NTNU
Norwegian University of
Science and Technology

Ane Vigre Håland

2020

Master's thesis

Master's thesis

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

Ane Vigre Håland

Designing an analysis tool for film based dosimetry and applications

Evaluation of breast cancer treatment plans in external radiotherapy, using GafChromic EBT3 film and a flat-bed scanner

July 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Designing an analysis tool for film based dosimetry and applications

Evaluation of breast cancer treatment plans in external radiotherapy, using GafChromic EBT3 film and a flat-bed scanner

Ane Vigre Håland

Applied Physics and Mathematics
Submission date: July 2020
Supervisor: Jomar Frengen
Co-supervisor: Signe Danielsen

Norwegian University of Science and Technology
Department of Physics

Preface

This project was carried out at St. Olavs Hospital during the spring of 2020 under supervision by medical physicist Jomar Frengen, and head of Department, education and research, at the cancer clinic and associate at the Department of Physics Signe Danielsen. Due to the COVID-19 outbreak, the time spent in the cancer clinic was significantly reduced as an infection control measure. Instead, the focus was shifted towards making an analysis tool in Python, which could be done from home.

Thanks to St. Olavs Hospital for supplying necessary equipment, and to Signe Danielsen for good feedback on the content of the rapport. A large thanks goes to Jomar Frengen for help to perform the experiments, giving a motivational insight into the field of radiation therapy and good feedback in writing the rapport. Lastly, a thank you goes to my friends and family, as well as my boyfriend, Bjørn, for providing a lot of emotional support.

Abstract

This study has shown that the nonuniformity effect along the detector array in radiochromic film dosimetry using a CCD-based flat-bed scanner, can be properly corrected for using one absolute correction matrix independent of dose level, as shown for GafChromic EBT3 film. However, more investigations towards characterising the GafChromic XRQA2 must be done before it can be used in the clinic. Especially the energy dependence of the XRQA2 film should be studied in more detail. A Python program named FIDORA was developed to perform various analysis associated with film dosimetry, using GafChromic film and an Epson v750 Pro flat-bed scanner. FIDORA performs a correction of the nonuniform read-out of the scanner and corrects for all three color channels in landscape mode. FIDORA provides the opportunity to establish calibration curves based on films irradiated to reference doses, and can accept multiple images irradiated at the same reference dose, and use the average in order to reduce the influence of the scan-to-scan variation. Other functionalities offered by FIDORA is to map the dose in a scanned image, as well as evaluation of profiles for a given region of interest, using a calibration of choice. The profiles functionality in FIDORA enables the comparison of a film measurement with the dose plan matrix that is calculated in RayStation 8B, the treatment planning system used in St. Olavs Hospital. Based on the investigation of several treatment plans, FIDORA poses as a good film-based dosimetry tool and can be applied to various regions where one is interested in validating the calculated dose in the treatment planning system.

FIDORA was applied to investigate the build-up dose to the target breast, as well as the dose to the contralateral breast (CLB). The build-up distance in the target breast, measured from the entrance dose to 90% of the target dose, resulted in a slightly asymmetrical film measure of the medial and lateral segment of the breast for all treatment plans, yielding less lateral skin sparing. The dose from 15 fractions in the CLB, measured with GafChromic EBT3 film, resulted in an allover higher measured dose in the 90° collimator angle plans than what was calculated in the dose plan, with the only exception being a very high entrance dose observed in the dose plan at medial incidence. An evaluated treatment plan employing a 0° collimator angle demonstrated an all over better correspondence between the calculated and measured dose than what was seen in the 90° collimator angle plans, but also showed a very high entrance dose in the dose plan at medial incidence that was not found in the film measurement. These findings might indicate that the linear accelerator model in RayStation is not as reliable outside the fields limited by the (lower) jaws. Evaluating the treatment plans investigated in this project, the potential reduction in dose to the CLB using a collimator angle of 90° demonstrated little sparing effect to the CLB. Instead, a sparing effect to the CLB was found through the use of a filter-free VMAT treatment plan. This plan offered at worst a 38% reduction in dose to the center of the CLB compared to a tangential field-in-field plan.

Sammendrag

Filmdosimetri ved bruk av radiokromisk EBT3 film og en transmisjonsskanner av type flat-bed CCD viser en ikke-uniform skanner-avlesning, som kan korrigeres ved hjelp av en absolutt korreksjonsmatrise som er uavhengig av dosenivå. Det ble også forsøkt å bruke radiokromisk XR-QA2 film, men denne må undersøkes mer før den kan tas i bruk på klinikken. Spesielt energiavhengigheten til radiokromisk XR-QA2 film må undersøkes nærmere. Et Python-program, døpt FIDORA, ble utviklet for å fungere som et filmdosimetri-verktøy, sammen med radiokromisk film og en Epson v750 Pro skanner av type flat-bed. FIDORA utfører en korreksjon av den ikke-uniforme skanner-avlesningen og korrigerer for alle tre fargekanaler i landskapsmodus. FIDORA gir muligheten til å etablere kalibreringskurver basert på filmer som er bestrålt med en referansedose, og aksepterer flere skannede bilder av den samme referansedosen. En kan dermed bruke den gjennomsnittlige skanner-avlesningen for å minimere effekten av skann-til-skann variasjoner. Andre funksjonaliteter som finnes i FIDORA er muligheten til å kartlegge dosen i en skannet film og evaluere profiler for et gitt område, ved bruk av en valgfri kalibreringskurve som er lagret i FIDORA. Profil-funksjonaliteten i FIDORA muliggjør en sammenligning av filmmålinger og doseplan-beregningene gjort i RayStation 8B, som er behandlingsplanleggingssystemet brukt ved St. Olavs Hospital. Basert på behandlingsplanene som ble undersøkt i dette prosjektet viser FIDORA seg å være et pålitelig filmdosimetri-verktøy som kan anvendes i undersøkelsen av områder hvor en ønsker å validere den beregnede dosen i behandlingsplanleggingssystemet.

FIDORA ble anvendt til å undersøke oppbyggingsdosen i målbrystet og dosen til motsatt bryst. Oppbyggingsdistansen i målbrystet ble målt fra inngangsdosen ved hudens overflate, til dosen nådde 90% av måldosen. Dette resulterte i en asymmetrisk oppbyggingsdistanse ved det mediale og laterale segmentet av målbrystet, noe som potensielt kan resultere i mer hudskade i det laterale segmentet av brystet. Dosen fra 15 fraksjoner til motsatt bryst ble målt ved radiokromisk EBT3 film, og resulterte i en jevnt over høyere dose enn det som ble beregnet i doseplanene med 90° kollimatorvinkel, med unntak av en veldig høy inngangsdose som ble observert ved det mediale segmentet i doseplanen og ikke i filmmålingene. Denne forskjellen mellom målt of beregnet dose ble imidlertid ikke funnet for en behandlingsplan med 0° kollimatorvinkel. Denne planen viste en langt bedre overensstemmelse mellom de målte og beregnede dosene, med unntak av den høye inngangsdosen som ble observert ved det mediale segmentet i doseplanen. Dette indikerer muligens at lineærakselerator-modellen som brukes i RayStation ikke er like pålitelig utenfor feltet som er definert av Y-blenderne. Behandlingsplanene i dette prosjektet som brukte en kollimatorvinkel lik 90° hadde et potensiale til å redusere dosen som ble gitt til det motsatte brystet. Denne besparende effekten viste seg dog å være mindre enn antatt. Istedenfor ble det observert en betydelig besparende effekt i dosen som ble gitt til motsatt bryst i en filter-fri VMAT behandlingsplan. Denne planen viste seg å redusere dosen til motsatt bryst ved minst 38%, sammenlignet med en tangentiell field-in-field plan.

Table of Contents

List of Figures

Abbreviations

Symbol	=	**definition**
CLB	=	Contralateral breast
LINAC	=	Linear accelerator
MU	=	Monitor units
ROI	=	Region of interest
MLC	=	Multi leaf collimator
EBT	=	External beam therapy
VMAT	=	Volumetric modulated arc therapy
IMRT	=	Intensity modulated radiotherapy
GUI	=	Graphical user interface
CCD	=	Charged coupled device
RGB	=	Red, green and blue (used as color channels)
PV	=	Pixel value
OD	=	Optical density

Chapter 1

Introduction

1.1 External radiotherapy in cancer treatment

External radiotherapy is a common treatment modality and is performed by a linear accelerator. The external radiation penetrates the patient's tissue and is focused at the tumor volume. As the radiation interacts with the surface of the patient, the skin, and goes further into the tissue, a dose is deposited along the entire radiation path. So even if the tumor is situated 5 cm into the breast tissue, all the tissue along the radiation path will be subject to an energy transfer and will absorb some dose. So, the natural side effect of every external radiotherapy is that healthy tissue surrounding or in proximity to the actual tumor volume is irradiated simultaneously. Consequently, every treatment course is a compromise, wanting to kill every tumor cell, but at the same time spare as much healthy tissue as possible. Since the goal of radiotherapy is to damage and eventually kill cells, healthy tissue exposed to radiation will also experience unwanted reactions which can manifest itself early (during the treatment course) or several years later.

Breast cancer is worldwide the leading cancer among women [25]. External radiotherapy used to treat early-stage breast cancer may cause severe side-effects to surrounding healthy tissue, including skin, heart, lung and contralateral breast (CLB). So, if the primary aim of the treatment, to kill all the cancer cells, is not affected, these side-effects should be minimized [1]. Yet, to model the dose to the CLB and to the skin in the target breast can be difficult with the existing models available in the treatment planning system. The absorbed dose to the CLB is accumulated due to exposure from several field arrangements and multiple dose deliveries. The dose to the CLB is not large compared to the dose given to the target breast, but it is still worth investigating more, as a significant number of breast cancer patients are diagnosed with secondary cancer in the CLB sometime after their primary treatment [25]. The absorbed dose to the skin in the target breast is also difficult to calculate correctly, due to this being an area where the existing models endeavors to calculate the dose.

1.2 Radiochromic film

The complexity in today's treatment planning, using complicated algorithms to simulate radiation interactions with tissue which enables a conform dose distribution with reduced margins, requires a verification of that the deposited dose is in accordance with the prescribed dose. In radiotherapy in Norway today most clinics use diodes or ion chambers when performing quality controls of their treatment planning system. Although this is sufficient for most of the clinical applications, newer methods and more accurate dose delivery has made it increasingly important to study smaller radiation fields with small and steep changes in dose. For this application, the existing quality control devices are not able to give a continuous readout and can therefore not give a good representation of such complex details. An alternative to the quality control devices or detectors mentioned above is radiochromic film, which becomes dyed in response to radiation. Unlike diodes and ion chambers, radiochromic film can provide a continuous 2D representation of a dose distribution, and can easily be placed inside phantoms at different depths and with different densities. Thus, radiochromic film can describe small fields and complex details that is inaccessible with other detectors due to the size of these detectors limiting the spatial resolution needed to measure high dose gradients. To digitize the information in the radiated film, it must be scanned in a flat-bed scanner in transmission mode. Then, the dose can be calculated from the change in optical density in the film.

Radiochromic film EBT was released to the market in the beginning of 2000. This film was characterized and accepted as a good tool in quality control. Unfortunately, the second generation of the film, EBT2, proved to give a poor representation of the dose distribution [12]. As a consequence, the method of using radiochromic film has not been much used. In 2011 the newest generation, EBT3, was released, which has shown a higher potential than the second. Earlier studies of EBT3 measures using a flat-bed scanner have shown a need for correction due to an imperfect system. A correction method was developed during a project [16] the autumn of 2019 and is based on absolute subtraction of a correction matrix that is valid in the whole clinical dose range.

1.3 Aim of project

In this project the GafChromic EBT3 film will be used for dosimetric verification of the treatment planning system RayStation 8B. The first part of the project is to develop a Python program that can be employed as a film analysis tool, so that the dosimetric information in the film becomes accessible. The second part of this project is to apply this program to investigate the quality of the treatment planning system, by comparing calculated dose distributions with those measured with the film for different treatment techniques that are being evaluated for the treatment of breast cancer. The parameters that will be studied for the evaluation of the breast cancer treatment techniques are the build-up dose to the target breast and the dose to the CLB.

Chapter 2

Background

2.1 Physical principals of radiation therapy

Radiobiology is the study of the action and effect of ionizing radiation on living organisms [17]. This topic is fundamental in understanding how high-energy photons, which will be the radiation type used in this project, interacts with human tissue. When radiation deposits energy in biological tissue, there are distinct mechanisms that are relevant to distinguish between: excitation and ionization. An excitation is the rising of an orbital electron to a higher energy level, and without ejection of an electron. Ionization on the other hand is the ejection of orbital electrons from an atom or a molecule, and in general requires more energy than excitations [17].

2.1.1 Types of ionizing radiation

There are two main types of ionizing radiation, electromagnetic and particulate, that are relevant to distinguish between. Electromagnetic ionizing radiation include both x-rays and γ-rays, and are waves that carry electromagnetic radiant energy. These are commonly referred to as photons, and are characterized by having zero rest mass and that they carry no charge. The distinction between different types of photons refers to where they are created. X-rays are produced extranuclearly. In practice often by an electrical device that accelerates charged particles towards a target, where some of the incoming kinetic energy is converted to X-rays. γ-rays are produced intranuclearly, and that means that they are emitted by radioactive isotopes in a decay process. [17].

Due to photons carrying no charge, electromagnetic radiation is not directly ionizing. What is actually causing chemical or biological damage are not the photons themselves, but secondary charged particles such as electrons, released upon the ionization. Electrons can be produced when photons interact with matter, and they can interact directly with the absorbing material and produce damage [17]. Electrons, as well as protons, α-particles and other heavy charged ions are examples of particulate radiation. They are all used in radiotherapy or more specialized facilities. However, the most relevant particle in radiotherapy is the electron. The electrons are accelerated in a linear accelerator to a desired energy, and can either be used directly, or will be sent to collide with a heavy metal target, to produce photons [17].

2.1.2 Biological effects of radiation

The overall goal of radiotherapy is to damage and eventually kill tumor cells, while sparing normal tissue. The critical target in any cell is the the DNA of the cell, and so the mechanism of cell killing is primarily to produce damages (lesions) to the DNA by breaking chemical bonds [17]. Many of the lesions that are produced by the radiation in DNA are repaired by the cell's own reparation mechanisms.

A single DNA strand break has little biological effect as the cell can repair itself by using the complementary DNA strand as a template. A double strand break, that is a break in each of the two strands in close proximity of each other, may lead to a lethal damage. That is because when the double strand break cleaves the DNA helix, the repair that follow is less likely to succeed. So the outcome is a smaller chance of repair for a double strand DNA break compared to a single strand DNA break.

It is common to distinguish between direct and indirect action of radiation, and these effects are illustrated in Figure 2.1. Direct action of radiation comes from secondary charged particles that are liberated from incoming radiation. Most often these secondary charged particles are electrons, which can interact directly with the DNA. Indirect action of radiation occurs when the liberated, charged, particles interact with other atoms or molecules in the cell, especially water molecules, to produce free radicals. Free radicals are highly chemically reactive molecules, due to their unpaired orbital electron that can damage the DNA [17].

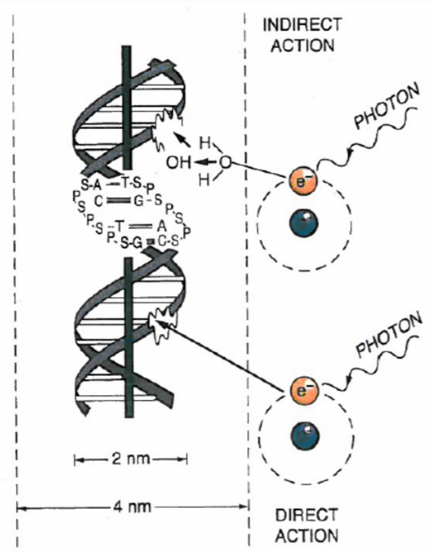


Figure 2.1: Direct and indirect actions of radiation, illustrated with an ejected orbital electron and free radicals. The DNA helix consists of two strands of nucleotides, which are composed of a nitrogenous base, a sugar group and a phosphate group. Nucleotides are linked to their neighboring nucleotides by covalent bonds between a phosphate and a sugar group, whilst the nitrogenous bases are associated to each other through hydrogen-bonds. A double strand break will cleave the DNA helix, and is considered the most important lesion produced by radiation. Courtesy of [17].

2.2 Interaction of radiation with matter

As mentioned earlier photons do not have electrical charge nor mass, and can therefore not directly ionize (and thereby damage) matter. Photons interact and release electrons, through three separate interaction mechanisms with matter. The probability of interaction for the three different interaction mechanisms depends both on the photon energy and the atomic composition of the material the photon is interacting with. The general rule (for low atomic numbers, Z) is that low-energy photons may be absorbed by an absorbing material (photoelectric effect) followed by characteristic X-ray emission. Mid-energy photons may scatter and loose energy through collisions with atomic electrons (compton effect), while high energy photons in the proximity of a nucleus may interact and be transformed into an electron pair, that is a positron and an electron (pair production) [17].

Photoelectric effect: At low photon energies the photoelectric effect is the most likely reaction to occur. A photon close to an atom or molecule is absorbed with the material, and an electron from the innermost orbital is ejected. This is followed by a relaxation of an electron from an outer orbital, filling the vacancy in the inner orbital, and thus production of a characteristic X-ray. However, the energy of the incoming photon must exceed a threshold energy to cause ionization, equal to the binding energy of the particular electron of the absorbing material, for the reaction to occur. The ejected electron obtains the remaining energy, E_e, that the photon carried

$$E_e = h\nu - E_b \tag{2.1}$$

Here $h\nu$ is the incident photon energy, and E_b is the binding energy of the electron in its respective orbital [17].

Compton scattering: The scattering of photons from interaction with atomic electrons in the outermost orbital, that usually results in a photon with reduced energy. This occurs as some of the incidents photon energy is transferred to the recoiling electron, giving rise to a secondary electron. These secondary electrons are important in radiotherapy, as they are directly responsible for interacting with matter. The kinetic energy of the recoiling electron, K, depends on the scattering angle of the photon as well as the incident photon energy.

$$K = E_\gamma - E'_\gamma = E - mc^2, \tag{2.2}$$

where E_γ and E'_γ is the incident and resultant energy of the photon, respectively. E is the total enegy of the recoil electron including its rest mass energy mc^2 [17].

Pair production: The creation of an electron and a positron, occurring at high photon energies. For pair production to occur, the incident energy of the photon must exceed the energy of the rest mass of two electrons. That is, $h\nu \geqslant 2mc^2$, for the reaction to be possible. If the incident photon energy exceeds the rest mass of two electrons, the remaining energy is converted to electron and positron kinetic energy. Also for this reaction to be possible, both energy and momentum must be conserved, requiring an electrical field to be present, usually from a nucleus. Thus, a pair production reaction cannot occur in free space. The reaction can be expressed as:

$$K_- + K_+ = E_\gamma - 2mc^2 \tag{2.3}$$

where K_- and K_+ is the kinetic energy of the electron and positron, respectively. E_γ is the energy of the incident photon, and $2mc^2$ is the rest mass of the electron and proton [17].

Which of the three reactions are most likely to occur depends on the incident photon energy as well as the atomic number of the absorber material. An overview of the different reactions can be seen in Figure 2.2.

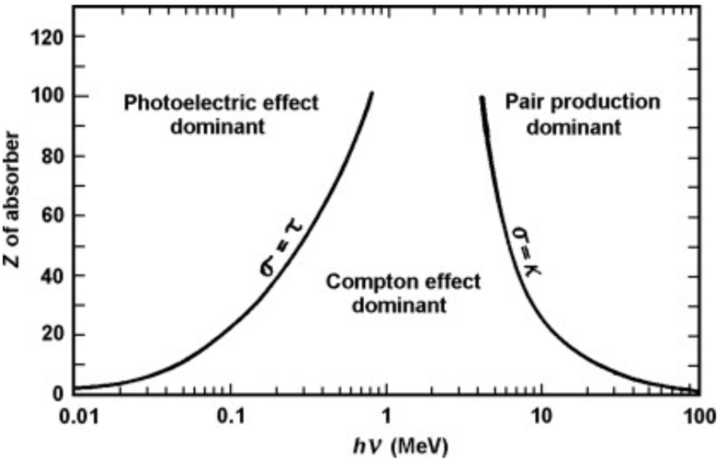


Figure 2.2: The relative importance of various processes of photons interaction with matter. Which process dominates depends on the absorbing material Z, and the photon energy $h\nu$. At the solid lines the cross-sections of two processes are equal. Here σ, τ and κ represents the cross-section of the Compton effect, photoelectric effect and pair production, respectively. Courtesy of [10].

The cross-section is a quantity that expresses the likelihood of an interaction between two particles. It is defined as the area transverse to the particles relative motion where they can meet and interact with each other [17]. Figure 2.2 indicates which interaction mechanism is more likely at a certain energy and absorber material. In external radiotherapy it is common to operate at photon energies between 0.1 and 20 MeV [10], and the effective atomic number of different human tissue typically is lower than 10 [35], resulting in the Compton effect being the most dominant interaction mechanism for photons used in radiotherapy. As a consequence the dose is delivered by atomic electrons, which are set in motion in the Compton scattering process, and not by the primary photons directly.

Another effect worth mentioning when dealing with electrons, is bremsstrahlung. Bremsstrahlung refers to the production of radiation produced from the deceleration of a charged particle, often an electron. When secondary electrons are produced, they can also give away energy and produce photons through Bremsstrahlung, in addition to depositing dose into the tissue[10]. This is the reason why, not all of the photon energy is deposited locally.

2.2.1 5 R's of radiobiology and fractionation

Only considering the physical reactions following radiation, is not enough to understand the mechanisms of radiotherapy. After irradiation, there are many factors determining the radiosensitivity and thus survival of cells. These factor are often known as the 5 R's of radiobiology, indicating that the survival fraction is a combination of different biological processes. Those are repair of sublethal damage, reassortment, repopulation, reoxygenation and radiosensitivity, which is an intrinsic property of the cell [17]. The repair of sublethal damage (SLD) is the repair of double-strand breaks in DNA before they interact to form lethal lesions. This is seen in split-dose experiments, where it can be seen that some of the SLD produced in one fraction has been repaired between two subsequent fractions of radiaiton. Reassortment or redistribution is the progression of cells through the cell cycle. Most of the cells surviving a fraction is in a radioresistant part of the cell cycle (G1 and S), and naturally will progress into more radiosensitive parts of the cell cycle (G2+M) after some time. For cancer cells, the cell cycle is often shorter than for normal cells, and therefore these cells will end up in radiosensitive phases within a shorter amount of time than normal cells. Repopulation describes that clonogenic cells will continue to divide, also through the course of radiation, and is something that must be accounted for [17]. These effects are visualized in Figure 2.3.

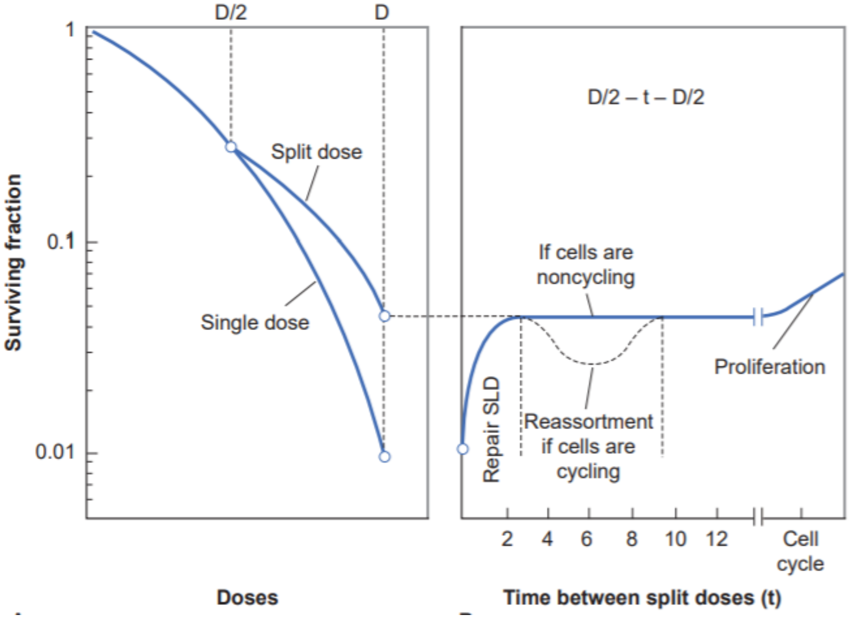


Figure 2.3: Summary of the repair of sublethal damage as evidenced by a split-dose experiment. Courtesy of [17].

Reoxygenation of cells is an important factor that is based on the oxygen fixation hypothesis, and the fact that tumors might be poorly oxygenated due to poor blood supply. This often leads to tissue hypoxia in the tumor, and radiation is less effective here. But during radiation, the ”outer layer” of a tumor might be killed, and the inner layer becomes closer to the blood supply, and becomes more oxygenated [17]. (The idea of a tumor being a symmetrical sphere is of course not true, and is just a simplifaction to visualize this effect.) This makes this oxygenated layer more radiosensitive than before at the next fraction. Therefore, fractionation is essential to reoxygenate tumor cells, so that these cells become more radiosensitive.

This is the motivation for using fractionation in radiotherapy. A treatment regime is chosen so that it will increase the therapeutic window. That is, it will further separate the survival of tumor cells and normal tissue cells. This is consistent with the allover goal of radiotherapy, to kill as many tumor cells as possible, and have as little normal skin toxicity as possible.

2.3 Dosimetry

Radiation interacts with matter in a series of processes where energy is converted and deposited in the material. Dosimetry is a method that provides a physical parameter to predict biological effects following radiation, and is essential for the outcome of a patient's treatment. One distinguishes between absolute and relative dosimetry, where the absolute dosimetry are measurements that provides an absolute dose determination in Gray (Gy). Relative dosimetry on the other hand provide measurements that need to be compared to a absolute reference measurement, to give an absolute dose determination in Gray [7].
There are several dosimetric quantities that can be used to determine biological effects, and for photons the most common ones are fluence, Kerma, charged particle equilibrium (CPE) and absorbed dose.

2.3.1 Dosimetric quantities for photons

This section will introduce and define several dosimetric quantities for photons that are useful to be familiar with.

Absorbed dose (D) is defined as the mean energy imparted (ϵ) by ionizing radiation per unit mass of an infinitesimal volume [7],

$$D = \frac{d\bar{\epsilon}}{dm} = \lim_{m \to 0} \frac{\bar{\epsilon}}{m} = \lim_{V \to 0} \frac{1}{\rho} \frac{\bar{\epsilon}}{V} \tag{2.4}$$

where $\bar{\epsilon}$ is transferred energy (energy imparted) and dm is the unit mass in the point where the dose is measured. The absorbed dose can also be expressed in terms of mass density, ρ (SI unit kg/m^3), and volume, V. The energy is not necessarily absorbed at the same place as where the energy was transferred. The energy imparted, can be expressed as

$$\epsilon = R_{in} - R_{out} + \sum Q, \tag{2.5}$$

where R_{in} is the sum of the energies (excluding rest mass) of all those charged and uncharged ionizing particles that enter the volume (radiant energy). R_{out} is accordingly the sum of all energies (excluding rest mass) of all particles that leave the volume, as illustrated in Figure 2.4. $\sum Q$ is the sum of all changes of the rest mass energy of nuclei and elementary particles in any nuclear transformations that occur in the volume [26], and is equal to zero for Compton scattering.

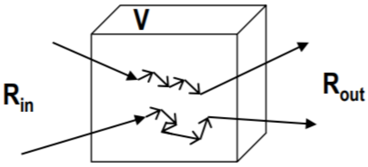


Figure 2.4: An illustration of the particles entering and leaving a volume, V. Note how the energy is not necessarily absorbed at the point of interaction, where it was transferred. Small arrows indicate how the creation of secondary electrons interact and deposit energy along their path, in many steps. Courtesy of [26].

Fluence is another dosimetric quantity, and can be described in various ways. Particle fluence, Φ, can be expressed as the expected number of particles, N, crossing the cross-section of a unit sphere, dA,

$$\Phi = \frac{dN}{dA}. \tag{2.6}$$

Alternatively, particle fluence can be expressed as the quotient of the sum of the track lengths, ds, of the particles crossing the elementary sphere and the volume of the sphere,

$$\Phi = \frac{\sum \delta s}{dV}, \tag{2.7}$$

as illustrated in Figure 2.5.

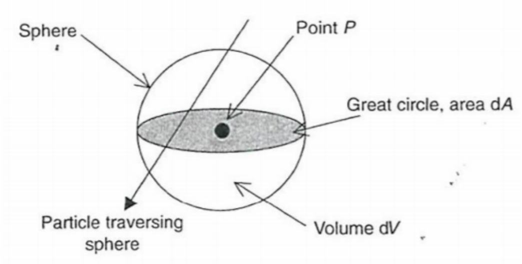


Figure 2.5: Illustration of a particle striking a finite sphere surrounding point P, with the sphere reduced to an infinitesimal one at P with at cross section of dA. The direction of the radiation is not taken into account. Courtesy of [26].

Another expression that is derived from the particle fluence, is the energy fluence, Ψ. Energy fluence is the product of the energy, E, with the particle fluence, Φ,

$$\Psi = E\Phi.$$

When the radiation contains of a spectrum of energies, the energy fluence is expressed as [10]

$$\Psi = \int_0^{E_{max}} E\Phi_E dE.$$

In order to relate the fluence and the absorbed dose, other useful quantities must be defined. In a photon interaction, energy is transferred to kinetic energy of secondary electrons which will impart their energy close to the point where they were released. However, one must distinguish between the energy that is transferred, and the energy that is actually absorbed. This can be expressed in the mass energy transfer coefficient, as well as the mass energy absorption coefficient. The mass energy transfer coefficient can be expressed as

$$\frac{\mu_{tr}}{\rho} = \frac{1}{\rho}\frac{dE_{tr}}{ENdl}, \tag{2.8}$$

where μ_{tr} is the fraction of the photon energy transferred to kinetic energy for charged particles (electrons) pr unit length. N is the number of uncharged particles, each with energy E, passing a thin slab of material with length dl. From this expression, we can derive the expression for the mass energy absorption coefficient,

$$\frac{\mu_{en}}{\rho} = (1-g)\frac{\mu_{tr}}{\rho} \tag{2.9}$$

where μ_{en} is the fraction of the photon energy that is absorbed pr unit length. The mass energy absorption coefficient allows for energy loss of the electrons to secondary photons, presented by g [26]. In low-Z material (e. g. soft tissue) at low photon energies, the energy loss from the electrons comes almost entirely from ionizing collisions. Then the effect of brehmsstrahlung is small, and $\mu_{tr} \approx \mu_{en}$, hence $g \approx 0$.

KERMA is short for Kinetic Energy Released per unit MAss. It is defined as the sum of the initial kinetic energies, dE_{tr}, of all the charged ionizing particles liberated by uncharged ionizing particles in a material of mass dm, divided by the mass dm [15]. Thus, Kerma can be expressed as,

$$K = \frac{dE_{tr}}{dm} = \mu_{tr}E\frac{Ndl}{dm} = \frac{\mu_{tr}}{\rho}E\frac{Ndl}{dV}, \tag{2.10}$$

where K represents Kerma. Substituting dm with ρdV and Ndl with $\sum \delta s$, another expression for Kerma is obtained, where one can see that it is directly proportional to the energy fluence, Ψ,

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

The kinetic energy for the electrons can be deposited through inelastic collisions with atomic electrons (mainly) and through radiation losses in collisions with atomic nuclei. This yields a division of the Kerma quantity into two components, K_{col} and K_{rad}, where $K = K_{col} + K_{rad}$. The first part,

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

is the expectation value of the net energy transferred to charged particles per unit mass at the point of interest, excluding both radiative energy loss and energy passed from one charged particle to another (energy deposition in or near the electron track). The other part, $K_{rad} =$ is the part of Kerma that leads to the production of radiative photons (bremsstrahlung, annihilation).

One can relate Kerma and the absorbed dose when certain conditions are fulfilled. Revisiting the mean energy imparted, and assuming Compton scattering, $\sum Q = 0$, one obtains the expression,

$$\epsilon = R_{in} + R_{out}. \tag{2.12}$$

One can also express the mean imparted energy as,

$$\epsilon = E^n_{tr} - E^n_{out} + E^n_{in}, \tag{2.13}$$

where $E^n_{tr} = E_{tr}(1 - g)$. If the electron track that leaves the layer is replaced by an identical track that enters the layer, one has that $E^n_{in} = E^n_{out}$, and thus $\epsilon = E^n_{tr}$, which is the condition named charged particle equilibrium (CPE). When this holds, one can finally relate the absorbed dose, D (see Equation 2.4) with the collision part of the Kerma, K_{col},

$$D \overset{\text{CPE}}{=} K_{col} = \frac{\mu}{\rho} E\Phi, \tag{2.14}$$

where μ represents μ_{en}. CPE exists in a volume, V, in an irradiated medium if each charged particle of a given type and energy leaving the volume is replaced by an identical particle of the same energy entering V [26], as illustrated in Figure 2.6.

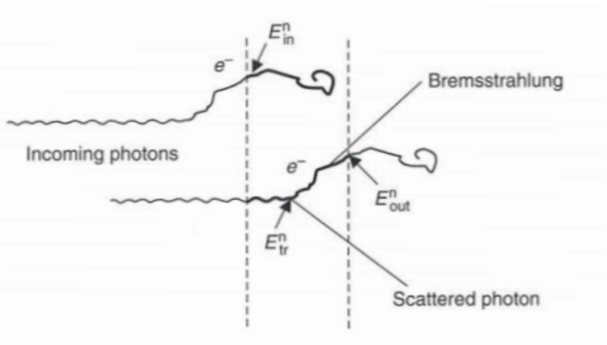


Figure 2.6: An illustration of the charged particle equilibrium situation, where $E^n_{in} = E^n_{out}$, and thus $\epsilon = E^n_{tr}$. Courtesy of [26].

