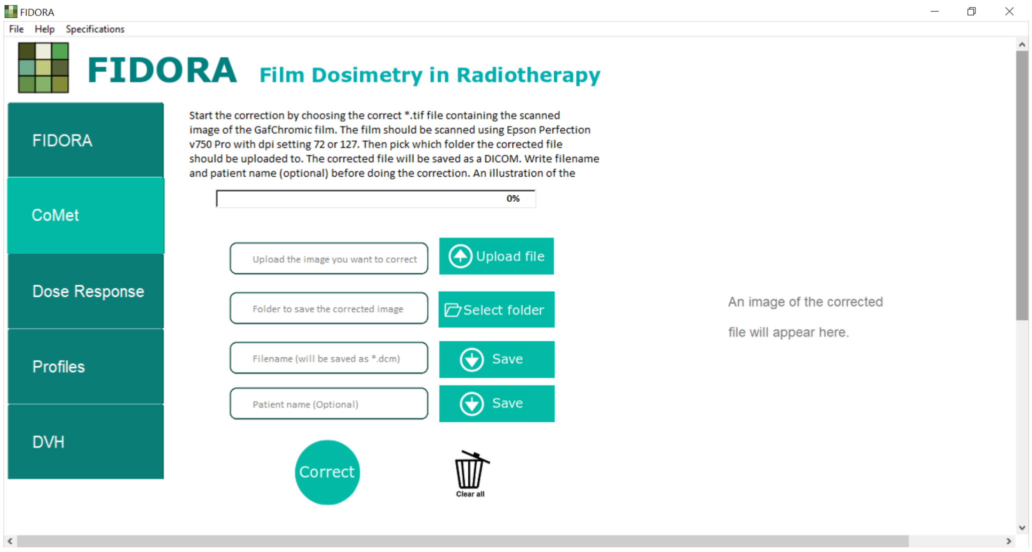


Figure 4.6: Profiles from a corrected image and a non-corrected image shown in green and red, respectively. The profiles are taken in the scanner's lateral direction across the centre of the scanner surface.

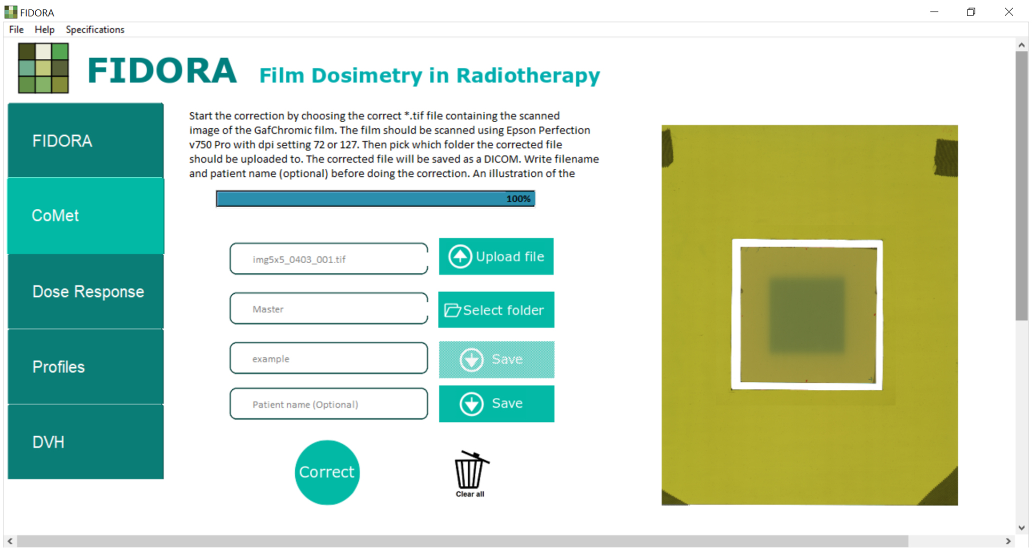
Figure 4.7 shows the result of the Fidora tab meant to perform image corrections, CoMet and Figure 4.8 shows how it looks after a successful run. The the user is asked to upload a file holding the scanned image of film, and to choose a folder where the DICOM file holding the corrected image will be stored. The user will also be asked to input a filename before all these choices will be confirmed as valid and the user can perform the correction by clicking the correction button. Since the corrected image is stored as a DICOM file the user has the option to write in the patient name/ID.


FIDORA
File Help Specifications
FIDORA Film Dosimetry in Radiotherapy
FIDORA
CoMet
Dose Response
Profiles
DVH
0%
Upload the image you want to correct
Upload file
Folder to save the corrected image
Select folder
Filename (will be saved as *.dcm)
Save
Patient name (Optional)
Save
Correct
Clear all

Start the correction by choosing the correct *.tif file containing the scanned image of the GafChromic film. The film should be scanned using Epson Perfection v750 Pro with dpi setting 72 or 127. Then pick which folder the corrected file should be uploaded to. The corrected file will be saved as a DICOM. Write filename and patient name (optional) before doing the correction. An illustration of the

An image of the corrected file will appear here.

Figure 4.7: Screenshot of the first tab in Fidora - CoMet. In this module the user is able to perform background corrections on a scanned image of the film.



Start the correction by choosing the correct *.tif file containing the scanned image of the GafChromic film. The film should be scanned using Epson Perfection v750 Pro with dpi setting 72 or 127. Then pick which folder the corrected file should be uploaded to. The corrected file will be saved as a DICOM. Write filename and patient name (optional) before doing the correction. An illustration of the

img5x5_0403_001.tif

Master

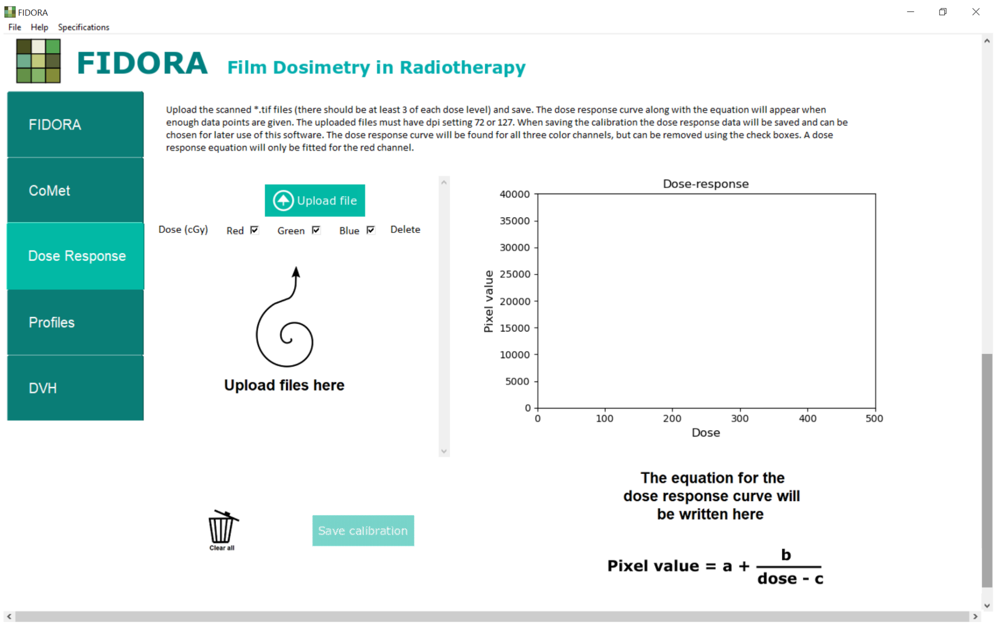
example

Patient name (Optional)

Figure 4.8: Screenshot of the first tab in Fidora, CoMet, after a successful run. The corrected image here is used as an example.

4.1.2 Dose Response

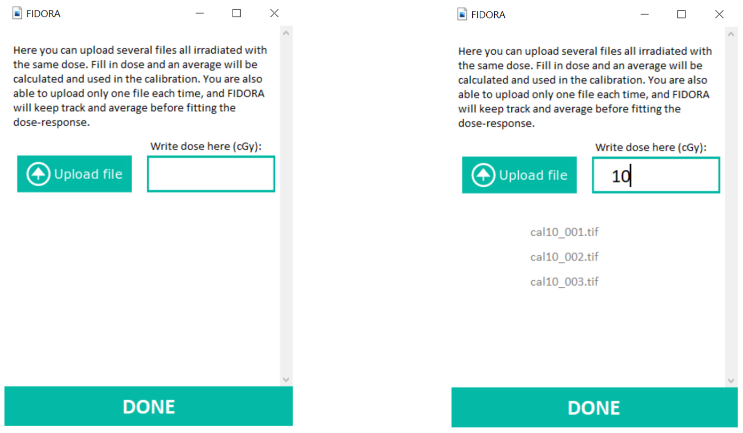
Figure 4.9 shows how Fidoras second tab, Dose response, turned out. The user can upload scanned images of the film and perform a calibration of the film. When uploading the images, the user must first write to Fidora what dose the film is irradiated with, then the user is able to upload several files holding images of film irradiated with the same dose (see Figure 4.10). To decrease any scan to scan uncertainties it is often recommended that one uses several measurements of film irradiated with same dose. All measurements belonging to the same dose level will be averaged. As enough datapoints are collected for Fidora to fit the dose response curve a plot will be shown in the window along with the written equation, minimum, maximum and average standard deviations of the measurements and the total uncertainty of the fitting. For every dose level uploaded to Fidora a string showing the dose and measured response in each color channel is written to the screen. This gives the user an overview of the datapoints and the opportunity to delete a set of datapoints if wanted as well as hide any of the color channels in the plot. Since the properties of the GafChromic film is crucial to its ability to relate dose to pixel value, it is necessary to fit the method of finding the dose response curve with that in mind. Assuming every film sheet belonging to the same LOT number is created under the same conditions, it is reasonable to assume that their properties are the same. As such, it should only be necessary to perform a calibration for every new LOT. Therefore, Fidora gives the possibility for the user to save the performed calibration and register the related LOT number, so that one can reuse it every time Fidora is in use.



Upload the scanned *.tif files (there should be at least 3 of each dose level) and save. The dose response curve along with the equation will appear when enough data points are given. The uploaded files must have dpi setting 72 or 127. When saving the calibration the dose response data will be saved and can be chosen for later use of this software. The dose response curve will be found for all three color channels, but can be removed using the check boxes. A dose response equation will only be fitted for the red channel.

$$\text{Pixel value} = a + \frac{b}{\text{dose} - c}$$

Figure 4.9: Screenshot of the second tab in Fidora - Dose response. In this module the user are able to perform a calibration as well as investigating the dose response curve, which will be plotted on the screen.



Here you can upload several files all irradiated with the same dose. Fill in dose and an average will be calculated and used in the calibration. You are also able to upload only one file each time, and FIDORA will keep track and average before fitting the dose-response.

Write dose here (cGy):

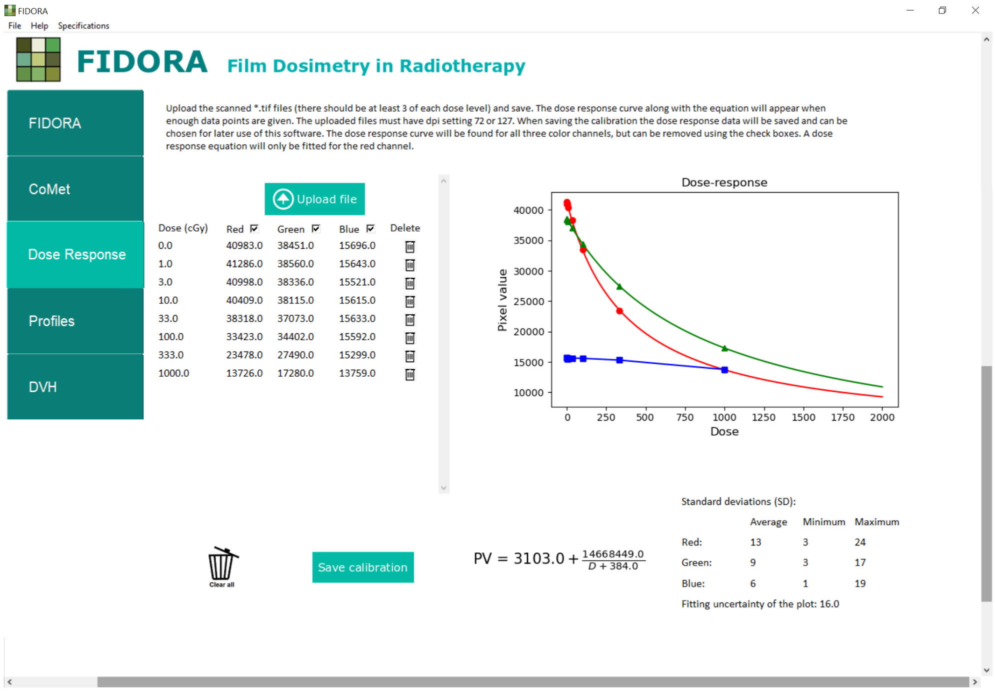
Here you can upload several files all irradiated with the same dose. Fill in dose and an average will be calculated and used in the calibration. You are also able to upload only one file each time, and FIDORA will keep track and average before fitting the dose-response.

Write dose here (cGy): 10

cal10_001.tif
cal10_002.tif
cal10_003.tif

Figure 4.10: Screenshot of the window where the user uploads images of scanned films. The left image shows how the window looks at the beginning and the right image shows how it look when files have been uploaded.

Figure 4.11 shows how the tab Dose Response looks like after enough measurements have been uploaded and the dose response has been fitted. Also, the dose response equation is written to the screen, along with the uncertainties related to the results. Figure 4.12 shows how the window for saving of the calibration looks like. Here the user writes in the LOT-number and saves the calibration to the computer.



Upload the scanned *.tif files (there should be at least 3 of each dose level) and save. The dose response curve along with the equation will appear when enough data points are given. The uploaded files must have dpi setting 72 or 127. When saving the calibration the dose response data will be saved and can be chosen for later use of this software. The dose response curve will be found for all three color channels, but can be removed using the check boxes. A dose response equation will only be fitted for the red channel.

Dose (cGy)	Red ✓	Green ✓	Blue ✓	Delete
0.0	40983.0	38451.0	15696.0	🗑
1.0	41286.0	38560.0	15643.0	🗑
3.0	40998.0	38336.0	15521.0	🗑
10.0	40409.0	38115.0	15615.0	🗑
33.0	38318.0	37073.0	15633.0	🗑
100.0	33423.0	34402.0	15592.0	🗑
333.0	23478.0	27490.0	15299.0	🗑
1000.0	13726.0	17280.0	13759.0	🗑

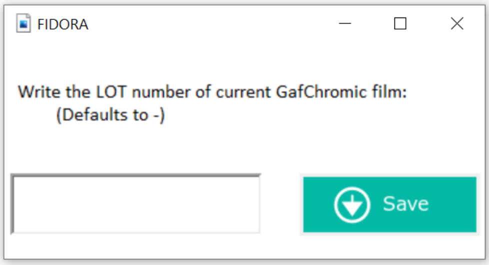
$$PV = 3103.0 + \frac{14668449.0}{D + 384.0}$$

Standard deviations (SD):

	Average	Minimum	Maximum
Red:	13	3	24
Green:	9	3	17
Blue:	6	1	19

Fitting uncertainty of the plot: 16.0

Figure 4.11: Screenshot of how the tab Dose Response looks like after enough measurements have been uploaded and the dose response has been fitted.



Write the LOT number of current GafChromic film:
(Defaults to -)

Figure 4.12: Screenshot of the window where the user can save the performed calibration for later use in Fidora.

Testing the calibration done by Fidora

Figure 4.13 shows the dose response curves when the calibration is performed in Fidora and when the calibration is done with manual readings in Python.

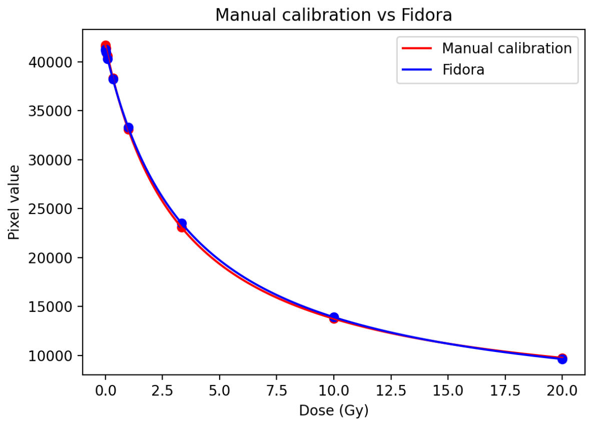


Figure 4.13: Plot showing the dose response curves both when calibration is done in Fidora and manual calibration, with fitting uncertainty ± 0.07Gy and ± 0.02Gy respectively.

Figure 4.14 holds the dose response curves when using a flattening filter free beam as the calibration is done in Fidora and done using a manual calibration in Python. Figure 4.15 shows the dose response curves created in Fidora when using a flattening filter free beam and where the used beam has been subjected to a flattening filter.

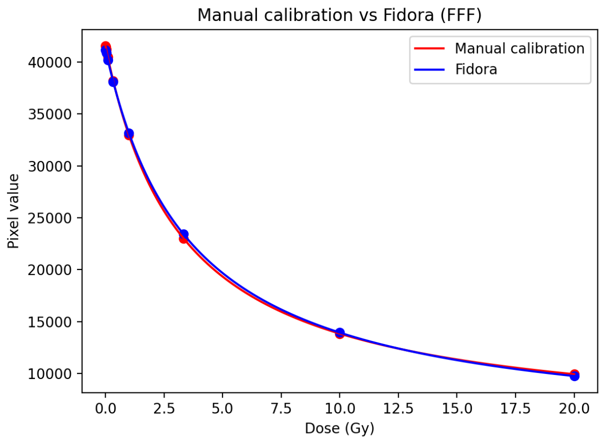


Figure 4.14: Plot showing the dose response curves both when calibration is done in Fidora and manual calibration using a flattening filter free beam, with fitting uncertainty ± 0.06Gy and ± 0.02Gy respectively.

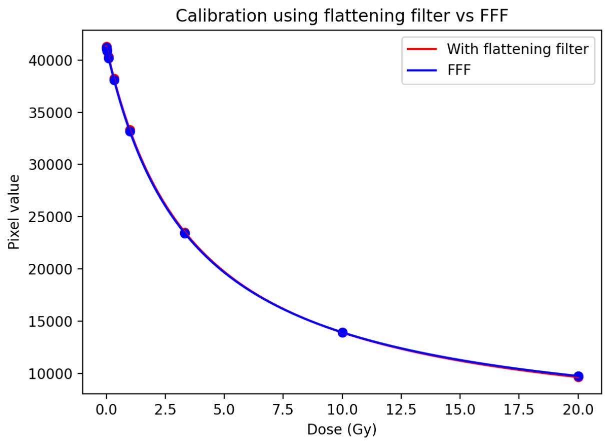


Figure 4.15: Plot showing the dose response curves created in Fidora for one flattening filter free beam and one beam using the filter, with fitting uncertainty ± 0.06Gy and ± 0.07Gy respectively.

Table 4.2 shows the total uncertainty related to fitting of the calibration curve, and Table 4.3 shows the result of the Hausdorff distances (see Section 2.5).

Experiment	**Fitting uncertainty**
Manual calibration	± 0.02 Gy
Calibration in Fidora	± 0.07 Gy
Manual calibration (FFF)	± 0.02 Gy
Calibration in Fidora (FFF)	± 0.06 Gy

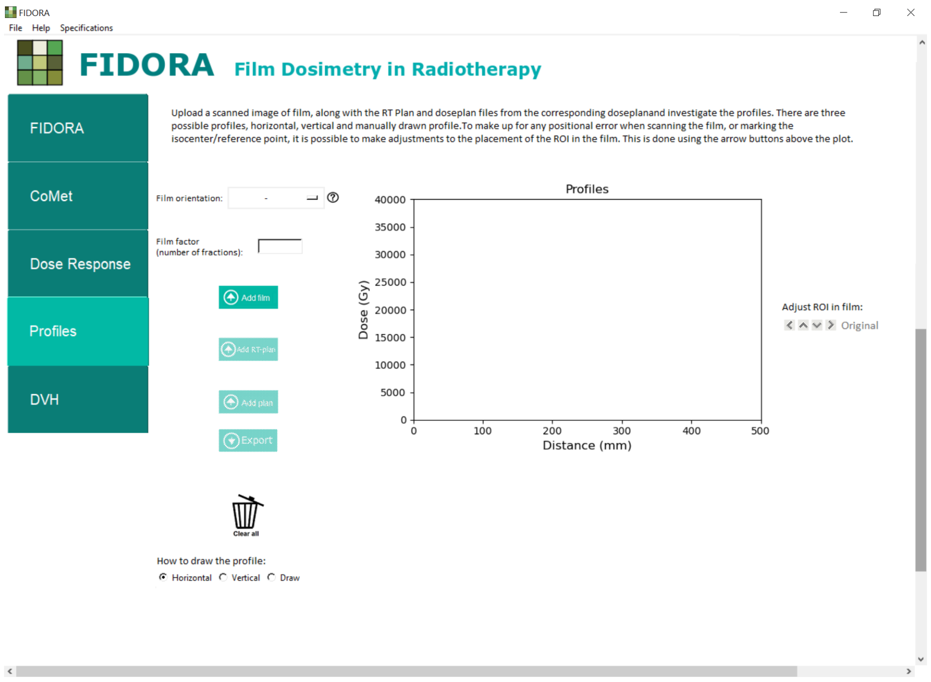
Table 4.2: Total uncertainty related to the fitting of the dose response curves. These numbers will depend on how many calibration points are used.

Experiment	**Directed Hausdorff Distance in dose**
Manual calibration vs Fidora	0.00012 Gy
Manual calibration vs Fidora (FFF)	0.00009 Gy
With filter vs FFF (Fidora)	0.00006 Gy

Table 4.3: Directed Hausdorff distance showing the largest difference in dose along each curve.

4.1.3 Profiles

Figure 4.16 shows how the fourth tab in Fidora, Profiles, turned out. When entering this tab, the user must input the film orientation to be able to move on to uploading the scanned image of the film. In cases where several fractions are used during treatment but the film only has received one fraction, the number of fractions needs to be entered at the textbox referring to it. This is because the dose plan holds the total delivered dose throughout the whole treatment.

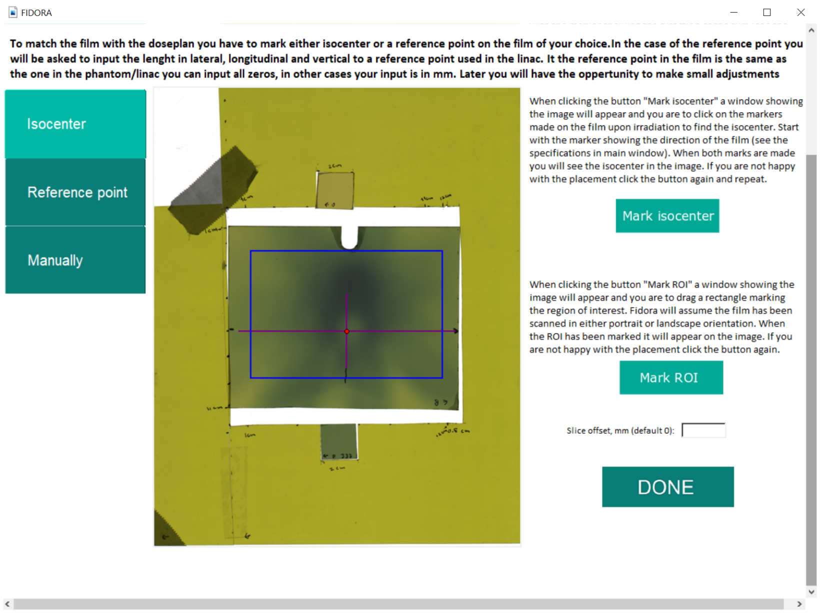


Upload a scanned image of film, along with the RT Plan and doseplan files from the corresponding doseplanand investigate the profiles. There are three possible profiles, horizontal, vertical and manually drawn profile.To make up for any positional error when scanning the film, or marking the isocenter/reference point, it is possible to make adjustments to the placement of the ROI in the film. This is done using the arrow buttons above the plot.

How to draw the profile: (● Horizontal) Vertical Draw

Figure 4.16: Screenshot of the fourth tab in Fidora - Profiles. In this module the user can upload scanned images of film to study horizontal, vertical or manually drawn profiles and compare the results in film and dose plan.

Then the user can upload the scanned film, which will be shown in a new window. In this window the user will select either to input an isocenter or a reference point to be used in cases where film is not placed at the isocenter. In either case the user will mark the image according to the markers on the film (see Section 3.1.1) and select a region of interest defining the area where the measurements will be done. This region will be the same region that is used in the dose plan later. See Figure 4.17 for an example of how this window look in the case where the isocenter is selected. There the purple lines indicate the marked lines leading to isocenter and the red dot indicates isocenter. The blue rectangle is the marked ROI.



To match the film with the doseplan you have to mark either isocenter or a reference point on the film of your choice.In the case of the reference point you will be asked to input the lenght in lateral, longitudinal and vertical to a reference point used in the linac. It the reference point in the film is the same as the one in the phantom/linac you can input all zeros, in other cases your input is in mm. Later you will have the oppertunity to make small adjustments

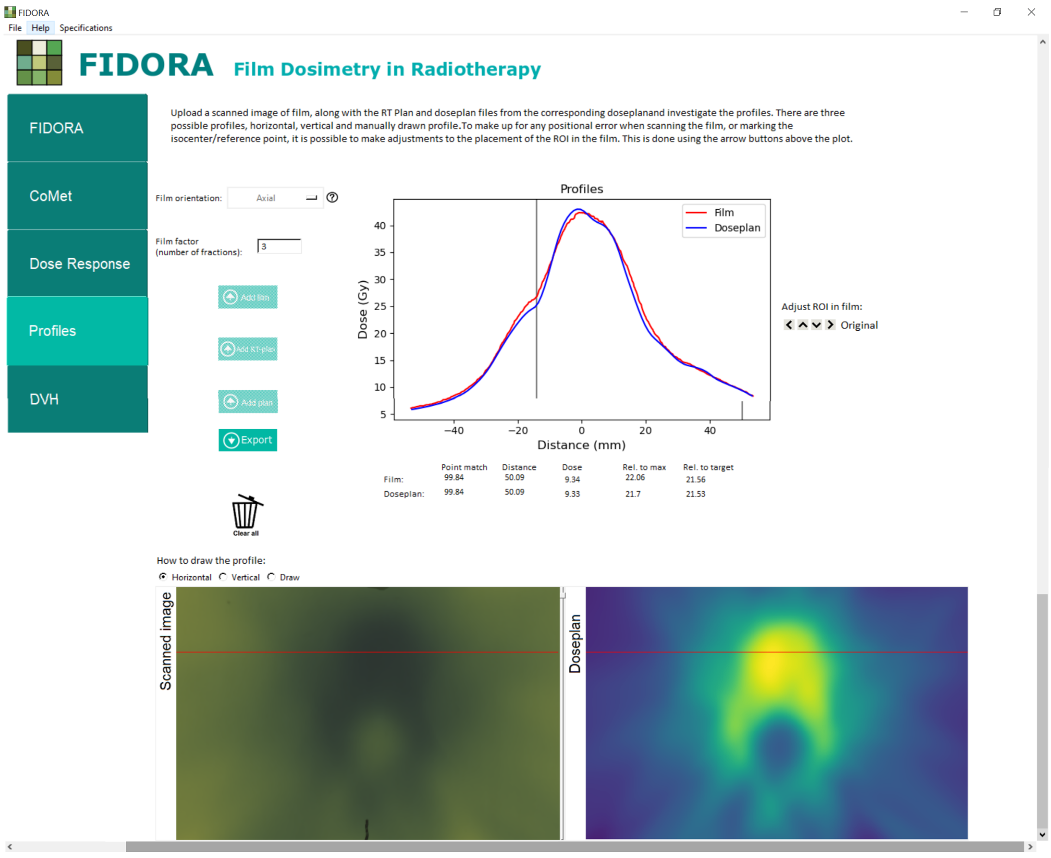
When clicking the button "Mark isocenter" a window showing the image will appear and you are to click on the markers made on the film upon irradiation to find the isocenter. Start with the marker showing the direction of the film (see the specifications in main window). When both marks are made you will see the isocenter in the image. If you are not happy with the placement click the button again and repeat.

When clicking the button "Mark ROI" a window showing the image will appear and you are to drag a rectangle marking the region of interest. Fidora will assume the film has been scanned in either portrait or landscape orientation. When the ROI has been marked it will appear on the image. If you are not happy with the placement click the button again.

Slice offset, mm (default 0):

Figure 4.17: Screenshot of the daughter window where user marks isocenter/reference point and ROI. This is a pop-up window showing as the film has been uploaded.

After clicking the done button, a new window opens where the user will have to choose an earlier completed calibration. Fidora then uses the fitted dose response curve to map the read pixel values to dose. When closing the calibration window, the button for uploading the RT Plan file is activated and the user will have to upload a RT Plan. Then the button for uploading the dose plan will be activated for the user. When pushing the button to upload the dose plan Fidora asks if the user is going to upload several dose plans (which may be the case in some treatment plans, e.g. some VMAT plans). As all necessary files has been uploaded the user returns to the main window where the profiles are plotted. The user can choose to draw horizontal, vertical or manually drawn profiles. To be able to adjust for small placements error during scanning the user can use the arrow buttons to adjust the ROI in the film to better fit the dose plan. By hovering the computer mouse across the profiles, a table with information about the plot is written to the window. See Figure 4.18 for how the tab looks as both film and treatment plan has been uploaded and profiles has been drawn.



Upload a scanned image of film, along with the RT Plan and doseplan files from the corresponding doseplanand investigate the profiles. There are three possible profiles, horizontal, vertical and manually drawn profile.To make up for any positional error when scanning the film, or marking the isocenter/reference point, it is possible to make adjustments to the placement of the ROI in the film. This is done using the arrow buttons above the plot.

Film orientation: Axial

Film factor (number of fractions): 3

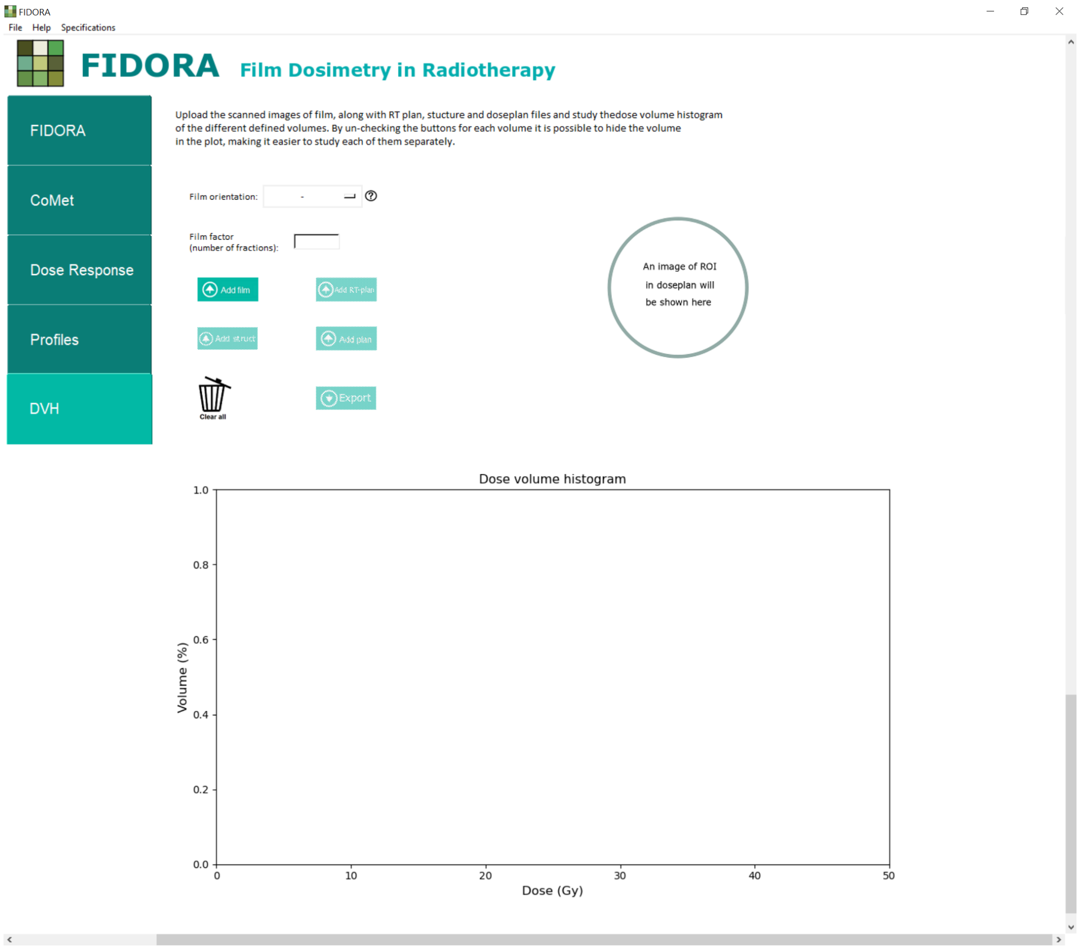
	Point match	Distance	Dose	Rel. to max	Rel. to target
Film:	99.84	50.09	9.34	22.06	21.56
Doseplan:	99.84	50.09	9.33	21.7	21.53

How to draw the profile:
[✓ Horizontal] Vertical Draw

Figure 4.18: Screenshot of the Profiles tab after film and treatment plan has been uploaded and profiles drawn. Here the user can make adjustments to the placement of the film to make up for positioning errors.

4.1.4 Dose volume histogram

In Figure 4.19 is the result of how the fifth tab in Fidora, DVH, turned out. When the user enters this tab the first thing to do is to fill out the film orientation along with how many fractions are planned in the dose plan compared to what was given to the film. Then the user has to upload the film, which will open a new window where the user has to choose between marking the uploaded film with isocenter or a reference point (see Section 3.1.1) and define a region of interest. This is necessary for the positioning of the film onto the dose plan. Figure 4.17 shows an example of how this window can look when marking the isocenter. There the purple lines indicate the marked lines leading to isocenter and the red dot indicates isocenter. The blue rectangle is the marked ROI. As the processing of the film is done the user has to choose an earlier saved calibration to convert the read pixel values in the film to dose. As this is done the button to upload the RT Plan will be activated, and as the user presses this button the RT plan will be uploaded and read, which in turn will activate the button to upload the structure file.



Upload the scanned images of film, along with RT plan, stucture and doseplan files and study thedose volume histogram of the different defined volumes. By un-checking the buttons for each volume it is possible to hide the volume in the plot, making it easier to study each of them separately.

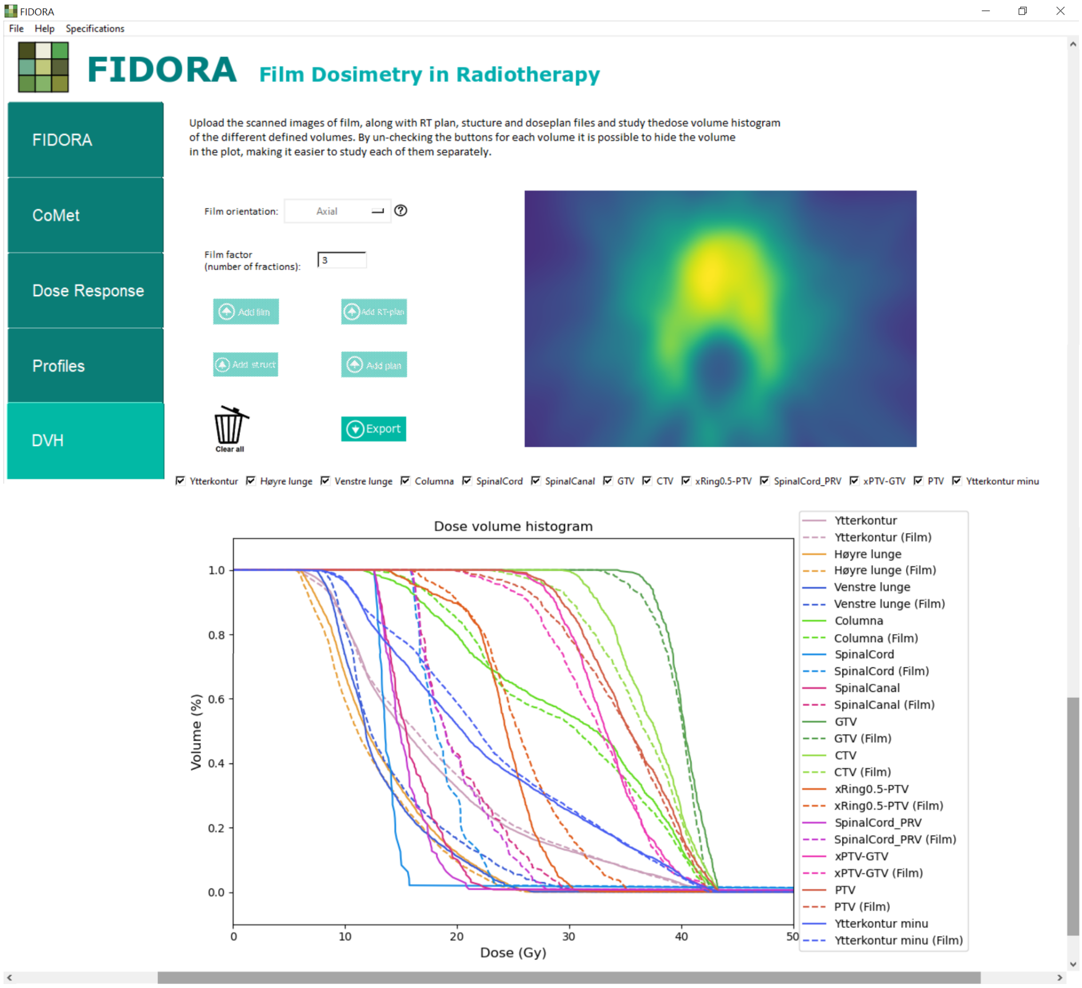
Film orientation: -

Film factor (number of fractions):

An image of ROI in doseplan will be shown here

Figure 4.19: Screenshot of the fifth tab in Fidora - DVH. In this module the user can upload scanned images of film to study the dose volume histogram and compare the results in film and dose plan.

The structure file will read every contour given by the treatment planning system and activate the button to upload the dose plan. While pressing the buttons and uploading files for RT Plan, structure and dose plan there will be no events on the screen as each of them is dependent on the former. And only as all three files have been read a complete dataset has been collected. When finished uploading the dose plan the dose volume histogram for each contour in the region of interest will automatically be plotted to the screen. A list of check buttons makes it possible for the user to un-check any volume and remove it from the plot. As the dose volume histograms are plotted a button to export the plot will be activated, giving the user the opportunity to download the plot to the local computer. Figure 4.20 shows how the tab DVH in Fidora looks like after a successful run.



Upload the scanned images of film, along with RT plan, stucture and doseplan files and study thedose volume histogram of the different defined volumes. By un-checking the buttons for each volume it is possible to hide the volume in the plot, making it easier to study each of them separately.

Film orientation: Axial

Film factor (number of fractions): 3

☑ Ytterkontur ☑ Høyre lunge ☑ Venstre lunge ☑ Columna ☑ SpinalCord ☑ SpinalCanal ☑ GTV ☑ CTV ☑ xRing0.5-PTV ☑ SpinalCord_PRV ☑ xPTV-GTV ☑ PTV ☑ Ytterkontur minu

Figure 4.20: Screenshot of the fifth tab in Fidora - DVH to illustrate how it looks after a successful run.

4.2 MLC model in RayStation

In stereotactic radiotherapy the accuracy of the delivery is crucial, and it is important that the models in the treatment planning system is reliable in regard to the actual delivered dose. One of the uncertainties, especially related to small fields, is the modelling of the MLC in the linac as this will be defining for the penumbra. Here the MLC model in RayStation is studied by looking at measured dose, penumbra and field size for 5 different field sizes. In addition, profiles of a field partially blocked by the MLC leaves has been looked at to see any differences in blocking with the side or the tip of the leaf. In Figure 4.21 the x-profiles belonging to both scanned film and dose plan are plotted. In Tables 4.4, 4.5 and 4.6 the centre dose measured as an average over 5mm is found as the film has been filtered using a 5 pixel median filter, no filter and 15 pixel median filter respectively.

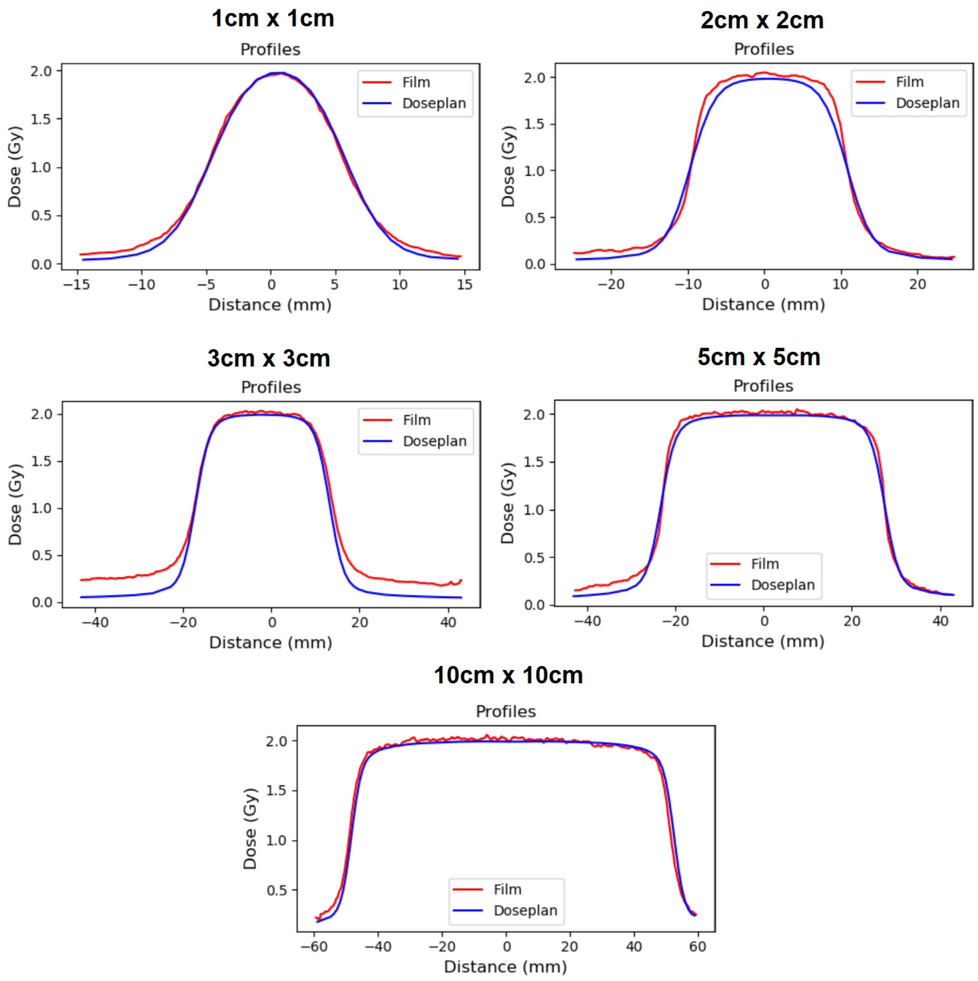


Figure 4.21: X-profiles from 1cm$x$1cm, 2cm$x$2cm, 3cm$x$3cm, 5cm$x$5cm and 10cm$x$10cm fields. The film is represented with the red curve, while the dose plan is represented with a blue curve.

Field size (cm$x$cm)	Film (Gy)	Dose plan (Gy)	Δ (Gy)
1cm$x$1cm	1.97	1.99	-0.02
2cm$x$2cm	2.02	1.98	0.04
3cm$x$3cm	2.03	1.99	0.04
5cm$x$5cm	1.99	1.99	0.00
10cm$x$10cm	2.03	1.99	0.04

Table 4.4: Dose measured in the centre of the x-profile as the film has been filtered using a 5-pixel median filter. The centre is measured as an average over 5mm around centre point.

Field size (cm$x$cm)	Film (Gy)	Dose plan (Gy)	Δ (Gy)
1cm$x$1cm	1.98	1.99	-0.01
2cm$x$2cm	2.03	1.98	0.05
3cm$x$3cm	2.00	1.99	0.01
5cm$x$5cm	2.00	1.99	0.01
10cm$x$10cm	2.01	1.99	0.02

Table 4.5: Dose measured in the centre of the x-profile as the film has not been filtered. The centre is measured as an average over 5mm around centre point.

Field size (cm$x$cm)	Film (Gy)	Dose plan (Gy)	Δ (Gy)
1cm$x$1cm	1.97	1.99	-0.02
2cm$x$2cm	2.03	1.98	0.05
3cm$x$3cm	2.02	1.99	0.03
5cm$x$5cm	1.99	1.99	0.00
10cm$x$10cm	2.01	1.99	0.02

Table 4.6: Dose measured in the centre of the x-profile as the film has been filtered using a 15-pixel median filter. The centre is measured as an average over 5mm around centre point.

Tables 4.7, 4.8 and 4.9 shows the left and right penumbra on the x-profiles in both film and dose plan as well as the difference (film - dose plan) when the film has been filtered using a 5 pixel median filter, no filter and a 15 pixel filter respectively. The penumbra is defined as the distance between 20% to 80% max dose.

	Film (mm)		**Dose plan (mm)**		**Δ (mm)**	
Field size	Left	Right	Left	Right	Left	Right
1cm$x$1cm	4.38	4.51	4.03	3.68	0.35	0.83
2cm$x$2cm	3.80	3.80	4.69	5.02	-0.89	-1.22
3cm$x$3cm	7.08	6.55	4.36	5.09	2.72	1.46
5cm$x$5cm	4.78	4.37	4.61	4.58	0.17	-0.21
10cm$x$10cm	6.58	6.58	5.38	5.39	1.20	1.19

Table 4.7: Left and right penumbra measured in x-profile as the film has been filtered using a 5-pixel median filter. The penumbra is measured between 20% and 80% of max dose.

	Film (mm)		**Dose plan (mm)**		**Δ (mm)**	
Field size	Left	Right	Left	Right	Left	Right
1cm$x$1cm	4.69	4.70	4.00	3.39	0.69	1.31
2cm$x$2cm	3.82	3.69	4.35	4.68	-0.53	-0.99
3cm$x$3cm	7.14	6.35	4.16	5.60	2.98	0.75
5cm$x$5cm	4.99	5.21	4.79	4.59	0.20	0.62
10cm$x$10cm	6.40	6.71	5.79	5.49	0.61	1.22

Table 4.8: Left and right penumbra measured in x-profile as the film has not been filtered. The penumbra is measured between 20% and 80% of max dose.

	Film (mm)		**Dose plan (mm)**		**Δ (mm)**	
Field size	Left	Right	Left	Right	Left	Right
1cm$x$1cm	4.50	4.39	4.28	3.38	0.22	1.01
2cm$x$2cm	3.79	3.79	4.15	4.38	-0.36	-0.59
3cm$x$3cm	7.33	6.93	5.15	4.95	2.18	1.98
5cm$x$5cm	4.87	5.07	4.84	4.84	0.03	0.23
10cm$x$10cm	6.72	6.72	5.81	5.50	0.91	1.22

Table 4.9: Left and right penumbra measured in x-profile as the film has been filtered using a 15-pixel median filter. The penumbra is measured between 20% and 80% of max dose.

Table 4.10 holds the measurements of the field sizes defined by full width half maximum in the x-profiles. The filed sizes are measured as the film has been subjected to a 5-pixel median filter and the dept of the film upon irradiation was 10 cm in a solid water phantom.

Field size	Film (mm)	Dose plan (mm)
1cm$x$1cm	10.68	10.42
2cm$x$2cm	20.46	20.65
3cm$x$3cm	30.26	30.48
5cm$x$5cm	50.33	50.45
10cm$x$10cm	100.76	100.85

Table 4.10: Measurements of the field size in both film and dose plan. The field size is defined by the limits of 50% of centre dose in x-profile. Here the film has been filtered using a 5-pixel median filter.

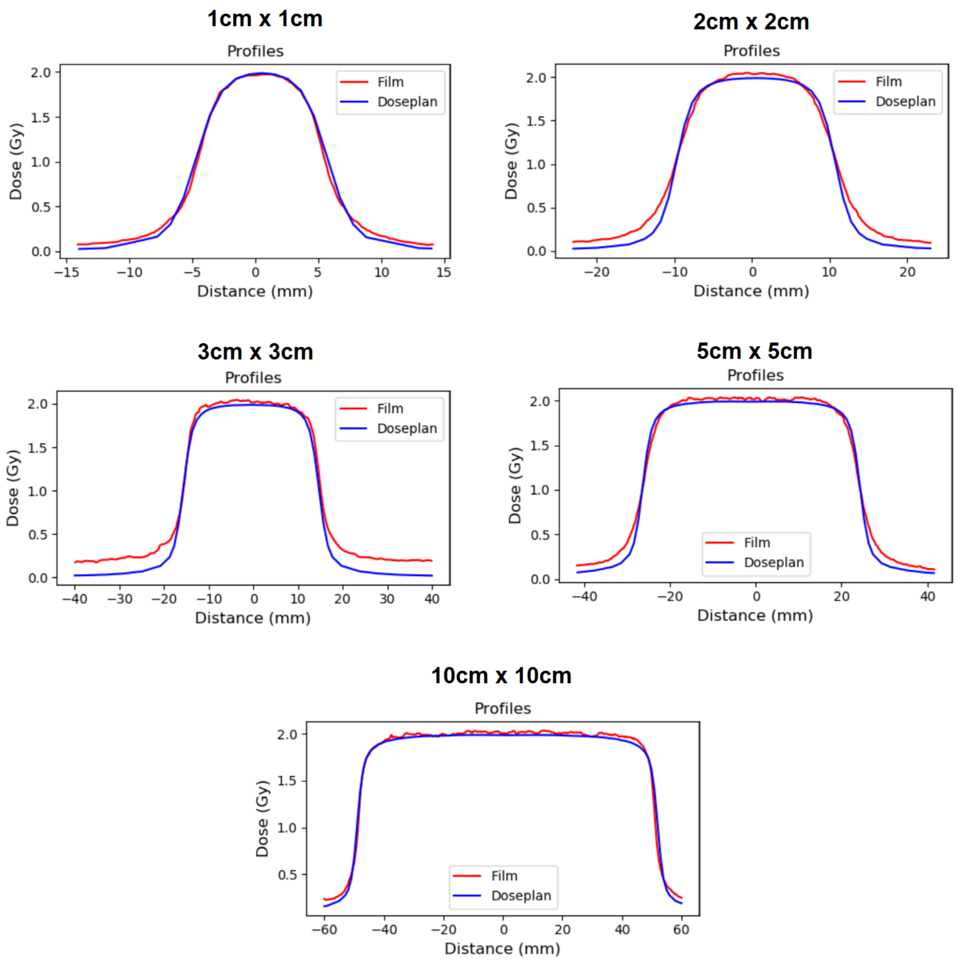


Figure 4.22: Y-profiles from 1cm$x$1cm, 2cm$x$2cm, 3cm$x$3cm, 5cm$x$5cm and 10cm$x$10cm fields. The film is represented with the red curve, while the dose plan is represented with a blue curve.

In Figure 4.22 the y-profiles belonging to both scanned film and dose plan are plotted. In Tables 4.11, 4.12 and 4.13 the centre dose measured as an average over 5mm is found as the film has been filtered using a 5-pixel median filter, no filter and 15-pixel median filter respectively.

Field size (cm$x$cm)	Film (Gy)	Dose plan (Gy)	Δ (Gy)
1cm$x$1cm	1.97	1.99	0.02
2cm$x$2cm	2.04	1.98	0.06
3cm$x$3cm	2.03	1.99	0.04
5cm$x$5cm	1.98	1.99	-0.01
10cm$x$10cm	2.00	1.99	0.01

Table 4.11: Dose measured in the centre of the y-profile as the film has been filtered using a 5-pixel median filter. The centre is measured as an average over 5mm around centre point.

Field size (cm$x$cm)	Film (Gy)	Dose plan (Gy)	Δ (Gy)
1cm$x$1cm	2.00	1.99	0.01
2cm$x$2cm	2.03	1.98	0.05
3cm$x$3cm	2.03	1.99	0.04
5cm$x$5cm	2.00	1.99	0.01
10cm$x$10cm	1.98	1.99	-0.01

Table 4.12: Dose measured in the centre of the y-profile as the film has not been filtered. The centre is measured as an average over 5mm around centre point.

Field size (cm$x$cm)	Film (Gy)	Dose plan (Gy)	Δ (Gy)
1cm$x$1cm	1.97	1.99	-0.02
2cm$x$2cm	2.02	1.98	0.04
3cm$x$3cm	2.02	1.99	0.03
5cm$x$5cm	1.98	1.99	-0.01
10cm$x$10cm	2.01	1.99	0.02

Table 4.13: Dose measured in the centre of the y-profile as the film has been filtered using a 15-pixel median filter. The centre is measured as an average over 5mm around centre point.

Tables 4.14, 4.15 and 4.16 shows the left and right penumbra on the y-profiles in both film and dose plan as well as the difference (film - dose plan) when the film has been filtered using a 5-pixel median filter, no filter and a 15-pixel filter respectively. The penumbra is defined as the distance between 20% to 80% max dose.

	Film (mm)		**Dose plan (mm)**		**Δ (mm)**	
Field size	Left	Right	Left	Right	Left	Right
1cm$x$1cm	3.05	2.87	2.90	3.03	0.15	-0.16
2cm$x$2cm	5.31	5.06	2.99	3.09	2.32	1.97
3cm$x$3cm	5.22	4.74	3.40	2.94	1.82	1.80
5cm$x$5cm	6.12	5.53	3.95	3.36	2.17	2.17
10cm$x$10cm	4.77	4.77	4.77	3.87	0.00	0.90

Table 4.14: Left and right penumbra measured in y-profile as the film has been filtered using a 5-pixel median filter. The penumbra is measured between 20% and 80% of max dose.

	Film (mm)		**Dose plan (mm)**		**Δ (mm)**	
Field size	Left	Right	Left	Right	Left	Right
1cm$x$1cm	3.04	2.71	3.13	3.21	-0.09	-0.50
2cm$x$2cm	5.55	5.16	3.10	3.23	2.45	1.93
3cm$x$3cm	4.70	5.04	3.23	3.47	1.47	1.57
5cm$x$5cm	4.18	4.38	3.39	3.38	0.79	1.00
10cm$x$10cm	5.50	5.19	3.67	3.67	1.83	1.52

Table 4.15: Left and right penumbra measured in y-profile as the film has not been filtered. The penumbra is measured between 20% and 80% of max dose.

	Film (mm)		**Dose plan (mm)**		**Δ (mm)**	
Field size	Left	Right	Left	Right	Left	Right
1cm$x$1cm	2.96	3.26	3.26	3.06	-0.30	0.20
2cm$x$2cm	5.21	5.09	3.27	3.15	1.94	1.94
3cm$x$3cm	5.01	5.01	3.34	3.33	1.67	1.68
5cm$x$5cm	6.16	5.77	3.98	3.18	2.18	2.59
10cm$x$10cm	4.95	4.95	4.02	4.02	0.93	0.93

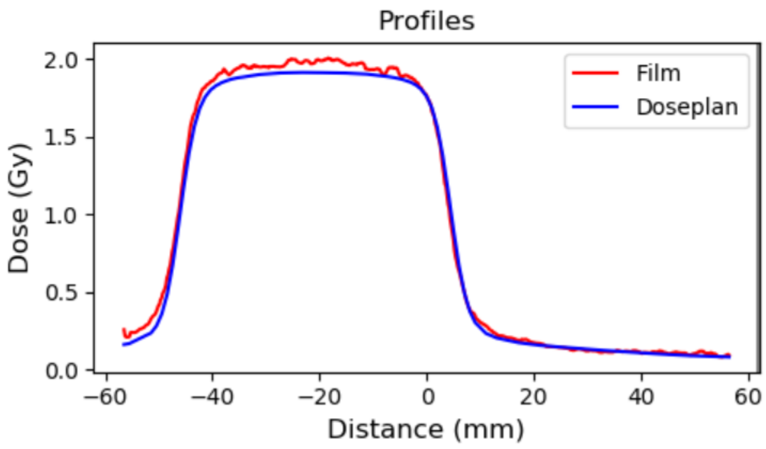
Table 4.16: Left and right penumbra measured in y-profile as the film has been filtered using a 15-pixel median filter. The penumbra is measured between 20% and 80% of max dose.

Table 4.17 holds the measurements of the field sizes defined by full width half maximum in the y-profiles. The filed sizes are measured as the film has been subjected to a 5-pixel median filter and the dept of the film upon irradiation was 10 cm in a solid water phantom.

Field size	Film (mm)	Dose plan (mm)
1cm$x$1cm	10.21	10.84
2cm$x$2cm	20.63	20.60
3cm$x$3cm	30.77	30.98
5cm$x$5cm	50.4	50.32
10cm$x$10cm	100.41	101.46

Table 4.17: Measurements of the field size in both film and dose plan. The field size is defined by the limits of 50% of centre dose in y-profile. Here the film has been filtered using a 5-pixel median filter.

Figure 4.23 and 4.24 shows the plotted profiles of a 10cm$x$10cm field with one quadrant blocked out by the MLC side and tip respectively. Table 4.18 and 4.19 shows the measurements done over the MLC leaves side and tip. The penumbra is defined as the distance between 20% to 80% max dose.



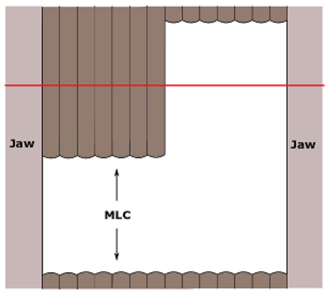
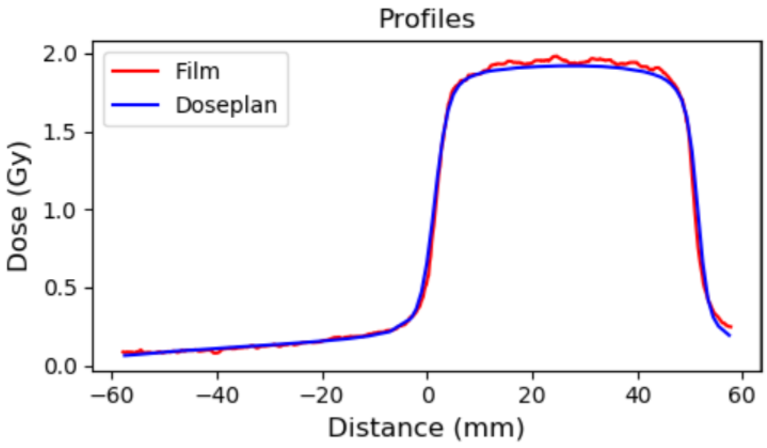


Figure 4.23: Horizontal profile of a 10cm$x$10cm field with one quadrant blocked out by the MLC leaves. The penumbra is due to blocking from the MLC side. The drawing at the right illustrates where the profile was taken.



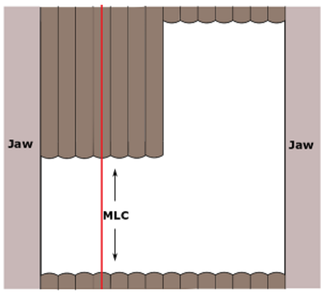


Figure 4.24: Vertical profile of a 10cm$x$10cm field with one quadrant blocked out by the MLC leaves. The penumbra is due to blocking from the MLC tip. The drawing at the right illustrates where the profile was taken.

Type of measurment	Film (mm)	Dose plan (mm)
Penumbra at blocking	6.28	5.68
Length of peak	50.26	50.86

Table 4.18: Measurements of the penumbra (defined between 20% and 80% of max dose) and the length of the peak. The profile is drawn across the side of the leaf in the MLC.

Type of measurment	Film (mm)	Dose plan (mm)
Penumbra at blocking	4.98	5.35
Length of peak	49.54	50.24

Table 4.19: Measurements of the penumbra (defined between 20% and 80% of max dose) and the length of the peak. The profile is drawn across the tip of the leaf in the MLC

4.3 Stereotactic treatment plans

Four stereotactic treatment plans were created to use as a proof of concept for Fidora. The four treatment plans, called V1, V2, V3 and V4, are all stereotactic treatment plans aimed at delivering a high and concise dose to GTV, CTV and PTV while at the same time minimize the dose delivered to organs at risk e.g. lungs and spinal cord (see Figure 4.25 for volumes). In this case the spinal cord will be of special interest as GTV, CTV and PTV are very close and the spinal cord generally has a very low window of tolerance. The results for each treatment plan will be stated in turn and compared in the discussion chapter. The results of V5-V8 will be presented along with its corresponding treatment plan.

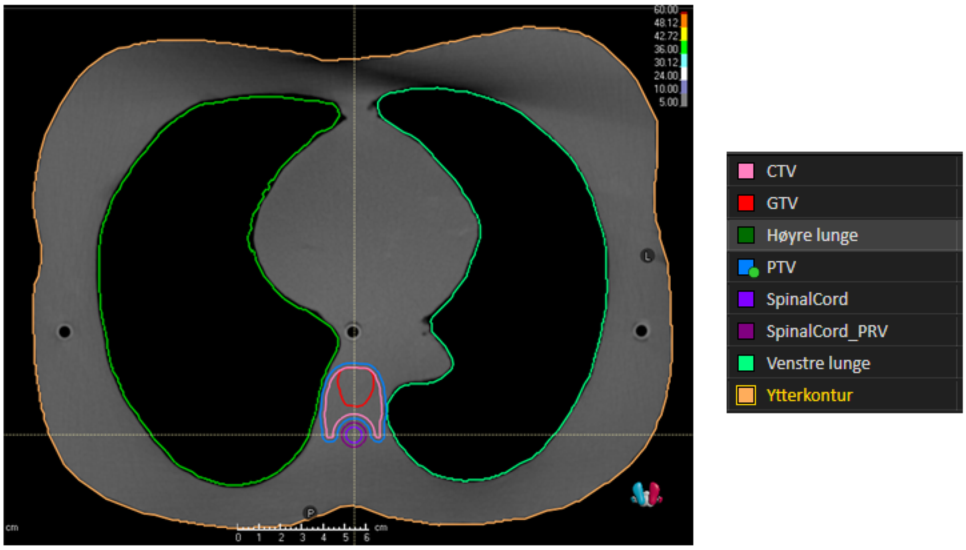


Figure 4.25: Phantom showing the different volumes of interest.

4.3.1 Treatment plan V1

Figure 4.26 shows two profiles (a and c) going through the spinal cord and one profile (b) going through the area of highest dose at the GTV, CTV and PTV. In profile (b) lines are drawn to indicate the placement of the spinal cord and the parameter used during optimization of the treatment plan saying a maximum dose of 18Gy to this area. In Figure 4.27 profiles (1) and (2) are plotted, corresponding to the profiles (a) and (c) respectively in Figure 4.26. The two profiles are meant for comparison of the high dose area in the GTV, CTV and PTV.

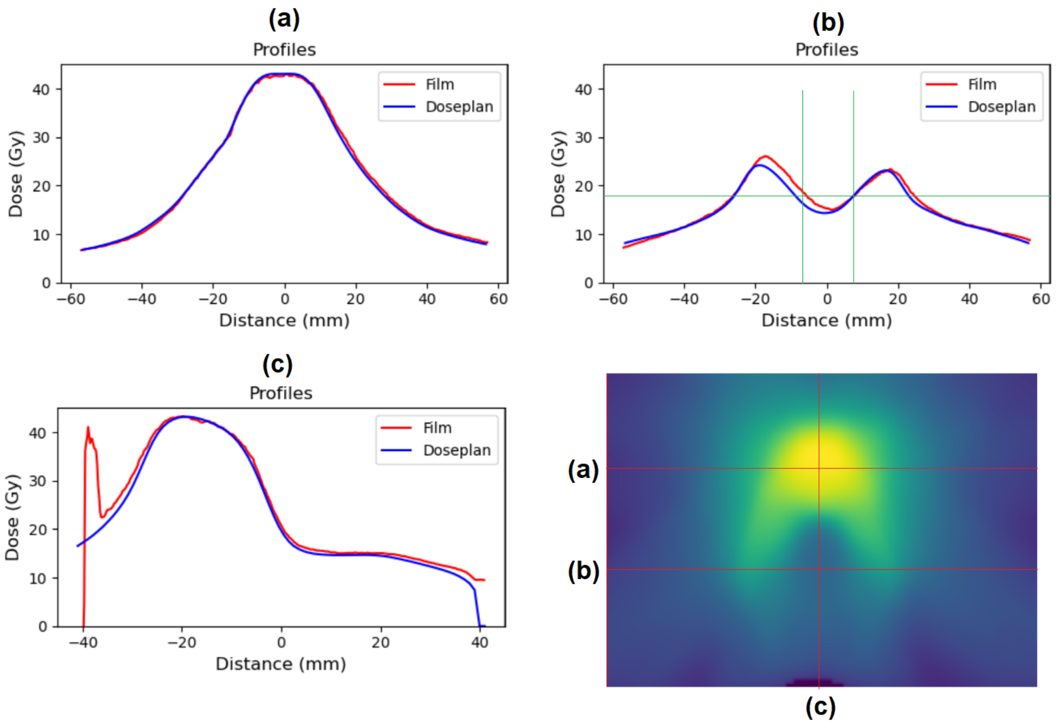


Figure 4.26: Profiles from treatment plan V1. Their number correspond to the numbered lines in the last image. The green, vertical lines in profile (b) illustrates the placement of the spinal cord, while the horizontal, green line indicates dose 18Gy which comes from one of the parameters in the optimization of the treatment plan.

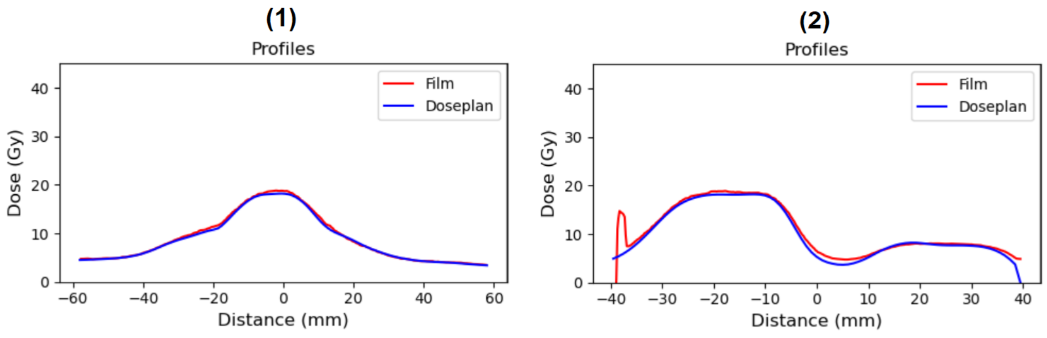


Figure 4.27: Profiles from treatment plan V5 for comparison at the high dose areas with treatment plan V1 shown in Figure 4.26. Plots (1) and (2) corresponds to plots (a) and (c) in Figure 4.26 respectively.

Figure 4.28 shows the dose volume histograms for treatment plan V1. As the film is only 2D, the volume will be a flat volume with a height of 1mm and area corresponding to the chosen region of interest shown in Figure 4.26. In Figure 4.29 only the curves belonging to the spinal cord and spinal cord PRV is shown. From the dose volume histogram found for the spinal cord and spinal cord PRV the maximum dose was found, and are given in Table 4.20 together with minimum dose in spinal cord measured in profile (b) in Figure 4.26.

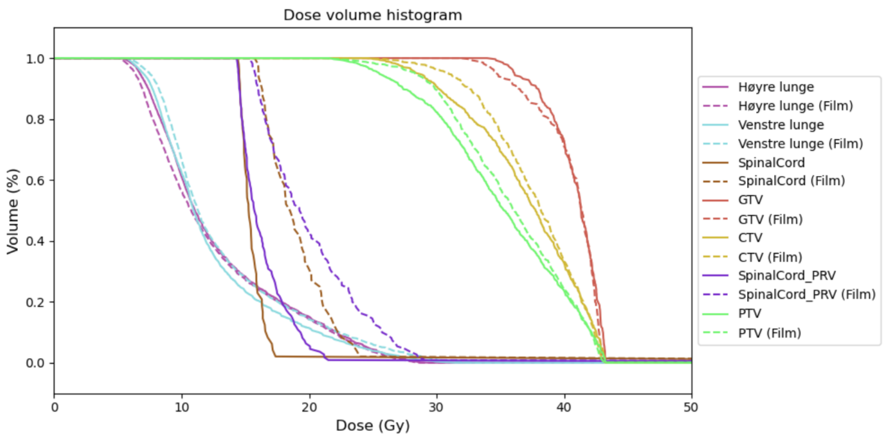


Figure 4.28: Dose volume histogram of treatment plan V1 for both film and dose plan. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.26.

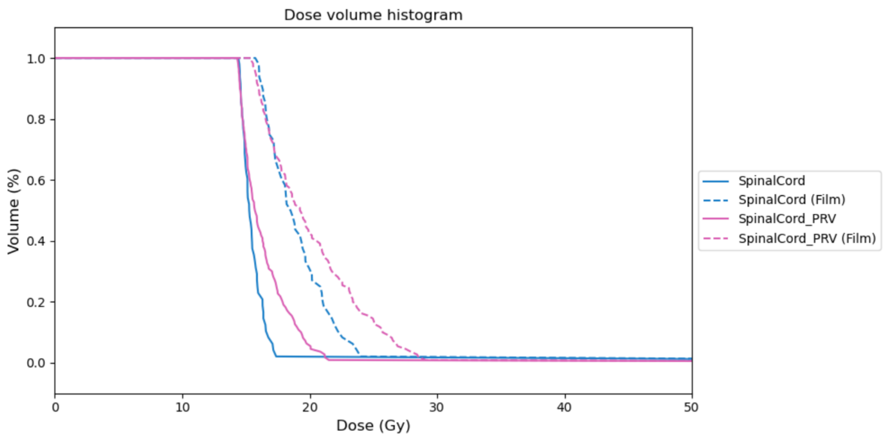


Figure 4.29: Dose volume histogram of treatment plan V1 for both film and dose plan showing only the curve for spinal cord and spinal cord PRV. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.26.

Measurement	Film (Gy)	Dose plan (Gy)	Δ (Gy)
Minimum dose in spinal cord found in profile	15.8	14.3	1.5
Maximum dose measured in DVH in the spinal cord	24.0	17.4	6.6
Maximum dose measured in DVH in the spinal cord PRV	28.8	21.3	7.5

Table 4.20: Measurements from profiles and dose volume histogram for treatment plan V1

4.3.2 Treatment plan V2

Figure 4.30 shows two profiles (a and c) going through the spinal cord and one profile (b) going through the area of highest dose at the GTV, CTV and PTV. In profile (b) lines are drawn to indicate the placement of the spinal cord and the parameter used during optimization of the treatment plan saying a maximum dose of 18Gy to this area. In Figure 4.31 profiles (1) and (2) are plotted, corresponding to the profiles (a) and (c) respectively in Figure 4.30. The two profiles are meant for comparison of the high dose area in the GTV, CTV and PTV.

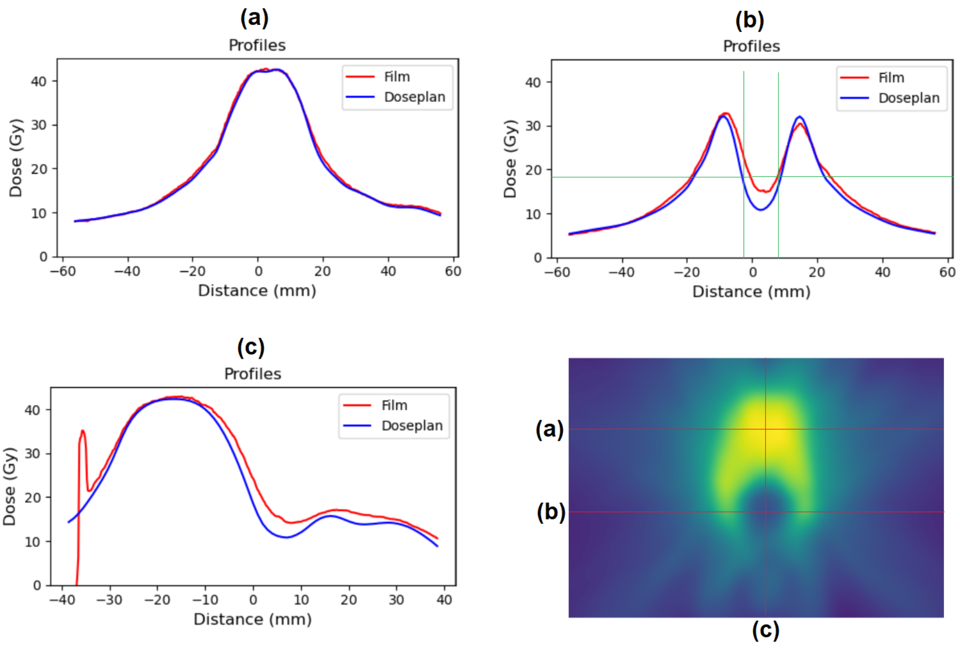
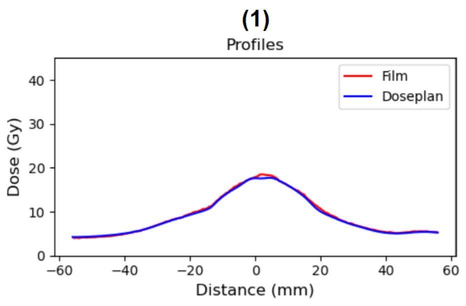


Figure 4.30: Profiles from treatment plan V2. Their number correspond to the numbered lines in the last image. The green, vertical lines in profile (b) illustrates the placement of the spinal cord, while the horizontal, green line indicates dose 18Gy which comes from one of the parameters in the optimization of the treatment plan.



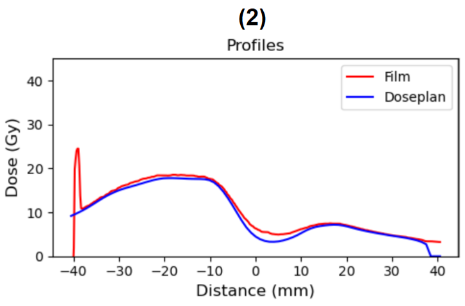


Figure 4.31: Profiles from treatment plan V6 for comparison at the high dose areas with treatment plan V2 shown in Figure 4.30. Plots (1) and (2) corresponds to plots (a) and (c) in Figure 4.30 respectively.

Figure4.32 shows the dose volume histograms for treatment plan V2. As the film is only 2D, the volume will be a flat volume with a height of 1mm and area corresponding to the chosen region of interest shown in Figure 4.30. In Figure 4.33 only the curves belonging to the spinal cord and spinal cord PRV is shown. From the dose volume histogram found for the spinal cord and spinal cord PRV the maximum dose was found, and are given in Table 4.21 together with minimum dose in spinal cord measured in profile (b) in Figure 4.30.

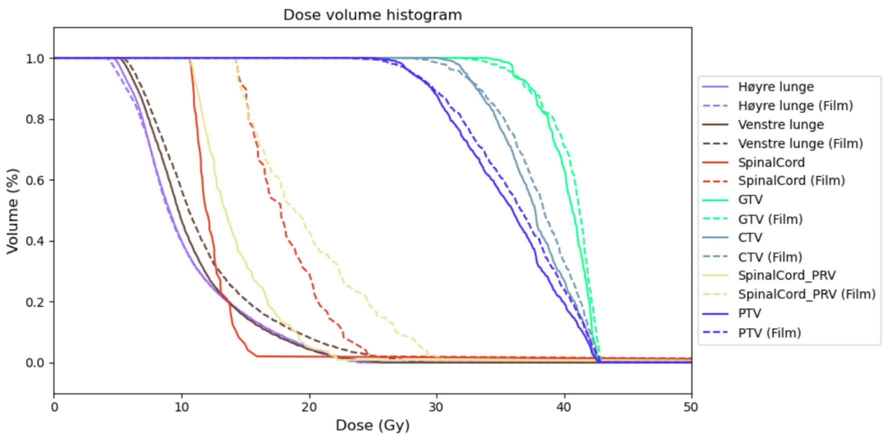


Figure 4.32: Dose volume histogram of treatment plan V2 for both film and dose plan. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.30.

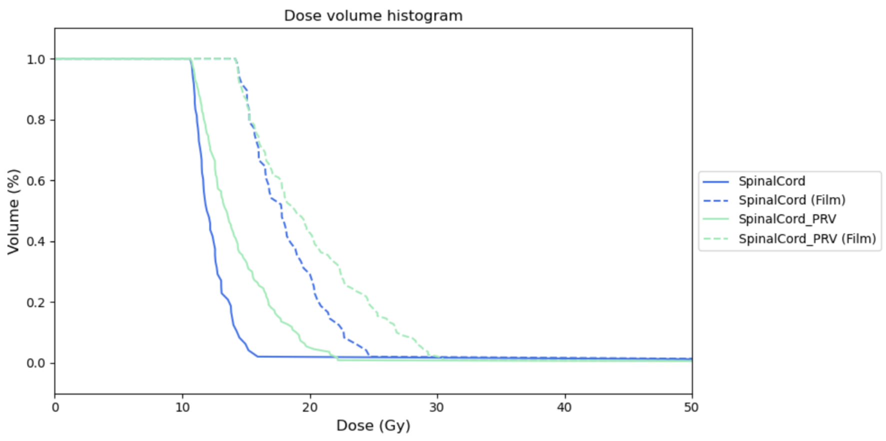


Figure 4.33: Dose volume histogram of treatment plan V2 for both film and dose plan showing only the curve for spinal cord and spinal cord PRV. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.30.

Measurement	Film (Gy)	Dose plan (Gy)	Δ (Gy)
Minimum dose in spinal cord found in profile	17.3	10.9	6.4
Maximum dose measured in DVH in the spinal cord	24.7	15.9	8.8
Maximum dose measured in DVH in the spinal cord PRV	30.2	21.9	8.3

Table 4.21: Measurements from profiles and dose volume histogram for treatment plan V2

4.3.3 Treatment plan V3

Figure 4.34 shows two profiles (a and c) going through the spinal cord and one profile (b) going through the area of highest dose at the GTV, CTV and PTV. In profile (b) lines are drawn to indicate the placement of the spinal cord and the parameter used during optimization of the treatment plan saying a maximum dose of 18Gy to this area. In Figure 4.35 profiles (1) and (2) are plotted, corresponding to the profiles (a) and (c) respectively in Figure 4.34. The two profiles are meant for comparison of the high dose area in the GTV, CTV and PTV.

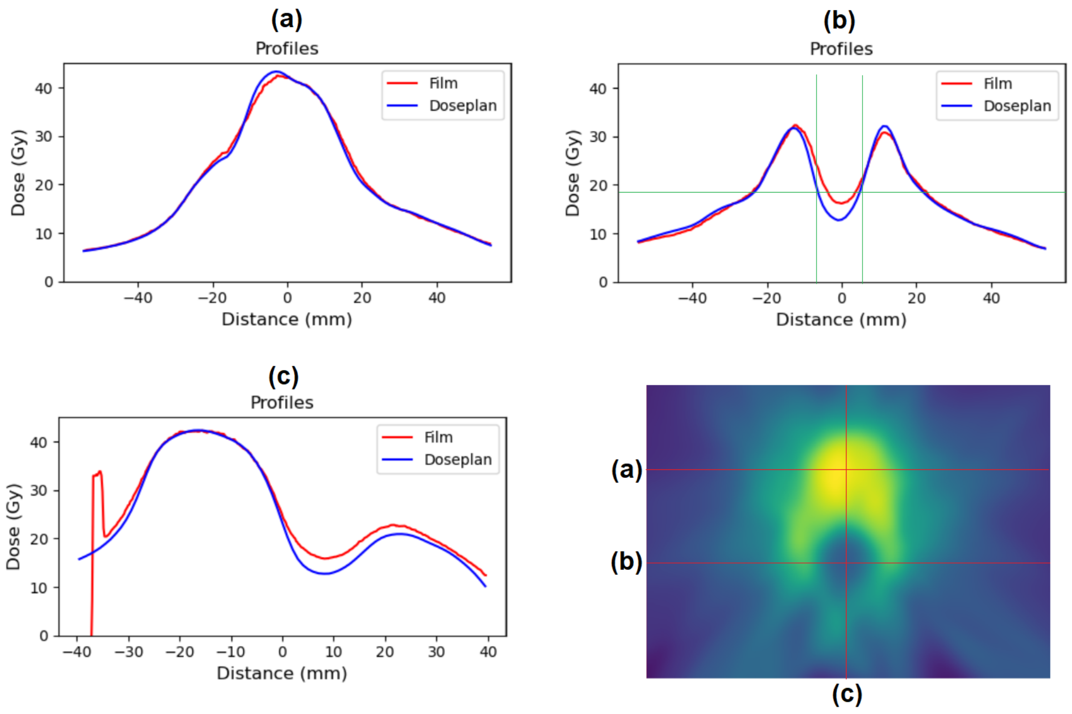


Figure 4.34: Profiles from treatment plan V3. Their number correspond to the numbered lines in the last image. The green, vertical lines in profile (b) illustrates the placement of the spinal cord, while the horizontal, green line indicates dose 18Gy which comes from one of the parameters in the optimization of the treatment plan.

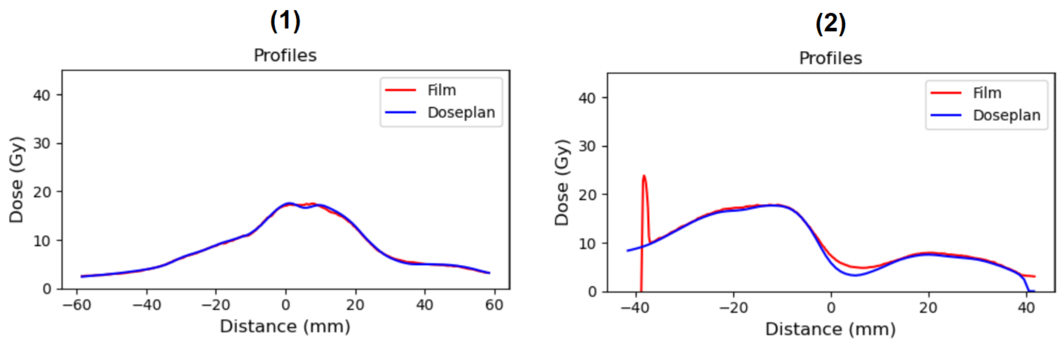


Figure 4.35: Profiles from treatment plan V7 for comparison at the high dose areas with treatment plan V3 shown in Figure 4.34. Plots (1) and (2) corresponds to plots (a) and (c) in Figure 4.34 respectively.

Figure4.36 shows the dose volume histograms for treatment plan V3. As the film is only 2D, the volume will be a flat volume with a height of 1mm and area corresponding to the chosen region of interest shown in Figure 4.34. In Figure 4.37 only the curves belonging to the spinal cord and spinal cord PRV is shown. From the dose volume histogram found for the spinal cord and spinal cord PRV the maximum dose was found, and are given in Table 4.22 together with minimum dose in spinal cord measured in profile (b) in Figure 4.34.

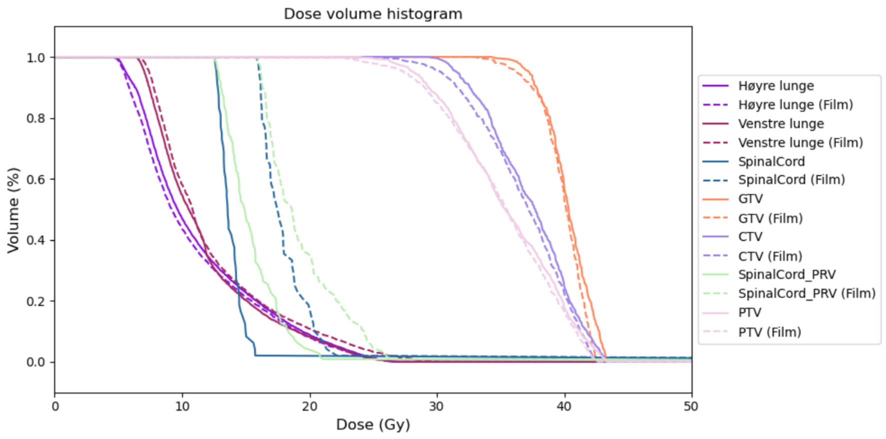


Figure 4.36: Dose volume histogram of treatment plan V3 for both film and dose plan. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.34.

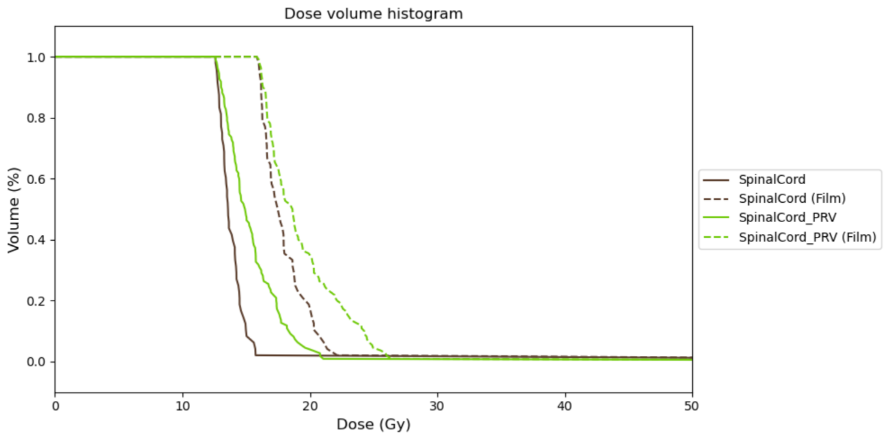


Figure 4.37: Dose volume histogram of treatment plan V3 for both film and dose plan showing only the curve for spinal cord and spinal cord PRV. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.34.

Measurement	Film (Gy)	Dose plan (Gy)	Δ (Gy)
Minimum dose in spinal cord found in profile	17.7	13.4	4.3
Maximum dose measured in DVH in the spinal cord	22.1	15.7	6.4
Maximum dose measured in DVH in the spinal cord PRV	26.0	20.8	5.2

Table 4.22: Measurements from profiles and dose volume histogram for treatment plan V3

4.3.4 Treatment plan V4

Figure 4.38 shows two profiles (a and c) going through the spinal cord and one profile (b) going through the area of highest dose at the GTV, CTV and PTV. In profile (b) lines are drawn to indicate the placement of the spinal cord and the parameter used during optimization of the treatment plan saying a maximum dose of 18Gy to this area. In Figure 4.39 profiles (1) and (2) are plotted, corresponding to the profiles (a) and (c) respectively in Figure 4.38. The two profiles are meant for comparison of the high dose area in the GTV, CTV and PTV.

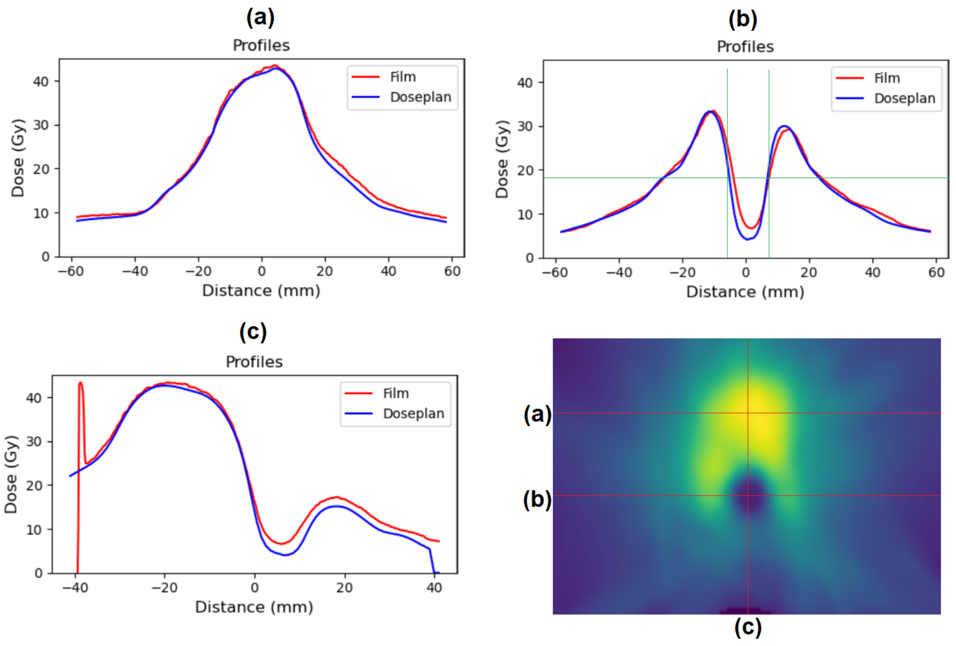


Figure 4.38: Profiles from treatment plan V4. Their number correspond to the numbered lines in the last image. The green, vertical lines in profile (b) illustrates the placement of the spinal cord, while the horizontal, green line indicates dose 18Gy which comes from one of the parameters in the optimization of the treatment plan.

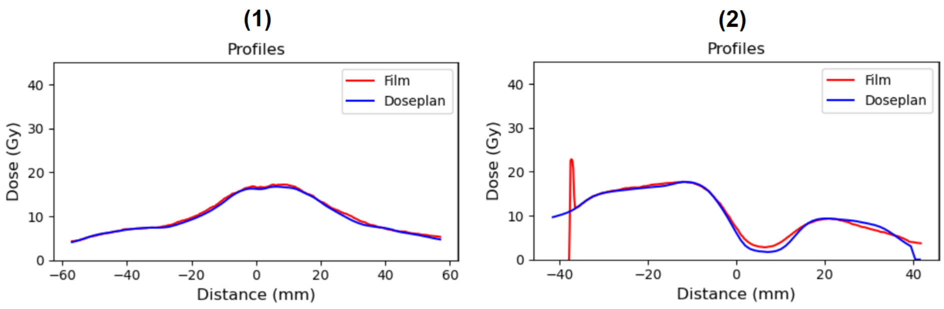


Figure 4.39: Profiles from treatment plan V8 for comparison at the high dose areas with treatment plan V4 shown in Figure 4.38. Plots (1) and (2) corresponds to plots (a) and (c) in Figure 4.38 respectively.

Figure4.40 shows the dose volume histograms for treatment plan V4. As the film is only 2D, the volume will be a flat volume with a height of 1mm and area corresponding to the chosen region of interest shown in Figure 4.38. In Figure 4.41 only the curves belonging to the spinal cord and spinal cord PRV is shown. From the dose volume histogram found for the spinal cord and spinal cord PRV the maximum dose was found, and are given in Table 4.23 together with minimum dose in spinal cord measured in profile (b) in Figure 4.38.

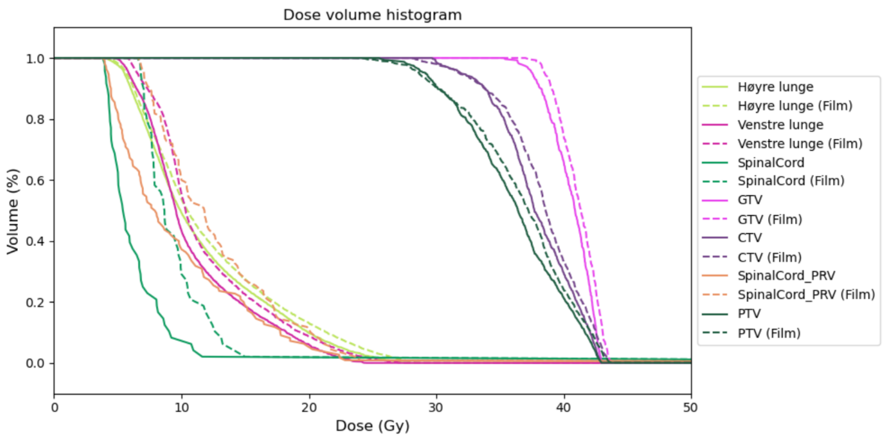


Figure 4.40: Dose volume histogram of treatment plan V4 for both film and dose plan. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.38.

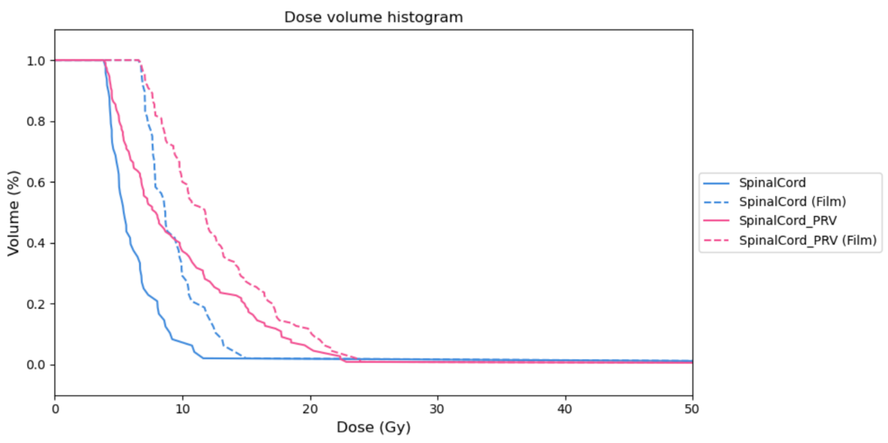


Figure 4.41: Dose volume histogram of treatment plan V4 for both film and dose plan showing only the curve for spinal cord and spinal cord PRV. The volume is a flat volume with height 1mm and area equal to the region shown in Figure 4.38.

Measurement	Film (Gy)	Dose plan (Gy)	Δ (Gy)
Minimum dose in spinal cord found in profile	8.1	4.8	3.3
Maximum dose measured in DVH in the spinal cord	15.0	11.6	3.4
Maximum dose measured in DVH in the spinal cord PRV	23.8	22.4	1.4

Table 4.23: Measurements from profiles and dose volume histogram for treatment plan V4

Chapter 5

Discussion

5.1 Fidora

As mentioned in Section 2.3.3 the relationship between the readout in the scanner and dose is modelled using a 'reciprocal linear vs dose' fitting (see Equation 2.18). Although this is a good model for both the red and green color channel, it has been found during this work that the blue color channel seems to have a dose resolution that is too low for the model to fit. Because of this it was decided that the result for the blue channel would only be shown as measured points and no fit was attempted. This decision is backed up by the fact that when the dose response resolution is as low as it showed for the blue channel it will not give a good mapping from scanner readout to dose, and therefore it could not have been used anyways. Also, since a single channel method has been chosen as the method used in Fidora the red color channel is better for analysis, based on its resolution and wavelength (see Section 2.3.3).

5.1.1 CoMet

In Figures 4.2 and 4.3 the non-uniform readout in both lateral direction and scanning direction in the scanner is established. Even though the pixel value error is relatively small (max readout is pixel value 65535), it is seen in Figure 4.6 that this results in an error around 0.075Gy at the most. Since the non-linear readout is assumed to be dose independent this could possibly generate a large uncertainty for the lower dose levels. In Figure 4.6 it is also seen at that at the same position as the maximum error for the uncorrected image the error in the corrected image is around 0.015Gy, which supports the need for a correction. In Table 4.1 and Figures 4.4 and 4.5 it can be seen that when the correction is averaged over all dose levels the standard deviation becomes much larger than if one used dose level specific corrections (found in Figures 4.2 and 4.3). For positions close to the centre of the scanner surface it seems as if the standard deviation becomes larger than the error due to the non-uniform readout from the scanner. This makes an argument that the correction might be more accurate if it had been created specific for each dose level. However, no systematic relationship between the deviation in intensity and dose was found. Meaning this would result in a much heavier workload as that would require a new correction matrix to be made for each dose used throughout every dose plan.

5.1.2 Dose Response

This tab was in first place created to perform calibration of the film to be able to map the pixel values in the scanned image to dose. In addition, it can be used to study the dose response curve and the dose resolution of the different color channels, red, green and blue. To be able to create a reliable calibration it is important to use known dose levels, which is most stable at the centre field, and scan them in a position with a stable readout. Because of the non-linear readout in the scanner, the most reliable position at the scanner surface is at the centre, and therefore the known dose levels should be scanned there. Therefore, Fidora assumes that all film pieces being scanned for calibration is placed at the centre of the scanner surface. This introduces an uncertainty related to position of the film. If the user misplaced the film piece even by only a few millimetres, Fidora will not read the film piece at its most accurate point. This uncertainty will increase if the irradiation of the film piece used a flattening filter free beam, as this might give a different dose level than expected at the read position. In Section 4.1.2 the dose response curves produced by Fidora was compared to dose response curves produced when performing a manual calibration, both in the case of beams using flattening filter and being flattening filter free. Figures 4.13, 4.14 and 4.15 shows a slightly better resolution in the manually performed calibration and also that the fitting uncertainty of the curves are larger for Fidora ($\pm$0.07Gy and $\pm$0.06Gy) than for the manual calibration (both $\pm$0.02Gy). However, the differences are very small and almost impossible to quantify visually in the plots. Therefore the directed Hausdorff distance, defined by the largest difference between the two datasets, was calculated in each plot (see Table 4.3) and found to be very small in all three cases. The largest difference in dose found in Figure 4.13 was 0.00012Gy, the largest difference in dose in Figure 4.14 was 0.00009Gy and the largest difference found in Figure 4.15 was 0.00006Gy. From visual inspection of plots the first two directed Hausdorff distances is expected to be found somewhere between 2.5Gy and 7.5Gy. This means that the relative error is between 0.0048% and 0.0016% for the case where the flattening filter was used and between 0.0036% and 0.0012% for the case where the beam was flattening filter free. These errors are very small, and it seems that Fidora produces good fittings of the dose response curves and can be used in calibration of the film.

5.1.3 Profiles

In the tab Profiles the user must follow several steps before being able to study the profiles. It is necessary to upload the scanned image of the film, the RT Plan file and the dose plan. In addition, the user must define the orientation of the film while being irradiated and how many fractions are used in the dose plan relative to how many fractions has been used on the film. All of these must be defined or uploaded in a given order as the next step always depends on what was read from the previous step. This makes the workflow a bit cumbersome, and Fidora would benefit from a more general approach. Despite the fact that there are many steps before plotting any profiles, Fidora works well and since it gives the user the opportunity to stay in control of the data and make adjustments to the final ROI it is well suited as an analysing tool.

5.1.4 DVH

As for the tab Profiles the workflow in DVH is a bit cumbersome as there are four separate data files needed to be uploaded before the dose volume histograms are plotted. Fidora would benefit from a more general approach, which could work in the same matter as for the tab Profiles. But once all files have been uploaded the plotting of the dose volume histograms works well and Fidora is well suited to calculate and study the DVH of different structures. A challenge when using film as a dosimeter when obtaining the dose volume histogram is the fact that the film makes a 2D measurement, when dose volume histograms usually are created with 3D measurements. In addition, because of the limitations with the size of the film (203.2mm x 210mm) not all contours defined in the dose plan can fit on one sheet of film. Because of this the volume being calculated in the tab DVH is a flat volume (height 1, 2 or 3mm, depending on the resolution of the dose plan) and with an area not covering the whole slice. This means that the dose volume histogram will not be comparable with other DVHs taken with another positioning of the film or other regions of interest. However, as a comparison between the measured dose on the film and the planned dose plan it works very well, which means that it can still operates as a quality assurance tool.

5.1.5 MLC models in Raystation

In Figures 4.21 and 4.22 the horizontal and vertical profiles, respectively, are plotted for all five different field sizes. Overall, it appears to be a good match between the profiles belonging to the dose plan and the ones belonging to the film. However, there are some notes to be made. From Tables 4.4, 4.5 and 4.6, which shows the centre dose measured at the x-profiles, it can be seen that the film has a trend to measure a higher dose than what was expected, which in this case should be close to 2Gy. The dose plan measures centre doses between 1.98Gy and 1.99Gy while the film measures the centre dose in the range between 1.97Gy and 2.04Gy. That means that the deviation in the dose plan is at most 1% and the deviation in the film is at most 2%, which is both well within the tolerance level that would be accepted by a pass/fail test at 3% (Li et al. (2011)). The largest deviation between film and dose plan was found to be 0.06Gy, which was in the case of the 5-pixel median filter of the y-profile for field size 2cm$x$2cm. In this case the film measured a dose 2.04Gy while the dose plan was at 1.98Gy, meaning that the uncertainty is 3% at the most which is still within the limits of acceptance. To see how sensitive the measurements were to noise related to the radiochromic film, the centre dose was measured without filter, with a 5 pixel (1mm) median filter and with a 15 pixel (3mm) median filter. In the tables mentioned and also in Tables 4.11, 4.12 and 4.13, which gives the same measurements but for the y-profile, it is found that the use of median filter does not influence the centre dose. Tables 4.10 and 4.17 shows the measured field sizes in both film and dose plan defined by full width at half maximum. In both film and dose plan and for all field sizes the measured field size tends to be higher than what was defined for each field. In the dose plan it was expected no deviations in the field size compared to the settings. However, since the dose

plan is represented as a matrix with 1mm distance from one voxel centre to the next, it is very reasonable that the true half of maximum dose was not found at a precise distance. This means that the uncertainty related to the reading of the full width at half maximum inherit an uncertainty of at least 1mm. As such, all measurements of the field size in the dose plan, with a deviation ranging from 0.32mm-0.98mm, is within an acceptable range. As the resolution in the film is better compared to the dose plan, with 0.2mm between pixel centres, a similar but lower uncertainty is expected. In addition, systematic and random errors related to using film as a dosimeter also play into the results (Saur and Frengen (2008)). Overall, it appears as the actual measured field size done in the film is a good match to what was expected.
Tables 4.7 - 4.9 and 4.14 - 4.16 shows the measured penumbras for the case with no filter, 5-pixel median filter and 15-pixel median filter used on the film. In neither the x-profiles nor the y-profiles it seems like the use of a median filter alters the measurements of the penumbra. As such it was decided to use a 5-pixel median filter on all film to remove any "salt and pepper"-artefacts from the scanning, as was suggested by the producer of the EBT3 film (Ashland (2020)). By investigating the mentioned tables, it was found that for field sizes 3cm$x$3cm, 5cm$x$5cm and 10cm$x$10cm the penumbra was slightly wider in the film compared to the dose plan. For field sizes 1cm$x$1cm and 2cm$x$2cm the penumbra was both wider in some cases and narrower in others, suggesting that the modelling of these small fields is not as good as for larger fields. This is in compliance with findings done in another study (Zaghian et al. (2020)). Looking at Figures 4.21 and 4.22 it can be seen that neither of the small fields, 1cm$x$1cm and 2cm$x$2cm, obtains a real plateau region as the edges are too close together and the penumbras overlap. As described in Section 2.4.7 the effect of having penumbras overlap happens as the radiation source in the linac is partially blocked out, which will not be represented in the treatment planning system (Wan (2017)). The overlapping penumbras can be part of the explanation as to why the small fields are not modelled good enough.
From Figures 4.21 and 4.22 it can be seen that the film and dose plan follow each other well for doses up to 2Gy, but in all cases the film measures a higher dose compared to the dose plan at both ends of the profiles. It seems as if the problem is most prominent for doses below 0.5Gy. This is a known problem in radiotherapy, namely that the dose plans are bad at modelling the dose distribution outside of the field. Other studies, Howell et al. (2010) and Wang and Ding (2018), has shown that the modelling of dose out of field can be underestimated as much as 30-40%. In this case the film illustrates this by measuring a higher out-of-field dose than what was given by the dose plan, which is an obvious flaw in the modelling of the dose plan.
Figure 4.23 shows the horizontal profile of a 10cm$x$10cm field with one quadrant blocked out by the MLC leaves. The penumbra on the right side of the profile is due to blocking from the MLC side. Figure 4.24 shows the vertical profile of a 10cm$x$10cm field with one quadrant blocked out by the MLC leaves. Here the left penumbra is due to blocking from the MLC tip. In both profiles there seems to be a good compliance between the film and dose plan. From Table 4.18 it can be seen that the penumbra is a bit wider in the film compared to the dose plan, while Table 4.19 shows that the penumbra in the film is shorter compared to the dose plan. This suggests that the tip of the MLC leaf gives a better cut-off at the field edge compared to the side of the leaf. As described in Section 2.4.6 the

modelling of the MLC in the treatment planning system has an uncertainty related to the modelling of the tip-width of the leaves. Since the radiation field is modelled using a light field, which diverge, and the MLC tip is rounded there will be an offset between the field defined by the MLC side and tip. It is expected that the MLC side will give a wider field having a less steep cut-off compared to the MLC tip. This means that the results given in Tables 4.18 and 4.19 are as expected and shows that the modelling of the MLC in the treatment planning system works well.
All together RayStation seems to model the MLC leaves correct, at least for field sizes larger than 3cm$x$3cm, and there does not seem to be much difference between blocking with the tip or side of the leaf. When using field sizes smaller than 3cm$x$3cm the smearing effect does create a less reliable penumbra, with measured deviations up to 2.98mm. In traditional radiotherapy, where the standardized volumes are created with safety margins that can range up to 10 mm, e.g. for prostate cancer (Van Herk (2004)), the differences in measured and planned penumbra is small and within an acceptable range. In stereotactic radiotherapy the margins are generally smaller and for e.g. the spinal cord it is set to 2mm, which is considered enough to account for the inner motion of the spinal cord (Oztek et al. (2020)). Then variations in the penumbra seen in this work, ranging up to 2.98 mm, is too high. This means that plans being optimized in the treatment planning system to be within the margin may in fact be delivering a higher dose to the critical areas outside the field than what can be accepted. With the new reports from ICRU and IAEA (see Section 2.4.8), there is a hope that the smaller fields will be better modelled in the treatment planning system at a later time.

5.1.6 Stereotactic treatment

The four stereotactic treatment plans, V1, V2, V3 and V4, were all created within a clinical dose range. Since stereotactic radiotherapy operates with high doses and few fractions that means that the delivered dose to the film is high, in this case around 12.5Gy at the most. With the optimum dose range for the film being 0.2-10Gy that means that the treatment plans operate outside of the optimal range. To be able to study the high dose regions, corresponding to the GTV, CTV and PTV in this case, it was necessary to see how well the film represented these high doses. To do this, four additional treatment plans were created, based on V1, V2, V3 and V4. The standard dose in these four additional plans were set to 5Gy meaning that they were inside the optimal range for the film. Limitations regarding dose-rate and MLC made it impossible to create the additional plans by pure scaling. Therefore they are build on the same principles, but optimized separately. However, they still work for comparison in the high dose regions. By comparing profile (a) and (c) to (1) and (2) in Figure 4.26 to 4.27, Figure 4.30 to 4.31, Figure 4.34 to 4.35 and Figure 4.38 to 4.39 it can be seen that the film seems to be representing the dose well even for the high doses. Even though treatment plans V1-V4 has a maximum dose overreaching the optimal range of the film it looks as if it still fits well. Looking at the dose response curves (Figure 4.13-4.15) at doses above 10Gy it agrees with the dose resolution still being high enough to difference small dose changes. As such it was decided that treatment plans V1-V4 are good enough to get a precise study of the treatment plans, and no additional downscaled versions are needed. This is in accordance with another study looking into characterization of the GafChromic film where they found the film being usable up to 40Gy (Borca et al.

(2013)). In that study they also looked at how the dose response of the film is dependent on the energy used in the linac, which means that if repeating the measurements done here with an energy other than 6MV the satisfying resolution of the film in high dose ranges should be re-checked.
In Figures 4.26, 4.30, 4.34 and 4.38 profile (c) has a peak at the left side which is due to the cut out done in the film pieces as described in Section 3.3.2. Where the film is cut it splinters at the edge giving a dark color which will be read as a high dose in Fidora. As a result, this area of the plot should be ignored. This effect can also be seen in i profile (2) in Figures 4.27, 4.31, 4.27 and 4.27. In addition profile (c) in Figures 4.26 and 4.38 shows dose level in the dose plan dropping to 0Gy at the right side, while the film has a much higher level. This is due to the fact that the film piece extended the boundaries of the phantom upon being irradiation, and while the dose delivered to the patient is zero outside of the body the film will measure the radiation there.
As mentioned in Section 3.3.2 the film only received one out of three planned fractions in all of the treatment plans. That means that when processing the film in Fidora the measurements were multiplied with 3. Systematic errors related to doing dosimetry with GafChromic EBT film (Saur and Frengen (2008)) will not be as affected by this as three separate readings would still add up to the same error. But for random errors this upscaling of the measurements could result in a higher deviation than what should be expected, and in addition the usual decrease of the random error through averaging over several measurements does not happen in this case. However, in earlier studies (Saur and Frengen (2008)) these errors have been classified as not significant, and as a result this upscaling of the film is not considered as a problem.
In general looking at all Figures 4.26-4.41 it seem as if the dose plan as a tendency of underestimating the dose compared to the measurements done by the film. This is especially true for the lower doses, which can be seen in both the profiles and the dose volume histograms. In stereotactic radiotherapy the treatment plans are built up of many small fields being added together. As described in Section 2.4.7 a problem with small fields is overlapping penumbras, which in turn will give larger field sizes than intended. Assuming that this is the case for these treatment plans, this could explain why the dose plan shows a lower dose compared to the film. In the treatment planning system RayStation the radiation source is not modelled in a way that covers part of it when using small fields (Wan (2017)), resulting in the overlapping penumbras not being modelled to the dose plan. If all small fields in the treatment plan irradiating the film is in fact larger than intended, that means that when added up the dose will be higher than expected across the whole field. In RayStation the source size is one of the parameters one can adjust during the optimization of the treatment plan. This illustrates some of the complexity and trouble with creating good models to represent what is being delivered in the linac. With many different parameters that can be altered and each of them always being adjusted with the means of fitting the specific treatment case, it is hard to get a general ruling of how to create the best plan, especially with stereotactic radiotherapy which is a relatively new method. This makes it almost impossible to redo experiments with identical models. In addition, one of the fundamental ways of tuning the models is to use measurements to compare results. In stereotactic radiotherapy one of the problems have been the lack of a good dosimeter to do precise readings of the dose, as the most common dosimeters have a too large spatial

extent to get accurate readings of steep gradients. The results in this work suggests that the small fields have not been modelled well enough. The opportunity to adjust parameter such as the source size in RayStation makes it possible to do more investigation into how to better model the small fields. Such studies have not been done in this work but could be useful in further work.
Profile (b) in Figures 4.26, 4.30, 4.34 and 4.38 shows the horizontal profile crossing the spinal cord. In those profiles it can be seen that the left peak tends to be measured to a higher dose by the film than intended, while the right peak tends to be measured to a lower dose by the film. In addition, the minimum dose at the centre of the spinal cord seems to be higher than planned. Looking at the results in Section 4.2 it was found that the modelling of the MLC for small fields sizes was not that great, and in fact profiles with field sizes under 3cm$x$3cm were affected by the fact that they never plateaued. The peaks given in profile (b) in Figures 4.26, 4.30, 4.34 and 4.38 is created by small fields and designed with steep gradients, meaning that this area will be highly affected by the effect seen in Section 4.2. This, together with the effects discussed above with higher doses given by overlapping penumbras can partly explain the unexpected profiles across the spinal cord. However, the whole reason for such profiles are not known, and should be studied further in future work. The deviating behaviour of profiles (b) can be seen again in the dose volume histograms in Figures 4.29, 4.33, 4.37 and 4.41 where it becomes obvious that the film measures a higher overall dose than what was expected.
Looking again at profile (b) in Figures 4.26, 4.30, 4.34 and 4.38 where the placement of the spinal cord is illustrated by the green lines it is possible to study the delivered dose to this slice of the spinal cord, and in particular the minimum and maximum dose. The minimum measured dose in the spinal cord and the maximum measured dose in both spinal cord and spinal cord PRV is given in Tables 4.20, 4.21, 4.22 and 4.23. With the criteria of 18Gy as maximum dose to the spinal cord and 21Gy as the maximum dose to the spinal cord PRV none of the treatment plans measured within the limits. Only treatment plan V4 measured an acceptable value for the spinal cord with a maximum dose of 15Gy. This illustrates some of the challenges with working in stereotactic radiotherapy, where the models are still not good enough to create reliable treatment plans. Looking at Tables 4.7-4.9 and 4.14-4.16 in Section 4.2 it was found that the differences in penumbra measured in film compared to dose plan exceeded 2mm and by that gives a very large uncertainty in stereotactic radiotherapy where margins are very small.
Comparing the treatment plans with each other one can compare V1 and V2, V2 and V3 and V2 and V4. V1 and V2 are the same except for V1 being with the use of a flattening filter. Looking at Figures 4.28 and 4.32 there does not seem to be much of a difference in the dose delivered to the GTV, CTV, PTV and lungs between the two treatment plans. A small increase in the delivered dose to the spinal cord in V2 (maximum dose 24.7Gy) compared to V1 (maximum dose 24.0Gy) goes in favour of choosing V1. This was unexpected as it has been seen in earlier reports that removing the flattening filter should reduce the out-of-field dose in stereotactic radiotherapy (Xiao et al. (2015)). However, looking at the differences found in dose plan and measured dose by the film, with deviations 6.6Gy and 8.8Gy for V1 and V2 respectively, it seems as if V2 might have been modelled more imprecise than V1 meaning that the uncertainty related to V2 is larger. Then the small difference in maximum dose between V1 and V2, measured as 0.7Gy, makes it difficult to

conclude which treatment plan should be preferred. The treatment time for V1 proved to be twice as long as for V2, which is an argument that V2 might be the better choice.

V2 and V3 difference in V2 being delivered as one arc while V3 is delivered with two small arcs, avoiding direct radiation from the front of the phantom. Looking at Figures 4.32 and 4.36 a small gain in the delivered dose to GTV, CTV and PTV is seen in V3 compared to V2. But for lungs there is not much of a difference. In V2 the maximum dose delivered to the spinal cord was 24.7Gy, while maximum dose to spinal cord in V3 was 22.1Gy. In addition, the deviation between measured dose by the film and dose plan was found to be 8.8Gy and 6.4Gy for V2 and V3 respectively, meaning that V3 seems to be built on a more reliable model compared to V2. With a slightly higher dose to GTV, CTV and PTV and at the same time a lower dose given to the spinal cord V3 proves to be a better plan than V2. The attempt to avoid direct irradiation to the front of the phantom, and by that directly at the spinal cord, seems to have worked. In addition, the delivery time for each treatment was about the same.

V4 and V2 are equal except for the collimator which in V4 are rotated 90 degrees, meaning that it is the jaws blocking out the spinal cord instead of the MLC. In these plans it seems as if the dose delivered to GTV, CTV, PTV and lungs are about the same, but the difference in dose delivered to the spinal cord is large. In Figures 4.33 and 4.41 and Tables 4.20 and 4.23 it is seen that the dose delivered to the spinal cord is considerably reduced moving from plan V2 to V4, going from a maximum dose of 24,7Gy to 15Gy. In addition, the treatment time is the same for both treatment plan, meaning that V4 seems to be favourable compared to V2. This shows how the blocking with the jaws works well at reducing the dose to the spinal cord and is an interesting result to further investigate in future work.

5.1.7 Further work in Fidora

In Fidora the position matching of the film onto the dose plan is done by reading the DICOM file RT Plan which follows when creating a dose plan in RayStation. To minimize the number of files and overview needed by the user a better solution could be to search the dose plan for a matching isocenter given by the film. An attempt of this was done in Fidora, but the solution took either too much of the computers Central Processing Unit (CPU) if running threads or too much time when running single program flow. If able to get around this problem in future work a new attempt could be worth the time. In the current version of Fidora when using a created reference point in the phantom to place the film, its position in the dose plan is based on isocenter and the displacement in vertical, longitudinal and lateral directions. Since these are parameters read from the RT Plan that means this must be renewed in a possible update of Fidora.

In the current version of Fidora it is assumed that the user places the film parallel to the scanner's sides. That means that the user must place the two given directions, up and side, in true positions relative to the scanner directions. If this is not correct the readout will be a little off compared to the dose plan. The reason why Fidora is written like this is because the data is stored as a matrix, which means that if one wants to be able to read the film when scanned in an angle the program needs to fit the readout through closest neighbour instead if reading the actual value. But if one can find a good solution to this problem, a further developed version of Fidora could benefit from being given this functionality as this would increase the generality of the program.

At this point it is only possible to upload one scanned image of a film and use this to compare to dose plan. The size of the film piece is limited both by its initial size (203.2mm x 254mm) and the size of the scanner surface (297mm x 210mm). In addition, since it is recommended that the film is scanned in landscape orientation that means that the largest possible film piece to scan has a size of (203.2mm x 210mm). Since Fidora was initially developed to handle treatment plans treating the breast or using stereotactic treatment, this was not a problem. But a further development version of Fidora where it was possible to upload several images of film and merge them together to create a larger field is relatively easily accomplished and would increase the usefulness of Fidora. In addition, this would serve a great deal to the usefulness of the tab DVH.

One of the central functionalities developed in Fidora during this work has been the development of the tab Profiles. As the profiles has been plotted in Fidora the user can use the computer mouse to hover across the plot and read out different results from the current position of the computer mouse. To further facilitate the work of reading the results in the plot, e.g. find penumbras, Fidora could be upgraded with the possibility of pointing out different positions in the plot and save the results on the screen for further comparison or reading of results as e.g. penumbra.

In the current version of Fidora the user can adjust the position of the film in two directions according to the dose plan. The two directions are defined by the plane at which the film has been irradiated in the phantom relative to the patient position (see Section 3.1.1). This means that the positioning of the film inside the phantom and the phantom on the treatment table are relatively large uncertainties that are not accounted for in Fidora. A further development of Fidora where the user is able to also adjust in the third direction, meaning that the user are able to scroll across the number of frames in the dose plan, could benefit the accuracy of Fidora.

Chapter 6

Conclusion

During this work it has been found that the GafChromic film EBT3 has worked well as a dosimeter when working with stereotactic treatment plans. As the film performs a 2D, continuous reading and at the same time proves to possess an adequate dose resolution it does a great job at representing the steep gradients and high doses delivered in stereotactic radiotherapy. To be able to use film as a dosimeter one is reliant on a working analysing tool, which should be flexible and with the opportunity to make changes upon needs. As such, an analysing software named Fidora has been developed during this work. In all the experiments done in this work Fidora has been used as the analysing tool, and it has been found that Fidora is reliable and presents the results as expected. The experiments of the MLC models and the stereotactic treatment plans both were performed as a proof of concept for Fidora. In both cases Fidora has worked well as an analysing tool, proven to be able to discover deviation in measured dose compared to planned dose plans as well as comparing different treatment plans.

In stereotactic radiotherapy one is dependent on precise delivery of dose, and as seen in this work the optimized dose plan often deviate from the measured fields. When adding together many small fields, which is the case in stereotactic treatment plans, it is important that each of the fields are modelled accurately. As seen in this work the penumbra seems to be one of the issues when modelling small fields in the treatment planning system. One of the assumed reasons for this is the poor modelling of overlapping penumbras, created by partially covered radiation source, which is not modelled in treatment planning systems such as RayStation. This results in wider fields during delivery of the dose for each of small fields, creating an overall larger dose than planned. The work with tuning a model to correctly represent the output from the linac is difficult and there are a lot of compromises that needs to be done when optimizing for all parameters. In addition, the tuning of the model depends on reliable measurements done during dosimetry, which for small fields has been difficult with the lack of a good dosimeter. In future work the tuning of the models in the treatment planning systems to better represent small fields should be prioritized and using film together with Fidora has proven to be a good way of analysing the measurements.

Bibliography

Ahnesjö, A., 1989. Collapsed cone convolution of radiant energy for photon dose calculation in heterogeneous media. Medical physics 16.

Ahnesjö, A., Aspradakis, M.M., 1999. Dose calculations for external photon beams in radiotherapy. Physics in Medicine & Biology 44, R99.

Andreo, P., Burns, D.T., Nahum, A.E., Seuntjens, J., Attix, F.H., 2017. Fundamentals of ionizing radiation dosimetry. John Wiley & Sons.

Ashland, 2020. Gafchromic ebt. URL: `http://www.gafchromic.com/gafchromic-film/radiotherapy-films/EBT/index.asp`.

Attix, F.H., 2008. Introduction to radiological physics and radiation dosimetry. John Wiley & Sons.

van Battum, L., 2018. Filmdosimetry: past and future .

Benedict, S.H., Yenice, K.M., Followill, D., Galvin, J.M., Hinson, W., Kavanagh, B., Keall, P., Lovelock, M., Meeks, S., Papiez, L., et al., 2010. Stereotactic body radiation therapy: the report of aapm task group 101. Medical physics 37, 4078–4101.

Borca, V.C., Pasquino, M., Russo, G., Grosso, P., Cante, D., Sciacero, P., Girelli, G., Porta, M.R.L., Tofani, S., 2013. Dosimetric characterization and use of gafchromic ebt3 film for imrt dose verification. Journal of applied clinical medical physics 14, 158–171.

Bourland, J.D., 2016. Radiation oncology physics, in: Clinical radiation oncology. Elsevier.

Brady, L.W., Heilmann, H., Molls, M., 2006. New technologies in radiation oncology. Springer.

Chen, S., Yi, B.Y., Yang, X., Xu, H., Prado, K.L., D'Souza, W.D., 2015. Optimizing the mlc model parameters for imrt in the raystation treatment planning system. Journal of applied clinical medical physics 16.

Dang, T.M., Peters, M.J., Hickey, B., Semciw, A., 2017. Efficacy of flattening-filter-free beam in stereotactic body radiation therapy planning and treatment: A systematic review with meta-analysis. Journal of medical imaging and radiation oncology 61, 379–387.

Das, I.J., Ding, G.X., Ahnesjö, A., 2008. Small fields: nonequilibrium radiation dosimetry. Medical physics 35, 206–215.

Elekta, 2020. Agility™ intelligent beam shaping. URL: `https://www.elekta.com/dam/jcr:6f125384-1fe5-47ea-b190-f9560f1eea81/Agility-product-brochure.pdf`.

Ewing, D., 1998. The oxygen fixation hypothesis: a reevaluation. American journal of clinical oncology 21, 355–361.

Håland, A.V., Gustavsen, S., 2019. Film based dosimetry.

Howell, R.M., Scarboro, S.B., Kry, S.F., Yaldo, D.Z., 2010. Accuracy of out-of-field dose calculations by a commercial treatment planning system. Physics in Medicine & Biology 55, 6999.

Jordan, T.J., Williams, P.C., 1994. The design and performance characteristics of a multi-leaf collimator. Physics in Medicine & Biology 39, 231.

Kennedy, J., 2012. Bresenham integer only line drawing algorithm. Santa Monica College, Santa Monica, CA 90405.

Li, H., Dong, L., Zhang, L., Yang, J.N., Gillin, M.T., Zhu, X.R., 2011. Toward a better understanding of the gamma index: Investigation of parameters with a surface-based distance method a. Medical physics 38, 6730–6741.

Liu, C., Simon, T.A., Fox, C., Li, J., Palta, J.R., 2008. Multileaf collimator characteristics and reliability requirements for imrt elekta system. International Journal of Radiation Oncology* Biology* Physics 71, S89–S92.

Low, D.A., Moran, J.M., Dempsey, J.F., Dong, L., Oldham, M., 2011. Dosimetry tools and techniques for imrt. Medical physics 38, 1313–1338.

Mathot, M., Sobczak, S., Hoornaert, M.T., 2014. Gafchromic film dosimetry: four years experience using filmqa pro software and epson flatbed scanners. Physica Medica 30, 871–877.

Mayles, P., Nahum, A., Rosenwald, J.C., 2007. Handbook of radiotherapy physics: theory and practice. CRC Press.

Micke, A., Lewis, D.F., Yu, X., 2011. Multichannel film dosimetry with nonuniformity correction. Medical physics 38, 2523–2534.

Oztek, M.A., Mayr, N.A., Mossa-Basha, M., Nyflot, M., Sponseller, P.A., Wu, W., Hofstetter, C.P., Saigal, R., Bowen, S.R., Hippe, D.S., et al., 2020. The dancing cord: Inherent spinal cord motion and its effect on cord dose in spine stereotactic body radiation therapy. Neurosurgery .

Palmans, H., Andreo, P., Huq, M.S., Seuntjens, J., Christaki, K.E., Meghzifene, A., 2018. Dosimetry of small static fields used in external photon beam radiotherapy: Summary of trs-483, the iaea–aapm international code of practice for reference and relative dose determination. Medical physics 45, e1123–e1145.

Parwaie, W., Refahi, S., Ardekani, M.A., Farhood, B., 2018. Different dosimeters/detectors used in small-field dosimetry: Pros and cons. Journal of medical signals and sensors 8, 195.

Podgorsak, E.B., et al., 2005. Radiation oncology physics. Vienna: International Atomic Energy Agency , 22–84 and 161–222.

Polykarpou, E., Kyriakides, E., 2016. Parameter estimation for measurement-based load modeling using the levenberg-marquardt algorithm, in: 2016 18th Mediterranean Electrotechnical Conference (MELECON), IEEE. pp. 1–6.

Proimos, B.S., 1960. Synchronous field shaping in rotational megavolt therapy. Radiology 74, 753–757.

Sánchez-Doblado, F., Hartmann, G., Pena, J., Roselló, J., Russiello, G., Gonzalez-Castaño, D., 2007. A new method for output factor determination in mlc shaped narrow beams. Physica medica 23, 58–66.

Sauer, O.A., Wilbert, J., 2007. Measurement of output factors for small photon beams. Medical physics 34, 1983–1988.

Saur, S., Frengen, J., 2008. Gafchromic ebt film dosimetry with flatbed ccd scanner: a novel background correction method and full dose uncertainty analysis. Medical physics 35, 3094–3101.

Sorriaux, J., Kacperek, A., Rossomme, S., Lee, J.A., Bertrand, D., Vynckier, S., Sterpin, E., 2013. Evaluation of gafchromic® ebt3 films characteristics in therapy photon, electron and proton beams. Physica Medica 29, 599–606.

Trump, J.G., Wright, K.A., Smedal, M.I., Salzman, F.A., 1961. Synchronous field shaping and protection in 2-million-volt rotational therapy. Radiology 76, 275–275.

Van Herk, M., 2004. Errors and margins in radiotherapy, in: Seminars in radiation oncology, Elsevier. pp. 52–64.

Wan, J., 2017. Exploring RayStation Treatment Planning System: Commissioning Varian TrueBeam Photon and Electron Energies, and Feasibility of Using FFF Photon Beam to Deliver Conventional Flat Beam. Ph.D. thesis. University of Toledo.

Wang, L., Ding, G.X., 2018. Estimating the uncertainty of calculated out-of-field organ dose from a commercial treatment planning system. Journal of applied clinical medical physics 19, 319–324.

Wangler, T.P., 2008. RF Linear accelerators. John Wiley & Sons.

Wen, N., Lu, S., Kim, J., Qin, Y., Huang, Y., Zhao, B., Liu, C., Chetty, I.J., 2016. Precise film dosimetry for stereotactic radiosurgery and stereotactic body radiotherapy quality assurance using gafchromic™ ebt3 films. Radiation Oncology 11, 132.

Wilke, L., Andratschke, N., Blanck, O., Brunner, T.B., Combs, S.E., Grosu, A.L., Moustakis, C., Schmitt, D., Baus, W.W., Guckenberger, M., 2019. Icru report 91 on prescribing, recording, and reporting of stereotactic treatments with small photon beams. Strahlentherapie und Onkologie 195, 193–198.

Winer-Muram, H.T., Jennings, S.G., Meyer, C.A., Liang, Y., Aisen, A.M., Tarver, R.D., McGarry, R.C., 2003. Effect of varying ct section width on volumetric measurement of lung tumors and application of compensatory equations. Radiology 229, 184–194.

World Health Orginasation, 2018. Cancer. URL: `https://www.who.int/news-room/fact-sheets/detail/cancer`.

Xiao, Y., Kry, S.F., Popple, R., Yorke, E., Papanikolaou, N., Stathakis, S., Xia, P., Huq, S., Bayouth, J., Galvin, J., et al., 2015. Flattening filter-free accelerators: a report from the aapm therapy emerging technology assessment work group. Journal of applied clinical medical physics 16, 12–29.

Zaghian, R., SedighiPashaki, A., Haghparast, A., Gholami, M., Mohammadi, M., 2020. Investigation of collapsed-cone algorithm accuracy in small fields and heterogeneous environments. Journal of Biomedical Physics and Engineering .

Zhao, C., Shi, W., Deng, Y., 2005. A new hausdorff distance for image matching. Pattern Recognition Letters 26, 581–586.