CPE is a delicate dosimetric condition that can be achieved if certain conditions are fulfilled. The most important are that the photon field is not significantly attenuated, and that the range of the electrons is short compared to the diameter of the volume V. CPE is very well approximated at depths beyond the dose maximum in media irradiated by photons below around 1 MeV. For higher energies CPE does not hold, and so D is no longer equal to K_{col}, D is in fact only proportional to K_{col} at such energies. This is referred to as transient charged particle equilibrium (TCPE) [26]. The relation between the absorbed dose, D, and the collision Kerma, K_{col} can be viewed in Figure 2.7. From Figure 2.7 it becomes evident that the build-up region, which will be explained in more detail later, is not subject to CPE, and that this might affect calculations based on CPE that is performed in the build-up region.

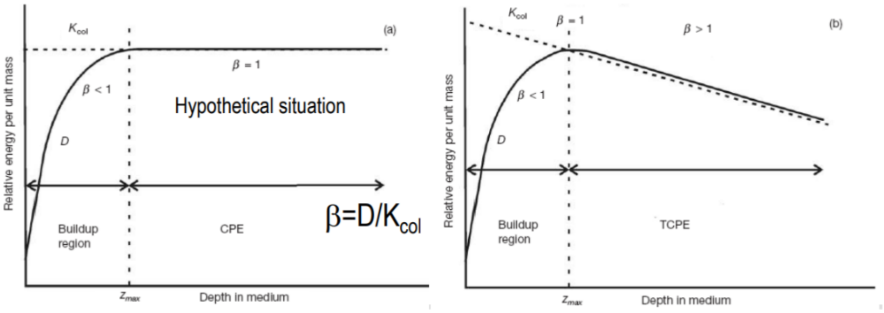


Figure 2.7: Collision kerma and absorbed dose as a function of depth in a medium irradiated by a high energy photon beam for a) the hypothetical case of no photon attenuation or scattering and for b) the realistic case. The x-axis represents the depth in medium, and the build-up region is defined as the region within the medium before the dose reaches its maximum value. Courtesy of [26].

2.3.2 Depth dose profiles

Depth dose profiles describe how dose is deposited into the depth of tissue. As mentioned previously, the dose is delivered by secondary atomic electrons, rather than from the primary photons themselves. These electrons move and deposit dose through collisions. Thus, the dose is deposited a bit further away form the point of photon interaction. So this cloud of secondary electrons takes some distance to collide and to build up the deposited dose, referred to as the build-up region.

The dose that is accumulated at the boundary between the air and the patients skin is referred to as the surface dose. In radiotherapy, this surface dose is only about 10%-30% of the maximum dose for a photon beam [3], as can be viewed in Figure [30]. That is why the build-up region is of great interest and is clinically useful as it spares the skin.

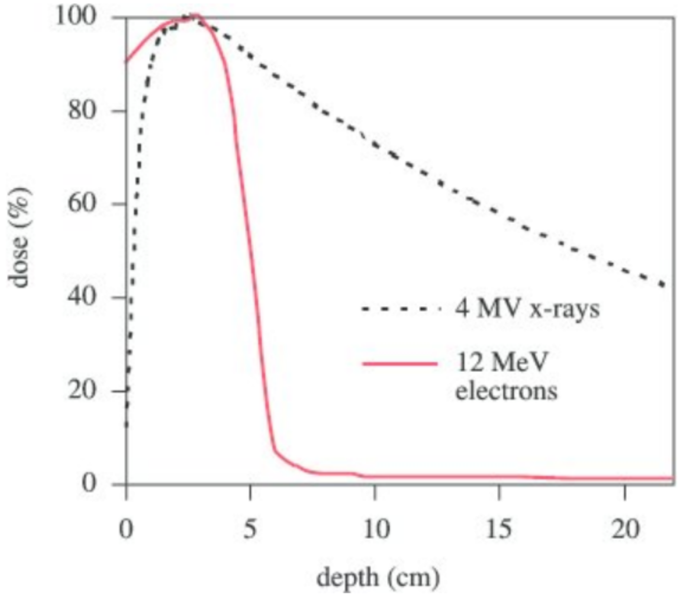


Figure 2.8: Photon and electron depth–dose curves. Courtesy of [30]. The surface dose (at depth 0 cm) is about 75%-95% of the maximum dose for an electron beam and only about 10%-30% for a photon (x-ray) beam.

The dose deposited within the first few millimeters of skin varies considerably due to the characteristic build-up of the photon beam. Starting at about 10%-30% of the maximum dose at the surface, and reaching 100% of the maximum within few centimeters into the tissue. To avoid skin complications, the surface dose is typically considered as an important criteria in the treatment plan [3]. Therefore, knowledge about the accurate surface dose is essential for assessing the skin damage, and designing a treatment plan. However, dose calculations performed by the treatment planning system (TPS) are known to be inaccurate in regions of electronic disequilibrium, like the build-up region [1]. So to gain more information about the actual dose given in the build-up region, it is necessary to study this area in more detail.

2.3.3 Dosimetric quantities for electrons

This section will introduce and define some important dosimetric quantities for electrons. Electrons are particles, in contrary to photons. Therefore many definitions will differ from the equivalent quantity provided for photons, given in 2.3.1.

Linear stopping power, S, is defined as the average energy loss by the electron per unit path length, dx,

$$S = \frac{dE}{dx}. \tag{2.15}$$

From this we can derive the total mass-energy stopping power. It is defined as the linear stopping power divided by the mass density of the absorbing medium, ρ.

$$\left(\frac{S}{\rho}\right)_{tot} = \left(\frac{S}{\rho}\right)_{col} + \left(\frac{S}{\rho}\right)_{rad} \tag{2.16}$$

The mass-energy stopping power is not dependent of mass density, except for the density effect. The density effect describes how a charged particle polarizes the medium, and thus the effective Coulomb force on a fast charged particle by atoms distant from the particle track is reduced. This effect is significant for dense materials.

Dose is absorbed due to electrons slowing down. Evaluating the energy that is deposited in a thin layer of material with thickness dl, only the energy deposited locally, dE, will contribute to the absorbed dose. Therefore the stopping power due to collisions with electrons, S_{col}, is of interest, yielding,

$$dE = S_{col} dlN. \tag{2.17}$$

Here, N is the number of electron tracks incident perpendicularly on a material of thickness dl. When expressing the energy deposited locally it is appropriate to use the collision stopping power, S_{col}, rather than the total stopping power, S, as the latter would include the energy lost in the form of bremsstrahlung that would escape the thin layer of interest. Dividing the expression above by dm or it's substitution, ρdV, one obtains,

$$\frac{dE}{dm} = \frac{S_{col}}{\rho}\frac{Ndl}{dV} = \frac{S_{col}}{\rho}\Phi. \tag{2.18}$$

This gives us the expression,

$$D = \frac{S_{col}}{\rho}\Phi, \tag{2.19}$$

where S_{col}/ρ is the mass stopping power, and Φ is the fluence [26].

CEMA is short for Converted Energy per unit MAss, and is the equivalent to Kerma for photons. CEMA is defines as energy lost by charged particles, excluding secondary electrons (δ-rays), in a mass dm. CEMA is equal to dose, D, when δ-ray equilibrium exists. That is, charged particle kinetic energy leaving a small volume is replaced by an equal amount entering the volume deposited in it,

$$D \overset{\text{d-eqm}}{=} \frac{S_{col}}{\rho}\Phi[26], \tag{2.20}$$

After all these dosimetric quantities have been defined, it is about time to introduce how dose is measured.

2.3.4 Measuring dose in tissue

To measure absorbed dose one must be able to find a relation between the measured ionizations in a probe (i.e. an ionization chamber) injected into a medium, and the absorbed dose in the medium at a given position. Dose is defined in a point, and will vary from point to point in a medium. However, any measurements preformed by a detector provides the dose in a volume, which means that the average dose in this volume is measured. The signal from a radiation detector will generally be proportional to the energy absorbed in the detector material. The relation between the absorbed dose in the detector and the absorbed dose in the medium is described using cavity theory. Cavity theory builds on two opposing conditions:

1. The dose must be roughly constant throughout the volume,
2. and $\bar{\epsilon}$ must be so high that statistical fluctuations becomes negligible,

where the first condition is in favour of a small measuring volume, while the second condition suggests a large measuring volume. Assuming these conditions hold, the aim is to find a relation between the absorbed dose to the medium/material, D_{med}, and the absorbed dose to the detector, D_{det}.

For large photon detectors (compared to the range of electrons) one assumes monoenergetic photons incident on a phantom of material, med, with energy fluence Ψ at the depth of interest, z, and sufficient CPE. Then the relation can be expressed as the ratio of absorbed doses,

$$f_Q = \left(\frac{D_{med,z}}{D_{det}}\right)_Q = \frac{(\mu_{en}/\rho)_{med}}{(\mu_{en}/\rho)_{det}} \tag{2.21}$$

where Q is the radiation quality. Note that this expression assumes that the detector does not disturb the photon fluence, see Equation 2.14. In the megavoltage energy range (which is used in radiotherapy) it is in fact impossible for radiation detectors to fulfil the large photon detector condition without becoming impractically large.

Since the large photon detectors are impractical, it is more interesting to investigate the small photon detectors (small compared to ranges of electrons). Such a cavity is referred to as a Bragg-Gray cavity, and must hold these conditions:

- The cavity must not disturb the particle fluence existing in the absence of the cavity (cavity small compared to the electron range) (CPE or TCPE), i.e. $\Phi_{det} = \Phi_{med,z}$.
- The absorbed dose in the cavity is deposited entirely by the charged particles crossing it (photon interactions negligible)

Air-filled ionization chamber used in MV-radiotherapy is in fact a Bragg-Gray cavity, and will be explained in more detail later in the Dosimeter section. Unfortunately, the extent of the detector is too small for CPE to be established, and therefore one cannot use the photon energy fluence to express the ratio of the absorbed doses. Instead, one must use the relationship between electron fluence and absorbed dose [26],

$$f_Q = \left(\frac{D_{med,z}}{D_{det}}\right)_Q = \frac{\Phi_{med,z}\,(S_{col}/\rho)_{med}}{\Phi_{det}\,(S_{col}/\rho)_{det}} = \frac{(S_{col}/\rho)_{med}}{(S_{col}/\rho)_{det}}. \tag{2.22}$$

Recall that Φ is the particle fluence. When the ratio, f_Q, is obtained one is able to calculate the dose to the material, according to the formula,

$$D_{med} = f_Q D_{det} \tag{2.23}$$

as illustrated in Figure 2.9.

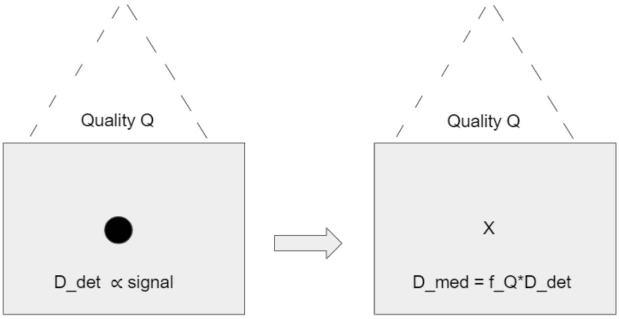


Figure 2.9: Illustration of measurement in cavity, given that the detector is large enough for CPE to exist. When the detected signal, being proportional to the absorbed dose to the detector, D_{det}, is found, one can calculate the absorbed dose to the material, $D_{med} = f_Q D_{det}$, for a given radiation quality, Q. Figure is adapted from [26].

In an ionization chamber, the absorbed dose in the detector (which is typically a gas) is given as:

$$D_{det} = \frac{Q}{m_{det}} \cdot \left(\frac{\bar{W}}{e}\right)_{det} \tag{2.24}$$

Here, Q is the ionization per unit volume, produced in the cavity material (SI unit Coulomb), and m_{det} is the mass of the gas (SI unit kg). $\left(\frac{\bar{W}}{e}\right)_{det}$ is the mean energy required to produce an ion pair in the detector cavity, divided by the charge of an electron (SI units Joules/Coulomb) [7].

2.3.5 Dosimeter

A dosimeter is a device/system that is able to measure the average absorbed dose deposited in its sensitive volume by ionizing radiation. A dosimeter should preferably be an absolute dosimeter. That is, being able to convert the measurement to an absorbed dose in Gray directly. However, the characteristics of such a dosimeter is more complex, and has a poorer spatial resolution, compared to its inferior relative dosimeters. That is why sometimes a relative dosimeter might become useful, especially in the cases where one depends on a high spatial resolution.

To be considered a good dosimeter, there are several desirable properties the dosimeter should have. The measurements taken by the dosimeter should be repeatable, easy to reproduce, have good accuracy (proximity of expectation value) and precision (small standard deviation). Also, the dosimeter should have a known (linear) response of energy and dose/dose rate. That is, no saturation of the measured signal for increasing dose/dose rate. Further, the dosimeter should not have any directional dependence. As well as sufficient spatial resolution and insensitivity or known response to influence quantities, such as temperature, directional effect etc [7]. There exists several different dosimeters, and most common is the ionization chamber. The ionization chamber is the standard dosimeter in the clinic, and provides measurements of absolute dose. An ionization chamber is composed of a gas cavity and an electric field roughly speaking [16].

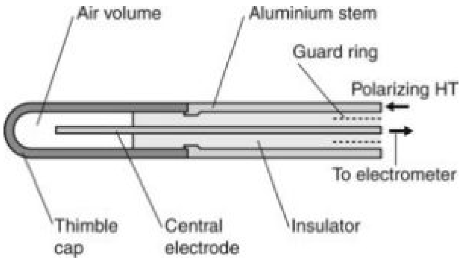


Figure 2.10: A principle sketch of a cylindrical ionization chamber. Courtesy of [26].

If the chamber is irradiated with ionizing radiation, ions will be formed in the gas cavity, and depending on charge will be drawn towards the central electrode or chamber wall due to the electric field the ions are experiencing. The absorbed dose can then be calculated from the charge accumulated on the electrodes of the ions that were caught, see Equation 2.23 and 2.24.

Some advantages provided by the ionization chambers are that they provide absolute dose measurements, have a linear dose response and gives constant response over time. But there are some limitations to the results acquired form an ionization chamber due to the size of the dosimeter and the distance between the discrete measurements. In order to achieve charged particle equilibrium, the gas cavity, which is the measuring volume within the inoization chamber, must be of a certain size. However, the size of the gas cavity should be small in relation to the dose gradient. This puts restrictions on how good the spatial resolution of the ionization chamber can be, especially when measuring dose profiles, isodose-curves and depth dose-curves in areas of high dose gradient [7]. These imperfections of the ionization chamber as a dosimetric tool is one of the motivations for using GafChromic film as a complementary dosimetry technique. Especially for small-field measurements, which require high spatial resolution. The GafChromic film provides a continuous 2D relative dose distribution, with as good resolution as the instrument reading the film (in our case a scanner), and can be useful if one wants to determine the dose distribution for modern treatment techniques, which will be introduced later. These modern techniques employs radiation from several angles, and possibly several segments with different intensity modulation per angle. The complexity of these treatment modalities poses the need for a high spatial resolution dosimeter, which can distinguish between steep dose changes in a small radiation field [38].

2.4 Film dosimetry

2.4.1 Radiochromic film

Radiochromic film is a chemical dosimeter which uses the optical characteristics of a dye to map the dose distribution. Radiochromic reactions are defined as direct coloring of a medium following absorption of radiation, without the need for thermal, optical or chemical development or reinforcement [9]. Just like the film in a polaroid camera self-develops after exposure, becoming a picture, radiochromic film also self-develops after being exposed to radiation. That is, the film darkens in color, where it has been irradiated. In this context, radiation is related to high-energy radiation, associated with external radiation therapy, but one must also account for lower-energy radiation arising from other sources. The radiochromic films are relatively light insensitive, but will be colored if exposed to light over time. One has to be especially careful when dealing with fluorescent light and regular sunlight, as these may contain UV-light, known to interact and cause dyeing of the film. Therefore one should not expose the films to any more light than necessary, and store the films in a light proof envelope.

The radiochromic film contains crystals filled with monomers that react upon irradiation by polymerization, forming polymer chains. When this happens the film will darken and become less transparent, and the transparency will decrease with increasing dose. Thus, the degree of polymerization increases with increasing absorbed dose. The polymerization is instantaneous, and leads to almost full color development within a very short time (milliseconds after irradiation). In this period, shortly after irradiation, the color development occurs at a high rate. However, some radiation-chemical reactions in polymers occurs at a slower rate, and so the total color development will take longer time [29]. Therefore, the total radiochromic reactions are not saturated before hours have passed. As a consequence, scanning of the film typically occurs no earlier than 12 hours after irradiation.

2.4.2 GafChromic EBT3 film

The method of measuring absorbed dose, based on radiochromic reactions, has been used since the late 1980s. Over the last decade radiochromic films have undergone a lot of development and the uncertainty in determination of dose has improved. In 2004 a film named GafChromic EBT was launched. EBT is short for external beam therapy, which is the radiation technique that is used when irradiating the film. The first generation of GafChromic EBT films was initially shown to be approximately energy independent over a wide range of energies. However, the later batches did not show such as good energy independence, and eventually went out of production [16]. The second generation, EBT2, showed improved energy dependency compared to the first generation, EBT, but had other undesirable properties such as unwanted Newton rings [12].

GafChromic EBT3 (8x10 inch) is the latest generation of GafChromic EBT films, and is the radiochromic film used in this project. It is composed of an active layer and two outer polyester layers. The active layer consists of crystals filled with a monomer (diacetylene), that react upon irradiation by polymerization, forming polymer chains. When this happens the film will darken and become less transparent, and the transparency will decrease with increasing dose. It is this property that is used when relating the intensity read out in a scanner and the prescribed dose. The dynamic range of the EBT3 GafChromic film is between 0.1 Gy to 20 Gy, but optimum dose range lies between 0.2 Gy to 10 Gy, which is well suited for applications such as VMAT and IMRT. These values are delivered by Ashland, the producer of the EBT3 GafChromic film. [5].

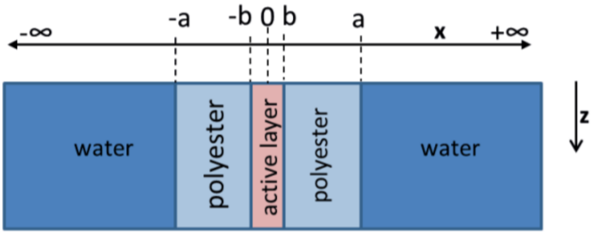


Figure 2.11: A drawing of the composition of the EBT3 GafChromic film. It is composed of an active layer (of thickness 25 um) and two outer polyester layers (both of thickness 125 um). The active layer consists of crystals filled with a monomer (diacetylene: Lithium pentacosa-10,12-diynoate (LiPCDA)), that react upon irradiation by polymerization, forming polymer chains. [37].

The polymer chains in the active layer absorbs light in typical bands at wavelengths of 635 (red) and 585 (green) [37],[23], as can be seen in Figure 2.12. This yields the highest resolution in the red and green color channel. However, the absorbance maximum is at the wavelength corresponding to the red light. Therefore one usually only uses the red color channel when reading out intensity in the images scanned of irradiated film.

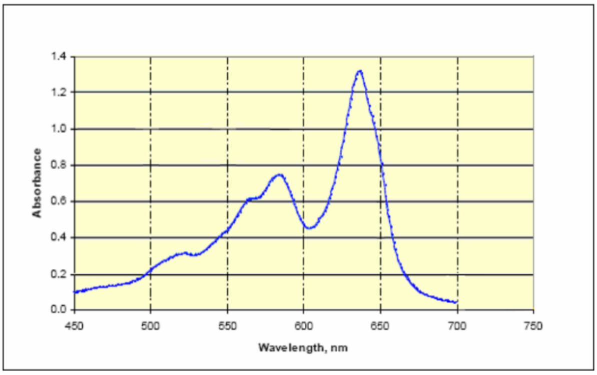


Figure 2.12: The EBT3 film exhibit two absorption bands centered at around 636 and 585 nm. Courtesy of [23].

2.4.3 GafChromic XR-QA2 film

GafChromic XR-QA film is a radiochromic film designed specifically to be used in dosimetry and radiology applications. GafChromic XR-QA2 film is a newer version replacing the initial XR-QA model. The latter film is more sensitive to a lower dose range from 1 to 200 mGy and energy range from 20 to 200 kVp [2]. GafChromic XR-QA2 film is a reflective type of film, as opposed to the GafChromic EBT3 film. It consists of five layers: a 97-m-thick yellow polyester layer, 15-m-thick pressure-sensitive adhesive layer, 25-m-thick active layer, 3-m-thick surface layer, and 97-m-thick opaque white polyester layer, as illustrated in Figure 2.13.

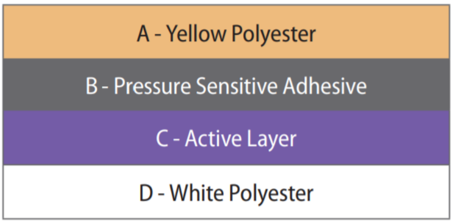


Figure 2.13: GafChromic XR-QA2 film consists of five layers: a 97-m-thick yellow polyester layer, 15-m-thick pressure-sensitive adhesive layer, 25-m-thick active layer, 3-m-thick surface layer, and 97-m-thick opaque white polyester layer [6].

The atomic composition of the active layer of the film is made up of H, N, O, C, Li, Br, Bi, and Cs [6]. The inclusion of several high-Z elements such as Bi (Z = 83) increases the photoelectric cross-section (i.e. probability of reaction to occur) and boosts the sensitivity of the film to the lower energy X-rays, making it suitable for dosimetric purposes in diagnostic dosimetry and radiology [2].

2.4.4 Optical density

When irradiating the radiochromic film the dye will make the film less transparent. This will result in a lower intensity measured when using a transmission scanner. The higher the dose absorbed by the film, the lower will the measured intensity (in pixel values) be. The optical density (OD) of a medium describes the mediums ability to delay the transmission of light, meaning that the OD of the film will increase with increasing dose. In the pixel at position (x,y) the optical density is given by the Lambert-Beer law

$$OD(x,y) = log_{10} \frac{I_i(x,y)}{I_t(x,y} = -\xi \cdot OP(x,y) \tag{2.25}$$

Here I$_i$ is the incident light intensity, I$_t$ is the transmitted light intensity and x and y are the coordinates in lateral and scanning direction respectively. ξ is the absorption coefficient of the material (in this case the film), and OP is the length of the optical path. The geometry and model of the flat-bed scanner will influence the OD in different positions in the scanner [37].

The change in the optical density (OD) of the film is directly proportional to the absorbed dose of ionizing radiation [2]. Optical density can be expressed as:

$$OD = \frac{2^{16}}{PV + 1} [14], [2], \tag{2.26}$$

where PV is the pixel value that is read by the scanner. The value 2^{16} reefers to the choice of a 48bit RGB TIFF-format of the scanned film, where each color channel holds 16 bits, or 2^{16} grey values.

2.4.5 Flat-bed scanner

Following irradiation of GafChromic EBT film, the film must be scanned with a flat-bed transmission scanner to digitize the color development that has occurred in the film. Ideally, the scanner should be able to transmit an isotropic light source through the film, creating homogeneous lighting conditions, and thereby read out intensities through detectors, into pixels. However, the light in a flat-bed scanner is anisotropic, and this can lead to systematic errors in the detector read-out. In addition, there are other factors affecting the detector read-out in a flat-bed scanner. In an earlier study, [37], the effects on the response in a flat bed-scanner from cross talk, optical path and polarization were looked into. These three effects were found to be fully responsible for the change in optical density (OD) in the lateral direction. Despite this, the producer of the GafChromic film mentions the finite anisotropic light source as the reason for the lateral variation [16]. As both the film type (GafChromic EBT3) and the scanner design used at St. Olavs Hospital is the same as the referenced study, this project adopts their results.

When using a transmission scanner, the light is transmitted through the scanner surface, and detected on the other side, by individual photodiode detectors. The spatial resolution of the scanner is limited by how many detectors the scanner is made up of. One can view the detector plate as a 2D grid of photodiodes, each detecting signal intensity, and sorting intensity into the three color channels (RGB) based on wavelength [13]. Thus, each pixel value is a decomposition of detected intensity, and is stored as an array of [red intensity, green intensity, blue intensity]. This principle is the basis for all further image processing and data representation and is the reason why results will be represented in terms of pixel values. Pixel values are absolute measurements and allows for different results to be compared.

2.4.6 Image processing with Python

To analyze the scanned image, one must use a suitable tool. Since the intensities in the image are represented by pixels, it is reasonable to evaluate the image using a suitable software. Python is a suitable programming language that can make such a software. The advantage of writing a program in Python is that it is free, it is very much used, and can therefore be modified by persons with knowledge of Python. Python can do various tasks, and the only obstacle is the programmer's knowledge of Python. Examples of what makes Python useful as a dosimetry tool is that it can create a Graphical user interface (GUI), read DICOM files, analyse dose plan matrices, read pixel values from an image, and much more.

2.5 Evaluation metrics

2.5.1 CT

In external radiotherapy, the treatment planning is based on knowledge of the patient's anatomy and positioning of the tumor. This knowledge comes from 3D imaging of the patient, obtained by computed tomography (CT) scans. Images are obtained by X-ray transmission computed tomography in many different directions. Quantitatively, transverse tomograms can be used to compute radiation dose distributions, based on the patient-specific geometry and the density of the tissues [8]. CT numbers are quantitative density numbers, given on the Hounsfield scale. The Hounsfield Unit (HU) scale presents a linear transformation of the measured linear attenuation coefficient μ, given by

$$HU = \frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}} \times 1000 \tag{2.27}$$

Here, μ_{air} and μ_{water} are the linear attenuation coefficients of air and water, respectively. The equation transforms the measured linear attenuation, μ, such that it is defined as zero HU for water, and -1000 HU for air, at standard temperature and pressure [10]. The HU is commonly used in CT scanners to express density in a standardized form.

2.5.2 Volume definitions

The International Commission of Radiation Units (ICRU) have defined useful characteristic values for distributions that are relevant for radiotherapy [15]. These definitions and values are important in making sure that medical physicists, radiation oncologists and other people working with radiotherapy have a common language.

In radiotherapy there is a need to define three-dimensional contours from the planning CT (pCT) of the patient. Volumes for external beam radiation therapy are defined by the Norwegian Radiation Protection Authority (NRPA) in the radiation report 2012:09 [24]. From the pCT, there are two types of volumes that should be defined. That is the gross tumor volume (GTV) and the clinical target volume (CTV). The GTV is the visualized tumor from the pCT, and thus called an anatomical volume. It consists of the primary tumor as well as regional lymph nodes, more distant metastases, and/or local residues. In many cases the GTV can be removed surgically. The CTV contains the GTV in addition to areas where one suspects subclinical malignant disease. To ensure that the CTV gets the requested dose, margins for different deviations and variations, in addition to tumor spread are commonly considered. The internal target volume (ITV) consists of the CTV and a margin (internal margin, IM) that considers internal moving, such as breathing and anatomical changes from the previous dose delivery (fraction). The ITV also accounts for uncertainties in target delineation. The setup margin (SM) accounts for patient movement and inaccuracies in patient alignment and beam fields between succeeding fractions. Such deviations can occur if a patient is moving during the fraction or between different fractions, or if the equipment is poorly adjusted. Together, the SM and IM, makes up the total margin (TM). TM encloses all inaccuracies and variations in patients and equipment. More accurately, the TM is defined as

$$TM = \sqrt{IM^2 + SM^2} \tag{2.28}$$

The TM is usually added to the CTV, and together defines an important geometrical volume: the planning target volume (PTV). The PTV aims to ensure that the requested dose, with an acceptable likelihood, is delivered to the CTV, with all geometrical uncertainties included in the TM. In other words, it can be assumed that the prescribed dose is delivered to the CTV, as long as the CTV moves only within the boundaries given by the PTV. A schematic of the volumes and margins defined in this section is shown in Figure 2.14.

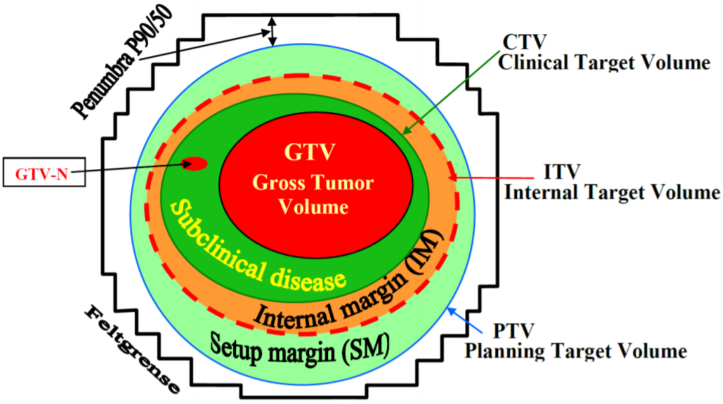


Figure 2.14: Graphical representation of various volume definitions, given by the NRPA [24]. The penumbra is defined as the distance between the 90% and 50% dose levels on a cross-section dose profile that is perpendicular to the central axis at a given depth. The GTV-N is a GTV for a lymph node placed outside the main GTV.

2.5.3 Dose and volume parameters

In addition to the volumes defined in the section above, there are quantities related to plan optimization and evaluation of quality that should be introduced. The average absorbed dose, D_{avg}, to a volume V, is defined as

$$D_{avg} = \frac{1}{V} \int_0^{D_{max}} D \frac{dV(D)}{dD} dD \tag{2.29}$$

Here D_{max} is the maximum dose to the volume V, and $dV(D)/dD$ represents the infinitesimal increment of volume per absorbed dose at absorbed dose, D. A much used value is D_{50} or D_{median}, that represents the absorbed dose that 50% of the volume receives. For a given PTV (defined in the section above), D_{median} is often referred to as the "typical dose" to the PTV. This nomenclature naturally applies to other values of interest as well, giving rise to a more general equation for a given region of interest (ROI):

$$D_x = \text{The dose received by x\% of the volume} \tag{2.30}$$

These definitions are much used, especially in reporting the near-minimum (D_{98}) and near-maximum (D_{02}) absorbed dose. Lastly, and quite predictable, D_{min} is the minimum dose to a given ROI [15].

Another way to come at it is to ask how large is the volume that received a given amount of dose? The volume that received a specified dose, y, can be presented as

$$V_y = \text{The volume that received } y \text{ Gy to a given ROI} \tag{2.31}$$

The unit gray, with symbol Gy, is the derived unit of ionizing radiation dose. It is defined as the absorption of one joule of radiation energy per kilogram of matter: Gy=J/kg [17].

An important tool in investigating dose distribution and dose coverage is the dose-volume histogram (DVH). It relates the radiation dose to tissue volume and is most commonly visualized in a two-dimensional graph, where the x-axis indicates the dose, and the y-axis indicates the percentage of volume [15].

2.6 Treatment planning and delivery

2.6.1 Treatment planning system

A treatment planning system is an important tool used for planning and evaluation of treatment of cancer. It allows clinicians to plan optimum treatment parameters to match the desired treatment goals and constraints, using images and dosimetric data. Dose depositions are calculated based on the physical processes described in Section 2.2, as well as models that describe the stochastic effects of radiation interactions. Various treatment plannings systems exist, such as Elekta's Monaco, Varian's Eclipse and RaySearch's Raystation. RayStation 8B is the treatment planning system used at St. Olavs Hospital, and will therefore be used in this project.

2.6.2 Patient coordinate system in RayStation

Patient coordinates are stored in data files, and follow the DICOM standard. DICOM is short for Digital Imaging Communications in Medicine. The DICOM file holds important information in different attributes, and is useful to be familiar with. Among other things it contains the 3D dose matrix, that holds the calculated doses of the treatment volume. The DICOM file also specifies how the patient is positioned, in the patient position attribute, and this will affect how the dose matrix is oriented.

2.6.3 Delivery of photons

Linear accelerator, Elekta Synergy

This section is adapted from a previous project [16], performed by the author. A linear accelerator uses accelerated electrons, either directly or transformed into photons, to form an ionizing beam. The machine is built up by a modulator, electron gun, RF power source, accelerator waveguide, bending system and beam shaping and focusing. The modulator sends pulses to both the electron gun and the RF power source. The RF power source is a vacuum tube which generates microwaves (RF frequency) of high power. This leads to a propagating electromagnetic field inside the waveguide. At the same time the electron gun delivers electrons into the waveguide and the electrons with the right phase relative to the electromagnetic field will be accelerated. Since the frequency of the microwaves is fixed, the geometry of the waveguide is made to increase the wavelength and therefore the speed of the electrons. At the end of the waveguide there is a bending system which deflects the electrons using a magnet. Now the electrons can either produce photons through Bremsstrahlung or they can be used directly by letting them hit a scattering foil.