Appendix

Fidora - code scripts

notebook.py

```
#-----------------------------------------------------------------------------
#
#
# Version 17.08.20
#
# Written by Stine Gustavsen and Ane Vigre Haaland as part of
# a master thesis in Biophysics and Medical Technology at NTNU.
# The program is originally meant to be used at St. Olavs Hospital
# in the radiation clinic.
#
#-----------------------------------------------------------------------------

import tkinter as tk
from tkinter import ttk, INSERT, DISABLED, GROOVE, CURRENT, Radiobutton, \
    NORMAL, ACTIVE, messagebox, Menu, IntVar, Checkbutton, FLAT, PhotoImage, Label,\
        SOLID, N, S, W, E, END, LEFT, Scrollbar, RIGHT, Y, BOTH, TOP, OptionMenu, \
            SUNKEN, RIDGE, BOTTOM, X
import Globals
import re
import CoMet_functions, intro_tab_functions, Map_Dose_functions
import Dose_response_functions, Profile_functions, DVH_functions
from PIL import Image, ImageTk
import os
import sys


Globals.form.title("FIDORA")
Globals.form.configure(bg='#ffffff')
Globals.form.state('zoomed')

Globals.form.tk.call('wm', 'iconphoto', Globals.form._w, \
    PhotoImage(file='logo_fidora.png'))
```

```
Globals.form.iconbitmap(default='logo_fidora.png')
load = Image.open("fidora_logo.png")
render = ImageTk.PhotoImage(load)
label = Label(Globals.scroll_frame, image=render)
label.image = render
label.grid(row = 0, column = 0, sticky=W)
label.config(bg='#FFFFFF')

Globals.tab_parent.add(Globals.intro_tab, text='FIDORA')
Globals.tab_parent.add(Globals.tab1, text='CoMet')
Globals.tab_parent.add(Globals.tab2, text='Dose Response')
#Globals.tab_parent.add(Globals.tab3, text='Map dose') #Under development
Globals.tab_parent.add(Globals.tab4, text='Profiles')
Globals.tab_parent.add(Globals.tab5,text='DVH')

#--------------------------------------------------------------------------------
# Set the style for all GUI related to Fidora.
# Style is choosen by the authors
#
#Horizontal.TProgressbar -> progressbar used in CoMet
#TNotebook -> Notebook which holds every functionality in Fidora
#TNotebook.tab -> set style for each tab
#Treeview -> listbox used in Profiles
#--------------------------------------------------------------------------------
style = ttk.Style()
style.theme_create('MyStyle', parent= 'classic', settings={
    ".": {
        "configure": {
            "background": '#FFFFFF',
            "font": 'red'
        }
    },
    "Horizontal.TProgressbar":{
        "configure": {
            "background": '#2C8EAD',
            "bordercolor": '#32A9CE',
            "troughcolor": "#ffffff",
        }
    },
    "TNotebook": {
        "configure": {
            "background":'#ffffff',
            "tabmargins": [5, 5, 10, 10],
            "tabposition": 'wn',
            "borderwidth": 0,
        }
    },
    "TNotebook.Tab": {
        "configure": {
            "background": '#0A7D76',
            "foreground": '#ffffff',
            "padding": [30,35, 20,35],
            "font": ('#FFFFFF', '15'),
            "borderwidth": 1,
            "equalTabs": True,
            "width": 13
        },
        "map": {
            "background": [("selected", '#02B9A5')],
            "expand": [("selected", [1, 1, 1, 0])]
        }
    },
    "Treeview":{
        "configure":{
            "font": ('calibri', '9'),
            "highlightthickness": 0,
            "relief": FLAT,
            "borderwidth": 0
        }
    },
    "Treeview.Heading":{
        "configure":{
            "font": ('calibri', '9'),
            "highlightthickness": 0,
            "relief": FLAT,
            "borderwidth": 0,
            "anchor": W
        }
    }
})
style.theme_use('MyStyle')

#------------------------------------------------------------------------
# Creating a menubar (visible at top left of window)
# Buttons: File, Help, Specification
#    File -> Restart, open, exit
#    Help -> Help, about
#    Specification -> Scanner settings, calibration, raystation
#------------------------------------------------------------------------
menubar = Menu(Globals.form)
```

```
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Restart", command=CoMet_functions.nothingButton)
filemenu.add_command(label="Open", command=CoMet_functions.nothingButton)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=Globals.form.quit)
menubar.add_cascade(label="File", menu=filemenu)

helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help", command=CoMet_functions.nothingButton)
helpmenu.add_command(label="About", command=CoMet_functions.nothingButton)
menubar.add_cascade(label="Help", menu=helpmenu)

scannermenu=Menu(menubar, tearoff=0)
scannermenu.add_command(label="Scanner settings", \
    command=intro_tab_functions.createScannerSettingsWindow)
scannermenu.add_command(label="Calibration", \
    command=intro_tab_functions.createCalibrationWindow)
scannermenu.add_command(label="Raystation", \
    command=intro_tab_functions.createRaystationWindow)
menubar.add_cascade(label="Specifications", menu=scannermenu)

Globals.form.config(menu=menubar)

#----------------------------------------------------------------------------
# Upload all images used in Fidora.
# Images is created by the authors.
#----------------------------------------------------------------------------
upload_button_file = "uploadbutton3.png"
Globals.upload_button_image = ImageTk.PhotoImage(file=upload_button_file)

select_folder_button_file = "select_folder_button2.png"
select_folder_image = ImageTk.PhotoImage(file=select_folder_button_file)

help_button_file = "help_button.png"
Globals.help_button = ImageTk.PhotoImage(file=help_button_file)

done_button_file = "done_button.png"
Globals.done_button_image = ImageTk.PhotoImage(file=done_button_file)

CoMet_border_dark_file = "border.png"
CoMet_border_dark = ImageTk.PhotoImage(file=CoMet_border_dark_file)

CoMet_border_light_file = "border_light.png"
CoMet_border_light = ImageTk.PhotoImage(file=CoMet_border_light_file)
```

```
CoMet_save_button_file = "save_button2.png"
CoMet_save_button = ImageTk.PhotoImage(file=CoMet_save_button_file)
Globals.save_button = ImageTk.PhotoImage(file=CoMet_save_button_file)

CoMet_correct_button_file = "correct_button.png"
CoMet_correct_button_image= ImageTk.PhotoImage(file=CoMet_correct_button_file)

CoMet_clear_all_button_file = "icon_clear_all.png"
CoMet_clear_all_button_image = ImageTk.PhotoImage(file=CoMet_clear_all_button_file)

dose_response_clear_all_button_file = "icon_clear_all_small.png"
dose_response_clear_all_button_image = \
    ImageTk.PhotoImage(file=dose_response_clear_all_button_file)

CoMet_empty_image_file = "empty_corrected_image.png"
CoMet_empty_image_image = ImageTk.PhotoImage(file=CoMet_empty_image_file)

dose_response_calibration_button_file = "save_calibration_button.png"
dose_response_calibration_button_image = \
    ImageTk.PhotoImage(file=dose_response_calibration_button_file)

dose_response_dose_border_file = "dose_border.png"
Globals.dose_response_dose_border = \
    ImageTk.PhotoImage(file=dose_response_dose_border_file)

profiles_add_doseplan_button_file = "add_doseplan_button.png"
Globals.profiles_add_doseplan_button_image = \
    ImageTk.PhotoImage(file=profiles_add_doseplan_button_file)

profiles_add_film_button_file = "add_film_button.png"
profiles_add_film_button_image = \
    ImageTk.PhotoImage(file=profiles_add_film_button_file)

profiles_add_rtplan_button_file = "add_rtplan_button.png"
profiles_add_rtplan_button_image = \
    ImageTk.PhotoImage(file=profiles_add_rtplan_button_file)

profiles_showPlanes_file = "planes.png"
Globals.profiles_showPlanes_image = \
    ImageTk.PhotoImage(file=profiles_showPlanes_file)

profiles_showDirections_file = 'depth_directions.png'
Globals.profiles_showDirections_image = \
    ImageTk.PhotoImage(file=profiles_showDirections_file)
```

```
profiles_mark_isocenter_button_file = 'mark_isocenter_button.png'
Globals.profiles_mark_isocenter_button_image = \
    ImageTk.PhotoImage(file=profiles_mark_isocenter_button_file)

profiles_mark_ROI_button_file = "mark_ROI_button.png"
Globals.profiles_mark_ROI_button_image = \
    ImageTk.PhotoImage(file=profiles_mark_ROI_button_file)

profiles_scanned_image_text_image_file = "scanned_image_text_image.png"
Globals.profiles_scanned_image_text_image = \
    ImageTk.PhotoImage(file=profiles_scanned_image_text_image_file)

profiles_film_dose_map_text_image_file = "film_dose_map_text_image.png"
Globals.profiles_film_dose_map_text_image = \
    ImageTk.PhotoImage(file=profiles_film_dose_map_text_image_file)

profiles_doseplan_text_image_file = "doseplan_text_image.png"
Globals.profiles_doseplan_text_image = \
    ImageTk.PhotoImage(file=profiles_doseplan_text_image_file)

profiles_mark_point_file = "mark_point_button.png"
Globals.profiles_mark_point_button_image = \
    ImageTk.PhotoImage(file=profiles_mark_point_file)

profiles_add_doseplans_button_file = "add_doseplan.png"
Globals.profiles_add_doseplans_button_image = \
    ImageTk.PhotoImage(file=profiles_add_doseplans_button_file)

adjust_button_left_file = "adjust_button_left.png"
Globals.adjust_button_left_image = ImageTk.PhotoImage(file=adjust_button_left_file)

adjust_button_right_file = "adjust_button_right.png"
Globals.adjust_button_right_image = \
    ImageTk.PhotoImage(file=adjust_button_right_file)

adjust_button_down_file = "adjust_button_down.png"
Globals.adjust_button_down_image = ImageTk.PhotoImage(file=adjust_button_down_file)

adjust_button_up_file = "adjust_button_up.png"
Globals.adjust_button_up_image = ImageTk.PhotoImage(file=adjust_button_up_file)

dose_response_upload_files_here_file = "upload_here_dose_response.png"
Globals.dose_response_upload_files_here = \
    ImageTk.PhotoImage(file=dose_response_upload_files_here_file)
```

```
dose_response_equation_will_be_here_file = "equation_will_be_here.png"
Globals.dose_response_equation_written_here = \
    ImageTk.PhotoImage(file=dose_response_equation_will_be_here_file)

export_plot_file = "export_plot_button.png"
Globals.export_plot_button_image = ImageTk.PhotoImage(file=export_plot_file)

DVH_add_structure_file = "add_structure.png"
DVH_add_structure_button_image = ImageTk.PhotoImage(file=DVH_add_structure_file)

DVH_temp_image_file = "temp_doseplan_roi.png"
Globals.dVH_temp_image = ImageTk.PhotoImage(file=DVH_temp_image_file)


####################################                ################################
#################################### Tab 1 INTRO TAB ################################
####################################                ################################

#-----------------------------------------------------------------------------
# Code to place all widgets on the first tab "Fidora" in Fidora
# They are all placed in a canvas, intro_tab_canvas given in the global tab
# intro_tab, which is defined in the file Globals.py.
#-----------------------------------------------------------------------------
intro_tab_canvas = tk.Canvas(Globals.intro_tab)
intro_tab_canvas.config(bg='#ffffff', bd = 0, relief=FLAT, highlightthickness=0)

tab1_text_box = tk.Frame(intro_tab_canvas, height=230, width=400)
tab1_text_box.grid(row=0, column=0, pady=(30,30), padx=(55,0))
tab1_text_box.config(bd=0, bg='#E5f9ff')

tab1_title_text = tk.Text(tab1_text_box, height=1, width=6)
tab1_title_text.insert(END, "CoMet")
tab1_title_text.grid(in_=tab1_text_box, row=0, column = 0, \
    pady=(15,5), padx=(10,10))
tab1_title_text.config(state=DISABLED, bd=0, bg ='#E5f9ff', \
    fg='#130e07', font=('calibri', '25', 'bold'))
tab1_text_box.grid_columnconfigure(0,weight=1)
tab1_text_box.grid_rowconfigure(0,weight=1)

tab1_text = tk.Text(tab1_text_box, height=4, width=43)
tab1_text.grid(in_=tab1_text_box, row=1, column=0, sticky=N+S+W+E, \
    pady=(0,0), padx=(20,20))
tab1_text.insert(INSERT,"Correct your scanned images using CoMet. A method \n\
developed to correct for non-uniformity introduced\n by the scanner. \
The correction is based on absolute \nsubtraction.")
```

```
tab1_text.config(state=DISABLED, bd=0, bg='#E5f9ff', \
    fg='#130E07', font=('calibri', '13'))
tab1_text_box.grid_columnconfigure(1,weight=1)
tab1_text_box.grid_rowconfigure(1,weight=1)

tab1_box_figure = Image.open("icon_comet.png")
tab1_figure = ImageTk.PhotoImage(tab1_box_figure)
tab1_figure_label = Label(tab1_text_box, image=tab1_figure)
tab1_figure_label.image = tab1_figure
tab1_figure_label.grid(row=3, sticky=N+S+W+E, pady=(0,10))
tab1_figure_label.config(bg='#E5f9ff')
tab1_text_box.grid_columnconfigure(3, weight=1)
tab1_text_box.grid_rowconfigure(3, weight=1)

tab2_text_box = tk.Frame(intro_tab_canvas, height=230, width=400)
tab2_text_box.grid(row=0, column=1, pady=(30,30), padx=(65,0))
tab2_text_box.config(bd=0, bg='#E5f9ff')

tab2_title = tk.Text(tab2_text_box, height=1, width=12)
tab2_title.grid(in_=tab2_text_box, row=0, column = 0, \
    pady=(15,5), padx=(10,10))
tab2_title.insert(INSERT, "Dose response")
tab2_title.config(state=DISABLED, bd=0, bg = '#E5f9ff', \
    fg='#130e07', font=('calibri', '25', 'bold'))
tab2_text_box.grid_columnconfigure(0, weight=1)
tab2_text_box.grid_rowconfigure(0, weight=1)

tab2_text = tk.Text(tab2_text_box, height=4, width=43)
tab2_text.grid(in_=tab2_text_box, row=1, column=0, \
    sticky=N+S+W+E, pady=(0,0), padx=(20,20))
tab2_text.insert(INSERT,"Make a calibration curve and read the dose response \n\
function. For every new batch of GafChromic film \nthere is a need to update\
the dose response. All three \nchannels (RGB) are read and calculated.")
tab2_text.config(state=DISABLED, bd=0, bg='#E5f9ff', \
    fg='#130E07', font=('calibri', '13'))
tab2_text_box.grid_columnconfigure(1, weight=1)
tab2_text_box.grid_rowconfigure(1, weight=1)

tab2_box_figure = Image.open("icon_map_dose.png")
tab2_figure = ImageTk.PhotoImage(tab2_box_figure)
tab2_figure_label = Label(tab2_text_box, image=tab2_figure)
tab2_figure_label.image = tab2_figure
tab2_figure_label.grid(row=3, sticky=N+S+W+E, pady=(0,10))
tab2_figure_label.config(bg='#E5f9ff')
tab2_text_box.grid_columnconfigure(3, weight=1)
```

```
tab2_text_box.grid_rowconfigure(3, weight=1)

tab3_text_box = tk.Frame(intro_tab_canvas, height=230, width=400)
tab3_text_box.grid(row=1, column=0, pady=(0,30), padx=(55,0))
tab3_text_box.config(bd=0, bg='#E5f9ff')

tab3_title = tk.Text(tab3_text_box, height=1, width=8)
tab3_title.grid(in_=tab3_text_box, row=0, column = 0, pady=(15,5), padx=(10,10))
tab3_title.insert(INSERT, "Profiles")
tab3_title.config(state=DISABLED, bd=0, bg = '#E5f9ff', \
    fg='#130e07', font=('calibri', '25', 'bold'))
tab3_text_box.grid_columnconfigure(0, weight=1)
tab3_text_box.grid_rowconfigure(0, weight=1)

tab3_text = tk.Text(tab3_text_box, height=4, width=43)
tab3_text.grid(in_=tab3_text_box, row=1, column=0, \
    sticky=N+S+W+E, pady=(0,0), padx=(20,20))
tab3_text.insert(INSERT,"Investigate profiles measured using GafChromic \n\
film and compare with the profiles in your treatment \nplan. Draw vertical, \
horizontal or manually drawn profiles.")
tab3_text.config(state=DISABLED, bd=0, bg='#E5f9ff', \
    fg='#130E07', font=('calibri', '13'))
tab3_text_box.grid_columnconfigure(1, weight=1)
tab3_text_box.grid_rowconfigure(1, weight=1)

tab3_box_figure = Image.open("icon_dose_response.png")
tab3_figure = ImageTk.PhotoImage(tab3_box_figure)
tab3_figure_label = Label(tab3_text_box, image=tab3_figure)
tab3_figure_label.image = tab3_figure
tab3_figure_label.grid(row=3, sticky=N+S+W+E, pady=(0,10))
tab3_figure_label.config(bg='#E5f9ff')
tab3_text_box.grid_columnconfigure(3, weight=1)
tab3_text_box.grid_rowconfigure(3, weight=1)

tab4_text_box = tk.Frame(intro_tab_canvas, height=230, width=400)
tab4_text_box.grid(row=1, column=1, pady=(0,30), padx=(65,0))
tab4_text_box.config(bd=0, bg='#E5f9ff')

tab4_title = tk.Text(tab4_text_box, height=1, width=7)
tab4_title.grid(in_=tab4_text_box, row=0, column = 0, pady=(15,5), padx=(10,10))
tab4_title.insert(INSERT, "DVH")
tab4_title.config(state=DISABLED, bd=0, bg = '#E5f9ff', \
    fg='#130e07', font=('calibri', '25', 'bold'))
tab4_text_box.grid_columnconfigure(0,weight=1)
tab4_text_box.grid_rowconfigure(0, weight=1)
```

```
tab4_text = tk.Text(tab4_text_box, height=4, width=43)
tab4_text.grid(in_=tab4_text_box, row=1, column=0, \
    sticky=N+S+W+E, pady=(0,0), padx=(20,20))
tab4_text.insert(INSERT,"Study the dose volume histogram measured\n\
in your scanned film and doseplan for comparison. \nInclude the \
volumes of your choice.")
tab4_text.config(state=DISABLED, bd=0, bg='#E5f9ff', \
    fg='#130E07', font=('calibri', '13'))
tab4_text_box.grid_columnconfigure(1, weight=1)
tab4_text_box.grid_rowconfigure(1, weight=1)

tab4_box_figure = Image.open("icon_profiles.png")
tab4_figure = ImageTk.PhotoImage(tab4_box_figure)
tab4_figure_label = Label(tab4_text_box, image=tab4_figure)
tab4_figure_label.image = tab4_figure
tab4_figure_label.grid(row=3, sticky=N+S+W+E, pady=(0,10))
tab4_figure_label.config(bg='#E5f9ff')
tab4_text_box.grid_columnconfigure(3, weight=1)
tab4_text_box.grid_rowconfigure(3, weight=1)

intro_tab_canvas.grid(row=0, column=0, sticky=N+S+W)

##########################################                ######################################
########################################## TAB 2 — CoMet ######################################
##########################################                ######################################

#-----------------------------------------------------------------------------------------------
# Code to place widgets in the second tab "CoMet" in Fidora
# They are all placed in the global canvas tab1_canvas defined
# in the the file Globals.py. Where functions is called the
# functions are written in the file CoMet_functions.py
#-----------------------------------------------------------------------------------------------

Globals.tab1_canvas.config(bg='#ffffff', bd = 0, relief=FLAT, highlightthickness=0)

CoMet_explained = tk.Text(Globals.tab1_canvas, height=5, width=84)
CoMet_explained.insert(INSERT, \
"Start the correction by choosing the correct *.tif file containing the scanned \n\
image of the GafChromic film. The film should be scanned using Epson Perfection \n\
v750 Pro with dpi setting 72 or 127. Then pick which folder the corrected file \n\
should be uploaded to. The corrected file will be saved as a DICOM. Write \
filename \nand patient name (optional) before doing the correction. An illustration \
of the \ncorrected image will appear.")
CoMet_explained.grid(row=0, column = 0, columnspan=1, \
    sticky=N+S+E+W, padx=(20,0), pady=(10,10))
Globals.tab1_canvas.grid_columnconfigure(0, weight=0)
Globals.tab1_canvas.grid_rowconfigure(0, weight=0)
CoMet_explained.config(state=DISABLED, bg='#ffffff', \
    font=('calibri', '11'), relief=FLAT)

Globals.CoMet_border_1_label = Label(Globals.tab1_canvas, \
    image = CoMet_border_dark,width=50)
Globals.CoMet_border_1_label.image=CoMet_border_dark
Globals.CoMet_border_1_label.grid(row=2, column=0, columnspan=2, \
    sticky=N+E+W, padx=(0,80), pady=(0,0))
Globals.tab1_canvas.grid_columnconfigure(1, weight=0)
Globals.tab1_canvas.grid_rowconfigure(1, weight=0)
Globals.CoMet_border_1_label.config(bg='#ffffff', borderwidth=0)

CoMet_upload_button_frame = tk.Frame(Globals.tab1_canvas)
CoMet_upload_button_frame.grid(row=2, column = 0, \
    padx = (200, 0), pady=(0,0), sticky=N)
Globals.tab1_canvas.grid_columnconfigure(2, weight=0)
Globals.tab1_canvas.grid_rowconfigure(2, weight=0)
CoMet_upload_button_frame.config(bg = '#ffffff')

CoMet_upload_button = tk.Button(CoMet_upload_button_frame, text='Browse', \
    image = Globals.upload_button_image, cursor='hand2',font=('calibri', '9'), \
        relief=FLAT, state=ACTIVE, command=CoMet_functions.UploadAction)
CoMet_upload_button.pack(expand=True, fill=BOTH)
CoMet_upload_button.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
CoMet_upload_button.image = Globals.upload_button_image

Globals.CoMet_uploaded_file_text = tk.Text(Globals.CoMet_border_1_label, \
    height=1, width=31)
Globals.CoMet_uploaded_file_text.grid(row=0, column=0, columnspan=2, \
    sticky=E, pady=(20,20), padx=(100,0))
Globals.CoMet_uploaded_file_text.insert(INSERT, \
    "Upload the image you want to correct")
Globals.CoMet_uploaded_file_text.config(state=DISABLED, bd=0, \
    font=('calibri', '10'), fg='gray', bg='#ffffff')

Globals.CoMet_border_2_label = Label(Globals.tab1_canvas, image = \
    CoMet_border_dark,width=50)
Globals.CoMet_border_2_label.image=CoMet_border_dark
Globals.CoMet_border_2_label.grid(row=3, column=0, columnspan=2, \
    sticky = N+W+E, padx = (0, 80), pady=(5,0))
Globals.tab1_canvas.grid_columnconfigure(3, weight=0)
Globals.tab1_canvas.grid_rowconfigure(3, weight=0)
Globals.CoMet_border_2_label.config(bg='#ffffff', borderwidth=0)

CoMet_folder_button_frame = tk.Frame(Globals.tab1_canvas)
CoMet_folder_button_frame.grid(row=3, column = 0, padx = (200, 0), \
    pady=(10,0), sticky=N)
Globals.tab1_canvas.grid_columnconfigure(4, weight=0)
Globals.tab1_canvas.grid_rowconfigure(4, weight=0)
CoMet_folder_button_frame.config(bg = '#ffffff')

CoMet_folder_button = tk.Button(CoMet_folder_button_frame, text='Browse', \
    image = select_folder_image ,cursor='hand2',font=('calibri', '14'),\
    relief=FLAT, state=ACTIVE, command=CoMet_functions.setCoMet_export_folder)
CoMet_folder_button.pack(expand=True, fill=BOTH)
CoMet_folder_button.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
CoMet_folder_button.image=select_folder_image

CoMet_save_to_folder = tk.Text(Globals.CoMet_border_2_label, height=1, width=31)
CoMet_save_to_folder.grid(row=0, column=0, columnspan=2, \
    sticky=E, pady=(20,20), padx=(100,0))
CoMet_save_to_folder.insert(INSERT,"Folder to save the corrected image")
CoMet_save_to_folder.config(state=DISABLED, bd=0, \
    font=('calibri', '10'), fg='gray', bg='#ffffff')

def testFilename():
#------------------------------------------------------------
# Function to test the filename the user chooses for
# the corrected image. The filename must be no longer
# than 20 characters and only letters and/or
# numbers are valid characters. Default: "Error!".
# Once approved the filename is given to the global
# variable CoMet_corrected_image_filename
#------------------------------------------------------------
    Globals.CoMet_corrected_image_filename.set\
        (Globals.CoMet_save_filename.get("1.0",'end-1c'))
    if(Globals.CoMet_corrected_image_filename.get() == \
        " " or Globals.CoMet_corrected_image_filename.get() == "Filename"):
        Globals.CoMet_corrected_image_filename.set("Error!")
    elif(len(Globals.CoMet_corrected_image_filename.get()) >21):
        messagebox.showerror("Error", "The filename must be under 20 characters")
        Globals.CoMet_corrected_image_filename.set("Error!")
    elif(re.match("^[A-Za-z0-9_]*$", \
        (Globals.CoMet_corrected_image_filename.get()).lstrip())==None):
        messagebox.showerror\
            ("Error","Filename can only contain letters and/or numbers")
        Globals.CoMet_corrected_image_filename.set("Error!")
    else:
        Globals.CoMet_save_button_1.config(state=DISABLED)
        Globals.CoMet_save_filename.config(state=DISABLED)
        Globals.CoMet_progressbar_counter += 1
        Globals.CoMet_progressbar["value"] = Globals.CoMet_progressbar_counter*25
        Globals.CoMet_progressbar_text = \
            tk.Text(Globals.tab1_canvas, width = 5, height=1)
        Globals.CoMet_progressbar_text.grid(row=1, column=0, columnspan=1, \
            sticky=E, padx=(0,158), pady=(0,36))
        Globals.CoMet_progressbar_text.insert(INSERT, \
            str(Globals.CoMet_progressbar_counter*25) + "%")
        if(Globals.CoMet_progressbar_counter*25 == 100):
            Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0, \
                relief=FLAT, bg='#2C8EAD', font=('calibri', '10', 'bold'))
        else:
            Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0, \
                relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))


Globals.CoMet_border_3_label = Label(Globals.tab1_canvas, \
    image = CoMet_border_dark)
Globals.CoMet_border_3_label.image=CoMet_border_dark
Globals.CoMet_border_3_label.grid(row=4, column=0, columnspan=2, \
    sticky = N+W+E, padx = (0,80), pady=(5,0))
Globals.tab1_canvas.grid_columnconfigure(5, weight=0)
Globals.tab1_canvas.grid_rowconfigure(5, weight=0)
Globals.CoMet_border_3_label.config(bg='#ffffff', borderwidth=0)

Globals.CoMet_save_button_frame_1 = tk.Frame(Globals.tab1_canvas)
Globals.CoMet_save_button_frame_1.grid(row=4, column = 0, \
    padx = (200, 0), pady=(10,0), sticky=N)
Globals.tab1_canvas.grid_columnconfigure(6, weight=0)
Globals.tab1_canvas.grid_rowconfigure(6, weight=0)
Globals.CoMet_save_button_frame_1.config(bg = '#ffffff')

Globals.CoMet_save_button_1 = tk.Button(Globals.CoMet_save_button_frame_1, \
    text='Save', image = CoMet_save_button ,cursor='hand2',font=('calibri', '14'),\
        relief=FLAT, state=ACTIVE, command=testFilename)
Globals.CoMet_save_button_1.pack(expand=True, fill=BOTH)
Globals.CoMet_save_button_1.config(bg='#ffffff', \
    activebackground='#ffffff', activeforeground='#ffffff', highlightthickness=0)
Globals.CoMet_save_button_1.image = CoMet_save_button
Globals.CoMet_save_filename = tk.Text(Globals.CoMet_border_3_label,\
     height=1, width=30)
Globals.CoMet_save_filename.grid(row=0, column=0, columnspan=2, \
    sticky=E+W, pady=(20,20), padx=(100,0))
Globals.CoMet_save_filename.insert(END,"Filename (will be saved as *.dcm)")
Globals.CoMet_save_filename.config(state=NORMAL, bd=0, \
    font=('calibri', '10'), fg='gray', bg='#ffffff')

def writeFilename(event):
#---------------------------------------------------
# Callback function to mouse-events.
# Function to delete the default text "Filename (will
# be saved as *.dcm)\n" in the global textbox
# CoMet_save_filename when focus is on textbox and
# insert the default text as focus is out unless
# a filename has been written by the user.
#---------------------------------------------------
    current = Globals.CoMet_save_filename.get("1.0", tk.END)
    if(current == "Filename (will be saved as *.dcm)\n"):
        Globals.CoMet_save_filename.delete("1.0", tk.END)
    else:
        Globals.CoMet_save_filename.insert("1.0", \
            "Filename (will be saved as *.dcm)")

Globals.CoMet_save_filename.bind("<FocusIn>", writeFilename)
Globals.CoMet_save_filename.bind("<FocusOut>", writeFilename)


def testName():
#---------------------------------------------------
# Function to test if the user defined global
# variable CoMet_patientName has been given a valid
# value. The patient name/ID must be less than 30
# characters and letters and/or numbers. Once the
# value is approved it is given to the gloabal
# variable CoMet_patientName
#---------------------------------------------------
    Globals.CoMet_patientName.set(CoMet_save_patientName.get("1.0",'end-1c'))
    if(Globals.CoMet_patientName.get() == " " or \
        Globals.CoMet_patientName.get() == "Patient name"):
        Globals.CoMet_patientName.set("Error!")
    elif(len(Globals.CoMet_patientName.get()) >31):
        messagebox.showerror("Error", "The Name must be under 30 characters")
        Globals.CoMet_patientName.set("Error!")
    elif(re.match("^[A-Za-z0-9_]*$", \
        (Globals.CoMet_patientName.get()).lstrip())==None):
        messagebox.showerror("Error",\
            "Name can only contain english letters and no spaces")
        Globals.CoMet_patientName.set("Error!")
    else:
        CoMet_save_button_2.config(state=DISABLED)
        CoMet_save_patientName.config(state=DISABLED)



Globals.CoMet_border_4_label = Label(Globals.tab1_canvas, \
    image = CoMet_border_dark)
Globals.CoMet_border_4_label.image=CoMet_border_dark
Globals.CoMet_border_4_label.grid(row=5, column=0, columnspan=2, \
    sticky =N+W+E, padx = (0, 80), pady=(0,3))
Globals.tab1_canvas.grid_columnconfigure(7, weight=0)
Globals.tab1_canvas.grid_rowconfigure(7, weight=0)
Globals.CoMet_border_4_label.config(bg='#ffffff', borderwidth=0)

CoMet_save_button_frame_2 = tk.Frame(Globals.tab1_canvas)
CoMet_save_button_frame_2.grid(row=5, column = 0, \
    padx = (200, 0), pady=(3,0), sticky=N)
Globals.tab1_canvas.grid_columnconfigure(8, weight=0)
Globals.tab1_canvas.grid_rowconfigure(8, weight=0)
CoMet_save_button_frame_2.config(bg = '#ffffff')

CoMet_save_button_2 = tk.Button(CoMet_save_button_frame_2, \
    text='Save', image = CoMet_save_button ,cursor='hand2',\
        font=('calibri', '14'),relief=FLAT, state=ACTIVE, command=testName)
CoMet_save_button_2.pack(expand=True, fill=BOTH)
CoMet_save_button_2.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
CoMet_save_button_2.image = CoMet_save_button

CoMet_save_patientName = tk.Text(Globals.CoMet_border_4_label, height=1, width=30)
CoMet_save_patientName.grid(row=0, column=0, columnspan=2, \
    sticky=E+W, pady=(20,20), padx=(100,0))
CoMet_save_patientName.insert(END,"Patient name (Optional)")
CoMet_save_patientName.config(state=NORMAL, bd=0, \
    font=('calibri', '10'), fg='gray', bg='#ffffff')

def writePname(event):
#-----------------------------------------------------------
# Callback function to mouse-events.
# Function to delete the default text "Patient name
# (Optional)\n" when focus is on the textbox
# CoMet_save_patientName, and insert the default text
# when focus is out of the textbox, unless the user
# has defined a valid value to the variable
# CoMet_save_patientName
#-----------------------------------------------------------
    current = CoMet_save_patientName.get("1.0", tk.END)
    if(current == "Patient name (Optional)\n"):
        CoMet_save_patientName.delete("1.0", tk.END)
    else:
        CoMet_save_patientName.insert("1.0", "Patient name (Optional)")

CoMet_save_patientName.bind("<FocusIn>", writePname)
CoMet_save_patientName.bind("<FocusOut>", writePname)

CoMet_correct_button_frame = tk.Frame(Globals.tab1_canvas)
CoMet_correct_button_frame.grid(row=6, column=0,rowspan=1, \
    padx = (150, 0), pady=(10,0), sticky=W)
Globals.tab1_canvas.grid_columnconfigure(9, weight=0)
Globals.tab1_canvas.grid_rowconfigure(9, weight=0)
CoMet_correct_button_frame.config(bg = '#ffffff')

CoMet_correct_button = tk.Button(CoMet_correct_button_frame, text='Correct', \
    image = CoMet_correct_button_image ,cursor='hand2',font=('calibri', '14'),\
        relief=FLAT, state=ACTIVE, command=CoMet_functions.Correct)
CoMet_correct_button.pack(expand=True, fill=BOTH)
CoMet_correct_button.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
CoMet_correct_button.image = CoMet_correct_button_image

Globals.CoMet_print_corrected_image = tk.Canvas(Globals.tab1_canvas)
Globals.CoMet_print_corrected_image.grid(row=0, column=2, rowspan=7, \
    sticky=N+W+S, pady=(20,0), padx=(0,150))
Globals.CoMet_print_corrected_image.config(bg='#ffffff', bd = 0, \
    relief=FLAT, highlightthickness=0)
Globals.tab1_canvas.grid_columnconfigure(11,weight=0)
Globals.tab1_canvas.grid_rowconfigure(11, weight=0)
Globals.CoMet_print_corrected_image.create_image(180,250,\
    image=CoMet_empty_image_image)
Globals.CoMet_print_corrected_image.image=CoMet_empty_image_image


def clearAll():
#-----------------------------------------------------------
# Function to reset every variable created in the
# tab CoMet. This will delete every value saved
# in variables related to CoMet
#------------------------------------------------
    #Clear out filename
    Globals.CoMet_uploaded_file_text = tk.Text(Globals.CoMet_border_1_label, \
        height=1, width=31)
    Globals.CoMet_uploaded_file_text.grid(row=0, column=0, columnspan=2, \
        sticky=E+W, pady=(20,20), padx=(100,0))
    Globals.CoMet_uploaded_file_text.insert(INSERT, \
        "Upload the image you want to correct")
    Globals.CoMet_uploaded_file_text.config(state=DISABLED, \
        bd=0, font=('calibri', '10'), fg='gray', bg='#ffffff')
    Globals.CoMet_uploaded_filename.set("Error!")

    #Clear out folder
    CoMet_save_to_folder = tk.Text(Globals.CoMet_border_2_label, \
        height=1, width=31)
    CoMet_save_to_folder.grid(row=0, column=0, columnspan=2, \
        sticky=E+W, pady=(20,20), padx=(100,0))
    CoMet_save_to_folder.insert(INSERT,"Folder to save the corrected image")
    CoMet_save_to_folder.config(state=DISABLED, bd=0, font=('calibri', '10'), \
        fg='gray', bg='#ffffff')
    Globals.CoMet_export_folder.set("Error!")

    #Clear filename of corrected file
    Globals.CoMet_save_filename = tk.Text(Globals.CoMet_border_3_label, \
        height=1, width=30)
    Globals.CoMet_save_filename.grid(row=0, column=0, columnspan=2, \
        sticky=E+W, pady=(20,20), padx=(100,0))
    Globals.CoMet_save_filename.insert(END,"Filename (will be saved as *.dcm)")
    Globals.CoMet_save_filename.config(state=NORMAL, bd=0, \
        font=('calibri', '10'), fg='gray', bg='#ffffff')
    Globals.CoMet_corrected_image_filename.set("Error!")
    Globals.CoMet_save_button_1.config(state=ACTIVE)


    def writeFilename(event):
#------------------------------------------------
# Callback function to mouse-events.
# Function to delete the default text "Filename (will
# be saved as *.dcm)\n" in the global textbox
# CoMet_save_filename when focus is on textbox and
# insert the default text as focus is out unless
# a filename has been written by the user.
#------------------------------------------------
        current = Globals.CoMet_save_filename.get("1.0", tk.END)
        if(current == "Filename (will be saved as *.dcm)\n"):
            Globals.CoMet_save_filename.delete("1.0", tk.END)
        else:
            Globals.CoMet_save_filename.insert("1.0", \
                "Filename (will be saved as *.dcm)")

    Globals.CoMet_save_filename.bind("<FocusIn>", writeFilename)
    Globals.CoMet_save_filename.bind("<FocusOut>", writeFilename)

    #Clear patientname
    CoMet_save_patientName = tk.Text(Globals.CoMet_border_4_label, \
        height=1, width=30)
    CoMet_save_patientName.grid(row=0, column=0, columnspan=2, \
        sticky=E+W, pady=(20,20), padx=(100,0))
    CoMet_save_patientName.insert(END,"Patient name (Optional)")
    CoMet_save_patientName.config(state=NORMAL, bd=0, \
        font=('calibri', '10'), fg='gray', bg='#ffffff')
    Globals.CoMet_patientName.set("Error!")
    CoMet_save_button_2.config(state=ACTIVE)

    def writePname(event):
#----------------------------------------------------------------
# Callback function to mouse-events.
# Function to delete the default text "Patient name
# (Optional)\n" when focus is on the textbox
# CoMet_save_patientName, and insert the default text
# when focus is out of the textbox, unless the user
# has defined a valid value to the variable
# CoMet_save_patientName
#----------------------------------------------------------------
        current = CoMet_save_patientName.get("1.0", tk.END)
        if(current == "Patient name (Optional)\n"):
            CoMet_save_patientName.delete("1.0", tk.END)
        else:
            CoMet_save_patientName.insert("1.0", "Patient name (Optional)")


    CoMet_save_patientName.bind("<FocusIn>", writePname)
    CoMet_save_patientName.bind("<FocusOut>", writePname)

    #Clear image
    Globals.CoMet_print_corrected_image.delete('all')
    Globals.CoMet_print_corrected_image.create_image\
        (123,148,image=CoMet_empty_image_image)
    Globals.CoMet_print_corrected_image.image = CoMet_empty_image_image
    #Clear progressbar
    Globals.CoMet_progressbar["value"]=0
    Globals.CoMet_progressbar_counter = 0
    Globals.CoMet_progressbar_check_file = True
    Globals.CoMet_progressbar_check_folder = True
    CoMet_progressbar_text = tk.Text(Globals.tab1_canvas, height=1, width=5)
    CoMet_progressbar_text.grid(row=1, column=0, columnspan=1, \
        sticky=E, padx=(0,158), pady=(0,36))
    CoMet_progressbar_text.insert(INSERT, "0%")
    CoMet_progressbar_text.config(state=DISABLED, bd=0, \
        relief=FLAT, bg='#ffffff',font=('calibri', '10', 'bold'))


CoMet_clear_all_button_frame = tk.Frame(Globals.tab1_canvas)
CoMet_clear_all_button_frame.grid(row=6, column=0, rowspan=1, \
    padx=(350,0), pady=(10,0), sticky=W)
Globals.tab1_canvas.grid_columnconfigure(13, weight=0)
Globals.tab1_canvas.grid_rowconfigure(13, weight=0)
CoMet_clear_all_button_frame.config(bg='#ffffff')

CoMet_clear_all_button = tk.Button(CoMet_clear_all_button_frame, text="Clear all",\
    image=dose_response_clear_all_button_image, cursor='hand2', \
        font=('calibri', '14'), relief=FLAT, state=ACTIVE, command=clearAll)
CoMet_clear_all_button.pack(expand=True, fill=BOTH)
CoMet_clear_all_button.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
CoMet_clear_all_button.image=dose_response_clear_all_button_image

Globals.tab1_canvas.pack(expand=True, fill=BOTH)


########################################                            ####################################
######################################## TAB 3 -- Dose Response ####################################
########################################                            ####################################

#--------------------------------------------------------------------------------------------
# Code to place all widgets related to the tab "Dose Response" in Fidora.
# They are all placed in the global canvas tab2_canvas defines in
# the file Globals.py. Whenever a function is called the function
# is written in the file Dose_Response_Function.
#--------------------------------------------------------------------------------------------

Globals.tab2_canvas.config(bg='#ffffff', bd = 0, relief=FLAT, highlightthickness=0)


dose_response_explain_text = tk.Text(Globals.tab2_canvas, height=4, width=140)
```

```
dose_response_explain_text.insert(INSERT, "\
Upload the scanned *.tif files (there should be at least 3 of each dose level) \
and save. The dose response curve along with the equation will appear when \n\
enough data points are given. The uploaded files must have dpi setting 72 or 127. \
When saving the calibration the dose response data will be saved and can be \
\nchosen for later use of this software. The dose response curve will be found \
for all three color channels, but can be removed using the check boxes. A dose \
\nresponse equation will only be fitted for the red channel.  " )
dose_response_explain_text.grid(row=0, column=0, columnspan=5, \
    sticky=N+S+E+W, pady=(20,20), padx=(20,10))
Globals.tab2_canvas.grid_columnconfigure(0, weight=0)
Globals.tab2_canvas.grid_rowconfigure(0, weight=0)
dose_response_explain_text.config(state=DISABLED, \
    font=('calibri', '11'), bg ='#ffffff', relief=FLAT)

dose_response_upload_button_frame = tk.Frame(Globals.tab2_canvas_files)
dose_response_upload_button_frame.grid(row=0, column = 0, \
    columnspan=8, padx = (60, 0), pady=(10,5))
Globals.tab2_canvas_files.grid_columnconfigure(0, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(0, weight=0)
dose_response_upload_button_frame.config(bg = '#ffffff')

dose_response_upload_button = \
    tk.Button(dose_response_upload_button_frame, text='Upload file', \
    image=Globals.upload_button_image,cursor='hand2', font=('calibri', '14'), \
        relief=FLAT, state=ACTIVE, command=Dose_response_functions.create_window)
dose_response_upload_button.pack(expand=True, fill=BOTH)
dose_response_upload_button.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
dose_response_upload_button.image = Globals.upload_button_image

check1 = Checkbutton(Globals.tab2_canvas_files, variable=Globals.dose_response_var1,\
    command=Dose_response_functions.plot_dose_response)
check1.grid(row=1, column=1, sticky=E, padx=(30,15))
Globals.tab2_canvas_files.grid_columnconfigure(5, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(5, weight=0)
check1.config(bg='#ffffff')

check2 = Checkbutton(Globals.tab2_canvas_files, \
    variable=Globals.dose_response_var2, \
        command=Dose_response_functions.plot_dose_response)
check2.grid(row=1, column=3, sticky=E, padx=(45,15))
Globals.tab2_canvas_files.grid_columnconfigure(6, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(6, weight=0)
check2.config(bg='#ffffff')
```

```
check3 = Checkbutton(Globals.tab2_canvas_files, \
    variable=Globals.dose_response_var3, \
        command=Dose_response_functions.plot_dose_response)
check3.grid(row=1, column=5, sticky=E, padx=(35,10))
Globals.tab2_canvas_files.grid_columnconfigure(7, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(7, weight=0)
check3.config(bg='#ffffff')

red = tk.Text(Globals.tab2_canvas_files, height=1, width=4)
red.insert(INSERT, "Red")
red.grid(row=1, column=1, sticky=W, padx=(0,0))
Globals.tab2_canvas_files.grid_columnconfigure(1, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(1, weight=0)
red.config(state=DISABLED, bd=0, font=('calibri', '12'))

green = tk.Text(Globals.tab2_canvas_files, height=1, width=5)
green.insert(INSERT, "Green")
green.grid(row = 1, column = 3, sticky=W, padx=(0,0))
Globals.tab2_canvas_files.grid_columnconfigure(2, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(2, weight=0)
green.config(state=DISABLED, bd=0, font=('calibri', '12'))

blue = tk.Text(Globals.tab2_canvas_files, height=1, width=4)
blue.insert(INSERT, "Blue")
blue.grid(row=1, column=5, sticky=W, padx=(0,0))
Globals.tab2_canvas_files.grid_columnconfigure(3, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(3, weight=0)
blue.config(state=DISABLED, bd=0, font=('calibri', '12'))

dose_title = tk.Text(Globals.tab2_canvas_files, height=1, width=10)
dose_title.insert(INSERT, "Dose (cGy)")
dose_title.grid(row=1, column=0, sticky=N+S+W+E, padx=(0,15))
Globals.tab2_canvas_files.grid_columnconfigure(4, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(4, weight=0)
dose_title.config(state=DISABLED, bd=0, font=('calibri', '12'))

Globals.upload_files_here_canvas = tk.Canvas(Globals.tab2_canvas_files)
Globals.upload_files_here_canvas.grid(row=3, column=0, \
    columnspan=13, sticky=N+S+W+E)
Globals.tab2_canvas_files.grid_columnconfigure(50, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(50, weight=0)
Globals.upload_files_here_canvas.config(relief=FLAT, bd=0, \
    bg='#ffffff',highlightthickness=0)
Globals.upload_files_here_canvas.create_image(70,30,\
    image=Globals.dose_response_upload_files_here, anchor='nw')

Globals.dose_response_equation_image = \
    tk.Canvas(Globals.dose_response_equation_frame, width=650)
Globals.dose_response_equation_image.grid(row=0, column=0, sticky=N+S+W+E)
Globals.dose_response_equation_frame.grid_columnconfigure(50, weight=0)
Globals.dose_response_equation_frame.grid_rowconfigure(50, weight=0)
Globals.dose_response_equation_image.config(bg='#ffffff', \
    relief=FLAT, highlightthickness=0)
Globals.dose_response_equation_image.create_image(170,0, \
    image=Globals.dose_response_equation_written_here, anchor='nw')

dose_response_save_calibration_button_frame = tk.Frame(Globals.tab2_canvas)
dose_response_save_calibration_button_frame.grid(row=3, column = 1, \
    sticky=N+S+E+W, padx=(0,0), pady=(0,0))
Globals.tab2_canvas.grid_columnconfigure(10, weight=0)
Globals.tab2_canvas.grid_rowconfigure(10, weight=0)
dose_response_save_calibration_button_frame.config\
    (bg = '#ffffff', height=1, width=100)
dose_response_save_calibration_button_frame.grid_propagate(0)

Globals.dose_response_save_calibration_button = \
    tk.Button(dose_response_save_calibration_button_frame, text='Save calibration', \
        image=dose_response_calibration_button_image, \
            cursor='hand2', font=('calibri', '12'),relief=FLAT, state=DISABLED, \
                command=Dose_response_functions.saveCalibration)
Globals.dose_response_save_calibration_button.pack(expand=True, fill=BOTH, side=TOP)
Globals.dose_response_save_calibration_button.config(bg='#ffffff', \
    activebackground='#ffffff', activeforeground='#ffffff', highlightthickness=0)
Globals.dose_response_save_calibration_button.image = \
    dose_response_calibration_button_image

dose_response_clear_all_button_frame = tk.Frame(Globals.tab2_canvas)
dose_response_clear_all_button_frame.grid(row=3, column=0, \
    sticky=N+S+E+W, padx=(30,0), pady=(0,0))
Globals.tab2_canvas.grid_columnconfigure(11, weight=0)
Globals.tab2_canvas.grid_rowconfigure(11, weight=0)
dose_response_clear_all_button_frame.config(bg='#ffffff', height=1, width=100)
dose_response_clear_all_button_frame.grid_propagate(0)

dose_response_clear_all_button = tk.Button(dose_response_clear_all_button_frame, \
    text='Clear all', image=dose_response_clear_all_button_image, \
        cursor='hand2', font=('calibri', '12'), relief=FLAT, state=ACTIVE, \
            command=Dose_response_functions.clear_all)
dose_response_clear_all_button.pack(expand=True, fill=BOTH, side=TOP)
dose_response_clear_all_button.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
dose_response_clear_all_button.image = dose_response_clear_all_button_image

delete_text = tk.Text(Globals.tab2_canvas_files, height=1, widt=7)
delete_text.insert(INSERT, "Delete")
delete_text.grid(row=1, column=7, sticky=N+S+E+W, padx=(0,0))
Globals.tab2_canvas_files.grid_columnconfigure(4, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure(4, weight=0)
delete_text.config(state=DISABLED, bd=0, font=('calibri', '12'))

Globals.tab2_canvas.pack(expand=True, fill=BOTH)



##########################################                    ###################################
########################################## TAB 4 -- Profiles ###################################
##########################################                    ###################################
#----------------------------------------------------------------------------------------------
# Code to place all widgets related to the tab "Profiles" in Fidora
# They are all placed in the global canvas tab4_canvas defines
# in the file Globals.py.
# Whereever a function is called the function is written in
# the file Profiles_functions.py.
#----------------------------------------------------------------------------------------------

Globals.tab4_canvas.config(bg='#ffffff', bd = 0, relief=FLAT, highlightthickness=0)

profiles_explain_text = tk.Text(Globals.tab4_canvas, height=4, width=140)
profiles_explain_text.insert(INSERT, "\
Upload a scanned image of film, along with the RT Plan and doseplan files from the \
corresponding doseplan and investigate the profiles. There are three \npossible \
profiles, horizontal, vertical and manually drawn profile.To make up for any \
positional error when scanning the film, or marking the \nisocenter/reference \
point, it is possible to make adjustments to the placement of the ROI in the film. \
This is done using the arrow buttons above the plot." )
profiles_explain_text.grid(row=0, column=0, columnspan=3, \
    sticky=N+S+E+W, pady=(20,20), padx=(20,10))
Globals.tab4_canvas.grid_columnconfigure(0, weight=0)
Globals.tab4_canvas.grid_rowconfigure(0, weight=0)
profiles_explain_text.config(state=DISABLED, \
    font=('calibri', '11'), bg ='#ffffff', relief=FLAT)
profiles_explain_text.grid_propagate(0)

profiles_upload_film_frame = tk.Frame(Globals.tab4_canvas)
```

```
profiles_upload_film_frame.grid(row=3, column = 0, sticky=N+S+W+E)
Globals.tab4_canvas.grid_columnconfigure(1, weight=0)
Globals.tab4_canvas.grid_rowconfigure(1, weight=0)
profiles_upload_film_frame.config(bg = '#ffffff', height=1, width=1)
profiles_upload_film_frame.grid_propagate(0)

Globals.profiles_upload_button_film = \
    tk.Button(profiles_upload_film_frame, text='Browse',\
    image = profiles_add_film_button_image, cursor='hand2',font=('calibri', '14'), \
        relief=FLAT, state=ACTIVE, command=Profile_functions.UploadFilm)
Globals.profiles_upload_button_film.pack(expand=True, fill=BOTH)
Globals.profiles_upload_button_film.config(bg='#ffffff', activebackground='#ffffff',\
    activeforeground='#ffffff', highlightthickness=0)
Globals.profiles_upload_button_film.image = profiles_add_film_button_image

profiles_upload_doseplan_frame = tk.Frame(Globals.tab4_canvas)
profiles_upload_doseplan_frame.grid(row=5, column = 0, sticky=N+S+E+W)
Globals.tab4_canvas.grid_columnconfigure(3, weight=0)
Globals.tab4_canvas.grid_rowconfigure(3, weight=0)
profiles_upload_doseplan_frame.config(bg = '#ffffff', height=1, width=1)
profiles_upload_doseplan_frame.grid_propagate(0)

Globals.profiles_upload_button_doseplan = \
    tk.Button(profiles_upload_doseplan_frame, text='Browse',\
    image=Globals.profiles_add_doseplan_button_image, cursor='hand2', \
        font=('calibri', '14'), relief=FLAT, state=DISABLED, \
            command=Profile_functions.UploadDoseplan_button_function)
Globals.profiles_upload_button_doseplan.pack(expand=True, fill=BOTH)
Globals.profiles_upload_button_doseplan.configure(bg='#ffffff', \
    activebackground='#ffffff', activeforeground='#ffffff', highlightthickness=0)
Globals.profiles_upload_button_doseplan.image = \
    Globals.profiles_add_doseplan_button_image

profiles_upload_rtplan_frame = tk.Frame(Globals.tab4_canvas)
profiles_upload_rtplan_frame.grid(row=4, column=0, sticky=N+S+W+E)
Globals.tab4_canvas.grid_columnconfigure(10, weight=0)
Globals.tab4_canvas.grid_rowconfigure(10, weight=0)
profiles_upload_rtplan_frame.config(bg='#ffffff', height=1, width=1)
profiles_upload_rtplan_frame.grid_propagate(0)

Globals.profiles_upload_button_rtplan = \
    tk.Button(profiles_upload_rtplan_frame, text='Browse',\
    image=profiles_add_rtplan_button_image,cursor='hand2', font=('calibri', '14'), \
        relief=FLAT, state=DISABLED, command=Profile_functions.UploadRTplan)
Globals.profiles_upload_button_rtplan.pack(expand=True, fill=BOTH)
Globals.profiles_upload_button_rtplan.configure(bg='#ffffff', \
    activebackground='#ffffff', activeforeground='#ffffff', highlightthickness=0)
Globals.profiles_upload_button_rtplan.image=profiles_add_rtplan_button_image

profiles_film_orientation_frame = tk.Frame(Globals.tab4_canvas)
profiles_film_orientation_frame.grid(row=1, column=0, sticky=N+S+W+E)
profiles_film_orientation_frame.config(relief=FLAT, highlightthickness=0, \
    bd=0, bg='#ffffff', height=1, width=1)
Globals.tab4_canvas.grid_columnconfigure(2, weight=0)
Globals.tab4_canvas.grid_rowconfigure(2, weight=0)
profiles_film_orientation_frame.grid_propagate(0)

film_orientation_menu_text = tk.Text(profiles_film_orientation_frame, \
    width=14, height=1)
film_orientation_menu_text.insert(INSERT, "Film orientation:")
film_orientation_menu_text.config(state=DISABLED, \
    font=('calibri', '10'), bd = 0, relief=FLAT)
film_orientation_menu_text.pack(side=LEFT)

Globals.profiles_film_orientation_menu = OptionMenu(profiles_film_orientation_frame,\
    Globals.profiles_film_orientation, 'Axial', 'Coronal', 'Sagittal')
Globals.profiles_film_orientation_menu.pack(side=LEFT)
Globals.profiles_film_orientation_menu.config(bg = '#ffffff', width=15, relief=FLAT)

profiles_film_orientation_help_frame = tk.Frame(profiles_film_orientation_frame)
profiles_film_orientation_help_frame.pack(side=LEFT)
profiles_film_orientation_help_frame.configure(bg='#ffffff')

profiles_help_button_orientation = \
    tk.Button(profiles_film_orientation_help_frame, text='help',\
        image=Globals.help_button, cursor='hand2', font=('calibri', '14'), \
            relief=FLAT, state=ACTIVE, command=Profile_functions.help_showPlanes)
profiles_help_button_orientation.pack(expand=True, fill=BOTH)
profiles_help_button_orientation.configure(bg='#ffffff',activebackground='#ffffff',\
    activeforeground='#ffffff', highlightthickness=0)
profiles_help_button_orientation.image=Globals.help_button

profiles_film_factor_frame = tk.Frame(Globals.tab4_canvas)
profiles_film_factor_frame.grid(row=2, column=0, sticky=N+S+W+E)
Globals.tab4_canvas.grid_columnconfigure(30, weight=0)
Globals.tab4_canvas.grid_rowconfigure(30, weight=0)
profiles_film_factor_frame.config(relief=FLAT, \
    highlightthickness=0, bd=0, bg='#ffffff', height=1, width=1)
profiles_film_factor_frame.grid_propagate(0)
```

```
profiles_film_factor = tk.Text(profiles_film_factor_frame, width=20, height=2)
profiles_film_factor.insert(INSERT, "Film factor \n(number of fractions):")
profiles_film_factor.config(state=DISABLED, \
    font=('calibri', '10'), bd = 0, relief=FLAT)
profiles_film_factor.pack(side=LEFT)

Globals.profiles_film_factor_input = \
    tk.Text(profiles_film_factor_frame, width=8, height=1)
Globals.profiles_film_factor_input.pack(side=LEFT)
Globals.profiles_film_factor_input.insert(INSERT, " ")
Globals.profiles_film_factor_input.config(state=NORMAL, \
    font=('calibri', '10'), bd = 2, bg='#ffffff')
Globals.tab4_canvas.grid_columnconfigure(31, weight=0)
Globals.tab4_canvas.grid_rowconfigure(31, weight=0)

profiles_resetAll_frame = tk.Frame(Globals.tab4_canvas)
profiles_resetAll_frame.grid(row=7,column=0, sticky=N+S+W+E)
Globals.tab4_canvas.grid_columnconfigure(5, weight=0)
Globals.tab4_canvas.grid_rowconfigure(5, weight=0)
profiles_resetAll_frame.config(bg='#ffffff', height=1, width=1)
profiles_resetAll_frame.grid_propagate(0)

profiles_resetAll_button = tk.Button(profiles_resetAll_frame, text='Reset', \
    image=dose_response_clear_all_button_image, cursor='hand2', \
        font=('calibri', '14'),relief=FLAT, state=ACTIVE, \
            command=Profile_functions.clearAll)
profiles_resetAll_button.pack(expand=True, fill=BOTH)
profiles_resetAll_button.configure(bg='#ffffff', activebackground='#ffffff', \
activeforeground='#ffffff', highlightthickness=0)
profiles_resetAll_button.image = dose_response_clear_all_button_image

Globals.profiles_adjust_button_left = \
    tk.Button(Globals.profiles_redefine_film_ROI_frame, \
    text="left", image=Globals.adjust_button_left_image,cursor='hand2', \
        font=('calibri', '12'), relief=FLAT, state=DISABLED, command=lambda: \
            Profile_functions.adjustROILeft\
                (Globals.profiles_choice_of_profile_line_type.get()))
Globals.profiles_adjust_button_left.pack(side=LEFT)
Globals.profiles_adjust_button_left.config(bg='#ffffff', activebackground='#ffffff',\
    activeforeground='#ffffff', highlightthickness=0)
Globals.profiles_adjust_button_left.image = Globals.adjust_button_left_image

Globals.profiles_adjust_button_up = \
    tk.Button(Globals.profiles_redefine_film_ROI_frame, \
        text="left", image=Globals.adjust_button_up_image,cursor='hand2', \
            font=('calibri', '12'), relief=FLAT, state=DISABLED, command=lambda: \
                Profile_functions.adjustROIUp\
                    (Globals.profiles_choice_of_profile_line_type.get()))
Globals.profiles_adjust_button_up.pack(side=LEFT)
Globals.profiles_adjust_button_up.config(bg='#ffffff', activebackground='#ffffff',\
    activeforeground='#ffffff', highlightthickness=0)
Globals.profiles_adjust_button_up.image = Globals.adjust_button_up_image

Globals.profiles_adjust_button_down = \
    tk.Button(Globals.profiles_redefine_film_ROI_frame, \
    text="left", image=Globals.adjust_button_down_image,cursor='hand2', \
        font=('calibri', '12'), relief=FLAT, state=DISABLED, command=lambda: \
            Profile_functions.adjustROIDown\
                (Globals.profiles_choice_of_profile_line_type.get()))
Globals.profiles_adjust_button_down.pack(side=LEFT)
Globals.profiles_adjust_button_down.config(bg='#ffffff', activebackground='#ffffff',\
    activeforeground='#ffffff', highlightthickness=0)
Globals.profiles_adjust_button_down.image = Globals.adjust_button_down_image

Globals.profiles_adjust_button_right = \
    tk.Button(Globals.profiles_redefine_film_ROI_frame, text="left", \
        image=Globals.adjust_button_right_image,cursor='hand2', \
            font=('calibri', '12'),relief=FLAT, state=DISABLED, command=lambda: \
                Profile_functions.adjustROIRight\
                    (Globals.profiles_choice_of_profile_line_type.get()))
Globals.profiles_adjust_button_right.pack(side=LEFT)
Globals.profiles_adjust_button_right.config(bg='#ffffff',activebackground='#ffffff',\
    activeforeground='#ffffff', highlightthickness=0)
Globals.profiles_adjust_button_right.image = Globals.adjust_button_right_image

Globals.profiles_adjust_button_return = \
    tk.Button(Globals.profiles_redefine_film_ROI_frame, \
    text="Original",cursor='hand2', font=('calibri', '12'), relief=FLAT, \
        state=DISABLED, command=lambda:Profile_functions.\
            returnToOriginalROICoordinates\
                (Globals.profiles_choice_of_profile_line_type.get()))
Globals.profiles_adjust_button_return.pack(side=LEFT)
Globals.profiles_adjust_button_return.config(bg='#ffffff',\
    activebackground='#ffffff',activeforeground='#ffffff', highlightthickness=0)

profiles_export_plot_frame= tk.Frame(Globals.tab4_canvas)
profiles_export_plot_frame.grid(row=6,column=0, sticky=N+S+W+E)
Globals.tab4_canvas.grid_columnconfigure(20, weight=0)
Globals.tab4_canvas.grid_rowconfigure(20, weight=0)
profiles_export_plot_frame.config(bg='#ffffff', height=1, width=1)
profiles_export_plot_frame.grid_propagate(0)

Globals.profiles_export_plot_button = \
    tk.Button(profiles_export_plot_frame, text = "Export plot", \
        image=Globals.export_plot_button_image, cursor='hand2', \
            font=('calibri', '12'), relief=FLAT, state=DISABLED, \
                command=Profile_functions.nothing_function)
Globals.profiles_export_plot_button.pack()
Globals.profiles_export_plot_button.config(bg='#ffffff',activebackground='#ffffff',\
    activeforeground='#ffffff', highlightthickness=0)
Globals.profiles_export_plot_button.image = Globals.export_plot_button_image



Globals.profiles_choice_of_profile_line_type.trace_add('write', \
    Profile_functions.trace_profileLineType)



Globals.tab4_canvas.pack(expand=True, fill=BOTH)

##################################          ########################################
################################## Tab 5 DVH ########################################
##################################          ########################################
#-----------------------------------------------------------------------------------
# Code to place all widgets related to the tab 'DVH' in Fidora.
# They are all places in the global canvas tab5_canvas defined
# in the file Globals.py.
# Whereever a function is called the function is written in the
# file DVH_functions.py
#-----------------------------------------------------------------------------------

Globals.tab5_canvas.config(bg='#ffffff', bd = 0, relief=FLAT, highlightthickness=0)

DVH_explain_text = tk.Text(Globals.tab5_canvas, height=4, width=140)
DVH_explain_text.insert(INSERT, "\
Upload the scanned images of film, along with RT plan, stucture and doseplan \
files and study the dose volume histogram \nof the different defined volumes. \
By un-checking the buttons for each volume it is possible to hide the volume \n\
in the plot, making it easier to study each of them separately.")
DVH_explain_text.grid(row=0, column=0, columnspan=7, \
    sticky=N+S+E+W, pady=(20,20), padx=(20,10))
Globals.tab5_canvas.grid_columnconfigure(0, weight=0)
Globals.tab5_canvas.grid_rowconfigure(0, weight=0)
DVH_explain_text.config(state=DISABLED, \
    font=('calibri', '11'), bg ='#ffffff', relief=FLAT)
```

```
DVH_upload_film_frame = tk.Frame(Globals.tab5_canvas)
DVH_upload_film_frame.grid(row=3,column=0,padx=(50,40),pady=(10,20),sticky=N+S+W)
Globals.tab5_canvas.grid_columnconfigure(1, weight=0)
Globals.tab5_canvas.grid_rowconfigure(1, weight=0)
DVH_upload_film_frame.config(bg = '#ffffff')

Globals.DVH_upload_button_film = tk.Button(DVH_upload_film_frame, text='Browse', \
    image = profiles_add_film_button_image, cursor='hand2',font=('calibri', '14'),\
        relief=FLAT, state=ACTIVE, command=DVH_functions.UploadFilm)
Globals.DVH_upload_button_film.pack(expand=True, fill=BOTH)
Globals.DVH_upload_button_film.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
Globals.DVH_upload_button_film.image = profiles_add_film_button_image

DVH_upload_doseplan_frame = tk.Frame(Globals.tab5_canvas)
DVH_upload_doseplan_frame.grid(row=4,column=0,\
    padx=(210,0),sticky=N+S+W,pady=(10,20))
Globals.tab5_canvas.grid_columnconfigure(3, weight=0)
Globals.tab5_canvas.grid_rowconfigure(3, weight=0)
DVH_upload_film_frame.config(bg = '#ffffff')

Globals.DVH_upload_button_doseplan = \
    tk.Button(DVH_upload_doseplan_frame, text='Browse',\
    image=Globals.profiles_add_doseplan_button_image,cursor='hand2', \
        font=('calibri', '14'),relief=FLAT, state=DISABLED, \
            command=DVH_functions.UploadDoseplan_button_function)
Globals.DVH_upload_button_doseplan.pack(expand=True, fill=BOTH)
Globals.DVH_upload_button_doseplan.configure(bg='#ffffff', \
    activebackground='#ffffff',activeforeground='#ffffff', highlightthickness=0)
Globals.DVH_upload_button_doseplan.image = \
    Globals.profiles_add_doseplan_button_image

DVH_upload_rtplan_frame = tk.Frame(Globals.tab5_canvas)
DVH_upload_rtplan_frame.grid(row=3, column=0, \
    padx=(210,0), sticky=N+S+W, pady=(10,20))
Globals.tab5_canvas.grid_columnconfigure(10, weight=0)
Globals.tab5_canvas.grid_rowconfigure(10, weight=0)
DVH_upload_rtplan_frame.config(bg='#ffffff')

Globals.DVH_upload_button_rtplan = tk.Button(DVH_upload_rtplan_frame, text='Browse',\
    image=profiles_add_rtplan_button_image,cursor='hand2', font=('calibri', '14'),\
        relief=FLAT, state=DISABLED, command=DVH_functions.UploadRTplan)
Globals.DVH_upload_button_rtplan.pack(expand=True, fill=BOTH)
Globals.DVH_upload_button_rtplan.configure(bg='#ffffff', activebackground='#ffffff',\
activeforeground='#ffffff', highlightthickness=0)
```

```
Globals.DVH_upload_button_rtplan.image=profiles_add_rtplan_button_image

DVH_upload_struct_frame = tk.Frame(Globals.tab5_canvas)
DVH_upload_struct_frame.grid(row=4, column=0, \
    padx=(50,40), sticky=N+S+W, pady=(10,20))
Globals.tab5_canvas.grid_columnconfigure(32, weight=0)
Globals.tab5_canvas.grid_rowconfigure(32, weight=0)
DVH_upload_struct_frame.config(bg='#ffffff')

Globals.DVH_upload_button_struct = tk.Button(DVH_upload_struct_frame, \
    text='Upload struct', image=DVH_add_structure_button_image,\
        cursor='hand2', font=('calibri', '14'), \
            relief=FLAT, state=DISABLED, command=DVH_functions.UploadStruct)
Globals.DVH_upload_button_struct.pack(expand=True, fill=BOTH)
Globals.DVH_upload_button_struct.configure(bg='#ffffff', \
    activebackground='#ffffff', activeforeground='#ffffff', highlightthickness=0)
Globals.DVH_upload_button_struct.image = DVH_add_structure_button_image

film_orientation_frame = tk.Frame(Globals.tab5_canvas)
film_orientation_frame.grid(row=1, column=0, sticky=N+S+E+W, \
    pady=(10,20), padx=(40,30))
Globals.tab5_canvas.grid_columnconfigure(2, weight=0)
Globals.tab5_canvas.grid_rowconfigure(2, weight=0)
film_orientation_frame.config(bg='#ffffff', bd=0, highlightthickness=0, relief=FLAT)

film_orientation_menu_text = tk.Text(film_orientation_frame, width=14, height=1)
film_orientation_menu_text.insert(INSERT, "Film orientation:")
film_orientation_menu_text.config(state=DISABLED, \
    font=('calibri', '10'), bd = 0, relief=FLAT)
film_orientation_menu_text.pack(side=LEFT)

Globals.DVH_film_orientation_menu = OptionMenu(film_orientation_frame, \
    Globals.DVH_film_orientation, 'Axial', 'Coronal', 'Sagittal')
Globals.DVH_film_orientation_menu.pack(side=LEFT)
Globals.DVH_film_orientation_menu.config(bg = '#ffffff', width=15, relief=FLAT)

DVH_film_orientation_help_frame = tk.Frame(film_orientation_frame)
DVH_film_orientation_help_frame.pack(side=LEFT)
DVH_film_orientation_help_frame.configure(bg='#ffffff')

DVH_help_button_orientation = \
    tk.Button(DVH_film_orientation_help_frame, text='help', \
    image=Globals.help_button, cursor='hand2', font=('calibri', '14'), \
        relief=FLAT, state=ACTIVE, command=DVH_functions.help_showPlanes)
DVH_help_button_orientation.pack(expand=True, fill=BOTH)
DVH_help_button_orientation.configure(bg='#ffffff',activebackground='#ffffff',\
    activeforeground='#ffffff', highlightthickness=0)
DVH_help_button_orientation.image=Globals.help_button

DVH_film_factor_frame = tk.Frame(Globals.tab5_canvas)
DVH_film_factor_frame.grid(row=2, column=0, sticky=N+S+E+W, \
    pady=(10,20), padx=(40,30))
Globals.tab5_canvas.grid_columnconfigure(30, weight=0)
Globals.tab5_canvas.grid_rowconfigure(30, weight=0)
DVH_film_factor_frame.config(bg='#ffffff', bd=0, highlightthickness=0, relief=FLAT)

DVH_film_factor = tk.Text(DVH_film_factor_frame, width=20, height=2)
DVH_film_factor.insert(INSERT, "Film factor \n(number of fractions):")
DVH_film_factor.config(state=DISABLED, font=('calibri', '10'), bd = 0, relief=FLAT)
DVH_film_factor.pack(side=LEFT)

Globals.DVH_film_factor_input = tk.Text(DVH_film_factor_frame, width=8, height=1)
Globals.DVH_film_factor_input.pack(side=LEFT)
Globals.DVH_film_factor_input.insert(INSERT, " ")
Globals.DVH_film_factor_input.config(state=NORMAL, \
    font=('calibri', '10'), bd = 2, bg='#ffffff')

DVH_resetAll_frame = tk.Frame(Globals.tab5_canvas)
DVH_resetAll_frame.grid(row=5,column=0, padx=(50,40), sticky=S+N+W, pady=(10,20))
Globals.tab5_canvas.grid_columnconfigure(5, weight=0)
Globals.tab5_canvas.grid_rowconfigure(5, weight=0)
profiles_resetAll_frame.config(bg='#ffffff')

DVH_resetAll_button = tk.Button(DVH_resetAll_frame, text='Reset', \
    image=dose_response_clear_all_button_image, \
    cursor='hand2', font=('calibri', '14'), relief=FLAT, state=ACTIVE, \
        command=DVH_functions.clearAll)
DVH_resetAll_button.pack(expand=True, fill=BOTH)
DVH_resetAll_button.configure(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
DVH_resetAll_button.image = dose_response_clear_all_button_image

DVH_export_frame = tk.Frame(Globals.tab5_canvas)
DVH_export_frame.grid(row=5,column=0, padx=(210,0), sticky=S+N+W, pady=(10,20))
Globals.tab5_canvas.grid_columnconfigure(50, weight=0)
Globals.tab5_canvas.grid_rowconfigure(50, weight=0)
DVH_export_frame.config(bg='#ffffff')

Globals.DVH_export_button = tk.Button(DVH_export_frame, text='Reset', \
    image=Globals.export_plot_button_image, \
    cursor='hand2', font=('calibri', '14'), relief=FLAT, state=DISABLED, \
        command=DVH_functions.nothingButton)
Globals.DVH_export_button.pack(expand=True, fill=BOTH)
Globals.DVH_export_button.configure(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
Globals.DVH_export_button.image = Globals.export_plot_button_image

Globals.temp_image_canvas = tk.Canvas(Globals.DVH_view_film_doseplan_ROI)
Globals.temp_image_canvas.grid(row=0, column=0, sticky=N+S+W+E)
Globals.temp_image_canvas.config(bg='#ffffff', bd=0, \
    highlightthickness=0,relief=FLAT)
Globals.temp_image_canvas.create_image(270,150, image=Globals.dVH_temp_image)



Globals.tab5_canvas.pack(expand=True, fill=BOTH)


############################################ End statement ####################################
Globals.form.mainloop()
```

Globals.py

```
#---------------------------------------------------------------------------------------------
#
# Globals.py
# version 26.07.20
#
# File to create global variables that can be used in several files
# related to Fidora
#
#---------------------------------------------------------------------------------------------


import tkinter as tk
from tkinter import ttk, StringVar, IntVar, Scrollbar, RIGHT, Y, \
    HORIZONTAL, E, W, N, S, BOTH, Frame, Canvas, LEFT, FLAT, INSERT, DISABLED, ALL,\
        X, BOTTOM, DoubleVar, PanedWindow, RAISED, TOP, Radiobutton, \
            CENTER, BooleanVar
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

```
#----------------------------------------------------------------------
# Create global variables that will be defined as images
#----------------------------------------------------------------------
global upload_button_image
global dose_response_dose_border
global save_button
global help_button
global done_button_image
global profiles_add_doseplan_button_image
global profiles_add_doseplans_button_image
global adjust_button_left_image
global adjust_button_right_image
global adjust_button_up_image
global adjust_button_down_image
global dose_response_upload_files_here
global dose_response_equation_written_here
global export_plot_button_image
global dVH_temp_image


#----------------------------------------------------------------------
# Create the main window form, in which a frame is placed.
# Scrollbar is defined in main window
#----------------------------------------------------------------------
global form
form = tk.Tk()

over_all_frame = tk.Frame(form, bd=0, relief=FLAT)
over_all_canvas = Canvas(over_all_frame)

xscrollbar = Scrollbar(over_all_frame, orient=HORIZONTAL, \
    command=over_all_canvas.xview)
yscrollbar = Scrollbar(over_all_frame, command=over_all_canvas.yview)

scroll_frame = ttk.Frame(over_all_canvas)
scroll_frame.bind("<Configure>", \
    lambda e: over_all_canvas.configure(scrollregion=over_all_canvas.bbox('all')))

over_all_canvas.create_window((0,0), window=scroll_frame, anchor='nw')
over_all_canvas.configure(xscrollcommand=xscrollbar.set, \
    yscrollcommand=yscrollbar.set)

over_all_frame.config(highlightthickness=0, bg='#ffffff')
over_all_canvas.config(highlightthickness=0, bg='#ffffff')
over_all_frame.pack(expand=True, fill=BOTH)
```

```
over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
over_all_frame.grid_columnconfigure(0, weight=1)
over_all_frame.grid_rowconfigure(0, weight=1)
xscrollbar.grid(row=1, column=0, sticky=E+W)
over_all_frame.grid_columnconfigure(1, weight=0)
over_all_frame.grid_rowconfigure(1, weight=0)
yscrollbar.grid(row=0, column=1, sticky=N+S)
over_all_frame.grid_columnconfigure(2, weight=0)
over_all_frame.grid_rowconfigure(2, weight=0)

#------------------------------------------------------------------------
# Create the Notebook (tab_parent) which holds the different tabs in Fidora
# Then each tab is is created as a frame. In each frame the global canvases
# are placed holding all widgets placed in notebook.py
#------------------------------------------------------------------------
global tab_parent
tab_parent = ttk.Notebook(scroll_frame)
tab_parent.borderWidth=0
tab_parent.grid(row=1, column=0, sticky=E+W+N+S, pady=(0,0), padx =(0,0))

global intro_tab
intro_tab = ttk.Frame(tab_parent)
intro_tab.config(relief=FLAT)
global tab1
tab1 = ttk.Frame(tab_parent)
global tab2
tab2 = ttk.Frame(tab_parent)
global tab3
tab3 = ttk.Frame(tab_parent)
global tab4
tab4 = ttk.Frame(tab_parent)
global tab5
tab5 = ttk.Frame(tab_parent)

global tab1_canvas
tab1_canvas = tk.Canvas(tab1)
global tab2_canvas
tab2_canvas = tk.Canvas(tab2)
global tab3_canvas
tab3_canvas = tk.Canvas(tab3)
global tab4_canvas
tab4_canvas = tk.Canvas(tab4)
global tab5_canvas
tab5_canvas= tk.Canvas(tab5)
```

```
#----------------------------------------------------------------------------
# All variables defined here are related to the tab CoMet and will be used in
# both notebook.py and CoMet_functions.py
#----------------------------------------------------------------------------
global CoMet_progressbar
CoMet_progressbar = ttk.Progressbar(tab1_canvas,orient ="horizontal",\
    length =400, mode ="determinate")
CoMet_progressbar.grid(row=1, column=0, columnspan=1, \
    sticky=W+S, pady=(0,35), padx=(55,50))
tab1_canvas.grid_columnconfigure(12, weight=0)
tab1_canvas.grid_rowconfigure(12, weight=0)
CoMet_progressbar["maximum"] = 100
CoMet_progressbar["value"] = 0

global CoMet_progressbar_counter
CoMet_progressbar_counter = 0

global CoMet_progressbar_check_file
CoMet_progressbar_check_file = True

global CoMet_progressbar_check_folder
CoMet_progressbar_check_folder = True

global CoMet_progressbar_text
CoMet_progressbar_text = tk.Text(tab1_canvas, height=1, width=5)
CoMet_progressbar_text.grid(row=1, column=0, columnspan=1, \
    sticky=E, padx=(0,158), pady=(0,36))
tab1_canvas.grid_columnconfigure(14, weight=0)
tab1_canvas.grid_rowconfigure(14, weight=0)
CoMet_progressbar_text.insert(INSERT, "0%")
CoMet_progressbar_text.config(state=DISABLED, bd=0, relief=FLAT, \
    bg='#ffffff',font=('calibri', '10', 'bold'))

global CoMet_dpi
CoMet_dpi = StringVar(tab1)
CoMet_dpi.set("127")

global CoMet_saveAs
CoMet_saveAs = tk.StringVar(tab1)
CoMet_saveAs.set(".dcm")

global CoMet_uploaded_filename
CoMet_uploaded_filename=StringVar(tab1)
CoMet_uploaded_filename.set("Error!")
```

```
global CoMet_export_folder
CoMet_export_folder=StringVar(tab1)
CoMet_export_folder.set("Error!")

global CoMet_image_to_canvas

global CoMet_correcte_image_filename_box

global CoMet_corrected_image_filename
CoMet_corrected_image_filename=StringVar(tab1)
CoMet_corrected_image_filename.set("Error!")

global CoMet_patientName
CoMet_patientName=StringVar(tab1)
CoMet_patientName.set("Error!")