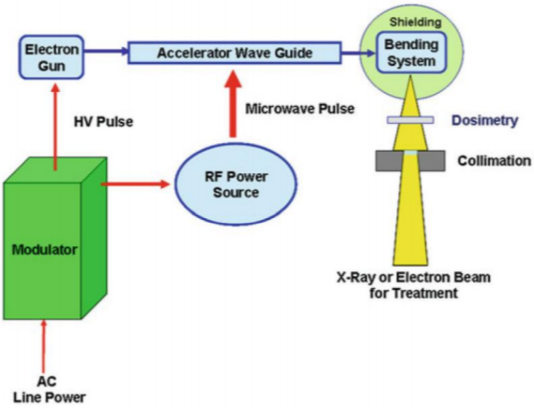


Figure 2.15: Simple schematic diagram of a linear accelerator. Courtesy of [33].

To make the beam clinical usable it must be shaped. First the beam goes through the primary collimator which defines the original field size of the beam. The primary collimator can be an open aperture (used for electrons and low energy x-rays) or a filtered aperture (used for high energy x-rays). Then the beam can be flattened using a flattening filter in what is called the secondary filter. Because the bremsstrahlung photons are more forward peaked, the fluence profile will be cone-shaped. The flattening filter has been an important component to decrease this effect. With the advanced techniques in use today the flattening filter is no longer needed, but is still used. 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 (MLCs) to define the field. MLCs are closely spaced, mobile "leaves" made of high-density, high-atomic number material. Due to the high atomic number, when an individual leaf moves into the beam path, it will attenuate, or block dose in that area [19]. Also, wedges and shutter can be used to modify the intensity of the beam [33].

The LINAC delivers radiation in discrete quanta called monitor units (MU). A monitor unit is in the order of 1/100 Gy, but will vary significantly with field size and interaction depth among other things.

Conventional treatment planning

Conventional treatment planning is as the name indicates a well established technique, and in general quick and reliable. One shapes the radiation field using blocks, wedges and MLCs in order to obtain the most conformal shape as possible to the target volume. Typical beam arrangements are the opposing beams, tangential beams, three-beam arrangements as well as the four-field box technique [34].

Tangential beams are common in irradiation of breast. The posterior borders of the field are aligned to avoid divergence into the lung, and wedges are used to achieve a more uniform dose distribution [10], as can be seen in Figure 2.16.

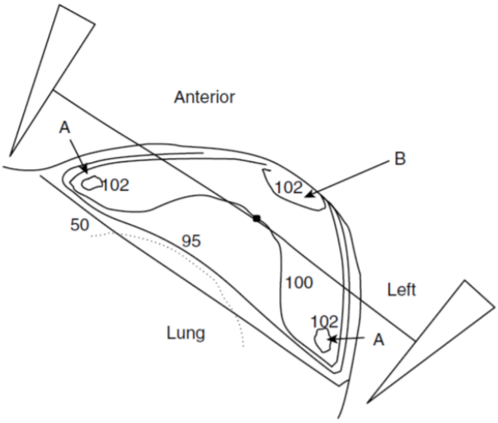


Figure 2.16: Illustration of tangential beams setup for irradiation of breast. The posterior borders of the field are aligned to avoid divergence into the lung, and wedges are used to achieve a more uniform dose distribution. The isodose curves drawn in the figure indicate how much depth dose, relative to the prescribed dose, each region is receiving. Courtesy of [26].

Three-dimensional conformal radiotherapy

Conformal radiotherapy shapes the radiation beams to closely fit the area of the tumor, where the position of the tumor is predetermined from a CT image. This technique is also called three-dimensional conformal radiotherapy (3D-CRT), and is a very common type of radiotherapy [19]. The positioning of the MLCs relative to the patient table can be seen in Figure 2.17, and the shaping of the beam from the beam's eye view (BEV) can be seen in Figure 2.18.

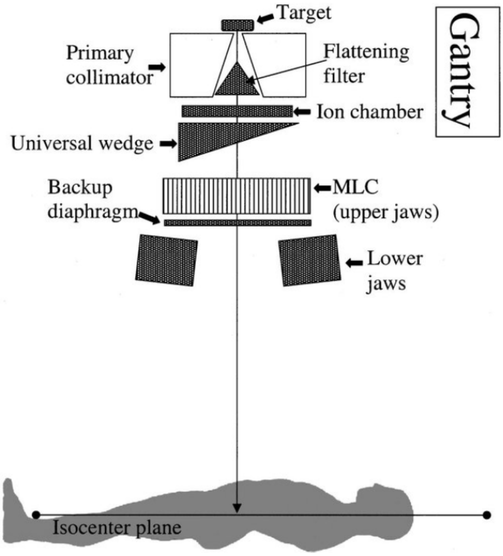


Figure 2.17: A schematic diagram of the different beam shaping features available in the linear accelerator. This is a simplified side view of the gantry head with collimator angle at 0° in relationship to the patient lying on the treatment table. When retracted, the MLC collimator is lateral to the patient. Courtesy of [27].

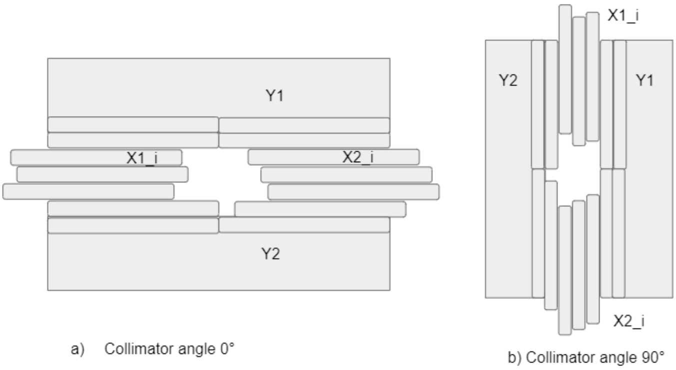


Figure 2.18: An illustration of the collimators in the gantry of the linear accelerator. The multi-leaf collimators (MLCs), X1$_i$ and X2$_i$, and the jaws, Y1 and Y2, are shown in the Beam's eye view (BEV) at a) 0° collimator angle and b) 90° collimator angle. The MLCs and the jaws enables conformal radiotherapy.

The choice of collimator angle is significant, as it is shown in several studies [36], [22] that it affects the total leakage dose given to tissue outside the PTV. Leakage dose refers to the leakage of radiation between the individual MLC leaves [22], and is an effect that should be minimized.

Tongue-and-groove effect

The purpose of the tongue-and-groove construction of the MLC is to minimize interleaf radiation transmission. Different vendors have different approaches to the tongue-and-groove construction, but they are all constructed with the same purpose. In Elekta, the MLCs are designed as seen in Figure 2.19, where the Agility model is the one used in this project. Due to the design of the leaf sides, the tongue-and-groove effect occurs for certain MLC applications such as the abutment of fields where the beam edges are defined by the sides of the leaves [18].

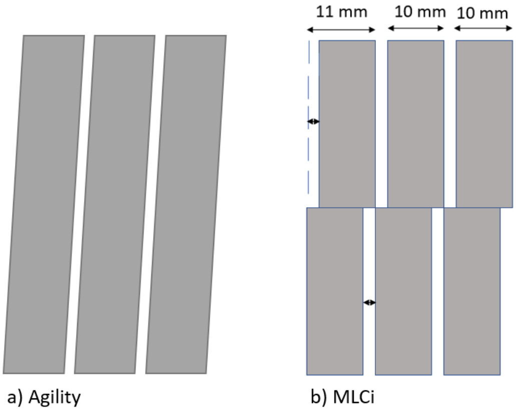


Figure 2.19: An illustration of two different designs of the Elekta multi-leaf collimator. The agility and MLCi model are illustrated. For the MLCi model the tongue width can be characterized by the horizontal distance from the upper left end, to the lower left end on an individual MLC leaf. The groove width can be characterized by the horizontal distance from the right lower end to the right upper end.

2.6.4 IMRT and VMAT

Intensity modulated radiotherapy (IMRT) and Volumetric Modulated Arc Therapy (VMAT) are so-called inverse treatment planning techniques, which means that a clinical goal is first set, and then a treatment plan is made to fulfill these objectives [10]. IMRT is a type of conformal radiotherapy, and uses more gantry angles than conventional treatment. The techniques is often described as ”step-and-shoot”. At each gantry angle, the beam is modulated in intensity, and the field is shaped. This improves the conformity of the beam to the treatment volume, as each field conformation at the different gantry angles are tailored to the BEV of the tumor. VMAT further improves the ability to conform the target volume, as it irradiates during the gantry is moving. Thus, the intensity and collimator shape in a

VMAT treatment can be changed almost continuously. Its goals are (compared to IMRT) to shorten the treatment time, reduce radiation leakage, and thus minimize the probability of secondary cancers arising from the treatment [36]. An illustration of the VMAT treatment technique can be seen in Figure 2.20.

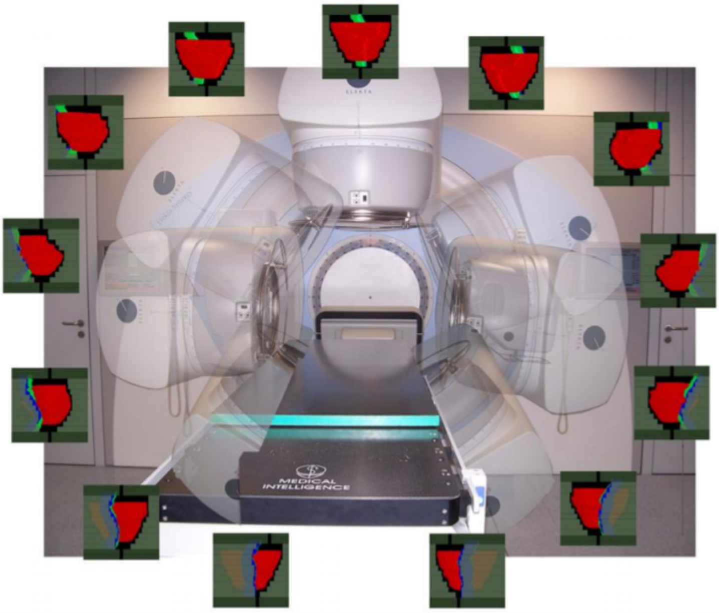


Figure 2.20: An illustration of the VMAT treatment technique. The gantry is moving almost continuously, and the collimators are shifting shape to conform the radiation to the target volume. Courtesy of [10].

2.6.5 Treatment planning of breast cancer

The major organs at risk in breast cancer radiotherapy are the heart, the lungs, the skin and the contralateral breast, as seen in Figure 2.21. The aim is to spare these OARs, due to considerations regarding acute and late effects to radiation. Such effects can include late cardiac toxicities, skin burns as well as radiation-induced cancer, among other things.

For breast cancer there are national guidelines recommending a $D_{avg} \leq 2$ Gy to the heart, and a V_{18} Gy $\leq 15\%$ to the lungs, for a fractionation regime yielding 2.67 Gy × 15 [28]. The minimum required dose to the PTV is 90% of the prescribed dose , but the 95% isodose is more common to use in the clinic. The PTV might also be expanded to include some air above the chest. This is done to ensure robustness in that direction, in case of breast deformation during the course of treatment. If the breast swells or develops a seroma (collection of fluid under the surface), causing the CTV to expand away from the lungs, such a PTV will ensure coverage.

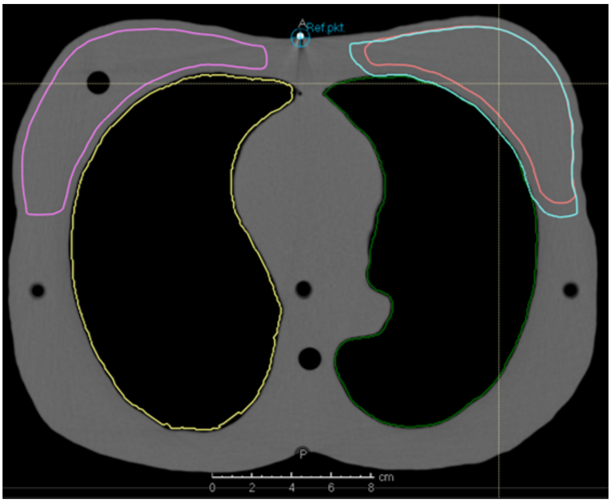


Figure 2.21: A CT image including the delineation of important regions of interest when irradiating a breast. Target volumes: the CTV of the left breast (pink), the PTV of the left breast (blue), and organs at risk: the left lung (green), the right lung (yellow) and the contralateral breast (purple).

In Norway today, breast cancer radiotherapy is typically delivered in 15 fractions with a prescribed dose to the CTV of 40 Gy, or 25 fractions with a prescribed dose of 50 Gy [28]. 3D-CRT is considered the standard treatment technique for breast cancer patients in Norway. However, hybrid plans consisting of tangential 3D-CRT fields with VMAT supplementing arcs should be considered if the prescribed dose is not met in deep regions of the chest [28].

2.7 Systems for verification of dose distribution

For verification of a treatment plan a phantom is irradiated and the dose distribution is measured with dosimeters. In the experiments described in Chapter 3 GafChromic film is used as the measuring system (dosimeter), but here there will be presented other methods to be able to discuss this system's pros and cons.

2.7.1 Point dosimeters

Thermoluminescence Dosimeters (TLD), diodes and ionization chambers are conventional dosimeters that can be placed in positions in the radiation field [14]. If the point measurements are performed for many points in a grid pattern, they will provide a 3-dimensional dose distribution with poor spatial resolution [10]. This process is a very time-consuming process unless detector arrays are used [14]. A disadvantage that comes with using diodes or ionization chambers is that there is a need for a scan of the detectors in water, or one must use a detector array of some sort. One can place diodes or ionization chambers into phantoms to present more realistic patient geometry, such as inhomogenities, but this is mostly limited to a few pre-defined positions in the phantom. Therefore, the resolution that can be obtained using point dosimeters is not very good.

2.7.2 Film

A calibrated radiochromic film can be placed into a phantom and be exposed to irradiation, and thus present the dose distribution in a plane. Film dosimetry provides 2-dimensional dose distributions with high spatial resolution [10]. Radiochromic film is easy to use, and can be stacked in different directions and be used to present a pseudo-3-dimensional dose distribution. Radiochromic film also gives the opportunity to construct a phantom with varying densities in a whole different way compared to diodes/ionization chambers. An important advantage using film, is that it is easy and quick to handle, and only requires to be scanned after irradiated.

2.7.3 Chemical dosimeters (gel)

Radiation induces chemical changes in the gel, and provides a 3-dimensional dose distribution. Using monomer/polymer based gels, a polymer network is created upon irradiation [10]. The dose distribution must be read either through MRI or tomographic scanning of optical density. The 3-dimensional dose distribution is a great advantage using gel as a dosimeter, but the necessity to scan the gel afterwards is more time and resource demanding compared to scanning a film.

Chapter 3

Materials and Methods

First, this chapter will give an overview of the procedure, or workflow, that was followed to perform film dosimetry in general. Then a presentation of the analysis method will be provided, focusing on the desired functionality of the film dosimetry analysis tool, FIDORA, made by the author and another physics student, Stine Gustavsen. At last, the experiments conducted to investigate the dose to the contralateral breast as well as the dose in the buildup area will be described in more detailed.

3.1 Workflow in film dosimetry

3.1.1 GafChromic film

The experiments conducted in this project will all be performed with GafChromic EBT3 film as well as GafChromic XR-QA2 film, together with an anthropomorphic thorax phantom. GafChromic EBT3 film is suitable for doses in the range of 0.2 Gy to 20 Gy as previously stated. GafChromic XR-QA2 film on the other hand is more sensitive for lower doses, in the range of 0.001 Gy to 0.2 Gy. EBT3 will be used to investigate the absorbed dose from one fraction to the target breast, as well as the dose to the contralateral breast (CLB) from 15 fractions. XR-QA2 will be used to measure the absorbed dose from one fraction to the CLB.

3.1.2 Calibration

In order to relate the absorbed dose in the GafChromic films with the pixel value from the scanning of the films, a calibration curve had to be established in advance under conditions resembling reference conditions. The reference conditions refer to the standard geometry defined by the IAEA TRS-398 protocol [20]. This protocol gives the absolute dose determination under reference conditions, and when using these specifications, one can assume that the linear accelerator is perfect and that the irradiated field is uniform and to the given dose. According to the protocol, a standardized geometry with field size 10x10cm is used, and the irradiation is perpendicular to the film. The film is placed between the I'mRT Phantom, which is a SolidWater slab phantoms, with a 10 cm depth and 90 cm distance from the phantom surface to the source [20]. Since the reference conditions assumes that the film is placed in water, the use of a different material introduces a phantom factor that must be related to the measurement in water. The I'mRT phantom is composed of nearly water equivalent RW3 material, which resembles the human body. In detail, RW3 is composed of 98% Polystyrol and 2% TiO_2, with density $1.045 g/cm^3$ [21].

In this project all experiments have been performed on films of the same lot number, but a new calibration curve was made each day to account for daily variations in the linac output. Also the film has a tendency to break at the edges where it is cut, therefore it is necessary to use larger pieces than the field of interest in measurements.

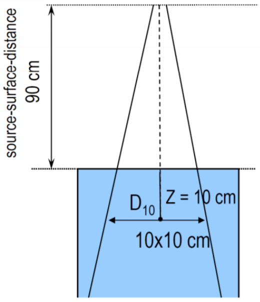


Figure 3.1: An illustration of standard geometry as defined in TRS-398, with a source-surface-distance of 100 cm. [11]

Irradiation under reference conditions is illustrated in Figure 3.1. An equivalent irradiation setup, only with an I'mRT phantom instead of water, was performed to establish three different calibration curves. The radiation quality used in this project was a 6 MV photon beam. For GafChromic EBT3 film one calibration curve was made using a filter-free radiation beam, and another calibration curve was made using a filtered radiation beam. Also, a calibration curve was established for the GafChromic XR-QA2 film using a filtered radiation beam. A calibration curve was fit on the form [6]:

$$d_x(D) = a + \frac{b}{D - c} \tag{3.1}$$

where $d_x(D)$ is the optical density of the film in scanner channel x at dose D, and a, b, c are the equation parameters to be fitted [6]. This function type is beneficial to use, as it has a rational behavior with respect to the physical reality (ref. Ashland [6]). That is, the optical density of the film increases with increasing exposure but approaches a near constant value at high exposure, which is consistent with a saturation of the polymerization that occurs within the active layer of a GafChromic film upon irradiation. This is only true within the valid dose range (ref. Ashland [6]).

In this project GafChromic film was cut into nine 2 cm x 2 cm pieces for both GafChromic EBT3 film and XR-QA2 film. The film pieces were irradiated in geometric progression with 0, 1, 3, 10, 33, 100, 333, 1000 and 2000 cGy under reference conditions.

3.1.3 Scanning and correction method

The GafChromic films were scanned with a flat-bed scanner, Epson v750 Pro. As the dye in the GafChromic films undergoes polymerization upon irradiation the GafChromic EBT3 film will be less transmitting, and the GafChromic XR-QA2 film will be less reflective. Higher doses will result in less transmission through the film for EBT3, or less reflectance for XR-QA2, which is detected on the photodiode detectors. So by relating the light transmission or reflection in the film with the absorbed dose, the dose distribution can be measured.

The film were always scanned in the same orientation (ref. Ashland [6]). This is due to a tendency of the particles in the active layer to align along the short side of the film. The result of that being an anisotropic light scattering during scanning. The producers suggested using landscape orientation, see Figure 3.2, which means that the original short side of the film is parallel to the direction of scanning. The landscape orientation was therefore chosen in this project as well [4].

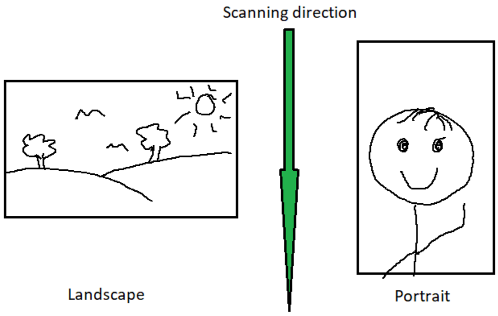


Figure 3.2: Orientation in the scanner

The scanners output is an image (format multi-TIFF) where each pixel corresponds to a small area of the film. The dose map is then found by investigating the optical density through the irradiated film and using a calibration curve made in advance.

The flat-bed scanner used at St. Olavs hospital gives a non-uniform read-out of the film. Since the finite light source is placed along the center of the scanning direction the intensity will decrease when moving away from the center axis. To correct for this non-uniform scanning, a correction method was developed, using GafChromic EBT3 film. The result is a correction matrix that will be used to perform an absolute subtraction.

This section is based on a previous project performed by the author [16]. To investigate how the transmission of light in the scanner is non-uniform and dose dependent, different doses where scanned over different positions of the scanning surface. The area of interest on the scanner surface was chosen as a 10x10cm square at the center of the scanner. The positions was read as a 5x5-matrix, resulting in 25 points in total. A film was cut out into six 2x2cm pieces and irradiated with the doses, 0, 25, 50, 100, 200 and 400 cGy. To limit post-exposure effects the irradiated film was placed in an opaque envelope and kept for at least 12 hours before being scanned [6]. A mask with a 10x10cm cut-out was used during scanning to assure right placement of the film pieces. All the film pieces were scanned in each of the 25 positions, using the set-up described with 127 dpi, and is illustrated in Figure 3.3. The reason for using many small film pieces, instead of one 10x10cm film per dose level, is that the irradiation beam is most reliable in proximity of the isocenter in terms of absorbed dose.

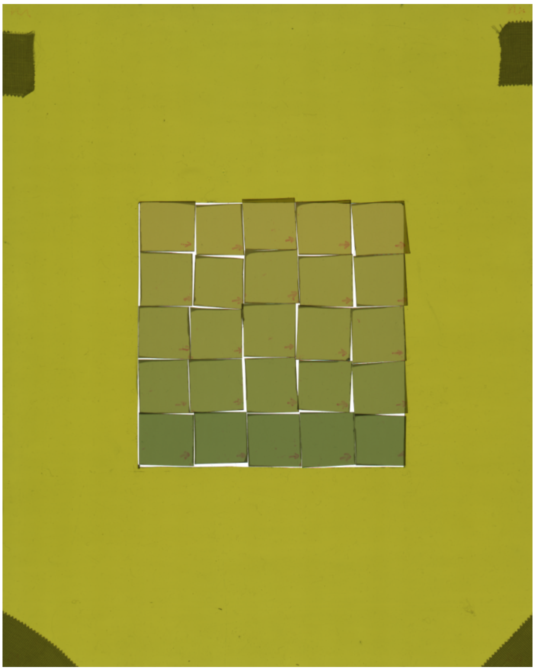


Figure 3.3: An illustration of the method/setup used in the scanner when constructing the correction matrix. Film pieces irradiated at different reference doses where moved and scanned in all 25 positions in the grid. In that way, the variation in readout in all the 25 positions in the middle of the scanner surface was investigated.

A glass plate was used to make sure that the film was flat on the scanning surface, since differences in optical path lengths have shown to induce artefacts in the image. Using Python for image processing, pixel values in a 3x3mm (15x15 pixels) ROI were collected in the middle of the film pieces and averaged. By repeating this for every dose in every position the result is a 5x5-map representing each dose at the scanner surface. The center element was chosen as the reference value and a correction matrix could be found by calculating the difference of each position value compared to the reference. With the correction matrix as a basis, a function taking position as parameters was fitted, using cubic interpolation and extrapolation. It has been shown that the optical density of the EBT3 film increases for each scan it is exposed to [16]. Because of this, it is chosen that the film should only be scanned one time in every position. This means loss of generality and an increase in the uncertainty. To account for that, this procedure is preformed five times with landscape orientation, and averaged.

3.1.4 Image processing

The software package “Epson scan” was used when scanning with the model Epson V750 Pro. The program was set to professional mode, transparency mode, positive film, 48-bit color, all adjustment setting off and a resolution of 127 dpi(dots per inch) corresponding to 0.2mm/pixel. The scan was saved as a raw file in TIFF-format, and read and processed using Python.

The read-out of the tiff-file is represented as a 3D-matrix where the last dimension holds the RGB-channels. Since the EBT3-film has an absorption maxima at wavelength 636 nm, the response is most pronounced in the 600 nm to 700 nm area, answering to the red part of the visible light spectrum [37]. However, all color channels was used when making a correction [16].

It was found in an earlier project that it is necessary to do 3-5 warm-up scans before scanning the film to avoid artifacts [16],[31]. This is important in order to stabilize the light source in the scanner, and thus produce equal lighting conditions at each scan.

3.2 Film dosimetry with FIDORA

FIDORA, short for ”film dosimetry in radiation therapy”, is a python-based program developed by the author in cooperation with another biophysics student. The aim of the program is to function as an analysis tool when using film dosimetry. Filmdosimetry is as earlier mentioned a good dosimetry alternative in certain cases, but the results need processing afterwards to become accessible to the staff at the cancer clinic. Data processing and analysis of the irradiated and scanned GafChromic films, are the tasks that FIDORA aims to fulfill. The flow chart presented in Figure 3.4 gives an overview of the the tasks such an analysis tool should be able to perform.

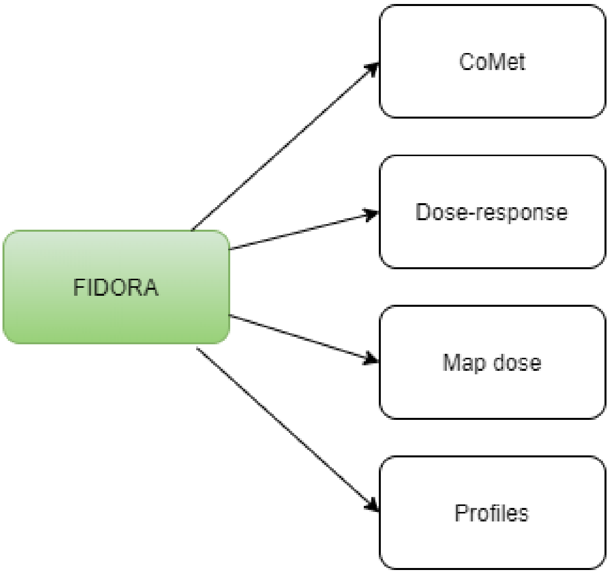


Figure 3.4: A schematic overview of the program FIDORA.

3.2.1 Correction Method, CoMet

Due to the anisotropic light conditions in the scanner-readout, there is a need to correct for this in the scanned image. Therefore, FIDORA must apply the correction method specified in Section 3.1.3 onto the scanned image, and this is the aim of the tab CoMet, short for correction method. The flow chart presented in Figure 3.5 gives an overview of the CoMet tab. The correction is based on 25 points at a 10x10cm area in the middle of the scanner. From this, all points within the 10x10cm square were found through cubic interpolation, and points outside where found through extrapolation. After interpolation and extrapolation, the effective corrected area is a 12x12cm square in the middle of the scanner.

In addition to the correction due to anisotropic light conditions, there is also a need to correct for salt- and-pepper noise in the scanned images. This type of noise can be caused by sharp and sudden disturbances in the image signal, and can be observed as sparsely occurring low and high intensity pixels. Median filtering is excellent at reducing this type of noise, and will be used in this project. The filtering algorithm will scan the entire image, using a small kernel (matrix) of a suitable size, and recalculate the value of the center pixel by taking the median of all of the values inside the matrix. After applying a median filter the image is effectively smoothed. That is, very low or very high intensity pixel values will be removed. If these deviating pixels were not to be removed, one would end up interpreting these as very low or very high dose values when converting the scanned image to a dose map.

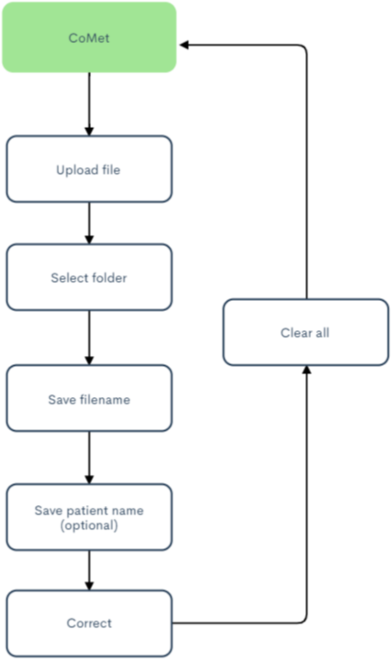


Figure 3.5: A schematic overview of the tab CoMet in FIDORA, responsible for performing a correction method on the scanned image.

3.2.2 Dose-response

The establishment of a calibration curve is the purpose of the tab Dose-response. It should enable the user to upload scanned film pieces (in the middle of the scanner) irradiated with known user-defined reference doses. For each dose level, the user should be able to upload several scanned images, and the average will be calculated. After enough calibration points are uploaded, the program should try to optimize a calibration curve on the form given in Equation 3.1. The calibration curve should be plotted for each color channel, and a standard deviation for the scan-to-scan variation occurring between multiple scans of the same reference films should be presented to give an indication of the scan-to-scan variation. After the calibration curve is established, the user should be able to store the calibration curve, so that it can be used later. A schematic overview of the tab Dose-response, can be seen in Figure 3.6.

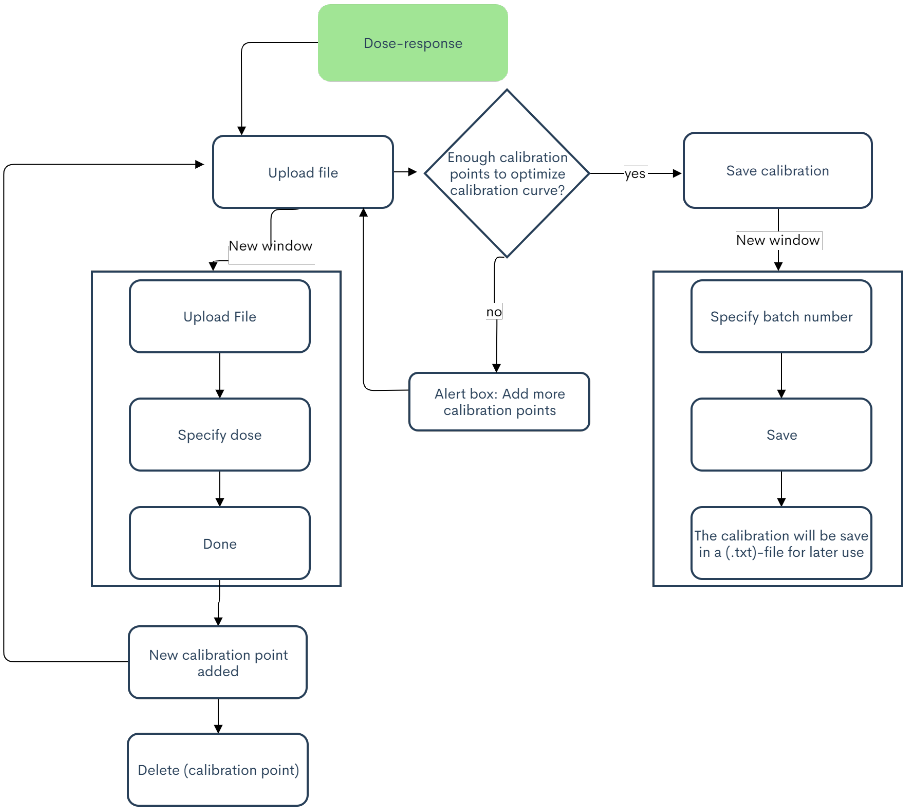

Dose-response
Upload file
Enough calibration points to optimize calibration curve?
yes
Save calibration
New window
Upload File
Specify dose
Done
no
Alert box: Add more calibration points
New window
Specify batch number
Save
The calibration will be save in a (.txt)-file for later use
New calibration point added
Delete (calibration point)

Figure 3.6: A diagram of the tab Dose-response in FIDORA, which is responsible for establishing a calibration curve. The user can upload calibration points, and when there are enough points for the program to optimize a calibration curve, the calibration curve for each color channel will be plotted and the calibration function will be given, associated with standard deviations for scan-to-scan variations.

3.2.3 Map dose

The Map dose tab in FIDORA should be able to convert a user-defined region of interest in the scanned image into a dose map, using a calibration curve made in Dose-response. A schematic overview of the tab Map dose can be seen in Figure 3.7.

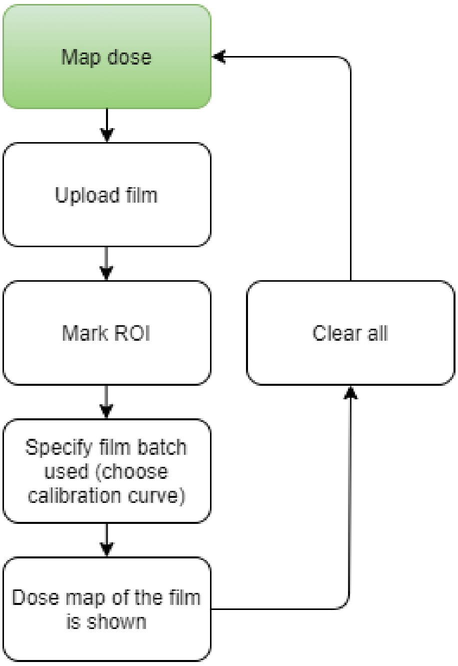

Map dose
Upload film
Mark ROI
Specify film batch used (choose calibration curve)
Dose map of the film is shown
Clear all

Figure 3.7: A schematic overview of the tab Map dose in FIDORA, which enables the user to upload a scanned film, choose a calibration curve and map the pixel values to dose values in a chosen ROI.

3.2.4 Profiles

The Profiles tab in FIDORA should be able to plot a profile of a user-defined region of interest in the scanned film, and map this to the corresponding region in the dose plan matrix. That is, one should be able to compare profiles in the film and in the dose plan matrix. This is particularly useful when investigating the build-up, where one can use the profile for evaluation. A schematic overview of the tab Profiles can be seen in Figure 3.8.

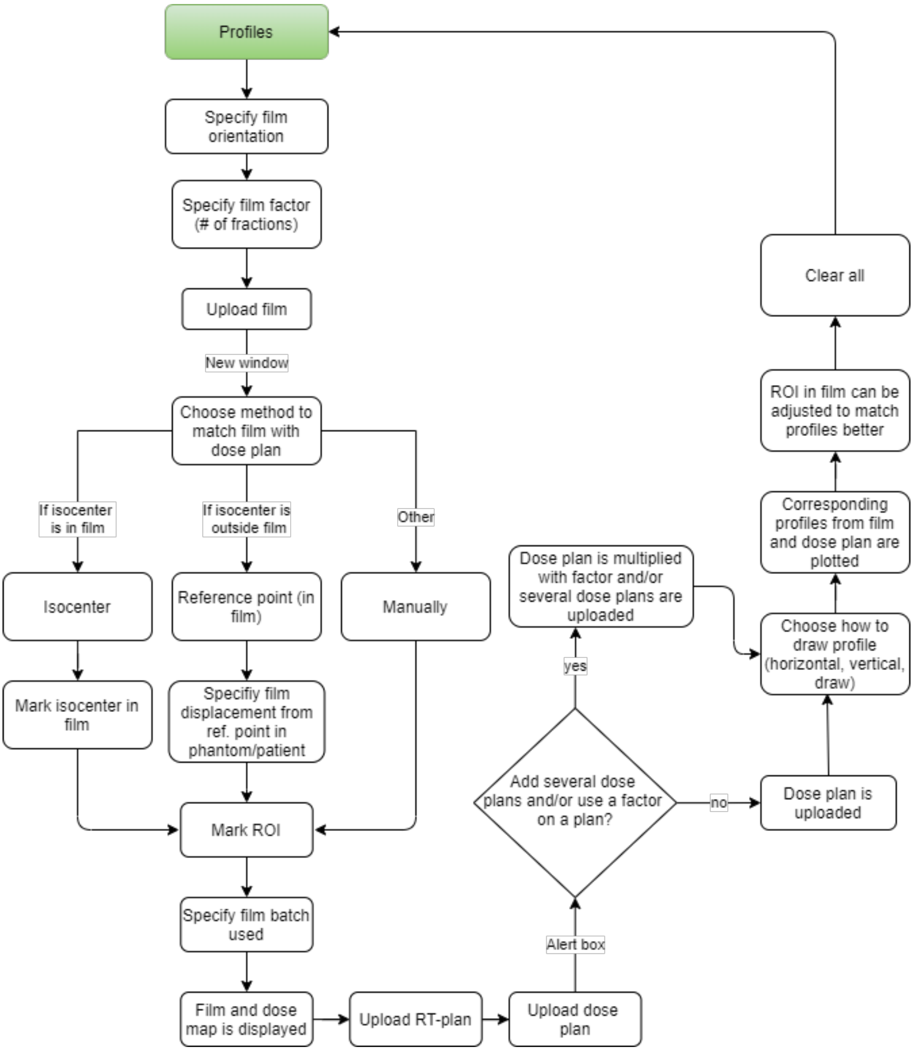

Profiles
Specify film orientation
Specify film factor (# of fractions)
Upload film
New window
Choose method to match film with dose plan
If isocenter is in film
If isocenter is outside film
Other
Isocenter
Reference point (in film)
Manually
Mark isocenter in film
Specifiy film displacement from ref. point in phantom/patient
Mark ROI
Specify film batch used
Film and dose map is displayed
Upload RT-plan
Upload dose plan
Alert box
Add several dose plans and/or use a factor on a plan?
yes
no
Dose plan is multiplied with factor and/or several dose plans are uploaded
Dose plan is uploaded
Choose how to draw profile (horizontal, vertical, draw)
Corresponding profiles from film and dose plan are plotted
ROI in film can be adjusted to match profiles better
Clear all

Figure 3.8: A schematic overview of the tab Profiles in FIDORA, which is responsible for plotting profiles that enables comparison of the film and the dose plan matrix along a profile of choice.

3.3 Experimental setup, treatment planning and field arrangements

3.3.1 Phantom

An anthropomorphic female thorax phantom was used throughout this project. The phantom is made of 18 transversal slices of RW3 with a density of 1.045g/cm^3. Each slice is 10 mm thick, and the lungs were represented by a material with density 0.28 g/cm^3 [1].

The entire phantom with GafChromic films positioned between adjacent slices are shown in figure 3.9. Figure 3.9 shows the setup used, where the phantom is placed in the head first supine (HFS) position. That is, head towards the gun, and with the back on the patient table. That yields that the right breast on the image is the phantom's left breast, and the left breast on the image is the phantom's right breast.

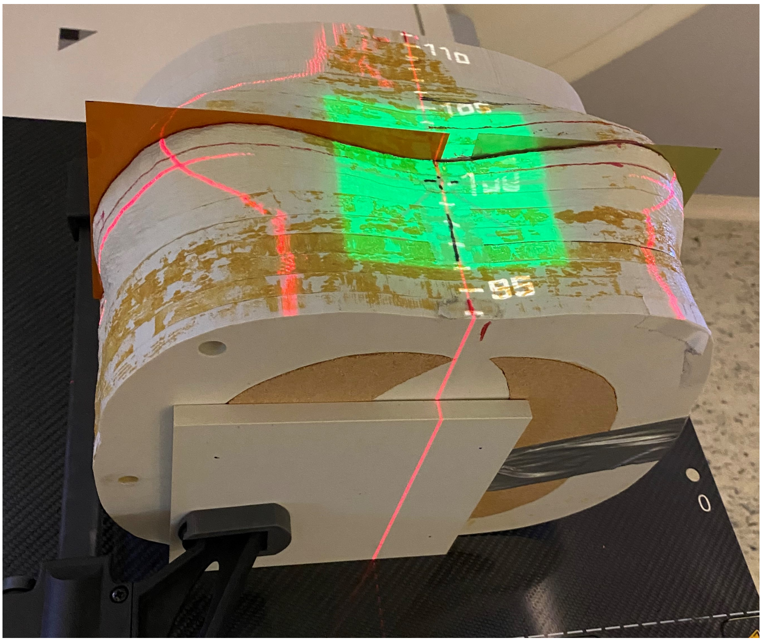


Figure 3.9: An anthropomorphic female thorax phantom is shown. The phantom is made of 18 transversal slices of RW3 with a density of 1.045g/cm^3. Each slice is 10 mm thick, and the lungs were represented by a material with density 0.28 g/cm^3 [1]. GafChromic films, EBT3 and XR-QA2 are here placed between adjacent transversal slices.

The GafChromic films used in the following experiments, XR-QA2 and EBT3, were positioned between two phantom slices, as shown in figure 3.10. The GafChromic EBT3 film was placed so that it covers the left breast, and the GafChromic XR-QA2 film was positioned so that it covers the contralateral (opposite) breast. Similarly, this was also done with the EBT3 film to investigate the dose to the CLB in some treatment plans. EBT3 film was positioned to cover the CLB in the same ways as the XR-QA2 film, only it was cut in a smaller piece. The setup with the EBT3 film is shown in Figure 3.11. Paper tape was used to fasten the films onto the phantom, as have been done in previous studies [1] and recommended in Handbook of X-ray imaging: Physics and technology [39]. A more durable tape was used to hold the phantom slices together, ensuring as little air gaps as achievable between the adjacent slices.

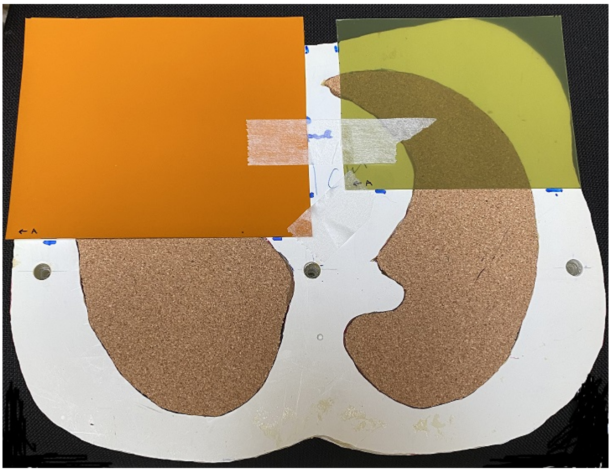


Figure 3.10: A transversal slice of the anthropomorphic female thorax phantom is shown. Here the GafChromic EBT3 film is positioned so that it covers the left breast, that is to be irradiated. The GafChromic XR-QA2 film is placed so that it covers the contralateral (opposite, here right) breast.

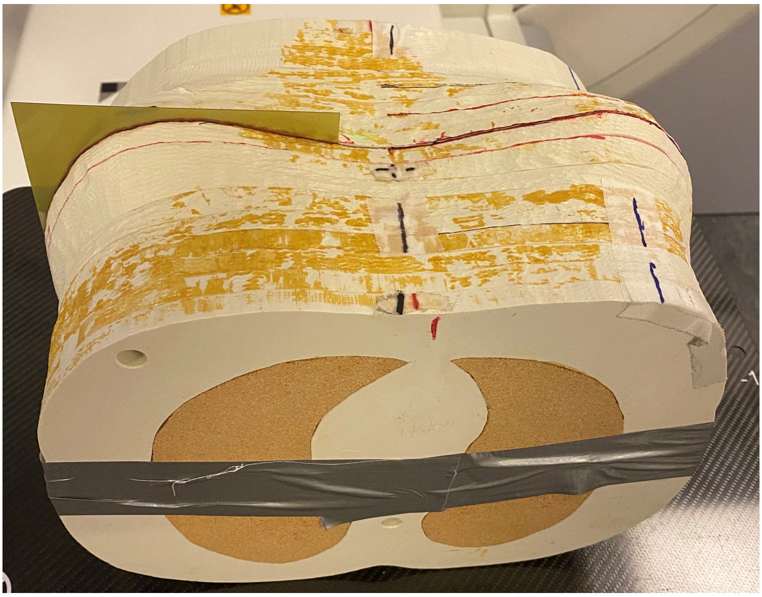


Figure 3.11: A transversal slice of the anthropomorphic female thorax phantom is shown. Here the GafChromic EBT3 film is positioned so that it covers the contralateral (opposite, here right) breast.

3.3.2 Volume definitions

The phantom was scanned with a Siemens Emotion CT scanner with 5 mm slice thickness and centre-centre spacing. The resulting CT image was then used to define the region of interest (ROI). The external contour as well as lungs were delineated, and a ROI was constructed by subtracting the lung volumes from the external contour. This ROI, being the entire phantom except from the lungs, was assigned a uniform density of 1.045g/cm^3. For a patient one would use the CT image to obtain the density values, but in this case, the phantom was constructed with a known material, RW3, with known density. CT imaging enables quite good density reconstruction, but it is never perfect. Therefore, it is more accurate to assign the density values oneself when dealing with a phantom of known material.

After the ROI was chosen a clinical target volume (CTV) and the contralateral breast (CLB) were delineated with 5 mm margin to the phantom surface and lung. A planning target volume (PTV) was created by adding margins of 10 mm to the CTV in all directions with two exceptions: in the posterior region and in the medial region margins of 5 mm were added. Then, the parts of the PTV that was closer than 5 mm from the outer contour was removed. Additional margins in the superficial region is not added to the PTV to account for breast swelling and deformation of the breast. Instead, a robust arrangement of treatment fields is chosen, delivering an open field extending into air at the superficial regions of the breast, as seen in Figure 3.12.

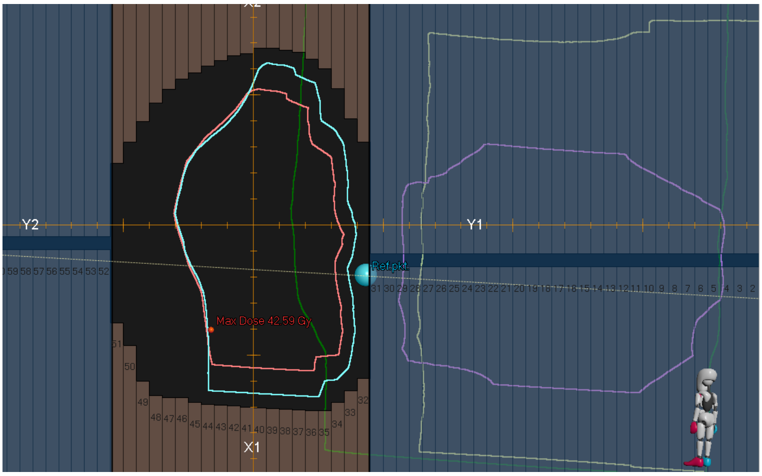


Figure 3.12: Beam's eye view of a lateral treatment field. This is a robust arrangement of an open tangential field extending into air at the superficial regions of the breast.

3.3.3 Irradiation techniques

Several different breast treatment plans were made in RayStation, based on the delineated volumes from the CT scan. The treatment plan consists of a combination of open tangential fields, tangential segments as well as VMAT arcs. Following choice of beam setup, the treatment plan is optimized to deliver the prescribed dose to the breast, and simultaneously constrict the absorbed dose to the OARs, such as the contralateral breast, lungs and heart.

Tangential standards

A standard tangential plan consisting of medial and lateral fields with aligned posterior field borders was made, as seen in Figure 3.13. The superficial parts of the breast were divided into three regions of interest, and these regions (medial, central and lateral) are illustrated in the isocentre plane in Figure 3.14. Additional tangential segments (field-in-field (FiF) technique) or VMAT arcs were used to achieve a homogeneous dose distribution inside the CTV. The treatment fields are summarized in Table 3.1. The treatment plans criteria used in optimization was that 95% of the prescribed dose should cover 98% of the PTV.

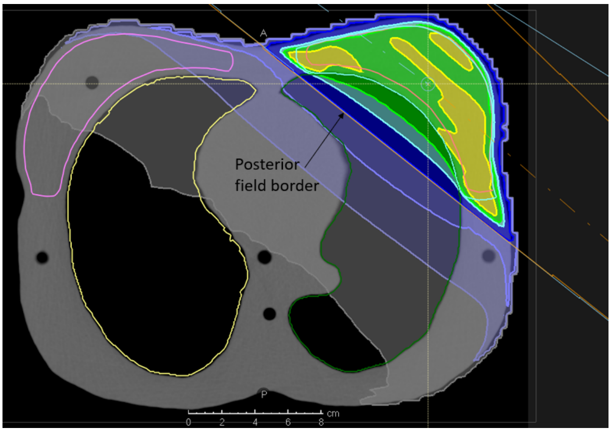
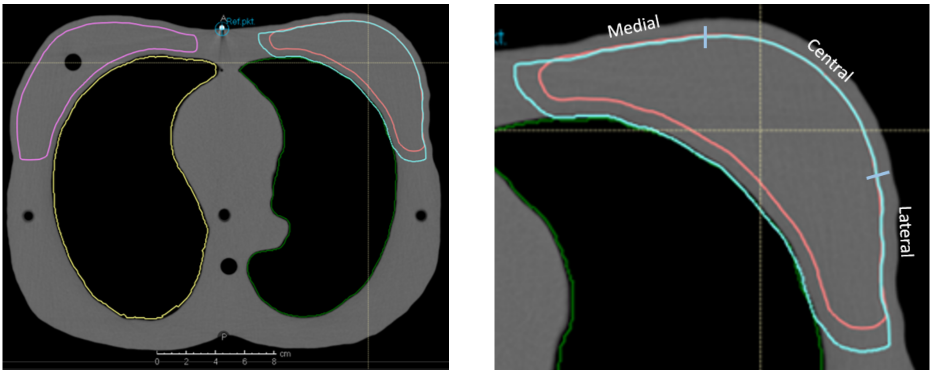


Figure 3.13: A standard tangential plan setup, consisting of medial (blue) and lateral (orange) fields with aligned posterior field borders.

Most treatment plans were transported and delivered without any errors. However, one of the treatment plans were subject to errors due to information lost in transportation between systems. Plan V3, described in Table 3.1, was supposed to have 5° couch angle, but during manual transport of files between systems, this information was lost. So, a couch angle of 0° was used instead.



(a) A CT image of the phantom used, including delineation of important regions of interest when irradiating the breast.

(b) The PTV of the left breast (blue) is divided into three segments: medial, central and lateral, as indicated in the figure.

Figure 3.14: The superficial parts of the breast were divided into three regions of interest (medial, central and lateral) which are illustrated in the isocentre plane.

Treatment fields							
Plan name	Tangential open fields	Tangential segments	VMAT arcs (# segments per arc)	VMAT arcs, gantry angle. Start-stop	Collimator angle	Couch angle	Filtered/ FFF beam
V1 - tangential FiF (field-in-field)	1 medial, 1 lateral	1 lateral, 2 medial			0°	5°	Filtered
V2 - tangential FiF 90col	1 medial, 1 lateral	1 lateral, 2 medial			90°	5°	Filtered
V3* - hybrid VMAT	1 medial, 1 lateral		1 lateral, 1 medial (25)	131°-101°, 336°-306°	0°	0°*	Filtered
V5 - VMAT short arcs 0col			1 medial, 1 lateral (37)	346°-296°, 161°-111°	5°	5°	Filtered
V6 - VMAT short arcs 90col			1 medial, 1 lateral (37)	346°-296°, 161°-111°	90°	5°	Filtered
V7 - VMAT FFF short arcs			1 medial, 1 lateral (37)	346°-296°, 161°-111°	90°	5°	FFF
V8 - Medial FFF			3 medial, 1 central (25)	296°-320°, 330°-296°, 355°-325°, 20° - 355°	90°	5°	FFF

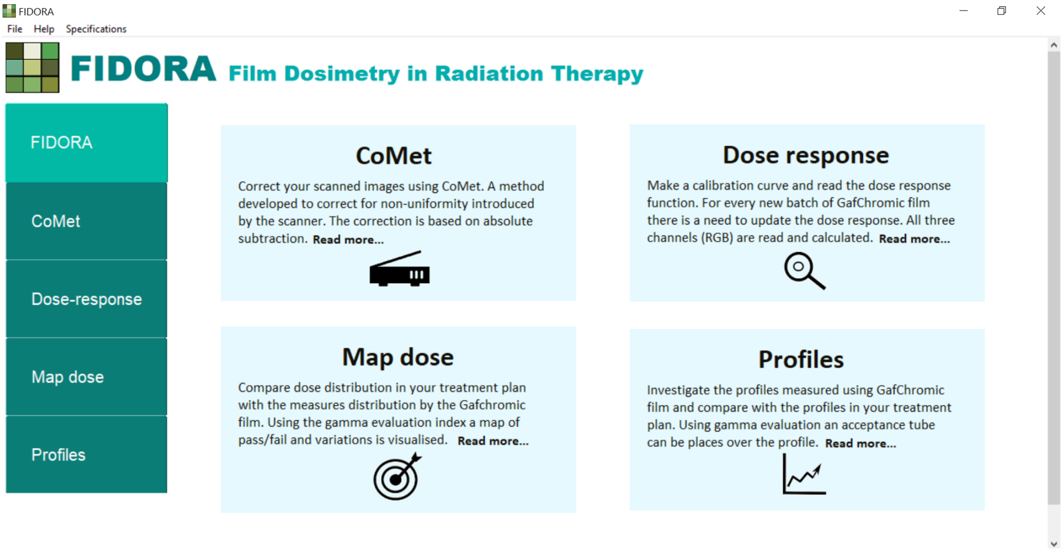
Table 3.1: Tangential treatment fields used in this project. The treatment plans are a combination of open tangential fields plus additional medial and lateral segments or VMAT arcs to give better dose coverage. *Plan V3 was supposed to have 5° couch angle, but during manual transport of files between systems, this information was lost. So, a couch angle of 0° was used instead. Plan V5 used a collimator angle set to 5° to avoid tongue-and-groove effect.

Chapter 4

Results

4.1 FIDORA

Figure 4.1 show a print screen of the film dosimetry software, FIDORA, developed during this project. The program presented in this project is the current version of the Python based, open source software developed by the author and another physics student, Stine Gustavsen.



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. **Read more...**

Dose response

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

Map dose

Compare dose distribution in your treatment plan with the measures distribution by the Gafchromic film. Using the gamma evaluation index a map of pass/fail and variations is visualised. **Read more...**

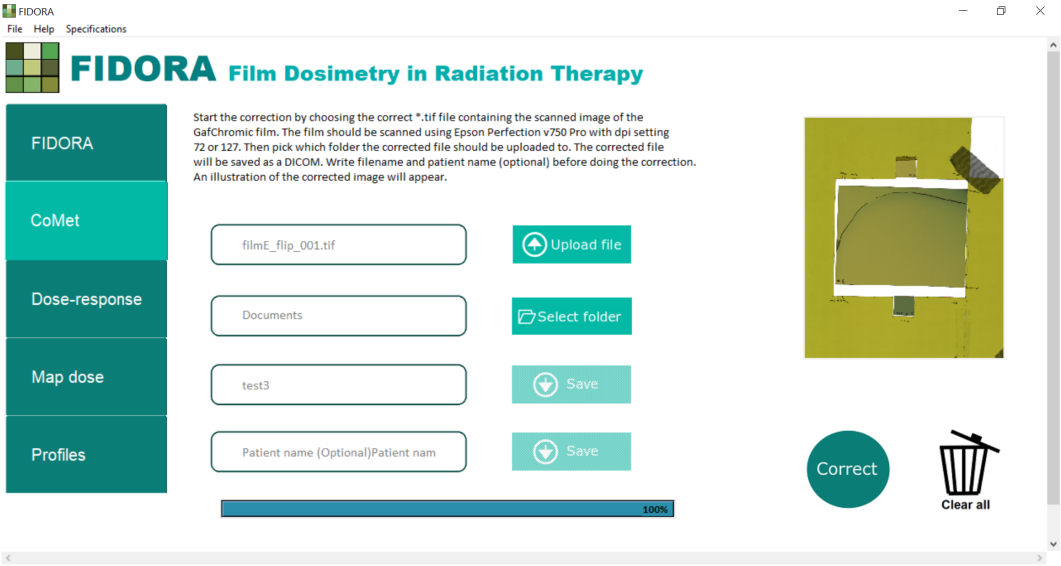
Profiles

Investigate the profiles measured using GafChromic film and compare with the profiles in your treatment plan. Using gamma evaluation an acceptance tube can be places over the profile. **Read more...**

Figure 4.1: A print screen of the film dosimetry software, FIDORA, developed during this project.

4.1.1 CoMet

The CoMet tab in FIDORA employs the dose independent lateral correction in each color channel as seen in Figure 4.13. The correction is represented as a 3D matrix that is stored within FIDORA and is subtracted from the scanned image uploaded in FIDORA. The CoMet (Correction Method) tab enables a user to upload an image of a scanned film, and performs a correction on the image, as shown in Figure 4.2. The absolute correction matrix that is subtracted from the uploaded image is small compared to the actual pixel values of the image, and is impossible for a user to detect visually. An illustration of the corrected image (with poor resolution) will appear to the user, but the corrected image (with full resolution), is stored in a desired folder, indicated by the user.



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 corrected image will appear.

filmE_flip_001.tif

Documents

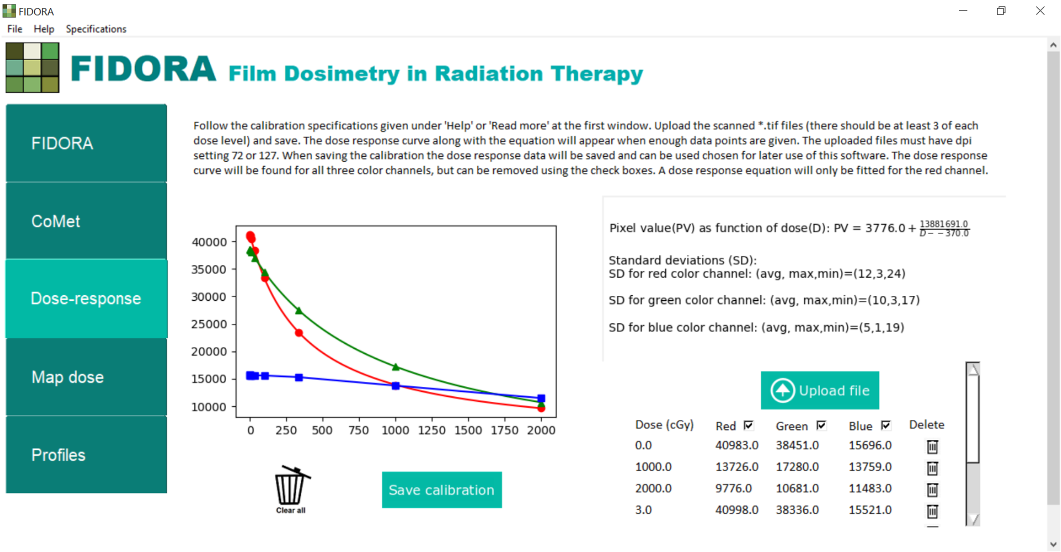
test3

Patient name (Optional)Patient nam

Figure 4.2: A print screen of the CoMet tab in FIDORA. The CoMet tab enables a user to upload an image of a scanned film, and performs a correction on the image.

4.1.2 Dose-response

The dose-response tab enables a user to upload known calibration films with known reference doses, scanned in the center of the scanner, and makes a calibration curve based on the formula, $d_x(D) = a + b/(D-c)$, as shown in Figure 4.3. $d_x(D)$ is the optical density of the film in scanner channel x at dose D, and a, b and c are the equation parameters to be fitted.



Follow the calibration specifications given under 'Help' or 'Read more' at the first window. 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 used 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.

Pixel value(PV) as function of dose(D): PV = $3776.0 + \frac{13881691.0}{D - -370.0}$

Standard deviations (SD):
SD for red color channel: (avg, max,min)=(12,3,24)

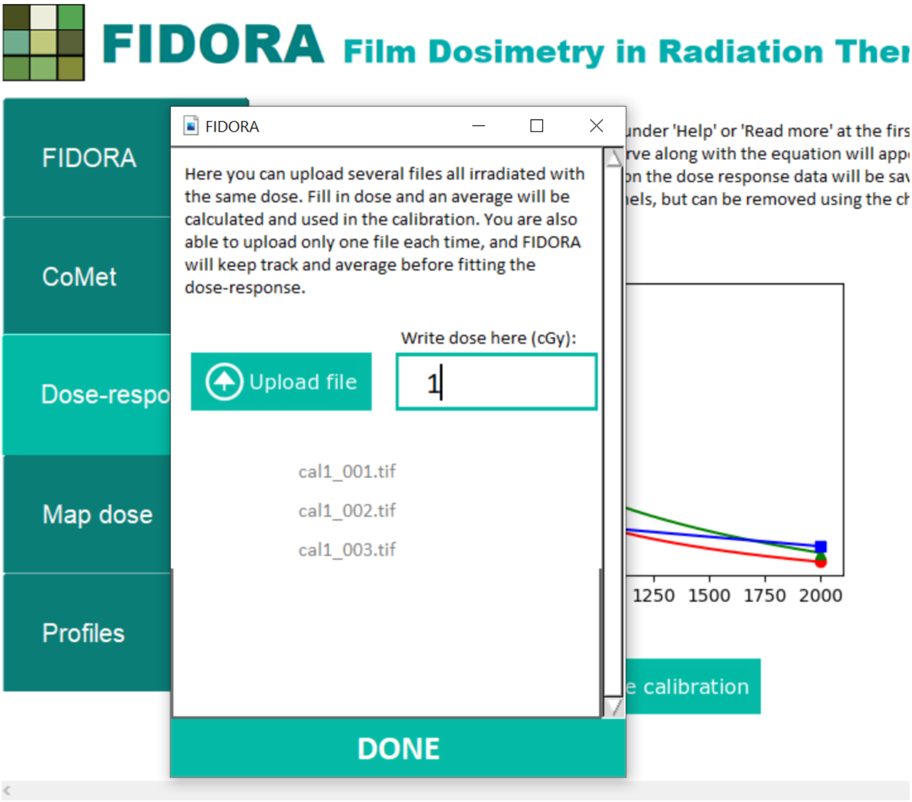
SD for green color channel: (avg, max,min)=(10,3,17)

SD for blue color channel: (avg, max,min)=(5,1,19)

Dose (cGy)	Red ☑	Green ☑	Blue ☑	Delete
0.0	40983.0	38451.0	15696.0	🗑
1000.0	13726.0	17280.0	13759.0	🗑
2000.0	9776.0	10681.0	11483.0	🗑
3.0	40998.0	38336.0	15521.0	🗑

Figure 4.3: A print screen of the dose-response tab in FIDORA. The dose-response tab enables a user to upload known calibration films with known reference doses, scanned in the center of the scanner, and makes a calibration curve based on the formula, d$_x$(D) = a + b/(D-c). d$_x$(D) is the optical density of the film in scanner channel x at dose D, and a, b and c are the equation parameters to be fitted.

In the dose-response tab, the user can upload scanned images of film irradiated with known doses. When clicking ”upload file” another window will appear, as shown in Figure 4.4. The user indicates the reference dose, in units of cGy, and has the option to upload one or more scanned images with the given reference dose. If more than one image is chosen for a given reference dose, an average of the images pixel values is used in the calibration, and this will contribute in the calculation of standard deviations between multiple scans of a reference film at a given dose level, within each color channels. Thus, the standard deviation gives an indication of the scan-to-scan variation among multiple scans of the same reference dose.



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): 1

cal1_001.tif

cal1_002.tif

cal1_003.tif

ınder 'Help' or 'Read more' at the firs
rve along with the equation will app
on the dose response data will be sav
ıels, but can be removed using the cł

Figure 4.4: A print screen of a tab within the dose-response tab in FIDORA, that enables the user to upload reference doses. The user indicates the reference dose, in units of cGy, and has the option to upload one or more scanned images with the given reference dose. If more than one image is chosen for a given reference dose, an average image is used in the calibration, and this will contribute in the calculation of standard deviations between multiple scans of a reference film at a given dose level, within each color channels.

4.1.3 Map dose

The tab Map dose in FIDORA is responsible for showing a dose map of an uploaded image of a scanned film. The tab enables the user to upload a scanned film and choose the region of interest that will later become the region which is mapped to dose values. The tab uses one of the available calibration curves, that is made in advance in the tab Dose-response, as seen in Figure 4.6. The Map dose tab is shown in Figure 4.5.

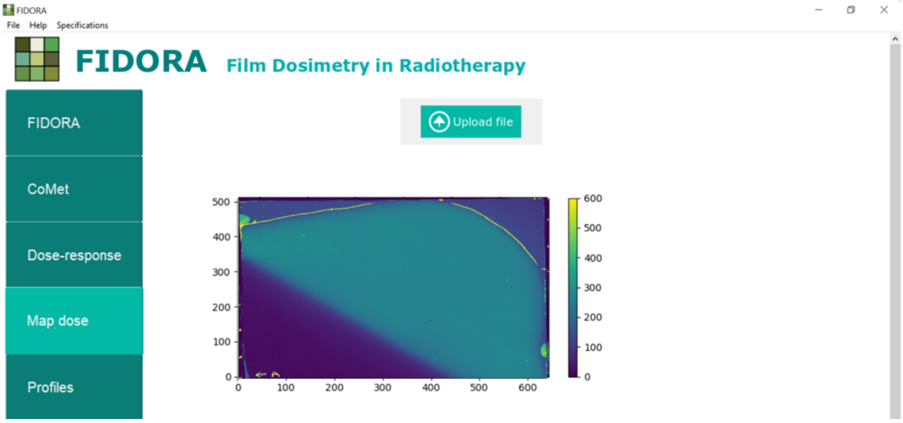
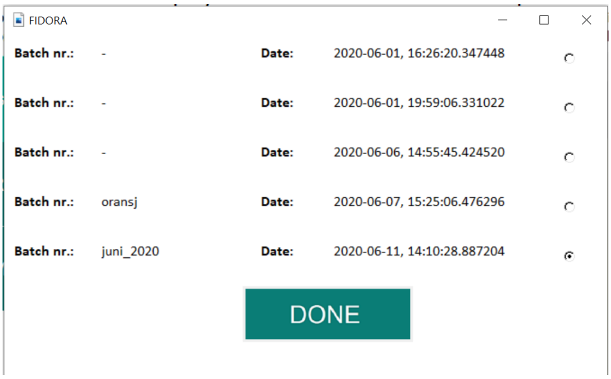


Figure 4.5: A print screen of the Map dose tab in FIDORA. The Map dose tab enables the user to upload an image of a scanned film and will map it to doses, using an available calibration curve made in the tab Dose-response.