global CoMet_correctedImage
CoMet_correctedImage=None

global CoMet_border_1_label
CoMet_border_1_label = tk.Label(tab1_canvas)

global CoMet_border_2_label
CoMet_border_2_label = tk.Label(tab1_canvas)

global CoMet_border_3_label
CoMet_border_3_label = tk.Label(tab1_canvas)

global CoMet_border_4_label
CoMet_border_4_label = tk.Label(tab1_canvas)

global CoMet_save_button_frame_1
CoMet_save_button_frame_1 = tk.Frame(tab1_canvas)

global CoMet_save_button_1
CoMet_save_button_1 = tk.Button(CoMet_save_button_frame_1)

global CoMet_save_filename
CoMet_save_filename = tk.Text(CoMet_border_3_label, height=1, width=30)

global CoMet_print_corrected_image
CoMet_print_corrected_image = tk.Canvas(tab1_canvas)

global CoMet_uploaded_file_text
```

```
#———————————————————————————————————————————————————————————————
# All variables defined here are related to the tab Dose Response
# and will be used in notebook.py and Dose_response_functions.py
#———————————————————————————————————————————————————————————————
tab2_files_frame = tk.Frame(tab2_canvas)
tab2_files_frame.config(relief=FLAT, bg='#ffffff', highlightthickness=0, bd=0)

tab2_scroll_canvas = tk.Canvas(tab2_files_frame)
tab2_scroll_canvas.config(bg='#ffffff', height=220, width=400,\
    highlightthickness=0, bd=0, relief=FLAT)
tab2_scroll_canvas.grid_propagate(0)

scroll = Scrollbar(tab2_files_frame, command=tab2_scroll_canvas.yview)
scroll.config(relief=FLAT)

scrollable_frame= tk.Frame(tab2_scroll_canvas)

scrollable_frame.bind("<Configure>", \
    lambda e: tab2_scroll_canvas.configure\
        (scrollregion=tab2_scroll_canvas.bbox('all')))
tab2_scroll_canvas.create_window((0,0), window=scrollable_frame, anchor='nw')
tab2_scroll_canvas.configure(yscrollcommand=scroll.set)

global tab2_canvas_files
tab2_canvas_files = tk.Canvas(scrollable_frame)
tab2_canvas_files.config(relief=FLAT, bg='#ffffff', \
    highlightthickness=0, bd=0)
tab2_canvas_files.pack(fill =BOTH, expand=True)

tab2_files_frame.grid(row=1, column=0, columnspan=2, rowspan=2, \
        sticky=N+S+E+W, padx=(10,0), pady=(10,0))
tab2_canvas.grid_columnconfigure(1, weight=0)
tab2_canvas.grid_rowconfigure(1, weight=0)
tab2_scroll_canvas.pack(side=LEFT, fill=BOTH, expand=True)
scroll.pack(side=RIGHT, fill=Y)

global dose_response_save_calibration_button

global doseResponse_dpi
doseResponse_dpi=StringVar()
doseResponse_dpi.set("127")

global dose_response_var1
dose_response_var1= IntVar()
```

```
dose_response_var1.set(1)

global dose_response_var2
dose_response_var2 = IntVar()
dose_response_var2.set(1)

global dose_response_var3
dose_response_var3 = IntVar()
dose_response_var3.set(1)

global dose_response_uploaded_filenames
dose_response_uploaded_filenames = np.array([])

global dose_response_new_window_row_count
dose_response_new_window_row_count = 4

global dose_response_new_window_weight_count
dose_response_new_window_weight_count = 4

global avg_red_vector
avg_red_vector = []

global avg_green_vector
avg_green_vector = []

global avg_blue_vector
avg_blue_vector = []

global dose_response_files_row_count
dose_response_files_row_count = 2

global dose_response_files_weightcount
dose_response_files_weightcount = 8

global dose_response_inOrOut
dose_response_inOrOut = True

global dose_response_delete_buttons
dose_response_delete_buttons = []

global dose_response_red_list
dose_response_red_list = []

global dose_response_green_list
dose_response_green_list = []
```

```
global dose_response_blue_list
dose_response_blue_list = []

global dose_response_dose_list
dose_response_dose_list = []

global popt_red
popt_red = np.zeros(3)
global pcov_red
pcov_red = np.zeros(3)

global dose_response_batch_number
dose_response_batch_number = "-"

global dose_response_equation_frame
dose_response_equation_frame = tk.Frame(tab2_canvas)
dose_response_equation_frame.grid(row=3, column=3, columnspan=1, \
    sticky=E+W+N+S, padx=(30,0), pady=(0,0))
tab2_canvas.grid_columnconfigure(8, weight=0)
tab2_canvas.grid_rowconfigure(8, weight=0)
dose_response_equation_frame.config(bg='#ffffff', \
    relief=FLAT, highlightthickness=0, width=650, height=210)
dose_response_equation_frame.grid_propagate(0)

global dose_response_equation_image

global dose_response_plot_frame
dose_response_plot_frame = tk.Frame(tab2_canvas)
dose_response_plot_frame.grid(row=1, column=3, rowspan=2, columnspan=4, \
    sticky=N+S+E+W, pady=(0,5), padx=(30,5))
tab2_canvas.grid_columnconfigure(9, weight=0)
tab2_canvas.grid_rowconfigure(9, weight=0)
dose_response_plot_frame.config(bg='#ffffff', \
    relief=FLAT, highlightthickness=0, height=460,width=650)
dose_response_plot_frame.grid_propagate(0)

dose_response_fig = Figure(figsize=(6,4))
dose_response_a = dose_response_fig.add_subplot(111, ylim=(0,40000), xlim=(0,500))
dose_response_plot_canvas = FigureCanvasTkAgg\
    (dose_response_fig, master=dose_response_plot_frame)
dose_response_plot_canvas.get_tk_widget().grid(row=0,column=0,\
    columnspan=4, sticky=N+S+E+W, padx=(5,0), pady=(0,0))
dose_response_a.set_title ("Dose-response", fontsize=12)
dose_response_a.set_ylabel("Pixel value", fontsize=12)
dose_response_a.set_xlabel("Dose", fontsize=12)
dose_response_fig.tight_layout()

global dose_response_sd_list_red
dose_response_sd_list_red = []

global dose_response_sd_list_green
dose_response_sd_list_green = []

global dose_response_sd_list_blue
dose_response_sd_list_blue = []

global dose_response_sd_avg_red
dose_response_sd_avg_red = DoubleVar()
dose_response_sd_avg_red.set(0)

global dose_response_sd_avg_green
dose_response_sd_avg_green = DoubleVar()
dose_response_sd_avg_green.set(0)

global dose_response_sd_avg_blue
dose_response_sd_avg_blue = DoubleVar()
dose_response_sd_avg_blue.set(0)

global dose_response_sd_min_red
dose_response_sd_min_red = DoubleVar()
dose_response_sd_min_red.set(0)

global dose_response_sd_min_red_dose
dose_response_sd_min_red_dose = StringVar()
dose_response_sd_min_red_dose.set('—')

global dose_response_sd_min_green
dose_response_sd_min_green = DoubleVar()
dose_response_sd_min_green.set(0)

global dose_response_sd_min_green_dose
dose_response_sd_min_green_dose = StringVar()
dose_response_sd_min_green_dose.set('—')

global dose_response_sd_min_blue
dose_response_sd_min_blue = DoubleVar()
dose_response_sd_min_blue.set(0)

global dose_response_sd_min_blue_dose
```

```
dose_response_sd_min_blue_dose = StringVar()
dose_response_sd_min_blue_dose.set('-')

global dose_response_sd_max_red
dose_response_sd_max_red = DoubleVar()
dose_response_sd_max_red.set(0)

global dose_response_sd_max_red_dose
dose_response_sd_max_red_dose = StringVar()
dose_response_sd_max_red_dose.set('-')

global dose_response_sd_max_green
dose_response_sd_max_green = DoubleVar()
dose_response_sd_max_green.set(0)

global dose_response_sd_max_green_dose
dose_response_sd_max_green_dose = StringVar()
dose_response_sd_max_green_dose.set('-')

global dose_response_sd_max_blue
dose_response_sd_max_blue = DoubleVar()
dose_response_sd_max_blue.set(0)

global dose_response_sd_max_blue_dose
dose_response_sd_max_blue_dose = StringVar()
dose_response_sd_max_blue_dose.set('-')

global upload_files_here_canvas

#------------------------------------------------------------------------------
# All variables defined here is related to the tab Map Dose
# and will be used in notebook.py and Map_dose_functions.py
#------------------------------------------------------------------------------

global map_dose_film_dataset
map_dose_film_dataset=StringVar(tab3)
map_dose_film_dataset.set("Error!")

global map_dose_isocenter_map_x_coord_scaled
map_dose_isocenter_map_x_coord_scaled = []

global map_dose_isocenter_map_x_coord_unscaled
map_dose_isocenter_map_x_coord_unscaled = []

global map_dose_isocenter_map_y_coord_scaled
```

```
map_dose_isocenter_map_y_coord_scaled = []

global map_dose_isocenter_map_y_coord_unscaled
map_dose_isocenter_map_y_coord_unscaled = []

global map_dose_icocenter_film

global map_dose_film_batch
map_dose_film_batch = IntVar()
map_dose_film_batch.set(0)

global map_dose_ROI_x_start
map_dose_ROI_x_start = IntVar()
map_dose_ROI_x_start.set(0)

global map_dose_ROI_y_start
map_dose_ROI_y_start = IntVar()
map_dose_ROI_y_start.set(0)

global map_dose_ROI_x_end
map_dose_ROI_x_end = IntVar()
map_dose_ROI_x_end.set(0)

global map_dose_ROI_y_end
map_dose_ROI_y_end = IntVar()
map_dose_ROI_y_end.set(0)

#------------------------------------------------------------------------------------
# All variables defined here are related to the tab Profiles and
# will be used in notebook.py and Profiles_functions.py
#------------------------------------------------------------------------------------
global profiles_film_orientation
profiles_film_orientation = StringVar()
profiles_film_orientation.set('—')

global profiles_film_orientation_menu

global profiles_film_dataset
global profiles_film_dataset_red_channel
global profiles_film_dataset_red_channel_dose
global profiles_film_variable_ROI_coords

global profiles_film_dataset_ROI
global profiles_film_dataset_ROI_red_channel
global profiles_doseplan_dataset_ROI
global profiles_film_dataset_ROI_red_channel_dose

global profiles_view_film_doseplan_ROI
profiles_view_film_doseplan_ROI = tk.Canvas(tab4_canvas)
profiles_view_film_doseplan_ROI.grid(row=8, column=0, \
    columnspan=10, rowspan=10, sticky=N+W)
tab4_canvas.grid_columnconfigure(11, weight=0)
tab4_canvas.grid_rowconfigure(11, weight=0)
profiles_view_film_doseplan_ROI.config(bg='#ffffff', \
    relief=FLAT, highlightthickness=0)

global profile_plot_canvas
profile_plot_canvas = tk.Canvas(tab4_canvas)
profile_plot_canvas.grid(row=1, column=2, rowspan=7, columnspan=2, \
    sticky=N+E+W, pady=(0,5), padx=(5,10))
tab4_canvas.grid_columnconfigure(4, weight=0)
tab4_canvas.grid_rowconfigure(4, weight=0)
profile_plot_canvas.config(bg='#ffffff', relief=FLAT, \
    highlightthickness=0, width=950, height=520)
profile_plot_canvas.grid_propagate(0)

profiles_fig = Figure(figsize=(6,4))
profiles_a = profiles_fig.add_subplot(111, ylim=(0,40000), xlim=(0,500))
profiles_plot_canvas = FigureCanvasTkAgg(profiles_fig, master=profile_plot_canvas)
profiles_plot_canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, \
    rowspan=4, sticky=N+E+W+S, padx=(5,0), pady=(0,0))
profiles_a.set_title ("Profiles", fontsize=12)
profiles_a.set_ylabel("Dose (Gy)", fontsize=12)
profiles_a.set_xlabel("Distance (mm)", fontsize=12)
profiles_fig.tight_layout()

global profiles_showPlanes_image
global profiles_showDirections_image

global profiles_depth
global profiles_depth_float

global profiles_film_factor_input

global profiles_mark_isocenter_button_image
global profiles_mark_ROI_button_image
global profiles_mark_point_button_image

global profiles_iscoenter_coords
profiles_iscoenter_coords = []
```

```
global profiles_film_isocenter
global profiles_film_reference_point

global profiles_distance_isocenter_ROI
profiles_distance_isocenter_ROI = []
global profiles_distance_reference_point_ROI
profiles_distance_reference_point_ROI = []

global profiles_mark_isocenter_up_down_line
profiles_mark_isocenter_up_down_line = []
global profiles_mark_isocenter_right_left_line
profiles_mark_isocenter_right_left_line = []
global profiles_mark_isocenter_oval
profiles_mark_isocenter_oval = []
global profiles_mark_ROI_rectangle
profiles_mark_ROI_rectangle = []

global profiles_mark_reference_point_oval
profiles_mark_reference_point_oval = []

global profiles_ROI_coords
profiles_ROI_coords = []

global profiles_done_button
profiles_done_button = None
global profiles_done_button_reference_point
profiles_done_button_reference_point = None

global profiles_isocenter_check
profiles_isocenter_check=False
global profiles_reference_point_check
profiles_reference_point_check = False

global profiles_ROI_check
profiles_ROI_check = False
global profiles_ROI_reference_point_check
profiles_ROI_reference_point_check = False

global profiles_film_batch
profiles_film_batch = IntVar()
profiles_film_batch.set(0)

global profiles_popt_red
profiles_popt_red = np.zeros(3)
```

```
global profiles_upload_button_doseplan
global profiles_upload_button_film
global profiles_upload_button_rtplan

global profiles_dataset_doseplan
profiles_dataset_doseplan = None
global profiles_dataset_rtplan

global profiles_test_if_added_doseplan
global profiles_test_if_added_rtplan
profiles_test_if_added_doseplan = False
profiles_test_if_added_rtplan = False

global profiles_isocenter_mm

global profiles_dose_scaling_doseplan
profiles_dose_scaling_doseplan = []

global profiles_max_dose_film

global profiles_choose_profile_canvas
profiles_choose_profile_canvas = tk.Canvas(profiles_view_film_doseplan_ROI)
profiles_choose_profile_canvas.grid(row=0, column=0, sticky=N+S+W)
profiles_choose_profile_canvas.config(bg='#ffffff', \
    relief=FLAT, highlightthickness=0)

global profiles_adjust_ROI_canvas
profiles_adjust_ROI_canvas = tk.Canvas(profile_plot_canvas)
profiles_adjust_ROI_canvas.grid(row=2, column=4, sticky=N+W)
profiles_adjust_ROI_canvas.config(bg='#ffffff', \
    relief=FLAT, highlightthickness=0)

global profiles_choice_of_profile_line_type
profiles_choice_of_profile_line_type = StringVar()
profiles_choice_of_profile_line_type.set("h")

profiles_choose_profile_type_text = \
    tk.Text(profiles_choose_profile_canvas, height=1)
profiles_choose_profile_type_text.insert(INSERT, "How to draw the profile:")
profiles_choose_profile_type_text.pack(side=TOP)
profiles_choose_profile_type_text.config(bg='#ffffff', relief=FLAT, \
highlightthickness=0, state=DISABLED, font=('calibri', '11'))

Radiobutton(profiles_choose_profile_canvas, text="Horizontal", \
    variable=profiles_choice_of_profile_line_type, value="h", \
        bg='#ffffff', cursor='hand2').pack(side=LEFT)
Radiobutton(profiles_choose_profile_canvas, text="Vertical", \
    variable=profiles_choice_of_profile_line_type, value='v', \
        bg='#ffffff', cursor='hand2').pack(side=LEFT)
Radiobutton(profiles_choose_profile_canvas, text="Draw", \
    variable=profiles_choice_of_profile_line_type, value="d", \
        bg='#ffffff', cursor='hand2').pack(side=LEFT)

profiles_adjust_ROI_text = \
    tk.Text(profiles_adjust_ROI_canvas, width=20, height=1)
profiles_adjust_ROI_text.insert(INSERT, "Adjust ROI in film: ")
profiles_adjust_ROI_text.config(state=DISABLED, \
    font=('calibri', '11'), bg='#ffffff', relief=FLAT, bd=0)
profiles_adjust_ROI_text.pack(side=TOP, padx=(0,0))

global profiles_redefine_film_ROI_frame
profiles_redefine_film_ROI_frame = tk.Frame(profiles_adjust_ROI_canvas)
profiles_redefine_film_ROI_frame.pack(side=BOTTOM, padx=(0,0))
profiles_redefine_film_ROI_frame.config(bg='#ffffff')
global profiles_adjust_button_left
global profiles_adjust_button_right
global profiles_adjust_button_down
global profiles_adjust_button_up

global profiles_film_panedwindow
profiles_film_panedwindow = \
    PanedWindow(profiles_view_film_doseplan_ROI, orient='horizontal')
profiles_film_panedwindow.grid(row=1, column=0, \
    columnspan=3, rowspan=5, sticky=N+W)
profiles_film_panedwindow.configure(sashrelief = RAISED, showhandle=True)

global profiles_scanned_image_text_image
global profiles_film_dose_map_text_image
global profiles_doseplan_text_image

global doseplan_write_image
global film_write_image
global doseplan_write_image_width
global doseplan_write_image_height
global doseplan_write_image_var_x
doseplan_write_image_var_x= 0
global doseplan_write_image_var_y
doseplan_write_image_var_y = 0
global profiles_coordinate_in_dataset
```

```
profiles_coordinate_in_dataset = 0

global profiles_first_time_in_drawProfiles
profiles_first_time_in_drawProfiles = True

global new_window_factor_textbox

global profiles_doseplan_lateral_displacement
global profiles_doseplan_vertical_displacement
global profiles_doseplan_longitudianl_displacement
global profiles_doseplan_patient_position

global profiles_reference_point_in_doseplan

global profiles_input_lateral_displacement
global profiles_input_longitudinal_displacement
global profiles_input_vertical_displacement

global profiles_isocenter_or_reference_point

global profiles_lateral
global profiles_vertical
global profiles_longitudinal

global profiles_number_of_doseplans
profiles_number_of_doseplans = 0
global profiles_number_of_doseplans_row_count
profiles_number_of_doseplans_row_count = 4
global profiles_doseplans_grid_config_count
profiles_doseplans_grid_config_count = 6
global profiles_doseplans_filenames
profiles_doseplans_filenames = []
global profiles_doseplans_factor_text
profiles_doseplans_factor_text = []
global profiles_doseplans_factor_input
profiles_doseplans_factor_input = []

global profiles_doseplan_dataset_ROI_several
profiles_doseplan_dataset_ROI_several = []
global profiles_several_img
profiles_several_img = []

global profiles_film_factor

global profiles_lines
```

```
profiles_lines = []

global end_point
end_point = None

global profiles_line_coords_film
global profiles_line_coords_doseplan

global profiles_dataset_film_variable_draw
global profiles_dataset_doesplan_variable_draw

global max_dose_doseplan

global profiles_slice_offset
global profiles_offset

global profiles_export_plot_button


#------------------------------------------------------------------------------------------
# All variables defined here are related to the tab DVH and
# will be used in notebook.py and DVH_functions.py
#------------------------------------------------------------------------------------------

global DVH_film_orientation
DVH_film_orientation = StringVar()
DVH_film_orientation.set('-')

#global DVH_doseplans_scroll_frame

global DVH_number_of_doseplans
DVH_number_of_doseplans = 0
global DVH_number_of_doseplans_row_count
DVH_number_of_doseplans_row_count = 4
global DVH_doseplans_grid_config_count
DVH_doseplans_grid_config_count = 6
global DVH_doseplans_filenames
DVH_doseplans_filenames = []
global DVH_doseplans_factor_text
DVH_doseplans_factor_text = []
global DVH_doseplans_factor_input
DVH_doseplans_factor_input = []

global DVH_doseplan_dataset_ROI_several
DVH_doseplan_dataset_ROI_several = []
```

```
global DVH_several_img
DVH_several_img = []

global profiles_film_factor

global DVH_film_orientation_menu

global DVH_film_factor_input
global DVH_film_factor

global DVH_film_dataset
global DVH_film_dataset_red_channel

global DVH_film_dataset_ROI
global DVH_film_dataset_ROI_red_channel
global DVH_doseplan_dataset_ROI

global DVH_film_dataset_ROI_red_channel_dose

global DVH_film_write_image
global DVH_film_dose_write_image

global DVH_max_dose_film
global DVH_max_dose_doseplan

global DVH_view_film_doseplan_ROI
DVH_view_film_doseplan_ROI = tk.Canvas(tab5_canvas)
DVH_view_film_doseplan_ROI.grid(row=1, column=1, rowspan=5, \
    sticky=S+E+W+N, pady=(0,5), padx=(5,10), columnspan=6)
tab5_canvas.grid_columnconfigure(11, weight=0)
tab5_canvas.grid_rowconfigure(11, weight=0)
DVH_view_film_doseplan_ROI.config(bg='#ffffff', \
    relief=FLAT, highlightthickness=0)

global temp_image_canvas


global DVH_iscoenter_coords
DVH_iscoenter_coords = []

global DVH_film_isocenter

global DVH_film_reference_point
global DVH_distance_isocenter_ROI
DVH_distance_isocenter_ROI = []

global DVH_distance_reference_point_ROI
DVH_distance_reference_point_ROI = []

global DVH_mark_isocenter_up_down_line
DVH_mark_isocenter_up_down_line = []
global DVH_mark_isocenter_right_left_line
DVH_mark_isocenter_right_left_line = []

global DVH_mark_isocenter_oval
DVH_mark_isocenter_oval = []

global DVH_mark_ROI_rectangle
DVH_mark_ROI_rectangle = []

global DVH_mark_reference_point_oval
DVH_mark_reference_point_oval = []

global DVH_ROI_coords
DVH_ROI_coords = []

global DVH_film_variable_ROI_coords

global DVH_done_button
DVH_done_button = None

global DVH_done_button_reference_point
DVH_done_button_reference_point = None

global DVH_isocenter_check
DVH_isocenter_check=False

global DVH_reference_point_check
DVH_reference_point_check = False

global DVH_ROI_check
DVH_ROI_check = False

global DVH_ROI_reference_point_check
DVH_ROI_reference_point_check = False

global DVH_film_batch
DVH_film_batch = IntVar()
```

```
DVH_film_batch.set(0)

global DVH_popt_red
DVH_popt_red = np.zeros(3)

global DVH_upload_button_doseplan

global DVH_upload_button_film

global DVH_upload_button_rtplan

global DVH_upload_button_struct

global DVH_dataset_doseplan
global DVH_dataset_rtplan
global DVH_dataset_structure_file

global DVH_test_if_added_doseplan
global DVH_test_if_added_rtplan
global DVH_test_if_added_struct
DVH_test_if_added_doseplan = False
DVH_test_if_added_rtplan = False
DVH_test_if_added_struct = False

global DVH_isocenter_mm

global DVH_dose_scaling_doseplan

global DVH_contour_names
global DVH_ROIContourSequence

global DVH_contours
DVH_contours = []

global DVH_doseplan_write_image
global DVH_doseplan_write_image_width
global DVH_doseplan_write_image_height

global DVH_doseplan_lateral_displacement
global DVH_doseplan_vertical_displacement
global DVH_doseplan_longitudianl_displacement
global DVH_doseplan_patient_position

global DVH_reference_point_in_doseplan
global DVH_input_lateral_displacement
global DVH_input_longitudinal_displacement
global DVH_input_vertical_displacement

global DVH_slice_offset
global DVH_offset

global DVH_isocenter_or_reference_point

global DVH_lateral
global DVH_vertical
global DVH_longitudinal

global DVH_plot_canvas
DVH_plot_canvas = tk.Canvas(tab5_canvas)
DVH_plot_canvas.grid(row=7, column=0, rowspan=30, columnspan=7, \
    sticky=N+E+W, pady=(0,5), padx=(5,10))
tab5_canvas.grid_columnconfigure(40, weight=0)
tab5_canvas.grid_rowconfigure(40, weight=0)
DVH_plot_canvas.config(bg='#ffffff', relief=FLAT, \
    highlightthickness=0, width=500, height=650)
DVH_plot_canvas.grid_propagate(0)

global DVH_list_contours_canvas
DVH_list_contours_canvas = tk.Canvas(tab5_canvas)
DVH_list_contours_canvas.grid(row=6, column=0, columnspan=7, sticky=N+S+W+E)
DVH_list_contours_canvas.config(height=35, bd=0, highlightthickness=0, \
    bg='#ffffff', relief=FLAT)
DVH_list_contours_canvas.grid_propagate(0)

DVH_initial_fig = Figure(figsize=(10,6))
DVH_a = DVH_initial_fig.add_subplot(111, ylim=(0,1), xlim=(0,50))
DVH_initial_plot_canvas = FigureCanvasTkAgg(DVH_initial_fig, master=DVH_plot_canvas)
DVH_initial_plot_canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, \
    sticky=N+E+W+S, padx=(5,0), pady=(0,0))
DVH_a.set_title ("Dose volume histogram", fontsize=12)
DVH_a.set_ylabel("Volume (%)", fontsize=12)
DVH_a.set_xlabel("Dose (Gy)", fontsize=12)
DVH_initial_fig.tight_layout()

global DVH_export_button


#------------------------------------------------------------------------------
# Here the correction matrix used to perform background corrections
# on scanned images of radiochromic film are uploaded from *.txt
# files. This will be used on all images of film being uploaded
# into Fidora. Fidora only uses the correction matrix with 127 dpi,
# but has the oppertunity to also use 72 dpi.
#———————————————————————————————————————————————————————————————————
global correction127_red
with open('red_127.txt', 'r') as f:
    correction127_red = [[float(num) for num in line.split(',')] for line in f]
correction127_red = np.matrix(correction127_red)
global correction127_green
with open('green_127.txt', 'r') as f:
    correction127_green = [[float(num) for num in line.split(',')] for line in f]
correction127_green = np.matrix(correction127_green)

global correction127_blue
with open('blue_127.txt', 'r') as f:
    correction127_blue = [[float(num) for num in line.split(',')] for line in f]
correction127_blue = np.matrix(correction127_blue)

global correction72_red
with open('output_red_72.txt', 'r') as f:
    correction72_red = [[float(num) for num in line.split(',')] for line in f]
correction72_red = np.matrix(correction72_red)

global correction72_green
with open('output_green_72.txt', 'r') as f:
    correction72_green = [[float(num) for num in line.split(',')] for line in f]
correction72_green = np.matrix(correction72_green)

global correction72_blue
with open('output_blue_72.txt', 'r') as f:
    correction72_blue = [[float(num) for num in line.split(',')] for line in f]
correction72_blue = np.matrix(correction72_blue)

global correctionMatrix127
correctionMatrix127 = np.zeros((1270,1016,3))
correctionMatrix127[:,:,0] = correction127_blue[:,:]
correctionMatrix127[:,:,1] = correction127_green[:,:]
correctionMatrix127[:,:,2] = correction127_red[:,:]

global correctionMatrix72
correctionMatrix72 = np.zeros((720,576,3))
correctionMatrix72[:,:,0] = correction72_blue[:,:]
correctionMatrix72[:,:,1] = correction72_green[:,:]
correctionMatrix72[:,:,2] = correction72_red[:,:]
```

CoMet_functions.py

```
#------------------------------------------------------------------
#
# CoMet_functions.py
# version 26.07.20
#
# To be used related to the tab CoMet in Fidora.
#
#------------------------------------------------------------------


import Globals
import tkinter as tk
from tkinter import filedialog, INSERT, DISABLED, messagebox, NORMAL, simpledialog, \
    PhotoImage, BOTH, E, S, N, W, ACTIVE, FLAT
import os
from os.path import normpath, basename
import cv2
from cv2 import imread, IMREAD_ANYCOLOR, IMREAD_ANYDEPTH, imwrite
import numpy as np
import SimpleITK as sitk
import pydicom
from PIL import Image, ImageTk


def nothingButton():
#----------------------------------------
# Function to only return
# Is used in cases where nothing should happen in
# a active button (they will later be reconfigured)
#----------------------------------------
    return


def UploadAction(event=None):
#----------------------------------------
# Function to upload the scanned image of film.
# Callback function to the button
# CoMet_upload_button in notebook.py.
#----------------------------------------
    Globals.CoMet_uploaded_filename.set(filedialog.askopenfilename())
    ext = os.path.splitext(Globals.CoMet_uploaded_filename.get())[-1].lower()
    if(ext==".tif"):
        Globals.CoMet_uploaded_file_text = \
            tk.Text(Globals.CoMet_border_1_label,  height=1, width=32)
        Globals.CoMet_uploaded_file_text.grid(row=0, column=0, columnspan=2, \
            sticky=E+W, pady=(20,20), padx=(100,0))
        Globals.CoMet_uploaded_file_text.insert(INSERT, \
            basename(normpath(Globals.CoMet_uploaded_filename.get())))
        Globals.CoMet_uploaded_file_text.config(state=DISABLED, bd=0, \
            font=('calibri', '10'), fg='gray', bg='#ffffff')

        if (Globals.CoMet_progressbar_check_file):
            Globals.CoMet_progressbar_counter +=1
            Globals.CoMet_progressbar_check_file = False
        Globals.CoMet_progressbar["value"] = Globals.CoMet_progressbar_counter*25
        Globals.CoMet_progressbar_text = \
            tk.Text(Globals.tab1_canvas, height = 1, width=5)
        Globals.CoMet_progressbar_text.grid(row=1, column=0, columnspan=1, \
            sticky=E, padx=(0,158), pady=(0,36))
        Globals.CoMet_progressbar_text.insert(INSERT, \
            str(Globals.CoMet_progressbar_counter*25)+"%")
        if(Globals.CoMet_progressbar_counter*25 == 100):
            Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0, \
                relief=FLAT, bg='#2C8EAD', font=('calibri', '10', 'bold'))
        else:
            Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0, \
                relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))

    elif(ext==""):
        Globals.CoMet_uploaded_filename.set("Error!")
    else:
        messagebox.showerror("Error", "The file must be a .tif file")
        Globals.CoMet_uploaded_filename.set("Error!")


def setCoMet_export_folder():
#--------------------------------------------------
# Function to choose which folder to place the
# corrected image resulting in CoMet.
# Callback function to the button
# CoMet_folder_button in notebook.py.
#--------------------------------------------------
    Globals.CoMet_export_folder.set(filedialog.askdirectory())
    if(Globals.CoMet_export_folder.get() == ""):
        Globals.CoMet_export_folder.set("Error!")
    else:
        current_folder = os.getcwd()
        os.chdir(Globals.CoMet_export_folder.get())
        save_to_folder=\
            tk.Text(Globals.CoMet_border_2_label, height=1, width=32)
        save_to_folder.grid(row=0, column=0, columnspan=3, \
            sticky=E+W, pady=(20,20), padx=(100,0))
        save_to_folder.insert(INSERT, \
            basename(normpath(Globals.CoMet_export_folder.get())))
        save_to_folder.config(state=DISABLED, bd=0, \
            font=('calibri', '10'), fg='gray', bg='#ffffff')
        os.chdir(current_folder)
        if(Globals.CoMet_progressbar_check_folder):
            Globals.CoMet_progressbar_counter +=1
            Globals.CoMet_progressbar_check_folder = False
        Globals.CoMet_progressbar["value"] = Globals.CoMet_progressbar_counter*25
        Globals.CoMet_progressbar_text = \
            tk.Text(Globals.tab1_canvas, height=1, width=5)
        Globals.CoMet_progressbar_text.grid(row=1, column=0, \
            columnspan=1, sticky=E, padx=(0,158), pady=(0,36))
        Globals.CoMet_progressbar_text.insert(INSERT, \
            str(Globals.CoMet_progressbar_counter*25) + "%")
        if(Globals.CoMet_progressbar_counter*25 == 100):
            Globals.CoMet_progressbar_text.config(state=DISABLED, \
                bd=0, relief=FLAT, bg='#2C8EAD', font=('calibri', '10', 'bold'))
        else:
            Globals.CoMet_progressbar_text.config(state=DISABLED, \
                bd=0, relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))


def checkAllWidgets(*args):
#------------------------------------------------------------
# Function to check if all actions has been done
# in CoMet and the user are ready to perform
# the correction on the uploaded image.
# Used in the function Correct() in current file.
#------------------------------------------------------------
    if(Globals.CoMet_uploaded_filename.get()=="Error!" or \
        Globals.CoMet_export_folder.get()=="Error!" or \
            Globals.CoMet_corrected_image_filename.get()=="Error!"):
        return False
    else:
        return True


def correctionMatrix():
#------------------------------------------------------------
# Function that performes the correction on the
# uploaded image. This happens as an absolute
# subtraction between the pixel values in the
# image and the correction matrix.
# Called in the function Correct() in current file.
#-----------------------------------------------------------
    dataset = cv2.imread(Globals.CoMet_uploaded_filename.get().lstrip(), \
        cv2.IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
    if(dataset is None):
        current_folder = os.getcwd()
        script_path = Globals.CoMet_uploaded_filename.get()
        parent = os.path.dirname(script_path)
        os.chdir(parent)
        dataset=cv2.imread(basename(normpath(script_path)), \
            cv2.IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
        os.chdir(current_folder)
    if(dataset is None):
         messagebox.showerror("Error", \
"Something has happen. Check that the filename only contain english letters")
         return

    if(dataset.shape[2] == 3):
        if(dataset.shape[0]==1270 and dataset.shape[1]==1016):
            temp = abs(dataset-Globals.correctionMatrix127)
            Globals.CoMet_correctedImage = np.clip(temp, 0, 65535)
        elif(dataset.shape[0]==720 and dataset.shape[1]==576):
            temp = abs(dataset - Globals.correctionMatrix72)
            Globals.CoMet_correctedImage = np.clip(temp, 0, 65535)
        else:
            messagebox.showerror("Error",\
"The resolution of the image is not \
consistent with dpi. Must be either 72 or 127")

    else:
        messagebox.showerror("Error",\
            "The uploaded image need to be in RGB-format")


def Correct():
#-----------------------------------------------------------
# Function to initiate the correction on the
# uploaded image.
# Callback function to the button
# CoMet_correct_button in notebook.py.
#-----------------------------------------------------------
    if(checkAllWidgets() is False):
        messagebox.showerror("Error", "All boxes must be filled")
        return
    current_folder = os.getcwd()
    os.chdir(Globals.CoMet_export_folder.get())
    if(os.path.exists(Globals.CoMet_export_folder.get() + '/' + \
        Globals.CoMet_corrected_image_filename.get().lstrip() + \
            Globals.CoMet_saveAs.get()) is True):
        os.chdir(current_folder)
        messagebox.showerror("Error", \
            "Filename already exists in folder. Please write a new filename")
        Globals.CoMet_progressbar_counter -= 1
        Globals.CoMet_progressbar["value"] = Globals.CoMet_progressbar_counter*25
        Globals.CoMet_progressbar_text = \
            tk.Text(Globals.tab1_canvas, width = 5, height=1)
        Globals.CoMet_progressbar_text.grid(row=1, column=0, \
            columnspan=1, sticky=E, padx=(0,158), pady=(0,36))
        Globals.CoMet_progressbar_text.insert(INSERT, \
            str(Globals.CoMet_progressbar_counter*25) + "%")
        if(Globals.CoMet_progressbar_counter*25 == 100):
            Globals.CoMet_progressbar_text.config(state=DISABLED, \
                bd=0, relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))
        else:
            Globals.CoMet_progressbar_text.config(state=DISABLED, \
                bd=0, relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))
        Globals.CoMet_save_button_1.config(state=ACTIVE)
        Globals.CoMet_save_filename.config(state=NORMAL)
        return

    os.chdir(current_folder)
    correctionMatrix()

    if (Globals.CoMet_correctedImage is None):
        messagebox.showerror("Error", \
"The image could not be corrected. \
Please check all the specifications and try again.")
        Globals.CoMet_progressbar["value"]=0
        Globals.CoMet_progressbar_text = \
            tk.Text(Globals.tab1_canvas, height=1, width=5)
        Globals.CoMet_progressbar_text.grid(row=1, column=0, columnspan=1, \
            sticky=E, padx=(0,158), pady=(0,36))
        Globals.CoMet_progressbar_text.insert(INSERT, "0%")
        Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0, \
            relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))
    else:
        Globals.CoMet_progressbar_counter +=1
        Globals.CoMet_progressbar["value"] = Globals.CoMet_progressbar_counter*25
        Globals.CoMet_progressbar_text = \
            tk.Text(Globals.tab1_canvas, height=1, width=5)
        Globals.CoMet_progressbar_text.grid(row=1, column=0, \
            columnspan=1, sticky=E, padx=(0,158), pady=(0,36))
        Globals.CoMet_progressbar_text.insert(INSERT, \
            str(Globals.CoMet_progressbar_counter*25) + "%")
        if(Globals.CoMet_progressbar_counter*25 == 100):
            Globals.CoMet_progressbar_text.config(state=DISABLED, \
                bd=0, relief=FLAT, bg='#2C8EAD', font=('calibri', '10', 'bold'))
        else:
            Globals.CoMet_progressbar_text.config(state=DISABLED, \
                bd=0, relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))

    R=Globals.CoMet_correctedImage[:,:,2]
    G=Globals.CoMet_correctedImage[:,:,1]
    B=Globals.CoMet_correctedImage[:,:,0]

    if(Globals.CoMet_dpi.get()=="127"):
        corrImg_dicom = np.zeros((1270,1016,3))
        corrImg_dicom = corrImg_dicom.astype('uint16')
        corrImg_dicom[:,:,0]=R; corrImg_dicom[:,:,1]=G;corrImg_dicom[:,:,2]=B
    elif(Globals.CoMet_dpi.get() =="72"):
        corrImg_dicom = np.zeros((720,576,3))
        corrImg_dicom = corrImg_dicom.astype('uint16')
        corrImg_dicom[:,:,0]=R; corrImg_dicom[:,:,1]=G;corrImg_dicom[:,:,2]=B
    else:
        messagebox.showerror("Error", \
"Wrong DPI in image. No correction.\n\
Please check all specifications and try again.")

    corrImg_dicom = np.moveaxis(corrImg_dicom,-2,1)
    corrImg_dicom = np.rollaxis(corrImg_dicom,2,0)
    img_dicom = sitk.GetImageFromArray(corrImg_dicom)
    current_folder = os.getcwd()
    os.chdir(Globals.CoMet_export_folder.get())
    sitk.WriteImage(img_dicom, \
        Globals.CoMet_corrected_image_filename.get().lstrip() + \
            Globals.CoMet_saveAs.get())
    os.chdir(current_folder)
    mod_NameAndModality = pydicom.dcmread(Globals.CoMet_export_folder.get() + \
        '/' + Globals.CoMet_corrected_image_filename.get().lstrip() + \
            Globals.CoMet_saveAs.get())
    mod_NameAndModality.Modality = "RTDOSE"
    if(Globals.CoMet_patientName.get() != "Error!"):
        mod_NameAndModality.PatientName = Globals.CoMet_patientName.get()
    else:
        mod_NameAndModality.PatientName = "First^Last"

    mod_NameAndModality.save_as(Globals.CoMet_export_folder.get() + '/' \
        + Globals.CoMet_corrected_image_filename.get().lstrip() \
            + Globals.CoMet_saveAs.get())

    ds = pydicom.dcmread(Globals.CoMet_export_folder.get() + '/' \
        + Globals.CoMet_corrected_image_filename.get().lstrip() \
            + Globals.CoMet_saveAs.get() )

    img = ds.pixel_array
    RGB_image = np.zeros((img.shape[1], img.shape[2], 3))

    for i in range(img.shape[0]):
        RGB_image[:,:,i] = img[i, :,:]

    img8 = (RGB_image/256).astype('uint8')
    img8 = cv2.resize(img8, dsize=(int(img8.shape[1]/2.67),\
        int(img8.shape[0]/2.67)))
    height, width, channels = img8.shape
    img8 = Image.fromarray(img8, 'RGB')

    Globals.CoMet_image_to_canvas =  ImageTk.PhotoImage(image=img8)
    Globals.CoMet_print_corrected_image.delete('all')
    Globals.CoMet_print_corrected_image.create_image(180,250,\
        image=Globals.CoMet_image_to_canvas)
    Globals.CoMet_print_corrected_image.image = Globals.CoMet_image_to_canvas
```

Dose_response_functions.py

```
#-----------------------------------------------------------------------
#
# Version 16.08.20
#
# Fuctions used in relation to the tab Dose Response
#-----------------------------------------------------------------------

import Globals
import tkinter as tk
import tkinter.ttk
from tkinter import filedialog, INSERT, DISABLED, messagebox, NORMAL, simpledialog, \
    PhotoImage, BOTH, Toplevel, GROOVE, ACTIVE, FLAT, N, S, W, E, ALL, ttk, LEFT, \
        RIGHT, Y, Label, X, END, Button, StringVar, Scrollbar
import cv2
import numpy as np
import os
from os.path import normpath, basename
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from scipy.optimize import curve_fit
from scipy.optimize import curve_fit, OptimizeWarning
from PIL import Image, ImageTk
import sys
from datetime import datetime
import re
import warnings
warnings.filterwarnings("error")

def nothingButton():
#---------------------------------------------------
# Function to only return
# Is used in cases where nothing should happen in
# a active button (they will later be reconfigured)
#---------------------------------------------------
    return



def saveCalibration():
#---------------------------------------------------
# Function to save the calibration performed
#
# This is a callback function for the button
# dose_response_save_calibration_button in
# notebook.py
#---------------------------------------------------
    ask_batch_window = tk.Toplevel(Globals.tab2)
    ask_batch_window.geometry("400x180")
    ask_batch_window.grab_set()
    ask_batch_window_canvas = tk.Canvas(ask_batch_window)
    ask_batch_window_canvas.config(bg='#ffffff', bd=0, highlightthickness=0)
    ask_batch_window_canvas.pack(expand=True, fill=BOTH)

    batch_info = tk.Text(ask_batch_window_canvas, width=50, height=3)
    batch_info.grid(row=0, column=0, columnspan=2, sticky=N+S+E+W, \
        padx=(10,10), pady=(30,10))
    ask_batch_window_canvas.grid_columnconfigure(0, weight=0)
    ask_batch_window_canvas.grid_rowconfigure(0, weight=0)
    batch_info.insert(INSERT, 'Write the LOT number of current GafChromic film:\n\
        (Defaults to -)')
    batch_info.config(state=DISABLED, bd = 0, font=('calibri', '12'))

    batch = tk.Text(ask_batch_window_canvas, width=20, height=1)
    batch.grid(row=1, column=0, sticky=N+S+W+E, padx=(5,5), pady=(10,10))
    ask_batch_window_canvas.grid_columnconfigure(1, weight=0)
    ask_batch_window_canvas.grid_rowconfigure(1, weight=0)
    batch.insert(INSERT, " ")
    batch.config(state=NORMAL, bd = 3, font=('calibri', '12'))

    def save_batch():
    #-----------------------------------------------------------
    # Function to read the earlier saved calibration-file
    #
    # This is a callback function for the button
    # save_batch_button in function save_calibration
    #-----------------------------------------------------------
        Globals.dose_response_batch_number= batch.get("1.0",'end-1c')
        if(Globals.dose_response_batch_number == " "):
            Globals.dose_response_batch_number = "-"
            save_batch_button.config(state=DISABLED)
            ask_batch_window.destroy()
        elif(re.match("^[A-Za-z0-9_]*$", \
            (Globals.dose_response_batch_number).lstrip())==None):
            messagebox.showerror("Error","LOT number can only contain letters and/or numbers")
            ask_batch_window.destroy()
            saveCalibration()
            return
        else:
            save_batch_button.config(state=DISABLED)
            ask_batch_window.destroy()

        f = open('calibration.txt', 'r')
        lines = f.readlines()
        f.close()
        string_to_file = str(datetime.now()) + " " + \
            str(Globals.dose_response_batch_number) + " " + \
            str(Globals.popt_red[0]) + " " + str(Globals.popt_red[1]) \
                + " " + str(Globals.popt_red[2]) + "\n"
        if(len(lines) < 5):
            f = open('calibration.txt', 'a')
```

```
            f.write(string_to_file)
            f.close()
        else:
            new_lines = [lines[1], lines[2], lines[3], lines[4], string_to_file]
            f = open('calibration.txt', 'w')
            for i in range(len(new_lines)):
                f.write(new_lines[i])
            f.close()

        messagebox.showinfo("Info", "The calibration has been saved")

    save_button_frame = tk.Frame(ask_batch_window_canvas)
    save_button_frame.grid(row=1, column = 1, padx=(5,5), pady=(10,10))
    ask_batch_window_canvas.grid_columnconfigure(2, weight=0)
    ask_batch_window_canvas.grid_rowconfigure(2, weight=0)
    save_button_frame.config(bg = '#ffffff')

    save_batch_button = tk.Button(save_button_frame, text='Save', \
        image=Globals.save_button, cursor='hand2',font=('calibri', '14'),\
            relief=FLAT, state=ACTIVE, command=save_batch)
    save_batch_button.pack(fill=BOTH, expand=True)
    save_batch_button.image = Globals.save_button


def UploadAction(new_window, event=None):
#-------------------------------------------------
# Function upload scanned film files.
#
# This is a callback function for the button
# upload_button in the function create_window()
#-------------------------------------------------
    file = filedialog.askopenfilename()
    ext = os.path.splitext(file)[-1].lower()
    if(ext==".tif"):
        Globals.dose_response_uploaded_filenames = \
            np.append(Globals.dose_response_uploaded_filenames, file)
        uploaded_filename = tk.Text(new_window, height=1, width=1)
        uploaded_filename.grid(row=Globals.dose_response_new_window_row_count, \
            column=0, columnspan=2, sticky=E+W, pady=(5,5), padx=(100,0))
        new_window.grid_columnconfigure\
            (Globals.dose_response_new_window_weight_count, weight=0)
        new_window.grid_rowconfigure\
            (Globals.dose_response_new_window_weight_count, weight=0)
        uploaded_filename.insert(INSERT, basename(normpath(file)))
        uploaded_filename.config(state=DISABLED, bd=0, \
            font=('calibri', '12'), fg='gray')
        Globals.dose_response_new_window_row_count+=1
        Globals.dose_response_new_window_weight_count+=1
    elif(ext==""):
        return
    else:
        messagebox.showerror("Error", "The file must be a .tif file")

def readImage(filename):
    image = cv2.imread(filename, cv2.IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
    if(image is None):
        current_folder = os.getcwd()
        parent = os.path.dirname(filename)
        os.chdir(parent)
        image=cv2.imread(basename(normpath(filename)), \
            cv2.IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
        os.chdir(current_folder)
    if(image is None):
         messagebox.showerror("Error", \
             "Something has happen. Check that the filename only contain latin")
         return

    if(image.shape[2] == 3):
        if(image.shape[0]==1270 and image.shape[1]==1016):
            Globals.doseResponse_dpi.set("127")
            image = abs(image-Globals.correctionMatrix127)
            image = np.clip(image, 0, 65535)
        elif(image.shape[0]==720 and image.shape[1]==576):
            Globals.doseResponse_dpi.set("72")
            image = abs(image - Globals.correctionMatrix72)
            image = np.clip(image, 0, 65535)
        else:
            messagebox.showerror("Error",\
                "The resolution of the image is not consistent with dpi")

    else:
        messagebox.showerror("Error","The uploaded image need to be in RGB-format")

    sum_red=0;sum_green=0;sum_blue=0
    if(Globals.doseResponse_dpi.get() == "127"):
        for i in range(622,647):
            for j in range(495, 520):
                sum_red += image[i,j,2]
                sum_green += image[i,j,1]
                sum_blue += image[i,j,0]
        sum_red = sum_red/(25*25)
        sum_green = sum_green/(25*25)
        sum_blue = sum_blue/(25*25)
        return sum_red, sum_green, sum_blue
    elif(Globals.doseResponse_dpi.get() == "72"):
        for i in range(352,367):
            for j in range(280,295):
                sum_red+=image[i,j,2]
                sum_green+=image[i,j,1]
                sum_blue+=image[i,j,0]
        sum_red = sum_red/(15*15)
        sum_green = sum_green/(15*15)
        sum_blue = sum_blue/(15*15)
        return sum_red, sum_green, sum_blue
    else:
        messagebox.showerror("Error", \
            "Something has gone wrong with the doseResponse_dpi")
        return False


def plot_dose_response():
#--------------------------------------------------
# Function to plot the dose response curve
#
# The function is called on in delete_lines(),
# avgAllFiles() and clear_all().
#--------------------------------------------------
    sd_red_arr=[];sd_green_arr=[];sd_blue_arr=[]
    temp_dose = [item[0] for item in Globals.avg_red_vector]
    temp_avg_red = [item[1] for item in Globals.avg_red_vector]
    temp_avg_green = [item[1] for item in Globals.avg_green_vector]
    temp_avg_blue = [item[1] for item in Globals.avg_blue_vector]

    for i in range(len(temp_dose)):
        sd_red_arr.append(np.std(Globals.dose_response_sd_list_red[i]))
        sd_green_arr.append(np.std(Globals.dose_response_sd_list_green[i]))
        sd_blue_arr.append(np.std(Globals.dose_response_sd_list_blue[i]))

    if(len(sd_red_arr) > 0):
        Globals.dose_response_sd_avg_red.set(sum(sd_red_arr)/len(sd_red_arr))
        Globals.dose_response_sd_avg_green.set(sum(sd_green_arr)/len(sd_green_arr))
        Globals.dose_response_sd_avg_blue.set(sum(sd_blue_arr)/len(sd_blue_arr))

        Globals.dose_response_sd_max_red.set(max(sd_red_arr))
        Globals.dose_response_sd_max_red_dose.set(str(temp_dose[sd_red_arr.index\
            (Globals.dose_response_sd_max_red.get())]))
        Globals.dose_response_sd_max_green.set(max(sd_green_arr))
        Globals.dose_response_sd_max_green_dose.set(str(temp_dose[sd_green_arr.index\
            (Globals.dose_response_sd_max_green.get())]))
        Globals.dose_response_sd_max_blue.set(max(sd_blue_arr))
        Globals.dose_response_sd_max_blue_dose.set(str(temp_dose[sd_blue_arr.index\
            (Globals.dose_response_sd_max_blue.get())]))

        Globals.dose_response_sd_min_red.set(min(sd_red_arr))
        Globals.dose_response_sd_min_red_dose.set(str(temp_dose[sd_red_arr.index\
            (Globals.dose_response_sd_min_red.get())]))
        Globals.dose_response_sd_min_green.set(min(sd_green_arr))
        Globals.dose_response_sd_min_green_dose.set(str(temp_dose[sd_green_arr.index\
            (Globals.dose_response_sd_min_green.get())]))
        Globals.dose_response_sd_min_blue.set(min(sd_blue_arr))
        Globals.dose_response_sd_min_blue_dose.set(str(temp_dose[sd_blue_arr.index\
            (Globals.dose_response_sd_min_blue.get())]))

    else:
        Globals.dose_response_sd_avg_red.set(0)
        Globals.dose_response_sd_avg_green.set(0)
        Globals.dose_response_sd_avg_blue.set(0)
        Globals.dose_response_sd_max_red.set(0)
        Globals.dose_response_sd_max_red_dose.set('-')
        Globals.dose_response_sd_max_green.set(0)
        Globals.dose_response_sd_max_green_dose.set('-')
        Globals.dose_response_sd_max_blue.set(0)
        Globals.dose_response_sd_max_blue_dose.set('-')
        Globals.dose_response_sd_min_red.set(0)
        Globals.dose_response_sd_min_red_dose.set('-')
        Globals.dose_response_sd_min_green.set(0)
        Globals.dose_response_sd_min_green_dose.set('-')
        Globals.dose_response_sd_min_blue.set(0)
        Globals.dose_response_sd_min_blue_dose.set('-')

    for widget in Globals.dose_response_plot_frame.winfo_children():
        widget.destroy()

    fig = Figure(figsize=(6,4))
    a = fig.add_subplot(111)
    canvas = FigureCanvasTkAgg(fig, master=Globals.dose_response_plot_frame)
    canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, sticky=N+S+E+W,\
         padx=(25,0), pady=(0,0))
    if(Globals.dose_response_var1.get()):
        a.errorbar(temp_dose,temp_avg_red,yerr=sd_red_arr, fmt='ro')
```

```
if(Globals.dose_response_var2.get()):
    a.errorbar(temp_dose, temp_avg_green, yerr=sd_green_arr, fmt='g^')
if(Globals.dose_response_var3.get()):
    a.errorbar(temp_dose, temp_avg_blue, yerr=sd_blue_arr, fmt='bs')

if(len(temp_avg_red) > 3):
    sorted_temp_red = sorted(Globals.avg_red_vector,key=lambda l:l[0])
    sorted_temp_avg_red = [item[1] for item in sorted_temp_red]
    sorted_temp_dose = [item[0] for item in sorted_temp_red]

    sorted_temp_green = sorted(Globals.avg_green_vector, key=lambda l:l[0])
    sorted_temp_avg_green = [item[1] for item in sorted_temp_green]

    sorted_temp_blue = sorted(Globals.avg_blue_vector, key=lambda l:l[0])
    sorted_temp_avg_blue = [item[1] for item in sorted_temp_blue]

    try:
        Globals.popt_red, Globals.pcov_red = curve_fit(fitted_dose_response, \
            sorted_temp_dose, sorted_temp_avg_red, \
                p0=[1700, 15172069, -390], maxfev=10000)
        popt_green, pcov_green = curve_fit(fitted_dose_response, \
            sorted_temp_dose, sorted_temp_avg_green, \
                p0=[1700, 15172069, -390], maxfev=10000)

        try:
            Globals.dose_response_equation_image.destroy()
        except:
            nothingButton()
        perr = np.sqrt(np.diag(Globals.pcov_red))
        perr_prosent = np.array([perr[0]/Globals.popt_red[0], \
            perr[1]/Globals.popt_red[1], perr[2]/Globals.popt_red[2]])*100
        fitting_uncertainty = np.sqrt(perr_prosent[0]**2 + \
            perr_prosent[1]**2 + perr_prosent[2]**2)
        xdata = np.linspace(0,2000,1001)
        ydata_red = np.zeros(len(xdata));ydata_green=np.zeros(len(xdata))
        for i in range(len(xdata)):
            ydata_red[i] = fitted_dose_response(xdata[i], Globals.popt_red[0], \
                Globals.popt_red[1], Globals.popt_red[2])
            ydata_green[i] = fitted_dose_response(xdata[i], popt_green[0], \
                popt_green[1], popt_green[2])
        if(Globals.dose_response_var1.get()):
            a.plot(xdata, ydata_red, color='red')
        if(Globals.dose_response_var2.get()):
            a.plot(xdata, ydata_green, color='green')
        if(Globals.dose_response_var3.get()):
```

```
        a.plot(sorted_temp_dose, sorted_temp_avg_blue , color='blue')

    a.set_title("Dose-response", fontsize=12)
    a.set_ylabel("Pixel value", fontsize=12)
    a.set_xlabel("Dose", fontsize=12)
    fig.tight_layout()

    out_text_function = "Pixel value = " + str(round(Globals.popt_red[0])) \
        + " + " + str(round(Globals.popt_red[1])) + "/(dose - (" \
            + str(round(Globals.popt_red[2])) + "))"
    standardavvik_rgb = "Standard deviation red = " + \
        str(round(Globals.dose_response_sd_avg_red.get()))
    fitting_inc = str(round(fitting_uncertainty))
    def drawEquation(a,b,c):
        tmptext = StringVar()
        if c < 0:
            c = abs(c)
            a=str(a);b=str(b);c=str(c)
            latex= a   + "+ " "\\frac {" + f"{b}" + "}{"+ "D" + "+" \
                + f"{c}" + "}"
        else:
            a=str(a);b=str(b);c=str(c)
            latex= a   + "+ " "\\frac {" + f"{b}" + "}{"+ "D" + "-" \
                + f"{c}" + "}"

        tmptext.set(latex)
        tmptext = "$"+tmptext.get()+"$"

        axLatex.clear()
        axLatex.text(0.01,0.5, "PV = "+tmptext, fontsize = 6)
        canvasLatex.draw()

    labelLatex = Label(Globals.dose_response_equation_frame)
    labelLatex.grid(row=0,column=0, rowspan=15, sticky=N+W)
    labelLatex.config(bg='#ffffff', bd=0, highlightthickness=0)



    figLatex = matplotlib.figure.Figure(figsize=(1.2, 1), dpi=250)
    figLatex.subplots_adjust(bottom=-0.01, top=1.2, left=-0.01, right=2)
    axLatex = figLatex.add_subplot(111)

    canvasLatex = FigureCanvasTkAgg(figLatex, master=labelLatex)
    canvasLatex.get_tk_widget().grid(row=0, column=0, sticky="N")
    canvasLatex._tkcanvas.grid(row=0, column=0,sticky="N")
```

```
axLatex.get_xaxis().set_visible(False)
axLatex.get_yaxis().set_visible(False)

a_=round(Globals.popt_red[0])
b_=round(Globals.popt_red[1])
c_=round(Globals.popt_red[2])
drawEquation(a_,b_,c_)


def drawSD(sd_text,sd_text_1,sd_text_2,sd_text_3,sd_text_4,sd_text_5,\
    sd_text_6,sd_text_7,sd_text_8,sd_text_9,sd_text_10,sd_text_11,\
    sd_text_12,sd_text_13,sd_text_14,sd_text_15,sd_text_16,sd_text_17):

    text = "Standard deviations (SD): "
    sd_text.config(state=NORMAL)
    sd_text.insert(INSERT, text)
    sd_text.config(state=DISABLED)

    sd_text_1.config(state=NORMAL)
    sd_text_1.insert(INSERT, " ")
    sd_text_1.config(state=DISABLED)

    sd_text_2.config(state=NORMAL)
    sd_text_2.insert(INSERT, "Average")
    sd_text_2.config(state=DISABLED)

    sd_text_3.config(state=NORMAL)
    sd_text_3.insert(INSERT, "Minimum")
    sd_text_3.config(state=DISABLED)

    sd_text_4.config(state=NORMAL)
    sd_text_4.insert(INSERT, "Maximum")
    sd_text_4.config(state=DISABLED)

    sd_text_5.config(state=NORMAL)
    sd_text_5.insert(INSERT, "Red:")
    sd_text_5.config(state=DISABLED)

    avgR=str(round(Globals.dose_response_sd_avg_red.get()))
    sd_text_6.config(state=NORMAL)
    sd_text_6.insert(INSERT, avgR)
    sd_text_6.config(state=DISABLED)

    minR=str(round(Globals.dose_response_sd_min_red.get()))
    sd_text_7.config(state=NORMAL)
```

```
sd_text_7.insert(INSERT, minR)
sd_text_7.config(state=DISABLED)

maxR=str(round(Globals.dose_response_sd_max_red.get()))
sd_text_8.config(state=NORMAL)
sd_text_8.insert(INSERT, maxR)
sd_text_8.config(state=DISABLED)

sd_text_9.config(state=NORMAL)
sd_text_9.insert(INSERT, "Green:")
sd_text_9.config(state=DISABLED)

avgG=str(round(Globals.dose_response_sd_avg_green.get()))
sd_text_10.config(state=NORMAL)
sd_text_10.insert(INSERT, avgG)
sd_text_10.config(state=DISABLED)

minG=str(round(Globals.dose_response_sd_min_green.get()))
sd_text_11.config(state=NORMAL)
sd_text_11.insert(INSERT, minG)
sd_text_11.config(state=DISABLED)

maxG=str(round(Globals.dose_response_sd_max_green.get()))
sd_text_12.config(state=NORMAL)
sd_text_12.insert(INSERT, maxG)
sd_text_12.config(state=DISABLED)

sd_text_13.config(state=NORMAL)
sd_text_13.insert(INSERT, "Blue:")
sd_text_13.config(state=DISABLED)

avgB=str(round(Globals.dose_response_sd_avg_blue.get()))
sd_text_14.config(state=NORMAL)
sd_text_14.insert(INSERT, avgB)
sd_text_14.config(state=DISABLED)

minB=str(round(Globals.dose_response_sd_min_blue.get()))
sd_text_15.config(state=NORMAL)
sd_text_15.insert(INSERT, minB)
sd_text_15.config(state=DISABLED)

maxB=str(round(Globals.dose_response_sd_max_blue.get()))
sd_text_16.config(state=NORMAL)
sd_text_16.insert(INSERT, maxB)
sd_text_16.config(state=DISABLED)
```

```
    latex_fitting="Fitting uncertainty of the plot: " + fitting_inc
    out = str(latex_fitting)
    sd_text_17.config(state=NORMAL)
    sd_text_17.insert(INSERT, out)
    sd_text_17.config(state=DISABLED)

sd_text = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=40)
sd_text.grid(row=0, column=1, columnspan=4, sticky=N+S+W+E)
sd_text.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_1 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=4)
sd_text_1.grid(row=1, column=1, sticky=N+S+W+E)
sd_text_1.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_2 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=1)
sd_text_2.grid(row=1, column=2, sticky=N+S+W+E)
sd_text_2.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_3 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=1)
sd_text_3.grid(row=1, column=3, sticky=N+S+W+E)
sd_text_3.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_4 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=1)
sd_text_4.grid(row=1, column=4, sticky=N+S+W+E)
sd_text_4.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_5 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=4)
sd_text_5.grid(row=2, column=1, sticky=N+S+W+E)
sd_text_5.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_6 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=1)
sd_text_6.grid(row=2, column=2, sticky=N+S+W+E)
sd_text_6.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_7 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=1)
sd_text_7.grid(row=2, column=3, sticky=N+S+W+E)
sd_text_7.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_8 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=1)
sd_text_8.grid(row=2, column=4, sticky=N+S+W+E)
sd_text_8.config(state=DISABLED, bd=0, font=('calibri', '12'))
sd_text_9 = tk.Text(Globals.dose_response_equation_frame, \
    height=1, width=4)
```

```
            sd_text_9.grid(row=3, column=1, sticky=N+S+W+E)
            sd_text_9.config(state=DISABLED, bd=0, font=('calibri', '12'))
            sd_text_10 = tk.Text(Globals.dose_response_equation_frame, \
                height=1, width=1)
            sd_text_10.grid(row=3, column=2, sticky=N+S+W+E)
            sd_text_10.config(state=DISABLED, bd=0, font=('calibri', '12'))
            sd_text_11 = tk.Text(Globals.dose_response_equation_frame, \
                height=1, width=1)
            sd_text_11.grid(row=3, column=3, sticky=N+S+W+E)
            sd_text_11.config(state=DISABLED, bd=0, font=('calibri', '12'))
            sd_text_12 = tk.Text(Globals.dose_response_equation_frame, \
                height=1, width=1)
            sd_text_12.grid(row=3, column=4, sticky=N+S+W+E)
            sd_text_12.config(state=DISABLED, bd=0, font=('calibri', '12'))
            sd_text_13 = tk.Text(Globals.dose_response_equation_frame, \
                height=1, width=4)
            sd_text_13.grid(row=4, column=1, sticky=N+S+W+E)
            sd_text_13.config(state=DISABLED, bd=0, font=('calibri', '12'))
            sd_text_14 = tk.Text(Globals.dose_response_equation_frame, \
                height=1, width=1)
            sd_text_14.grid(row=4, column=2, sticky=N+S+W+E)
            sd_text_14.config(state=DISABLED, bd=0, font=('calibri', '12'))
            sd_text_15 = tk.Text(Globals.dose_response_equation_frame, \
                height=1, width=1)
            sd_text_15.grid(row=4, column=3, sticky=N+S+W+E)
            sd_text_15.config(state=DISABLED, bd=0, font=('calibri', '12'))
            sd_text_16 = tk.Text(Globals.dose_response_equation_frame, \
                height=1, width=1)
            sd_text_16.grid(row=4, column=4, sticky=N+S+W+E)
            sd_text_16.config(state=DISABLED, bd=0, font=('calibri', '12'))
            sd_text_17 = tk.Text(Globals.dose_response_equation_frame, \
                height=1, width=40)
            sd_text_17.grid(row=5, column=1, columnspan=4, sticky=N+S+W+E)
            sd_text_17.config(state=DISABLED, bd=0, font=('calibri', '12'))
            drawSD(sd_text,sd_text_1,sd_text_2,sd_text_3,sd_text_4,sd_text_5,\
                sd_text_6,sd_text_7,sd_text_8,sd_text_9,sd_text_10,sd_text_11,\
                sd_text_12,sd_text_13,sd_text_14,sd_text_15,sd_text_16,sd_text_17)

            Globals.dose_response_save_calibration_button.config(state=ACTIVE)
        except OptimizeWarning:
            messagebox.showwarning("Warning", \
"It appears that you have optimization problems. \
Try adding more data points to improve the optimization.\
 Or, check that your specified dose matches your uploaded files.")
        except RuntimeError:
            messagebox.showwarning("Warning", \
"It appears that you have optimization problems. \
Try adding more data points to improve the optimization. \
Or, check that your specified dose matches your uploaded files.")

    a.set_title("Dose-response", fontsize=12)
    a.set_ylabel("Pixel value", fontsize=12)
    a.set_xlabel("Dose", fontsize=12)
    fig.tight_layout()


def delete_line(delete_button):
#---------------------------------------------------
# Function to delete a measurement
#
# The function is a callback funtion to the
# button delete_button defined in function
# avgAllFiles()
#---------------------------------------------------
    button_index = Globals.dose_response_delete_buttons.index(delete_button)
    Globals.dose_response_red_list[button_index].destroy()
    Globals.dose_response_green_list[button_index].destroy()
    Globals.dose_response_blue_list[button_index].destroy()
    Globals.dose_response_dose_list[button_index].destroy()
    Globals.dose_response_delete_buttons[button_index].destroy()
    del(Globals.dose_response_red_list[button_index])
    del(Globals.dose_response_green_list[button_index])
    del(Globals.dose_response_blue_list[button_index])
    del(Globals.dose_response_dose_list[button_index])

    if(len(Globals.dose_response_delete_buttons) > 1):
        del(Globals.avg_red_vector[button_index])
        del(Globals.avg_green_vector[button_index])
        del(Globals.avg_blue_vector[button_index])
        del(Globals.dose_response_delete_buttons[button_index])
        del(Globals.dose_response_sd_list_red[button_index])
        del(Globals.dose_response_sd_list_green[button_index])
        del(Globals.dose_response_sd_list_blue[button_index])
    else:
        Globals.avg_red_vector = []
        Globals.avg_green_vector = []
        Globals.avg_blue_vector = []
        Globals.dose_response_delete_buttons = []
        Globals.dose_response_sd_list_red = []
        Globals.dose_response_sd_list_green = []
        Globals.dose_response_sd_list_blue = []

    Globals.dose_response_files_row_count = 2
    for i in range(len(Globals.dose_response_delete_buttons)):
        Globals.dose_response_red_list[i].grid\
            (row=Globals.dose_response_files_row_count, column=1, \
                sticky=N+S+W+E, padx=(0,0))
        Globals.dose_response_green_list[i].grid\
            (row=Globals.dose_response_files_row_count, column=3, \
                sticky=N+S+W+E, padx=(0,0))
        Globals.dose_response_blue_list[i].grid\
            (row=Globals.dose_response_files_row_count, column=5, \
                sticky=N+S+W+E, padx=(0,5))
        Globals.dose_response_dose_list[i].grid\
            (row=Globals.dose_response_files_row_count, column=0, \
                sticky=N+S+W+E, padx=(0,15))
        Globals.dose_response_delete_buttons[i].grid\
            (row=Globals.dose_response_files_row_count, column=7, \
                sticky=N+S+W+E, padx=(5,5))
        Globals.dose_response_files_row_count+=1

    if(len(Globals.dose_response_delete_buttons) < 4):
        Globals.dose_response_save_calibration_button.config(state=DISABLED)

    plot_dose_response()

def fitted_dose_response(D, a, b, c):
#---------------------------------------------
# Function to map between pixel value and dose
#
# The function is a called in the funtion
# plot_dose_response()
#---------------------------------------------
    return a + b/(D-c)


def avgAllFiles(write_dose_box, new_window):
#---------------------------------------------
# Function to find the mean of uploaded
# images with same dose.
#
# The function is a callback function for
# the button done_button in create_window()
#---------------------------------------------
    dose_input = write_dose_box.get("1.0",'end-1c')
```

```
if (dose_input == " "):
    messagebox.showerror("Error", "Input dose")
    return
try:
    dose_input = float(dose_input)
except:
    messagebox.showerror("Error","The dose must be a number")
    return
if(len(Globals.dose_response_uploaded_filenames) == 0):
    messagebox.showerror("Error", "No files uploaded")
    return

try:
    Globals.upload_files_here_canvas.destroy()
except:
    nothingButton()

avg_red=0;avg_green=0;avg_blue=0
red_temp_sd_list = []; green_temp_sd_list = []; blue_temp_sd_list = []
for i in range(0, len(Globals.dose_response_uploaded_filenames)):
    if(readImage(Globals.dose_response_uploaded_filenames[i])==False):
        messagebox.showerror("Error", "A mistake has happend in readImage()")
        return
    red, green, blue = readImage(Globals.dose_response_uploaded_filenames[i])
    avg_red+=red
    avg_green+=green
    avg_blue+=blue

    red_temp_sd_list.append(red)
    green_temp_sd_list.append(green)
    blue_temp_sd_list.append(blue)



avg_red = avg_red/len(Globals.dose_response_uploaded_filenames)
avg_green = avg_green/len(Globals.dose_response_uploaded_filenames)
avg_blue = avg_blue/len(Globals.dose_response_uploaded_filenames)
temp_dose = [item[0] for item in Globals.avg_red_vector]
isTest = False
try:
    indx = temp_dose.index(dose_input)
    Globals.avg_red_vector[indx][1] = (avg_red + \
        Globals.avg_red_vector[indx][1])/2
    Globals.avg_green_vector[indx][1] = (avg_green + \
        Globals.avg_green_vector[indx][1])/2
    Globals.avg_blue_vector[indx][1] = (avg_blue + \
        Globals.avg_blue_vector[indx][1])/2

    for i in range(0, len(red_temp_sd_list)):
        Globals.dose_response_sd_list_red[indx].append(red_temp_sd_list[i])
        Globals.dose_response_sd_list_green[indx].append(green_temp_sd_list[i])
        Globals.dose_response_sd_list_blue[indx].append(blue_temp_sd_list[i])

except:
    Globals.avg_red_vector.append([dose_input, avg_red])
    Globals.avg_green_vector.append([dose_input, avg_green])
    Globals.avg_blue_vector.append([dose_input, avg_blue])

    Globals.dose_response_sd_list_red.append(red_temp_sd_list)
    Globals.dose_response_sd_list_green.append(green_temp_sd_list)
    Globals.dose_response_sd_list_blue.append(blue_temp_sd_list)

    isTest = True

temp_dose = [item[0] for item in Globals.avg_red_vector]

if(isTest):
    result_red = tk.Text(Globals.tab2_canvas_files, height=1, width=7)
    result_red.insert(INSERT, round(avg_red))
    result_red.grid(row=Globals.dose_response_files_row_count, column=1, \
        sticky=N+S+W+E, padx=(0,0))
    Globals.tab2_canvas_files.grid_columnconfigure\
        (Globals.dose_response_files_weightcount, weight=0)
    Globals.tab2_canvas_files.grid_rowconfigure\
        (Globals.dose_response_files_weightcount, weight=0)
    result_red.config(state=DISABLED, bd=0, font=('calibri', '12'))
    Globals.dose_response_red_list.append(result_red)
    Globals.dose_response_files_weightcount+=1

    result_green = tk.Text(Globals.tab2_canvas_files, height=1, width=7)
    result_green.insert(INSERT, round(avg_green))
    result_green.grid(row=Globals.dose_response_files_row_count, column=3, \
        sticky=N+S+W+E, padx=(0,0))
    Globals.tab2_canvas_files.grid_columnconfigure\
        (Globals.dose_response_files_weightcount, weight=0)
    Globals.tab2_canvas_files.grid_rowconfigure\
        (Globals.dose_response_files_weightcount, weight=0)
    result_green.config(state=DISABLED, bd=0, font=('calibri', '12'))
    Globals.dose_response_green_list.append(result_green)
    Globals.dose_response_files_weightcount+=1
```

```
result_blue = tk.Text(Globals.tab2_canvas_files, height=1, width=7)
result_blue.insert(INSERT, round(avg_blue))
result_blue.grid(row=Globals.dose_response_files_row_count, column=5, \
    sticky=N+S+W+E, padx=(0,5))
Globals.tab2_canvas_files.grid_columnconfigure\
    (Globals.dose_response_files_weightcount, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure\
    (Globals.dose_response_files_weightcount, weight=0)
result_blue.config(state=DISABLED, bd=0, font=('calibri', '12'))
Globals.dose_response_blue_list.append(result_blue)
Globals.dose_response_files_weightcount+=1

dose_print = tk.Text(Globals.tab2_canvas_files, height=1, width=10)
dose_print.insert(INSERT, dose_input)
dose_print.grid(row=Globals.dose_response_files_row_count, column=0, \
    sticky=N+S+W+E, padx=(0,15))
Globals.tab2_canvas_files.grid_columnconfigure\
    (Globals.dose_response_files_weightcount, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure\
    (Globals.dose_response_files_weightcount, weight=0)
dose_print.config(state=DISABLED, bd=0, font=('calibri', '12'))
Globals.dose_response_dose_list.append(dose_print)
Globals.dose_response_files_weightcount+=1

path = os.path.dirname(sys.argv[0])
path = path + r"\delete.png"
img = ImageTk.PhotoImage(file=path)

delete_button = tk.Button(Globals.tab2_canvas_files, text='Remove', \
    image=img, cursor='hand2',font=('calibri', '18'),\
        highlightthickness= 0, relief=FLAT, state=ACTIVE, width = 15)
delete_button.image = img
Globals.dose_response_delete_buttons.append(delete_button)
delete_button.config(command=lambda: delete_line(delete_button))
delete_button.grid(row=Globals.dose_response_files_row_count, column=7, \
    sticky=N+S+W+E, padx=(5,5))
Globals.tab2_canvas_files.grid_columnconfigure\
    (Globals.dose_response_files_weightcount, weight=0)
Globals.tab2_canvas_files.grid_rowconfigure\
    (Globals.dose_response_files_weightcount, weight=0)
delete_button.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
Globals.dose_response_files_row_count+=1
Globals.dose_response_files_weightcount+=1
```

```
    else:
        Globals.dose_response_red_list[indx].config(state=NORMAL)
        Globals.dose_response_red_list[indx].delete('1.0', END)
        Globals.dose_response_red_list[indx].insert(INSERT, \
            round(Globals.avg_red_vector[indx][1]))
        Globals.dose_response_red_list[indx].config(state=DISABLED)

        Globals.dose_response_green_list[indx].config(state=NORMAL)
        Globals.dose_response_green_list[indx].delete('1.0', END)
        Globals.dose_response_green_list[indx].insert(INSERT, \
            round(Globals.avg_green_vector[indx][1]))
        Globals.dose_response_green_list[indx].config(state=DISABLED)

        Globals.dose_response_blue_list[indx].config(state=NORMAL)
        Globals.dose_response_blue_list[indx].delete('1.0', END)
        Globals.dose_response_blue_list[indx].insert(INSERT, \
            round(Globals.avg_blue_vector[indx][1]))
        Globals.dose_response_blue_list[indx].config(state=DISABLED)

    plot_dose_response()
    new_window.destroy()

def create_window():
#---------------------------------------------------
# Function to create the window where the user
# upload the files of scanned film
#
# The function is a callback function for
# the button dose_response_upload_button
# in notebook.py
#---------------------------------------------------
    new_window = tk.Toplevel(Globals.tab2)
    new_window.geometry("360x500")
    new_window.grab_set()

    new_window_frame = tk.Frame(new_window)
    new_window_frame.config(relief=FLAT, bg='#ffffff', highlightthickness=0)

    new_window_scroll_canvas = tk.Canvas(new_window_frame)
    new_window_scroll_canvas.config(bg='#ffffff', height=450, \
        width=200,highlightthickness=0, relief=FLAT)
    new_window_scroll_canvas.grid_propagate(0)

    new_window_scroll = Scrollbar(new_window_frame, \
        command=new_window_scroll_canvas.yview)
    scrollable_frame= tk.Frame(new_window_scroll_canvas)
    scrollable_frame.config(highlightthickness=0, relief=FLAT)

    scrollable_frame.bind("<Configure>", lambda e: \
        new_window_scroll_canvas.configure\
            (scrollregion=new_window_scroll_canvas.bbox('all')))
    new_window_scroll_canvas.create_window\
        ((0,0), window=scrollable_frame, anchor='nw')
    new_window_scroll_canvas.configure(yscrollcommand=new_window_scroll.set)

    new_window_canvas = tk.Canvas(scrollable_frame)
    new_window_canvas.config(relief=FLAT, bg='#ffffff', highlightthickness=0)
    new_window_canvas.pack(fill=BOTH, expand=True)

    new_window_frame.pack(expand=True, fill = BOTH)
    new_window_scroll_canvas.pack(side=LEFT, fill=BOTH, expand=True)
    new_window_scroll.pack(side=RIGHT, fill=Y)


    Globals.dose_response_uploaded_filenames = []

    explain_text = tk.Text(new_window_canvas, height=11, width = 47)
    explain_text.grid(row=0, column = 0, rowspan = 3, columnspan=2, \
        sticky=N+S+W+E, pady=(20,10), padx=(10,10))
    new_window_canvas.grid_columnconfigure(0, weight=0)
    new_window_canvas.grid_rowconfigure(0, weight=0)
    explain_text.insert(INSERT, "\
Here you can upload several files all irradiated with \nthe same dose. \
Fill in dose and an average will be \ncalculated and used in the calibration. \
You are also \nable to upload only one file each time, and FIDORA \nwill keep \
track and average before fitting the \ndose-response.")
    explain_text.config(state=DISABLED, bd=0, font=('calibri', '11'))

    write_dose_box_frame = tk.Frame(new_window_canvas)
    write_dose_box_frame.grid(row=2, column=1, sticky=N+S+E+W,\
         pady=(0,30), padx=(0,10))
    new_window_canvas.grid_columnconfigure(1, weight=0)
    new_window_canvas.grid_rowconfigure(1, weight=0)
    write_dose_box_frame.config(bg='#ffffff')

    dose_border_label = Label(write_dose_box_frame, \
        image = Globals.dose_response_dose_border)
    dose_border_label.image=Globals.dose_response_dose_border
    dose_border_label.config(bg='#ffffff', borderwidth=0)
    dose_border_label.pack(expand=True, fill=BOTH)

    write_dose_text = tk.Text(new_window_canvas, height=1, width=19)
    write_dose_text.insert(INSERT, "Write dose here (cGy):")
    write_dose_text.config(state=DISABLED, bd=0, \
        font=('calibri', '11'), bg='#ffffff')
    write_dose_text.grid(row=1, column=1, sticky=E+W, pady=(140,0), padx=(5,5))
    new_window_canvas.grid_columnconfigure(3, weight=0)
    new_window_canvas.grid_rowconfigure(3, weight=0)

    write_dose_box = tk.Text(dose_border_label, height=1, width=8)
    write_dose_box.grid(row=0,column=0, sticky=N+S+W+E, pady=(10,0), padx=(20,5))
    write_dose_box.insert(INSERT, " ")
    write_dose_box.config(state=NORMAL, bd=0, font=('calibri', '18'), bg='#ffffff')

    upload_button_frame = tk.Frame(new_window_canvas)
    upload_button_frame.grid(row=2, column=0, sticky=N+S+W+E, pady=(0,30))
    new_window_canvas.grid_columnconfigure(2, weight=0)
    new_window_canvas.grid_rowconfigure(2, weight=0)
    upload_button_frame.config(bg='#ffffff')

    upload_button = tk.Button(upload_button_frame, text='Upload file', \
        image=Globals.upload_button_image, cursor='hand2', font=('calibri', '14'), \
        relief=FLAT, state=ACTIVE,command=lambda: UploadAction(new_window_canvas))
    upload_button.pack(expand=True, fill=BOTH)
    upload_button.config(bg='#ffffff', activebackground='#ffffff', \
        activeforeground='#ffffff', highlightthickness=0)
    upload_button.image=Globals.upload_button_image

    Globals.dose_response_inOrOut = True
    done_button = tk.Button(new_window, text='DONE', cursor='hand2', \
        font=('calibri', '20', 'bold'),relief=FLAT, state=ACTIVE,\
            command=lambda: avgAllFiles(write_dose_box, new_window))
    done_button.config(activebackground='#04BAA6', bg= '#04BAA6', \
        activeforeground='#ffffff', fg='#ffffff', height=1)
    done_button.pack(expand=True, fill=X)




def clear_all():
#-----------------------------------------------------
# Function to clear are variables
#
# The function is a callback function for
# the button dose_response_clear_all_button
# in notebook.py
#-----------------------------------------------------
    for i in range(len(Globals.dose_response_delete_buttons)):
        Globals.dose_response_red_list[i].destroy()
        Globals.dose_response_green_list[i].destroy()
        Globals.dose_response_blue_list[i].destroy()
        Globals.dose_response_delete_buttons[i].destroy()
        Globals.dose_response_dose_list[i].destroy()

    Globals.dose_response_dose_list = []
    Globals.dose_response_red_list = []
    Globals.dose_response_green_list = []
    Globals.dose_response_blue_list = []
    Globals.dose_response_delete_buttons = []

    Globals.dose_response_sd_list_red = []
    Globals.dose_response_sd_list_green = []
    Globals.dose_response_sd_list_blue = []

    Globals.dose_response_var1.set(1)
    Globals.dose_response_var2.set(1)
    Globals.dose_response_var3.set(1)

    Globals.avg_red_vector = []
    Globals.avg_green_vector = []
    Globals.avg_blue_vector = []

    Globals.dose_response_batch_number = "-"
    Globals.popt_red = np.zeros(3)
    Globals.dose_response_inOrOut = True
    Globals.dose_response_files_weightcount = 8
    Globals.dose_response_files_row_count = 2

    Globals.dose_response_save_calibration_button.config(state=DISABLED)

    plot_dose_response()
```

Profile_functions.py

```
#-----------------------------------------------------------------------------------------------
#
# Version 17.08.20
#
```

```
# Fuctions used in relation to the tab Profiles
#-----------------------------------------------------------------------------

import Globals
import tkinter as tk
from tkinter import filedialog, INSERT, DISABLED, messagebox, NORMAL, simpledialog,\
    PhotoImage, BOTH, Canvas, N, S, W, E, ALL, Frame, SUNKEN, Radiobutton, \
        GROOVE, ACTIVE, FLAT, END, Scrollbar, HORIZONTAL, VERTICAL, \
            ttk, TOP, RIGHT, LEFT, ttk
import os
from os.path import normpath, basename
from PIL import Image, ImageTk
import cv2
from cv2 import imread, IMREAD_ANYCOLOR, IMREAD_ANYDEPTH, imwrite
import pydicom
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import \
    FigureCanvasTkAgg,  NavigationToolbar2Tk
import numpy as np

def nothing_function():
#---------------------------------------------
# Function to only return
# Is used in cases where nothing should happen in
# a active button (they will later be reconfigured)
#---------------------------------------------
    return


def clearAll():
#---------------------------------------------
# Function to clear all variables
#
# This functions is a callback to the button
# profiles_resetAll_button in notebook.py
#---------------------------------------------
    Globals.profiles_film_orientation.set('-')
    Globals.profiles_film_orientation_menu.config(state=ACTIVE, \
        bg = '#ffffff', width=15, relief=FLAT)

    Globals.profiles_film_dataset_red_channel_dose = None
    Globals.profiles_film_variable_ROI_coords = None

    Globals.profiles_film_dataset_ROI = None
    Globals.profiles_film_dataset_ROI_red_channel = None
    Globals.profiles_doseplan_dataset_ROI = None
    Globals.profiles_film_dataset_ROI_red_channel_dose = None

    profiles_fig = Figure(figsize=(6,4))
    profiles_a = profiles_fig.add_subplot(111, \
        ylim=(0,40000), xlim=(0,500))
    profiles_plot_canvas = FigureCanvasTkAgg(profiles_fig, \
        master=Globals.profile_plot_canvas)
    profiles_plot_canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, \
        rowspan=4, sticky=N+E+W+S, padx=(5,0), pady=(0,0))
    profiles_a.set_title ("Profiles", fontsize=12)
    profiles_a.set_ylabel("Dose (Gy)", fontsize=12)
    profiles_a.set_xlabel("Distance (mm)", fontsize=12)
    profiles_fig.tight_layout()

    Globals.profiles_depth = None
    Globals.profiles_depth_float = None
    Globals.profiles_film_factor_input = None

    Globals.profiles_distance_reference_point_ROI = []

    Globals.profiles_mark_isocenter_up_down_line = []
    Globals.profiles_mark_isocenter_right_left_line = []
    Globals.profiles_mark_isocenter_oval = []
    Globals.profiles_mark_ROI_rectangle = []

    Globals.profiles_isocenter_check=False
    Globals.profiles_reference_point_check = False
    Globals.profiles_ROI_check = False
    Globals.profiles_ROI_reference_point_check = False

    Globals.profiles_film_batch.set(0)

    Globals.profiles_popt_red = np.zeros(3)

    Globals.profiles_upload_button_doseplan.config(state=DISABLED)
    Globals.profiles_upload_button_film.config(state=ACTIVE)
    Globals.profiles_upload_button_rtplan.config(state=DISABLED)

    Globals.profiles_dataset_doseplan = None
    Globals.profiles_dataset_rtplan = None
    Globals.profiles_test_if_added_doseplan = False
    Globals.profiles_test_if_added_rtplan = False

    Globals.profiles_isocenter_mm = None

    Globals.profiles_dose_scaling_doseplan = []

    Globals.profiles_max_dose_film = None

    Globals.profiles_choice_of_profile_line_type.set("h")

    for widget in Globals.profiles_film_panedwindow.winfo_children():
        widget.destroy()

    Globals.doseplan_write_image = None
    Globals.film_write_image = None
    Globals.doseplan_write_image_width = None
    Globals.doseplan_write_image_height = None
    Globals.doseplan_write_image_var_x= 0
    Globals.doseplan_write_image_var_y = 0
    Globals.profiles_coordinate_in_dataset = 0

    Globals.profiles_first_time_in_drawProfiles = True

    Globals.profiles_doseplan_lateral_displacement = None
    Globals.profiles_doseplan_vertical_displacement = None
    Globals.profiles_doseplan_longitudianl_displacement = None
    Globals.profiles_doseplan_patient_position = None

    Globals.profiles_input_lateral_displacement = None
    Globals.profiles_input_longitudinal_displacement = None
    Globals.profiles_input_vertical_displacement = None

    Globals.profiles_isocenter_or_reference_point = None

    Globals.profiles_lateral = None
    Globals.profiles_vertical = None
    Globals.profiles_longitudinal = None

    Globals.profiles_number_of_doseplans = 0
    Globals.profiles_number_of_doseplans_row_count = 4
    Globals.profiles_doseplans_grid_config_count = 6
    Globals.profiles_doseplans_filenames = []
    Globals.profiles_doseplans_factor_text = []
```

```
    Globals.profiles_doseplans_factor_input = []

    Globals.profiles_reference_point_in_doseplan = None

    Globals.profiles_doseplan_dataset_ROI_several = []
    Globals.profiles_several_img = []

    Globals.profiles_film_factor = None

    Globals.profiles_lines = []

    Globals.end_point = None

    Globals.profiles_line_coords_film = None
    Globals.profiles_line_coords_doseplan = None

    Globals.profiles_dataset_film_variable_draw = None
    Globals.profiles_dataset_doesplan_variable_draw = None

    Globals.max_dose_doseplan = None

    Globals.profiles_iscoenter_coords = []
    Globals.profiles_film_isocenter = None
    Globals.profiles_film_reference_point = None
    Globals.profiles_mark_isocenter_up_down_line = []
    Globals.profiles_mark_isocenter_right_left_line = []
    Globals.profiles_mark_isocenter_oval = []
    Globals.profiles_mark_reference_point_oval = []
    Globals.profiles_mark_ROI_rectangle = []
    Globals.profiles_ROI_coords = []

    Globals.profiles_isocenter_check = False
    Globals.profiles_ROI_check = False
    Globals.profiles_reference_point_check = False
    Globals.profiles_ROI_reference_point_check = False

    Globals.profiles_upload_button_film.config(state=ACTIVE)
    Globals.profiles_upload_button_doseplan.config(state=DISABLED)
    Globals.profiles_upload_button_rtplan.config(state=DISABLED)

    Globals.profiles_distance_isocenter_ROI = []

    Globals.profiles_film_dataset = None
    Globals.profiles_film_dataset_red_channel = None
    Globals.profiles_film_dataset_ROI = None
    Globals.profiles_film_dataset_ROI_red_channel = None

    Globals.profiles_film_match_isocenter_dataset = np.zeros((7,7))

    Globals.profiles_dataset_doseplan = None
    Globals.profiles_dataset_rtplan = None
    Globals.profiles_isocenter_mm = None
    Globals.profiles_test_if_added_rtplan = False
    Globals.profiles_test_if_added_doseplan = False

    Globals.profiles_export_plot_button.config(state=DISABLED)

    Globals.profiles_slice_offset = None
    Globals.profiles_offset = None

    Globals.tab4_canvas.unbind("<Up>")
    Globals.tab4_canvas.unbind("<Down>")
    return

def getCoordsInRandomLine(x1,y1,x2,y2):
#---------------------------------------------------------------
# Function to calculate the coordinates
# in a random line
#
# This functions is called when drawing
# manual profiles in DrawProfile()
#---------------------------------------------------------------
    points = []
    issteep = abs(y2-y1) - abs(x2-x1)
    if issteep > 0:
        x1, y1 = y1, x1
        x2, y2 = y2, x2
    rev = False
    if x1 > x2:
        x1, x2 = x2, x1
        y1, y2 = y2, y1
        rev = True
    deltax = x2 - x1
    deltay = abs(y2-y1)
    error = int(deltax / 2)
    y = y1
    ystep = None
    if y1 < y2:
        ystep = 1
    else:
        ystep = −1
    for x in range(x1, x2 + 1):
        if issteep:
            points.append((y, x))
        else:
            points.append((x, y))
        error −= deltay
        if error < 0:
            y += ystep
            error += deltax
    if rev:
        points.reverse()

    return points


def drawProfiles(even):
#−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
# Function to draw profiles
#
# This functions is a called at almost all
# functions in Profil_functions
#−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    if Globals.profiles_choice_of_profile_line_type.get() == 'h' \
        or Globals.profiles_choice_of_profile_line_type.get() == 'v':
        Globals.profiles_lines = []

    if Globals.profiles_dataset_doseplan == None:
        return

    Globals.profiles_adjust_button_right.config(state=ACTIVE)
    Globals.profiles_adjust_button_left.config(state=ACTIVE)
    Globals.profiles_adjust_button_down.config(state=ACTIVE)
    Globals.profiles_adjust_button_up.config(state=ACTIVE)
    Globals.profiles_adjust_button_return.config(state=ACTIVE)



    def draw(line_orient, dataset_film, dataset_doseplan):
        Globals.profile_plot_canvas.delete('all')
        fig= Figure(figsize=(6,4))
        a = fig.add_subplot(111)
        plot_canvas = FigureCanvasTkAgg(fig, master=Globals.profile_plot_canvas)
        plot_canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, \
            rowspan=5, sticky=N+E+W+S, padx=(5,0), pady=(0,0))
        cols = \
        (' ', 'Point match', 'Distance', 'Dose', 'Rel. to max', 'Rel. to target')
        listBox = \
        ttk.Treeview(Globals.profile_plot_canvas, columns=cols, show='headings')
        for col in cols:
            listBox.heading(col, text=col, anchor=W)
            listBox.column(col ,width=84, stretch=False, anchor=W)
        listBox.grid(row=8, column=0, columnspan=4)
        lst = [['Film: ', ' ', ' ', ' ', ' ', ' '],\
            ['Doseplan: ', ' ', ' ', ' ', ' ', ' ']]
        for i, (name, m, dis, d, rdROI, rdTarget) in enumerate(lst):
            listBox.insert("", "end", values=(name, m, dis, d, rdROI, rdTarget))
        v_line = a.axvline(x=0, ymin=0, ymax=50, c='gray')



        if line_orient == 'h':
            if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
                dy = Globals.profiles_doseplan_dataset_ROI.shape[1]/2
            elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
                dy = Globals.profiles_doseplan_dataset_ROI.shape[1]*2/2
            else:
                dy = Globals.profiles_doseplan_dataset_ROI.shape[1]*3/2
            dx = dataset_film.shape[1]*0.2/2
            x = np.linspace(-dx,dx, dataset_film.shape[1])
            y = np.linspace(-dy,dy, Globals.profiles_doseplan_dataset_ROI.shape[1])
            plot_film = dataset_film[Globals.profiles_coordinate_in_dataset,:]/100
            plot_doseplan = \
                dataset_doseplan[Globals.profiles_coordinate_in_dataset, :]
            film = a.plot(x,plot_film, color='r', label='Film')
            dose = a.plot(y,plot_doseplan, color='b', label='Doseplan')
        elif line_orient == 'v':
            if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
                dy = Globals.profiles_doseplan_dataset_ROI.shape[0]/2
            elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
                dy = Globals.profiles_doseplan_dataset_ROI.shape[0]*2/2
            else:
                dy = Globals.profiles_doseplan_dataset_ROI.shape[0]*3/2
            dx = dataset_film.shape[0]*0.2/2
            x = np.linspace(-dx,dx, dataset_film.shape[0])
            y = np.linspace(-dy,dy, Globals.profiles_doseplan_dataset_ROI.shape[0])
            plot_film = dataset_film[:,Globals.profiles_coordinate_in_dataset]/100
            plot_doseplan = \
                dataset_doseplan[:, Globals.profiles_coordinate_in_dataset]
            film=a.plot(x,plot_film, color='r', label='Film')
            dose=a.plot(y,plot_doseplan, color='b', label='Doseplan')
        elif line_orient == 'd':
            start_f_x, start_f_y = Globals.profiles_line_coords_film[0]
            end_f_x, end_f_y = Globals.end_point
            dx=np.sqrt(((end_f_x-start_f_x)*0.2)**2 + \
                ((end_f_y-start_f_y)*0.2)**2)/2
            if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
                start_d_x, start_d_y = Globals.profiles_line_coords_doseplan[0]
                end_d_x, end_d_y = Globals.end_point
                end_d_x=end_d_x/5; end_d_y=end_d_y/5
                dy=np.sqrt(((end_d_x-start_d_x))**2 + ((end_d_y-start_d_y))**2)/2
            elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
                start_d_x, start_d_y = Globals.profiles_line_coords_doseplan[0]
                end_d_x, end_d_y = Globals.end_point
                end_d_x=end_d_x/10; end_d_y=end_d_y/10
                dy=np.sqrt(((end_d_x-start_d_x)*2)**2 + \
                    ((end_d_y-start_d_y)*2)**2)/2
            else:
                start_d_x, start_d_y = Globals.profiles_line_coords_doseplan[0]
                end_d_x, end_d_y = Globals.end_point
                end_d_x=end_d_x/15; end_d_y=end_d_y/15
                dy=np.sqrt(((end_d_x-start_d_x)*3)**2 + \
                    ((end_d_y-start_d_y)*3)**2)/2



            x = np.linspace(-dx,dx,len(dataset_film))
            y = np.linspace(-dy,dy,len(dataset_doseplan))
            plot_film=dataset_film/100
            plot_doseplan=dataset_doseplan
            film = a.plot(x,plot_film, color='r', label='Film')
            dose= a.plot(y,plot_doseplan, 'b', label='Doseplan')

        else:
            messagebox.showerror("Error", \
                "Fatal error. Something has gone wrong, try again \n(Code: draw")
            return

        def export_plot():
            plt.figure()
            plt.plot(x, plot_film, color='r', label='Film')
            plt.plot(y, plot_doseplan, 'b', label='Doseplan')
            plt.ylabel('Dose (Gy)')
            plt.xlabel('Distance (mm)')
            plt.title("Profiles")
            plt.show()

        Globals.profiles_export_plot_button.configure\
            (state=ACTIVE, command=export_plot)
        a.legend()
        a.set_title("Profiles", fontsize=12)
        a.set_ylabel("Dose (Gy)", fontsize=12)
        a.set_xlabel("Distance (mm)", fontsize=12)

        def mouseMove(event):
            if event.inaxes == a:
                dist = event.xdata
                idx_film = np.searchsorted(x, dist)
                idx_doseplan = np.searchsorted(y, dist)
                if idx_film == 0:
                    idx_film = 0
                elif idx_film == len(x):
                    idx_film = len(x)-1
                else:
                    if abs(x[idx_film-1]-dist) < abs(x[idx_film]-dist):
                        idx_film = idx_film-1
                    else:
                        idx_film = idx_film
                if idx_doseplan == 0:
                    idx_doseplan = 0
                elif idx_doseplan == len(y):
                    idx_doseplan = len(y)-1
                else:
                    if abs(y[idx_doseplan-1]-dist) < abs(y[idx_doseplan]-dist):
                        idx_doseplan = idx_doseplan-1
                    else:
                        idx_doseplan = idx_doseplan

                idx_film = int(np.round(idx_film))
                if idx_film < 0:
                    idx_film = 0
                if idx_film >= len(plot_film):
                    idx_film = len(plot_film) - 1
                idx_doseplan = int(np.round(idx_doseplan))
                if idx_doseplan < 0:
                    idx_doseplan = 0
                if idx_doseplan >= len(plot_doseplan):
                    idx_doseplan = len(plot_doseplan) - 1

                match_text = "\tGraph match: \t"
                match = str(np.round(min(plot_film[idx_film], \
                    plot_doseplan[idx_doseplan])/max(plot_film[idx_film], \
                        plot_doseplan[idx_doseplan])*100, 2)) + "\n"
```

```
distance_text = "Distance:\t "
dose_text = "Dose: \t"
rel_target_dose_text = "Relative to target dose: \t "
rel_mx_dose_ROI_text = "Relative to max dose in ROI: \n"
distance = str(np.round(dist,2)) + "\n"
film = "FILM:   \t"
dose_film = str(np.round(plot_film[idx_film],2)) + "\t"
rel_target_dose_film = str(np.round(100*plot_film[idx_film]/\
    Globals.max_dose_doseplan,2)) + "\t\t\t"
rel_mx_dose_ROI_film = str(np.round(100*plot_film[idx_film]/\
    np.max(plot_film),2)) + "\n"
doseplan = "DOSEPLAN:  \t"
dose_doseplan = \
    str(np.round(plot_doseplan[idx_doseplan],2)) + "\t"
rel_target_dose_doseplan =  \
    str(np.round(100*plot_doseplan[idx_doseplan]/\
        Globals.max_dose_doseplan,2)) + "\t\t\t"
rel_mx_dose_ROI_doseplan =  \
    str(np.round(100*plot_doseplan[idx_doseplan]/+
    np.max(plot_doseplan),2))
notation = match_text + distance_text + dose_text, \
    rel_target_dose_text + rel_mx_dose_ROI_text +\
        film + dose_film + rel_target_dose_film + \
            rel_mx_dose_ROI_film+ doseplan + dose_doseplan\
                + rel_target_dose_doseplan + \
                    rel_mx_dose_ROI_doseplan

children = listBox.get_children()
for item in children:
    listBox.delete(item)
lst = [['Film: ', match, distance, dose_film, \
    rel_mx_dose_ROI_film, rel_target_dose_film],\
    ['Doseplan: ', match, distance, dose_doseplan, \
        rel_mx_dose_ROI_doseplan, rel_target_dose_doseplan]]
for i, (name, m, dis, d, rdROI, rdTarget) in enumerate(lst):
    listBox.insert("", "end", \
        values=(name, m, dis, d, rdROI, rdTarget))
y_min = max(plot_film[idx_film], plot_doseplan[idx_doseplan])\
    -0.3*max(np.max(plot_film), np.max(plot_doseplan))
if y_min < 0:
    y_min = 0
y_max = max(plot_film[idx_film], plot_doseplan[idx_doseplan])\
    +0.3*max(np.max(plot_film), np.max(plot_doseplan))
if y_max > max(np.max(plot_film), np.max(plot_doseplan)):
    y_max =  max(np.max(plot_film), np.max(plot_doseplan))
```

```
                v_line.set_xdata(dist)
                v_line.set_visible(True)
                fig.canvas.draw_idle()

                def freezeData(event):
                    fig.canvas.mpl_disconnect(cid)
                    v_line.set_visible(False)
                    fig.canvas.draw_idle()
                    def startData(event):
                        fig.canvas.mpl_disconnect(cid2)
                        fig.canvas.mpl_disconnect(cid3)
                        draw(line_orient, dataset_film, dataset_doseplan)


                    cid3 = fig.canvas.mpl_connect('button_press_event', startData)

                cid2 = fig.canvas.mpl_connect('button_press_event', freezeData)
            else:
                return

        cid3 = None
        cid = fig.canvas.mpl_connect('motion_notify_event', mouseMove)
        fig.tight_layout()


    if even:
        draw('d', Globals.profiles_dataset_film_variable_draw, \
            Globals.profiles_dataset_doesplan_variable_draw)
        return


    if(Globals.profiles_choice_of_profile_line_type.get() == 'h'\
         and Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]):
        dataset_film = np.zeros(\
            (Globals.profiles_doseplan_dataset_ROI.shape[0], \
                Globals.profiles_film_dataset_ROI_red_channel_dose.shape[1]))
        for i in range(dataset_film.shape[0]-1):
            dataset_film[i,:] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose[int((i*5)+2),:]
        try:
            dataset_film[dataset_film.shape[0]-1,:] =\
                 Globals.profiles_film_dataset_ROI_red_channel_dose[int\
                     ((dataset_film.shape[0]-1)*5+2), :]
        except:
            dataset_film[dataset_film.shape[0]-1,:] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose\
                    [Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0]-1,:]


        line_doseplan = Globals.doseplan_write_image.create_line\
            (0,Globals.doseplan_write_image_var_x,\
            Globals.doseplan_write_image_width,\
                Globals.doseplan_write_image_var_x, fill='red')
        line_film = Globals.film_write_image.create_line\
            (0,Globals.doseplan_write_image_var_x,\
            Globals.doseplan_write_image_width,\
                Globals.doseplan_write_image_var_x, fill='red')

        Globals.profiles_lines.append(line_doseplan)
        Globals.profiles_lines.append(line_film)

        def up_button_pressed(event):
            temp_x = Globals.doseplan_write_image_var_x - 5
            if(temp_x < 0):
                return
            Globals.doseplan_write_image_var_x = temp_x
            Globals.profiles_coordinate_in_dataset = int(temp_x/5)
            Globals.doseplan_write_image.coords(line_doseplan,0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            Globals.film_write_image.coords(line_film, 0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)

        def down_button_pressed(event):
            temp_x = Globals.doseplan_write_image_var_x + 5
            if(temp_x >= Globals.doseplan_write_image_height):
                return
            Globals.profiles_coordinate_in_dataset = int(temp_x/5)
            Globals.doseplan_write_image_var_x = temp_x
            Globals.doseplan_write_image.coords(line_doseplan,0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            Globals.film_write_image.coords(line_film, 0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
                draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)



        Globals.form.bind("<Up>", up_button_pressed)
        Globals.form.bind("<Down>", down_button_pressed)

        if Globals.profiles_first_time_in_drawProfiles:
            Globals.profiles_first_time_in_drawProfiles = False
            draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)
    elif(Globals.profiles_choice_of_profile_line_type.get()=='h' \
        and Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
        dataset_film = np.zeros(\
            (Globals.profiles_doseplan_dataset_ROI.shape[0], \
                Globals.profiles_film_dataset_ROI_red_channel_dose.shape[1]))
        for i in range(dataset_film.shape[0]-1):
            dataset_film[i,:] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose[int((i*10)+5),:]
        try:
            dataset_film[dataset_film.shape[0]-1,:] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose\
                    [int((dataset_film.shape[0]-1)*10+5), :]
        except:
            dataset_film[dataset_film.shape[0]-1,:] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose\
                    [Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0]-1,:]



        line_doseplan = Globals.doseplan_write_image.create_line\
            (0,Globals.doseplan_write_image_var_x,\
                Globals.doseplan_write_image_width,\
                    Globals.doseplan_write_image_var_x, fill='red')
        line_film = Globals.film_write_image.create_line\
            (0,Globals.doseplan_write_image_var_x,\
                Globals.doseplan_write_image_width,\
                    Globals.doseplan_write_image_var_x, fill='red')

        Globals.profiles_lines.append(line_doseplan)
        Globals.profiles_lines.append(line_film)

        def up_button_pressed(event):
            temp_x = Globals.doseplan_write_image_var_x - 10
            if(temp_x < 0):
                return
            Globals.doseplan_write_image_var_x = temp_x
            Globals.profiles_coordinate_in_dataset = int(temp_x/10)
            Globals.doseplan_write_image.coords(line_doseplan,0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            Globals.film_write_image.coords(line_film, 0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)

        def down_button_pressed(event):
            temp_x = Globals.doseplan_write_image_var_x + 10
            if(temp_x >= Globals.doseplan_write_image_height):
                return
            Globals.profiles_coordinate_in_dataset = int(temp_x/10)
            Globals.doseplan_write_image_var_x = temp_x
            Globals.doseplan_write_image.coords(line_doseplan,0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            Globals.film_write_image.coords(line_film, 0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)

        Globals.form.bind("<Up>", up_button_pressed)
        Globals.form.bind("<Down>", down_button_pressed)

        if Globals.profiles_first_time_in_drawProfiles:
            Globals.profiles_first_time_in_drawProfiles = False
            draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)

    elif(Globals.profiles_choice_of_profile_line_type.get() == 'h' \
        and Globals.profiles_dataset_doseplan.PixelSpacing==[3, 3]):
        dataset_film = np.zeros(\
            (Globals.profiles_doseplan_dataset_ROI.shape[0], \
                Globals.profiles_film_dataset_ROI_red_channel_dose.shape[1]))
        for i in range(dataset_film.shape[0]-1):
            dataset_film[i,:] = Globals.profiles_film_dataset_ROI_red_channel_dose\
                [int((i*15)+7),:]
        try:
            dataset_film[dataset_film.shape[0]-1,:] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose\
                    [int((dataset_film.shape[0]−1)*15+7), :]
        except:
            dataset_film[dataset_film.shape[0]−1,:] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose\
                    [Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0]−1,:]


        line_doseplan = Globals.doseplan_write_image.\
            create_line(0,Globals.doseplan_write_image_var_x,\
                Globals.doseplan_write_image_width,\
                    Globals.doseplan_write_image_var_x, fill='red')
        line_film = Globals.film_write_image.create_line\
            (0,Globals.doseplan_write_image_var_x,\
                Globals.doseplan_write_image_width,\
                    Globals.doseplan_write_image_var_x, fill='red')

        Globals.profiles_lines.append(line_doseplan)
        Globals.profiles_lines.append(line_film)

        def up_button_pressed(event):
            temp_x = Globals.doseplan_write_image_var_x − 15
            if(temp_x < 0):
                #Outside the frame
                return
            #inside the frame
            Globals.doseplan_write_image_var_x = temp_x
            Globals.profiles_coordinate_in_dataset = int(temp_x/15)
            Globals.doseplan_write_image.coords(line_doseplan,0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            Globals.film_write_image.coords(line_film, 0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)

        def down_button_pressed(event):
            temp_x = Globals.doseplan_write_image_var_x + 15
            if(temp_x >= Globals.doseplan_write_image_height):
                #Outside the frame
                return
            #Inside the frame
            Globals.profiles_coordinate_in_dataset = int(temp_x/15)
            Globals.doseplan_write_image_var_x = temp_x
            Globals.doseplan_write_image.coords(line_doseplan,0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            Globals.film_write_image.coords(line_film, 0,\
                Globals.doseplan_write_image_var_x,\
                    Globals.doseplan_write_image_width,\
                        Globals.doseplan_write_image_var_x)
            draw('h', dataset_film,Globals.profiles_doseplan_dataset_ROI)

        Globals.form.bind("<Up>", up_button_pressed)
        Globals.form.bind("<Down>", down_button_pressed)

        if Globals.profiles_first_time_in_drawProfiles:
            Globals.profiles_first_time_in_drawProfiles = False
            draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)

    elif(Globals.profiles_choice_of_profile_line_type.get() == 'v' \
        and Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]):
        dataset_film = np.zeros(\
            (Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0],\
                Globals.profiles_doseplan_dataset_ROI.shape[1]))
        for i in range(dataset_film.shape[1]-1):
            dataset_film[:,i] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose[:,int((i*5)+2)]
        try:
            dataset_film[:,dataset_film.shape[1]-1] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose\
                    [:,int((dataset_film.shape[1]-1)*5+2)]
        except:
            dataset_film[:,dataset_film.shape[1]-1] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose[:,\
                    Globals.profiles_film_dataset_ROI_red_channel_dose.shape[1]-1]


        line_doseplan = Globals.doseplan_write_image.create_line\
            (Globals.doseplan_write_image_var_y, 0,\
                Globals.doseplan_write_image_var_y, \
                    Globals.doseplan_write_image_height, fill='red')
        line_film = Globals.film_write_image.create_line\
            (Globals.doseplan_write_image_var_y,0,\
                Globals.doseplan_write_image_var_y,\
                    Globals.doseplan_write_image_height, fill='red')

        Globals.profiles_lines.append(line_doseplan)
```

```
        Globals.profiles_lines.append(line_film)

        def left_button_pressed(event):
            temp_y = Globals.doseplan_write_image_var_y - 5
            if(temp_y < 0):
                return
            Globals.doseplan_write_image_var_y = temp_y
            Globals.profiles_coordinate_in_dataset = int(temp_y/5)
            Globals.doseplan_write_image.coords(line_doseplan,\
                Globals.doseplan_write_image_var_y, 0,\
                    Globals.doseplan_write_image_var_y, \
                        Globals.doseplan_write_image_height)
            Globals.film_write_image.coords(line_film, \
                Globals.doseplan_write_image_var_y, 0,\
                    Globals.doseplan_write_image_var_y, \
                        Globals.doseplan_write_image_height)
            draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)

        def right_button_pressed(event):
            temp_y = Globals.doseplan_write_image_var_y + 5
            if(temp_y >= Globals.doseplan_write_image_width):
                return
            Globals.profiles_coordinate_in_dataset = int(temp_y/5)
            Globals.doseplan_write_image_var_y = temp_y
            Globals.doseplan_write_image.coords(line_doseplan,\
                Globals.doseplan_write_image_var_y, 0,\
                    Globals.doseplan_write_image_var_y, \
                        Globals.doseplan_write_image_height)
            Globals.film_write_image.coords(line_film, \
                Globals.doseplan_write_image_var_y, 0,\
                    Globals.doseplan_write_image_var_y, \
                        Globals.doseplan_write_image_height)
            draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)


        Globals.form.bind("<Left>", left_button_pressed)
        Globals.form.bind("<Right>", right_button_pressed)

        if Globals.profiles_first_time_in_drawProfiles:
            Globals.profiles_first_time_in_drawProfiles = False
            draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)

    elif(Globals.profiles_choice_of_profile_line_type.get() == 'v' and \
        Globals.profiles_dataset_doseplan.PixelSpacing == [2, 2]):
```

```
dataset_film = np.zeros(\
    (Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0], \
        Globals.profiles_doseplan_dataset_ROI.shape[1]))
for i in range(dataset_film.shape[1]-1):
    dataset_film[:,i] = \
        Globals.profiles_film_dataset_ROI_red_channel_dose[:,int((i*10)+5)]
try:
    dataset_film[:,dataset_film.shape[1]-1] = \
        Globals.profiles_film_dataset_ROI_red_channel_dose[:,\
            int((dataset_film.shape[1]-1)*10+5)]
except:
    dataset_film[:,dataset_film.shape[1]-1] = \
        Globals.profiles_film_dataset_ROI_red_channel_dose[:,\
            Globals.profiles_film_dataset_ROI_red_channel_dose.shape[1]-1]



line_doseplan = Globals.doseplan_write_image.create_line\
    (Globals.doseplan_write_image_var_y, 0,\
        Globals.doseplan_write_image_var_y, \
            Globals.doseplan_write_image_height, fill='red')
line_film = Globals.film_write_image.create_line\
    (Globals.doseplan_write_image_var_y,0,\
        Globals.doseplan_write_image_var_y,\
            Globals.doseplan_write_image_height, fill='red')

Globals.profiles_lines.append(line_doseplan)
Globals.profiles_lines.append(line_film)

def left_button_pressed(event):
    temp_y = Globals.doseplan_write_image_var_y - 10
    if(temp_y < 0):
        return
    Globals.doseplan_write_image_var_y = temp_y
    Globals.profiles_coordinate_in_dataset = int(temp_y/10)
    Globals.doseplan_write_image.coords(line_doseplan,\
        Globals.doseplan_write_image_var_y, 0,\
            Globals.doseplan_write_image_var_y, \
                Globals.doseplan_write_image_height)
    Globals.film_write_image.coords(line_film, \
        Globals.doseplan_write_image_var_y, 0,\
            Globals.doseplan_write_image_var_y, \
                Globals.doseplan_write_image_height)
    draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)

def right_button_pressed(event):
```

```
            temp_y = Globals.doseplan_write_image_var_y + 10
            if(temp_y >= Globals.doseplan_write_image_width):
                return
            Globals.profiles_coordinate_in_dataset = int(temp_y/10)
            Globals.doseplan_write_image_var_y = temp_y
            Globals.doseplan_write_image.coords(line_doseplan,\
                Globals.doseplan_write_image_var_y, 0,\
                    Globals.doseplan_write_image_var_y, \
                        Globals.doseplan_write_image_height)
            Globals.film_write_image.coords(line_film, \
                Globals.doseplan_write_image_var_y, 0,\
                    Globals.doseplan_write_image_var_y, \
                        Globals.doseplan_write_image_height)
            draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)


        Globals.form.bind("<Left>", left_button_pressed)
        Globals.form.bind("<Right>", right_button_pressed)

        if Globals.profiles_first_time_in_drawProfiles:
            Globals.profiles_first_time_in_drawProfiles = False
            draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)

    elif(Globals.profiles_choice_of_profile_line_type.get() == 'v' and \
        Globals.profiles_dataset_doseplan.PixelSpacing == [3, 3]):
        dataset_film = np.zeros(\
            (Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0], \
                Globals.profiles_doseplan_dataset_ROI.shape[1]))
        for i in range(dataset_film.shape[1]-1):
            dataset_film[:,i] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose[:,int((i*15)+7)]
        try:
            dataset_film[:,dataset_film.shape[1]-1] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose[:,\
                    int((dataset_film.shape[1]-1)*15+7)]
        except:
            dataset_film[:,dataset_film.shape[1]-1] = \
                Globals.profiles_film_dataset_ROI_red_channel_dose[:,\
                    Globals.profiles_film_dataset_ROI_red_channel_dose.shape[1]-1]


        line_doseplan = Globals.doseplan_write_image.create_line\
            (Globals.doseplan_write_image_var_y, 0,\
                Globals.doseplan_write_image_var_y,\
                    Globals.doseplan_write_image_height, fill='red')
```

```
line_film = Globals.film_write_image.create_line\
    (Globals.doseplan_write_image_var_y,0,\
        Globals.doseplan_write_image_var_y,\
            Globals.doseplan_write_image_height, fill='red')

Globals.profiles_lines.append(line_doseplan)
Globals.profiles_lines.append(line_film)

def left_button_pressed(event):
    temp_y = Globals.doseplan_write_image_var_y - 15
    if(temp_y < 0):
        return
    Globals.doseplan_write_image_var_y = temp_y
    Globals.profiles_coordinate_in_dataset = int(temp_y/15)
    Globals.doseplan_write_image.coords(line_doseplan,\
        Globals.doseplan_write_image_var_y, 0,\
            Globals.doseplan_write_image_var_y, \
                Globals.doseplan_write_image_height)
    Globals.film_write_image.coords(line_film, \
        Globals.doseplan_write_image_var_y, 0,\
            Globals.doseplan_write_image_var_y, \
                Globals.doseplan_write_image_height)
    draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)

def right_button_pressed(event):
    temp_y = Globals.doseplan_write_image_var_y + 15
    if(temp_y >= Globals.doseplan_write_image_width):
        return
    Globals.profiles_coordinate_in_dataset = int(temp_y/15)
    Globals.doseplan_write_image_var_y = temp_y
    Globals.doseplan_write_image.coords(line_doseplan,\
        Globals.doseplan_write_image_var_y, 0,\
            Globals.doseplan_write_image_var_y, \
                Globals.doseplan_write_image_height)
    Globals.film_write_image.coords(line_film, \
        Globals.doseplan_write_image_var_y, 0,\
            Globals.doseplan_write_image_var_y, \
                Globals.doseplan_write_image_height)
    draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)


Globals.form.bind("<Left>", left_button_pressed)
Globals.form.bind("<Right>", right_button_pressed)

if Globals.profiles_first_time_in_drawProfiles:
```

```
            Globals.profiles_first_time_in_drawProfiles = False
            draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)
    elif(Globals.profiles_choice_of_profile_line_type.get() == 'd'\
         and Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]):
        start_point = [0,0]
        def mousePushed(event):
            start_point = [event.y, event.x]
            if not len(Globals.profiles_lines)==0:
                Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
                Globals.film_write_image.delete(Globals.profiles_lines[1])
                Globals.profiles_lines = []

            line_doseplan = Globals.doseplan_write_image.create_line(start_point[1],\
                 start_point[0],start_point[1],start_point[0], fill='red')
            line_film = Globals.film_write_image.create_line(start_point[1], \
                start_point[0],start_point[1],start_point[0], fill='red')

            Globals.profiles_lines.append(line_doseplan)
            Globals.profiles_lines.append(line_film)

            def mouseMoving(event):
                Globals.doseplan_write_image.coords(line_doseplan, \
                    start_point[1], start_point[0], event.x, event.y)
                Globals.film_write_image.coords(line_film, start_point[1], \
                    start_point[0], event.x, event.y)



            Globals.film_write_image.bind("<B1-Motion>", mouseMoving)

            def mouseReleased(event):
                Globals.end_point = [event.y, event.x]
                Globals.doseplan_write_image.coords(line_doseplan, \
                    start_point[1], start_point[0], event.x, event.y)
                Globals.film_write_image.coords(line_film, \
                    start_point[1], start_point[0], event.x, event.y)
                Globals.profiles_line_coords_film = getCoordsInRandomLine\
                    (start_point[1], start_point[0], \
                        Globals.end_point[1], Globals.end_point[0])
                Globals.profiles_line_coords_doseplan = getCoordsInRandomLine\
                    (int(start_point[1]/5), int(start_point[0]/5), \
                    int(Globals.end_point[1]/5), int(Globals.end_point[0]/5))
                Globals.profiles_dataset_film_variable_draw = \
                    np.zeros(len(Globals.profiles_line_coords_film))
                Globals.profiles_dataset_doesplan_variable_draw=\
                    np.zeros(len(Globals.profiles_line_coords_doseplan))


                for i in range(len(Globals.profiles_dataset_film_variable_draw)):
                    coord = Globals.profiles_line_coords_film[i]
                    try:
                        Globals.profiles_dataset_film_variable_draw[i] = \
                            Globals.profiles_film_dataset_ROI_red_channel_dose\
                                [coord[0]-1, coord[1]-1]
                    except:
                        return
                for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
                    coord = Globals.profiles_line_coords_doseplan[i]
                    try:
                        Globals.profiles_dataset_doesplan_variable_draw[i] =\
                             Globals.profiles_doseplan_dataset_ROI\
                                 [coord[0]-1, coord[1]-1]
                    except:
                        return
                draw('d', Globals.profiles_dataset_film_variable_draw, \
                    Globals.profiles_dataset_doesplan_variable_draw)

            Globals.film_write_image.bind("<ButtonRelease-1>", mouseReleased)
        Globals.film_write_image.bind("<Button-1>", mousePushed)


    elif(Globals.profiles_choice_of_profile_line_type.get() == 'd' and \
        Globals.profiles_dataset_doseplan.PixelSpacing == [2, 2]):
        start_point = [0,0]
        def mousePushed(event):
            start_point = [event.y, event.x]
            if not len(Globals.profiles_lines)==0:
                Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
                Globals.film_write_image.delete(Globals.profiles_lines[1])
                Globals.profiles_lines = []

            line_doseplan = Globals.doseplan_write_image.create_line(start_point[1],\
                start_point[0],start_point[1],start_point[0], fill='red')
            line_film = Globals.film_write_image.create_line(start_point[1],\
                start_point[0],start_point[1],start_point[0], fill='red')

            Globals.profiles_lines.append(line_doseplan)
            Globals.profiles_lines.append(line_film)

            def mouseMoving(event):
```

```
        Globals.doseplan_write_image.coords(line_doseplan, \
            start_point[1], start_point[0], event.x, event.y)
        Globals.film_write_image.coords(line_film, start_point[1], \
            start_point[0], event.x, event.y)



    Globals.film_write_image.bind("<B1-Motion>", mouseMoving)

    def mouseReleased(event):
        Globals.end_point = [event.y, event.x]
        Globals.doseplan_write_image.coords(line_doseplan, start_point[1],\
            start_point[0], event.x, event.y)
        Globals.film_write_image.coords(line_film, start_point[1], \
            start_point[0], event.x, event.y)
        Globals.profiles_line_coords_film = getCoordsInRandomLine\
            (start_point[1], start_point[0], \
                Globals.end_point[1], Globals.end_point[0])
        Globals.profiles_line_coords_doseplan = \
            getCoordsInRandomLine(int(start_point[1]/10), \
                int(start_point[0]/10), \
                    int(Globals.end_point[1]/10), \
                        int(Globals.end_point[0]/10))
        Globals.profiles_dataset_film_variable_draw = \
            np.zeros(len(Globals.profiles_line_coords_film))
        Globals.profiles_dataset_doesplan_variable_draw=\
            np.zeros(len(Globals.profiles_line_coords_doseplan))

        for i in range(len(Globals.profiles_dataset_film_variable_draw)):
            coord = Globals.profiles_line_coords_film[i]
            try:
                Globals.profiles_dataset_film_variable_draw[i] = \
                    Globals.profiles_film_dataset_ROI_red_channel_dose\
                        [coord[0]-1, coord[1]-1]
            except:
                return

        for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
            try:
                Globals.profiles_dataset_doesplan_variable_draw[i] = \
                    Globals.profiles_doseplan_dataset_ROI[int\
                        (Globals.profiles_line_coords_doseplan[i][1])-1,\
                            int(Globals.profiles_line_coords_doseplan[i][0])-1]
            except:
                return
            draw('d', Globals.profiles_dataset_film_variable_draw, \
                Globals.profiles_dataset_doesplan_variable_draw)

        Globals.film_write_image.bind("<ButtonRelease-1>", mouseReleased)
    Globals.film_write_image.bind("<Button-1>", mousePushed)
elif(Globals.profiles_choice_of_profile_line_type.get() == 'd' and \
    Globals.profiles_dataset_doseplan.PixelSpacing == [3, 3]):
    start_point = [0,0]
    def mousePushed(event):
        start_point = [event.y, event.x]
        if not len(Globals.profiles_lines)==0:
            Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
            Globals.film_write_image.delete(Globals.profiles_lines[1])
            Globals.profiles_lines = []

        line_doseplan = Globals.doseplan_write_image.create_line\
            (start_point[1], start_point[0],\
                start_point[1],start_point[0], fill='red')
        line_film = Globals.film_write_image.create_line(start_point[1], \
            start_point[0],start_point[1],start_point[0], fill='red')

        Globals.profiles_lines.append(line_doseplan)
        Globals.profiles_lines.append(line_film)

        def mouseMoving(event):
            Globals.doseplan_write_image.coords(line_doseplan, start_point[1], \
                start_point[0], event.x, event.y)
            Globals.film_write_image.coords(line_film, start_point[1], \
                start_point[0], event.x, event.y)



        Globals.film_write_image.bind("<B1-Motion>", mouseMoving)

        def mouseReleased(event):
            Globals.end_point = [event.y, event.x]
            Globals.doseplan_write_image.coords(line_doseplan, start_point[1], \
                start_point[0], event.x, event.y)
            Globals.film_write_image.coords(line_film, start_point[1], \
                start_point[0], event.x, event.y)
            Globals.profiles_line_coords_film = getCoordsInRandomLine\
                (start_point[1], start_point[0], \
                    Globals.end_point[1], Globals.end_point[0])
            Globals.profiles_line_coords_doseplan = getCoordsInRandomLine\
                (int(start_point[1]/15), int(start_point[0]/15), \
                    int(Globals.end_point[1]/15), int(Globals.end_point[0]/15))
                Globals.profiles_dataset_film_variable_draw = \
                    np.zeros(len(Globals.profiles_line_coords_film))
                Globals.profiles_dataset_doesplan_variable_draw=\
                    np.zeros(len(Globals.profiles_line_coords_doseplan))

                for i in range(len(Globals.profiles_dataset_film_variable_draw)):
                    coord = Globals.profiles_line_coords_film[i]
                    try:
                        Globals.profiles_dataset_film_variable_draw[i] = \
                            Globals.profiles_film_dataset_ROI_red_channel_dose\
                                [coord[0]-1, coord[1]-1]
                    except:
                        return
                for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
                    try:
                        Globals.profiles_dataset_doesplan_variable_draw[i] = \
                            Globals.profiles_doseplan_dataset_ROI[int\
                                (Globals.profiles_line_coords_doseplan[i][0])-1, \
                        int(Globals.profiles_line_coords_doseplan[i][1])-1]
                    except:
                        return

                draw('d', Globals.profiles_dataset_film_variable_draw, \
                    Globals.profiles_dataset_doesplan_variable_draw)

            Globals.film_write_image.bind("<ButtonRelease-1>", mouseReleased)
        Globals.film_write_image.bind("<Button-1>", mousePushed)
    else:
        messagebox.showerror("Error", \
            "Fatal error. Something went wrong, try again \n(Code: drawProfiles)")
        return

def trace_profileLineType(var, indx, mode):
#---------------------------------------------------
# Function to trace the profile type
#
# Function is a callback to
# profiles_choice_of_profile_line_type.trace_add
# in notebook.py
#---------------------------------------------------
    test_drawProfiles()


def test_drawProfiles():
```

```
#————————————————————————————————————————————————
# Function to make the winwow ready for plotting
#
# This functions is a called in
# trace_profileLineType()
#————————————————————————————————————————————————
    if Globals.profiles_dataset_doseplan == None:
        return
    else:
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_write_image.delete(Globals.profiles_lines[1])
        Globals.form.unbind("<Up>")
        Globals.form.unbind("<Down>")
        Globals.form.unbind("<Left>")
        Globals.form.unbind("<Rigth>")
        Globals.profiles_first_time_in_drawProfiles = True
        drawProfiles(False)


def adjustROILeft(line_orient):
#————————————————————————————————————————————————
# Function to adjust ROI to the left
#
# This functions is a callback to the button
# profiles_adjust_button_left in notebook.py
#————————————————————————————————————————————————
    if not line_orient == 'd':
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_write_image.delete(Globals.profiles_lines[1])
    if(Globals.profiles_film_variable_ROI_coords[2]−1< 0):
        messagebox.showwarning("Warning", \
            "Reached end of film \n(Code: adjustROILeft)")
        return
    Globals.profiles_film_variable_ROI_coords = \
        [Globals.profiles_film_variable_ROI_coords[0],\
            Globals.profiles_film_variable_ROI_coords[1],\
                Globals.profiles_film_variable_ROI_coords[2]−1, \
                    Globals.profiles_film_variable_ROI_coords[3]−1]
    Globals.profiles_film_dataset_ROI_red_channel_dose = \
        Globals.profiles_film_dataset_red_channel_dose\
            [Globals.profiles_film_variable_ROI_coords[0]:\
                Globals.profiles_film_variable_ROI_coords[1],\
                Globals.profiles_film_variable_ROI_coords[2]:\
                    Globals.profiles_film_variable_ROI_coords[3]]
    Globals.profiles_first_time_in_drawProfiles = True
    if line_orient == 'd':
        for i in range(len(Globals.profiles_dataset_film_variable_draw)):
            coord = Globals.profiles_line_coords_film[i]
            try:
                Globals.profiles_dataset_film_variable_draw[i] = \
                    Globals.profiles_film_dataset_ROI_red_channel_dose\
                        [coord[0]-1, coord[1]-1]
            except:
                return
        for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
            try:
                Globals.profiles_dataset_doesplan_variable_draw[i] =\
                     Globals.profiles_doseplan_dataset_ROI[int\
                         (Globals.profiles_line_coords_doseplan[i][0])-1, \
                             int(Globals.profiles_line_coords_doseplan[i][1])-1]
            except:
                return
        drawProfiles(True)
    else:
        drawProfiles(False)

def adjustROIRight(line_orient):
#---------------------------------------------------
# Function to adjust ROI to the right
#
# This functions is a callback to the button
# profiles_adjust_button_right in notebook.py
#---------------------------------------------------
    if not line_orient == 'd':
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_write_image.delete(Globals.profiles_lines[1])
    if(Globals.profiles_film_variable_ROI_coords[3]+1 > \
        Globals.profiles_film_dataset_red_channel_dose.shape[1]):
        messagebox.showwarning("Warning", \
            "Reached end of film \n(Code: adjustROIRight)")
        return
    Globals.profiles_film_variable_ROI_coords = \
        [Globals.profiles_film_variable_ROI_coords[0], \
            Globals.profiles_film_variable_ROI_coords[1],\
            Globals.profiles_film_variable_ROI_coords[2]+1, \
                Globals.profiles_film_variable_ROI_coords[3]+1]
    Globals.profiles_film_dataset_ROI_red_channel_dose = \
        Globals.profiles_film_dataset_red_channel_dose\
            [Globals.profiles_film_variable_ROI_coords[0]:\
                Globals.profiles_film_variable_ROI_coords[1],\
                Globals.profiles_film_variable_ROI_coords[2]:\
                    Globals.profiles_film_variable_ROI_coords[3]]
    Globals.profiles_first_time_in_drawProfiles = True
    if line_orient == 'd':
        for i in range(len(Globals.profiles_dataset_film_variable_draw)):
            coord = Globals.profiles_line_coords_film[i]
            try:
                Globals.profiles_dataset_film_variable_draw[i] = \
                    Globals.profiles_film_dataset_ROI_red_channel_dose\
                        [coord[0]-1, coord[1]-1]
            except:
                return
        for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
            try:
                Globals.profiles_dataset_doesplan_variable_draw[i] = \
                    Globals.profiles_doseplan_dataset_ROI[int\
                        (Globals.profiles_line_coords_doseplan[i][0])-1, \
                            int(Globals.profiles_line_coords_doseplan[i][1])-1]
            except:
                return
        drawProfiles(True)
    else:
        drawProfiles(False)

def adjustROIUp(line_orient):
#--------------------------------------------------
# Function to adjust ROI up
#
# This functions is a callback to the button
# profiles_adjust_button_up in notebook.py
#--------------------------------------------------
    if not line_orient == 'd':
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_write_image.delete(Globals.profiles_lines[1])
    if(Globals.profiles_film_variable_ROI_coords[0]-1 < 0):
        messagebox.showwarning("Warning", \
            "Reached end of film \n(Code: adjustROIUp)")
        return
    Globals.profiles_film_variable_ROI_coords = \
        [Globals.profiles_film_variable_ROI_coords[0]-1, \
            Globals.profiles_film_variable_ROI_coords[1]-1,\
                Globals.profiles_film_variable_ROI_coords[2], \
                    Globals.profiles_film_variable_ROI_coords[3]]
    Globals.profiles_film_dataset_ROI_red_channel_dose = \
        Globals.profiles_film_dataset_red_channel_dose\
            [Globals.profiles_film_variable_ROI_coords[0]:\
                Globals.profiles_film_variable_ROI_coords[1],\
                    Globals.profiles_film_variable_ROI_coords[2]:\
                        Globals.profiles_film_variable_ROI_coords[3]]
    Globals.profiles_first_time_in_drawProfiles = True
    if line_orient == 'd':
        for i in range(len(Globals.profiles_dataset_film_variable_draw)):
            coord = Globals.profiles_line_coords_film[i]
            try:
                Globals.profiles_dataset_film_variable_draw[i] = \
                    Globals.profiles_film_dataset_ROI_red_channel_dose\
                        [coord[0]-1, coord[1]-1]
            except:
                return
        for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
            try:
                Globals.profiles_dataset_doesplan_variable_draw[i] = \
                    Globals.profiles_doseplan_dataset_ROI[int\
                        (Globals.profiles_line_coords_doseplan[i][0])-1, \
                            int(Globals.profiles_line_coords_doseplan[i][1])-1]
            except:
                return
        drawProfiles(True)
    else:
        drawProfiles(False)

def adjustROIDown(line_orient):
#--------------------------------------------
# Function to adjust ROI down
#
# This functions is a callback to the button
# profiles_adjust_button_down in notebook.py
#--------------------------------------------
    if not line_orient == 'd':
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_write_image.delete(Globals.profiles_lines[1])
    if(Globals.profiles_film_variable_ROI_coords[1]+1 > \
        Globals.profiles_film_dataset_red_channel_dose.shape[0]):
        messagebox.showwarning("Warning", \
            "Reached end of film \n(Code: adjustROIDown)")
        return
    Globals.profiles_film_variable_ROI_coords = \
        [Globals.profiles_film_variable_ROI_coords[0]+1, \
            Globals.profiles_film_variable_ROI_coords[1]+1,\
                Globals.profiles_film_variable_ROI_coords[2], \
                    Globals.profiles_film_variable_ROI_coords[3]]
    Globals.profiles_film_dataset_ROI_red_channel_dose = \
        Globals.profiles_film_dataset_red_channel_dose\
            [Globals.profiles_film_variable_ROI_coords[0]:\
                Globals.profiles_film_variable_ROI_coords[1],\
                    Globals.profiles_film_variable_ROI_coords[2]:\
                        Globals.profiles_film_variable_ROI_coords[3]]
    Globals.profiles_first_time_in_drawProfiles = True
    if line_orient == 'd':
        for i in range(len(Globals.profiles_dataset_film_variable_draw)):
            coord = Globals.profiles_line_coords_film[i]
            try:
                Globals.profiles_dataset_film_variable_draw[i] = \
                    Globals.profiles_film_dataset_ROI_red_channel_dose\
                        [coord[0]-1, coord[1]-1]
            except:
                return
        for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
            try:
                Globals.profiles_dataset_doesplan_variable_draw[i] = \
                    Globals.profiles_doseplan_dataset_ROI[int\
                        (Globals.profiles_line_coords_doseplan[i][0])-1, \
                            int(Globals.profiles_line_coords_doseplan[i][1])-1]
            except:
                return
        drawProfiles(True)
    else:
        drawProfiles(False)

def returnToOriginalROICoordinates(line_orient):
#---------------------------------------------------
# Function to adjust ROI to original placement
#
# This functions is a callback to the button
# profiles_adjust_button_return in notebook.py
#---------------------------------------------------
    if not line_orient == 'd':
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_write_image.delete(Globals.profiles_lines[1])
    Globals.profiles_film_variable_ROI_coords = \
        [Globals.profiles_ROI_coords[0][1], Globals.profiles_ROI_coords[2][1],\
                Globals.profiles_ROI_coords[0][0], Globals.profiles_ROI_coords[1][0]]

    Globals.profiles_film_dataset_ROI_red_channel_dose = \
        Globals.profiles_film_dataset_red_channel_dose\
            [Globals.profiles_film_variable_ROI_coords[0]:\
                Globals.profiles_film_variable_ROI_coords[1],\
                    Globals.profiles_film_variable_ROI_coords[2]:\
                        Globals.profiles_film_variable_ROI_coords[3]]
    Globals.profiles_first_time_in_drawProfiles = True
    if line_orient == 'd':
        for i in range(len(Globals.profiles_dataset_film_variable_draw)):
            coord = Globals.profiles_line_coords_film[i]
            try:
                Globals.profiles_dataset_film_variable_draw[i] = \
                    Globals.profiles_film_dataset_ROI_red_channel_dose\
                        [coord[0]-1, coord[1]-1]
            except:
                return
        for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
            Globals.profiles_dataset_doesplan_variable_draw[i] = \
                Globals.profiles_doseplan_dataset_ROI[int\
                    (Globals.profiles_line_coords_doseplan[i][0])-1, \
                        int(Globals.profiles_line_coords_doseplan[i][1])-1]
        drawProfiles(True)
    else:
        drawProfiles(False)

def pixel_to_dose(P,a,b,c):
#---------------------------------------------------
# Function to map between pixel value and dose
#
#---------------------------------------------------
    ret = c + b/(P-a)
    return ret

def processDoseplan_usingReferencePoint(only_one):
#---------------------------------------------------
# Function to process the doseplan as the
# user has choosen to use reference point
#
# This functions is a called in UploadDoseplan()
#---------------------------------------------------

    ################ RT Plan ##################
    iso_1 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[0] - \
        Globals.profiles_dataset_rtplan.BeamSequence[0].\
            ControlPointSequence[0].IsocenterPosition[0])
    iso_2 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[1] - \
        Globals.profiles_dataset_rtplan.BeamSequence[0].\
            ControlPointSequence[0].IsocenterPosition[1])
    iso_3 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[2] - \
        Globals.profiles_dataset_rtplan.BeamSequence[0].\
            ControlPointSequence[0].IsocenterPosition[2])

    Globals.profiles_isocenter_mm = [iso_1, iso_2, iso_3]

    try:
        Globals.profiles_vertical = int(Globals.profiles_vertical)
    except:
        messagebox.showerror("Error", "Could not read the \
vertical displacements\n (Code: displacements to integer)")
        return
    try:
        Globals.profiles_lateral = int(Globals.profiles_lateral)
    except:
        messagebox.showerror("Error", "Could not read the \
lateral displacements\n (Code: displacements to integer)")
        return
    try:
        Globals.profiles_longitudinal = int(Globals.profiles_longitudinal)
    except:
        messagebox.showerror("Error", "Could not read the \
longitudinal displacements\n (Code: displacements to integer)")
        return

    lateral = Globals.profiles_lateral
    longit = Globals.profiles_longitudinal
    vertical = Globals.profiles_vertical
    isocenter_px = np.zeros(3)
    distance_in_doseplan_ROI_reference_point_px = []
    if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
        isocenter_px[0] = np.round(iso_1)
        isocenter_px[1] = np.round(iso_2)
        isocenter_px[2] = np.round(iso_3)

        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.profiles_distance_reference_point_ROI[0][0]),\
                np.round(Globals.profiles_distance_reference_point_ROI[0][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.profiles_distance_reference_point_ROI[1][0]),\
                np.round(Globals.profiles_distance_reference_point_ROI[1][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.profiles_distance_reference_point_ROI[2][0]),\
                np.round(Globals.profiles_distance_reference_point_ROI[2][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.profiles_distance_reference_point_ROI[3][0]),\
                np.round(Globals.profiles_distance_reference_point_ROI[3][1])])

        lateral_px = np.round(lateral)
        vertical_px = np.round(vertical)
        longit_px = np.round(longit)

        doseplan_lateral_displacement_px = \
            np.round(Globals.profiles_doseplan_lateral_displacement)
        doseplan_vertical_displacement_px = \
            np.round(Globals.profiles_doseplan_vertical_displacement)
        doseplan_longitudinal_displacement_px = \
            np.round(Globals.profiles_doseplan_longitudianl_displacement)

    elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
        isocenter_px[0] = np.round(iso_1/2)
        isocenter_px[1] = np.round(iso_2/2)
        isocenter_px[2] = np.round(iso_3/2)

        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.profiles_distance_reference_point_ROI[0][0])/2),\
                np.round((Globals.profiles_distance_reference_point_ROI[0][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.profiles_distance_reference_point_ROI[1][0])/2),\
                np.round((Globals.profiles_distance_reference_point_ROI[1][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.profiles_distance_reference_point_ROI[2][0])/2),\
                np.round((Globals.profiles_distance_reference_point_ROI[2][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.profiles_distance_reference_point_ROI[3][0])/2),\
                np.round((Globals.profiles_distance_reference_point_ROI[3][1])/2)])

        lateral_px = np.round(lateral/2)
        vertical_px = np.round(vertical/2)
        longit_px = np.round(longit/2)

        doseplan_lateral_displacement_px = \
            np.round((Globals.profiles_doseplan_lateral_displacement)/2)
        doseplan_vertical_displacement_px = \
            np.round((Globals.profiles_doseplan_vertical_displacement)/2)
        doseplan_longitudinal_displacement_px = \
            np.round((Globals.profiles_doseplan_longitudianl_displacement)/2)

    else:
```

```
    isocenter_px[0] = np.round(iso_1/3)
    isocenter_px[1] = np.round(iso_2/3)
    isocenter_px[2] = np.round(iso_3/3)