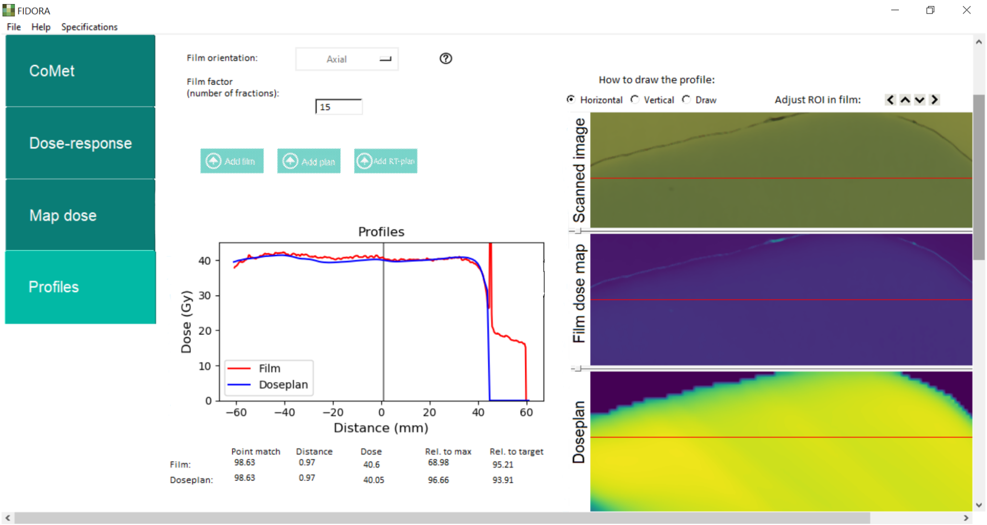


Batch nr.:	-	**Date:**	2020-06-01, 16:26:20.347448	○
Batch nr.:	-	**Date:**	2020-06-01, 19:59:06.331022	○
Batch nr.:	-	**Date:**	2020-06-06, 14:55:45.424520	○
Batch nr.:	oransj	**Date:**	2020-06-07, 15:25:06.476296	○
Batch nr.:	juni_2020	**Date:**	2020-06-11, 14:10:28.887204	●

Figure 4.6: A print screen of the pop-up window that enables the user to choose one of the calibration curves made in the Dose-response tab in FIDORA.

4.1.4 Profiles

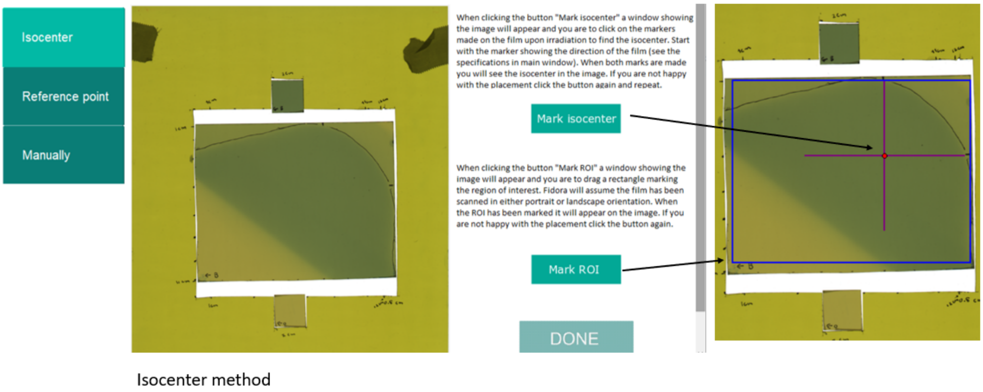
The Profiles tab in FIDORA enables the user to plot a profile of a user-defined region of interest (ROI) in the scanned film, and map this to the corresponding region in the dose plan matrix, as shown in Figure 4.7. Since the film is scanned with a better resolution (with 127 dpi the resolution is 0.2mm/pixel) than what is used in the plan optimization (at least 1x1x1 mm/voxel), the difference in resolution must also be mapped before comparing profiles. After the ROI and resolution is matched for the film and the dose plan matrix, type of profile can be chosen and will be plotted. The user has to choose between horizontal, vertical or a ”draw”- function to make the profiles. The horizontal and vertical profiles can be adjusted in the ROI, while the ”draw”-function enables the user to draw an arbitrary line anywhere in the ROI. After choosing type of profile, the profile along the chosen line will be plotted for both film and dose plan. Due to uncertainties in positioning of the film in the phantom and in the scanner, there is an option to adjust the chosen ROI so that the profiles match better. This can be done by the user, which has the possibility to move the ROI to the left, right, up or down, and can also change the ROI back to its original position.



	Point match	Distance	Dose	Rel. to max	Rel. to target
Film:	98.63	0.97	40.6	68.98	95.21
Doseplan:	98.63	0.97	40.05	96.66	93.91

Figure 4.7: A print screen of the Profiles tab in FIDORA. The Profiles tab enables the user to upload a scanned film, and compare it to the corresponding dose plan. The user can chose between horizontal, vertical and user-defined profiles. The profiles are drawn in the dose map of the film, and is mapped to the dose plan, so that the corresponding region can be evaluated along a profile of choice. The image of the ”scanned image”, ”film dose map” and ”dose plan” can be dragged up and down to be displayed more or less.

In order to evaluate the film and the dose plan, the coordinate systems of the two must be matched. The user can choose between different methods to do so. The isocenter method is shown in Figure 4.8, and the reference point method is shown in Figure 4.9.

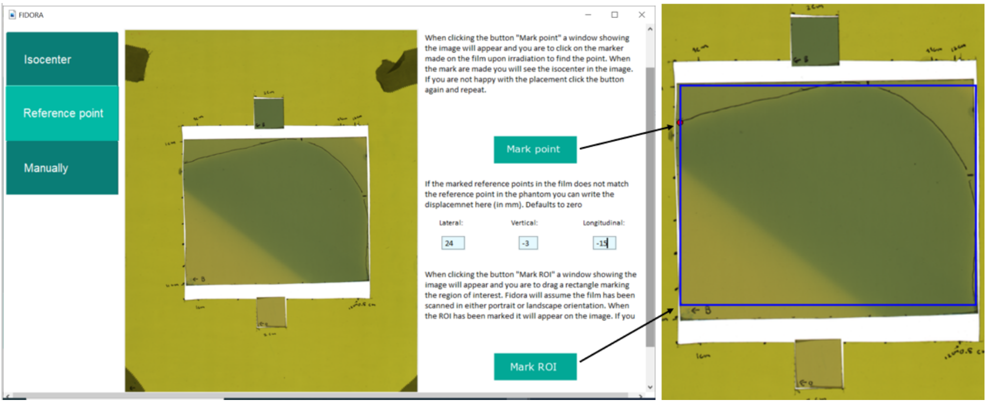


When clicking the button "Mark isocenter" a window showing the image will appear and you are to click on the markers made on the film upon irradiation to find the isocenter. Start with the marker showing the direction of the film (see the specifications in main window). When both marks are made you will 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.

Isocenter method

Figure 4.8: A print screen of the isocenter method in the Profiles tab in FIDORA.



When clicking the button "Mark point" a window showing the image will appear and you are to click on the marker made on the film upon irradiation to find the point. When the mark are made you will see the isocenter in the image. If you are not happy with the placement click the button again and repeat.

If the marked reference points in the film does not match the reference point in the phantom you can write the displacemnet here (in mm). Defaults to zero

Lateral: 24

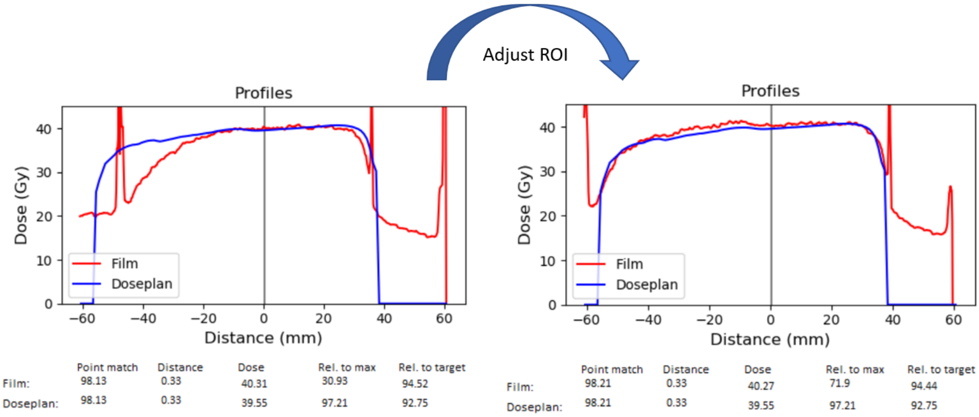
Vertical: -3

Longitudinal: -15

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

Figure 4.9: A print screen of the reference point method in the Profiles tab in FIDORA.

When a method is chosen to match the film with the dose plan, and associated RT-plan and dose plan are uploaded, profiles can be drawn by the user. When a profile is drawn (horizontal, vertical or draw-function), a plot of the values along the profile will appear. Due to errors in positioning, an option to adjust the ROI in the film is given to the user, as shown in Figure 4.10. This can be done by pressing buttons (up, down, left, right) which effectively shifts the ROI in the film one pixel in the chosen direction. There is also an option to move the ROI back to the original position, by the button ”original”. One can adjust the ROI to see where the film and dose plan has a better match. The degree of matching can be viewed by observing the overlapping profiles, but can also be read from specific values in the table below the plot. Such values are ”point match”, ”dose”, ”relative to maximum in ROI” and ”relative to target”. Point match indicates how similar the dose is in a given position, which is chosen by hoovering over the plot. Relative to maximum ROI indicates the relative dose at the given point, compared to the maximum dose at the chosen profile. This must be used with care, since the use of a permanent marker might give high dose spikes if it intercepts the profile. Relative to target indicates the relative dose compared to the target dose, and will in most cases be the most interesting parameter to evaluate.



	Point match	Distance	Dose	Rel. to max	Rel. to target
Film:	98.13	0.33	40.31	30.93	94.52
Doseplan:	98.13	0.33	39.55	97.21	92.75

	Point match	Distance	Dose	Rel. to max	Rel. to target
Film:	98.21	0.33	40.27	71.9	94.44
Doseplan:	98.21	0.33	39.55	97.21	92.75

Figure 4.10: A print screen of the plot of the profiles along a user-defined line, with values of interest displayed, in the Profiles tab in FIDORA. When hoovering over the plot, the table below the plot will list values that corresponds to the x-value that is hoovered over. The grey, vertical line, indicates the x-position that is chosen by hoovering over with the mouse. If the profiles have a poor match, one can adjust the ROI in the film, and see if the profiles match better.

4.2 Experimental results

4.2.1 Correction method

The correction method provides a correction for the anisotropic light conditions in the scanner. Figure 4.11 and 4.12 shows the deviation in pixel value from the center of the scanner surface for the red color channel, in the scanner surface in the lateral direction and in the scanning direction, respectively. It can be seen that the absolute deviation in intensity is higher towards the edges, but the systematic relation to dose is low. There is a tendency of higher pixel value variation relative to the center for higher doses in the lateral direction, as seen in Figure 4.11, but this is not consistent for the entire scanning area that was investigated. As a result it was decided that the correction matrix would be made independent on dose, so an average was made. A lateral profile of the resulting correction matrix (that will correct both in the lateral and scanning direction of the scanner surface) is shown is Figure 4.13.

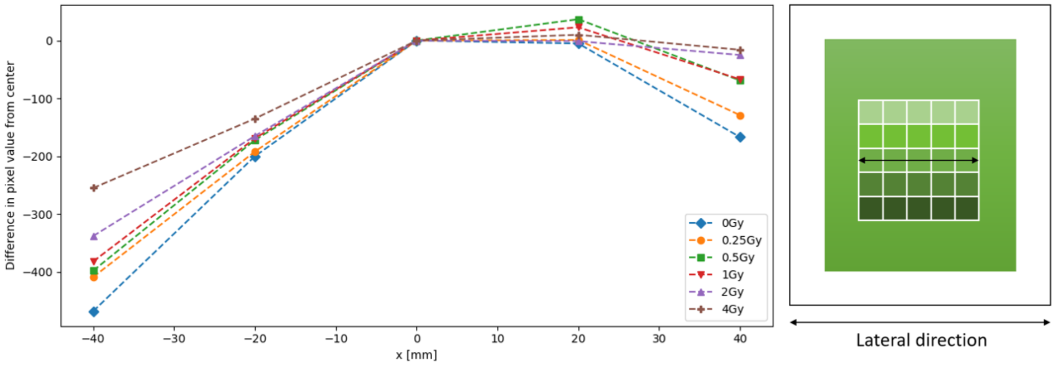


Figure 4.11: Profiles of the deviation in pixel value compared to the center across the scanner surface in lateral direction for different dose levels.

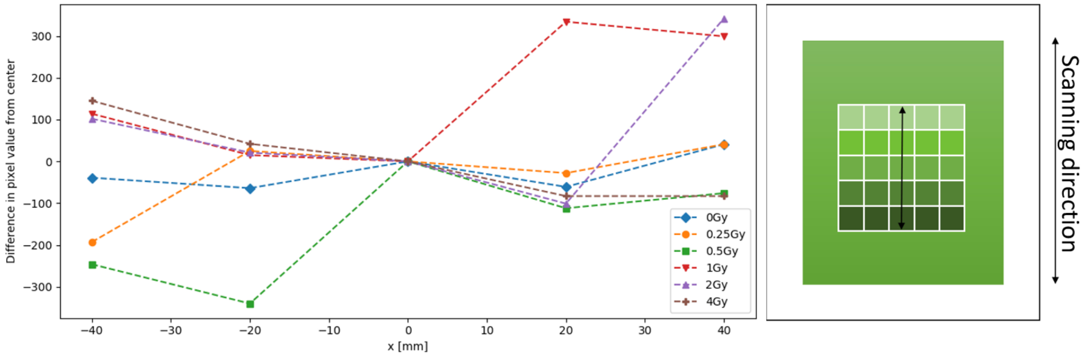


Figure 4.12: Profiles of the deviation in pixel value compared to the center across the scanner surface in scanning direction for different dose levels.

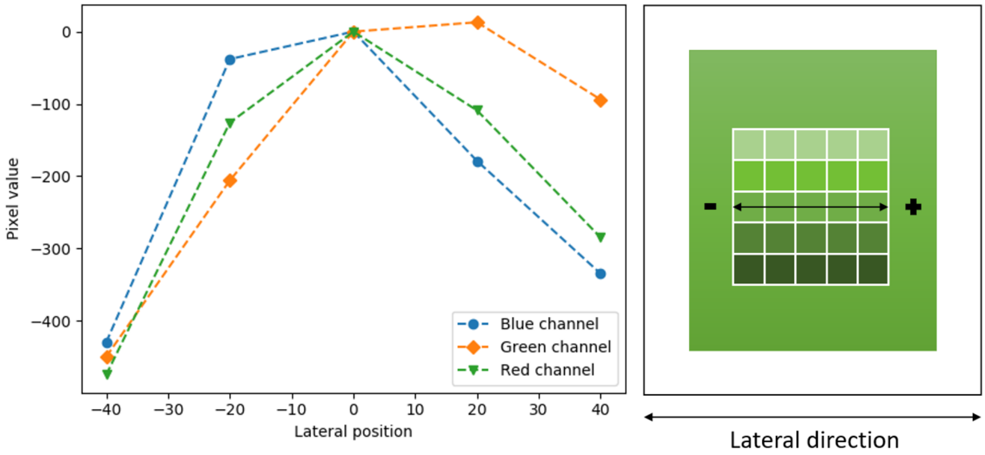


Figure 4.13: A lateral profile, which is the average of all doses obtained by the method described in Section 3.1.3, is shown for all color channels. This profile is shown to illustrate the resulting correction matrix that will be used to correct the investigated non-uniform read-out over the scanning surface.

The correction method that is used in FIDORA is based on the average differences in intensity for different doses over the investigated scanner surface, as illustrated in Figure 4.13. The investigated scanner surface was a 10x10cm area at the center of the scanner, where a grid of 25 correction values were obtained from the method described in Section 3.1.3, and the intermediate correction values were obtained by cubic interpolation. Also, extrapolation were used to obtain correction values outside the 10x10cm investigated scanning area, resulting in a correction matrix that corrects for a 12x12cm area in the center of the scanner surface.

4.3 Experimental results obtained with FIDORA

4.3.1 Calibration curves

The Dose-response tab in FIDORA was used to establish calibration curves of interest. Only the red color channel is used in further evaluations of the calibration curves, as this color channel demonstrates the largest change in optical density due an absorption maximum centered around wavelengths corresponding to red, as shown in Figure 2.12. Three scans of each reference dose is used, so that the calibration curves employs the average scanner read-out of three successive scans. This is done to reduce the scan-to-scan influence, and the resulting standard deviations between successive scans of the same reference dose are indicated for the red color channel. To reduce the influence of daily variations in the linac, a new calibration curve was made each of the two days the experiments went on. The calibration curves as well as the experiments were established over the course of two different days, so applying a calibration curve established the same day as the experiment was conducted is therefore relevant.

- One filtered calibration curve was made for GafChromic EBT3 film and GafChromic XR-QA2 film during the first day, and can be seen in Figure 4.14 and 4.17, respectively. At this day the build-up dose to the target breast was investigated using EBT3, along with the dose to the CLB using XR-QA2 film.
- One filtered and one filter-free calibration curve was made for the GafChromic EBT3 film during the second day, and can be seen in Figure 4.15 and 4.16, respectively. At this day the dose to the CLB was investigated using XR-QA2.

4.3.2 Calibration curve for GafChromic EBT3 film

The calibration curve (obtained the second day of experiments) from nine reference doses at 0, 1, 3, 10, 33, 100, 333, 1000 and 2000 cGy using a filtered radiation beam can be seen in Figure 4.15. The calibration curve (from second day of experiments) obtained from equivalent reference doses using a filter-free radiation beam can be seen in Figure 4.16.

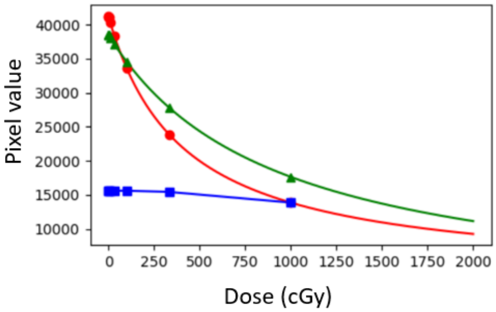


Figure 4.14: Calibration curve (from first day of experiments) obtained from eight reference doses at 0, 1, 3, 10, 33, 100, 333 and 1000 cGy using a filtered radiation beam, and GafChromic EBT3 film. The red, green and blue fitted lines indicates the red, green and blue color channels, respectively. The horizontal axis holds the doses in cGy, and the vertical axis holds the pixel value (PV). The calibration curve is established using FIDORA, and the resulting equation is PV = 2831+15497108/(D-(-403)).

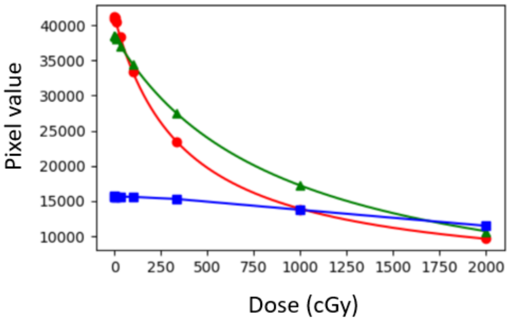


Figure 4.15: Calibration curve (from second day of experiments) obtained from nine reference doses at 0, 1, 3, 10, 33, 100, 333, 1000 and 2000 cGy using a filtered radiation beam, and GafChromic EBT3 film. The red, green and blue fitted lines indicates the red, green and blue color channels, respectively. The horizontal axis holds the doses in cGy, and the vertical axis holds the pixel value (PV). The calibration curve is established using FIDORA, and the resulting equation is PV = 3776+13881691/(D-(-370)).

The calibration established for GafChromic EBT3 film for the filtered radiation beam (at first day of experiments) yielded a calibration on the form:

$$PV = 2831 + \frac{15497108}{D - (-403)}, \tag{4.1}$$

Likewise, the calibration established for GafChromic EBT3 film for the filtered radiation beam (at second day of experiments) yielded a calibration on the form:

$$PV = 3776 + \frac{13881691}{D - (-370)}, \tag{4.2}$$

where PV is pixel value and D is the absorbed dose. The parameters were obtained by curve fitting of the parameters in Equation 3.1. The associated standard deviations resulting from multiple (3) scans of the filtered reference doses are calculated in FIDORA. For the red color channel the average, minimum and maximum standard deviations (SD) are:

$$SD_{red}(avg, min, max) = (21, 1, 47) \tag{4.3}$$

for the first day of experiments, and

$$SD_{red}(avg, min, max) = (12, 3, 24) \tag{4.4}$$

for the second day of experiments.

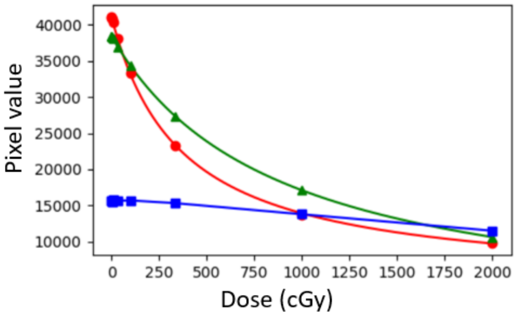


Figure 4.16: Calibration curve obtained from nine reference doses at 0, 1, 3, 10, 33, 100, 333, 1000 and 2000 cGy, each scanned three times, using a filter-free radiation beam, and GafChromic EBT3 film. The red, green and blue fitted lines indicates the red, green and blue color channels, respectively. The horizontal axis holds the doses in cGy, and the vertical axis holds the pixel value (PV). The calibration curve is established using FIDORA, and the resulting equation is PV = 4037+13482787/(D-(-363)).

The calibration established for GafChromic EBT3 film for the filter-free radiation beam yielded a calibration on the form

$$PV = 4037 + \frac{13482787}{D - (-363)} \tag{4.5}$$

The associated standard deviations resulting from multiple (3) scans of the filter-free reference doses are calculated in FIDORA. For the red color channel the average, minimum and maximum standard deviations (SD) are:

$$SD_{red}(avg, min, max) = (18, 2, 29) \tag{4.6}$$

To illustrate the variation between the filtered (Figure 4.15) and filter-free (Figure 4.16) calibration curve established at the second day of experiments, a calculation example follows. For a reference dose, D, of 200 cGy, the resulting difference in pixel values, PV, between the different calibration curves established at the second day of experiments is:

$$PV_{filtered}(200cGy) - PV_{filter-free}(200cGy) = 28130 - 27985 = 145 \tag{4.7}$$

This absolute difference between the two calibration curves corresponds to approximately 0.5% of the PV$_{filtered}$ value. Likewise, the differences in calibration curves will lead to a difference in the interpreted dose:

$$D_{filtered}(28130) - D_{filter-free}(28130) = 200\text{cGy} - 197\text{cGy} = 3\text{cGy} \tag{4.8}$$

This absolute difference yields an approximate 1.5% difference in interpreted dose value between the two calibration curves.

4.3.3 Calibration curve for GafChromic XR-QA2 film

The calibration curve for GafChromic XR-QA2 film established from nine reference doses at 0, 1, 3, 10, 33, 100 and 333 cGy, each scanned three times, using a filtered radiation beam can be seen in Figure 4.17.

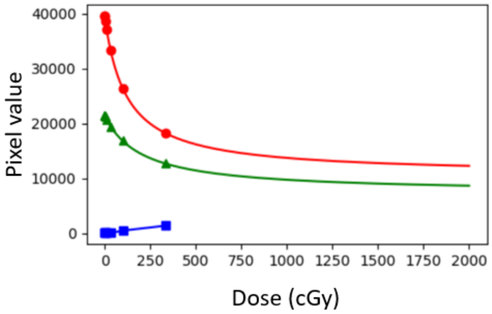


Figure 4.17: Calibration curve obtained from nine reference doses at 0, 1, 3, 10, 33, 100 and 333 cGy using a filtered radiation beam and GafChromic XR-QA2 film. The red, green and blue fitted lines indicates the red, green and blue color channels, respectively. The horizontal axis holds the doses in cGy, and the vertical axis holds the pixel value (PV). The calibration curve is established using FIDORA, and the resulting equation is PV = 10642+3418601/(D-(-118)).

The calibration established for GafChromic XR-QA2 film for the filtered radiation beam yielded a calibration on the form:

$$PV = 10623 + \frac{3425872}{D - (-110)} \tag{4.9}$$

The standard deviations resulting from multiple (3) scans of the filtered reference doses are calculated in FIDORA. For the red color channel the average, minimum and maximum standard deviations (SD) are:

$$SD_{red}(avg, min, max) = (11, 5, 18) \tag{4.10}$$

4.3.4 Validation of calibration curve

In total, four different calibration curves were made in the course of two different days. The deviation between the reference doses and the doses obtained by applying the calibration curves on the known reference doses, seen in Figure 4.14, 4.15, 4.16 and 4.17, can be seen in Table 4.1.This table evaluates the calibration curves using reference doses that are scanned in the middle of the scanner surface to avoid the influence of a non-uniform scanner readout.

Deviation between reference doses and doses obtained from calibration curves				
Reference doses (cGy)	EBT3 filtered (cGy) Day 1	EBT3 filtered (cGy) Day 2	EBT3 FFF (cGy) Day 2	XR-QA2 filtered (cGy) Day 1
0	2.06	3.09	1.93	-0.44
1	-0.09	0.08	0.52	-0.84
3	2.34	2.94	3.69	3.64
10	9.87	8.94	8.62	10.11
33	32.94	31.88	32.08	31.99
100	100.28	98.23	97.81	99.46
333	332.99	334.58	335.55	332.84
Standard deviation (0-333cGy)	0.75	0.94	0.80	0.59
1000	1000.72	1025.14	1021.13	
Standard deviation (0-1000 cGy)	0.69	8.45	7.00	
2000		1943.62	1948.47	
Standard deviation (0-2000 cGy)		19.06	17.17	

Table 4.1: The table shows the reference doses used to establish various calibration curves, and the associated doses that are obtained by applying the calibration curves on scanned GafChromic films radiated to reference doses, using only the red color channel. Also, the standard deviation between the reference dose and the values obtained by the calibration curves are given for reference doses starting at 0 cGy and up to 333 cGy, 1000 cGy and 2000 cGy, respectively. The GafChromic XR-QA2 film does not include any higher calibration points than 333 cGy, since this would extend quite far beyond the dynamic range of this film [6].

The calibration curves were also evaluated when being applied to the investigation of the dose in the various treatment plans. Since the films used in these experiments were cut into much larger pieces than the calibration films (2x2cm), the influence of the non-uniform readout over the scanner-surface was introduced as an additional factor. For film measurements at the contralateral breast (CLB) it was found that the calibration curves for the GafChromic EBT3 film in regions on the film at approximately 0Gy in the dose plan, resulted in a measure of dose varying between -0.02Gy and 0.05Gy. While the calibration curve made for the GafChromic XR-QA2 film in regions on the film corresponding to approximately 0Gy in the dose plan, resulted in a measure of dose varying between 0Gy and -0.5Gy. This error is an order of magnitude larger than that of the EBT3 film. Horizontal profiles of the CLB measurements, employing the calibration curve for the GafChromic XR-QA2 film of treatment plans, V1, V2, V6 and V7 (see Table 3.1) can be seen in Figure 4.18 as a demonstration of this calibration curve. Due to the significant underestimation of dose observed in Figure 4.18 at certain areas, this calibration curve will not be used in further investigation of dose to the contralateral breast. The grey line in the profile showing treatment plan V2, in Figure 4.18, shows an area where the calibration curve along with the scanned XR-QA2 film measures the dose to be -0.50Gy.

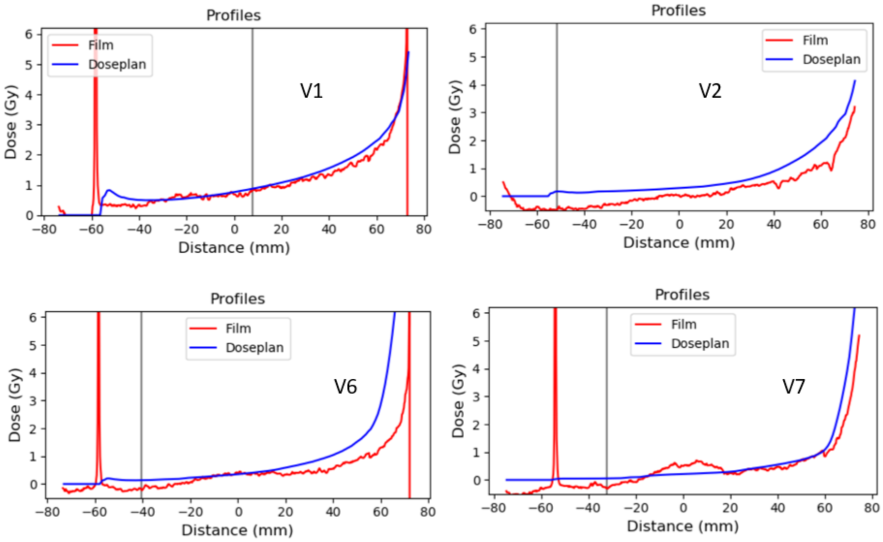


Figure 4.18: Horizontal profiles employing the calibration curve for the GafChromic XR-QA2 film of treatment plans, V1, V2, V6 and V7, as described in Table 3.1. Using GafChromic XR-QA2 film with its associated calibration curve, there is a significant underestimation of dose observed at certain areas. The grey line in the profile of V2 demonstrates a region where the calibration curve measures the dose to be -0.50 Gy.

4.3.5 Build-up in target breast

Build-up dose to the target breast was investigated using the Profiles tab in FIDORA, employing the filtered calibration curve that was irradiated the same day as the experiment was conducted.

The build-up was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. To ensure that the profile measures a depth-dose distance in calculating the build-up, the profiles were drawn perpendicular to the surface of the breast, with incidence on the medial, central and lateral segment of the breast, as seen in Figure 4.19 .

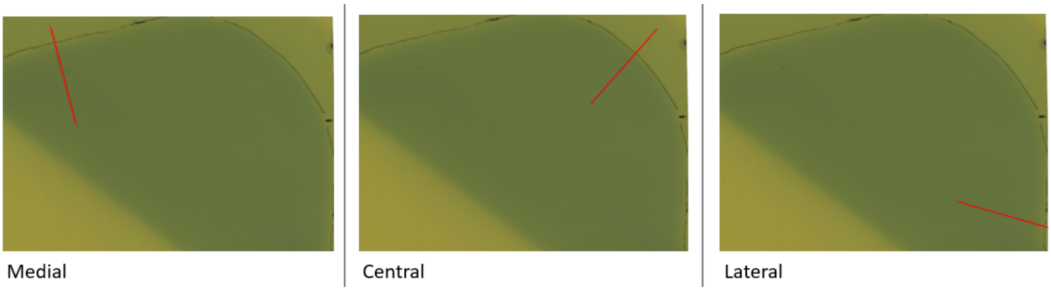
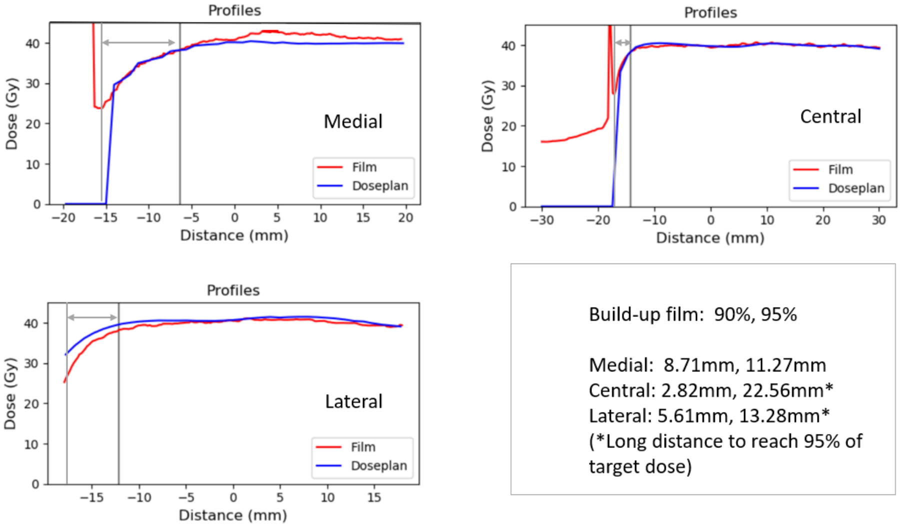


Figure 4.19: Profiles taken at medial, central and lateral incidence on the target breast in the isocentre plane, shown as a red line in the corresponding images.

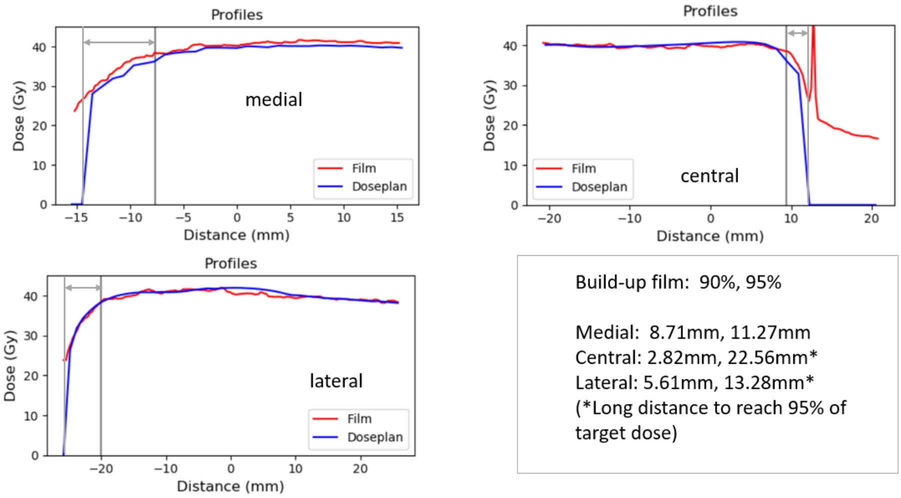
The Profiles tab in FIDORA was used to plot a profile at a desired region in the film, and mapped this to the corresponding profile in the dose plan. This enabled a comparison between the build-up distance at the medial, central and lateral incidence of the target breast, calculated in the film and in the dose plan, as seen in Figure 4.20, 4.21, 4.22, 4.23, 4.24, 4.26 and 4.25 for the various treatment plans described in Table 3.1. The resulting build-up from various treatment plans are summarized in Table 4.2. The high spike in dose, in the profiles arising from the film, is due to the permanent marker which indicates the surface of the phantom.