    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_reference_point_ROI[0][0])/3),\
            np.round((Globals.profiles_distance_reference_point_ROI[0][1])/3)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_reference_point_ROI[1][0])/3),\
            np.round((Globals.profiles_distance_reference_point_ROI[1][1])/3)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_reference_point_ROI[2][0])/3),\
            np.round((Globals.profiles_distance_reference_point_ROI[2][1])/3)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_reference_point_ROI[3][0])/3),\
            np.round((Globals.profiles_distance_reference_point_ROI[3][1])/3)])

    lateral_px = np.round(lateral/3)
    vertical_px = np.round(vertical/3)
    longit_px = np.round(longit/3)

    doseplan_lateral_displacement_px = \
        np.round((Globals.profiles_doseplan_lateral_displacement)/3)
    doseplan_vertical_displacement_px = \
        np.round((Globals.profiles_doseplan_vertical_displacement)/3)
    doseplan_longitudinal_displacement_px = \
        np.round((Globals.profiles_doseplan_longitudianl_displacement)/3)

temp_ref_point_doseplan = np.zeros(3)

if(Globals.profiles_doseplan_patient_position=='HFS'):
    temp_ref_point_doseplan[0] = \
        int(isocenter_px[0]+ doseplan_lateral_displacement_px - lateral_px)
    temp_ref_point_doseplan[1] = \
        int(isocenter_px[1]- doseplan_vertical_displacement_px + vertical_px)
    temp_ref_point_doseplan[2] = \
        int(isocenter_px[2]+ doseplan_longitudinal_displacement_px - longit_px)
elif(Globals.profiles_doseplan_patient_position=='HFP'):
    temp_ref_point_doseplan[0] = \
        isocenter_px[0]- doseplan_lateral_displacement_px+ lateral_px
    temp_ref_point_doseplan[1] = \
        isocenter_px[1]+ doseplan_vertical_displacement_px - vertical_px
    temp_ref_point_doseplan[2] = \
        isocenter_px[2]+ doseplan_longitudinal_displacement_px - longit_px
elif(Globals.profiles_doseplan_patient_position=='HFDR'):
    temp_ref_point_doseplan[0] = \
        isocenter_px[0]- doseplan_vertical_displacement_px + vertical_px
    temp_ref_point_doseplan[1] = \
        isocenter_px[1]+ doseplan_lateral_displacement_px - lateral_px
    temp_ref_point_doseplan[2] = \
        isocenter_px[2]+ doseplan_longitudinal_displacement_px - longit_px
elif(Globals.profiles_doseplan_patient_position=='HFDL'):
    temp_ref_point_doseplan[0] = \
        isocenter_px[0]+ doseplan_vertical_displacement_px - vertical_px
    temp_ref_point_doseplan[1] = \
        isocenter_px[1]- doseplan_lateral_displacement_px + lateral_px
    temp_ref_point_doseplan[2] = \
        isocenter_px[2]+ doseplan_longitudinal_displacement_px - longit_px
elif(Globals.profiles_doseplan_patient_position=='FFS'):
    temp_ref_point_doseplan[0] = \
        isocenter_px[0]- doseplan_lateral_displacement_px + lateral_px
    temp_ref_point_doseplan[1] = \
        isocenter_px[1]+ doseplan_vertical_displacement_px - vertical_px
    temp_ref_point_doseplan[2] = \
        isocenter_px[2]- doseplan_longitudinal_displacement_px + longit_px
elif(Globals.profiles_doseplan_patient_position=='FFP'):
    temp_ref_point_doseplan[0] = \
        isocenter_px[0]+ doseplan_lateral_displacement_px- lateral_px
    temp_ref_point_doseplan[1] = \
        isocenter_px[1]- doseplan_vertical_displacement_px + vertical_px
    temp_ref_point_doseplan[2] = \
        isocenter_px[2]- doseplan_longitudinal_displacement_px + longit_px
elif(Globals.profiles_doseplan_patient_position=='FFDR'):
    temp_ref_point_doseplan[0] = \
        isocenter_px[0]- doseplan_vertical_displacement_px + vertical_px
    temp_ref_point_doseplan[1] = \
        isocenter_px[1]- doseplan_lateral_displacement_px + lateral_px
    temp_ref_point_doseplan[2] = \
        isocenter_px[2]- doseplan_longitudinal_displacement_px + longit_px
else:
    temp_ref_point_doseplan[0] = \
        isocenter_px[0] + doseplan_vertical_displacement_px - vertical_px
    temp_ref_point_doseplan[1] = \
        isocenter_px[1] + doseplan_lateral_displacement_px - lateral_px
    temp_ref_point_doseplan[2] = \
        isocenter_px[2]- doseplan_longitudinal_displacement_px + longit_px

Globals.profiles_reference_point_in_doseplan = temp_ref_point_doseplan
reference_point = np.zeros(3)
```

```
########################## Doseplan #####################################
if(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
    ==[1, 0, 0, 0, 1, 0]):
    reference_point[0] = temp_ref_point_doseplan[2]
    reference_point[1] = temp_ref_point_doseplan[1]
    reference_point[2] = temp_ref_point_doseplan[0]
    if(Globals.profiles_film_orientation.get()=='Coronal'):
        dataset_swapped = \
            np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
        temp_ref = reference_point[0]
        reference_point[0] = reference_point[1]
        reference_point[1] = temp_ref
    elif(Globals.profiles_film_orientation.get()=='Sagittal'):
        dataset_swapped = \
            np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
        temp_ref = reference_point[0]
        reference_point[0] = reference_point[2]
        reference_point[2] = temp_ref
    elif(Globals.profiles_film_orientation.get()=='Axial'):
        dataset_swapped = Globals.profiles_dataset_doseplan.pixel_array
    else:
        messagebox.showerror("Error", "Something has gone wrong here.")
        clearAll()
        return
elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
    ==[1, 0, 0, 0, 0, 1]):
    reference_point[0] = temp_ref_point_doseplan[1]
    reference_point[1] = temp_ref_point_doseplan[2]
    reference_point[2] = temp_ref_point_doseplan[0]
    if(Globals.profiles_film_orientation.get()=='Coronal'):
        dataset_swapped = Globals.profiles_dataset_doseplan.pixel_array
    elif(Globals.profiles_film_orientation.get()=='Sagittal'):
        dataset_swapped = \
            np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
        temp_ref = reference_point[0]
        reference_point[0] = reference_point[2]
        reference_point[2] = temp_ref
        dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
        temp_ref = reference_point[1]
        reference_point[1] = reference_point[2]
        reference_point[2] = temp_ref
    elif(Globals.profiles_film_orientation.get()=='Axial'):
        dataset_swapped = \
            np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
        temp_ref = reference_point[0]
```

```
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
        ==[0, 1, 0, 1, 0, 0]):
        reference_point[0] = temp_ref_point_doseplan[2]
        reference_point[1] = temp_ref_point_doseplan[0]
        reference_point[2] = temp_ref_point_doseplan[1]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
        ==[0, 1, 0, 0, 0, 1]):
        reference_point[0] = temp_ref_point_doseplan[0]
        reference_point[1] = temp_ref_point_doseplan[2]
        reference_point[2] = temp_ref_point_doseplan[1]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped =\
                 np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped =\
                 np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped =\
                 np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
        ==[0, 0, 1, 1, 0, 0]):
        reference_point[0] = temp_ref_point_doseplan[1]
        reference_point[1] = temp_ref_point_doseplan[0]
        reference_point[2] = temp_ref_point_doseplan[2]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped\
                 = np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped \
                = np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped =\
                 np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
        ==[0, 0, 1, 0, 1, 0]):
        reference_point[0] = temp_ref_point_doseplan[0]
        reference_point[1] = temp_ref_point_doseplan[1]
        reference_point[2] = temp_ref_point_doseplan[2]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped =\
                 np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped = Globals.profiles_dataset_doseplan.pixel_array
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    else:
        messagebox.showerror("Error", "Something has gone wrong.")
        clearAll()
        return
    if(reference_point[0]<0 or reference_point[0]>dataset_swapped.shape[0]):
        messagebox.showerror("Error", "Reference point is outside of dosematrix\n\
            (Code: first dimension, number of frames in dosematrix)")
        return
    if(reference_point[1]<0 or reference_point[1]>dataset_swapped.shape[1]):
        messagebox.showerror("Error", "Reference point is outside of dosematrix\n\
            (Code: second dimension, rows in dosematrix)")
        return
    if(reference_point[2]<0 or reference_point[2]>dataset_swapped.shape[2]):
        messagebox.showerror("Error", "Reference point is outside of dosematrix\n\
            (Code: third dimension, columns in dosematrix)")
        return

    dose_slice = dataset_swapped[int(reference_point[0]),:,:]

    doseplan_ROI_coords = []
    top_left_test_side = False; top_left_test_down = False
    top_right_test_side = False; top_right_test_down = False
    bottom_left_test_side = False; bottom_left_test_down = False
    bottom_right_test_side = False; bottom_right_test_down = False
    top_left_side_corr = 0; top_left_down_corr = 0
    top_right_side_corr = 0; top_right_down_corr = 0
    bottom_left_side_corr = 0; bottom_left_down_corr = 0
    bottom_right_side_corr = 0; bottom_right_down_corr = 0


    top_left_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[0][0]
    top_left_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[0][1]
    if(top_left_to_side < 0):
        top_left_test_side = True
        top_left_side_corr = abs(top_left_to_side)
        top_left_to_side = 0
    if(top_left_to_side > dose_slice.shape[1]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(top_left_down < 0):
        top_left_test_down = True
        top_left_down_corr = abs(top_left_down)
        top_left_down = 0
    if(top_left_down > dose_slice.shape[0]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return

    top_right_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[1][0]
    top_right_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[1][1]
    if(top_right_to_side < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(top_right_to_side > dose_slice.shape[1]):
        top_right_test_side = True
        top_right_side_corr = top_right_to_side - dose_slice.shape[1]
        top_right_to_side = dose_slice.shape[1]
    if(top_right_down < 0):
        top_right_test_down = True
        top_right_down_corr = abs(top_right_down)
        top_right_down = 0
    if(top_right_down > dose_slice.shape[0]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return

    bottom_left_to_side = reference_point[2] -\
         distance_in_doseplan_ROI_reference_point_px[2][0]
    bottom_left_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[2][1]
    if(bottom_left_to_side < 0):
        bottom_left_test_side = True
        bottom_left_side_corr = abs(bottom_left_to_side)
        bottom_left_to_side = 0
    if(bottom_left_to_side > dose_slice.shape[1]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_left_down < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_left_down > dose_slice.shape[0]):
        bottom_left_down_corr = bottom_left_down - dose_slice.shape[0]
        bottom_left_down = dose_slice.shape[0]
        bottom_left_test_down = True

    bottom_right_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[3][0]
    bottom_right_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[3][1]
    if(bottom_right_to_side < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_right_to_side > dose_slice.shape[1]):
        bottom_right_side_corr = bottom_right_to_side - dose_slice.shape[1]
        bottom_right_to_side = dose_slice.shape[1]
        bottom_right_test_side = True
    if(bottom_right_down < 0):
        messagebox.showerror("Fatal Error", "Fatal error:\
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_right_down > dose_slice.shape[0]):
        bottom_right_down_corr = bottom_right_down - dose_slice.shape[0]
        bottom_right_down = dose_slice.shape[0]
        bottom_right_test_down = True


    if(top_right_test_side or top_right_test_down \
        or top_left_test_side or top_left_test_down \
            or bottom_right_test_side or bottom_right_test_down or \
                bottom_left_test_side or bottom_left_test_down):
        ROI_info = "Left side: " + str(max(top_left_side_corr, \
            bottom_left_side_corr)) + " pixels.\n" + "Right side: " + \
                str(max(top_right_side_corr, bottom_right_side_corr))+" pixels.\n "\
                    + "Top side: " + str(max(top_left_down_corr, \
                        top_right_down_corr)) + " pixels.\n" + "Bottom side: " \
                            +str(max(bottom_left_down_corr, bottom_right_down_corr))+\
                                " pixels."
        messagebox.showinfo("ROI info", "The ROI marked \
on the film did not fit with the size of the doseplan and had to \
            be cut.\n" + ROI_info )

    doseplan_ROI_coords.append([top_left_to_side, top_left_down])
    doseplan_ROI_coords.append([top_right_to_side, top_right_down])
    doseplan_ROI_coords.append([bottom_left_to_side, bottom_left_down])
    doseplan_ROI_coords.append([bottom_right_to_side, bottom_right_down])

    if only_one:
        Globals.profiles_doseplan_dataset_ROI = \
            dose_slice[int(top_left_down):int(bottom_left_down), \
                int(top_left_to_side):int(top_right_to_side)]\
                    *Globals.profiles_dataset_doseplan.DoseGridScaling

        img=Globals.profiles_doseplan_dataset_ROI
        if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
            img = cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5))
        elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
            img = cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10))
        else:
            img = cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15))

        mx=np.max(img)
        Globals.max_dose_doseplan = mx
        img = img/mx
        PIL_img_doseplan_ROI = Image.fromarray(np.uint8(cm.viridis(img)*255))

        wid = PIL_img_doseplan_ROI.width;heig = PIL_img_doseplan_ROI.height
        doseplan_canvas = tk.Canvas(Globals.profiles_film_panedwindow)
        doseplan_canvas.grid(row=2, column=0, sticky=N+S+W+E)
        Globals.profiles_film_panedwindow.add(doseplan_canvas, \
            height=max(heig, Globals.profiles_doseplan_text_image.height()), \
                width=wid + Globals.profiles_doseplan_text_image.width())
        doseplan_canvas.config(bg='#ffffff', relief=FLAT, highlightthickness=0, \
            height=max(heig, Globals.profiles_doseplan_text_image.height()), \
                width=wid + Globals.profiles_doseplan_text_image.width())


        Globals.doseplan_write_image = tk.Canvas(doseplan_canvas)
        Globals.doseplan_write_image.grid(row=0,column=1,sticky=N+S+W+E)
        Globals.doseplan_write_image.config(bg='#ffffff', relief=FLAT, \
            highlightthickness=0, width=wid, height=heig)

        doseplan_text_image_canvas = tk.Canvas(doseplan_canvas)
        doseplan_text_image_canvas.grid(row=0,column=0,sticky=N+S+W+E)
        doseplan_text_image_canvas.config(bg='#ffffff', relief=FLAT,\
            highlightthickness=0, width=Globals.profiles_doseplan_text_image\
                .width(), height=Globals.profiles_doseplan_text_image.height())
        scaled_image_visual = PIL_img_doseplan_ROI
        scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual)
        Globals.doseplan_write_image_width = scaled_image_visual.width()
        Globals.doseplan_write_image_height = scaled_image_visual.height()
        Globals.doseplan_write_image.create_image(0,0,\
            image=scaled_image_visual, anchor="nw")
        Globals.doseplan_write_image.image = scaled_image_visual
        doseplan_text_image_canvas.create_image(0,0,\
            image=Globals.profiles_doseplan_text_image, anchor="nw")
        doseplan_text_image_canvas.image=Globals.profiles_doseplan_text_image

        drawProfiles(False)

    else:
        img=dose_slice[int(top_left_down):int(bottom_left_down),\
             int(top_left_to_side):int(top_right_to_side)]

        Globals.profiles_doseplan_dataset_ROI_several.append(img)
        Globals.profiles_number_of_doseplans+=1

        if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
            Globals.profiles_several_img.append(img)
        elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
            Globals.profiles_several_img.append(img)
        else:
            Globals.profiles_several_img.append(img)


def processDoseplan_usingIsocenter(only_one):
#------------------------------------------------------
# Function to process the doseplan as the
# user has choosen to use isocenter
#
# This functions is a called in UploadDoseplan()
#------------------------------------------------------
    ################  RT Plan  ######################
    iso_1 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[0] -\
         Globals.profiles_dataset_rtplan.BeamSequence[0].\
             ControlPointSequence[0].IsocenterPosition[0])
    iso_2 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[1] -\
         Globals.profiles_dataset_rtplan.BeamSequence[0].\
             ControlPointSequence[0].IsocenterPosition[1])
    iso_3 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[2] -\
        Globals.profiles_dataset_rtplan.BeamSequence[0].\
            ControlPointSequence[0].IsocenterPosition[2])
```

```
Globals.profiles_isocenter_mm = [iso_1, iso_2, iso_3]

isocenter_px = np.zeros(3)
distance_in_doseplan_ROI_reference_point_px = []
if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
    isocenter_px[0] = np.round(iso_1)
    isocenter_px[1] = np.round(iso_2)
    isocenter_px[2] = np.round(iso_3)

    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round(Globals.profiles_distance_isocenter_ROI[0][0]),\
            np.round(Globals.profiles_distance_isocenter_ROI[0][1])])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round(Globals.profiles_distance_isocenter_ROI[1][0]),\
            np.round(Globals.profiles_distance_isocenter_ROI[1][1])])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round(Globals.profiles_distance_isocenter_ROI[2][0]),\
            np.round(Globals.profiles_distance_isocenter_ROI[2][1])])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round(Globals.profiles_distance_isocenter_ROI[3][0]),\
            np.round(Globals.profiles_distance_isocenter_ROI[3][1])])

elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
    isocenter_px[0] = np.round(iso_1/2)
    isocenter_px[1] = np.round(iso_2/2)
    isocenter_px[2] = np.round(iso_3/2)


    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_isocenter_ROI[0][0])/2),\
            np.round((Globals.profiles_distance_isocenter_ROI[0][1])/2)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_isocenter_ROI[1][0])/2),\
            np.round((Globals.profiles_distance_isocenter_ROI[1][1])/2)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_isocenter_ROI[2][0])/2),\
            np.round((Globals.profiles_distance_isocenter_ROI[2][1])/2)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_isocenter_ROI[3][0])/2),\
            np.round((Globals.profiles_distance_isocenter_ROI[3][1])/2)])

else:
    isocenter_px[0] = np.round(iso_1/3)
    isocenter_px[1] = np.round(iso_2/3)
    isocenter_px[2] = np.round(iso_3/3)

    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_isocenter_ROI[0][0])/3),\
            np.round((Globals.profiles_distance_isocenter_ROI[0][1])/3)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_isocenter_ROI[1][0])/3),\
            np.round((Globals.profiles_distance_isocenter_ROI[1][1])/3)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_isocenter_ROI[2][0])/3),\
            np.round((Globals.profiles_distance_isocenter_ROI[2][1])/3)])
    distance_in_doseplan_ROI_reference_point_px.append\
        ([np.round((Globals.profiles_distance_isocenter_ROI[3][0])/3),\
            np.round((Globals.profiles_distance_isocenter_ROI[3][1])/3)])

reference_point = np.zeros(3)

######################## Doseplan ##################################
if(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
    ==[1, 0, 0, 0, 1, 0]):
    reference_point[0] = isocenter_px[2]
    reference_point[1] = isocenter_px[1]
    reference_point[2] = isocenter_px[0]
    if(Globals.profiles_film_orientation.get()=='Coronal'):
        dataset_swapped = \
            np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
        temp_ref = reference_point[0]
        reference_point[0] = reference_point[1]
        reference_point[1] = temp_ref
    elif(Globals.profiles_film_orientation.get()=='Sagittal'):
        dataset_swapped = \
            np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
        temp_ref = reference_point[0]
        reference_point[0] = reference_point[2]
        reference_point[2] = temp_ref
    elif(Globals.profiles_film_orientation.get()=='Axial'):
        dataset_swapped = Globals.profiles_dataset_doseplan.pixel_array
    else:
        messagebox.showerror("Error", "Something has gone wrong here.")
        clearAll()
        return
elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
    ==[1, 0, 0, 0, 0, 1]):
    reference_point[0] = isocenter_px[1]
    reference_point[1] = isocenter_px[2]
```

```
        reference_point[2] = isocenter_px[0]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped = Globals.profiles_dataset_doseplan.pixel_array
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
        ==[0, 1, 0, 1, 0, 0]):
        reference_point[0] = isocenter_px[2]
        reference_point[1] = isocenter_px[0]
        reference_point[2] = isocenter_px[1]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped =\
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
        ==[0, 1, 0, 0, 0, 1]):
        reference_point[0] = isocenter_px[0]
        reference_point[1] = isocenter_px[2]
        reference_point[2] = isocenter_px[1]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
        ==[0, 0, 1, 1, 0, 0]):
        reference_point[0] = isocenter_px[1]
        reference_point[1] = isocenter_px[0]
        reference_point[2] = isocenter_px[2]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped =\
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient\
        ==[0, 0, 1, 0, 1, 0]):
        reference_point[0] = isocenter_px[0]
        reference_point[1] = isocenter_px[1]
        reference_point[2] = isocenter_px[2]
        if(Globals.profiles_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            dataset_swapped = np.swapaxes(dataset_swapped, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            dataset_swapped = Globals.profiles_dataset_doseplan.pixel_array
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    else:
        messagebox.showerror("Error", "Something has gone wrong.")
        clearAll()
        return


    ######################## Match film and doseplan ##################################

    if Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]:
        offset = int(np.round(Globals.profiles_offset))
        dose_slice = dataset_swapped[int(reference_point[0] + offset)]
    elif Globals.profiles_dataset_doseplan.PixelSpacing == [2, 2]:
        offset = int(np.round(Globals.profiles_offset/2))
        dose_slice = dataset_swapped[int(reference_point[0] + offset)]
    else:
        offset = int(np.round(Globals.profiles_offset/3))
        dose_slice = dataset_swapped[int(reference_point[0]+ offset)]


    doseplan_ROI_coords = []
    top_left_test_side = False; top_left_test_down = False
    top_right_test_side = False; top_right_test_down = False
    bottom_left_test_side = False; bottom_left_test_down = False
    bottom_right_test_side = False; bottom_right_test_down = False
    top_left_side_corr = 0; top_left_down_corr = 0
    top_right_side_corr = 0; top_right_down_corr = 0
    bottom_left_side_corr = 0; bottom_left_down_corr = 0
    bottom_right_side_corr = 0; bottom_right_down_corr = 0


    top_left_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[0][0]
    top_left_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[0][1]
    if(top_left_to_side < 0):
        top_left_test_side = True
        top_left_side_corr = abs(top_left_to_side)
        top_left_to_side = 0
    if(top_left_to_side > dose_slice.shape[1]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(top_left_down < 0):
        top_left_test_down = True
        top_left_down_corr = abs(top_left_down)
        top_left_down = 0
    if(top_left_down > dose_slice.shape[0]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return

    top_right_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[1][0]
    top_right_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[1][1]
    if(top_right_to_side < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(top_right_to_side > dose_slice.shape[1]):
        top_right_test_side = True
        top_right_side_corr = top_right_to_side - dose_slice.shape[1]
        top_right_to_side = dose_slice.shape[1]
    if(top_right_down < 0):
        top_right_test_down = True
        top_right_down_corr = abs(top_right_down)
        top_right_down = 0
    if(top_right_down > dose_slice.shape[0]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return

    bottom_left_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[2][0]
    bottom_left_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[2][1]
    if(bottom_left_to_side < 0):
        bottom_left_test_side = True
        bottom_left_side_corr = abs(bottom_left_to_side)
        bottom_left_to_side = 0
    if(bottom_left_to_side > dose_slice.shape[1]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_left_down < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_left_down > dose_slice.shape[0]):
        bottom_left_down_corr = bottom_left_down - dose_slice.shape[0]
        bottom_left_down = dose_slice.shape[0]
        bottom_left_test_down = True

    bottom_right_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[3][0]
    bottom_right_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[3][1]
    if(bottom_right_to_side < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
            marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_right_to_side > dose_slice.shape[1]):
        bottom_right_side_corr = bottom_right_to_side - dose_slice.shape[1]
        bottom_right_to_side = dose_slice.shape[1]
        bottom_right_test_side = True
    if(bottom_right_down < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
            marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_right_down > dose_slice.shape[0]):
        bottom_right_down_corr = bottom_right_down - dose_slice.shape[0]
        bottom_right_down = dose_slice.shape[0]
        bottom_right_test_down = True
    if(top_right_test_side or top_right_test_down or \
        top_left_test_side or top_left_test_down or bottom_right_test_side \
            or bottom_right_test_down or bottom_left_test_side or \
                bottom_left_test_down):
        ROI_info = "Left side: " + str(max(top_left_side_corr, \
            bottom_left_side_corr)) + " pixels.\n"+ "Right side: " + \
                str(max(top_right_side_corr, bottom_right_side_corr)) + \
                    " pixels.\n "+ "Top side: " + str(max(top_left_down_corr,\
                         top_right_down_corr)) + " pixels.\n"+ "Bottom side: " + \
                             str(max(bottom_left_down_corr, bottom_right_down_corr))\
                                  + " pixels."
        messagebox.showinfo("ROI info", "The ROI marked on the \
film did not fit with the size of the doseplan and had to \
            be cut.\n" + ROI_info )

    doseplan_ROI_coords.append([top_left_to_side, top_left_down])
    doseplan_ROI_coords.append([top_right_to_side, top_right_down])
    doseplan_ROI_coords.append([bottom_left_to_side, bottom_left_down])
    doseplan_ROI_coords.append([bottom_right_to_side, bottom_right_down])

    if(only_one):
        Globals.profiles_doseplan_dataset_ROI = \
            dose_slice[int(top_left_down):int(bottom_left_down), \
                int(top_left_to_side):int(top_right_to_side)]*\
                    Globals.profiles_dataset_doseplan.DoseGridScaling



        img=Globals.profiles_doseplan_dataset_ROI
        if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
            img = cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5))
        elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
            img = cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10))
        else:
            img = cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15))

        mx=np.max(img)
        Globals.max_dose_doseplan = mx
        max_dose = mx
        img = img/mx
        PIL_img_doseplan_ROI = Image.fromarray(np.uint8(cm.viridis(img)*255))

        wid = PIL_img_doseplan_ROI.width;heig = PIL_img_doseplan_ROI.height
        doseplan_canvas = tk.Canvas(Globals.profiles_film_panedwindow)
        doseplan_canvas.grid(row=2, column=0, sticky=N+S+W+E)
        Globals.profiles_film_panedwindow.add(doseplan_canvas, \
            height=max(heig, Globals.profiles_doseplan_text_image.height()), \
                width=wid + Globals.profiles_doseplan_text_image.width())
        doseplan_canvas.config(bg='#ffffff', relief=FLAT, highlightthickness=0, \
            height=max(heig, Globals.profiles_doseplan_text_image.height()), \
                width=wid + Globals.profiles_doseplan_text_image.width())



        Globals.doseplan_write_image = tk.Canvas(doseplan_canvas)
        Globals.doseplan_write_image.grid(row=0,column=1,sticky=N+S+W+E)
        Globals.doseplan_write_image.config(bg='#ffffff', relief=FLAT, \
            highlightthickness=0, width=wid, height=heig)

        doseplan_text_image_canvas = tk.Canvas(doseplan_canvas)
        doseplan_text_image_canvas.grid(row=0,column=0,sticky=N+S+W+E)
        doseplan_text_image_canvas.config(bg='#ffffff', relief=FLAT, \
            highlightthickness=0,width=Globals.profiles_doseplan_text_image.width(),\
                height=Globals.profiles_doseplan_text_image.height())

        scaled_image_visual = PIL_img_doseplan_ROI
        scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual)
        Globals.doseplan_write_image_width = scaled_image_visual.width()
        Globals.doseplan_write_image_height = scaled_image_visual.height()
        Globals.doseplan_write_image.create_image(0,0,\
            image=scaled_image_visual, anchor="nw")
        Globals.doseplan_write_image.image = scaled_image_visual
        doseplan_text_image_canvas.create_image(0,0,\
            image=Globals.profiles_doseplan_text_image, anchor="nw")
        doseplan_text_image_canvas.image=Globals.profiles_doseplan_text_image

        drawProfiles(False)

    else:
        img=dose_slice[int(top_left_down):int(bottom_left_down),\
             int(top_left_to_side):int(top_right_to_side)]

        Globals.profiles_doseplan_dataset_ROI_several.append(img)
        Globals.profiles_number_of_doseplans+=1

        if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
            Globals.profiles_several_img.append(img)
        elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
            Globals.profiles_several_img.append(img)
        else:
            Globals.profiles_several_img.append(img)
```

```
def UploadRTplan():
#--------------------------------------------------------
# Function to upload the RT Plan
#
# This functions is a callback to the button
# profiles_upload_button_rtplan in notebook.py
#--------------------------------------------------------
    file = filedialog.askopenfilename()
    ext = os.path.splitext(file)[-1].lower()
    if(not(ext == '.dcm')):
        if(ext == ""):
            return
        else:
            messagebox.showerror("Error", "The file must be a *.dcm file")
            return

    current_folder = os.getcwd()
    parent = os.path.dirname(file)
    os.chdir(parent)
    dataset = pydicom.dcmread(file)
    os.chdir(current_folder)
    Globals.profiles_dataset_rtplan = dataset

    try:
        isocenter_mm = \
            dataset.BeamSequence[0].ControlPointSequence[0].IsocenterPosition
        Globals.profiles_isocenter_mm = isocenter_mm

    except:
        messagebox.showerror("Error", "Could not read the \
RT plan file. Try again or try another file.\n\
(Code: isocenter reading)")
        return

    try:
        Globals.profiles_doseplan_vertical_displacement = \
            dataset.PatientSetupSequence[0].TableTopVerticalSetupDisplacement
    except:
        messagebox.showerror("Error", "Could not read the RT plan file.\
Try again or try another file. \n\
(Code: vertical table displacement)")

    try:
        Globals.profiles_doseplan_lateral_displacement = \
            dataset.PatientSetupSequence[0].TableTopLateralSetupDisplacement
    except:
        messagebox.showerror("Error", "Could not read the RT plan file. \
Try again or try another file—\n\
(Code: lateral table displacement)")

    try:
        Globals.profiles_doseplan_longitudianl_displacement = \
            dataset.PatientSetupSequence[0].TableTopLongitudinalSetupDisplacement
    except:
        messagebox.showerror("Error", "Could not read the RT plan file.\
 Try again or try another file\n\
(Code: longitudinal table displacement)")

    try:
        patient_position = dataset.PatientSetupSequence[0].PatientPosition
        Globals.profiles_doseplan_patient_position = patient_position
    except:
        messagebox.showerror("Error", "Could not read the RT plan file. \
Try again or try another file\n\
(Code: Patient position)")

    if(not(patient_position=='HFS' or patient_position=='HFP' or \
        patient_position=='HFDR' or patient_position == 'HFDL' or \
            patient_position=='FFDR' or patient_position=='FFDL' or \
                patient_position=='FFP' or patient_position=='FFS')):
        messagebox.showerror("Error","Fidora does only support patient positions:\n\
            HFS, HFP, HFDR, HFDL, FFP, FFS, FFDR, FFDL")
        return

    Globals.profiles_test_if_added_rtplan = True
    Globals.profiles_upload_button_doseplan.config(state=ACTIVE)
    Globals.profiles_upload_button_rtplan.config(state=DISABLED)

def UploadDoseplan_button_function():
    yes = messagebox.askyesno("Question", "Are you going to \
upload several doseplans and/or use a factor on a plan?")
    if not yes:
        UploadDoseplan(True)
        return
```

```
several_doseplans_window = tk.Toplevel(Globals.tab4_canvas)
several_doseplans_window.geometry("600x500+10+10")
several_doseplans_window.grab_set()

doseplans_over_all_frame = \
    tk.Frame(several_doseplans_window, bd=0, relief=FLAT)
doseplans_over_all_canvas = Canvas(doseplans_over_all_frame)

doseplans_xscrollbar = Scrollbar(doseplans_over_all_frame, \
    orient=HORIZONTAL, command=doseplans_over_all_canvas.xview)
doseplans_yscrollbar = Scrollbar(doseplans_over_all_frame, \
    command=doseplans_over_all_canvas.yview)

Globals.doseplans_scroll_frame = ttk.Frame(doseplans_over_all_canvas)
Globals.doseplans_scroll_frame.bind("<Configure>", \
    lambda e: doseplans_over_all_canvas.configure\
        (scrollregion=doseplans_over_all_canvas.bbox('all')))

doseplans_over_all_canvas.create_window((0,0), \
    window=Globals.doseplans_scroll_frame, anchor='nw')
doseplans_over_all_canvas.configure(xscrollcommand=doseplans_xscrollbar.set, \
    yscrollcommand=doseplans_yscrollbar.set)

doseplans_over_all_frame.config(highlightthickness=0, bg='#ffffff')
doseplans_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
doseplans_over_all_frame.pack(expand=True, fill=BOTH)
doseplans_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
doseplans_over_all_frame.grid_columnconfigure(0, weight=1)
doseplans_over_all_frame.grid_rowconfigure(0, weight=1)
doseplans_xscrollbar.grid(row=1, column=0, sticky=E+W)
doseplans_over_all_frame.grid_columnconfigure(1, weight=0)
doseplans_over_all_frame.grid_rowconfigure(1, weight=0)
doseplans_yscrollbar.grid(row=0, column=1, sticky=N+S)
doseplans_over_all_frame.grid_columnconfigure(2, weight=0)
doseplans_over_all_frame.grid_rowconfigure(2, weight=0)

upload_doseplan_frame = tk.Frame(Globals.doseplans_scroll_frame)
upload_doseplan_frame.grid(row=0, column = 0, padx = (30,30),\
     pady=(30,0), sticky=N+S+E+W)
Globals.doseplans_scroll_frame.grid_columnconfigure(0, weight=0)
Globals.doseplans_scroll_frame.grid_rowconfigure(0, weight=0)
upload_doseplan_frame.config(bg = '#ffffff')

upload_button_doseplan = tk.Button(upload_doseplan_frame, \
        text='Browse', image=Globals.profiles_add_doseplans_button_image,\
            cursor='hand2', font=('calibri', '14'), relief=FLAT, state=ACTIVE, \
                command=lambda: UploadDoseplan(False))
    upload_button_doseplan.pack(expand=True, fill=BOTH)
    upload_button_doseplan.configure(bg='#ffffff', activebackground='#ffffff', \
        activeforeground='#ffffff', highlightthickness=0)
    upload_button_doseplan.image = Globals.profiles_add_doseplans_button_image

    def closeUploadDoseplans():
        if(len(Globals.profiles_doseplan_dataset_ROI_several) == 0):
            messagebox.showinfo("INFO", "No doseplan has been uploaded")
            return
        for i in range(len(Globals.profiles_doseplan_dataset_ROI_several)):
            if Globals.profiles_doseplans_factor_input[i].get\
                ("1.0", 'end-1c') == " ":
                factor = 1
            else:
                try:
                    factor = float(Globals.profiles_doseplans_factor_input[i].\
                        get("1.0", 'end-1c'))
                except:
                    messagebox.showerror("Error", "Invalid factor. \
Must be number.\n (Code: closeUploadDoseplans)")
                    return
            if i == 0:
                doseplan_ROI = Globals.profiles_doseplan_dataset_ROI_several[i]\
                    *Globals.profiles_dose_scaling_doseplan[i]
                doseplan_ROI= doseplan_ROI*factor

                img_ROI = Globals.profiles_several_img[i]\
                    *Globals.profiles_dose_scaling_doseplan[i]
                img_ROI = img_ROI*factor
            else:
                doseplan_ROI+= \
                    factor*Globals.profiles_doseplan_dataset_ROI_several[i]\
                        *Globals.profiles_dose_scaling_doseplan[i]
                img_ROI+= factor*Globals.profiles_several_img[i]*\
                    Globals.profiles_dose_scaling_doseplan[i]


        img_ROI = cv2.resize(img_ROI, dsize=(img_ROI.shape[1]*5,img_ROI.shape[0]*5))
        Globals.profiles_doseplan_dataset_ROI = doseplan_ROI
        mx=np.max(img_ROI)
        Globals.max_dose_doseplan = mx
        img_ROI = img_ROI/mx
        PIL_img_doseplan_ROI = Image.fromarray(np.uint8(cm.viridis(img_ROI)*255))

        wid = PIL_img_doseplan_ROI.width;heig = PIL_img_doseplan_ROI.height
        doseplan_canvas = tk.Canvas(Globals.profiles_film_panedwindow)
        doseplan_canvas.grid(row=2, column=0, sticky=N+S+W+E)
        Globals.profiles_film_panedwindow.add(doseplan_canvas, \
            height=max(heig, Globals.profiles_doseplan_text_image.height()), \
                width=wid + Globals.profiles_doseplan_text_image.width())
        doseplan_canvas.config(bg='#ffffff', relief=FLAT, highlightthickness=0, \
            height=max(heig, Globals.profiles_doseplan_text_image.height()), \
                width=wid + Globals.profiles_doseplan_text_image.width())


        Globals.doseplan_write_image = tk.Canvas(doseplan_canvas)
        Globals.doseplan_write_image.grid(row=0,column=1,sticky=N+S+W+E)
        Globals.doseplan_write_image.config(bg='#ffffff', relief=FLAT, \
            highlightthickness=0, width=wid, height=heig)

        doseplan_text_image_canvas = tk.Canvas(doseplan_canvas)
        doseplan_text_image_canvas.grid(row=0,column=0,sticky=N+S+W+E)
        doseplan_text_image_canvas.config(bg='#ffffff', \
            relief=FLAT, highlightthickness=0, \
                width=Globals.profiles_doseplan_text_image.width(), \
                    height=Globals.profiles_doseplan_text_image.height())

        scaled_image_visual = PIL_img_doseplan_ROI
        scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual)
        Globals.doseplan_write_image_width = scaled_image_visual.width()
        Globals.doseplan_write_image_height = scaled_image_visual.height()
        Globals.doseplan_write_image.create_image(0,0,\
            image=scaled_image_visual, anchor="nw")
        Globals.doseplan_write_image.image = scaled_image_visual
        doseplan_text_image_canvas.create_image(0,0\
            ,image=Globals.profiles_doseplan_text_image, anchor="nw")
        doseplan_text_image_canvas.image=Globals.profiles_doseplan_text_image

        Globals.profiles_doseplan_dataset_ROI = doseplan_ROI

        Globals.profiles_upload_button_doseplan.config(state=DISABLED)

        several_doseplans_window.after(500, \
            lambda: several_doseplans_window.destroy())
        drawProfiles(False)

    doseplans_done_button_frame = tk.Frame(Globals.doseplans_scroll_frame)
    doseplans_done_button_frame.grid(row=0, column = 1, \
        padx=(0,40), pady=(30,0), sticky=N+S+W+E)
    doseplans_done_button_frame.config(bg='#ffffff')
    Globals.doseplans_scroll_frame.grid_rowconfigure(3, weight=0)
    Globals.doseplans_scroll_frame.grid_columnconfigure(3, weight=0)

    doseplans_done_button = tk.Button(doseplans_done_button_frame, \
        text='Done', image=Globals.done_button_image, cursor='hand2', \
            font=('calibri', '14'), relief=FLAT, state=ACTIVE, \
                command=closeUploadDoseplans)
    doseplans_done_button.pack(expand=True, fill=BOTH)
    doseplans_done_button.configure(bg='#ffffff', activebackground='#ffffff', \
        activeforeground='#ffffff', highlightthickness=0)
    doseplans_done_button.image = Globals.done_button_image



    filename_title = tk.Text(Globals.doseplans_scroll_frame, width = 15, height= 1)
    filename_title.insert(INSERT, "Filename")
    filename_title.grid(row=2, column=0, sticky=N+S+E+W, pady=(40,0), padx=(45,15))
    filename_title.config(bg='#ffffff', relief=FLAT, \
    state=DISABLED, font=('calibri', '15', 'bold'))
    Globals.doseplans_scroll_frame.grid_rowconfigure(1, weight=0)
    Globals.doseplans_scroll_frame.grid_columnconfigure(1, weight=0)

    factor_title = tk.Text(Globals.doseplans_scroll_frame, width=30, height=2)
    factor_title.insert(INSERT, "Here you can write a factor to use \n\
on the doseplan. Defaults to 1.")
    factor_title.grid(row=2, column=1, sticky=N+W+S+E, pady=(37,10), padx=(15,25))
    factor_title.config(bg='#ffffff', relief=FLAT, state=DISABLED, \
    font=('calibri', '15', 'bold'))
    Globals.doseplans_scroll_frame.grid_columnconfigure(2,weight=0)
    Globals.doseplans_scroll_frame.grid_rowconfigure(2, weight=0)



def UploadDoseplan(only_one):
#---------------------------------------------------------
# Function to upload the doseplan
#
# This functions is a callback funtion to the button
# profiles_upload_button_doseplan in notebook.py
#---------------------------------------------------------
    file = filedialog.askopenfilename()
    ext = os.path.splitext(file)[-1].lower()
    if(not(ext == '.dcm')):
        if(ext == ""):
            return
        else:
            messagebox.showerror("Error", "The file must be a *.dcm file")
            return

    current_folder = os.getcwd()
    parent = os.path.dirname(file)
    os.chdir(parent)
    dataset = pydicom.dcmread(file)
    try:
        dose_summation_type = dataset.DoseSummationType
    except:
        messagebox.showerror("Error", "Could not upload the doseplan correctly. \
Try again or another file.\n (Code: dose summation)")
        return

    if(not(dose_summation_type == "PLAN")):
        ok = messagebox.askokcancel("Dose summation", "You did not upload \
the full doseplan. Do you want to continue?")
        if not ok:
            return
    os.chdir(current_folder)
    doseplan_dataset = dataset.pixel_array
    if(not((dataset.PixelSpacing==[1, 1] and dataset.SliceThickness==1) \
        or (dataset.PixelSpacing==[2, 2] and dataset.SliceThickness==2) \
        or (dataset.PixelSpacing==[3, 3] and dataset.SliceThickness==3))):
        messagebox.showerror("Error", \
            "The resolution in doseplan must be 1x1x1, 2x2x2 or 3x3x3")
        return
    if(not(dataset.ImageOrientationPatient==[1, 0, 0, 0, 1, 0] or \
        dataset.ImageOrientationPatient==[1, 0, 0, 0, 0, 1] or \
        dataset.ImageOrientationPatient==[0, 1, 0, 1, 0, 0] or \
        dataset.ImageOrientationPatient==[0, 1, 0, 0, 0, 1] or \
        dataset.ImageOrientationPatient==[0, 0, 1, 1, 0, 0] or \
        dataset.ImageOrientationPatient==[0, 0, 1, 0, 1, 0])):
        messagebox.showerror("Error", "The Image Orientation \
(Patient) must be parallel to one of the main axis and \
perpendicular to the two others.")
        return

    if not only_one and Globals.profiles_number_of_doseplans > 1:
        if(not (Globals.profiles_dataset_doseplan.PixelSpacing\
            ==dataset.PixelSpacing)):
            messagebox.showerror("Error", "Resolution of the \
doseplans must be equal. \n(Code: UploadDoseplan)")
            return
        if(not (Globals.profiles_dataset_doseplan.DoseGridScaling \
            == dataset.DoseGridScaling)):
            messagebox.showerror("Error", "Dose grid scaling of \
the doseplans must be equal. \n(Code: UploadDoseplan)")
            return
    Globals.profiles_dataset_doseplan = dataset
    Globals.profiles_dose_scaling_doseplan.append(dataset.DoseGridScaling)
    Globals.profiles_test_if_added_doseplan = True
    if(Globals.profiles_test_if_added_rtplan):
        if(Globals.profiles_isocenter_or_reference_point == "Isocenter"):
            processDoseplan_usingIsocenter(only_one)
        elif(Globals.profiles_isocenter_or_reference_point == "Ref_point"):
            processDoseplan_usingReferencePoint(only_one)
        else:
            messagebox.showerror("Error", "Something went wrong. \
Try again.\n (Code: processDoseplan)")
            return

    if only_one:
        Globals.profiles_upload_button_doseplan.config(state=DISABLED)

    if not only_one:
        filename = basename(normpath(file))
        textbox_filename = tk.Text(Globals.doseplans_scroll_frame, \
            width = 30, height = 1)
        textbox_filename.insert(INSERT, filename)
        textbox_filename.config(bg='#ffffff', font=('calibri', '12'), \
            state=DISABLED, relief=FLAT)
        textbox_filename.grid(row = Globals.profiles_number_of_doseplans_row_count,\
             column = 0, sticky=N+S+W+E, pady=(10,10), padx=(10,10))
        Globals.doseplans_scroll_frame.grid_columnconfigure\
            (Globals.profiles_doseplans_grid_config_count, weight=0)
        Globals.doseplans_scroll_frame.grid_rowconfigure\
            (Globals.profiles_doseplans_grid_config_count, weight=0)
        Globals.profiles_doseplans_filenames.append(textbox_filename)

        Globals.profiles_doseplans_grid_config_count+=1;

        textbox_factor = tk.Text(Globals.doseplans_scroll_frame,\
            width = 6, height = 1)
        textbox_factor.insert(INSERT, "Factor: ")
        textbox_factor.config(bg='#ffffff', font=('calibri', '12'), \
        state=DISABLED, relief=FLAT)
        textbox_factor.grid(row = Globals.profiles_number_of_doseplans_row_count, \
            column = 1, sticky=N+S+W+E, pady=(10,10), padx=(10,10))
        Globals.doseplans_scroll_frame.grid_columnconfigure\
            (Globals.profiles_doseplans_grid_config_count, weight=0)
        Globals.doseplans_scroll_frame.grid_rowconfigure\
            (Globals.profiles_doseplans_grid_config_count, weight=0)
        Globals.profiles_doseplans_factor_text.append(textbox_factor)

        Globals.profiles_doseplans_grid_config_count+=1;

        textbox_factor_input = tk.Text(Globals.doseplans_scroll_frame, \
            width=3, height=1)
        textbox_factor_input.insert(INSERT, " ")
        textbox_factor_input.config(bg='#E5f9ff', font=('calibri', '12'), \
        state=NORMAL, bd = 2)
        textbox_factor_input.grid(row = \
            Globals.profiles_number_of_doseplans_row_count, column = 1, \
                sticky=N+S, pady=(10,10), padx=(40,10))
        Globals.doseplans_scroll_frame.grid_columnconfigure\
            (Globals.profiles_doseplans_grid_config_count, weight=0)
        Globals.doseplans_scroll_frame.grid_rowconfigure\
            (Globals.profiles_doseplans_grid_config_count, weight=0)
        Globals.profiles_doseplans_factor_input.append(textbox_factor_input)

        Globals.profiles_number_of_doseplans_row_count+=1
        Globals.profiles_doseplans_grid_config_count+=1;


def markIsocenter(img, new_window_isocenter_tab, image_canvas, cv2Img):
#---------------------------------------------------
# Function to mark the isocenter
#
# This functions is a callback to button
# mark_isocenter_button in uploadFilm()
#---------------------------------------------------
    if(len(Globals.profiles_mark_isocenter_oval)>0):
        image_canvas.delete(Globals.profiles_mark_isocenter_up_down_line[0])
        image_canvas.delete(Globals.profiles_mark_isocenter_right_left_line[0])
        image_canvas.delete(Globals.profiles_mark_isocenter_oval[0])

        Globals.profiles_mark_isocenter_oval=[]
        Globals.profiles_mark_isocenter_right_left_line=[]
        Globals.profiles_mark_isocenter_up_down_line=[]

    Globals.profiles_iscoenter_coords = []
```

```
img_mark_isocenter = ImageTk.PhotoImage(image=img)
mark_isocenter_window = tk.Toplevel(new_window_isocenter_tab)
mark_isocenter_window.geometry("1035x620+10+10")
mark_isocenter_window.grab_set()

mark_isocenter_over_all_frame = tk.Frame(mark_isocenter_window,\
     bd=0, relief=FLAT)
mark_isocenter_over_all_canvas = Canvas(mark_isocenter_over_all_frame)

mark_isocenter_xscrollbar = Scrollbar(mark_isocenter_over_all_frame,\
     orient=HORIZONTAL, command=mark_isocenter_over_all_canvas.xview)
mark_isocenter_yscrollbar = Scrollbar(mark_isocenter_over_all_frame, \
    command=mark_isocenter_over_all_canvas.yview)

mark_isocenter_scroll_frame = ttk.Frame(mark_isocenter_over_all_canvas)
mark_isocenter_scroll_frame.bind("<Configure>", lambda e: \
    mark_isocenter_over_all_canvas.configure\
        (scrollregion=mark_isocenter_over_all_canvas.bbox('all')))

mark_isocenter_over_all_canvas.create_window((0,0), \
    window=mark_isocenter_scroll_frame, anchor='nw')
mark_isocenter_over_all_canvas.configure\
    (xscrollcommand=mark_isocenter_xscrollbar.set, \
        yscrollcommand=mark_isocenter_yscrollbar.set)

mark_isocenter_over_all_frame.config(highlightthickness=0, bg='#ffffff')
mark_isocenter_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
mark_isocenter_over_all_frame.pack(expand=True, fill=BOTH)
mark_isocenter_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
mark_isocenter_over_all_frame.grid_columnconfigure(0, weight=1)
mark_isocenter_over_all_frame.grid_rowconfigure(0, weight=1)
mark_isocenter_xscrollbar.grid(row=1, column=0, sticky=E+W)
mark_isocenter_over_all_frame.grid_columnconfigure(1, weight=0)
mark_isocenter_over_all_frame.grid_rowconfigure(1, weight=0)
mark_isocenter_yscrollbar.grid(row=0, column=1, sticky=N+S)
mark_isocenter_over_all_frame.grid_columnconfigure(2, weight=0)
mark_isocenter_over_all_frame.grid_rowconfigure(2, weight=0)

mark_isocenter_image_canvas = tk.Canvas(mark_isocenter_scroll_frame)
mark_isocenter_image_canvas.grid(row=0,column=0, rowspan=10, \
    columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
mark_isocenter_scroll_frame.grid_columnconfigure(0, weight=0)
mark_isocenter_scroll_frame.grid_rowconfigure(0, weight=0)

mark_isocenter_image_canvas.create_image(0,0,\
    image=img_mark_isocenter,anchor="nw")
mark_isocenter_image_canvas.image = img_mark_isocenter
mark_isocenter_image_canvas.config(cursor='hand2', bg='#ffffff', \
    relief=FLAT, bd=0, scrollregion=mark_isocenter_image_canvas.bbox(ALL), \
        height=img_mark_isocenter.height(), width=img_mark_isocenter.width())
mark_isocenter_image_canvas.grid_propagate(0)

def findCoords(event):
    mark_isocenter_image_canvas.create_oval\
        (event.x-2, event.y-2, event.x+2, event.y+2, fill='red')
    if(Globals.profiles_iscoenter_coords==[]):
        Globals.profiles_iscoenter_coords.append([event.x, event.y])
        mark_isocenter_image_canvas.config(cursor='hand2')

    elif(len(Globals.profiles_iscoenter_coords)==1):
        Globals.profiles_iscoenter_coords.append([event.x, event.y])
        Globals.profiles_film_isocenter = \
            [Globals.profiles_iscoenter_coords[0][0], \
                Globals.profiles_iscoenter_coords[1][1]]
        x1,y1 = Globals.profiles_iscoenter_coords[0]
        x4,y4 = Globals.profiles_iscoenter_coords[1]
        x2 = x1;y3=y4
        y2=2*Globals.profiles_film_isocenter[1]-y1
        x3=2*Globals.profiles_film_isocenter[0]-x4
        up_down_line = image_canvas.create_line\
            (int(x1/2),int(y1/2),int(x2/2),int(y2/2),\
                fill='purple', smooth=1, width=2)
        right_left_line = image_canvas.create_line\
            (int(x3/2),int(y3/2),int(x4/2),int(y4/2), \
                fill='purple', smooth=1, width=2)
        oval = image_canvas.create_oval\
            (int(Globals.profiles_film_isocenter[0]/2)-3, \
                int(Globals.profiles_film_isocenter[1]/2)-3,\
                    int(Globals.profiles_film_isocenter[0]/2)+3, \
                        int(Globals.profiles_film_isocenter[1]/2)+3, fill='red')

        Globals.profiles_mark_isocenter_up_down_line.append(up_down_line)
        Globals.profiles_mark_isocenter_right_left_line.append(right_left_line)
        Globals.profiles_mark_isocenter_oval.append(oval)

        mark_isocenter_window.after(500, lambda: mark_isocenter_window.destroy())
        Globals.profiles_isocenter_check = True
        if(Globals.profiles_ROI_check):
            Globals.profiles_done_button.config(state=ACTIVE)
mark_isocenter_image_canvas.bind("<Button 1>",findCoords)


def markReferencePoint(img, new_window_reference_point_tab, \
image_canvas_reference_tab, cv2Img):
#-------------------------------------------------------------
# Function to mark the reference point
#
# This functions is a callback to the button
# mark_point_button in uploadFilm()
#-------------------------------------------------------------

if(len(Globals.profiles_mark_reference_point_oval)>0):
    image_canvas_reference_tab.delete\
        (Globals.profiles_mark_reference_point_oval[0])
    Globals.profiles_mark_reference_point_oval=[]

img_mark_reference_point = ImageTk.PhotoImage(image=img)
mark_reference_point_window = tk.Toplevel(new_window_reference_point_tab)
mark_reference_point_window.geometry("1035x620+10+10")
mark_reference_point_window.grab_set()

mark_reference_point_over_all_frame = \
    tk.Frame(mark_reference_point_window, bd=0, relief=FLAT)
mark_reference_point_over_all_canvas = \
    Canvas(mark_reference_point_over_all_frame)

mark_reference_point_xscrollbar = \
    Scrollbar(mark_reference_point_over_all_frame, orient=HORIZONTAL, \
        command=mark_reference_point_over_all_canvas.xview)
mark_reference_point_yscrollbar = \
    Scrollbar(mark_reference_point_over_all_frame, \
        command=mark_reference_point_over_all_canvas.yview)

mark_reference_point_scroll_frame = \
    ttk.Frame(mark_reference_point_over_all_canvas)
mark_reference_point_scroll_frame.bind("<Configure>", \
    lambda e: mark_reference_point_over_all_canvas.configure\
        (scrollregion=mark_reference_point_over_all_canvas.bbox('all')))

mark_reference_point_over_all_canvas.create_window((0,0), \
    window=mark_reference_point_scroll_frame, anchor='nw')
mark_reference_point_over_all_canvas.configure\
    (xscrollcommand=mark_reference_point_xscrollbar.set, \
        yscrollcommand=mark_reference_point_yscrollbar.set)
```

```
mark_reference_point_over_all_frame.config(highlightthickness=0, bg='#ffffff')
mark_reference_point_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
mark_reference_point_over_all_frame.pack(expand=True, fill=BOTH)
mark_reference_point_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
mark_reference_point_over_all_frame.grid_columnconfigure(0, weight=1)
mark_reference_point_over_all_frame.grid_rowconfigure(0, weight=1)
mark_reference_point_xscrollbar.grid(row=1, column=0, sticky=E+W)
mark_reference_point_over_all_frame.grid_columnconfigure(1, weight=0)
mark_reference_point_over_all_frame.grid_rowconfigure(1, weight=0)
mark_reference_point_yscrollbar.grid(row=0, column=1, sticky=N+S)
mark_reference_point_over_all_frame.grid_columnconfigure(2, weight=0)
mark_reference_point_over_all_frame.grid_rowconfigure(2, weight=0)

mark_reference_point_image_canvas = \
    tk.Canvas(mark_reference_point_scroll_frame)
mark_reference_point_image_canvas.grid(row=0,column=0, rowspan=10, \
    columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
mark_reference_point_scroll_frame.grid_columnconfigure(0, weight=0)
mark_reference_point_scroll_frame.grid_rowconfigure(0, weight=0)

mark_reference_point_image_canvas.create_image(0,0,\
    image=img_mark_reference_point,anchor="nw")
mark_reference_point_image_canvas.image = img_mark_reference_point
mark_reference_point_image_canvas.config(cursor='hand2', \
    bg='#ffffff', relief=FLAT, bd=0, \
    scrollregion=mark_reference_point_image_canvas.bbox(ALL), \
        height=img_mark_reference_point.height(), \
            width=img_mark_reference_point.width())
mark_reference_point_image_canvas.grid_propagate(0)


def findCoords(event):
    mark_reference_point_image_canvas.create_oval\
        (event.x-2, event.y-2, event.x+2, event.y+2, fill='red')
    Globals.profiles_film_reference_point = [event.x, event.y]
    oval = image_canvas_reference_tab.create_oval\
        (int(Globals.profiles_film_reference_point[0]/2)-3, \
            int(Globals.profiles_film_reference_point[1]/2)-3, \
                int(Globals.profiles_film_reference_point[0]/2)+3, \
                    int(Globals.profiles_film_reference_point[1]/2)+3,fill='red')
    Globals.profiles_mark_reference_point_oval.append(oval)

    mark_reference_point_window.after(500, \
        lambda: mark_reference_point_window.destroy())
        Globals.profiles_reference_point_check = True
        if(Globals.profiles_ROI_reference_point_check):
            Globals.profiles_done_button_reference_point.config(state=ACTIVE)

    mark_reference_point_image_canvas.bind("<Button 1>",findCoords)

def markROI(img, tab, canvas, ref_point_test):
#------------------------------------------------
# Function to mark the ROI
#
# This functions is a callback to the button
# mark_ROI_button and mark_ROI_reference_point_button
# in UploadFilm()
#------------------------------------------------
    if(len(Globals.profiles_mark_ROI_rectangle)>0):
        canvas.delete(Globals.profiles_mark_ROI_rectangle[0])
        Globals.profiles_mark_ROI_rectangle = []

    Globals.profiles_ROI_coords = []

    img_mark_ROI = ImageTk.PhotoImage(image=img)
    mark_ROI_window = tk.Toplevel(tab)
    mark_ROI_window.geometry("1035x620+10+10")
    mark_ROI_window.grab_set()

    mark_ROI_over_all_frame = tk.Frame(mark_ROI_window, bd=0, relief=FLAT)
    mark_ROI_over_all_canvas = Canvas(mark_ROI_over_all_frame)

    mark_ROI_xscrollbar = Scrollbar(mark_ROI_over_all_frame, \
        orient=HORIZONTAL, command=mark_ROI_over_all_canvas.xview)
    mark_ROI_yscrollbar = Scrollbar(mark_ROI_over_all_frame, \
        command=mark_ROI_over_all_canvas.yview)

    mark_ROI_scroll_frame = ttk.Frame(mark_ROI_over_all_canvas)
    mark_ROI_scroll_frame.bind("<Configure>", lambda e: \
        mark_ROI_over_all_canvas.configure\
            (scrollregion=mark_ROI_over_all_canvas.bbox('all')))

    mark_ROI_over_all_canvas.create_window((0,0), \
        window=mark_ROI_scroll_frame, anchor='nw')
    mark_ROI_over_all_canvas.configure\
        (xscrollcommand=mark_ROI_xscrollbar.set, \
            yscrollcommand=mark_ROI_yscrollbar.set)

    mark_ROI_over_all_frame.config(highlightthickness=0, bg='#ffffff')
```

```
mark_ROI_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
mark_ROI_over_all_frame.pack(expand=True, fill=BOTH)
mark_ROI_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
mark_ROI_over_all_frame.grid_columnconfigure(0, weight=1)
mark_ROI_over_all_frame.grid_rowconfigure(0, weight=1)
mark_ROI_xscrollbar.grid(row=1, column=0, sticky=E+W)
mark_ROI_over_all_frame.grid_columnconfigure(1, weight=0)
mark_ROI_over_all_frame.grid_rowconfigure(1, weight=0)
mark_ROI_yscrollbar.grid(row=0, column=1, sticky=N+S)
mark_ROI_over_all_frame.grid_columnconfigure(2, weight=0)
mark_ROI_over_all_frame.grid_rowconfigure(2, weight=0)

mark_ROI_image_canvas = tk.Canvas(mark_ROI_scroll_frame)
mark_ROI_image_canvas.grid(row=0,column=0, rowspan=10, columnspan=3, \
    sticky=N+S+E+W, padx=(0,0), pady=(0,0))
mark_ROI_scroll_frame.grid_columnconfigure(0, weight=0)
mark_ROI_scroll_frame.grid_rowconfigure(0, weight=0)
mark_ROI_image_canvas.create_image(0,0,image=img_mark_ROI,anchor="nw")
mark_ROI_image_canvas.image = img_mark_ROI
mark_ROI_image_canvas.config(bg='#E5f9ff', relief=FLAT, bd=0, \
    scrollregion=mark_ROI_image_canvas.bbox(ALL), height=img_mark_ROI.height(),\
        width=img_mark_ROI.width())
mark_ROI_image_canvas.grid_propagate(0)

rectangle = mark_ROI_image_canvas.create_rectangle(0,0,0,0,outline='green')
rectangle_top_corner = []
rectangle_bottom_corner = []
def buttonPushed(event):
    rectangle_top_corner.append([event.x, event.y])

def buttonMoving(event):
    mark_ROI_image_canvas.coords(rectangle, rectangle_top_corner[0][0], \
        rectangle_top_corner[0][1], \
    event.x, event.y)

def buttonReleased(event):
    rectangle_bottom_corner.append([event.x, event.y])
    mark_ROI_image_canvas.coords(rectangle, rectangle_top_corner[0][0], \
        rectangle_top_corner[0][1],\
    rectangle_bottom_corner[0][0], rectangle_bottom_corner[0][1])
    mark_ROI_image_canvas.itemconfig(rectangle, outline='Blue')
    Globals.profiles_ROI_coords.append\
        ([rectangle_top_corner[0][0], rectangle_top_corner[0][1]])
    Globals.profiles_ROI_coords.append\
        ([rectangle_bottom_corner[0][0], rectangle_top_corner[0][1]])
        Globals.profiles_ROI_coords.append\
            ([rectangle_top_corner[0][0], rectangle_bottom_corner[0][1]])
        Globals.profiles_ROI_coords.append\
            ([rectangle_bottom_corner[0][0], rectangle_bottom_corner[0][1]])

        rect = canvas.create_rectangle\
            (int((rectangle_top_corner[0][0])/2), \
                int((rectangle_top_corner[0][1])/2),\
                    int((rectangle_bottom_corner[0][0])/2), \
                        int((rectangle_bottom_corner[0][1])/2), \
                            outline='Blue', width=2)
        Globals.profiles_mark_ROI_rectangle.append(rect)

        if(ref_point_test):
            Globals.profiles_ROI_reference_point_check = True
            if(Globals.profiles_reference_point_check):
                Globals.profiles_done_button_reference_point.config(state=ACTIVE)
        else:
            Globals.profiles_ROI_check = True
            if(Globals.profiles_isocenter_check):
                Globals.profiles_done_button.config(state=ACTIVE)



        mark_ROI_window.after(500, lambda: mark_ROI_window.destroy())

    mark_ROI_image_canvas.bind("<B1-Motion>", buttonMoving)
    mark_ROI_image_canvas.bind("<Button-1>", buttonPushed)
    mark_ROI_image_canvas.bind("<ButtonRelease-1>", buttonReleased)



def UploadFilm():
    if(Globals.profiles_film_orientation.get() == '-'):
        messagebox.showerror("Missing parameter", "Film orientation missing \n\
 (Code: UploadFilm)")
        return
    if Globals.profiles_film_factor_input.get("1.0", 'end-1c') == " ":
        Globals.profiles_film_factor = 1
    else:
        try:
            Globals.profiles_film_factor = \
                float(Globals.profiles_film_factor_input.get("1.0", 'end-1c'))
        except:
            messagebox.showerror("Missing parameter", "Film factor invalid \
format. \n (Code: UploadFilm)")
            return
    file = filedialog.askopenfilename()
    ext = os.path.splitext(file)[-1].lower()
    if(ext == '.tif'):
        current_folder = os.getcwd()
        parent = os.path.dirname(file)
        os.chdir(parent)
        img = Image.open(file)
        img = img.transpose(Image.FLIP_LEFT_RIGHT)
        cv2Img = cv2.imread(basename(normpath(file)), \
            cv2.IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
        cv2Img = cv2.medianBlur(cv2Img, 5)
        if(cv2Img is None):
            messagebox.showerror("Error", "Something has gone wrong. \
Check that the filename only contain english letters")
            return
        if(cv2Img.shape[2] == 3):
            if(cv2Img.shape[0]==1270 and cv2Img.shape[1]==1016):
                cv2Img = abs(cv2Img-Globals.correctionMatrix127)
                cv2Img = np.clip(cv2Img, 0, 65535)
                cv2Img = cv2.flip(cv2Img,1)
                img_scaled = img.resize((508, 635), Image.ANTIALIAS)
                img_scaled = ImageTk.PhotoImage(image=img_scaled)


                Globals.profiles_film_dataset = cv2Img
                Globals.profiles_film_dataset_red_channel = cv2Img[:,:,2]
            else:
                messagebox.showerror("Error","The resolution of the \
image is not consistent with dpi")
                return
        else:
            messagebox.showerror("Error","The uploaded image \
need to be in RGB-format")
            return

        os.chdir(current_folder)

        if(not (img.width == 1016)):
            messagebox.showerror("Error", "Dpi in image has to be 127")
            return

        Globals.profiles_film_orientation_menu.configure(state=DISABLED)
        Globals.profiles_film_factor_input.config(state=DISABLED)
        h = 635 + 20
        w = 508 + 625
        new_window = tk.Toplevel(Globals.tab4)
        new_window.geometry("%dx%d+0+0" % (w, h))
        new_window.grab_set()

        new_window_over_all_frame = tk.Frame(new_window, bd=0, relief=FLAT)
        new_window_over_all_canvas = Canvas(new_window_over_all_frame)

        new_window_xscrollbar = Scrollbar(new_window_over_all_frame, \
            orient=HORIZONTAL, command=new_window_over_all_canvas.xview)
        new_window_yscrollbar = Scrollbar(new_window_over_all_frame, \
            command=new_window_over_all_canvas.yview)

        new_window_scroll_frame = ttk.Frame(new_window_over_all_canvas)
        new_window_scroll_frame.bind("<Configure>", lambda e: \
            new_window_over_all_canvas.configure\
                (scrollregion=new_window_over_all_canvas.bbox('all')))

        new_window_over_all_canvas.create_window((0,0), \
            window=new_window_scroll_frame, anchor='nw')
        new_window_over_all_canvas.configure\
            (xscrollcommand=new_window_xscrollbar.set, \
                yscrollcommand=new_window_yscrollbar.set)

        new_window_over_all_frame.config(highlightthickness=0, bg='#ffffff')
        new_window_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
        new_window_over_all_frame.pack(expand=True, fill=BOTH)
        new_window_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
        new_window_over_all_frame.grid_columnconfigure(0, weight=1)
        new_window_over_all_frame.grid_rowconfigure(0, weight=1)
        new_window_xscrollbar.grid(row=1, column=0, sticky=E+W)
        new_window_over_all_frame.grid_columnconfigure(1, weight=0)
        new_window_over_all_frame.grid_rowconfigure(1, weight=0)
        new_window_yscrollbar.grid(row=0, column=1, sticky=N+S)
        new_window_over_all_frame.grid_columnconfigure(2, weight=0)
        new_window_over_all_frame.grid_rowconfigure(2, weight=0)

        new_window_explain_text = tk.Text(new_window_scroll_frame, \
            height= 3, width=120)
        new_window_explain_text.insert(INSERT, \
"To match the film with the doseplan you have to mark either isocenter or \
a reference point on the film of your choice.In the case of the reference \
point you \nwill be asked to input the lenght in lateral, longitudinal and \
vertical to a reference point used in the linac. It the reference point in \
the film is the same as \nthe one in the phantom/linac you can input all zeros, \
in other cases your input is in mm. Later you will have the oppertunity to make \
small adjustments \nto the placement of either the reference point or isocenter.")
        new_window_explain_text.config(state=DISABLED, \
            font=('calibri', '13', 'bold'), bg = '#ffffff', relief=FLAT)
        new_window_explain_text.grid(row=0, column=0, \
            columnspan=5, sticky=N+S+W+E, pady=(15,5), padx=(10,10))
        new_window_scroll_frame.grid_rowconfigure(0, weight=0)
        new_window_scroll_frame.grid_columnconfigure(0, weight=0)

        new_window_notebook = ttk.Notebook(new_window_scroll_frame)
        new_window_notebook.borderWidth=0
        new_window_notebook.grid(row=2, column=0, columnspan=5, \
            sticky=E+W+N+S, pady=(0,0), padx =(0,0))
        new_window_scroll_frame.grid_rowconfigure(4, weight=0)
        new_window_scroll_frame.grid_columnconfigure(4, weight=0)

        new_window_isocenter_tab = ttk.Frame(new_window_notebook)
        new_window_notebook.add(new_window_isocenter_tab, text='Isocenter')
        new_window_reference_point_tab = ttk.Frame(new_window_notebook)
        new_window_notebook.add\
            (new_window_reference_point_tab, text='Reference point')
        new_window_manually_tab = ttk.Frame(new_window_notebook)
        new_window_notebook.add(new_window_manually_tab, text='Manually')


        image_canvas = tk.Canvas(new_window_isocenter_tab)
        image_canvas.grid(row=0,column=0, rowspan=12, columnspan=3, \
            sticky=N+S+E+W, padx=(0,0), pady=(0,0))
        new_window_isocenter_tab.grid_rowconfigure(1, weight=0)
        new_window_isocenter_tab.grid_columnconfigure(1, weight=0)
        image_canvas.create_image(0,0,image=img_scaled,anchor="nw")
        image_canvas.image = img_scaled
        image_canvas.config(bg='#ffffff', relief=FLAT, bd=0, \
        scrollregion=image_canvas.bbox(ALL), \
            height=img_scaled.height(), width=img_scaled.width())
        image_canvas.grid_propagate(0)

        image_canvas_reference_tab = tk.Canvas(new_window_reference_point_tab)
        image_canvas_reference_tab.grid(row=0,column=0, rowspan=10, \
            columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
        new_window_reference_point_tab.grid_rowconfigure(1, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(1, weight=0)
        image_canvas_reference_tab.create_image(0,0,image=img_scaled,anchor="nw")
        image_canvas_reference_tab.image = img_scaled
        image_canvas_reference_tab.config(bg='#ffffff', relief=FLAT, \
        bd=0, scrollregion=image_canvas.bbox(ALL), \
            height=img_scaled.height(), width=img_scaled.width())
        image_canvas_reference_tab.grid_propagate(0)

        film_window_mark_isocenter_text = tk.Text(new_window_isocenter_tab, \
            width=55, height=7)
        film_window_mark_isocenter_text.insert(INSERT, \
"When clicking the button \"Mark isocenter\" a window showing \n\
the image will appear and you are to click on the markers \n\
made on the film upon irradiation to find the isocenter. Start \n\
with the marker showing the direction of the film (see the \n\
specifications in main window). When both marks are made \n\
you will see the isocenter in the image. If you are not happy \n\
with the placement click the button again and repeat.")
        film_window_mark_isocenter_text.config(bg='#ffffff', relief=FLAT,\
         bd=0, state=DISABLED, font=('calibri', '11'))
        film_window_mark_isocenter_text.grid(row=0, column=3, rowspan=3, \
            sticky=N+S+E+W, padx=(10,10), pady=(10,0))
        new_window_isocenter_tab.columnconfigure(2, weight=0)
        new_window_isocenter_tab.rowconfigure(2, weight=0)

        film_window_mark_reference_point_text = \
            tk.Text(new_window_reference_point_tab, width=55, height=5)
        film_window_mark_reference_point_text.insert(INSERT, \
"When clicking the button \"Mark point\" a window showing \n\
the image will appear and you are to click on the marker \n\
made on the film upon irradiation to find the point. When\n\
the mark are made you will see the isocenter in the image.\n\
If you are not happy with the placement click the button \n\
again and repeat.")
        film_window_mark_reference_point_text.config(bg='#ffffff', \
        relief=FLAT, bd=0, state=DISABLED, font=('calibri', '11'))
        film_window_mark_reference_point_text.grid(row=0, column=3, \
            rowspan=3, sticky=N+S+E+W, padx=(10,10), pady=(5,0))
        new_window_reference_point_tab.columnconfigure(2, weight=0)
        new_window_reference_point_tab.rowconfigure(2, weight=0)

        mark_isocenter_button_frame = tk.Frame(new_window_isocenter_tab)
        mark_isocenter_button_frame.grid(row=3, column=3, \
            padx=(10,10), pady=(0,10))
        mark_isocenter_button_frame.configure(bg='#ffffff')
        new_window_isocenter_tab.grid_columnconfigure(3, weight=0)
        new_window_isocenter_tab.grid_rowconfigure(3, weight=0)
        mark_isocenter_button = tk.Button(mark_isocenter_button_frame, \
            text='Browse', image=Globals.profiles_mark_isocenter_button_image,\
                cursor='hand2',font=('calibri', '14'), relief=FLAT, state=ACTIVE,\
                    command=lambda: markIsocenter(img, new_window_isocenter_tab, \
                        image_canvas, cv2Img))
        mark_isocenter_button.pack(expand=True, fill=BOTH)
        mark_isocenter_button.config(bg='#ffffff', activebackground='#ffffff', \
            activeforeground='#ffffff', highlightthickness=0)
        mark_isocenter_button.image=Globals.profiles_mark_isocenter_button_image

        mark_point_button_frame = tk.Frame(new_window_reference_point_tab)
        mark_point_button_frame.grid(row=3, column=3, padx=(10,10), pady=(30,0))
        mark_point_button_frame.configure(bg='#ffffff')
        new_window_reference_point_tab.grid_columnconfigure(3, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(3, weight=0)

        mark_point_button = tk.Button(mark_point_button_frame, text='Browse', \
            image=Globals.profiles_mark_point_button_image,cursor='hand2',\
                font=('calibri', '14'), relief=FLAT, state=ACTIVE, command=lambda:\
                    markReferencePoint(img, new_window_reference_point_tab, \
                        image_canvas_reference_tab, cv2Img))
        mark_point_button.pack(expand=True, fill=BOTH)
        mark_point_button.config(bg='#ffffff', activebackground='#ffffff', \
            activeforeground='#ffffff', highlightthickness=0)
        mark_point_button.image=Globals.profiles_mark_point_button_image

        write_displacement_relative_to_reference_point = \
            tk.Text(new_window_reference_point_tab, width = 55, height=3)
        write_displacement_relative_to_reference_point.insert(INSERT, "\
If the marked reference points in the film does not match\n\
the reference point in the phantom you can write the\n\
displacemnet here (in mm). Defaults to zero ")
        write_displacement_relative_to_reference_point.grid(row=4, \
            column=3, rowspan=2, sticky=N+S+E+W, padx=(10,10), pady=(0,10))
        write_displacement_relative_to_reference_point.config(bg='#ffffff', \
        relief=FLAT, bd=0, state=DISABLED, font=('calibri', '11'))
        new_window_reference_point_tab.grid_rowconfigure(6, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(6, weight=0)

        input_lateral_text = tk.Text(new_window_reference_point_tab, \
            width=12, height=1)
        input_lateral_text.insert(INSERT, "Lateral:")
        input_lateral_text.config(bg='#ffffff', relief=FLAT, bd=0, \
        state=DISABLED, font=('calibri', '10'))
        input_lateral_text.grid(row=5, column=3, sticky=N+S, \
            padx=(0,250), pady=(25,0))
        new_window_reference_point_tab.grid_rowconfigure(10, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(10, weight=0)