Build-up film: 90%, 95%

Medial: 8.71mm, 11.27mm
Central: 2.82mm, 22.56mm*
Lateral: 5.61mm, 13.28mm*
(*Long distance to reach 95% of target dose)

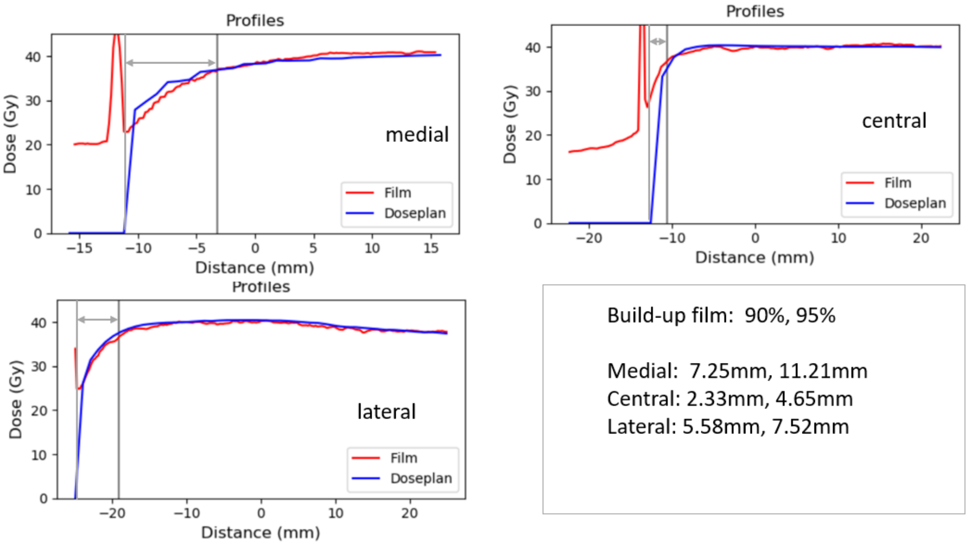
Figure 4.20: Profiles of V1 (tangential FiF) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The arrow indicates the distance in the plot from the entrance dose to where the film reaches 90% of the target dose.



Build-up film: 90%, 95%

Medial: 8.71mm, 11.27mm
Central: 2.82mm, 22.56mm*
Lateral: 5.61mm, 13.28mm*
(*Long distance to reach 95% of target dose)

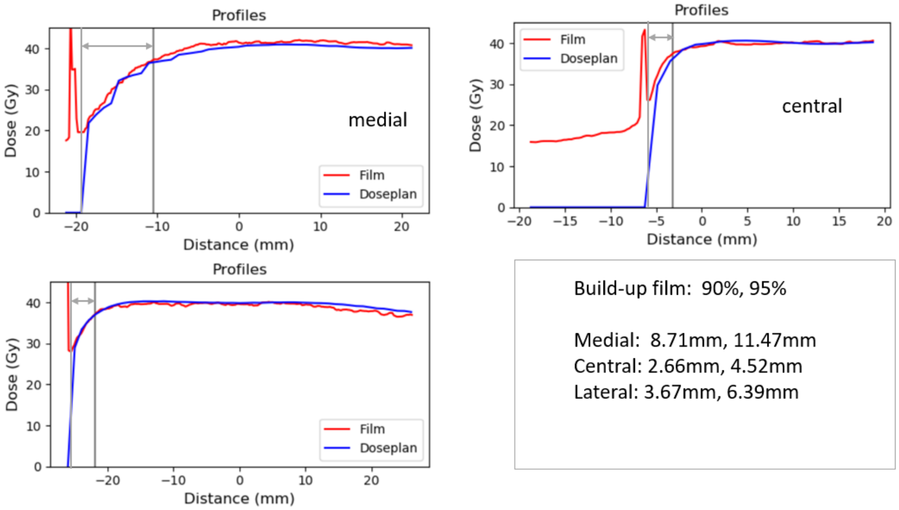
Figure 4.21: Profiles of V2 (tangential FiF 90col) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The arrow indicates the distance in the plot from the entrance dose to where the film reaches 90% of the target dose.



Build-up film: 90%, 95%

Medial: 7.25mm, 11.21mm
Central: 2.33mm, 4.65mm
Lateral: 5.58mm, 7.52mm

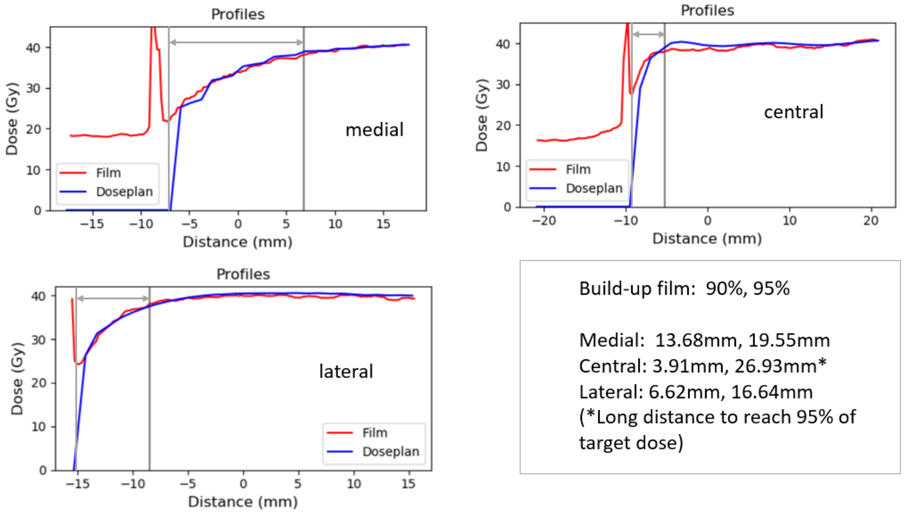
Figure 4.22: Profiles of V3 (hybrid VMAT) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The arrow indicates the distance in the plot from the entrance dose to where the film reaches 90% of the target dose.



Build-up film: 90%, 95%

Medial: 8.71mm, 11.47mm
Central: 2.66mm, 4.52mm
Lateral: 3.67mm, 6.39mm

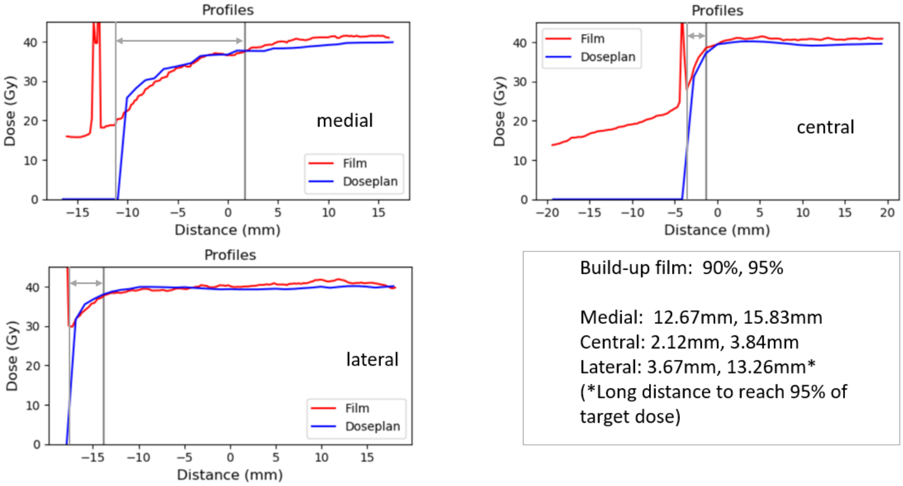
Figure 4.23: Profiles of V5 (VMAT short arcs 0col) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The arrow indicates the distance in the plot from the entrance dose to where the film reaches 90% of the target dose.



Build-up film: 90%, 95%

Medial: 13.68mm, 19.55mm
Central: 3.91mm, 26.93mm*
Lateral: 6.62mm, 16.64mm
(*Long distance to reach 95% of target dose)

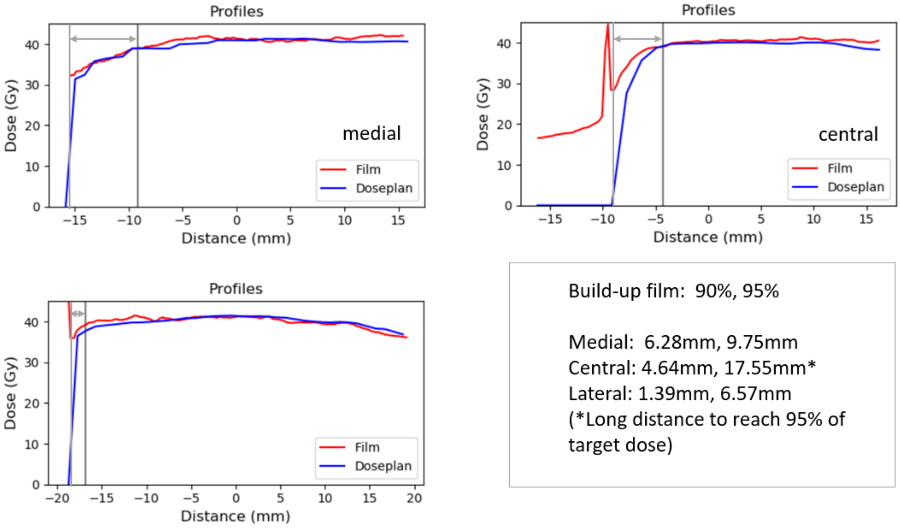
Figure 4.24: Profiles of V6 (VMAT short arcs 90col) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The arrow indicates the distance in the plot from the entrance dose to where the film reaches 90% of the target dose.



Build-up film: 90%, 95%

Medial: 12.67mm, 15.83mm
Central: 2.12mm, 3.84mm
Lateral: 3.67mm, 13.26mm*
(*Long distance to reach 95% of target dose)

Figure 4.25: Profiles of V7 (VMAT short arcs 90col) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The arrow indicates the distance in the plot from the entrance dose to where the film reaches 90% of the target dose.



Build-up film: 90%, 95%

Medial: 6.28mm, 9.75mm
Central: 4.64mm, 17.55mm*
Lateral: 1.39mm, 6.57mm
(*Long distance to reach 95% of target dose)

Figure 4.26: Profiles of V8 (medial FFF) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The arrow indicates the distance in the plot from the entrance dose to where the film reaches 90% of the target dose.

Build-up dose to target breast - measure of distance to reach 90% and 95% of target dose

	90%			95%		
	Medial	Central	Lateral	Medial	Central	Lateral
Plan name	Film, dose plan (relative diff.)	Film, dose plan (relative diff.)	Film, dose plan (relative diff.)	Film, dose plan	Film, dose plan	Film, dose plan
V1 - tangential FiF	8.71mm, 8.10mm (7%)	2.82mm, 3.07mm (9%)	5.61mm, 3.46mm (38%)	11.27mm, 13.73mm	22.56mm, 5.96mm	13.28mm, 7.39mm
V2 - tangential FiF 90col	6.76mm, 7.73mm (14.4%)	2.71mm, 3.25mm (19.9%)	5.11mm, 5.11mm (0.0%)	10.78mm, 18.51mm*	32.52mm, 6.05mm	11.17mm, 9.28mm
V3* - hybrid VMAT	7.25mm, 6.84mm (5.7%)	2.33mm, 2.56mm (9.9%)	5.58mm, 4.41mm (21.0%)	11.21mm, 12.37mm	4.65mm, 3.95mm	7.52mm, 6.49mm
V5 - VMAT short arcs 0col	8.71mm, 9.60mm (10.2%)	2.66mm, 3.10mm (16.5%)	3.67mm, 3.27mm (10.9%)	11.47mm, 14.12mm	4.52mm, 4.52mm	6.39mm, 6.39mm
V6 - VMAT short arcs 90col	13.68mm, 13.22mm (3.4%)	3.91mm, 2.83mm (27.6%)	6.62mm, 6.79mm (2.6%)	19.55mm, 20.84mm	26.93mm, 5.43mm	16.64mm, 11.95mm
V7 - VMAT FFF short arcs	12.67mm, 11.47mm (9.5%)	2.12mm, 3.03mm (42.9%)	3.67mm, 3.20mm (12.8%)	15.83mm, 23.37mm	3.84mm, 4.34mm	13.26mm, 7.24mm
V8 - Medial FFF	6.28mm, 6.28mm (0.0%)	4.64mm, 4.72mm (1.7%)	1.39mm, 3.08mm (121.6%)	9.75mm, 17.67mm	17.55mm, 13.08mm*	6.57mm, 14.45mm

Table 4.2: The table shows the build-up dose to the target breast. In each route the upper values are the film measurements, the middle values are the dose plan calculations, and the lower values indicated with the parenthesis are the relative differences between the film and dose plan.The build-up was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. To ensure that the profile measures a depth-dose distance in calculating the build-up, the profiles were drawn perpendicular to the surface of the breast, with incidence on the medial, central and lateral segment of the breast. *Indicates that 95% of the target dose was not reached within the profile, and so the distance was calculated based on the highest value obtained along the profile.

4.3.6 Dose to the contralateral breast

The dose to the contralateral breast (CLB) to some of the treatment plans seen in Table 3.1 was investigated using the Profiles tab in FIDORA. The dose was investigated by calculating the dose along a profile, for both film and dose plan. To compare profiles for different treatment plans, three different profiles were chosen for evaluation, as seen in Figure 4.27:

1. A horizontal profile, drawn to roughly coincide one cm below the sternum.

2. A vertical profile, with incidence on the central part of the CLB.
3. A diagonal profile, with incidence on the medial part of the CLB.

The resulting profiles for treatment plans V1, V2, V6 and V9 can be seen in Figure 4.28, 4.29, 4.30 and 4.31, respectively.

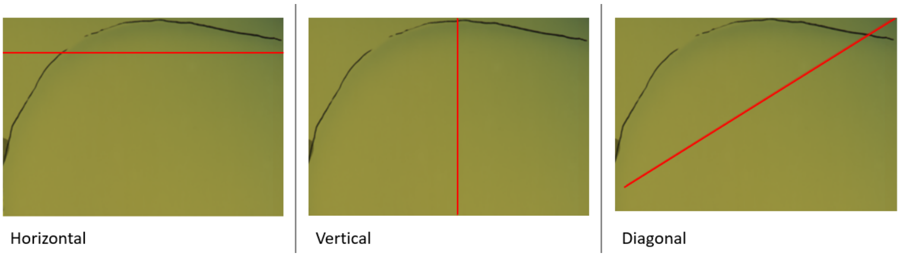
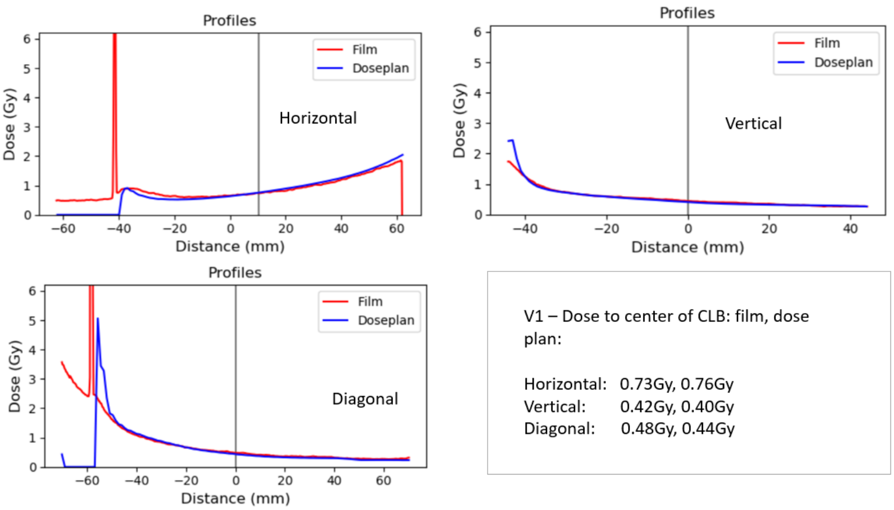


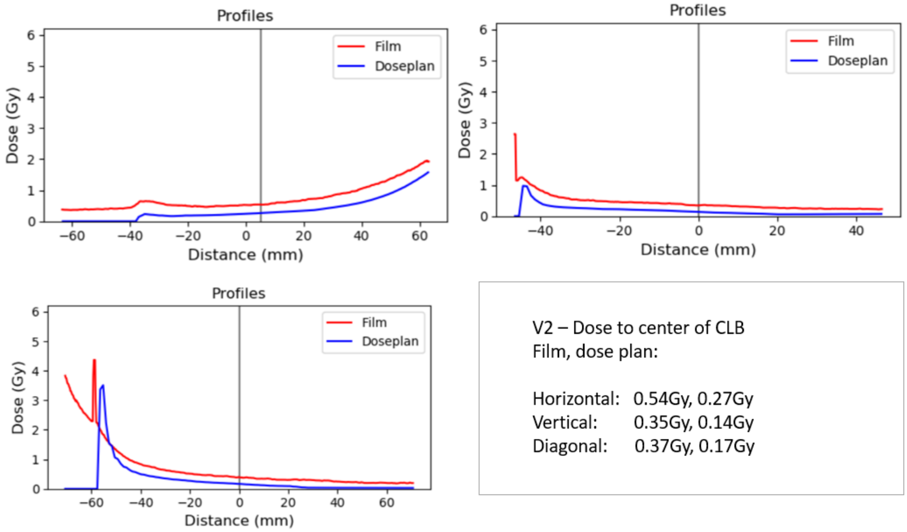
Figure 4.27: Horizontal, vertical and diagonal profiles drawn through the CLB in the isocentre plane, shown as a red line in the corresponding images.



V1 – Dose to center of CLB: film, dose plan:

Horizontal:	0.73Gy, 0.76Gy
Vertical:	0.42Gy, 0.40Gy
Diagonal:	0.48Gy, 0.44Gy

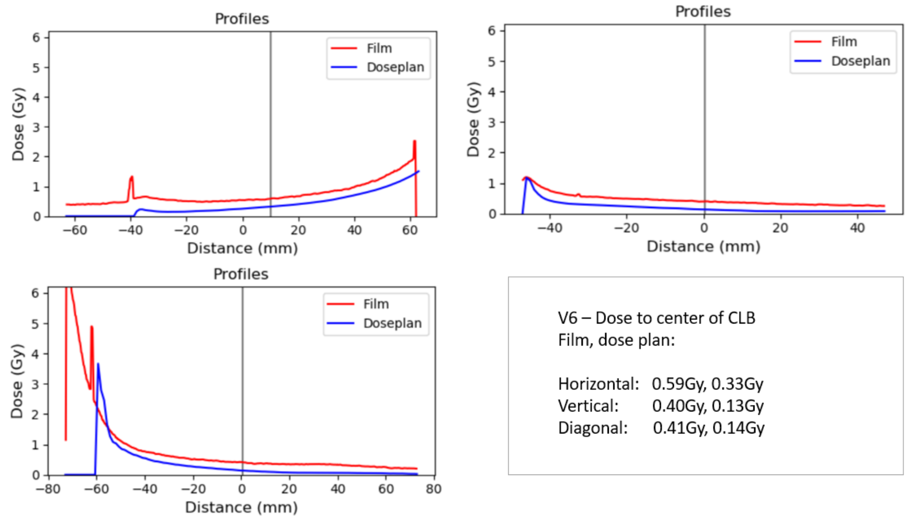
Figure 4.28: Horizontal, vertical and diagonal profiles of V1 (tangential FiF). The grey vertical line indicates the position along the profile that was evaluated.



V2 – Dose to center of CLB
Film, dose plan:

Horizontal: 0.54Gy, 0.27Gy
Vertical: 0.35Gy, 0.14Gy
Diagonal: 0.37Gy, 0.17Gy

Figure 4.29: Horizontal, vertical and diagonal profiles of V2 (tangential FiF 90col). The grey vertical line indicates the position along the profile that was evaluated.



V6 – Dose to center of CLB
Film, dose plan:

Horizontal: 0.59Gy, 0.33Gy
Vertical: 0.40Gy, 0.13Gy
Diagonal: 0.41Gy, 0.14Gy

Figure 4.30: Horizontal, vertical and diagonal profiles of V6 (VMAT short arcs 90col). The grey vertical line indicates the position along the profile that was evaluated.

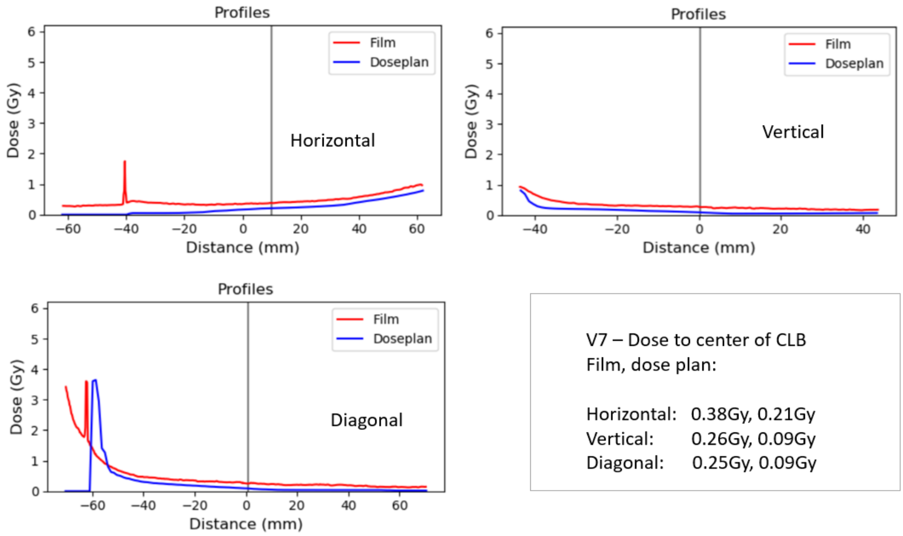


Figure 4.31: Horizontal, vertical and diagonal profiles of V9 (VMAT FFF short arcs 90col). The grey vertical line indicates the position along the profile that was evaluated.

Dose to the contralateral breast - measure of dose in the centre of profiles (Gy)						
Plan name	Horizontal film	Horizontal dose plan	Vertical film	Vertical dose plan	Diagonal film	Diagonal dose plan
V1 - tangential FiF	0.73	0.76	0.42	0.40	0.48	0.44
V2 - tangential FiF 90col	0.54	0.27	0.35	0.14	0.37	0.17
V6 - VMAT short arcs 90col	0.59	0.33	0.40	0.13	0.41	0.14
V7 - VMAT FFF short arcs	0.38	0.21	0.26	0.09	0.25	0.09

Table 4.3: The Table shows the measured and calculated doses to the contralateral breast (CLB), measured by GafChromic EBT3 film. Horizontal, vertical and diagonal profiles are drawn, and the dose is evaluated at the centre of the profiles, to measure the dose at the centre of the CLB. The doses are given in Gy.

Chapter 5

Discussion

5.1 Scanner parameters and correction matrix

Several aspects of the scanner were investigated in an earlier project [16] to see how the scanner properties contribute to the uncertainty related to using a flat-bed scanner in film dosimetry. It has been shown that uncertainties corresponding to warm-up, reproducibility and noise are all small, and by taking appropriate care when performing the measurements, these can be minimized and to some extent included into the correction matrix [16]. After a correction for non-uniform response is performed using the correction matrix the uncertainties related to using a flat-bed scanner are considered to be reduced to an acceptable level for the GafChromic EBT3 film.

The correction matrix was built with an assumption that the non-uniform scanner-readout was dependent on dose. However, the investigation of the scanner-readout for different dose levels did not show any clear systematic dependency on dose level, as seen in figure 4.11 and 4.12. There was a tendency of higher pixel value variation relative to the center for higher doses in the lateral direction, but this was not consistent for the entire scanning area that was investigated. Therefore, the correction matrix was made dose independent by averaging over the doses, and was used as one absolute correction at all dose levels. The same dose independence was found in a similar study by S. Saur and J. Frengen, [31].

5.2 FIDORA

The software FIDORA was developed in the programming language Python. Since Python is an open source language the program is easy to edit at a later time if desirable. Especially since Python is a popular dynamic object-oriented language taught in many universities and is therefore easily accessible. FIDORA is able to correct for the nonuniform read-out in the scanner, and provides the opportunity to establish calibration curves based on films irradiated at reference doses in the Dose-response tab. These calibration curves can be stored in FIDORA, and used later to map the dose in a scanned film in the Map dose tab, in addition to evaluate treatment plans in the Profiles tab. The Profiles tab is able to compare a film measurement with a dose plan matrix, from the treatment planning system. The chosen ROI in the scanned film can be adjusted to better match the dose plan matrix, which enables a better comparison between the film and the dose plan. Evaluating all profiles made with GafChromic EBT3 film in Chapter 4, the correspondence between the film measurement and the dose plan is quite good, and enables the study of complex dosimetric details occurring over a few millimeters. Some of the build-up distances observed are only a few millimeter, and can be studied using the GafChromic ET3 film along with FIDORA. Based on these observations, FIDORA poses as a good film-based dosimetry tool, and can be applied to various regions where one is interested in validating the calculated dose in the treatment planning system.

5.2.1 Calibration curve

A good calibration curve is essential in order to use the film as a reliable tool in the quality assurance (QA) of the treatment planning. To evaluate whether the calibration curve is reliable or not, the dose was mapped in a scanned reference film, with known dose, as seen in Table 4.1. The deviation from the actual reference dose and the mapped dose using the calibration curve will give an indication about the quality of the calibration curve, and this must be taken into consideration if the calibration curve is used in further investigations of treatment plans. The absolute differences between the reference doses and the doses obtained using the calibration curves (see Table 4.1) are of roughly the same magnitude at all dose levels (±1-3 cGy for doses below 1000cGy), but the relative difference is very high for the low doses, especially for 0 and 1 cGy. This means that all comparisons of dose plans with film measurements using these calibration curves will include this uncertainty, and in relative comparisons between calculated dose and film measurements at low doses this uncertainty will be more prominent. The filtered and filter-free calibration curve made the same day (second day of experiment) are almost identical for higher doses, but deviates a bit more below 10 cGy, as seen in Table 4.1. The magnitude of this deviation is related to the fact that small deviations in the linac output are accepted before a calibration of the linac is needed for the filtered or filter-free radiation beam. Thus, the daily variations in the linac output introduces another uncertainty that is carried over in all calibration curves. Comparing the two filtered calibration curves for first and second day of experiments for GafChromic EBT3 film (see Table 4.1), the relative deviation is smaller allover, and indicates that there was high stability in the linac output for the filtered beam of these two days. Another reason for why the filtered and filter-free calibration curves from the same day are slightly different might be explained from how the calibration films are placed when irradiated and later scanned and processed in FIDORA. When irradiating the reference film under reference conditions, the film should be placed in the middle of the isocenter. This is especially important when using a filter-free radiation beam, as the fluence is greater at the isocenter, and is reduced further away from the isocenter. An imprecise positioning of the calibration film might lead to a higher fluence at a point that is not at the center of the film. After irradiation the calibration film is scanned. A small area in the middle of the scanner corresponding to 25x25 pixels is read and averaged over to obtain the average pixel value.

If a calibration film is placed slightly inaccurate with respect to the center, which gives the most reliable readout, this might also affect the resulting average pixel value obtained.

The calibration curve found for the GafChromic XR-QA2 film using a filtered radiation beam was fitted to the same formula as the GafChromic EBT3 film. The XR-QA2 calibration curve, as seen in Table 4.1, yielded the lowest standard deviation from 0 to 333 cGy among all the calibration curves. However, a significant underestimation of dose at certain areas in the calibration curve was observed, yielding doses well below zero for the investigation of dose to the contralateral breast (CLB), as seen in Figure 4.18. The magnitude of this dose underestimation is significantly larger than the uncertainty due to the fitting of the calibration curve, as previously discussed. This might be explained by a higher energy dependence in the XR-QA2 film than the EBT3 film. Similarly, in "Handbook of X-ray Imaging: Physics and Technology" (2018) it is found that GafChromic XR-QA2 film is accompanied by a rather pronounced energy dependent response for beam qualities in diagnostic ranges [39]. Using a filtered radiation beam can lead to beam hardening, which removes a great deal of the low energy radiation that will be a part of the filter-free beam. As a consequence, the leakage dose to the CLB and the energy used in the filtered calibration curve might deviate significantly in energy. Therefore, this calibration curve should be used more carefully than the calibration curves obtained from EBT3, as the XR-QA2 film has not been validated as thoroughly as the EBT3 film has. In a previous project using GafChromic EBT3 film, the uncertainties of using EBT3 together with a flat-bed scanner was investigated at St. Olavs Hospital [16], and was the basis for the correction method that is integrated in FIDORA. The same uncertainty analysis has not been conducted for XR-QA2 together with the flat-bed scanner used at St. Olavs Hospital, and it therefore lacks a tailored correction method to reduce the non-uniform scanner-readout.

5.2.2 Dose to the contralateral breast

The significant underestimation of dose at certain areas in the calibration curve of the GafChromic XR-QA2 film, as seen in Figure 4.18, is the reason why the doses to the CLB were only further investigated through GafChromic EBT3 film. Using 15 fractions, the dose to the CLB was high enough to obtain reliable measurements with the EBT3 film. Interestingly, all the evaluated plans employing a 90° collimator angle (V2, V6 and V7) shows an allover higher dose measured in the film than what is calculated in the treatment planning system. The dose measured by the film is consequently higher than the dose in the dose plan, with the only exception being the high entrance dose in the dose plan at the medial side of the CLB, as seen for all measurements of the CLB in Figure 4.29, 4.30 and 4.31. This systematic deviation between measured and calculated dose is not found in the tangential FiF plan (V1) that employs a 0° collimator angle, as seen in Figure 4.28. Similar to the 90° collimator angle plans, the high entrance dose observed at medial incidence in the dose plan is not found in the V1 film measurement either. Interestingly, the opposite was found for the medial entrance dose to the CLB in a similar study by S. Saur, L. M. B. Fjellsboe, T. Lindmo and J. Frengen. Using Elekta Synergy linear accelerator, equipped with a MLCi multi leaf collimator and a 60° motorized physical wedge and measurements performed with GafChromic EBT film, the medial entrance dose to the CLB was measured to be higher than what the treatment planning system modelled [32]. These film measurements might indicate that the treatment planning system underestimates the dose to the CLB. This might imply that the linac model in RayStation is not as reliable outside the field limited by the (lower) jaws. Many of the treatment plans (see Table 4.2) employed a collimator angle of 90°, as this collimator choice offers a potential reduction in dose to the CLB due to less leakage dose through the (lower) jaws than the MLCs. Yet, evaluating the doses to the CLB in Table 4.3 a collimator angle of 90° demonstrated little sparing effect. Instead, a possible sparing effect is observed in V7, the VMAT filter-free plan, which gives the over all lowest dose to the CLB, measured in the center of all profiles investigated. For the different types of profiles investigated (horizontal, vertical and diagonal) V7 offers at worst a 38% reduction in dose to the centre of the CLB compared to V1, the tangential FiF plan. This is probably a result of less head scattering due to the removal of the flattening filter.

5.2.3 Build-up dose to target breast

From Table 4.2 the build-up dose can be evaluated for the treatment plans studied in this project. The relative difference between the measured build-up in the film and in the dose plan varied considerably for some of the treatment plans. However, the positioning errors arising from irradiation of the film in the phantom as well as scanning the film, are of the same scale of magnitude as some of the smallest build-up distances observed. Also, the resolution in the dose plan matrix is of 1x1x1mm compared to 0.2mm/pixel for the film, adding an intrinsic uncertainty in all comparisons between film and dose plan. Therefore, one can argue that a large relative difference, especially at areas with short build-up distance is not necessarily a result of a poor measurement. In the V7 plan at central incidence for instance, the relative difference in build-up distance to 90% of the target dose is 42.9%. But since the build-up measured by the film here is 2.12mm and 3.03mm in the dose plan, the difference is more likely to be a positioning error than an actual deviation between the modelled and calculated dose.

The build-up distance to 95% of the target dose proved difficult to use as parameter to compare between plans. Along many profiles, the measured dose reached 93% or 94% of the target dose at a reasonable distance from the surface, but never succeeded in fulfilling the prescribed dose of 95% of the target dose. In a standardized setup with a water phantom and normal incidence, 95% of the target dose is a useful parameter. However, in this experiment the maximum doses at the three areas (medial, central and lateral) will vary considerably. The treatment plan accepts doses varying from 95-105% in the target volume. Therefore, the build-up distance to 95% of the target dose may vary from 90-100% of the target dose. In the rest of this section, the build-up distance refers to the distance from the entrance dose to where the dose reaches 90% of the target dose.

For plans V1 and V2, only different in choice of collimator angle, the medial build-up distance to 90% of the target dose is slightly larger than the corresponding lateral build-up distance, as seen in Table 4.2. Given the symmetrical design of the treatment plans (with respect to the central part of the breast), one would expect the build-up distance of the medial and lateral segments to be similar. Evaluating the other treatment plans seen in Table

4.2, the same asymmetric build-up distance is in fact observed for all plans. This might be explained by the angles of the incoming beam in the medial and lateral segment of the breast. If the incidence at the lateral segment is less normal to the surface compared to the medial segment, this can result in a systematic reduced measure of the lateral build-up distance. The central build-up distance observed in various treatment plans, is in general the shortest for all plans except V8, since there are no central segments or arcs with incidence at the central part of the breast in these plans. Plan V8, the medial FFF (see Table 4.2), offers a potential reduction in the dose to the CLB due to less scattered radiation in the direction of the CLB. Evaluating the film measurements from V8, this treatment plan is likely to provide the best sparing of skin at the central part of the target breast, but in return give the most damage to the lateral part of the breast. This was also observed by S. Almberg, T. Lindmo and J. Frengen in a similar study [1] when evaluating a hybrid IMRT plan, consisting of medial segments and IMRT-fields.

5.3 Future work

To better answer the question about which treatment technique is most beneficial with regards to build-up dose in the target breast and dose to the contralateral breast, more film measurements are needed to get better statistics and measures of uncertainty.

5.3.1 GafChromic XR-QA2 film

In order to use GafChromic XR-QA2 film in the clinic, it should be investigated more. That is, the uncertainty contributions from film-film variations, intra-film noise, intra-film uniformity and uncertainty in the fitted curve for film response should be calculated, as has been done for the GafChromic EBT3 film [16]. Then, perhaps one is able to make a different correction matrix that can be applied to scanned GafChromic XR-QA2 films. This would give more reliable results for experiments analysed with the GafChromic XR-QA2 film in FIDORA. Also, the energy dependency should be studied in more detail, and a filter-free calibration curve should be established to see if the removal of beam hardening through the use of a filtered beam can demonstrate a more reliable calibration curve.

5.3.2 FIDORA

FIDORA is at this point, a dosimetry tool with specific applications. Future work, perhaps done by a future student, should include generalization of the program. A great advantage would be if FIDORA was able to analyse different GafChromic films with good reliability, and to accept scanned films from different flat-bed scanners. This would require investigation of several scanners, together with different GafChromic film types. This could result in several available correction matrices, or perhaps a more sophisticated correction method.

Generalization is a keyword when describing the future work with FIDORA. As of today, the program only accepts images in (*tif) format, with 127 dpi. This should preferably be made more general, so that FIDORA can accept images of different formats and dpi (dots per inch, yielding spatial resolution).

Matching the dose plan matrix with the scanned film was a great challenge throughout this project. However, it was of great importance, as the Profiles tab depended on comparing the scanned film with the corresponding area in the dose plan matrix. Today, there exists two different options for positioning in FIDORA:

1. If the isocenter is on the irradiated film, it can be recognized, and the distance from the isocenter to the reference point on the phantom can be mapped to the corresponding distance in the dose plan matrix.
2. If the isocenter is not on the irradiated film, as is the case when measuring the dose to the contralateral breast, one must use a different approach. When irradiating the GafChromic film one must note the relative displacement distance from a given reference point in the film to the reference point in the phantom, for all spatial directions (x,y,z).

The process of matching the dose plan matrix with the scanned film should also be made more general. Preferably, one should be able to upload a scanned film, together with a DICOM-file, and FIDORA should be able to match these. This approach is moving towards the field of machine learning, and is not trivial. One would also require a great set of test data to train the model, so that it could be trusted to match the scanned film and the dose plan matrix correctly.

Chapter 6

Conclusion

This study has shown that the nonuniformity effect along the detector array in radiochromic film dosimetry using a CCD-based flat-bed scanner, can be properly corrected for using one absolute correction matrix independent of dose level, as shown for GafChromic EBT3 film. However, more investigations towards characterising the GafChromic XR-QA2 must be done before it can be used in the clinic. Especially the energy dependence of the XR-QA2 film should be studied in more detail.

A Python program named FIDORA was developed to perform various analysis associated with film dosimetry, using GafChromic film and an Epson v750 Pro flat-bed scanner. FIDORA performs a correction of the nonuniform read-out of the scanner and corrects for all three color channels in landscape mode. FIDORA provides the opportunity to establish calibration curves based on films irradiated to reference doses, and can accept multiple images irradiated at the same reference dose, and use the average in order to reduce the influence of the scan-to-scan variation. Other functionalities offered by FIDORA is to map the dose in a scanned image, as well as evaluation of profiles for a given region of interest, using a calibration of choice. Based on the investigation of several treatment plans, FIDORA poses as a good film-based dosimetry tool and can be applied to various regions where one is interested in validating the calculated dose in the treatment planning system.

FIDORA was applied to investigate the build-up dose to the target breast, as well as the dose to the contralateral breast (CLB). The build-up distance in the target breast, measured from the entrance dose to 90% of the target dose, resulted in a slightly asymmetrical film measure of the medial and lateral segment of the breast for all treatment plans. This might be explained by the angles of the incoming beams. If the incidence at the lateral segment is less normal to the surface compared to the medial segment, this can result in a systematic reduced measure of the lateral build-up distance, and thus less lateral skin sparing. The dose from 15 fractions measured in the CLB with GafChromic EBT3 film yielded an allover higher dose than what was calculated in the dose plan for the treatment plans employing a 90° collimator angle (V2, V6 and V7), with the only exception being a very high entrance dose observed in the dose plan at medial incidence. The treatment plan employing a 0° collimator angle (V1) demonstrated an over all better correspondence between the calculated dose in the dose plan and the measured dose in the film, but also showed a very high entrance dose in the dose plan at medial incidence that was not found in the film measurement. These findings might indicate that the linac model in RayStation is not as reliable outside the fields limited by the (lower) jaws. Evaluating the various treatment plans investigated in this project, the potential reduction in dose to the CLB through the use of a collimator angle of 90° demonstrated little sparing effect to the CLB. Instead, a sparing effect to the CLB was found through the use of a filter-free VMAT treatment plan. This plan offered at worst a 38% reduction in dose to the center of the CLB compared to a tangential field-in-field plan.

Bibliography

[1] Almberg, S. S., Lindmo, T., & Frengen, J. (2011). Superficial doses in breast cancer radiotherapy using conventional and IMRT techniques: a film-based phantom study. *Radiotherapy and Oncology*, 100(2), 259-264.

[2] Alsadig, A. A., Abbas, S., Kandaiya, S., Ashikin, N. N. N., & Qaeed, M. A. (2017). Differential dose absorptions for various biological tissue equivalent materials using Gafchromic XR-QA2 film in diagnostic radiology. *Applied Radiation and Isotopes*, 129, 130-134.

[3] Apipunyasopon, L., Srisatit, S., & Phaisangittisakul, N. (2013). An investigation of the depth dose in the build-up region, and surface dose for a 6-MV therapeutic photon beam: Monte Carlo simulation and measurements. *Journal of radiation research*, 54(2), 374–382. https://doi.org/10.1093/jrr/rrs097

[4] Ashland. Efficient Protocols for Accurate Radiochromic Film Calibration and Dosimetry. GAFCHROMIC^{TM} DOSIMETRY MEDIA, TYPE EBT-3. Retrieved from: `http://www.gafchromic.com/documents/Efficient%20Protocols%20for%20Calibration%20and%20Dosimetry.pdf`

[5] Ashland. EBT3 specification sheet. GAFCHROMIC^{TM} DOSIMETRY MEDIA, TYPE EBT-3. Retrieved from: `http://www.gafchromic.com/documents/EBT3_Specifications.pdf`

[6] Ashland. XR-QA2 specification sheet. GAFCHROMIC^{TM} DOSIMETRY MEDIA, TYPE XR-QA2. Retrieved from: `http://www.gafchromic.com/documents/PC-11805_Gafchromic_XR.pdf`

[7] Attix FH. (1986). *Introduction to radiological physics and radiation dosimetry.* John Wiley & Sons Inc.

[8] Battista, J. J., Rider, W. D., & Van Dyk, J. (1980). Computed tomography for radiotherapy planning. *International Journal of Radiation Oncology Biology Physics*, 6(1), 99-107.

[9] Butson MJ, Yu PKN, Cheung T, Metcalfe P. (2003). Radiochromic film for medical radiation dosimetry. Mater. Sci. Eng. R 41, 61-120.

[10] Danielsen, S. & A. B. L. Marthinsen, FY8409 Radiaiton therapy physics. Course, NTNU, fall semester 2019.

[11] Danielsen, S. & Marthinsen, A.B.L (2019). FY8409,NTNU: Clinical Dosimetry [PowerPoint slides]. Retrieved from: `https://ntnu.blackboard.com/bbcswebdav/pid-815716-dt-content-rid-23620786_1/courses/194_FY8409_1_2019_H_1/Dosimetri_praktisk%20demo%202019.pdf`

[12] Das, I. J. (Ed.). (2017). *Radiochromic film: role and applications in radiation dosimetry.* CRC Press.

[13] Devic, S., Wang, Y. Z., Tomic, N., & Podgorsak, E. B. (2006). Sensitivity of linear CCD array based film scanners used for film dosimetry. *Medical physics*, 33(11), 3993-3996.

[14] Fuss, M., Sturtewagen, E., De Wagter, C., & Georg, D. (2007). Dosimetric characterization of GafChromic EBT film and its implication on film dosimetry quality assurance. *Physics in Medicine* & *Biology*, 52(14), 4211.

[15] Grégoire, V., & Mackie, T. R. (2011). State of the art on dose prescription, reporting and recording in intensity-modulated radiation therapy (ICRU report No. 83). Cancer/Radiothérapie, 15(6-7), 555-559.

[16] Gustavsen, S. & Håland, A. V. (2019). Film based dosimetry: Film based dosimetry using EBT3 and a flat-bed scanner.

[17] Hall, E. J. et al, Radiobiology for the Radiologist. Wolters Kluwer Health, 2011.

[18] Haryanto, F., Fippel, M., Bakai, A., & Nüsslin, F. (2004). Study on the tongue and groove effect of the Elekta multileaf collimator using Monte Carlo simulation and film dosimetry. *Strahlentherapie und Onkologie*, 180(1), 57-61.

[19] Herrmann, J. (2016). Clinical Cardio-oncology E-Book. Elsevier Health Sciences.

[20] IAEA International Atomic Energy Agency. Absorbed dose determination in external beam radiotherapy: An international Code of Practice for Dosimetry Based Standards of Absorbed Dose to Water. IAEA Technical Reports Series No. 398. Vienna: IAEA; 2000.

[21] IMRT phantom specification sheet. I'mRT phantom: a smart modular phantom for patient QA. IBA dosimetry America. Retrieved from: `https://www.iba-dosimetry.com/fileadmin/user_upload/products/02_radiation_therapy/imRT_phantom/I_mRT_Phantom_Rev.1b_0819.pdf`

[22] Kim, Y. H., Park, H. R., Kim, W. T., Kim, D. W., Ki, Y., Lee, J., ... & Nam, J. H. (2015). Effect of the collimator angle on dosimetric verification of volumetric modulated arc therapy. *Journal of the Korean Physical Society*, 67(1), 243-247.

[23] León-Marroquín, E. Y., Camacho-López, M. A., García-Garduño, O. A., Herrera-González, J. A., Villarreal-Barajas, J. E., Gutiérrez-Fuentes, R., & Contreras-Bulnes, R. (2016). Spectral analysis of the EBT3 radiochromic film irradiated with 6 MV x-ray radiation. *Radiation Measurements*, 89, 82-88.

[24] Levernes, S. Volumes and doses for external radiotherapy - Definitions and recommendations. Østerås: Norwegian Radiation Protection Authority, Language: Norwegian. Strålevern Rapport, 2012:9.

[25] Li, D., Weng, S., Zhong, C., Xu, D., & Yuan, Y. (2019). Risk of second primary cancers among long-term survivors of breast cancer. *Frontiers in Oncology*, 9, 1426.

[26] Mayles, P., Nahum, A., & Rosenwald, J. C. (2007). Handbook of radiotherapy physics: theory and practice. CRC Press.

[27] Mutic, S., Esthappan, J., & Klein, E. E. (2002). Peripheral dose distributions for a linear accelerator equipped with a secondary multileaf collimator and universal wedge. *Journal of applied clinical medical physics*, 3(4), 302-309.

[28] Naume, B. et al, Nasjonalt handlingsprogram med retningslinjer for diagnostikk, behandling og oppfølging av pasienter med brystkreft. Nasjonale faglige retningslinjer, IS-2736. Helsedirektoratet, vol. 11, pp. 79-100, 2018.

[29] Niroomand-Rad, A., Blackwell, C.R., Coursey, B.M., Gall, K.P., Galvin, J.M., McLaughlin, W.L., Meigooni, A.S., Nath, R., Rodgers, J.E. & Soares, C.G. (1998). Radiochromic film dosimetry: Recommendations of AAPM Radiation Therapy Committee Task Group 55, *Med. Phys*. 25 (11), 2093-2115.

[30] Oldham, M. (2001). Radiation physics and applications in therapeutic medicine. *Physics Education*, 36(6), 460.

[31] 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(7Part1), 3094-3101.

[32] Saur, S., Fjellsboe, L. M. B., Lindmo, T. & Frengen, J. (2009). Contralateral breast doses measured by film dosimetry: tangential techniques and an optimized IMRT technique. *Physics in Medicine & Biology, 54(15), 4743.*'

[33] Saur, S. (2019). FY8409,NTNU: Megavoltage electron accelerators for cancer treatment with photons and electrons [PowerPoint slides]. Retrieved from: `https://ntnu.blackboard.com/bbcswebdav/pid-748188-dt-content-rid-21867382_1/courses/194_FY8409_1_2019_H_1/Linac_NTNU2019.pdf`

[34] Shiau, A. C., Hsieh, C. H., Tien, H. J., Yeh, H. P., Lin, C. T., Shueng, P. W., & Wu, L. J. (2014). Left-sided whole breast irradiation with hybrid-IMRT and helical tomotherapy dosimetric comparison. *BioMed research international*, 2014.

[35] Singh, V. P., & Badiger, N. M. (2014). Effective atomic numbers of some tissue substitutes by different methods: a comparative study. *Journal of Medical Physics/Association of Medical Physicists of India*, 39(1), 24.

[36] Song, Y., Mueller, B. A., Laguna, J., Obcemea, C. H., Saleh, Z., Tang, X., ... & Mychalczak, B. R. (2016). Investigation of MLC Dose Leakage in Volumetric Modulated Arc Therapy Head and Neck Treatment. *International Journal of Radiation Oncology Biology Physics*, 96(2), E595-E596.

[37] van Battum, L. (2018). Filmdosimetry: past and future.

[38] Yeo, I.J. & Kim, J.O. (2004) *A Procedural Guide to Film Dosimetry: With Emphasis on IMRT.* Medical Physics Publishing.

[39] Zscherpel, U., & Ewert, U. (2018). Handbook of X-ray imaging: Physics and technology.

Appendix

Appendix A

FIDORA

The code that builds the python-based program FIDORA, short for Film Dosimetry in Radiotherapy, is included here. The current version of the program can be viewed at https://github.com/anevh/FIDORA. Each section in the appendix is one python script. The following scripts will be included:

- `notebook.py`
- `Globals.py`
- `gloVar.py`
- `Correction_functions.py`
- `CoMet_functions.py`
- `Dose_response_functions.py`
- `Map_dose.py`
- `Profile_functions.py`

notebook.py is responsible for making the graphical user interface (GUI), and calls for relevant functions and variables in functions.py scripts and in Globals.py and gloVar.py, respectively.

A.1 notebook.py

```
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
import Dose_response_functions, Profile_functions, DVH_functions
from PIL import Image, ImageTk
import os
import sys


Globals.form.title(”FIDORA”)
#lobals.form.geometry(”1250x600”)
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)# place(relwidt=0.61,relheight
    =0.15,
#relx=0.02, rely=0.0)
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’)
Globals.tab_parent.add(Globals.tab4, text=’Profiles’)
#Globals.tab_parent.add(Globals.tab5,text=’DVH’)

style = ttk.Style()
style.theme_create(’MyStyle’, parent= ’classic’, settings={
    ”.”: {
        ”configure”: {
            ”background”: ’#FFFFFF’, # All colors except for active tab−
    button
            ”font”: ’red’
        }
    },
    ”Horizontal.TProgressbar”:{
        ”configure”: {
            ”background”: ’#2C8EAD’,
            ”bordercolor”: ’#32A9CE’,
            ”troughcolor”: ”#ffffff”,
        }
    },
    "TNotebook": {
        "configure": {
            "background":'#ffffff', # color behind the notebook
            "tabmargins": [5, 5, 10, 10], # [left margin, upper margin,
            #right margin,margin beetwen tab and frames]
            "tabposition": 'wn',
            "borderwidth": 0,

        }
    },
    "TNotebook.Tab": {
        "configure": {
            "background": '#0A7D76', # Color of non selected tab-button
            "foreground": '#ffffff',
            "padding": [30,35, 20,35], # [space beetwen text and
    horizontal tab-button
            #border, space between text and vertical tab_button border]
            "font": ('#FFFFFF', '15'),
            "borderwidth": 1,
            "equalTabs": True,
            "width": 13

        },
        "map": {
            "background": [("selected", '#02B9A5')], # Color of active tab
            "expand": [("selected", [1, 1, 1, 0])] # [expanse of text]
        }
    },
    "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')


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_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 = "icon_correct.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)
######################################## INTRO TAB ###################################################


#scrollbar = Scrollbar(Globals.intro_tab)
#scrollbar.pack(side=RIGHT, fill=Y)#grid(row=0, column=1, sticky=N+S+E)#pack(side=RIGHT, fill=Y)
#Globals.intro_tab.grid_columnconfigure(0, weight=0)
#Globals.intro_tab.grid_rowconfigure(0, weight=0)
intro_tab_canvas = tk.Canvas(Globals.intro_tab)#, yscrollcommand=scrollbar.set)
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 \ndeveloped 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_readmore_text = tk.Text(tab1_text_box, height=1, width=1)
tab1_readmore_text.grid(row=1, column=0, sticky = N+S+W+E, pady=(65,0), padx = (110,0))
tab1_readmore_text.insert(INSERT,"Read more...")
tab1_readmore_text.config(state=DISABLED, bd=0, bg='#E5f9ff', fg='#130E07', font=('calibri', '12', 'bold'))
tab1_text_box.grid_columnconfigure(2,weight=1)
tab1_text_box.grid_rowconfigure(2,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)

"""
tab1_readmore = tk.Button(tab1_text_box, text='Read more',cursor='hand2', font=('calibri', '12', 'bold'),\
    relief=FLAT, state=tk.ACTIVE, width = 15, command=intro_tab_functions.readMore)
tab1_readmore.place(relwidth=0.25, relheight=0.13, relx=0.27, rely=0.054)
"""
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 \nfunction. 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_readmore_text = tk.Text(tab2_text_box, height=1, width=1)
tab2_readmore_text.grid(row=1, column=0, sticky = N+S+W+E, pady=(65,0), padx = (300,0))
tab2_readmore_text.insert(INSERT,"Read more...")
tab2_readmore_text.config(state=DISABLED, bd=0, bg='#E5f9ff', fg='#130E07', font=('calibri', '12', 'bold'))
tab2_text_box.grid_columnconfigure(2, weight=1)
```

```
tab2_text_box.grid_rowconfigure(2, weight=1)

tab2_box_figure = Image.open("icon_dose_response.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, "Map dose")
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,"Compare dose distribution in your treatment plan \nwith the measures distribution by the Gafchromic \nfilm.\
 Using the gamma evaluation index a map of \npass/fail and variations is visualised.")
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_readmore_text = tk.Text(tab3_text_box, height=1, width=1)
tab3_readmore_text.grid(row=1, column=0, sticky = N+S+W+E, pady=(65,0), padx = (285,0))
tab3_readmore_text.insert(INSERT,"Read more...")
tab3_readmore_text.config(state=DISABLED, bd=0, bg='#E5f9ff', fg='#130E07', font=('calibri', '12', 'bold'))
tab3_text_box.grid_columnconfigure(2, weight=1)
tab3_text_box.grid_rowconfigure(2, weight=1)

tab3_box_figure = Image.open("icon_map_dose.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, "Profiles")
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,"Investigate the profiles measured using GafChromic \nfilm and compare with the profiles in your treatment \nplan.\
 Using gamma evaluation an acceptance tube \ncan be places over the profile.")
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_readmore_text = tk.Text(tab4_text_box, height=1, width=1)
tab4_readmore_text.grid(row=1, column=0, sticky = N+S+W+E, pady=(65,0), padx = (235,0))
tab4_readmore_text.insert(INSERT,"Read more...")
tab4_readmore_text.config(state=DISABLED, bd=0, bg='#E5f9ff', fg='#130E07', font=('calibri', '12', 'bold'))
tab4_text_box.grid_columnconfigure(2, weight=1)
tab4_text_box.grid_rowconfigure(2, 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.configure(scrollregion = intro_tab_canvas.bbox("all"))
intro_tab_canvas.grid(row=0, column=0, sticky=N+S+W)#pack(side=LEFT, fill=BOTH)
#Globals.intro_tab.grid_columnconfigure(1, weight=2)
#Globals.intro_tab.grid_rowconfigure(1, weight=2)
#scrollbar.config(command=intro_tab_canvas.yview)

##################################### TAB 1 - CoMet ############################################


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

CoMet_explained = tk.Text(Globals.tab1_canvas, height=4, width=105)
CoMet_explained.insert(INSERT, \
"Start the correction by choosing the correct *.tif file containing the scanned image of \n\
GafChromic film. The film should be scanned using Epson Perfection v750 Pro with dpi setting \n\
72 or 127. Then pick which folder the corrected file should be uploaded to. The corrected file\n\
will be saved as a DICOM. Write filename and patient name (optional) before doing the correction.\n\
An illustration of the corrected 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=1, column=0, columnspan=2, sticky = W+E, padx = (0, 190), pady=(10,5))
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=1, column = 0, padx = (200, 0), pady=(10,5))
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', '14'), 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+W, pady=(20,20), padx=(80,0))
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', '12'), 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=2, column=0, columnspan=2, sticky = N+S+W+E, padx = (0, 190), pady=(0,15))
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=2, column = 0, padx = (200, 0), pady=(0,15))
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+W, pady=(25,0), padx=(80,0))
CoMet_save_to_folder.insert(INSERT,"Folder to save the corrected image")
CoMet_save_to_folder.config(state=DISABLED, bd=0, font=('calibri', '12'), fg='gray', bg='#ffffff')

## Function to test the filename the user chooses for the corrected image
def testFilename():
    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=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
        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=3, column=0, columnspan=2, sticky = W+E, padx = (0,190), pady=(0,15))
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=3, column = 0, padx = (200, 0), pady=(0,15))
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=(80,0))
Globals.CoMet_save_filename.insert(END,"Filename (will be saved as *.dcm)")
Globals.CoMet_save_filename.config(state=NORMAL, bd=0, font=('calibri', '12'), fg='gray', bg='#ffffff')


def writeFilename(event):
    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)


#Functioin to validate the patient name written in by the user
def testName():
    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 letters (not
      ,  ,  ) 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=4, column=0, columnspan=2, sticky =
    W+E, padx = (0, 190), pady=(5,0))
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=4, column = 0, padx = (200, 0), pady
    =(5,0))
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=(80,0))
CoMet_save_patientName.insert(END,"Patient name (Optional)")
CoMet_save_patientName.config(state=NORMAL, bd=0, font=('calibri', '12'),
    fg='gray', bg='#ffffff')

def writePname(event):
    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=4, column=2,rowspan=2, padx = (0, 0),
    pady=(0,0), 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 ,
    width=240, height=290)
Globals.CoMet_print_corrected_image.grid(row=0, column=2, rowspan=3,
    sticky=N+W+S+E, pady=(20,0), padx=(0,0))
Globals.CoMet_print_corrected_image.config(bg='#ffffff', bd = 0, relief=
    FLAT)
Globals.tab1_canvas.grid_columnconfigure(11,weight=0)
Globals.tab1_canvas.grid_rowconfigure(11, weight=0)
Globals.CoMet_print_corrected_image.create_image(123,148,image=
    CoMet_empty_image_image)
Globals.CoMet_print_corrected_image.image = CoMet_empty_image_image


def clearAll():
    #Clear out the 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=(80,0))
    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', '12'), 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=32)
    CoMet_save_to_folder.grid(row=0, column=0, columnspan=2, sticky=E+W,
    pady=(25,0), padx=(80,0))
    CoMet_save_to_folder.insert(INSERT,”Folder to save the corrected image
    ”)
    CoMet_save_to_folder.config(state=DISABLED, bd=0, font=('calibri', '12
    '), 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=(80,0))
    Globals.CoMet_save_filename.insert(END,"Filename (will be saved as *.dcm)")
    Globals.CoMet_save_filename.config(state=NORMAL, bd=0, font=('calibri', '12'), fg='gray', bg='#ffffff')
    Globals.CoMet_corrected_image_filename.set("Error!")
    Globals.CoMet_save_button_1.config(state=ACTIVE)

    def writeFilename(event):
        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=(80,0))
    CoMet_save_patientName.insert(END,"Patient name (Optional)")
    CoMet_save_patientName.config(state=NORMAL, bd=0, font=('calibri', '12'), fg='gray', bg='#ffffff')
    Globals.CoMet_patientName.set("Error!")
    CoMet_save_button_2.config(state=ACTIVE)

    def writePname(event):
        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=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
    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=4, column=2, rowspan=2, padx=(100,0), pady=(0,0), sticky=E)
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=CoMet_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=CoMet_clear_all_button_image

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

##################################### TAB 2 − Dose response ############################################

#”To be able to perform an accurate dose caluclations using GafChromic film EBT3 \n\
#it is necessary to create a dose−respons curve for each batch of film, in addition\n\
#to a calibration scan before/along every use. The respons of GafChromic film \n\
#EBT3 is modelled using a rational function, X(D,n) = a + b/(D−c), as this has \n\
#proven to fit well with the film behavior. In the model X(D,n) is the scanner \n\
#respons in color channel n and a, b and c are constants. Because of the nature \n\
#of asymptotic fitting functions a good fit will be achieved by using doses in \n\
#geomteric progression, D, nD, nnD, etc.. Also, to avoid scanner uncertainties\n\
#each dose should be scannet three times and uploaded here where an average will be used.”

#Irradiate film piece of size (Bestemt med maske?) with known doses. Place one and one\n\
#film piece in the center of the scanner and perfom three scans per dose. ”

```

```python
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, "\
Follow the calibration specifications given under 'Help' or 'Read more' at the first window. Upload the scanned *.tif files (there should be at least 3 of each\n\
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 \n\
setting 72 or 127. When saving the calibration the dose response data will be saved and can be used chosen for later use of this software. The dose response \n\
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_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'))

dose_response_save_calibration_button_frame = tk.Frame(Globals.tab2_canvas)
dose_response_save_calibration_button_frame.grid(row=2, column = 2, sticky=N+S+E+W, padx=(0,0), pady=(120,0))
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=2, column=1, sticky=N+S+E+W,
     padx=(0,0), pady=(120,0))
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 3 - Map dose
    ###########################################

#path = os.path.dirname(sys.argv[0])
#path= "upload.png"
#upload_button_image = ImageTk.PhotoImage(file=path)

Globals.tab3_canvas.config(bg='#ffffff', bd = 0, relief=FLAT,
    highlightthickness=0)


upload_film_data = tk.Button(Globals.tab3_canvas, text='Upload',image=
    Globals.upload_button_image, cursor='hand2', font=('calibri', '12'), \
    relief=FLAT, state=ACTIVE, width=12, command=lambda: Map_Dose.
    UploadAction("FILM"))
upload_film_data.place(relwidth=0.17, relheight=0.11, relx=0.3, rely=0.03)
upload_film_data.image = Globals.upload_button_image


Globals.tab3_canvas.pack(expand=True, fill=BOTH)
##################################### TAB 4 - Profiles
    ##########################################

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, "\
SliceThickness i plan m  v re ['1','1'], ['2','2'] eller ['3','3'], Filmen m  legges i xy, xz eller yz planet (lage figur?), Filmen m  scannes  \n\
parallelt med retningene i skanneren (programmet vil anta dette), man m  markere \"opp\" og \"bort\" p  filmen. N r man skanner m  man legge \n\
oppmerket oppover og bort merket mot h yre (kan man kreve dette i alle plan?). \
Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det, \n\
Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det, " )
profiles_explain_text.grid(row=0, column=0, columnspan=5, 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 ='#E5f9ff', relief=FLAT)

profiles_upload_film_frame = tk.Frame(Globals.tab4_canvas)
profiles_upload_film_frame.grid(row=3, column = 0, padx = (0, 240), pady=(10,0), sticky=N)
Globals.tab4_canvas.grid_columnconfigure(1, weight=0)
Globals.tab4_canvas.grid_rowconfigure(1, weight=0)
profiles_upload_film_frame.config(bg = '#ffffff')

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=3, column = 0, padx = (0,40), pady=(10,0), sticky=N)
Globals.tab4_canvas.grid_columnconfigure(3, weight=0)
Globals.tab4_canvas.grid_rowconfigure(3, weight=0)
profiles_upload_film_frame.config(bg = '#ffffff')

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=3, column=0, padx=(160,0), pady=(10,0), sticky=N)
Globals.tab4_canvas.grid_columnconfigure(10, weight=0)
Globals.tab4_canvas.grid_rowconfigure(10, weight=0)
profiles_upload_rtplan_frame.config(bg='#ffffff')

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

Globals.profiles_film_orientation_menu = OptionMenu(Globals.tab4_canvas, Globals.profiles_film_orientation, 'Axial', 'Coronal', 'Sagittal')
Globals.profiles_film_orientation_menu.grid(row=1, column=0, sticky=N+S, padx=(60,0))
Globals.tab4_canvas.grid_columnconfigure(2, weight=0)
Globals.tab4_canvas.grid_rowconfigure(2, weight=0)
Globals.profiles_film_orientation_menu.config(bg = '#ffffff', width=15, relief=FLAT)

film_orientation_menu_text = tk.Text(Globals.tab4_canvas, 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.grid(row=1, column=0, sticky=N+S+W, padx=(30,0), pady=(5,0))
Globals.tab4_canvas.grid_columnconfigure(3, weight=0)
Globals.tab4_canvas.grid_rowconfigure(3, weight=0)

profiles_film_orientation_help_frame = tk.Frame(Globals.tab4_canvas)
profiles_film_orientation_help_frame.grid(row=1, column=0, sticky=N+S+E, padx=(0,40))
Globals.tab4_canvas.grid_columnconfigure(6, weight=0)
Globals.tab4_canvas.grid_rowconfigure(6, weight=0)
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 = tk.Text(Globals.tab4_canvas, 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.grid(row=2, column=0, sticky=N+S+W, padx=(30,0), pady=(5,0))
Globals.tab4_canvas.grid_columnconfigure(30, weight=0)
Globals.tab4_canvas.grid_rowconfigure(30, weight=0)

Globals.profiles_film_factor_input = tk.Text(Globals.tab4_canvas, width=8, height=1)
Globals.profiles_film_factor_input.grid(row=2, column=0, sticky=E, padx=(0,160), pady=(5,0))
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=18,column=0, padx=(0,0), pady=(0,0), sticky=S)
Globals.tab4_canvas.grid_columnconfigure(5, weight=0)
Globals.tab4_canvas.grid_rowconfigure(5, weight=0)
profiles_resetAll_frame.config(bg='#ffffff')

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)

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


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

#################################### DVH tab 5 ##################################################3

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, "\
SliceThickness i plan m  v re ['1','1'], ['2','2'] eller ['3','3'], Filmen m  legges i xy, xz eller yz planet (lage figur?), Filmen m  scannes  \n\
parallelt med retningene i skanneren (programmet vil anta dette), man m  markere \"opp\" og \"bort\" p  filmen. N r man skanner m  man legge
\n\
oppmerket oppover og bort merket mot h yre (kan man kreve dette i alle plan?). \
Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det, \n\
Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det, " )
DVH_explain_text.grid(row=0, column=0, columnspan=5, 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 ='#E5f9ff', relief=FLAT)

DVH_upload_film_frame = tk.Frame(Globals.tab5_canvas)
DVH_upload_film_frame.grid(row=3, column = 0, padx = (0, 240), pady=(10,0), sticky=N)
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=3, column = 0, padx = (0,40), pady=(10,0), sticky=N)
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=(160,0), pady=(10,0), sticky=N)
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

Globals.DVH_film_orientation_menu = OptionMenu(Globals.tab5_canvas, Globals.DVH_film_orientation, 'Axial', 'Coronal', 'Sagittal')
Globals.DVH_film_orientation_menu.grid(row=1, column=0, sticky=N+S, padx=(60,0))
Globals.tab5_canvas.grid_columnconfigure(2, weight=0)
Globals.tab5_canvas.grid_rowconfigure(2, weight=0)
Globals.DVH_film_orientation_menu.config(bg = '#ffffff', width=15, relief=FLAT)

film_orientation_menu_text = tk.Text(Globals.tab5_canvas, 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.grid(row=1, column=0, sticky=N+S+W, padx=(30,0), pady=(5,0))
Globals.tab5_canvas.grid_columnconfigure(3, weight=0)
Globals.tab5_canvas.grid_rowconfigure(3, weight=0)

DVH_film_orientation_help_frame = tk.Frame(Globals.tab5_canvas)
DVH_film_orientation_help_frame.grid(row=1, column=0, sticky=N+S+E, padx=(0,40))
Globals.tab5_canvas.grid_columnconfigure(6, weight=0)
Globals.tab5_canvas.grid_rowconfigure(6, weight=0)
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 = tk.Text(Globals.tab5_canvas, 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.grid(row=2, column=0, sticky=N+S+W, padx=(30,0), pady=(5,0))
Globals.tab5_canvas.grid_columnconfigure(30, weight=0)
Globals.tab5_canvas.grid_rowconfigure(30, weight=0)

Globals.DVH_film_factor_input = tk.Text(Globals.tab5_canvas, width=8, height=1)
Globals.DVH_film_factor_input.grid(row=2, column=0, sticky=E, padx=(0,160), pady=(5,0))
Globals.DVH_film_factor_input.insert(INSERT, " ")
Globals.DVH_film_factor_input.config(state=NORMAL, font=('calibri', '10'),
      bd = 2, bg='#ffffff')
Globals.tab5_canvas.grid_columnconfigure(31, weight=0)
Globals.tab5_canvas.grid_rowconfigure(31, weight=0)
"""
DVH_resetAll_frame = tk.Frame(Globals.tab5_canvas)
DVH_resetAll_frame.grid(row=15,column=0, padx=(0,0), pady=(0,0), sticky=S)
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
"""
Globals.tab5_canvas.pack(expand=True, fill=BOTH)

####################################### End statements
     ##############################################
#Globals.tab_parent.place(relwidth=1, relheight=0.9, relx=0, rely=0.15)
Globals.form.mainloop()
```