        Globals.profiles_input_lateral_displacement = \
            tk.Text(new_window_reference_point_tab, width=5, height=1)
        Globals.profiles_input_lateral_displacement.insert(INSERT, " ")
        Globals.profiles_input_lateral_displacement.config(bg='#E5f9ff', \
        relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
        Globals.profiles_input_lateral_displacement.grid(row=5, column=3, \
            padx=(0,285), pady=(35,0))
        new_window_reference_point_tab.grid_rowconfigure(7, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(7, weight=0)

        input_vertical_text = tk.Text(new_window_reference_point_tab, \
            width=12, height=1)
        input_vertical_text.insert(INSERT, "Vertical:")
        input_vertical_text.config(bg='#ffffff', relief=FLAT, bd=0, \
        state=DISABLED, font=('calibri', '10'))
        input_vertical_text.grid(row=5, column=3, sticky=N+S, \
            padx=(0,0), pady=(25,0))
        new_window_reference_point_tab.grid_rowconfigure(11, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(11, weight=0)

        Globals.profiles_input_vertical_displacement = \
            tk.Text(new_window_reference_point_tab, width=4, height=1)
        Globals.profiles_input_vertical_displacement.insert(INSERT, " ")
        Globals.profiles_input_vertical_displacement.config(bg='#E5f9ff', \
        relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
        Globals.profiles_input_vertical_displacement.grid(row=5, \
            column=3, padx=(0,25), pady=(35,0))
        new_window_reference_point_tab.grid_rowconfigure(8, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(8, weight=0)

        input_long_text = tk.Text(new_window_reference_point_tab, width=12, height=1)
        input_long_text.insert(INSERT, "Longitudinal:")
        input_long_text.config(bg='#ffffff', relief=FLAT, bd=0, \
        state=DISABLED, font=('calibri', '10'))
        input_long_text.grid(row=5, column=3, sticky=N+S, padx=(250,0), pady=(25,0))
        new_window_reference_point_tab.grid_rowconfigure(12, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(12, weight=0)

        Globals.profiles_input_longitudinal_displacement = \
            tk.Text(new_window_reference_point_tab, width=5, height=1)
        Globals.profiles_input_longitudinal_displacement.insert(INSERT, " ")
```

```
        Globals.profiles_input_longitudinal_displacement.config(bg='#E5f9ff', \
        relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
        Globals.profiles_input_longitudinal_displacement.grid(row=5, \
            column=3, padx=(240,0), pady=(35,0))
        new_window_reference_point_tab.grid_rowconfigure(9, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(9, weight=0)

        film_window_mark_ROI_text = tk.Text(new_window_isocenter_tab, \
            width=55, height=7)
        film_window_mark_ROI_text.insert(INSERT, \
"When clicking the button \"Mark ROI\" a window showing the\n\
image will appear and you are to drag a rectangle marking \n\
the region of interest. Fidora will assume the film has been\n\
scanned in either portrait or landscape orientation. When\n\
the ROI has been marked it will appear on the image. If you\n\
are not happy with the placement click the button again.")
        film_window_mark_ROI_text.config(bg='#ffffff', relief=FLAT, bd=0, \
        state=DISABLED, font=('calibri', '11'))
        film_window_mark_ROI_text.grid(row=5, column=3, rowspan=4, \
            sticky=N+S+E+W, padx=(10,10), pady=(0,0))
        new_window_isocenter_tab.grid_columnconfigure(4, weight=0)
        new_window_isocenter_tab.grid_rowconfigure(4, weight=0)

        film_window_mark_ROI_reference_point_text = \
            tk.Text(new_window_reference_point_tab, width=55, height=5)
        film_window_mark_ROI_reference_point_text.insert(INSERT, \
"When clicking the button \"Mark ROI\" a window showing the\n\
image will appear and you are to drag a rectangle marking \n\
the region of interest. Fidora will assume the film has been\n\
scanned in either portrait or landscape orientation. When\n\
the ROI has been marked it will appear on the image. If you\n\
are not happy with the placement click the button again.")
        film_window_mark_ROI_reference_point_text.config(bg='#ffffff', relief=FLAT, \
        bd=0, state=DISABLED, font=('calibri', '11'))
        film_window_mark_ROI_reference_point_text.grid(row=6, column=3, rowspan=3, \
            sticky=N+E+W, padx=(10,10), pady=(10,0))
        new_window_reference_point_tab.grid_columnconfigure(4, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(4, weight=0)

        mark_ROI_button_frame = tk.Frame(new_window_isocenter_tab)
        mark_ROI_button_frame.grid(row=8, column=3, padx=(10,0), pady=(0,5))
        mark_ROI_button_frame.configure(bg='#ffffff')
        new_window_isocenter_tab.grid_columnconfigure(5, weight=0)
        new_window_isocenter_tab.grid_rowconfigure(5, weight=0)
```

```
mark_ROI_button = tk.Button(mark_ROI_button_frame, text='Browse', \
    image=Globals.profiles_mark_ROI_button_image,cursor='hand2',\
        font=('calibri', '14'), relief=FLAT, state=ACTIVE, \
            command=lambda: markROI(img, new_window_isocenter_tab, \
                image_canvas, False))
mark_ROI_button.pack(expand=True, fill=BOTH)
mark_ROI_button.config(bg='#ffffff', activebackground='#ffffff', \
    activeforeground='#ffffff', highlightthickness=0)
mark_ROI_button.image=Globals.profiles_mark_ROI_button_image

slice_offset_text = tk.Text(new_window_isocenter_tab, width=25, height=1)
slice_offset_text.insert(INSERT, "Slice offset, mm (default 0):")
slice_offset_text.config(state=DISABLED, font=('calibri', '10'), \
    bd = 0, relief=FLAT)
slice_offset_text.grid(row=9, column=3, padx=(5,110), pady=(0,0))
new_window_isocenter_tab.grid_columnconfigure(6, weight=0)
new_window_isocenter_tab.grid_rowconfigure(6, weight=0)

Globals.profiles_slice_offset = tk.Text(new_window_isocenter_tab, \
    width=8, height=1)
Globals.profiles_slice_offset.grid(row=9, column=3, \
    padx=(110,10), pady=(0,0))
Globals.profiles_slice_offset.insert(INSERT, " ")
Globals.profiles_slice_offset.config(state=NORMAL, \
    font=('calibri', '10'), bd = 2, bg='#ffffff')
new_window_isocenter_tab.grid_columnconfigure(7, weight=0)
new_window_isocenter_tab.grid_rowconfigure(7, weight=0)

mark_ROI_button_reference_point_frame = \
    tk.Frame(new_window_reference_point_tab)
mark_ROI_button_reference_point_frame.grid\
    (row=9, column=3, padx=(10,10), pady=(0,5))
mark_ROI_button_reference_point_frame.configure(bg='#ffffff')
new_window_reference_point_tab.grid_columnconfigure(5, weight=0)
new_window_reference_point_tab.grid_rowconfigure(5, weight=0)

mark_ROI_reference_point_button = \
    tk.Button(mark_ROI_button_reference_point_frame, text='Browse', \
        image=Globals.profiles_mark_ROI_button_image, cursor='hand2',\
            font=('calibri', '14'), relief=FLAT, state=ACTIVE, \
                command=lambda: markROI(img, \
                    new_window_reference_point_tab, \
                        image_canvas_reference_tab, True))
mark_ROI_reference_point_button.pack(expand=True, fill=BOTH)
mark_ROI_reference_point_button.config(bg='#ffffff', \
            activebackground='#ffffff', activeforeground='#ffffff', \
                highlightthickness=0)
        mark_ROI_reference_point_button.image=Globals.profiles_mark_ROI_button_image

        def finishFilmMarkers(ref_test):
            Globals.profiles_slice_offset.config(state=DISABLED)
            if(ref_test):
                if(not(Globals.profiles_input_lateral_displacement.\
                    get("1.0",'end-1c')==" ")):
                    try:
                        test = float(Globals.profiles_input_lateral_displacement.\
                            get("1.0",'end-1c'))
                        Globals.profiles_lateral = test
                    except:
                        messagebox.showerror("Error", "The displacements must \
be numbers\n (Code: lateral displacement)")
                        return
                else:
                    Globals.profiles_lateral = 0
                if(not(Globals.profiles_input_longitudinal_displacement.\
                    get("1.0",'end-1c')==" ")):
                    try:
                        test = \
                            float(Globals.profiles_input_longitudinal_displacement.\
                                get("1.0", 'end-1c'))
                        Globals.profiles_longitudinal = test
                    except:
                        messagebox.showerror("Error", "The displacements must \
be numbers\n (Code: longitudinal displacement)")
                        return
                else:
                    Globals.profiles_longitudinal = 0
                if(not(Globals.profiles_input_vertical_displacement.\
                    get("1.0",'end-1c')==" ")):
                    try:
                        test = float(Globals.profiles_input_vertical_displacement.\
                            get("1.0", 'end-1c'))
                        Globals.profiles_vertical = test
                    except:
                        messagebox.showerror("Error", "The displacements must \
be numbers\n (Code: vertical displacement)")
                        return
                else:
                    Globals.profiles_vertical = 0
                Globals.profiles_input_vertical_displacement.\
                    config(state=DISABLED)
                Globals.profiles_input_longitudinal_displacement.\
                    config(state=DISABLED)
                Globals.profiles_input_lateral_displacement.\
                    config(state=DISABLED)
            else:
                if not Globals.profiles_slice_offset.get("1.0",'end-1c')==" ":
                    try:
                        offset = float(Globals.profiles_slice_offset.\
                            get("1.0",'end-1c'))
                        Globals.profiles_offset = offset
                    except:
                        messagebox.showerror("Error", "Slice offset \
must be a number \n(Code: finishFilmMarkers(false)")
                        return
                else:
                    Globals.profiles_offset = 0
            if(ref_test):
                choose_batch_window = tk.Toplevel(new_window_reference_point_tab)
            else:
                choose_batch_window = tk.Toplevel(new_window_isocenter_tab)

            choose_batch_window.geometry("670x380+50+50")
            choose_batch_window.grab_set()

            choose_batch_frame = tk.Frame(choose_batch_window)
            choose_batch_frame.pack(expand=True, fill=BOTH)
            choose_batch_frame.configure(bg='#ffffff')

            batch_cnt = 0
            weight_cnt = 0
            read = open('calibration.txt', 'r')
            lines = read.readlines()
            read.close()
            row_cnt=0
            for l in lines:
                words = l.split()
                line = "Batch nr.  : " + words[2] + ".    Date:   "\
                     + words[0] + "  " + words[1] + "."
                write_batch_nr = tk.Text(choose_batch_frame, width=10, height=1)
                write_batch_nr.grid(row=row_cnt, column=0, sticky=N+S+W+E, \
                    padx=(10,5), pady=(10,10))
                choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
                choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
                write_batch_nr.insert(INSERT, "Batch nr.: ")
                write_batch_nr.config(state=DISABLED, bd = 0, \
                    font=('calibri', '12', 'bold'))
                weight_cnt+=1
                write_batch = tk.Text(choose_batch_frame, width=20, height=1)
                write_batch.grid(row=row_cnt, column=1, sticky=N+S+W+E,\
                     padx=(10,5), pady=(10,10))
                choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
                choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
                write_batch.insert(INSERT, words[2])
                write_batch.config(state=DISABLED, bd = 0, font=('calibri', '12'))
                weight_cnt+=1
                write_batch_date = tk.Text(choose_batch_frame, width=8, height=1)
                write_batch_date.grid(row=row_cnt, column=2, sticky=N+S+W+E, \
                    padx=(10,5), pady=(10,10))
                choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
                choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
                write_batch_date.insert(INSERT, "Date: ")
                write_batch_date.config(state=DISABLED, bd = 0, \
                    font=('calibri', '12', 'bold'))
                weight_cnt+=1
                write_date = tk.Text(choose_batch_frame, width=30, height=1)
                write_date.grid(row=row_cnt, column=3, sticky=N+S+W+E, \
                    padx=(10,5), pady=(10,10))
                choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
                choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
                write_date.insert(INSERT, words[0] + ", " + words[1] + "")
                write_date.config(state=DISABLED, bd = 0, font=('calibri', '12'))
                weight_cnt+=1

                Radiobutton(choose_batch_frame, text='',bg='#ffffff', \
                    cursor='hand2',font=('calibri', '14'), \
                        variable=Globals.profiles_film_batch, \
                            value=batch_cnt).grid(row=row_cnt, column=4, \
                                sticky=N+S+W+E, padx=(5,5), pady=(10,10))
                choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
                choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
                weight_cnt+=1;row_cnt+=1;batch_cnt+=1

            def set_batch():
                choose_batch_window.destroy()
                f = open('calibration.txt', 'r')
                lines = f.readlines()
                words = lines[Globals.profiles_film_batch.get()].split()
                Globals.profiles_popt_red[0] = float(words[3])
                Globals.profiles_popt_red[1] = float(words[4])
```

```
Globals.profiles_popt_red[2] = float(words[5])
f.close()

Globals.profiles_film_dataset_ROI_red_channel_dose = \
    np.zeros((Globals.profiles_film_dataset_ROI_red_channel\
        .shape[0],Globals.profiles_film_dataset_ROI_red_channel\
            .shape[1]))
for i in range\
    (Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0]):
    for j in range(Globals.\
        profiles_film_dataset_ROI_red_channel_dose.shape[1]):
        Globals.profiles_film_dataset_ROI_red_channel_dose[i,j] \
            = Globals.profiles_film_factor*pixel_to_dose\
                (Globals.profiles_film_dataset_ROI_red_channel[i,j],\
                    Globals.profiles_popt_red[0], \
                        Globals.profiles_popt_red[1], \
                            Globals.profiles_popt_red[2])

Globals.profiles_film_dataset_red_channel_dose = \
    np.zeros((Globals.profiles_film_dataset_red_channel.shape[0],\
    Globals.profiles_film_dataset_red_channel.shape[1]))
for i in range\
    (Globals.profiles_film_dataset_red_channel_dose.shape[0]):
    for j in range(Globals.\
        profiles_film_dataset_red_channel_dose.shape[1]):
        Globals.profiles_film_dataset_red_channel_dose[i,j] = \
            Globals.profiles_film_factor*pixel_to_dose(\
                Globals.profiles_film_dataset_red_channel[i,j], \
                    Globals.profiles_popt_red[0], \
                        Globals.profiles_popt_red[1], \
                            Globals.profiles_popt_red[2])

Globals.film_write_image.create_image(0,0,\
    image=scaled_image_visual, anchor="nw")
Globals.film_write_image.image = scaled_image_visual

mx_film=np.max(Globals.profiles_film_dataset_ROI_red_channel_dose)
Globals.profiles_max_dose_film = mx_film

film_scanned_image_text_canvas.create_image(0,0,\
    image=Globals.profiles_scanned_image_text_image, anchor="nw")
film_scanned_image_text_canvas.image = \
    Globals.profiles_scanned_image_text_image

new_window.destroy()
```

```
set_batch_button_frame = tk.Frame(choose_batch_frame)
set_batch_button_frame.grid(row=row_cnt, column=1, \
    columnspan=3, padx=(10,0), pady=(5,5))
set_batch_button_frame.configure(bg='#ffffff')
choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)

set_batch_button = tk.Button(set_batch_button_frame, text='OK', \
    image=Globals.done_button_image, cursor='hand2',\
    font=('calibri', '14'), relief=FLAT, state=ACTIVE,command=set_batch)
set_batch_button.pack(expand=True, fill=BOTH)
set_batch_button.image=Globals.done_button_image


img_ROI = Globals.profiles_film_dataset\
    [Globals.profiles_ROI_coords[0][1]:\
        Globals.profiles_ROI_coords[2][1],\
            Globals.profiles_ROI_coords[0][0]:\
                Globals.profiles_ROI_coords[1][0], :]
img_ROI_red_channel = img_ROI[:,:,2]
Globals.profiles_film_variable_ROI_coords = \
    [Globals.profiles_ROI_coords[0][1], \
        Globals.profiles_ROI_coords[2][1],\
    Globals.profiles_ROI_coords[0][0], \
        Globals.profiles_ROI_coords[1][0]]
Globals.profiles_film_dataset_ROI = img_ROI
Globals.profiles_film_dataset_ROI_red_channel = img_ROI_red_channel
R = img_ROI[:,:,2];B = img_ROI[:,:,0]; G = img_ROI[:,:,1]
img_ROI_RGB = np.zeros(img_ROI.shape)
img_ROI_RGB[:,:,0]=R; img_ROI_RGB[:,:,1]=G; img_ROI_RGB[:,:,2]=B
PIL_img_ROI = (img_ROI_RGB/256).astype('uint8')
PIL_img_ROI = Image.fromarray(PIL_img_ROI, 'RGB')
wid = PIL_img_ROI.width;heig = PIL_img_ROI.height

film_image_canvas = tk.Canvas(Globals.profiles_film_panedwindow)
film_image_canvas.grid(row=0,column=0, sticky=N+S+W+E)
Globals.profiles_film_panedwindow.add(film_image_canvas, \
    height=max(heig,Globals.profiles_scanned_image_text_image.height()),\
        width=wid + Globals.profiles_scanned_image_text_image.width())
film_image_canvas.config(bg='#ffffff', relief=FLAT, highlightthickness=0,\
    height=max(heig,Globals.profiles_scanned_image_text_image.height()),\
        width=wid + Globals.profiles_scanned_image_text_image.width())

Globals.film_write_image = tk.Canvas(film_image_canvas)
Globals.film_write_image.grid(row=0,column=1,sticky=N+S+W+E)
Globals.film_write_image.config(bg='#ffffff', relief=FLAT, \
    highlightthickness=0, width=wid, height=heig)

film_scanned_image_text_canvas=tk.Canvas(film_image_canvas)
film_scanned_image_text_canvas.grid(row=0,column=0,sticky=N+S+W+E)
film_scanned_image_text_canvas.config(bg='#ffffff', relief=FLAT, \
    highlightthickness=0, height=\
        Globals.profiles_scanned_image_text_image.height(), \
            width=Globals.profiles_scanned_image_text_image.width())

scaled_image_visual = PIL_img_ROI
scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual)

Globals.profiles_upload_button_doseplan.config(state=DISABLED)
Globals.profiles_upload_button_rtplan.config(state=ACTIVE)
Globals.profiles_upload_button_film.config(state=DISABLED)

if(ref_test):
    Globals.profiles_distance_reference_point_ROI.append\
        ([(Globals.profiles_film_reference_point[0]-\
            Globals.profiles_ROI_coords[0][0])*0.2, \
                (Globals.profiles_film_reference_point[1] -\
                    Globals.profiles_ROI_coords[0][1])*0.2])
    Globals.profiles_distance_reference_point_ROI.append\
        ([(Globals.profiles_film_reference_point[0] - \
            Globals.profiles_ROI_coords[1][0])*0.2,\
                (Globals.profiles_film_reference_point[1] - \
                    Globals.profiles_ROI_coords[1][1])*0.2])
    Globals.profiles_distance_reference_point_ROI.append\
        ([(Globals.profiles_film_reference_point[0] - \
            Globals.profiles_ROI_coords[2][0])*0.2,\
                (Globals.profiles_film_reference_point[1] - \
                    Globals.profiles_ROI_coords[2][1])*0.2])
    Globals.profiles_distance_reference_point_ROI.append\
        ([(Globals.profiles_film_reference_point[0] - \
            Globals.profiles_ROI_coords[3][0])*0.2,\
                (Globals.profiles_film_reference_point[1] - \
                    Globals.profiles_ROI_coords[3][1])*0.2])

    Globals.profiles_isocenter_or_reference_point = "Ref_point"
else:
    Globals.profiles_distance_isocenter_ROI.append\
        ([(Globals.profiles_film_isocenter[0]-\
            Globals.profiles_ROI_coords[0][0])*0.2, \
                    (Globals.profiles_film_isocenter[1] -\
                        Globals.profiles_ROI_coords[0][1])*0.2])
            Globals.profiles_distance_isocenter_ROI.append\
                ([(Globals.profiles_film_isocenter[0] - \
                    Globals.profiles_ROI_coords[1][0])*0.2,\
                        (Globals.profiles_film_isocenter[1] - \
                            Globals.profiles_ROI_coords[1][1])*0.2])
            Globals.profiles_distance_isocenter_ROI.append\
                ([(Globals.profiles_film_isocenter[0] - \
                    Globals.profiles_ROI_coords[2][0])*0.2,\
                        (Globals.profiles_film_isocenter[1] - \
                            Globals.profiles_ROI_coords[2][1])*0.2])
            Globals.profiles_distance_isocenter_ROI.append\
                ([(Globals.profiles_film_isocenter[0] - \
                    Globals.profiles_ROI_coords[3][0])*0.2,\
                        (Globals.profiles_film_isocenter[1] - \
                            Globals.profiles_ROI_coords[3][1])*0.2])

            Globals.profiles_isocenter_or_reference_point = "Isocenter"



    done_button_frame = tk.Frame(new_window_isocenter_tab)
    done_button_frame.grid(row=10, column=3, padx=(10,10), \
        pady=(5,5), sticky=N+S+W+E)
    done_button_frame.configure(bg='#ffffff')
    new_window_isocenter_tab.grid_columnconfigure(5, weight=0)
    new_window_isocenter_tab.grid_rowconfigure(5, weight=0)

    Globals.profiles_done_button = tk.Button(done_button_frame, \
        text='Done', image=Globals.done_button_image,\
        cursor='hand2', font=('calibri', '14'), relief=FLAT, \
            state=DISABLED, command=lambda: finishFilmMarkers(False))
    Globals.profiles_done_button.pack(expand=True, fill=BOTH)
    Globals.profiles_done_button.config(bg='#ffffff',activebackground='#ffffff',\
        activeforeground='#ffffff', highlightthickness=0)
    Globals.profiles_done_button.image=Globals.done_button_image

    done_button_reference_point_frame = tk.Frame(new_window_reference_point_tab)
    done_button_reference_point_frame.grid(row=10, column=3, \
        padx=(10,10), pady=(5,5), sticky=N+S+W+E)
    done_button_reference_point_frame.configure(bg='#ffffff')
    new_window_reference_point_tab.grid_columnconfigure(5, weight=0)
    new_window_reference_point_tab.grid_rowconfigure(5, weight=0)

    Globals.profiles_done_button_reference_point= \
            tk.Button(done_button_reference_point_frame, text='Done', \
                image=Globals.done_button_image,cursor='hand2', \
                    font=('calibri', '14'), relief=FLAT, state=DISABLED, \
                        command=lambda: finishFilmMarkers(True))
        Globals.profiles_done_button_reference_point.pack(expand=True, fill=BOTH)
        Globals.profiles_done_button_reference_point.config(bg='#ffffff', \
            activebackground='#ffffff', activeforeground='#ffffff', \
                highlightthickness=0)
        Globals.profiles_done_button_reference_point.image=Globals.done_button_image


    elif(ext==""):
        return
    else:
        messagebox.showerror("Error", "The file must be a *.tif file")


def help_showPlanes():
#------------------------------------------------------------
# Function to show the help-image in film orientation
#
# This functions is a callback to the button
# profiles_help_button_orientation in notebook.py
#------------------------------------------------------------
    new_window = tk.Toplevel(Globals.tab4)
    w = Globals.profiles_showPlanes_image.width()
    h = Globals.profiles_showPlanes_image.height()
    new_window.geometry("%dx%d+0+0" % (w, h))
    new_window.grab_set()

    canvas = tk.Canvas(new_window)
    canvas.config(relief=FLAT, bg='#ffffff', highlightthickness=0)
    canvas.create_image(0, 0, image=Globals.profiles_showPlanes_image, anchor='nw')
    canvas.pack(expand=True, fill=BOTH)
```

DVH_functions.py

```
#------------------------------------------------------------------------------------------
# DVH_functions.py
# version 18.08.20
#
# To be used related to the tab DVH in Fidora.
#
#------------------------------------------------------------------------------------------
```

```
import Globals
import tkinter as tk
from tkinter import filedialog, INSERT, DISABLED, messagebox, NORMAL,simpledialog,\
    PhotoImage, BOTH, Canvas, N, S, W, E, ALL, Frame, SUNKEN, Radiobutton, GROOVE, \
        ACTIVE, FLAT, END, Scrollbar, HORIZONTAL, VERTICAL, ttk, TOP, RIGHT, LEFT, \
            ttk, Checkbutton, IntVar
import os
from os.path import normpath, basename
from PIL import Image, ImageTk
import cv2
from cv2 import imread, IMREAD_ANYCOLOR, IMREAD_ANYDEPTH, imwrite
import pydicom
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import \
    FigureCanvasTkAgg,  NavigationToolbar2Tk
import numpy as np
import random

def nothingButton():
#--------------------------------------------------
# Function to only return
# Is used in cases where nothing should happen in
# a active button (they will later be reconfigured)
#--------------------------------------------------
    return

def clearAll():
#--------------------------------------------------
# Function to reset all variables
#
# This function is called when the user
# wants to restart the program
#--------------------------------------------------
    for widget in Globals.DVH_view_film_doseplan_ROI.winfo_children():
        widget.destroy()
    Globals.temp_image_canvas = tk.Canvas(Globals.DVH_view_film_doseplan_ROI)
    Globals.temp_image_canvas.grid(row=0, column=0, sticky=N+S+W+E)
    Globals.temp_image_canvas.config(bg='#ffffff',bd=0, \
        highlightthickness=0,relief=FLAT)
    Globals.temp_image_canvas.create_image(270,150, image=Globals.dVH_temp_image)
    Globals.DVH_export_button.config(state=DISABLED, command=nothingButton)

    Globals.DVH_film_orientation.set('—')
    Globals.DVH_number_of_doseplans = 0
    Globals.DVH_number_of_doseplans_row_count = 4
    Globals.DVH_doseplans_grid_config_count = 6
    Globals.DVH_doseplans_filenames = []
    Globals.DVH_doseplans_factor_text = []
    Globals.DVH_doseplans_factor_input = []
    Globals.DVH_doseplan_dataset_ROI_several = []
    Globals.DVH_several_img = []

    Globals.profiles_film_factor = None
    Globals.DVH_film_orientation_menu = None
    Globals.DVH_film_factor_input = None
    Globals.DVH_film_factor = None
    Globals.DVH_film_dataset = None
    Globals.DVH_film_dataset_red_channel = None
    Globals.DVH_film_dataset_ROI = None
    Globals.DVH_film_dataset_ROI_red_channel = None
    Globals.DVH_doseplan_dataset_ROI = None
    Globals.DVH_film_dataset_ROI_red_channel_dose = None
    Globals.DVH_film_write_image = None
    Globals.DVH_film_dose_write_image = None
    Globals.DVH_max_dose_film = None
    Globals.DVH_max_dose_doseplan = None

    Globals.DVH_iscoenter_coords = []
    Globals.DVH_film_isocenter = None
    Globals.DVH_film_reference_point = None
    Globals.DVH_distance_isocenter_ROI = []
    Globals.DVH_distance_reference_point_ROI = []
    Globals.DVH_mark_isocenter_up_down_line = []
    Globals.DVH_mark_isocenter_right_left_line = []
    Globals.DVH_mark_isocenter_oval = []
    Globals.DVH_mark_ROI_rectangle = []
    Globals.DVH_mark_reference_point_oval = []
    Globals.DVH_ROI_coords = []
    Globals.DVH_film_variable_ROI_coords = None

    Globals.DVH_done_button = None
    Globals.DVH_done_button_reference_point = None
    Globals.DVH_isocenter_check=False
    Globals.DVH_reference_point_check = False
```

```
    Globals.DVH_ROI_check = False
    Globals.DVH_ROI_reference_point_check = False
    Globals.DVH_film_batch.set(0)
    Globals.DVH_popt_red = np.zeros(3)

    Globals.DVH_upload_button_doseplan.config(state=DISABLED)
    Globals.DVH_upload_button_film.config(state=ACTIVE)
    Globals.DVH_upload_button_rtplan.config(state=DISABLED)
    Globals.DVH_upload_button_struct.config(state=DISABLED)

    Globals.DVH_dataset_doseplan=None
    Globals.DVH_dataset_rtplan = None
    Globals.DVH_dataset_structure_file = None

    Globals.DVH_test_if_added_doseplan = False
    Globals.DVH_test_if_added_rtplan = False
    Globals.DVH_test_if_added_struct = False

    Globals.DVH_isocenter_mm=None
    Globals.DVH_dose_scaling_doseplan = None
    Globals.DVH_contour_names = None
    Globals.DVH_ROIContourSequence = None
    Globals.DVH_contours = []

    Globals.DVH_doseplan_write_image = None
    Globals.DVH_doseplan_write_image_width = None
    Globals.DVH_doseplan_write_image_height = None
    Globals.DVH_doseplan_lateral_displacement = None
    Globals.DVH_doseplan_vertical_displacement = None
    Globals.DVH_doseplan_longitudianl_displacement = None
    Globals.DVH_doseplan_patient_position = None

    Globals.DVH_reference_point_in_doseplan = None
    Globals.DVH_input_lateral_displacement = None
    Globals.DVH_input_longitudinal_displacement = None
    Globals.DVH_input_vertical_displacement = None
    Globals.DVH_slice_offset = None
    Globals.DVH_offset = None
    Globals.DVH_isocenter_or_reference_point = None

    Globals.DVH_lateral = None
    Globals.DVH_vertical = None
    Globals.DVH_longitudinal = None

    for widget in Globals.DVH_plot_canvas.winfo_children():
        widget.destroy()
    DVH_initial_fig = Figure(figsize=(10,6))
    DVH_a = DVH_initial_fig.add_subplot(111, ylim=(0,1), xlim=(0,50))
    DVH_initial_plot_canvas = FigureCanvasTkAgg\
        (DVH_initial_fig, master=Globals.DVH_plot_canvas)
    DVH_initial_plot_canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, \
        sticky=N+E+W+S, padx=(5,0), pady=(0,0))
    DVH_a.set_title ("Dose volume histogram", fontsize=12)
    DVH_a.set_ylabel("Volume (%)", fontsize=12)
    DVH_a.set_xlabel("Dose (Gy)", fontsize=12)
    DVH_initial_fig.tight_layout()

    for widget in Globals.DVH_list_contours_canvas.winfo_children():
        widget.destroy()
    return

def calculateDVH(data, data_film, initial, include_contours):
#--------------------------------------------------
# Function to calculate the dose volume histogram
#
# The function is called in processDoseplan_Isocenter
# and processDoseplan_Reference_Point, as well as
# it is a callback function for the button
# check indise this function
#--------------------------------------------------
    names = [item[0] for item in data]
    dose_array = [item[1] for item in data]
    prc_array = [item[2] for item in data]
    dose_array_film = [item[1] for item in data_film]
    prc_array_film = [item[2] for item in data_film]
    if initial:
        include_contours = []
        checkbuttons = []
        for name in names:
            var = IntVar()
            var.set(1)
            check = Checkbutton(Globals.DVH_list_contours_canvas, \
                text=name, variable=var, command=lambda: \
                    calculateDVH(data, data_film, False, include_contours))
            check.pack(side=LEFT)
            check.config(bg='#ffffff')
            include_contours.append(var)
            checkbuttons.append(check)

    DVH_fig = Figure(figsize=(10,6))
    DVH_a = DVH_fig.add_subplot(111, ylim=(-0.1,1.1), xlim=(0,50))
    DVH_plot_canvas = FigureCanvasTkAgg(DVH_fig, master=Globals.DVH_plot_canvas)
    DVH_plot_canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, \
        sticky=N+E+W+S, padx=(5,0), pady=(0,0))
    legend_names = []
    for i in range(len(include_contours)):
        if include_contours[i].get():
            color = (random.random(), random.random(), random.random())
            plot_dose_array = list(dose_array[i])
            plot_dose_array.insert(0,0)
            plot_dose_array.insert(len(plot_dose_array),100)
            plot_prc_array = list(prc_array[i])
            plot_prc_array.insert(0,1)
            plot_prc_array.insert(len(plot_prc_array),0)
            DVH_a.plot(np.array(plot_dose_array),np.array(plot_prc_array), c=color)

            plot_dose_array_film = list(dose_array_film[i])
            plot_dose_array_film.insert(0,0)
            plot_dose_array_film.insert(len(plot_dose_array_film),100)
            plot_prc_array_film = list(prc_array_film[i])
            plot_prc_array_film.insert(0,1)
            plot_prc_array_film.insert(len(plot_prc_array_film), 0)
            DVH_a.plot(np.array(plot_dose_array_film), \
                np.array(plot_prc_array_film), c=color, ls='--')

            legend_names.append(names[i])
            legend_names.append(names[i] + " (Film)")

    def export_plot():
        messagebox.showinfo("Info", "This button has not been written")
        return

    Globals.DVH_export_button.config(state=ACTIVE, command=export_plot)

    DVH_a.legend(legend_names,loc='center left', bbox_to_anchor=(1, 0.5))
    DVH_a.set_title ("Dose volume histogram", fontsize=12)
    DVH_a.set_ylabel("Volume (%)", fontsize=12)
    DVH_a.set_xlabel("Dose (Gy)", fontsize=12)
    DVH_fig.tight_layout()

def processDoseplan_usingReferencePoint(only_one):
#-----------------------------------------------------------
# Function to process the doseplan when reference
# point is choosen to define the position.
# Is called in UploadDoseplan
```

```
#
# Parameter: only_one -to test if only one doseplan
# is uploaded in Fidora.
#-----------------------------------------------------

    ################# RT Plan ######################
    iso_1 = abs(Globals.DVH_dataset_doseplan.ImagePositionPatient[0] - \
        Globals.DVH_isocenter_mm[0])
    iso_2 = abs(Globals.DVH_dataset_doseplan.ImagePositionPatient[1] - \
        Globals.DVH_isocenter_mm[1])
    iso_3 = abs(Globals.DVH_dataset_doseplan.ImagePositionPatient[2] - \
        Globals.DVH_isocenter_mm[2])
    Globals.DVH_isocenter_mm = [iso_1, iso_2, iso_3]

    try:
        Globals.DVH_vertical = int(Globals.DVH_vertical)
    except:
        messagebox.showerror("Error", "Could not read the vertical \
displacements\n (Code: displacements to integer)")
        return
    try:
        Globals.DVH_lateral = int(Globals.DVH_lateral)
    except:
        messagebox.showerror("Error", "Could not read the lateral \
displacements\n (Code: displacements to integer)")
        return
    try:
        Globals.DVH_longitudinal = int(Globals.DVH_longitudinal)
    except:
        messagebox.showerror("Error", "Could not read the longitudinal \
displacements\n (Code: displacements to integer)")
        return

    lateral = Globals.DVH_lateral
    longit = Globals.DVHlongitudinal
    vertical = Globals.DVH_vertical
    isocenter_px = np.zeros(3)
    distance_in_doseplan_ROI_reference_point_px = []
    if(Globals.DVH_dataset_doseplan.PixelSpacing==[1, 1]):
        isocenter_px[0] = np.round(iso_1)
        isocenter_px[1] = np.round(iso_2)
        isocenter_px[2] = np.round(iso_3)

        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.DVH_distance_reference_point_ROI[0][0]),\
            np.round(Globals.DVH_distance_reference_point_ROI[0][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.DVH_distance_reference_point_ROI[1][0]),\
                np.round(Globals.DVH_distance_reference_point_ROI[1][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.DVH_distance_reference_point_ROI[2][0]),\
                np.round(Globals.DVH_distance_reference_point_ROI[2][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.DVH_distance_reference_point_ROI[3][0]),\
                np.round(Globals.DVH_distance_reference_point_ROI[3][1])])

        lateral_px = np.round(lateral)
        vertical_px = np.round(vertical)
        longit_px = np.round(longit)

        doseplan_lateral_displacement_px = \
            np.round(Globals.DVH_doseplan_lateral_displacement)
        doseplan_vertical_displacement_px = \
            np.round(Globals.DVH_doseplan_vertical_displacement)
        doseplan_longitudinal_displacement_px = \
            np.round(Globals.DVH_doseplan_longitudianl_displacement)

    elif(Globals.DVH_dataset_doseplan.PixelSpacing==[2, 2]):
        isocenter_px[0] = np.round(iso_1/2)
        isocenter_px[1] = np.round(iso_2/2)
        isocenter_px[2] = np.round(iso_3/2)

        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_reference_point_ROI[0][0])/2),\
                np.round((Globals.DVH_distance_reference_point_ROI[0][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_reference_point_ROI[1][0])/2),\
                np.round((Globals.DVH_distance_reference_point_ROI[1][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_reference_point_ROI[2][0])/2),\
                np.round((Globals.DVH_distance_reference_point_ROI[2][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_reference_point_ROI[3][0])/2),\
                np.round((Globals.DVH_distance_reference_point_ROI[3][1])/2)])

        lateral_px = np.round(lateral/2)
        vertical_px = np.round(vertical/2)
        longit_px = np.round(longit/2)

        doseplan_lateral_displacement_px = \
            np.round((Globals.DVH_doseplan_lateral_displacement)/2)
        doseplan_vertical_displacement_px = \
            np.round((Globals.DVH_doseplan_vertical_displacement)/2)
        doseplan_longitudinal_displacement_px = \
            np.round((Globals.DVH_doseplan_longitudianl_displacement)/2)

    else:
        isocenter_px[0] = np.round(iso_1/3)
        isocenter_px[1] = np.round(iso_2/3)
        isocenter_px[2] = np.round(iso_3/3)

        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_reference_point_ROI[0][0])/3),\
                np.round((Globals.DVH_distance_reference_point_ROI[0][1])/3)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_reference_point_ROI[1][0])/3),\
                np.round((Globals.DVH_distance_reference_point_ROI[1][1])/3)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_reference_point_ROI[2][0])/3),\
                np.round((Globals.DVH_distance_reference_point_ROI[2][1])/3)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_reference_point_ROI[3][0])/3),\
                np.round((Globals.DVH_distance_reference_point_ROI[3][1])/3)])

        lateral_px = np.round(lateral/3)
        vertical_px = np.round(vertical/3)
        longit_px = np.round(longit/3)

        doseplan_lateral_displacement_px = \
            np.round((Globals.DVH_doseplan_lateral_displacement)/3)
        doseplan_vertical_displacement_px = \
            np.round((Globals.DVH_doseplan_vertical_displacement)/3)
        doseplan_longitudinal_displacement_px = \
            np.round((Globals.DVH_doseplan_longitudianl_displacement)/3)

    temp_ref_point_doseplan = np.zeros(3)

    if(Globals.DVH_doseplan_patient_position=='HFS'):
        temp_ref_point_doseplan[0] = int(isocenter_px[0]+ \
            doseplan_lateral_displacement_px - lateral_px)
        temp_ref_point_doseplan[1] = int(isocenter_px[1]- \
            doseplan_vertical_displacement_px + vertical_px)
        temp_ref_point_doseplan[2] = int(isocenter_px[2]+ \
            doseplan_longitudinal_displacement_px - longit_px)
    elif(Globals.DVH_doseplan_patient_position=='HFP'):
        temp_ref_point_doseplan[0] = isocenter_px[0]− \
            doseplan_lateral_displacement_px+ lateral_px
        temp_ref_point_doseplan[1] = isocenter_px[1]+ \
            doseplan_vertical_displacement_px − vertical_px
        temp_ref_point_doseplan[2] = isocenter_px[2]+ \
            doseplan_longitudinal_displacement_px − longit_px
    elif(Globals.DVH_doseplan_patient_position=='HFDR'):
        temp_ref_point_doseplan[0] = isocenter_px[0]− \
            doseplan_vertical_displacement_px + vertical_px
        temp_ref_point_doseplan[1] = isocenter_px[1]+ \
            doseplan_lateral_displacement_px − lateral_px
        temp_ref_point_doseplan[2] = isocenter_px[2]+ \
            doseplan_longitudinal_displacement_px − longit_px
    elif(Globals.DVH_doseplan_patient_position=='HFDL'):
        temp_ref_point_doseplan[0] = isocenter_px[0]+ \
            doseplan_vertical_displacement_px − vertical_px
        temp_ref_point_doseplan[1] = isocenter_px[1]− \
            doseplan_lateral_displacement_px + lateral_px
        temp_ref_point_doseplan[2] = isocenter_px[2]+ \
            doseplan_longitudinal_displacement_px − longit_px
    elif(Globals.DVH_doseplan_patient_position=='FFS'):
        temp_ref_point_doseplan[0] = isocenter_px[0]− \
            doseplan_lateral_displacement_px + lateral_px
        temp_ref_point_doseplan[1] = isocenter_px[1]+ \
            doseplan_vertical_displacement_px − vertical_px
        temp_ref_point_doseplan[2] = isocenter_px[2]− \
            doseplan_longitudinal_displacement_px + longit_px
    elif(Globals.DVH_doseplan_patient_position=='FFP'):
        temp_ref_point_doseplan[0] = isocenter_px[0]+ \
            doseplan_lateral_displacement_px− lateral_px
        temp_ref_point_doseplan[1] = isocenter_px[1]− \
            doseplan_vertical_displacement_px + vertical_px
        temp_ref_point_doseplan[2] = isocenter_px[2]− \
            doseplan_longitudinal_displacement_px + longit_px
    elif(Globals.DVH_doseplan_patient_position=='FFDR'):
        temp_ref_point_doseplan[0] = isocenter_px[0]− \
            doseplan_vertical_displacement_px + vertical_px
        temp_ref_point_doseplan[1] = isocenter_px[1]− \
            doseplan_lateral_displacement_px + lateral_px
        temp_ref_point_doseplan[2] = isocenter_px[2]− \
            doseplan_longitudinal_displacement_px + longit_px
    else:
        temp_ref_point_doseplan[0] = isocenter_px[0] + \
            doseplan_vertical_displacement_px − vertical_px
        temp_ref_point_doseplan[1] = isocenter_px[1] + \
            doseplan_lateral_displacement_px — lateral_px
        temp_ref_point_doseplan[2] = isocenter_px[2]— \
            doseplan_longitudinal_displacement_px + longit_px

    Globals.DVH_reference_point_in_doseplan = temp_ref_point_doseplan
    reference_point = np.zeros(3)

    ######################## Struct file ###################################
    i=0
    for name in Globals.DVH_contour_names:
        sequences = Globals.DVH_ROIContourSequence[i].ContourSequence
        temp_x = [];temp_y = [];temp_z = []
        for sequence in sequences:
            if(Globals.DVH_dataset_doseplan.PixelSpacing==[1, 1]):
                temp_x.append([int(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[0]—x)) for x in sequence.ContourData[0::3]])
                temp_y.append([int(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[1]—y)) for y in sequence.ContourData[1::3]])
                temp_z.append([int(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[2]—z)) for z in sequence.ContourData[2::3]])
            elif(Globals.DVH_dataset_doseplan.PixelSpacing==[2, 2]):
                temp_x.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[0]—x)/2)) for \
                        x in sequence.ContourData[0::3]])
                temp_y.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[1]—y)/2)) for \
                        y in sequence.ContourData[1::3]])
                temp_z.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[2]—z)/2)) for \
                        z in sequence.ContourData[2::3]])
            else:
                temp_x.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[0]—x)/3)) for \
                        x in sequence.ContourData[0::3]])
                temp_y.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[1]—y)/3)) for \
                        y in sequence.ContourData[1::3]])
                temp_z.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                    ImagePositionPatient[2]—z)/3)) for \
                        z in sequence.ContourData[2::3]])
        Globals.DVH_contours.append\
            ([i, name.ROIObservationLabel, temp_x, temp_y, temp_z])
        i+=1

    ######################## Doseplan ###################################
```

```
if(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[1, 0, 0, 0, 1, 0]):
    reference_point[0] = temp_ref_point_doseplan[2]
    reference_point[1] = temp_ref_point_doseplan[1]
    reference_point[2] = temp_ref_point_doseplan[0]
    for i in range(len(Globals.DVH_contours)):
        temp_x = Globals.DVH_contours[i][2]
        temp_y = Globals.DVH_contours[i][3]
        temp_z = Globals.DVH_contours[i][4]
        Globals.DVH_contours[i][2] = temp_z
        Globals.DVH_contours[i][3] = temp_y
        Globals.DVH_contours[i][4] = temp_x

    if(Globals.DVH_film_orientation.get()=='Coronal'):
        dataset_swapped = \
            np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
        temp_ref = reference_point[0]
        reference_point[0] = reference_point[1]
        reference_point[1] = temp_ref
        for i in range(len(Globals.DVH_contours)):
            temp_c = Globals.DVH_contours[i][2]
            Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
            Globals.DVH_contours[i][3] = temp_c
    elif(Globals.DVH_film_orientation.get()=='Sagittal'):
        dataset_swapped = \
            np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
        temp_ref = reference_point[0]
        reference_point[0] = reference_point[2]
        reference_point[2] = temp_ref
        for i in range(len(Globals.DVH_contours)):
            temp_c = Globals.DVH_contours[i][2]
            Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][4] = temp_c
    elif(Globals.DVH_film_orientation.get()=='Axial'):
        dataset_swapped = Globals.DVH_dataset_doseplan.pixel_array
    else:
        messagebox.showerror("Error", "Something has gone wrong here.")
        clearAll()
        return
elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[1, 0, 0, 0, 0, 1]):
    reference_point[0] = temp_ref_point_doseplan[1]
    reference_point[1] = temp_ref_point_doseplan[2]
    reference_point[2] = temp_ref_point_doseplan[0]
    for i in range(len(Globals.DVH_contours)):
        temp_x = Globals.DVH_contours[i][2]
        temp_y = Globals.DVH_contours[i][3]
```

```
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_y
            Globals.DVH_contours[i][3] = temp_z
            Globals.DVH_contours[i][4] = temp_x

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = Globals.DVH_dataset_doseplan.pixel_array
        elif(Globals.DCH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[0, 1, 0, 1, 0, 0]):
        reference_point[0] = temp_ref_point_doseplan[2]
        reference_point[1] = temp_ref_point_doseplan[0]
        reference_point[2] = temp_ref_point_doseplan[1]
        for i in range(len(Globals.DVH_contours)):
            temp_x = Globals.DVH_contours[i][2]
            temp_y = Globals.DVH_contours[i][3]
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_z
            Globals.DVH_contours[i][3] = temp_x
            Globals.DVH_contours[i][4] = temp_y

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[0, 1, 0, 0, 0, 1]):
        reference_point[0] = temp_ref_point_doseplan[0]
        reference_point[1] = temp_ref_point_doseplan[2]
        reference_point[2] = temp_ref_point_doseplan[1]
        for i in range(len(Globals.DVH_contours)):
            temp_x = Globals.DVH_contours[i][2]
            temp_y = Globals.DVH_contours[i][3]
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_x
            Globals.DVH_contours[i][3] = temp_z
            Globals.DVH_contours[i][4] = temp_y

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[0, 0, 1, 1, 0, 0]):
        reference_point[0] = temp_ref_point_doseplan[1]
        reference_point[1] = temp_ref_point_doseplan[0]
        reference_point[2] = temp_ref_point_doseplan[2]
        for i in range(len(Globals.DVH_contours)):
            temp_x = Globals.DVH_contours[i][2]
            temp_y = Globals.DVH_contours[i][3]
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_y
            Globals.DVH_contours[i][3] = temp_x
            Globals.DVH_contours[i][4] = temp_z

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[0, 0, 1, 0, 1, 0]):
        reference_point[0] = temp_ref_point_doseplan[0]
        reference_point[1] = temp_ref_point_doseplan[1]
        reference_point[2] = temp_ref_point_doseplan[2]

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = Globals.DVH_dataset_doseplan.pixel_array
        elif(Globals.DCH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    else:
        messagebox.showerror("Error", "Something has gone wrong.")
        clearAll()
        return

    if(reference_point[0]<0 or reference_point[0]>dataset_swapped.shape[0]):
        messagebox.showerror("Error", "Reference point is outside of dosematrix\n\
            (Code: first dimension, number of frames in dosematrix)")
        return
    if(reference_point[1]<0 or reference_point[1]>dataset_swapped.shape[1]):
        messagebox.showerror("Error", "Reference point is outside of dosematrix\n\
            (Code: second dimension, rows in dosematrix)")
        return
    if(reference_point[2]<0 or reference_point[2]>dataset_swapped.shape[2]):
        messagebox.showerror("Error", "Reference point is outside of dosematrix\n\
            (Code: third dimension, columns in dosematrix)")
        return

    dose_slice = dataset_swapped[int(reference_point[0]),:,:]

    structures_in_plane = []
    for contour in Globals.DVH_contours:
        in_plane = False
        temp_row=[];temp_column=[]
        for i in range(len(contour[2])):
            lst = contour[2][i]
            for j in range(len(lst)):
                plane = lst[j]
                if(plane == int(reference_point[0])):
                    temp_row.append(contour[3][i][j])
                    temp_column.append(contour[4][i][j])
                    in_plane = True
        if in_plane:
            structures_in_plane.append([contour[1], temp_row, temp_column])


    doseplan_ROI_coords = []
    top_left_test_side = False; top_left_test_down = False
    top_right_test_side = False; top_right_test_down = False
    bottom_left_test_side = False; bottom_left_test_down = False
    bottom_right_test_side = False; bottom_right_test_down = False
    top_left_side_corr = 0; top_left_down_corr = 0
    top_right_side_corr = 0; top_right_down_corr = 0
    bottom_left_side_corr = 0; bottom_left_down_corr = 0
    bottom_right_side_corr = 0; bottom_right_down_corr = 0


    top_left_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[0][0]
    top_left_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[0][1]
    if(top_left_to_side < 0):
        top_left_test_side = True
        top_left_side_corr = abs(top_left_to_side)
        top_left_to_side = 0
    if(top_left_to_side > dose_slice.shape[1]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(top_left_down < 0):
        top_left_test_down = True
        top_left_down_corr = abs(top_left_down)
        top_left_down = 0
    if(top_left_down > dose_slice.shape[0]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
```

```
        clearAll()
        return

    top_right_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[1][0]
    top_right_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[1][1]
    if(top_right_to_side < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(top_right_to_side > dose_slice.shape[1]):
        top_right_test_side = True
        top_right_side_corr = top_right_to_side - dose_slice.shape[1]
        top_right_to_side = dose_slice.shape[1]
    if(top_right_down < 0):
        top_right_test_down = True
        top_right_down_corr = abs(top_right_down)
        top_right_down = 0
    if(top_right_down > dose_slice.shape[0]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return

    bottom_left_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[2][0]
    bottom_left_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[2][1]
    if(bottom_left_to_side < 0):
        bottom_left_test_side = True
        bottom_left_side_corr = abs(bottom_left_to_side)
        bottom_left_to_side = 0
    if(bottom_left_to_side > dose_slice.shape[1]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_left_down < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_left_down > dose_slice.shape[0]):
        bottom_left_down_corr = bottom_left_down - dose_slice.shape[0]
        bottom_left_down = dose_slice.shape[0]
        bottom_left_test_down = True

    bottom_right_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[3][0]
    bottom_right_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[3][1]
    if(bottom_right_to_side < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_right_to_side > dose_slice.shape[1]):
        bottom_right_side_corr = bottom_right_to_side - dose_slice.shape[1]
        bottom_right_to_side = dose_slice.shape[1]
        bottom_right_test_side = True
    if(bottom_right_down < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_right_down > dose_slice.shape[0]):
        bottom_right_down_corr = bottom_right_down - dose_slice.shape[0]
        bottom_right_down = dose_slice.shape[0]
        bottom_right_test_down = True


    if(top_right_test_side or top_right_test_down or top_left_test_side or \
        top_left_test_down or bottom_right_test_side or bottom_right_test_down \
            or bottom_left_test_side or bottom_left_test_down):
        ROI_info = "Left side: " \
            + str(max(top_left_side_corr, bottom_left_side_corr)) + " pixels.\n"\
                + "Right side: " + str(max(top_right_side_corr, \
                    bottom_right_side_corr)) + " pixels.\n "+ "Top side: " + \
                        str(max(top_left_down_corr, top_right_down_corr)) +\
                             " pixels.\n"+ "Bottom side: " + \
                                str(max(bottom_left_down_corr, \
                                    bottom_right_down_corr)) + " pixels."
        messagebox.showinfo("ROI info", "The ROI marked on \
the film did not fit with the size of the doseplan and had to \
be cut.\n" + ROI_info )

    for i in range(len(structures_in_plane)):
        for j in range(len(structures_in_plane[i][1])):
```

```
            row = structures_in_plane[i][1][j]
            if row < top_left_down:
                structures_in_plane[i][1][j] = 0
            elif row > bottom_left_down:
                structures_in_plane[i][1][j] = bottom_left_down-top_left_down
            else:
                structures_in_plane[i][1][j] = row - top_left_down
        for k in range(len(structures_in_plane[i][2])):
            column = structures_in_plane[i][2][k]
            if column < top_left_to_side:
                structures_in_plane[i][2][k] = 0
            elif column > top_right_to_side:
                structures_in_plane[i][2][k] = top_right_to_side-top_left_to_side
            else:
                structures_in_plane[i][2][k] = column - top_left_to_side


    doseplan_ROI_coords.append([top_left_to_side, top_left_down])
    doseplan_ROI_coords.append([top_right_to_side, top_right_down])
    doseplan_ROI_coords.append([bottom_left_to_side, bottom_left_down])
    doseplan_ROI_coords.append([bottom_right_to_side, bottom_right_down])

    if only_one:
        Globals.DVH_doseplan_dataset_ROI = \
            dose_slice[int(top_left_down):int(bottom_left_down), \
                int(top_left_to_side):int(top_right_to_side)]*\
                    Globals.DVH_dataset_doseplan.DoseGridScaling

        if(Globals.DVH_dataset_doseplan.PixelSpacing==[1, 1]):
            factor = 5
        elif(Globals.DVH_dataset_doseplan.PixelSpacing==[2, 2]):
            factor = 10
        else:
            factor = 15

        img=Globals.DVH_doseplan_dataset_ROI
        drawing_of_contours = dose_slice[int(top_left_down):int(bottom_left_down)+1,\
            int(top_left_to_side):int(top_right_to_side)+1]*\
                Globals.DVH_dataset_doseplan.DoseGridScaling
        img_film = Globals.DVH_film_dataset_ROI_red_channel_dose/100

        structures = []
        for j in range(len(structures_in_plane)):
            temp_struct = []
            for i in range(len(structures_in_plane[j][1])):
```

```
            drawing_of_contours[int(structures_in_plane[j][1][i]), \
                int(structures_in_plane[j][2][i])] = 0
            temp_struct.append(np.array([int(structures_in_plane[j][1][i]), \
                int(structures_in_plane[j][2][i])]))
        structures.append(temp_struct)

    dvh_data = []
    dvh_data_film = []
    for k in range(len(structures)):
        number_of_points = 0
        dose = []
        dose_film = []
        number_of_points_per_dose = []
        number_of_points_per_dose_film = []
        vertics = mpl.path.Path(structures[k])
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                if vertics.contains_point((i,j)):
                    number_of_points+=1
                    try:
                        idx = dose.index(img[i,j])
                        exist = True
                    except:
                        exist = False
                    if not exist:
                        dose.append(img[i,j])
                        number_of_points_per_dose.append(1)
                    else:
                        number_of_points_per_dose[idx] +=1
                    try:
                        idx_film = dose_film.index(img_film[i*factor, j*factor])
                        exist_film = True
                    except:
                        exist_film = False
                    if not exist_film:
                        if not i == img.shape[0]:
                            dose_film.append(img_film[i*factor,j*factor])
                            number_of_points_per_dose_film.append(1)
                    else:
                        number_of_points_per_dose_film[idx_film] +=1

        temp_dvh_data = []
        for i in range(len(dose)):
            temp_dvh_data.append([dose[i], number_of_points_per_dose[i]])
        temp_dvh_data.sort(key=lambda x: x[0])
        temp_dvh_data_film = []
        for i in range(len(dose_film)):
            temp_dvh_data_film.append([dose_film[i], \
                number_of_points_per_dose_film[i]])
        temp_dvh_data_film.sort(key=lambda x: x[0])

        dose_array = np.array([item[0] for item in temp_dvh_data])
        prc_array = [item[1] for item in temp_dvh_data]
        for i in range(len(dose_array)):
            temp = prc_array[i:len(prc_array)]
            prc_array[i] = sum(temp)
        dvh_data.append([structures_in_plane[k][0], dose_array, \
            np.array(prc_array)/number_of_points])

        dose_array_film = np.array([item[0] for item in temp_dvh_data_film])
        prc_array_film = [item[1] for item in temp_dvh_data_film]
        for i in range(len(dose_array)):
            temp = prc_array_film[i:len(prc_array_film)]
            prc_array_film[i] = sum(temp)
        dvh_data_film.append([structures_in_plane[k][0], dose_array_film, \
            np.array(prc_array_film)/number_of_points])

    if(Globals.DVH_dataset_doseplan.PixelSpacing==[1, 1]):
        img = cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5))
    elif(Globals.DVH_dataset_doseplan.PixelSpacing==[2, 2]):
        img = cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10))
    else:
        img = cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15))

    mx=np.max(img)
    Globals.DVH_max_dose_doseplan = mx*Globals.DVH_dose_scaling_doseplan
    img = img/mx
    PIL_img_doseplan_ROI = Image.fromarray(np.uint8(cm.viridis(img)*255))

    wid = PIL_img_doseplan_ROI.width;heig = PIL_img_doseplan_ROI.height
    Globals.temp_image_canvas.destroy()
    doseplan_canvas = tk.Canvas(Globals.DVH_view_film_doseplan_ROI)
    doseplan_canvas.grid(row=2, column=0, sticky=N+S+W+E)
    doseplan_canvas.config(bg='#ffffff', relief=FLAT, highlightthickness=0, \
        height=max(heig, Globals.profiles_doseplan_text_image.height()), \
            width=wid + Globals.profiles_doseplan_text_image.width())

    Globals.DVH_doseplan_write_image = tk.Canvas(doseplan_canvas)
    Globals.DVH_doseplan_write_image.grid(row=0,column=1,sticky=N+S+W+E)
```

```
        Globals.DVH_doseplan_write_image.config(bg='#ffffff', relief=FLAT,\
            highlightthickness=0, width=wid, height=heig)

        scaled_image_visual = PIL_img_doseplan_ROI
        scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual)
        Globals.DVH_doseplan_write_image_width = scaled_image_visual.width()
        Globals.DVH_doseplan_write_image_height = scaled_image_visual.height()
        Globals.DVH_doseplan_write_image.create_image(0,0,\
            image=scaled_image_visual, anchor="nw")
        Globals.DVH_doseplan_write_image.image = scaled_image_visual

        calculateDVH(dvh_data, dvh_data_film, True, [])

def processDoseplan_usingIsocenter(only_one):
#-----------------------------------------------------
# Function to process the doseplan when isocenter
#  is choosen to define the position.
# Is called in UploadDoseplan
#
# Parameter: only_one -to test if only one doseplan
# is uploaded in Fidora.
#-----------------------------------------------------

    ################ RT Plan ######################
    iso_1 = abs(Globals.DVH_dataset_doseplan.ImagePositionPatient[0] - \
        Globals.DVH_isocenter_mm[0])
    iso_2 = abs(Globals.DVH_dataset_doseplan.ImagePositionPatient[1] - \
        Globals.DVH_isocenter_mm[1])
    iso_3 = abs(Globals.DVH_dataset_doseplan.ImagePositionPatient[2] - \
        Globals.DVH_isocenter_mm[2])

    Globals.DVH_isocenter_mm = [iso_1, iso_2, iso_3]

    isocenter_px = np.zeros(3)
    distance_in_doseplan_ROI_reference_point_px = []
    if(Globals.DVH_dataset_doseplan.PixelSpacing==[1, 1]):
        isocenter_px[0] = np.round(iso_1)
        isocenter_px[1] = np.round(iso_2)
        isocenter_px[2] = np.round(iso_3)

        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.DVH_distance_isocenter_ROI[0][0]),\
                np.round(Globals.DVH_distance_isocenter_ROI[0][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.DVH_distance_isocenter_ROI[1][0]),\
                np.round(Globals.DVH_distance_isocenter_ROI[1][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.DVH_distance_isocenter_ROI[2][0]),\
                np.round(Globals.DVH_distance_isocenter_ROI[2][1])])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round(Globals.DVH_distance_isocenter_ROI[3][0]),\
                np.round(Globals.DVH_distance_isocenter_ROI[3][1])])

    elif(Globals.DVH_dataset_doseplan.PixelSpacing==[2, 2]):
        isocenter_px[0] = np.round(iso_1/2)
        isocenter_px[1] = np.round(iso_2/2)
        isocenter_px[2] = np.round(iso_3/2)

        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_isocenter_ROI[0][0])/2),\
                np.round((Globals.DVH_distance_isocenter_ROI[0][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_isocenter_ROI[1][0])/2),\
                np.round((Globals.DVH_distance_isocenter_ROI[1][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_isocenter_ROI[2][0])/2),\
                np.round((Globals.DVH_distance_isocenter_ROI[2][1])/2)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_isocenter_ROI[3][0])/2),\
                np.round((Globals.DVH_distance_isocenter_ROI[3][1])/2)])

    else:
        isocenter_px[0] = np.round(iso_1/3)
        isocenter_px[1] = np.round(iso_2/3)
        isocenter_px[2] = np.round(iso_3/3)

        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_isocenter_ROI[0][0])/3),\
                np.round((Globals.DVH_distance_isocenter_ROI[0][1])/3)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_isocenter_ROI[1][0])/3),\
                np.round((Globals.DVH_distance_isocenter_ROI[1][1])/3)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_isocenter_ROI[2][0])/3),\
                np.round((Globals.DVH_distance_isocenter_ROI[2][1])/3)])
        distance_in_doseplan_ROI_reference_point_px.append\
            ([np.round((Globals.DVH_distance_isocenter_ROI[3][0])/3),\
                np.round((Globals.DVH_distance_isocenter_ROI[3][1])/3)])

    reference_point = np.zeros(3)
```

```
######################### Struct file ####################################
i=0
for name in Globals.DVH_contour_names:
    sequences = Globals.DVH_ROIContourSequence[i].ContourSequence
    temp_x = [];temp_y = [];temp_z = []
    for sequence in sequences:
        if(Globals.DVH_dataset_doseplan.PixelSpacing==[1, 1]):
            temp_x.append([int(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[0]-x)) for x in sequence.ContourData[0::3]])
            temp_y.append([int(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[1]-y)) for y in sequence.ContourData[1::3]])
            temp_z.append([int(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[2]-z)) for z in sequence.ContourData[2::3]])
        elif(Globals.DVH_dataset_doseplan.PixelSpacing==[2, 2]):
            temp_x.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[0]-x)/2)) for \
                    x in sequence.ContourData[0::3]])
            temp_y.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[1]-y)/2)) for \
                    y in sequence.ContourData[1::3]])
            temp_z.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[2]-z)/2)) for \
                    z in sequence.ContourData[2::3]])
        else:
            temp_x.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[0]-x)/3)) for \
                    x in sequence.ContourData[0::3]])
            temp_y.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[1]-y)/3)) for \
                    y in sequence.ContourData[1::3]])
            temp_z.append([int(np.round(abs(Globals.DVH_dataset_doseplan.\
                ImagePositionPatient[2]-z)/3)) for \
                    z in sequence.ContourData[2::3]])
    Globals.DVH_contours.append\
        ([i, name.ROIObservationLabel, temp_x, temp_y, temp_z])
    i+=1

######################### Doseplan ######################################
if(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[1, 0, 0, 0, 1, 0]):

    reference_point[0] = isocenter_px[2]
    reference_point[1] = isocenter_px[1]
    reference_point[2] = isocenter_px[0]
```

```
        for i in range(len(Globals.DVH_contours)):
            temp_x = Globals.DVH_contours[i][2]
            temp_y = Globals.DVH_contours[i][3]
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_z
            Globals.DVH_contours[i][3] = temp_y
            Globals.DVH_contours[i][4] = temp_x

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = Globals.DVH_dataset_doseplan.pixel_array
        else:
            messagebox.showerror("Error", "Something has gone wrong here.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[1, 0, 0, 0, 0, 1]):
        reference_point[0] = isocenter_px[1]
        reference_point[1] = isocenter_px[2]
        reference_point[2] = isocenter_px[0]
        for i in range(len(Globals.DVH_contours)):
            temp_x = Globals.DVH_contours[i][2]
            temp_y = Globals.DVH_contours[i][3]
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_y
            Globals.DVH_contours[i][3] = temp_z
            Globals.DVH_contours[i][4] = temp_x
        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = Globals.DVH_dataset_doseplan.pixel_array
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[0, 1, 0, 1, 0, 0]):
        reference_point[0] = isocenter_px[2]
        reference_point[1] = isocenter_px[0]
        reference_point[2] = isocenter_px[1]
        for i in range(len(Globals.DVH_contours)):
            temp_x = Globals.DVH_contours[i][2]
            temp_y = Globals.DVH_contours[i][3]
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_z
            Globals.DVH_contours[i][3] = temp_x
            Globals.DVH_contours[i][4] = temp_y

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[0, 1, 0, 0, 0, 1]):
        reference_point[0] = isocenter_px[0]
        reference_point[1] = isocenter_px[2]
        reference_point[2] = isocenter_px[1]
        for i in range(len(Globals.DVH_contours)):
            temp_x = Globals.DVH_contours[i][2]
            temp_y = Globals.DVH_contours[i][3]
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_x
            Globals.DVH_contours[i][3] = temp_z
            Globals.DVH_contours[i][4] = temp_y

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[0, 0, 1, 1, 0, 0]):
        reference_point[0] = isocenter_px[1]
        reference_point[1] = isocenter_px[0]
        reference_point[2] = isocenter_px[2]
        for i in range(len(Globals.DVH_contours)):
            temp_x = Globals.DVH_contours[i][2]
            temp_y = Globals.DVH_contours[i][3]
            temp_z = Globals.DVH_contours[i][4]
            Globals.DVH_contours[i][2] = temp_y
            Globals.DVH_contours[i][3] = temp_x
            Globals.DVH_contours[i][4] = temp_z

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 1,2)
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_4
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    elif(Globals.DVH_dataset_doseplan.ImageOrientationPatient==[0, 0, 1, 0, 1, 0]):
        reference_point[0] = isocenter_px[0]
        reference_point[1] = isocenter_px[1]
        reference_point[2] = isocenter_px[2]

        if(Globals.DVH_film_orientation.get()=='Coronal'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
            dataset_swapped = np.swapaxes(dataset_swapped, 0,1)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][3]
                Globals.DVH_contours[i][3] = temp_c
        elif(Globals.DVH_film_orientation.get()=='Sagittal'):
            dataset_swapped = Globals.DVH_dataset_doseplan.pixel_array
        elif(Globals.DVH_film_orientation.get()=='Axial'):
            dataset_swapped = \
                np.swapaxes(Globals.DVH_dataset_doseplan.pixel_array, 0,2)
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[2]
            reference_point[2] = temp_ref
            for i in range(len(Globals.DVH_contours)):
                temp_c = Globals.DVH_contours[i][2]
                Globals.DVH_contours[i][2] = Globals.DVH_contours[i][4]
                Globals.DVH_contours[i][4] = temp_c
        else:
            messagebox.showerror("Error", "Something has gone wrong.")
            clearAll()
            return
    else:
        messagebox.showerror("Error", "Something has gone wrong.")
        clearAll()
        return



    ####################### Match film and doseplan ###############################
    if Globals.DVH_dataset_doseplan.PixelSpacing == [1, 1]:
        offset = int(np.round(Globals.DVH_offset))
        dose_slice = dataset_swapped[int(reference_point[0]) + offset,:,:]
    elif Globals.DVH_dataset_doseplan.PixelSpacing == [2, 2]:
        offset = int(np.round(Globals.DVH_offset/2))
        dose_slice = dataset_swapped[int(reference_point[0] + offset),:,:]
    else:
        offset = int(np.round(Globals.DVH_offset/3))
        dose_slice = dataset_swapped[int(reference_point[0]) + offset,:,:]


    structures_in_plane = []
    for contour in Globals.DVH_contours:
        in_plane = False
        temp_row=[];temp_column=[]
        for i in range(len(contour[2])):
            lst = contour[2][i]
            for j in range(len(lst)):
                plane = lst[j]
                if(plane == int(reference_point[0])+offset):
                    temp_row.append(contour[3][i][j])
                    temp_column.append(contour[4][i][j])
                    in_plane = True
        if in_plane:
            structures_in_plane.append([contour[1], temp_row, temp_column])

    doseplan_ROI_coords = []
    top_left_test_side = False; top_left_test_down = False
    top_right_test_side = False; top_right_test_down = False
    bottom_left_test_side = False; bottom_left_test_down = False
    bottom_right_test_side = False; bottom_right_test_down = False
    top_left_side_corr = 0; top_left_down_corr = 0
    top_right_side_corr = 0; top_right_down_corr = 0
    bottom_left_side_corr = 0; bottom_left_down_corr = 0
    bottom_right_side_corr = 0; bottom_right_down_corr = 0


    top_left_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[0][0]
    top_left_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[0][1]
    if(top_left_to_side < 0):
        top_left_test_side = True
        top_left_side_corr = abs(top_left_to_side)
        top_left_to_side = 0
    if(top_left_to_side > dose_slice.shape[1]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(top_left_down < 0):
        top_left_test_down = True
        top_left_down_corr = abs(top_left_down)
        top_left_down = 0
    if(top_left_down > dose_slice.shape[0]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return

    top_right_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[1][0]
    top_right_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[1][1]
    if(top_right_to_side < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(top_right_to_side > dose_slice.shape[1]):
        top_right_test_side = True
        top_right_side_corr = top_right_to_side - dose_slice.shape[1]
        top_right_to_side = dose_slice.shape[1]
    if(top_right_down < 0):
        top_right_test_down = True
        top_right_down_corr = abs(top_right_down)
        top_right_down = 0
    if(top_right_down > dose_slice.shape[0]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return

    bottom_left_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[2][0]
    bottom_left_down = reference_point[1] - \
        distance_in_doseplan_ROI_reference_point_px[2][1]
    if(bottom_left_to_side < 0):
        bottom_left_test_side = True
        bottom_left_side_corr = abs(bottom_left_to_side)
        bottom_left_to_side = 0
    if(bottom_left_to_side > dose_slice.shape[1]):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_left_down < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_left_down > dose_slice.shape[0]):
        bottom_left_down_corr = bottom_left_down - dose_slice.shape[0]
        bottom_left_down = dose_slice.shape[0]
        bottom_left_test_down = True

    bottom_right_to_side = reference_point[2] - \
        distance_in_doseplan_ROI_reference_point_px[3][0]
    bottom_right_down = reference_point[1] − \
        distance_in_doseplan_ROI_reference_point_px[3][1]
    if(bottom_right_to_side < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_right_to_side > dose_slice.shape[1]):
        bottom_right_side_corr = bottom_right_to_side − dose_slice.shape[1]
        bottom_right_to_side = dose_slice.shape[1]
        bottom_right_test_side = True
    if(bottom_right_down < 0):
        messagebox.showerror("Fatal Error", "Fatal error: \
marked ROI is out of range in doseplan. Try again")
        clearAll()
        return
    if(bottom_right_down > dose_slice.shape[0]):
        bottom_right_down_corr = bottom_right_down − dose_slice.shape[0]
        bottom_right_down = dose_slice.shape[0]
        bottom_right_test_down = True


    if(top_right_test_side or top_right_test_down or top_left_test_side or \
        top_left_test_down or bottom_right_test_side or bottom_right_test_down or\
            bottom_left_test_side or bottom_left_test_down):
            ROI_info = "Left side: " + str(max(top_left_side_corr, \
                bottom_left_side_corr)) + " pixels.\n"+ "Right side: " + \
                    str(max(top_right_side_corr, bottom_right_side_corr)) +\
                        " pixels.\n "+ "Top side: " + str(max(top_left_down_corr,\
                            top_right_down_corr)) + " pixels.\n"+ "Bottom side: " + \
                                str(max(bottom_left_down_corr, \
                                    bottom_right_down_corr)) + " pixels."
            messagebox.showinfo("ROI info", "The ROI marked on the film did not \
fit with the size of the doseplan and had to be cut.\n" + ROI_info)

    for i in range(len(structures_in_plane)):
        for j in range(len(structures_in_plane[i][1])):
            row = structures_in_plane[i][1][j]
            if row < top_left_down:
                structures_in_plane[i][1][j] = 0
            elif row > bottom_left_down:
                structures_in_plane[i][1][j] = bottom_left_down−top_left_down
            else:
                structures_in_plane[i][1][j] = row − top_left_down
        for k in range(len(structures_in_plane[i][2])):
            column = structures_in_plane[i][2][k]
            if column < top_left_to_side:
                structures_in_plane[i][2][k] = 0
            elif column > top_right_to_side:
                structures_in_plane[i][2][k] = top_right_to_side-top_left_to_side
            else:
                structures_in_plane[i][2][k] = column - top_left_to_side


    doseplan_ROI_coords.append([top_left_to_side, top_left_down])
    doseplan_ROI_coords.append([top_right_to_side, top_right_down])
    doseplan_ROI_coords.append([bottom_left_to_side, bottom_left_down])
    doseplan_ROI_coords.append([bottom_right_to_side, bottom_right_down])

    #dose_slice = cv2.flip(dose_slice, 1)
    if(only_one):
        Globals.DVH_doseplan_dataset_ROI = \
            dose_slice[int(top_left_down):int(bottom_left_down), \
                int(top_left_to_side):int(top_right_to_side)]*\
                    Globals.DVH_dataset_doseplan.DoseGridScaling

        if(Globals.DVH_dataset_doseplan.PixelSpacing==[1, 1]):
            factor = 5
        elif(Globals.DVH_dataset_doseplan.PixelSpacing==[2, 2]):
            factor = 10
        else:
            factor = 15

        img=Globals.DVH_doseplan_dataset_ROI
        img_film = Globals.DVH_film_dataset_ROI_red_channel_dose/100
        drawing_of_contours = dose_slice[int(top_left_down):\
            int(bottom_left_down)+1, int(top_left_to_side):int(top_right_to_side)+1]\
                *Globals.DVH_dataset_doseplan.DoseGridScaling

        structures = []
        for j in range(len(structures_in_plane)):
            temp_struct = []
            temp_struct_film = []
            for i in range(len(structures_in_plane[j][1])):
                drawing_of_contours[int(structures_in_plane[j][1][i]), \
                    int(structures_in_plane[j][2][i])] = 0
                temp_struct.append(np.array([int(structures_in_plane[j][1][i]), \
                    int(structures_in_plane[j][2][i])]))
            structures.append(temp_struct)
```

```
dvh_data = []
dvh_data_film = []

for k in range(len(structures)):
    number_of_points = 0
    dose = []
    dose_film = []
    number_of_points_per_dose = []
    number_of_points_per_dose_film = []
    vertics = mpl.path.Path(structures[k])
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if vertics.contains_point((i,j)):
                number_of_points+=1
                try:
                    idx = dose.index(img[i,j])
                    exist = True
                except:
                    exist = False
                if not exist:
                    dose.append(img[i,j])
                    number_of_points_per_dose.append(1)
                else:
                    number_of_points_per_dose[idx] +=1
                try:
                    idx_film = dose_film.index(img_film[i*factor, j*factor])
                    exist_film = True
                except:
                    exist_film = False
                if not exist_film:
                    if not i == img.shape[0]:
                        dose_film.append(img_film[i*factor,j*factor])
                        number_of_points_per_dose_film.append(1)
                else:
                    number_of_points_per_dose_film[idx_film] +=1

    temp_dvh_data = []
    for i in range(len(dose)):
        temp_dvh_data.append([dose[i], number_of_points_per_dose[i]])
    temp_dvh_data.sort(key=lambda x: x[0])

    temp_dvh_data_film = []
    for i in range(len(dose_film)):
        temp_dvh_data_film.append([dose_film[i], \
            number_of_points_per_dose_film[i]])
    temp_dvh_data_film.sort(key=lambda x: x[0])

    dose_array = np.array([item[0] for item in temp_dvh_data])
    prc_array = [item[1] for item in temp_dvh_data]
    for i in range(len(dose_array)):
        temp = prc_array[i:len(prc_array)]
        prc_array[i] = sum(temp)
    dvh_data.append([structures_in_plane[k][0], dose_array, \
        np.array(prc_array)/number_of_points])

    dose_array_film = np.array([item[0] for item in temp_dvh_data_film])
    prc_array_film = [item[1] for item in temp_dvh_data_film]
    for i in range(len(dose_array_film)):
        temp = prc_array_film[i:len(prc_array_film)]
        prc_array_film[i] = sum(temp)
    dvh_data_film.append([structures_in_plane[k][0], \
        dose_array_film, np.array(prc_array_film)/number_of_points])


if(Globals.DVH_dataset_doseplan.PixelSpacing==[1, 1]):
    img = cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5))
elif(Globals.DVH_dataset_doseplan.PixelSpacing==[2, 2]):
    img = cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10))
else:
    img = cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15))

mx=np.max(img)

Globals.DVH_max_dose_doseplan = mx*Globals.DVH_dose_scaling_doseplan
max_dose = mx*Globals.DVH_dose_scaling_doseplan
img = img/mx
PIL_img_doseplan_ROI = Image.fromarray(np.uint8(cm.viridis(img)*255))

wid = PIL_img_doseplan_ROI.width;heig = PIL_img_doseplan_ROI.height
Globals.temp_image_canvas.destroy()
doseplan_canvas = tk.Canvas(Globals.DVH_view_film_doseplan_ROI)
doseplan_canvas.grid(row=2, column=0, sticky=N+S+W+E)
doseplan_canvas.config(bg='#ffffff', relief=FLAT, highlightthickness=0, \
    height=max(heig, Globals.profiles_doseplan_text_image.height()), \
        width=wid + Globals.profiles_doseplan_text_image.width())


Globals.DVH_doseplan_write_image = tk.Canvas(doseplan_canvas)
```

```
        Globals.DVH_doseplan_write_image.grid(row=0,column=1,sticky=N+S+W+E)
        Globals.DVH_doseplan_write_image.config(bg='#ffffff', relief=FLAT, \
            highlightthickness=0, width=wid, height=heig)

        scaled_image_visual = PIL_img_doseplan_ROI
        scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual)
        Globals.DVH_doseplan_write_image_width = scaled_image_visual.width()
        Globals.DVH_doseplan_write_image_height = scaled_image_visual.height()
        Globals.DVH_doseplan_write_image.create_image(0,0,\
            image=scaled_image_visual, anchor="nw")
        Globals.DVH_doseplan_write_image.image = scaled_image_visual

        calculateDVH(dvh_data, dvh_data_film, True, [])


def UploadDoseplan(only_one):
#----------------------------------------------------
# Function to upload the doseplan
#
# Is called in function UploadDoseplan_button()
#----------------------------------------------------

    file = filedialog.askopenfilename()
    ext = os.path.splitext(file)[-1].lower()
    if(not(ext == '.dcm')):
        if(ext == ""):
            return
        else:
            messagebox.showerror("Error", "The file must be a *.dcm file")
            return

    current_folder = os.getcwd()
    parent = os.path.dirname(file)
    os.chdir(parent)
    dataset = pydicom.dcmread(file)
    try:
        dose_summation_type = dataset.DoseSummationType
    except:
        messagebox.showerror("Error", "Could not upload the \
doseplan correctly. Try again or another file.\n (Code: dose summation)")
        return

    if(not(dose_summation_type == "PLAN")):
        ok = messagebox.askokcancel("Dose summation", "You \
did not upload the full doseplan. Do you want to continue?")
```

```
        if not ok:
            return
    os.chdir(current_folder)
    doseplan_dataset = dataset.pixel_array
    if(not((dataset.PixelSpacing==[1, 1] and dataset.SliceThickness==1) \
        or (dataset.PixelSpacing==[2, 2] and dataset.SliceThickness==2) \
        or (dataset.PixelSpacing==[3, 3] and dataset.SliceThickness==3))):
        messagebox.showerror("Error", "The resolution \
in doseplan must be 1x1x1, 2x2x2 or 3x3x3")
        return
    if(not(dataset.ImageOrientationPatient==[1, 0, 0, 0, 1, 0] or \
        dataset.ImageOrientationPatient==[1, 0, 0, 0, 0, 1] or \
        dataset.ImageOrientationPatient==[0, 1, 0, 1, 0, 0] or \
        dataset.ImageOrientationPatient==[0, 1, 0, 0, 0, 1] or \
        dataset.ImageOrientationPatient==[0, 0, 1, 1, 0, 0] or \
        dataset.ImageOrientationPatient==[0, 0, 1, 0, 1, 0])):
        messagebox.showerror("Error", "The Image Orientation \
(Patient) must be parallel to one of the main \
axis and perpendicular to the two others.")
        return

    Globals.DVH_dataset_doseplan = dataset
    Globals.DVH_dose_scaling_doseplan = dataset.DoseGridScaling
    Globals.DVH_test_if_added_doseplan = True
    if(Globals.DVH_test_if_added_rtplan and Globals.DVH_test_if_added_struct):
        if(Globals.DVH_isocenter_or_reference_point == "Isocenter"):
            processDoseplan_usingIsocenter(only_one)
        elif(Globals.DVH_isocenter_or_reference_point == "Ref_point"):
            processDoseplan_usingReferencePoint(only_one)
        else:
            messagebox.showerror("Error", "Something went wrong. Try again.\n\
(Code: processDoseplan)")
            return

    if only_one:
        Globals.DVH_upload_button_doseplan.config(state=DISABLED)

def UploadDoseplan_button_function():
#------------------------------------------------------------
# Function to call on the function to upload doseplan
#
# Is a callback function for button
# DVH_upload_button_doseplan in notebook.py
#------------------------------------------------------------
    UploadDoseplan(True)
```

```
def UploadRTplan():
#------------------------------------------------------------
# Function to upload the RT Plan file
#
# Is a callback function for button
# DVH_upload_button_rtplan in notebook.py
#------------------------------------------------------------
    file = filedialog.askopenfilename()
    ext = os.path.splitext(file)[-1].lower()
    if(not(ext == '.dcm')):
        if(ext == ""):
            return
        else:
            messagebox.showerror("Error", "The file must be a *.dcm file")
            return

    current_folder = os.getcwd()
    parent = os.path.dirname(file)
    os.chdir(parent)
    dataset = pydicom.dcmread(file)
    os.chdir(current_folder)
    Globals.DVH_dataset_rtplan = dataset

    try:
        isocenter_mm = \
            dataset.BeamSequence[0].ControlPointSequence[0].IsocenterPosition
        Globals.DVH_isocenter_mm = isocenter_mm

    except:
        messagebox.showerror("Error", "Could not read the RT plan file. \
Try again or try another file.\n\
(Code: isocenter reading)")
        return

    try:
        Globals.DVH_doseplan_vertical_displacement = \
            dataset.PatientSetupSequence[0].TableTopVerticalSetupDisplacement
    except:
        messagebox.showerror("Error", "Could not read the RT plan file. \
Try again or try another file. \n\
(Code: vertical table displacement)")

    try:
        Globals.DVH_doseplan_lateral_displacement = \
            dataset.PatientSetupSequence[0].TableTopLateralSetupDisplacement
    except:
        messagebox.showerror("Error", "Could not read the RT plan file. \
Try again or try another file-\n\
(Code: lateral table displacement)")

    try:
        Globals.DVH_doseplan_longitudianl_displacement = \
            dataset.PatientSetupSequence[0].TableTopLongitudinalSetupDisplacement
    except:
        messagebox.showerror("Error", "Could not read the RT plan file. \
Try again or try another file\n\
(Code: longitudinal table displacement)")

    try:
        patient_position = dataset.PatientSetupSequence[0].PatientPosition
        Globals.DVH_doseplan_patient_position = patient_position
    except:
        messagebox.showerror("Error", "Could not read the RT plan file. \
Try again or try another file\n\
(Code: Patient position)")

    if(not(patient_position=='HFS' or patient_position=='HFP' or\
        patient_position=='HFDR' or patient_position == 'HFDL'or \
            patient_position=='FFDR' or patient_position=='FFDL' or \
                patient_position=='FFP' or patient_position=='FFS')):
        messagebox.showerror("Error", "Fidora does only support \
patient positions: \nHFS, HFP, HFDR, HFDL, FFP, FFS, FFDR, FFDL")
        return

    Globals.DVH_test_if_added_rtplan = True
    Globals.DVH_upload_button_struct.config(state=ACTIVE)
    Globals.DVH_upload_button_rtplan.config(state=DISABLED)

def UploadStruct():
#----------------------------------------------------
# Function to upload the structure file
#
# Is a callback function for button
# DVH_upload_button_struct in notebook.py
#----------------------------------------------------
    file = filedialog.askopenfilename()
    ext = os.path.splitext(file)[-1].lower()
    if(not(ext == '.dcm')):
        if(ext == ""):
            return
        else:
            messagebox.showerror("Error", "The file must be a *.dcm file")
            return

    current_folder = os.getcwd()
    parent = os.path.dirname(file)
    os.chdir(parent)
    dataset = pydicom.dcmread(file)
    os.chdir(current_folder)
    Globals.DVH_dataset_structure_file = dataset

    try:
        Globals.DVH_contour_names = dataset.RTROIObservationsSequence
    except:
        messagebox.showerror("Error", "You must upload a strucure file \n\
(Code: UploadStruct())")
        return

    try:
        Globals.DVH_ROIContourSequence = dataset.ROIContourSequence
    except:
        messagebox.showerror("Error", "You must upload a strucure file \n\
(Code: UploadStruct())")
        return

    Globals.DVH_test_if_added_struct = True
    Globals.DVH_upload_button_doseplan.config(state=ACTIVE)
    Globals.DVH_upload_button_struct.config(state=DISABLED)



def pixel_to_dose(P,a,b,c):
#-----------------------------------------------------
# Function to map between pixel value and dose
#
#-----------------------------------------------------
    ret = c + b/(P-a)
    return ret

def markIsocenter(img, new_window_isocenter_tab, image_canvas, cv2Img):
#-----------------------------------------------------
# Function mark the Isocenter
#
# Is a callback function for the button
# mark_isocenter_button in UploadFilm()
#---------------------------------------------------------
    if(len(Globals.DVH_mark_isocenter_oval)>0):
        image_canvas.delete(Globals.DVH_mark_isocenter_up_down_line[0])
        image_canvas.delete(Globals.DVH_mark_isocenter_right_left_line[0])
        image_canvas.delete(Globals.DVH_mark_isocenter_oval[0])

        Globals.DVH_mark_isocenter_oval=[]
        Globals.DVH_mark_isocenter_right_left_line=[]
        Globals.DVH_mark_isocenter_up_down_line=[]

    Globals.DVH_iscoenter_coords = []
    img_mark_isocenter = ImageTk.PhotoImage(image=img)
    mark_isocenter_window = tk.Toplevel(new_window_isocenter_tab)
    mark_isocenter_window.geometry("1035x620+10+10")
    mark_isocenter_window.grab_set()

    mark_isocenter_over_all_frame = tk.Frame(mark_isocenter_window, \
        bd=0, relief=FLAT)
    mark_isocenter_over_all_canvas = Canvas(mark_isocenter_over_all_frame)

    mark_isocenter_xscrollbar = Scrollbar(mark_isocenter_over_all_frame, \
        orient=HORIZONTAL, command=mark_isocenter_over_all_canvas.xview)
    mark_isocenter_yscrollbar = Scrollbar(mark_isocenter_over_all_frame, \
        command=mark_isocenter_over_all_canvas.yview)

    mark_isocenter_scroll_frame = ttk.Frame(mark_isocenter_over_all_canvas)
    mark_isocenter_scroll_frame.bind("<Configure>", lambda e: \
        mark_isocenter_over_all_canvas.configure\
            (scrollregion=mark_isocenter_over_all_canvas.bbox('all')))

    mark_isocenter_over_all_canvas.create_window((0,0), \
        window=mark_isocenter_scroll_frame, anchor='nw')
    mark_isocenter_over_all_canvas.configure\
        (xscrollcommand=mark_isocenter_xscrollbar.set, \
            yscrollcommand=mark_isocenter_yscrollbar.set)

    mark_isocenter_over_all_frame.config(highlightthickness=0, bg='#ffffff')
    mark_isocenter_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
    mark_isocenter_over_all_frame.pack(expand=True, fill=BOTH)
    mark_isocenter_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
    mark_isocenter_over_all_frame.grid_columnconfigure(0, weight=1)
    mark_isocenter_over_all_frame.grid_rowconfigure(0, weight=1)
    mark_isocenter_xscrollbar.grid(row=1, column=0, sticky=E+W)
    mark_isocenter_over_all_frame.grid_columnconfigure(1, weight=0)
    mark_isocenter_over_all_frame.grid_rowconfigure(1, weight=0)
    mark_isocenter_yscrollbar.grid(row=0, column=1, sticky=N+S)
    mark_isocenter_over_all_frame.grid_columnconfigure(2, weight=0)
    mark_isocenter_over_all_frame.grid_rowconfigure(2, weight=0)

    mark_isocenter_image_canvas = tk.Canvas(mark_isocenter_scroll_frame)
    mark_isocenter_image_canvas.grid(row=0,column=0, rowspan=10, \
        columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
    mark_isocenter_scroll_frame.grid_columnconfigure(0, weight=0)
    mark_isocenter_scroll_frame.grid_rowconfigure(0, weight=0)

    mark_isocenter_image_canvas.create_image(0,0,\
        image=img_mark_isocenter,anchor="nw")
    mark_isocenter_image_canvas.image = img_mark_isocenter
    mark_isocenter_image_canvas.config(cursor='hand2', \
        bg='#ffffff', relief=FLAT, bd=0, \
        scrollregion=mark_isocenter_image_canvas.bbox(ALL), \
            height=img_mark_isocenter.height(), width=img_mark_isocenter.width())
    mark_isocenter_image_canvas.grid_propagate(0)

    def findCoords(event):
        mark_isocenter_image_canvas.create_oval\
            (event.x-2, event.y-2, event.x+2, event.y+2, fill='red')
        if(Globals.DVH_iscoenter_coords==[]):
            Globals.DVH_iscoenter_coords.append([event.x, event.y])
            mark_isocenter_image_canvas.config(cursor='hand2')

        elif(len(Globals.DVH_iscoenter_coords)==1):
            Globals.DVH_iscoenter_coords.append([event.x, event.y])
            Globals.DVH_film_isocenter = \
                [Globals.DVH_iscoenter_coords[0][0], \
                    Globals.DVH_iscoenter_coords[1][1]]
            x1,y1 = Globals.DVH_iscoenter_coords[0]
            x4,y4 = Globals.DVH_iscoenter_coords[1]
            x2 = x1;y3=y4
            y2=2*Globals.DVH_film_isocenter[1]-y1
            x3=2*Globals.DVH_film_isocenter[0]-x4
            up_down_line = image_canvas.create_line\
                (int(x1/2),int(y1/2),int(x2/2),int(y2/2),\
                    fill='purple', smooth=1, width=2)
            right_left_line = image_canvas.create_line\
                (int(x3/2),int(y3/2),int(x4/2),int(y4/2), \
                    fill='purple', smooth=1, width=2)
            oval = image_canvas.create_oval\
                (int(Globals.DVH_film_isocenter[0]/2)-3, \
                    int(Globals.DVH_film_isocenter[1]/2)−3,\
                        int(Globals.DVH_film_isocenter[0]/2)+3, \
                            int(Globals.DVH_film_isocenter[1]/2)+3, fill='red')

            Globals.DVH_mark_isocenter_up_down_line.append(up_down_line)
            Globals.DVH_mark_isocenter_right_left_line.append(right_left_line)
            Globals.DVH_mark_isocenter_oval.append(oval)

            mark_isocenter_window.after(500, lambda: mark_isocenter_window.destroy())
            Globals.DVH_isocenter_check = True
            if(Globals.DVH_ROI_check):
                Globals.DVH_done_button.config(state=ACTIVE)

    mark_isocenter_image_canvas.bind("<Button 1>",findCoords)

def markReferencePoint(img, new_window_reference_point_tab, \
    image_canvas_reference_tab, cv2Img):
#−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
# Function to mark reference point
#
# Is a callback function for button
# mark_point_button in UploadFilm()
#−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    if(len(Globals.DVH_mark_reference_point_oval)>0):
        image_canvas_reference_tab.delete(Globals.DVH_mark_reference_point_oval[0])
        Globals.DVH_mark_reference_point_oval=[]

    img_mark_reference_point = ImageTk.PhotoImage(image=img)
    mark_reference_point_window = tk.Toplevel(new_window_reference_point_tab)
    mark_reference_point_window.geometry("1035x620+10+10")
    mark_reference_point_window.grab_set()

    mark_reference_point_over_all_frame = tk.Frame(mark_reference_point_window, \
        bd=0, relief=FLAT)
    mark_reference_point_over_all_canvas = \
        Canvas(mark_reference_point_over_all_frame)

    mark_reference_point_xscrollbar = \
        Scrollbar(mark_reference_point_over_all_frame, orient=HORIZONTAL, \
            command=mark_reference_point_over_all_canvas.xview)
    mark_reference_point_yscrollbar = \
        Scrollbar(mark_reference_point_over_all_frame, \
            command=mark_reference_point_over_all_canvas.yview)

    mark_reference_point_scroll_frame = \
        ttk.Frame(mark_reference_point_over_all_canvas)
    mark_reference_point_scroll_frame.bind("<Configure>", \
        lambda e: mark_reference_point_over_all_canvas.configure\
            (scrollregion=mark_reference_point_over_all_canvas.bbox('all')))

    mark_reference_point_over_all_canvas.create_window((0,0), \
        window=mark_reference_point_scroll_frame, anchor='nw')
    mark_reference_point_over_all_canvas.configure\
        (xscrollcommand=mark_reference_point_xscrollbar.set, \
            yscrollcommand=mark_reference_point_yscrollbar.set)

    mark_reference_point_over_all_frame.config(highlightthickness=0, bg='#ffffff')
    mark_reference_point_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
    mark_reference_point_over_all_frame.pack(expand=True, fill=BOTH)
    mark_reference_point_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
    mark_reference_point_over_all_frame.grid_columnconfigure(0, weight=1)
    mark_reference_point_over_all_frame.grid_rowconfigure(0, weight=1)
    mark_reference_point_xscrollbar.grid(row=1, column=0, sticky=E+W)
    mark_reference_point_over_all_frame.grid_columnconfigure(1, weight=0)
    mark_reference_point_over_all_frame.grid_rowconfigure(1, weight=0)
    mark_reference_point_yscrollbar.grid(row=0, column=1, sticky=N+S)
    mark_reference_point_over_all_frame.grid_columnconfigure(2, weight=0)
    mark_reference_point_over_all_frame.grid_rowconfigure(2, weight=0)

    mark_reference_point_image_canvas = tk.Canvas(mark_reference_point_scroll_frame)
    mark_reference_point_image_canvas.grid(row=0,column=0, rowspan=10, \
        columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
    mark_reference_point_scroll_frame.grid_columnconfigure(0, weight=0)
    mark_reference_point_scroll_frame.grid_rowconfigure(0, weight=0)

    mark_reference_point_image_canvas.create_image(0,0,\
        image=img_mark_reference_point,anchor="nw")
    mark_reference_point_image_canvas.image = img_mark_reference_point
    mark_reference_point_image_canvas.config(cursor='hand2', \
        bg='#ffffff', relief=FLAT, bd=0, \
        scrollregion=mark_reference_point_image_canvas.bbox(ALL), \
            height=img_mark_reference_point.height(), \
                width=img_mark_reference_point.width())
    mark_reference_point_image_canvas.grid_propagate(0)


    def findCoords(event):
        mark_reference_point_image_canvas.create_oval\
            (event.x—2, event.y—2, event.x+2, event.y+2, fill='red')
        Globals.DVH_film_reference_point = [event.x, event.y]
```

```
        oval = image_canvas_reference_tab.create_oval\
            (int(Globals.DVH_film_reference_point[0]/2)-3, \
                int(Globals.DVH_film_reference_point[1]/2)-3, \
                    int(Globals.DVH_film_reference_point[0]/2)+3, \
                        int(Globals.DVH_film_reference_point[1]/2)+3, fill='red')
        Globals.DVH_mark_reference_point_oval.append(oval)

        mark_reference_point_window.after(500, \
            lambda: mark_reference_point_window.destroy())
        Globals.DVH_reference_point_check = True
        if(Globals.DVH_ROI_reference_point_check):
            Globals.DVH_done_button_reference_point.config(state=ACTIVE)

    mark_reference_point_image_canvas.bind("<Button 1>",findCoords)

def markROI(img, tab, canvas, ref_point_test):
#-----------------------------------------------------------
# Function to mark ROI
#
# Is a callback function for button
# mark_ROI_button in UploadFilm()
#-----------------------------------------------------------
    if(len(Globals.DVH_mark_ROI_rectangle)>0):
        canvas.delete(Globals.DVH_mark_ROI_rectangle[0])
        Globals.DVH_mark_ROI_rectangle = []

    Globals.DVH_ROI_coords = []

    img_mark_ROI = ImageTk.PhotoImage(image=img)
    mark_ROI_window = tk.Toplevel(tab)
    mark_ROI_window.geometry("1035x620+10+10")
    mark_ROI_window.grab_set()

    mark_ROI_over_all_frame = tk.Frame(mark_ROI_window, bd=0, relief=FLAT)
    mark_ROI_over_all_canvas = Canvas(mark_ROI_over_all_frame)

    mark_ROI_xscrollbar = Scrollbar(mark_ROI_over_all_frame, \
        orient=HORIZONTAL, command=mark_ROI_over_all_canvas.xview)
    mark_ROI_yscrollbar = Scrollbar(mark_ROI_over_all_frame, \
        command=mark_ROI_over_all_canvas.yview)

    mark_ROI_scroll_frame = ttk.Frame(mark_ROI_over_all_canvas)
    mark_ROI_scroll_frame.bind("<Configure>", lambda e: \
        mark_ROI_over_all_canvas.configure\
            (scrollregion=mark_ROI_over_all_canvas.bbox('all')))
```

```
mark_ROI_over_all_canvas.create_window((0,0), \
    window=mark_ROI_scroll_frame, anchor='nw')
mark_ROI_over_all_canvas.configure\
    (xscrollcommand=mark_ROI_xscrollbar.set, \
        yscrollcommand=mark_ROI_yscrollbar.set)

mark_ROI_over_all_frame.config(highlightthickness=0, bg='#ffffff')
mark_ROI_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
mark_ROI_over_all_frame.pack(expand=True, fill=BOTH)
mark_ROI_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
mark_ROI_over_all_frame.grid_columnconfigure(0, weight=1)
mark_ROI_over_all_frame.grid_rowconfigure(0, weight=1)
mark_ROI_xscrollbar.grid(row=1, column=0, sticky=E+W)
mark_ROI_over_all_frame.grid_columnconfigure(1, weight=0)
mark_ROI_over_all_frame.grid_rowconfigure(1, weight=0)
mark_ROI_yscrollbar.grid(row=0, column=1, sticky=N+S)
mark_ROI_over_all_frame.grid_columnconfigure(2, weight=0)
mark_ROI_over_all_frame.grid_rowconfigure(2, weight=0)

mark_ROI_image_canvas = tk.Canvas(mark_ROI_scroll_frame)
mark_ROI_image_canvas.grid(row=0,column=0, rowspan=10, \
    columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
mark_ROI_scroll_frame.grid_columnconfigure(0, weight=0)
mark_ROI_scroll_frame.grid_rowconfigure(0, weight=0)
mark_ROI_image_canvas.create_image(0,0,image=img_mark_ROI,anchor="nw")
mark_ROI_image_canvas.image = img_mark_ROI
mark_ROI_image_canvas.config(bg='#E5f9ff', relief=FLAT, bd=0, \
    scrollregion=mark_ROI_image_canvas.bbox(ALL), \
        height=img_mark_ROI.height(), width=img_mark_ROI.width())
mark_ROI_image_canvas.grid_propagate(0)

rectangle = mark_ROI_image_canvas.create_rectangle(0,0,0,0,outline='green')
rectangle_top_corner = []
rectangle_bottom_corner = []
def buttonPushed(event):
    rectangle_top_corner.append([event.x, event.y])

def buttonMoving(event):
    mark_ROI_image_canvas.coords(rectangle, rectangle_top_corner[0][0], \
        rectangle_top_corner[0][1], event.x, event.y)

def buttonReleased(event):
    rectangle_bottom_corner.append([event.x, event.y])
    mark_ROI_image_canvas.coords(rectangle, rectangle_top_corner[0][0], \
            rectangle_top_corner[0][1],rectangle_bottom_corner[0][0], \
                rectangle_bottom_corner[0][1])
        mark_ROI_image_canvas.itemconfig(rectangle, outline='Blue')

        Globals.DVH_ROI_coords.append([rectangle_top_corner[0][0], \
            rectangle_top_corner[0][1]])
        Globals.DVH_ROI_coords.append([rectangle_bottom_corner[0][0], \
            rectangle_top_corner[0][1]])
        Globals.DVH_ROI_coords.append([rectangle_top_corner[0][0], \
            rectangle_bottom_corner[0][1]])
        Globals.DVH_ROI_coords.append([rectangle_bottom_corner[0][0], \
            rectangle_bottom_corner[0][1]])

        rect = canvas.create_rectangle(int((rectangle_top_corner[0][0])/2), \
            int((rectangle_top_corner[0][1])/2),\
                int((rectangle_bottom_corner[0][0])/2), \
                    int((rectangle_bottom_corner[0][1])/2), outline='Blue', width=2)
        Globals.DVH_mark_ROI_rectangle.append(rect)

        if(ref_point_test):
            Globals.DVH_ROI_reference_point_check = True
            if(Globals.DVH_reference_point_check):
                Globals.DVH_done_button_reference_point.config(state=ACTIVE)
        else:
            Globals.DVH_ROI_check = True
            if(Globals.DVH_isocenter_check):
                Globals.DVH_done_button.config(state=ACTIVE)


        mark_ROI_window.after(500, lambda: mark_ROI_window.destroy())

    mark_ROI_image_canvas.bind("<B1-Motion>", buttonMoving)
    mark_ROI_image_canvas.bind("<Button-1>", buttonPushed)
    mark_ROI_image_canvas.bind("<ButtonRelease-1>", buttonReleased)

def UploadFilm():
#----------------------------------------------------------------
# Function to upload the Film
#
# Is a callback function for button
# DVH_upload_button_film in notebook.py
#----------------------------------------------------------------
    if(Globals.DVH_film_orientation.get() == '-'):
        messagebox.showerror("Missing parameter", "Film orientation missing \n \
(Code: UploadFilm)")
```

```
        return
    if Globals.DVH_film_factor_input.get("1.0", 'end—1c') == " ":
        Globals.DVH_film_factor = 1
    else:
        try:
            Globals.DVH_film_factor = \
                float(Globals.DVH_film_factor_input.get("1.0", 'end—1c'))
        except:
            messagebox.showerror("Missing parameter", "Film factor invalid format.\n\
(Code: UploadFilm)")
            return

    file = filedialog.askopenfilename()
    ext = os.path.splitext(file)[—1].lower()
    if(ext == '.tif'):
        current_folder = os.getcwd()
        parent = os.path.dirname(file)
        os.chdir(parent)
        img = Image.open(file)
        img = img.transpose(Image.FLIP_LEFT_RIGHT)
        cv2Img = cv2.imread(basename(normpath(file)), \
            cv2.IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
        cv2Img = cv2.medianBlur(cv2Img, 5)
        if(cv2Img is None):
            messagebox.showerror("Error", "Something has gone wrong. \
Check that the filename only contain english letters")
            return
        if(cv2Img.shape[2] == 3):
            if(cv2Img.shape[0]==1270 and cv2Img.shape[1]==1016):
                cv2Img = abs(cv2Img—Globals.correctionMatrix127)
                cv2Img = np.clip(cv2Img, 0, 65535)
                cv2Img = cv2.flip(cv2Img,1)
                img_scaled = img.resize((508, 635), Image.ANTIALIAS)
                img_scaled = ImageTk.PhotoImage(image=img_scaled)


                Globals.DVH_film_dataset = cv2Img
                Globals.DVH_film_dataset_red_channel = cv2Img[:,:,2]
            else:
                messagebox.showerror("Error","The resolution of \
the image is not consistent with dpi")
                return
        else:
            messagebox.showerror("Error","The uploaded \
image need to be in RGB—format")
```

```
    return

os.chdir(current_folder)

if(not (img.width == 1016)):
    messagebox.showerror("Error", "Dpi in image has to be 127")
    return

Globals.DVH_film_orientation_menu.configure(state=DISABLED)
Globals.DVH_film_factor_input.config(state=DISABLED)

h = 635 + 20
w = 508 + 625
new_window = tk.Toplevel(Globals.tab5)
new_window.geometry("%dx%d+0+0" % (w, h))
new_window.grab_set()

new_window_over_all_frame = tk.Frame(new_window, bd=0, relief=FLAT)
new_window_over_all_canvas = Canvas(new_window_over_all_frame)

new_window_xscrollbar = Scrollbar(new_window_over_all_frame, \
    orient=HORIZONTAL, command=new_window_over_all_canvas.xview)
new_window_yscrollbar = Scrollbar(new_window_over_all_frame, \
    command=new_window_over_all_canvas.yview)

new_window_scroll_frame = ttk.Frame(new_window_over_all_canvas)
new_window_scroll_frame.bind("<Configure>", lambda e: \
    new_window_over_all_canvas.configure\
        (scrollregion=new_window_over_all_canvas.bbox('all')))

new_window_over_all_canvas.create_window((0,0), \
    window=new_window_scroll_frame, anchor='nw')
new_window_over_all_canvas.configure\
    (xscrollcommand=new_window_xscrollbar.set, \
        yscrollcommand=new_window_yscrollbar.set)

new_window_over_all_frame.config(highlightthickness=0, bg='#ffffff')
new_window_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
new_window_over_all_frame.pack(expand=True, fill=BOTH)
new_window_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
new_window_over_all_frame.grid_columnconfigure(0, weight=1)
new_window_over_all_frame.grid_rowconfigure(0, weight=1)
new_window_xscrollbar.grid(row=1, column=0, sticky=E+W)
new_window_over_all_frame.grid_columnconfigure(1, weight=0)
new_window_over_all_frame.grid_rowconfigure(1, weight=0)
```

```
        new_window_yscrollbar.grid(row=0, column=1, sticky=N+S)
        new_window_over_all_frame.grid_columnconfigure(2, weight=0)
        new_window_over_all_frame.grid_rowconfigure(2, weight=0)

        new_window_explain_text = tk.Text(new_window_scroll_frame,\
             height= 3, width=120)
        new_window_explain_text.insert(INSERT, \
"To match the film with the doseplan you have to mark either isocenter or a \
reference point on the film of your choice.In the case of the reference point \
you \nwill be asked to input the lenght in lateral, longitudinal and vertical \
to a reference point used in the linac. It the reference point in the film is \
the same as \nthe one in the phantom/linac you can input all zeros, in other \
cases your input is in mm. Later you will have the oppertunity to make small\
adjustments \nto the placement of either the reference point or isocenter.")
        new_window_explain_text.config(state=DISABLED, \
            font=('calibri', '13', 'bold'), bg = '#ffffff', relief=FLAT)
        new_window_explain_text.grid(row=0, column=0, columnspan=5, \
            sticky=N+S+W+E, pady=(15,5), padx=(10,10))
        new_window_scroll_frame.grid_rowconfigure(0, weight=0)
        new_window_scroll_frame.grid_columnconfigure(0, weight=0)

        new_window_notebook = ttk.Notebook(new_window_scroll_frame)
        new_window_notebook.borderWidth=0
        new_window_notebook.grid(row=2, column=0, columnspan=5, \
            sticky=E+W+N+S, pady=(0,0), padx =(0,0))
        new_window_scroll_frame.grid_rowconfigure(4, weight=0)
        new_window_scroll_frame.grid_columnconfigure(4, weight=0)

        new_window_isocenter_tab = ttk.Frame(new_window_notebook)
        new_window_notebook.add(new_window_isocenter_tab, text='Isocenter')
        new_window_reference_point_tab = ttk.Frame(new_window_notebook)
        new_window_notebook.add(new_window_reference_point_tab, \
            text='Reference point')
        new_window_manually_tab = ttk.Frame(new_window_notebook)
        new_window_notebook.add(new_window_manually_tab, text='Manually')

        image_canvas = tk.Canvas(new_window_isocenter_tab)
        image_canvas.grid(row=0,column=0, rowspan=12, columnspan=3, \
            sticky=N+S+E+W, padx=(0,0), pady=(0,0))
        new_window_isocenter_tab.grid_rowconfigure(1, weight=0)
        new_window_isocenter_tab.grid_columnconfigure(1, weight=0)
        image_canvas.create_image(0,0,image=img_scaled,anchor="nw")
        image_canvas.image = img_scaled
        image_canvas.config(bg='#ffffff', relief=FLAT, bd=0, \
            scrollregion=image_canvas.bbox(ALL), \
            height=img_scaled.height(), width=img_scaled.width())
        image_canvas.grid_propagate(0)

        image_canvas_reference_tab = tk.Canvas(new_window_reference_point_tab)
        image_canvas_reference_tab.grid(row=0,column=0, rowspan=10, \
            columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
        new_window_reference_point_tab.grid_rowconfigure(1, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(1, weight=0)
        image_canvas_reference_tab.create_image(0,0,image=img_scaled,anchor="nw")
        image_canvas_reference_tab.image = img_scaled
        image_canvas_reference_tab.config(bg='#ffffff', relief=FLAT, bd=0, \
            scrollregion=image_canvas.bbox(ALL), \
                height=img_scaled.height(), width=img_scaled.width())
        image_canvas_reference_tab.grid_propagate(0)

        film_window_mark_isocenter_text = \
            tk.Text(new_window_isocenter_tab, width=55, height=7)
        film_window_mark_isocenter_text.insert(INSERT, \
"When clicking the button \"Mark isocenter\" a window showing \n\
the image will appear and you are to click on the markers \n\
made on the film upon irradiation to find the isocenter. Start \n\
with the marker showing the direction of the film (see the \n\
specifications in main window). When both marks are made \n\
you will see the isocenter in the image. If you are not happy \n\
with the placement click the button again and repeat.")
        film_window_mark_isocenter_text.config(bg='#ffffff', relief=FLAT, bd=0, \
        state=DISABLED, font=('calibri', '11'))
        film_window_mark_isocenter_text.grid(row=0, column=3, rowspan=3, \
            sticky=N+S+E+W, padx=(10,10), pady=(10,0))
        new_window_isocenter_tab.columnconfigure(2, weight=0)
        new_window_isocenter_tab.rowconfigure(2, weight=0)

        film_window_mark_reference_point_text = \
            tk.Text(new_window_reference_point_tab, width=55, height=5)
        film_window_mark_reference_point_text.insert(INSERT, \
"When clicking the button \"Mark point\" a window showing \n\
the image will appear and you are to click on the marker \n\
made on the film upon irradiation to find the point. When\n\
the mark are made you will see the isocenter in the image.\n\
If you are not happy with the placement click the button \n\
again and repeat.")
        film_window_mark_reference_point_text.config(bg='#ffffff', \
            relief=FLAT, bd=0, state=DISABLED, font=('calibri', '11'))
        film_window_mark_reference_point_text.grid(row=0, column=3, \
            rowspan=3, sticky=N+S+E+W, padx=(10,10), pady=(5,0))
        new_window_reference_point_tab.columnconfigure(2, weight=0)
        new_window_reference_point_tab.rowconfigure(2, weight=0)

        mark_isocenter_button_frame = tk.Frame(new_window_isocenter_tab)
        mark_isocenter_button_frame.grid(row=3, column=3, padx=(10,10), pady=(0,10))
        mark_isocenter_button_frame.configure(bg='#ffffff')
        new_window_isocenter_tab.grid_columnconfigure(3, weight=0)
        new_window_isocenter_tab.grid_rowconfigure(3, weight=0)

        mark_isocenter_button = tk.Button(mark_isocenter_button_frame, \
            text='Browse', image=Globals.profiles_mark_isocenter_button_image,\
                cursor='hand2',font=('calibri', '14'), relief=FLAT, state=ACTIVE, \
                    command=lambda: markIsocenter(img, new_window_isocenter_tab, \
                        image_canvas, cv2Img))
        mark_isocenter_button.pack(expand=True, fill=BOTH)
        mark_isocenter_button.config(bg='#ffffff', activebackground='#ffffff', \
            activeforeground='#ffffff', highlightthickness=0)
        mark_isocenter_button.image=Globals.profiles_mark_isocenter_button_image

        mark_point_button_frame = tk.Frame(new_window_reference_point_tab)
        mark_point_button_frame.grid(row=3, column=3, padx=(10,10), pady=(30,0))
        mark_point_button_frame.configure(bg='#ffffff')
        new_window_reference_point_tab.grid_columnconfigure(3, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(3, weight=0)

        mark_point_button = tk.Button(mark_point_button_frame, text='Browse', \
            image=Globals.profiles_mark_point_button_image,\
                cursor='hand2',font=('calibri', '14'), relief=FLAT, \
                    state=ACTIVE, command=lambda: markReferencePoint(img, \
                        new_window_reference_point_tab, image_canvas_reference_tab, \
                            cv2Img))
        mark_point_button.pack(expand=True, fill=BOTH)
        mark_point_button.config(bg='#ffffff', activebackground='#ffffff', \
            activeforeground='#ffffff', highlightthickness=0)
        mark_point_button.image=Globals.profiles_mark_point_button_image

        write_displacement_relative_to_reference_point = \
            tk.Text(new_window_reference_point_tab, width = 55, height=3)
        write_displacement_relative_to_reference_point.insert(INSERT, "\
If the marked reference points in the film does not match\n\
the reference point in the phantom you can write the\n\
displacemnet here (in mm). Defaults to zero ")
        write_displacement_relative_to_reference_point.grid(row=4, column=3, \
            rowspan=2, sticky=N+S+E+W, padx=(10,10), pady=(0,10))
        write_displacement_relative_to_reference_point.config(bg='#ffffff', \
            relief=FLAT, bd=0, state=DISABLED, font=('calibri', '11'))
        new_window_reference_point_tab.grid_rowconfigure(6, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(6, weight=0)

        input_lateral_text = tk.Text(new_window_reference_point_tab, \
            width=12, height=1)
        input_lateral_text.insert(INSERT, "Lateral:")
        input_lateral_text.config(bg='#ffffff', relief=FLAT, bd=0, \
        state=DISABLED, font=('calibri', '10'))
        input_lateral_text.grid(row=5, column=3, sticky=N+S, \
            padx=(0,250), pady=(25,0))
        new_window_reference_point_tab.grid_rowconfigure(10, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(10, weight=0)

        Globals.DVH_input_lateral_displacement = \
            tk.Text(new_window_reference_point_tab, width=5, height=1)
        Globals.DVH_input_lateral_displacement.insert(INSERT, " ")
        Globals.DVH_input_lateral_displacement.config(bg='#E5f9ff', \
            relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
        Globals.DVH_input_lateral_displacement.grid(row=5, column=3, \
            padx=(0,285), pady=(35,0))
        new_window_reference_point_tab.grid_rowconfigure(7, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(7, weight=0)

        input_vertical_text = tk.Text(new_window_reference_point_tab, \
            width=12, height=1)
        input_vertical_text.insert(INSERT, "Vertical:")
        input_vertical_text.config(bg='#ffffff', relief=FLAT, bd=0, \
            state=DISABLED, font=('calibri', '10'))
        input_vertical_text.grid(row=5, column=3, sticky=N+S, \
            padx=(0,0), pady=(25,0))
        new_window_reference_point_tab.grid_rowconfigure(11, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(11, weight=0)

        Globals.DVH_input_vertical_displacement = \
            tk.Text(new_window_reference_point_tab, width=4, height=1)
        Globals.DVH_input_vertical_displacement.insert(INSERT, " ")
        Globals.DVH_input_vertical_displacement.config(bg='#E5f9ff', \
            relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
        Globals.DVH_input_vertical_displacement.grid(row=5, column=3, \
            padx=(0,25), pady=(35,0))
        new_window_reference_point_tab.grid_rowconfigure(8, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(8, weight=0)

        input_long_text = tk.Text(new_window_reference_point_tab, \
            width=12, height=1)
        input_long_text.insert(INSERT, "Longitudinal:")
        input_long_text.config(bg='#ffffff', relief=FLAT, bd=0, \
            state=DISABLED, font=('calibri', '10'))
        input_long_text.grid(row=5, column=3, sticky=N+S, \
            padx=(250,0), pady=(25,0))
        new_window_reference_point_tab.grid_rowconfigure(12, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(12, weight=0)

        Globals.DVH_input_longitudinal_displacement = \
            tk.Text(new_window_reference_point_tab, width=5, height=1)
        Globals.DVH_input_longitudinal_displacement.insert(INSERT, " ")
        Globals.DVH_input_longitudinal_displacement.config\
            (bg='#E5f9ff', relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
        Globals.DVH_input_longitudinal_displacement.grid\
            (row=5, column=3, padx=(240,0), pady=(35,0))
        new_window_reference_point_tab.grid_rowconfigure(9, weight=0)
        new_window_reference_point_tab.grid_columnconfigure(9, weight=0)

        film_window_mark_ROI_text = tk.Text(new_window_isocenter_tab,\
             width=55, height=7)
        film_window_mark_ROI_text.insert(INSERT, \
"When clicking the button \"Mark ROI\" a window showing the\n\
image will appear and you are to drag a rectangle marking \n\
the region of interest. Fidora will assume the film has been\n\
scanned in either portrait or landscape orientation. When\n\
the ROI has been marked it will appear on the image. If you\n\
are not happy with the placement click the button again.")
        film_window_mark_ROI_text.config(bg='#ffffff', relief=FLAT, \
            bd=0, state=DISABLED, font=('calibri', '11'))
        film_window_mark_ROI_text.grid(row=5, column=3, rowspan=4, \
            sticky=N+S+E+W, padx=(10,10), pady=(0,0))
        new_window_isocenter_tab.grid_columnconfigure(4, weight=0)
        new_window_isocenter_tab.grid_rowconfigure(4, weight=0)

        film_window_mark_ROI_reference_point_text = \
            tk.Text(new_window_reference_point_tab, width=55, height=5)
        film_window_mark_ROI_reference_point_text.insert(INSERT, \
"When clicking the button \"Mark ROI\" a window showing the\n\
image will appear and you are to drag a rectangle marking \n\
the region of interest. Fidora will assume the film has been\n\
scanned in either portrait or landscape orientation. When\n\
the ROI has been marked it will appear on the image. If you\n\
are not happy with the placement click the button again.")
        film_window_mark_ROI_reference_point_text.config(bg='#ffffff', \
            relief=FLAT, bd=0, state=DISABLED, font=('calibri', '11'))
        film_window_mark_ROI_reference_point_text.grid(row=6, column=3, \
            rowspan=3, sticky=N+E+W, padx=(10,10), pady=(10,0))
        new_window_reference_point_tab.grid_columnconfigure(4, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(4, weight=0)

        mark_ROI_button_frame = tk.Frame(new_window_isocenter_tab)
        mark_ROI_button_frame.grid(row=8, column=3, padx=(10,0), pady=(0,5))
        mark_ROI_button_frame.configure(bg='#ffffff')
        new_window_isocenter_tab.grid_columnconfigure(5, weight=0)
        new_window_isocenter_tab.grid_rowconfigure(5, weight=0)

        mark_ROI_button = tk.Button(mark_ROI_button_frame, text='Browse', \
            image=Globals.profiles_mark_ROI_button_image,cursor='hand2',\
                font=('calibri', '14'), relief=FLAT, state=ACTIVE, command=lambda:\
                    markROI(img, new_window_isocenter_tab, image_canvas, False))
        mark_ROI_button.pack(expand=True, fill=BOTH)
        mark_ROI_button.config(bg='#ffffff', activebackground='#ffffff', \
            activeforeground='#ffffff', highlightthickness=0)
        mark_ROI_button.image=Globals.profiles_mark_ROI_button_image

        slice_offset_text = tk.Text(new_window_isocenter_tab, width=25, height=1)
        slice_offset_text.insert(INSERT, "Slice offset, mm (default 0):")
        slice_offset_text.config(state=DISABLED, font=('calibri', '10'), \
            bd = 0, relief=FLAT)
        slice_offset_text.grid(row=9, column=3, padx=(5,110), pady=(0,0))
        new_window_isocenter_tab.grid_columnconfigure(6, weight=0)
        new_window_isocenter_tab.grid_rowconfigure(6, weight=0)

        Globals.DVH_slice_offset = tk.Text(new_window_isocenter_tab, \
            width=8, height=1)
        Globals.DVH_slice_offset.grid(row=9, column=3, padx=(110,10), pady=(0,0))
        Globals.DVH_slice_offset.insert(INSERT, " ")
        Globals.DVH_slice_offset.config(state=NORMAL, \
            font=('calibri', '10'), bd = 2, bg='#ffffff')
        new_window_isocenter_tab.grid_columnconfigure(7, weight=0)
        new_window_isocenter_tab.grid_rowconfigure(7, weight=0)

        mark_ROI_button_reference_point_frame = \
            tk.Frame(new_window_reference_point_tab)
        mark_ROI_button_reference_point_frame.grid(row=9, column=3, \
            padx=(10,10), pady=(0,5))
        mark_ROI_button_reference_point_frame.configure(bg='#ffffff')
        new_window_reference_point_tab.grid_columnconfigure(5, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(5, weight=0)
```

```
        mark_ROI_reference_point_button = \
            tk.Button(mark_ROI_button_reference_point_frame, text='Browse', \
                image=Globals.profiles_mark_ROI_button_image,cursor='hand2',\
                    font=('calibri', '14'), relief=FLAT, state=ACTIVE, \
                        command=lambda: markROI(img, \
                            new_window_reference_point_tab, \
                                image_canvas_reference_tab, True))
        mark_ROI_reference_point_button.pack(expand=True, fill=BOTH)
        mark_ROI_reference_point_button.config(bg='#ffffff', \
            activebackground='#ffffff', activeforeground='#ffffff', \
            highlightthickness=0)
        mark_ROI_reference_point_button.image=\
            Globals.profiles_mark_ROI_button_image

        def finishFilmMarkers(ref_test):
            Globals.DVH_slice_offset.config(state=DISABLED)
            if(ref_test):
                if(not(Globals.DVH_input_lateral_displacement.get\
                    ("1.0",'end-1c')==" ")):
                    try:
                        test = float(Globals.DVH_input_lateral_displacement.get\
                            ("1.0",'end-1c'))
                        Globals.DVH_lateral = test
                    except:
                        messagebox.showerror("Error", "The displacements \
must be numbers\n (Code: lateral displacement)")
                        return
                else:
                    Globals.DVH_lateral = 0
                if(not(Globals.DVH_input_longitudinal_displacement.get\
                    ("1.0",'end-1c')==" ")):
                    try:
                        test = float(Globals.DVH_input_longitudinal_displacement.get\
                            ("1.0", 'end-1c'))
                        Globals.DVH_longitudinal = test
                    except:
                        messagebox.showerror("Error", "The displacements must \
be numbers\n (Code: longitudinal displacement)")
                        return
                else:
                    Globals.DVH_longitudinal = 0
                if(not(Globals.DVH_input_vertical_displacement.get\
                    ("1.0",'end-1c')==" ")):
                    try:
                        test = float(Globals.DVH_input_vertical_displacement.get\
                            ("1.0", 'end-1c'))
                        Globals.DVH_vertical = test
                    except:
                        messagebox.showerror("Error", "The displacements \
must be numbers\n (Code: vertical displacement)")
                        return
                else:
                    Globals.DVH_vertical = 0
                Globals.DVH_input_vertical_displacement.config(state=DISABLED)
                Globals.DVH_input_longitudinal_displacement.config(state=DISABLED)
                Globals.DVH_input_lateral_displacement.config(state=DISABLED)
            else:
                if not Globals.DVH_slice_offset.get("1.0",'end-1c')==" ":
                    try:
                        offset = float(Globals.DVH_slice_offset.get("1.0",'end-1c'))
                        Globals.DVH_offset = offset
                    except:
                        messagebox.showerror("Error", "Slice offset must \
be a number \n(Code: finishFilmMarkers(false)")
                        return
                else:
                    Globals.DVH_offset = 0
            if(ref_test):
                choose_batch_window = tk.Toplevel(new_window_reference_point_tab)
            else:
                choose_batch_window = tk.Toplevel(new_window_isocenter_tab)

            choose_batch_window.geometry("670x380+50+50")
            choose_batch_window.grab_set()

            choose_batch_frame = tk.Frame(choose_batch_window)
            choose_batch_frame.pack(expand=True, fill=BOTH)
            choose_batch_frame.configure(bg='#ffffff')

            batch_cnt = 0
            weight_cnt = 0
            read = open('calibration.txt', 'r')
            lines = read.readlines()
            read.close()
            row_cnt=0
            for l in lines:
                words = l.split()
                line = "Batch nr. : " + words[2] + ".    Date:   " + words[0] +\
                     "  " + words[1] + "."
```

```
write_batch_nr = tk.Text(choose_batch_frame, width=10, height=1)
write_batch_nr.grid(row=row_cnt, column=0, sticky=N+S+W+E, \
    padx=(10,5), pady=(10,10))
choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
write_batch_nr.insert(INSERT, "Batch nr.: ")
write_batch_nr.config(state=DISABLED, bd = 0, \
    font=('calibri', '12', 'bold'))
weight_cnt+=1
write_batch = tk.Text(choose_batch_frame, width=20, height=1)
write_batch.grid(row=row_cnt, column=1, sticky=N+S+W+E, \
    padx=(10,5), pady=(10,10))
choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
write_batch.insert(INSERT, words[2])
write_batch.config(state=DISABLED, bd = 0, font=('calibri', '12'))
weight_cnt+=1
write_batch_date = tk.Text(choose_batch_frame, width=8, height=1)
write_batch_date.grid(row=row_cnt, column=2, sticky=N+S+W+E, \
    padx=(10,5), pady=(10,10))
choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
write_batch_date.insert(INSERT, "Date: ")
write_batch_date.config(state=DISABLED, bd = 0, \
    font=('calibri', '12', 'bold'))
weight_cnt+=1
write_date = tk.Text(choose_batch_frame, width=30, height=1)
write_date.grid(row=row_cnt, column=3, sticky=N+S+W+E, \
    padx=(10,5), pady=(10,10))
choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
write_date.insert(INSERT, words[0] + ", " + words[1] + "")
write_date.config(state=DISABLED, bd = 0, \
    font=('calibri', '12'))
weight_cnt+=1

Radiobutton(choose_batch_frame, text='',bg='#ffffff', \
    cursor='hand2',font=('calibri', '14'), \
        variable=Globals.DVH_film_batch, value=batch_cnt).\
            grid(row=row_cnt, column=4, sticky=N+S+W+E, \
                padx=(5,5), pady=(10,10))
choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
weight_cnt+=1;row_cnt+=1;batch_cnt+=1
```

```
def set_batch():
    choose_batch_window.destroy()
    f = open('calibration.txt', 'r')
    lines = f.readlines()
    words = lines[Globals.DVH_film_batch.get()].split()
    Globals.DVH_popt_red[0] = float(words[3])
    Globals.DVH_popt_red[1] = float(words[4])
    Globals.DVH_popt_red[2] = float(words[5])
    f.close()

    Globals.DVH_film_dataset_ROI_red_channel_dose = \
        np.zeros((Globals.DVH_film_dataset_ROI_red_channel.shape[0],\
        Globals.DVH_film_dataset_ROI_red_channel.shape[1]))
    for i in range\
        (Globals.DVH_film_dataset_ROI_red_channel_dose.shape[0]):
        for j in range\
            (Globals.DVH_film_dataset_ROI_red_channel_dose.shape[1]):
            Globals.DVH_film_dataset_ROI_red_channel_dose[i,j] = \
                Globals.DVH_film_factor*pixel_to_dose\
                    (Globals.DVH_film_dataset_ROI_red_channel[i,j], \
                        Globals.DVH_popt_red[0], \
                            Globals.DVH_popt_red[1], \
                                Globals.DVH_popt_red[2])

    Globals.DVH_film_dataset_red_channel_dose = \
        np.zeros((Globals.DVH_film_dataset_red_channel.shape[0],\
        Globals.DVH_film_dataset_red_channel.shape[1]))
    for i in range\
        (Globals.DVH_film_dataset_red_channel_dose.shape[0]):
        for j in range\
            (Globals.DVH_film_dataset_red_channel_dose.shape[1]):
            Globals.DVH_film_dataset_red_channel_dose[i,j] = \
                Globals.DVH_film_factor*pixel_to_dose\
                    (Globals.DVH_film_dataset_red_channel[i,j], \
                        Globals.DVH_popt_red[0], \
                            Globals.DVH_popt_red[1], \
                                Globals.DVH_popt_red[2])



    mx_film=np.max(Globals.DVH_film_dataset_ROI_red_channel_dose)
    Globals.DVH_max_dose_film = mx_film

    new_window.destroy()

set_batch_button_frame = tk.Frame(choose_batch_frame)
```

```
set_batch_button_frame.grid(row=row_cnt, column=1, columnspan=3, \
    padx=(10,0), pady=(5,5))
set_batch_button_frame.configure(bg='#ffffff')
choose_batch_frame.grid_columnconfigure(weight_cnt, weight=0)
choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)

set_batch_button = tk.Button(set_batch_button_frame, text='OK', \
    image=Globals.done_button_image, cursor='hand2',\
        font=('calibri', '14'), relief=FLAT, state=ACTIVE, \
            command=set_batch)
set_batch_button.pack(expand=True, fill=BOTH)
set_batch_button.image=Globals.done_button_image


img_ROI = Globals.DVH_film_dataset[Globals.DVH_ROI_coords[0][1]:\
    Globals.DVH_ROI_coords[2][1],Globals.DVH_ROI_coords[0][0]:\
        Globals.DVH_ROI_coords[1][0], :]
img_ROI_red_channel = img_ROI[:,:,2]
Globals.DVH_film_variable_ROI_coords = \
    [Globals.DVH_ROI_coords[0][1], Globals.DVH_ROI_coords[2][1],\
        Globals.DVH_ROI_coords[0][0], Globals.DVH_ROI_coords[1][0]]
Globals.DVH_film_dataset_ROI = img_ROI
Globals.DVH_film_dataset_ROI_red_channel = img_ROI_red_channel
R = img_ROI[:,:,2];B = img_ROI[:,:,0]; G = img_ROI[:,:,1]
img_ROI_RGB = np.zeros(img_ROI.shape)
img_ROI_RGB[:,:,0]=R; img_ROI_RGB[:,:,1]=G; img_ROI_RGB[:,:,2]=B


Globals.DVH_upload_button_doseplan.config(state=DISABLED)
Globals.DVH_upload_button_rtplan.config(state=ACTIVE)
Globals.DVH_upload_button_film.config(state=DISABLED)

if(ref_test):
    Globals.DVH_distance_reference_point_ROI.append\
        ([(Globals.DVH_film_reference_point[0]-\
            Globals.DVH_ROI_coords[0][0])*0.2, \
            (Globals.DVH_film_reference_point[1] -\
                Globals.DVH_ROI_coords[0][1])*0.2])
    Globals.DVH_distance_reference_point_ROI.append\
        ([(Globals.DVH_film_reference_point[0] - \
            Globals.DVH_ROI_coords[1][0])*0.2,\
                (Globals.DVH_film_reference_point[1] - \
                    Globals.DVH_ROI_coords[1][1])*0.2])
    Globals.DVH_distance_reference_point_ROI.append\
        ([(Globals.DVH_film_reference_point[0] - \
                    Globals.DVH_ROI_coords[2][0])*0.2,\
                        (Globals.DVH_film_reference_point[1] - \
                            Globals.DVH_ROI_coords[2][1])*0.2])
            Globals.DVH_distance_reference_point_ROI.append\
                ([(Globals.DVH_film_reference_point[0] - \
                    Globals.DVH_ROI_coords[3][0])*0.2,\
                        (Globals.DVH_film_reference_point[1] - \
                            Globals.DVH_ROI_coords[3][1])*0.2])

            Globals.DVH_isocenter_or_reference_point = "Ref_point"
        else:
            Globals.DVH_distance_isocenter_ROI.append\
                ([(Globals.DVH_film_isocenter[0]-\
                    Globals.DVH_ROI_coords[0][0])*0.2, \
                        (Globals.DVH_film_isocenter[1] -\
                            Globals.DVH_ROI_coords[0][1])*0.2])
            Globals.DVH_distance_isocenter_ROI.append\
                ([(Globals.DVH_film_isocenter[0] - \
                    Globals.DVH_ROI_coords[1][0])*0.2,\
                        (Globals.DVH_film_isocenter[1] - \
                            Globals.DVH_ROI_coords[1][1])*0.2])
            Globals.DVH_distance_isocenter_ROI.append\
                ([(Globals.DVH_film_isocenter[0] - \
                    Globals.DVH_ROI_coords[2][0])*0.2,\
                        (Globals.DVH_film_isocenter[1] - \
                            Globals.DVH_ROI_coords[2][1])*0.2])
            Globals.DVH_distance_isocenter_ROI.append\
                ([(Globals.DVH_film_isocenter[0] - \
                    Globals.DVH_ROI_coords[3][0])*0.2,\
                        (Globals.DVH_film_isocenter[1] - \
                            Globals.DVH_ROI_coords[3][1])*0.2])

            Globals.DVH_isocenter_or_reference_point = "Isocenter"


    done_button_frame = tk.Frame(new_window_isocenter_tab)
    done_button_frame.grid(row=10, column=3, padx=(10,10), \
        pady=(5,5), sticky=N+S+W+E)
    done_button_frame.configure(bg='#ffffff')
    new_window_isocenter_tab.grid_columnconfigure(5, weight=0)
    new_window_isocenter_tab.grid_rowconfigure(5, weight=0)

    Globals.DVH_done_button = tk.Button(done_button_frame, text='Done', \
        image=Globals.done_button_image,cursor='hand2', font=('calibri', '14'),\
            relief=FLAT, state=DISABLED, command=lambda: \
                finishFilmMarkers(False))
        Globals.DVH_done_button.pack(expand=True, fill=BOTH)
        Globals.DVH_done_button.config(bg='#ffffff', activebackground='#ffffff', \
            activeforeground='#ffffff', highlightthickness=0)
        Globals.DVH_done_button.image=Globals.done_button_image

        done_button_reference_point_frame = \
            tk.Frame(new_window_reference_point_tab)
        done_button_reference_point_frame.grid(row=10, column=3, \
            padx=(10,10), pady=(5,5), sticky=N+S+W+E)
        done_button_reference_point_frame.configure(bg='#ffffff')
        new_window_reference_point_tab.grid_columnconfigure(5, weight=0)
        new_window_reference_point_tab.grid_rowconfigure(5, weight=0)

        Globals.DVH_done_button_reference_point= \
            tk.Button(done_button_reference_point_frame, text='Done', \
                image=Globals.done_button_image,cursor='hand2', \
                    font=('calibri', '14'), relief=FLAT, state=DISABLED, \
                        command=lambda: finishFilmMarkers(True))
        Globals.DVH_done_button_reference_point.pack(expand=True, fill=BOTH)
        Globals.DVH_done_button_reference_point.config(bg='#ffffff', \
            activebackground='#ffffff', activeforeground='#ffffff', \
                highlightthickness=0)
        Globals.DVH_done_button_reference_point.image=Globals.done_button_image


    elif(ext==""):
        return
    else:
        messagebox.showerror("Error", "The file must be a *.tif file")


def help_showPlanes():
    new_window = tk.Toplevel(Globals.tab5)
    w = Globals.profiles_showPlanes_image.width()
    h = Globals.profiles_showPlanes_image.height()
    new_window.geometry("%dx%d+0+0" % (w, h))
    new_window.grab_set()

    canvas = tk.Canvas(new_window)
    canvas.config(relief=FLAT, bg='#ffffff', highlightthickness=0)
    canvas.create_image(0, 0, image=Globals.profiles_showPlanes_image, anchor='nw')
    canvas.pack(expand=True, fill=BOTH)
```