FIDORA/notebook.py

A.2 Globals.py

```
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


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 form
form = tk.Tk()

#Main—window
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)

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)

########################################   CoMet related ####################################################
global CoMet_progressbar
CoMet_progressbar = ttk.Progressbar(tab1_canvas,orient =”horizontal”, length = 550, mode =”determinate”)
CoMet_progressbar.grid(row=5, column=0, columnspan=1, sticky=W+S, pady=(27,0), 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=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
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
```

```python
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


########################################  Dose response related ###################################################
tab2_files_frame = tk.Frame(tab2_canvas)
tab2_files_frame.config(relief=FLAT, bg='#ffffff', highlightthickness=0)#, height=200, width=450)
#tab2_files_frame.grid_propagate(0)

tab2_scroll_canvas = tk.Canvas(tab2_files_frame)
tab2_scroll_canvas.config(bg='#ffffff', height=200, width=400, highlightthickness=0)
tab2_scroll_canvas.grid_propagate(0)

scroll = ttk.Scrollbar(tab2_files_frame, command=tab2_scroll_canvas.yview)

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=2, column=4, columnspan=1, rowspan=3, sticky=N)
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 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=1, column=4, columnspan=1, sticky=E+
    W+N, padx=(0,10), pady=(0,0))
tab2_canvas.grid_columnconfigure(8, weight=0)
tab2_canvas.grid_rowconfigure(8, weight=0)
dose_response_equation_frame.config(bg='#E5f9ff', relief=FLAT,
     highlightthickness=0, width=400, height=200)
dose_response_equation_frame.grid_propagate(0)


global dose_response_plot_frame
dose_response_plot_frame = tk.Frame(tab2_canvas)
dose_response_plot_frame.grid(row=1, column=0, rowspan=2, columnspan=4,
     sticky=N+S+E+W, pady=(0,5), padx=(5,5))
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=350,width=500)
dose_response_plot_frame.grid_propagate(0)

dose_response_fig = Figure(figsize=(5,3))
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('-')


########################################   Map dose related ###################################################


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   #Oppgitt ved [<,v] = [bortover, nedover]

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)

############################### Profiles ######################################

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

global profiles_film_orientation_menu

#Total doseplan
global profiles_film_dataset
#total doseplan in red channel
global profiles_film_dataset_red_channel
#Dose in whole film for red channel
global profiles_film_dataset_red_channel_dose
#Coords to ROI (will vary). Given as [index 0 from, index 0 to, index1 from, index1 to]
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=2, column=3, rowspan=25, sticky=E+W+N, pady=(0,5), padx=(5,10))
tab4_canvas.grid_columnconfigure(11, weight=0)
tab4_canvas.grid_rowconfigure(11, weight=0)
profiles_view_film_doseplan_ROI.config(bg='#E5f9ff', relief=FLAT, highlightthickness=0)


global profile_plot_canvas
profile_plot_canvas = tk.Canvas(tab4_canvas)
profile_plot_canvas.grid(row=4, column=0, rowspan=10, 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='#E5f9ff', relief=FLAT, highlightthickness=0, width=500, height=500)
profile_plot_canvas.grid_propagate(0)

profiles_fig = Figure(figsize=(5,3))
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, 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 = []

#Given from top left corner [right, down]
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_film_window
#global profiles_film_window_open
#profiles_film_window_open = False

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.pack()
profiles_choose_profile_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_choose_profile_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=LEFT, padx=(70,0))


global profiles_redefine_film_ROI_frame
profiles_redefine_film_ROI_frame = tk.Frame(profiles_choose_profile_canvas)
profiles_redefine_film_ROI_frame.pack(side=LEFT, padx=(0,100))
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='vertical')
profiles_film_panedwindow.pack()
profiles_film_panedwindow.configure(sashrelief = RAISED, showhandle=True)


#global profiles_film_tab_parent
#profiles_film_tab_parent = ttk.Notebook(profiles_film_notebook_canvas)
#profiles_film_tab_parent.borderWidth=0
#profiles_film_tab_parent.pack()

#global profiles_film_tab_image
#profiles_film_tab_image = ttk.Frame(profiles_film_tab_parent)
#profiles_film_tab_image.config(relief=FLAT)



#global profiles_film_tab_dose
#profiles_film_tab_dose = ttk.Frame(profiles_film_tab_parent)
#profiles_film_tab_dose.config(relief=FLAT, padding=[0,0,0,0])


#profiles_film_tab_parent.add(profiles_film_tab_image, text='Scanned film')
#profiles_film_tab_parent.add(profiles_film_tab_dose, text='Dose on film')

global profiles_scanned_image_text_image
global profiles_film_dose_map_text_image
global profiles_doseplan_text_image
global doseplan_write_image
global film_dose_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
############################## DVH related ############################################

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 profiles_lines
#profiles_lines = []

#global end_point
#end_point = None

#global profiles_line_coords_film
#global profiles_line_coords_doseplan
```

```
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=2, column=3, rowspan=25, sticky=E+W+N,
      pady=(0,5), padx=(5,10))
tab5_canvas.grid_columnconfigure(11, weight=0)
tab5_canvas.grid_rowconfigure(11, weight=0)
DVH_view_film_doseplan_ROI.config(bg='#E5f9ff', relief=FLAT,
     highlightthickness=0)

”””
global DVH_plot_canvas
DVH_plot_canvas = tk.Canvas(tab4_canvas)
DVH_plot_canvas.grid(row=3, column=0, rowspan=10, columnspan=2, sticky=N+E
     +W, pady=(0,5), padx=(5,10))
tab5_canvas.grid_columnconfigure(4, weight=0)
tab5_canvas.grid_rowconfigure(4, weight=0)
DVH_plot_canvas.config(bg='#E5f9ff', relief=FLAT, highlightthickness=0)

DVH_fig = Figure(figsize=(5,3))
DVH_a = profiles_fig.add_subplot(111, ylim=(0,40000), xlim=(0,500))
DVH_plot_canvas = FigureCanvasTkAgg(profiles_fig, master=
     profile_plot_canvas)
DVH_plot_canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, sticky=N
     +E+W, padx=(5,0), pady=(0,0))
DVH_a.set_title ("Profiles", fontsize=12)
DVH_a.set_ylabel("Pixel value", fontsize=12)
DVH_a.set_xlabel("Distance (mm)", fontsize=12)
DVH_fig.tight_layout()

global DVH_showPlanes_image
global DVH_showDirections_image

global DVH_depth
global DVH_depth_float
```

```
#global DVH_mark_isocenter_button_image
#global DVH_mark_ROI_button_image
#global DVH_mark_point_button_image
"""
global DVH_iscoenter_coords
DVH_iscoenter_coords = []

#Given from top left corner [right , down]
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_dataset_doseplan
global DVH_dataset_rtplan

global DVH_test_if_added_doseplan
global DVH_test_if_added_rtplan
DVH_test_if_added_doseplan = False
DVH_test_if_added_rtplan = False

global DVH_isocenter_mm


global DVH_dose_scaling_doseplan
”””
global DVH_max_dose_film


############## probably comment out this section ##############
DVH_choose_profile_canvas = tk.Canvas(DVH_view_film_doseplan_ROI)
DVH_choose_profile_canvas.pack()
DVH_choose_profile_canvas.config(bg='#ffffff', relief=FLAT,
    highlightthickness=0)
global DVH_choice_of_profile_line_type
DVH_choice_of_profile_line_type = StringVar()
DVH_choice_of_profile_line_type.set(”h”)

DVH_choose_profile_type_text = tk.Text(DVH_choose_profile_canvas, height
    =1)
DVH_choose_profile_type_text.insert(INSERT, ”How to draw the profile:”)
DVH_choose_profile_type_text.pack(side=TOP)
DVH_choose_profile_type_text.config(bg='#ffffff', relief=FLAT, \
highlightthickness=0, state=DISABLED, font=('calibri', '11'))
Radiobutton(DVH_choose_profile_canvas, text=”Horizontal”, variable=
    DVH_choice_of_profile_line_type, \
    value=”h”, bg='#ffffff', cursor='hand2').pack(side=LEFT)
Radiobutton(DVH_choose_profile_canvas, text=”Vertical”, \
    variable=DVH_choice_of_profile_line_type, value='v', bg='#ffffff',
    cursor='hand2').pack(side=LEFT)
Radiobutton(DVH_choose_profile_canvas, text=”Draw”, \
    variable=DVH_choice_of_profile_line_type, value=”d”, bg='#ffffff',
    cursor='hand2').pack(side=LEFT)
###########################################################################################
”””
global DVH_film_panedwindow
```

```
DVH_film_panedwindow = PanedWindow(DVH_view_film_doseplan_ROI, orient='vertical')
DVH_film_panedwindow.pack()
DVH_film_panedwindow.configure(sashrelief = RAISED, showhandle=True)

"""
global DVH_scanned_image_text_image
global DVH_film_dose_map_text_image
global DVH_doseplan_text_image
"""
global DVH_doseplan_write_image
global DVH_doseplan_write_image_width
global DVH_doseplan_write_image_height
"""
global DVH_doseplan_write_image_var_x
DVH_doseplan_write_image_var_x= 0


global DVH_new_window_factor_textbox ###
"""
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

############################### Correction matrix ######################################

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[:,:]
```

FIDORA/Globals.py

A.3 gloVar.py

```
#import necessary packages
import tkinter as tk
from tkinter import StringVar
import numpy as np


#make GUI-window global
global root
root = tk.Tk()


############################## initialize all global vairables ##########################################
global method
method="1"
```

```

global filename
filename=StringVar(root)
filename.set("Error!")

global dir_name
dir_name=StringVar(root)
dir_name.set("Error!")

global saveTo
saveTo=StringVar(root)
saveTo.set("Error!")

global pName
pName = StringVar(root)
pName.set("Error!")

global DPI
DPI = tk.StringVar(root)
DPI.set("127")

global comet
comet = tk.StringVar(root)
comet.set("1")

global filetype
filetype = tk.StringVar(root)
filetype.set(".dcm")

global saveAs
saveAs = tk.StringVar(root)
saveAs.set(".dcm")

global savetofolder

global Name


################################## read and globalize the correction matrices ###################################
global correction127_red
with open('output_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('output_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('output_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[:,:]

global correctedImage
correctedImage=None
```

FIDORA/gloVar.py

A.4 CorrectionFunctions.py

```
import numpy as np
import cv2
from cv2 import imread, IMREAD_ANYCOLOR, IMREAD_ANYDEPTH
from os.path import normpath, basename
import os
import gloVar
from tkinter import messagebox
import matplotlib.pyplot as plt

# Function to perform det correction using correction matrix
def correctionMatrix():
    dataset = cv2.imread(gloVar.filename.get().lstrip(), cv2.
    IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
    if(dataset is None):
        current_folder = os.getcwd()
        script_path = gloVar.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 does not contain  ,  , ")
         return

    if(dataset.shape[2] == 3):
        if(gloVar.DPI.get()=="127" and dataset.shape[0]==1270 and dataset.shape[1]==1016):
            gloVar.correctedImage = abs(dataset-gloVar.correctionMatrix127)
        elif(gloVar.DPI.get()=="72" and dataset.shape[0]==720 and dataset.shape[1]==576):
            gloVar.correctedImage = abs(dataset - gloVar.correctionMatrix72)
        else:
            messagebox.showerror("Error","The resolution of the image is not consistent with dpi:" + gloVar.DPI.get())

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

FIDORA/CorrectionFunctions.py

A.5 CoMet_functions.py

```
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

## Function to do nothing (temp)
def nothingButton():
    return

## Function to upload file
def UploadAction(event=None):
    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=(80,0))
        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', '12'), 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=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
        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!")

## Function to set dpi
def setCoMet_dpi():
    dpi = Globals.CoMet_dpi.get()
    print(dpi)
    return dpi

## Function to set the export folder chosen by the user
def setCoMet_export_folder():
    Globals.CoMet_export_folder.set(filedialog.askdirectory())
    if(Globals.CoMet_export_folder.get() == ""):
        #If this: the dialogbox was closed and no folder selected.
        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=(25,0), padx=(80,0))
        save_to_folder.insert(INSERT, basename(normpath(Globals.CoMet_export_folder.get())))
        save_to_folder.config(state=DISABLED, bd=0, font=('calibri', '12'), 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=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
        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'))


## Function to check that user has filled inn everything
def checkAllWidgets(*args):
    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


## Function to perform det correction using correction matrix
def correctionMatrix():
    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 does not contain   ,   ,   ")
         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")


## Function to perform the correction on the image
def Correct():
    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=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
        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_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=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
        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=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
        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() ) # read dicom image
    img = ds.pixel_array # get image 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')
    height, width, channels = img8.shape
    img8 = Image.fromarray(img8, 'RGB')

    img8 = img8.resize((250, 300))

    Globals.CoMet_image_to_canvas =  ImageTk.PhotoImage(image=img8)

    Globals.CoMet_print_corrected_image.create_image(123,148,image=Globals
    .CoMet_image_to_canvas)
    Globals.CoMet_print_corrected_image.image = Globals.
    CoMet_image_to_canvas
```

FIDORA/CoMet_functions.py

A.6 Dose_response _functions.py

```
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

#import sympy as sp
#from io import BytesIO

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
#matplotlib.rcParams['text.usetex'] = True #lagt til for    kunne skrive
     latex i string
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")
```

```
## Function to do nothing (temp)
def nothingButton():
    return



def saveCalibration():
    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():
        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):
    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 does not contain  ,  ,  ”)
         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():
    print(”sjekk ************************************************************”)
    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('-')

    print("sjekk2
    **********************************************************")
    fig = Figure(figsize=(5,3))
    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=(5,0), 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, 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)

            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')

            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()))
            #write_out_respons_function = tk.Text(Globals.dose_response_equation_frame)#, height=1, width=10)
            #write_out_respons_function.insert(INSERT, out_text_function )
            ##ekstra linje med standardavvik, pr ver    inserte de ogs
            #write_out_respons_function.insert(INSERT,standardavvik_rgb)
            def clickFunction(a,b,c):
                tmptext = StringVar()
                text = ”Pixel value(PV) as function of dose(D): ”
                a=str(a)   #str(round(Globals.popt_red[0]))
                b=str(b) #str(round(Globals.popt_red[1]))
                c=str(c) #str(round(Globals.popt_red[2]))
                latex= a   + ”+ ” ”\\frac {” + f”{b}” + ”}{”+ ”D” + ”−” + f”{c}” + ”}”
                avgR=str(round(Globals.dose_response_sd_avg_red.get())); minR=str(round(Globals.dose_response_sd_min_red.get())); maxR=str(round(Globals.dose_response_sd_max_red.get()))
                latexR=”(”+avgR+”,”+minR+”,”+maxR+”)”; textR=”\n\nStandard deviations (SD): \nSD for red color channel: (avg, max,min)=”
                avgG=str(round(Globals.dose_response_sd_avg_green.get())); minG=str(round(Globals.dose_response_sd_min_green.get())); maxG=str(round(Globals.dose_response_sd_max_green.get()))
                latexG=”(”+avgG+”,”+minG+”,”+maxG+”)”; textG=”\n\nSD for green color channel: (avg, max,min)=”
                avgB=str(round(Globals.dose_response_sd_avg_blue.get())); minB=str(round(Globals.dose_response_sd_min_blue.get())); maxB=str(round(Globals.dose_response_sd_max_blue.get()))
                latexB=”(”+avgB+”,”+minB+”,”+maxB+”)”; textB=”\n\nSD for blue color channel: (avg, max,min)=”

                tmptext.set(latex)

                #tmptext = entry.get()
                tmptext = ”$”+tmptext.get()+”$”

                axLatex.clear()
                axLatex.text(0.01, 0.3, text+”PV = ”+tmptext+textR+latexR+textG+latexG+textB+latexB, fontsize = 4)  #this is where the text is added to the axis
                canvasLatex.draw()

            #root = tk.Tk()
            #make a frame and place it with grid
            #mainframe = Frame(root)
            #mainframe.grid(row=0,column=0)

            #make a label and place it with grid
            labelLatex = Label(Globals.dose_response_equation_frame)
            labelLatex.grid(row=0,column=0)

            figLatex = matplotlib.figure.Figure(figsize=(2.4, 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”)   # (side=TOP, fill=BOTH, expand=1)

```

```
            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])
            clickFunction(a_,b_,c_)

            #displayButton = Button(Globals.dose_response_equation_frame, text="display equation",width=15,command=lambda: clickFunction(12,3,4))
            #displayButton.grid(row=1,column=0,sticky="N")

            #write_out_respons_function.grid(row=0, column=0, sticky=N+S+W+E, pady=(5,5), padx=(5,5))
            #Globals.dose_response_equation_frame.grid_columnconfigure(0, weight=0)
            #Globals.dose_response_equation_frame.grid_rowconfigure(0, weight=0)
            #write_out_respons_function.config(state=DISABLED, bd=0, font=('calibri', '12'), bg='#ffffff')
            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):
    #The button index equals the index in Globals.avg_red_vector etc.
    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):
    return a + b/(D-c)



## Function to find mean of uploaded images with same dose.
def avgAllFiles(write_dose_box, new_window):
    #First block is to test that everything is filled in and as expected.
    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

    #Calculates the mean in each color channel
    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():
    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)
    new_window_scroll_canvas.grid_propagate(0)

    new_window_scroll = ttk.Scrollbar(new_window_frame, command=
    new_window_scroll_canvas.yview)

    scrollable_frame= tk.Frame(new_window_scroll_canvas)

    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=(10,10), 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')

```

```python
    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():
    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()
```

FIDORA/Dose_response_functions.py

A.7 Map_dose.py

```
############################ Map dose ####################
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
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
import os
import sys
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

def dose_to_pixel(D,a,b,c):
    return a + b/(D-c)

def pixel_to_dose(P,a,b,c):
    return c + b/(P-a)


#### LEgg til medianfilter
def calculate_dose_map(cv2Img):
    wid = Globals.map_dose_ROI_x_end.get() - Globals.map_dose_ROI_x_start.
    get()
    heig = Globals.map_dose_ROI_y_end.get() - Globals.map_dose_ROI_y_start
    .get()
    print(wid, heig)
    doseMap_film = np.zeros((heig,wid))
    for i in range(heig):
        for j in range(wid):
            doseMap_film[i,j] = pixel_to_dose(cv2Img[Globals.
    map_dose_ROI_y_start.get()+i,Globals.map_dose_ROI_x_start.get()+j,2],
    \
                Globals.popt_red[0], Globals.popt_red[1], Globals.popt_red
    [2])


    fig = Figure(figsize=(0.8,0.8))
    a = fig.add_subplot(111)
    #test ane:
    #plot_image = cv2.flip(doseMap_film,−1) #fjern test etterp
    plot_image = a.pcolormesh(doseMap_film, cmap='viridis', rasterized=
    True, vmin=0, vmax=600)
    fig.colorbar(plot_image, ax=a)
    canvas_dosemap_film = FigureCanvasTkAgg(fig,master = Globals.tab3)
    canvas_dosemap_film.get_tk_widget().place(relwidth=0.6, relheight
    =0.55, relx = 0.03, rely=0.2)#relwidth=0.3, rely=0.1
    canvas_dosemap_film.draw()
    #plotte dosekartet (dette m  v re krympet (408,508))


def prepare_Image():
    cv2Img = cv2.imread(Globals.map_dose_film_dataset.get(), cv2.
    IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
    if(cv2Img is None):
        current_folder = os.getcwd()
        parent = os.path.dirname(Globals.map_dose_film_dataset.get())
        os.chdir(parent)
        cv2Img=cv2.imread(basename(normpath(Globals.map_dose_film_dataset.
    get())), cv2.IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
        os.chdir(current_folder)
    if(cv2Img is None):
         messagebox.showerror(”Error”, ”Something has happen. Check that
    the filename does not contain  ,  ,  ”)
         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)
        elif(cv2Img.shape[0]==720 and cv2Img.shape[1]==576):
            cv2Img = abs(cv2Img − Globals.correctionMatrix72)
            cv2Img = np.clip(cv2Img, 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”)
        return

    #Read last calibration done, or ask if one wish to change
    choose_batch_window = tk.Toplevel(Globals.tab3)
    choose_batch_window.geometry(”800x400”)
    choose_batch_window.grab_set()

    def set_batch():
        choose_batch_window.destroy()
        f = open('calibration.txt', 'r')
        lines = f.readlines()
        words = lines[Globals.map_dose_film_batch.get()].split()
        Globals.popt_red[0] = float(words[3])
        Globals.popt_red[1] = float(words[4])
        Globals.popt_red[2] = float(words[5])
        f.close()
        calculate_dose_map(cv2Img)

    batch_cnt = 0
    r = open('calibration.txt', 'r')
    lines = r.readlines()
    write_batch_y_coord = 0.3
    for l in lines:
        words = l.split()
        line = "Batch nr.  : " + words[2] + ".     Date:    " + words[0] + " " + words[1] + "."
        write_batch = tk.Text(choose_batch_window, width=1, height=1)
        write_batch.place(relwidth=0.7, relheight=0.1, relx = 0.1, rely=write_batch_y_coord)
        write_batch.insert(INSERT, line)
        write_batch.config(state=DISABLED, bd = 0, font=('calibri', '12'))

        Radiobutton(choose_batch_window, text='',cursor='hand2',font=('calibri', '14'), \
            variable=Globals.map_dose_film_batch, value=batch_cnt).place(relwidth=0.08, \
                relheight=0.1, relx=0.8, rely=write_batch_y_coord)

        write_batch_y_coord+=0.1; batch_cnt+=1

    ok_batch_button = tk.Button(choose_batch_window, text='OK', cursor='hand2',\
        font=('calibri', '14'), overrelief=GROOVE, state=tk.ACTIVE, width = 15, command=set_batch)
    ok_batch_button.place(relwidth=0.2, relheight=0.2, relx=0.4, rely=0.9)
    r.close()

def draw_ROI(img, scale_horizontal, scale_vertical):
    draw_ROI_window = tk.Toplevel(Globals.tab3)
    draw_ROI_window.grab_set()
    local_frame= Frame(draw_ROI_window, bd = 2, relief=SUNKEN)
    local_frame.grid_rowconfigure(0,weight=1)
    local_frame.grid_columnconfigure(0, weight=1)

    local_canvas = Canvas(local_frame, bd=0)
    local_canvas.grid(row=0,column=0, sticky=N+S+E+W)

    w = 10 + img.width()
    h = 10 + img.height()
    draw_ROI_window.geometry("%dx%d+0+0" % (w, h))

    local_canvas.create_image(0,0,image=img,anchor="nw")
    local_canvas.config(scrollregion=local_canvas.bbox(ALL), cursor='arrow')
    local_canvas.image= img

    rectangle = local_canvas.create_rectangle(0,0,0,0,outline='green')

    def buttonPushed(event):
        Globals.map_dose_ROI_x_start.set(event.x)
        Globals.map_dose_ROI_y_start.set(event.y)

    def buttonMoving(event):
        local_canvas.coords(rectangle, Globals.map_dose_ROI_x_start.get(),
     Globals.map_dose_ROI_y_start.get(), \
            event.x, event.y)

    def buttonReleased(event):
        Globals.map_dose_ROI_x_end.set(event.x)
        Globals.map_dose_ROI_y_end.set(event.y)
        local_canvas.coords(rectangle, Globals.map_dose_ROI_x_start.get(),
     Globals.map_dose_ROI_y_start.get(),\
            Globals.map_dose_ROI_x_end.get(), Globals.map_dose_ROI_y_end.
    get())
        local_canvas.itemconfig(rectangle, outline=’Blue’)
        answer = messagebox.askquestion(”Question”,”Happy with placement?”
    , parent=draw_ROI_window)
        if(answer==’yes’):
            Globals.map_dose_ROI_x_start.set(Globals.map_dose_ROI_x_start.
    get()*scale_horizontal)
            Globals.map_dose_ROI_y_start.set(Globals.map_dose_ROI_y_start.
    get()*scale_vertical)
            Globals.map_dose_ROI_x_end.set(Globals.map_dose_ROI_x_end.get
    ()*scale_horizontal)
            Globals.map_dose_ROI_y_end.set(Globals.map_dose_ROI_y_end.get
    ()*scale_vertical)
            prepare_Image()
            draw_ROI_window.destroy()

    local_canvas.bind(”<B1-Motion>”, buttonMoving)
    local_canvas.bind(”<Button-1>”, buttonPushed)
    local_canvas.bind(”<ButtonRelease-1>”, buttonReleased)

    local_frame.pack(fill=’both’, expand=1)


def draw_image_with_marks(img, scale_horizontal, scale_vertical,
    mark_isocenter_window, frame):
    #check_isocenter_window = tk.Toplevel(Globals.tab3)
    #check_isocenter_window.grab_set()
    #frame_local = Frame(mark_isocenter_window, bd=2, relief=SUNKEN) #
    check_isocenter_window, bd=2, relief=SUNKEN)
    #frame_local.grid_rowconfigure(0, weight=1)
    #frame_local.grid_columnconfigure(0, weight=1)
    canvas_local = Canvas(frame, bd=0)
    canvas_local.grid(row=0, column=0, sticky=N+S+E+W)

    #w = 10 + img.width()
    #h = 10 + img.height()
    #check_isocenter_window.geometry(”%dx%d+0+0” % (w, h))

    canvas_local.create_image(0,0,image=img,anchor=”nw”)
    canvas_local.config(scrollregion=canvas_local.bbox(ALL), cursor=’arrow
    ’)
    canvas_local.image= img
    canvas_local.create_oval(Globals.
    map_dose_isocenter_map_x_coord_unscaled[0]-2, Globals.
    map_dose_isocenter_map_y_coord_unscaled[0]-2,\
        Globals.map_dose_isocenter_map_x_coord_unscaled[0]+2, Globals.
    map_dose_isocenter_map_y_coord_unscaled[0]+2, fill='red')
    canvas_local.create_oval(Globals.
    map_dose_isocenter_map_x_coord_unscaled[1]-2, Globals.
    map_dose_isocenter_map_y_coord_unscaled[1]-2, \
        Globals.map_dose_isocenter_map_x_coord_unscaled[1]+2, Globals.
    map_dose_isocenter_map_y_coord_unscaled[1]+2, fill='red')
    canvas_local.create_oval(Globals.
    map_dose_isocenter_map_x_coord_unscaled[2]-2, Globals.
    map_dose_isocenter_map_y_coord_unscaled[2]-2,\
        Globals.map_dose_isocenter_map_x_coord_unscaled[2]+2, Globals.
    map_dose_isocenter_map_y_coord_unscaled[2]+2, fill='red')
    canvas_local.create_oval(Globals.
    map_dose_isocenter_map_x_coord_unscaled[3]-2, Globals.
    map_dose_isocenter_map_y_coord_unscaled[3]-2,\
        Globals.map_dose_isocenter_map_x_coord_unscaled[3]+2, Globals.
    map_dose_isocenter_map_y_coord_unscaled[3]+2, fill='red')

    canvas_local.create_line(Globals.
    map_dose_isocenter_map_x_coord_unscaled[0], Globals.
    map_dose_isocenter_map_y_coord_unscaled[0]\
        , Globals.map_dose_isocenter_map_x_coord_unscaled[1], Globals.
    map_dose_isocenter_map_y_coord_unscaled[1], \
            fill='purple', smooth=1, width=2)
    canvas_local.create_line(Globals.
    map_dose_isocenter_map_x_coord_unscaled[2], Globals.
    map_dose_isocenter_map_y_coord_unscaled[2]\
        , Globals.map_dose_isocenter_map_x_coord_unscaled[3], Globals.
    map_dose_isocenter_map_y_coord_unscaled[3], \
            fill='purple', smooth=1, width=2)

    x1 = Globals.map_dose_isocenter_map_x_coord_unscaled[0]
    x2 = Globals.map_dose_isocenter_map_x_coord_unscaled[1]
    x3 = Globals.map_dose_isocenter_map_x_coord_unscaled[2]
    x4 = Globals.map_dose_isocenter_map_x_coord_unscaled[3]
    y1 = Globals.map_dose_isocenter_map_y_coord_unscaled[0]
    y2 = Globals.map_dose_isocenter_map_y_coord_unscaled[1]
    y3 = Globals.map_dose_isocenter_map_y_coord_unscaled[2]
    y4 = Globals.map_dose_isocenter_map_y_coord_unscaled[3]



    if(y1==y2 and y3==y4):
        messagebox.showerror("Error", "Reference points are not correct.
    Try again.")
        check_isocenter_window.destroy()
        upload_film_data()
    elif(y1==y2):
        if(x1==x2):
            messagebox.showerror("Error", "Reference points are not
    correct. Try again.")
            check_isocenter_window.destroy()
            upload_film_data()
        else:
            a = 0; b=y1
            if(x3==x4):
                isocenter = [x3,y1]
            else:
                c=(y3-y4)/(x3-x4); d = y3 - c*x3
                isocenter = [(d-b)/(a-c), b]
    elif(y3==y4):
        if(x3==x4):
            messagebox.showerror("Error", "Reference points are not
    correct. Try again.")
            check_isocenter_window.destroy()
            upload_film_data()
        else:
            c = 0; d = y3
            if(x1==x2):
                isocenter = [x1,y3]
            else:
                a = (y1-y2)/(x1-x2); b = y1 - a*x1
                isocenter = [(d-b)/(a-c), d]
    else:
        if(x1==x2 and x3==x4):
            messagebox.showerror("Error", "Reference points are not
    correct. Try again.")
            check_isocenter_window.destroy()
            upload_film_data()
        elif(x1==x2):
            c = (y3-y4)/(x3-x4); d = y3 - c*x3
            isocenter = [x1, c*x1+d]
        elif(x3==x4):
            a = (y1-y2)/(x1-x2); b = y1 - a*x1
            isocenter = [x3, a*x3+d]
        else:
            a = (y1-y2)/(x1-x2)
            b = y1 - a*x1
            c = (y3-y4)/(x3-x4)
            d = y3 - c*x3
            isocenter = [(d-b)/(a-c), a*(d-b)/(a-c) + b]

    #frame.pack(fill='both', expand=1)
    if(isocenter[0] < 0 or isocenter[1] < 0 or isocenter[0] > 408 or
    isocenter[1] > 508):
        messagebox.showerror("Error", "Reference points are not correct.
    Try again.")
        mark_isocenter_window.destroy() #check_isocenter_window.destroy()
        upload_film_data()
    else:
        canvas_local.create_oval(isocenter[0]-6, isocenter[1]-6, isocenter
    [0]+6,isocenter[1]+6, outline="pink")
        answer = messagebox.askquestion("Question","Happy with placement?"
    , parent=mark_isocenter_window)#check_isocenter_window)
        if(answer=="yes"):
            Globals.map_dose_isocenter_film = [isocenter[0]*
    scale_horizontal, isocenter[1]*scale_vertical]
            mark_isocenter_window.destroy() #check_isocenter_window.
    destroy()
            draw_ROI(img, scale_horizontal, scale_vertical)

        else:
            mark_isocenter_window.destroy() #check_isocenter_window.
    destroy()
            upload_film_data()
            return







def upload_film_data():
    current_folder = os.getcwd()
    os.chdir(os.path.dirname(sys.argv[0]))
    img = Image.open(Globals.map_dose_film_dataset.get())
    if(not (img.width == 1016 or img.width == 576)):
        messagebox.showerror(”Error”, ”Dpi in image has to be 127 or 72”)
        return

    Globals.map_dose_isocenter_map_x_coord_scaled = []
    Globals.map_dose_isocenter_map_x_coord_unscaled = []
    Globals.map_dose_isocenter_map_y_coord_scaled = []
    Globals.map_dose_isocenter_map_y_coord_unscaled = []

    mark_isocenter_window = tk.Toplevel(Globals.tab3)
    mark_isocenter_window.grab_set()
    frame = Frame(mark_isocenter_window, bd=2, relief=SUNKEN)
    frame.grid_rowconfigure(0, weight=1)
    frame.grid_columnconfigure(0, weight=1)
    canvas = Canvas(frame, bd=0)
    canvas.grid(row=0, column=0, sticky=N+S+E+W)


    scale_horizontal = img.width/408
    scale_vertical = img.height/508
    img = img.resize((408,508))
    img = ImageTk.PhotoImage(image=img)
    os.chdir(current_folder)
    canvas.image = img

    w = 10 + img.width()
    h = 10 + img.height()
    mark_isocenter_window.geometry(”%dx%d+0+0” % (w, h))
    canvas.create_image(0,0,image=img,anchor=”nw”)
    canvas.config(scrollregion=canvas.bbox(ALL), cursor='sb_up_arrow')
    #x_coor = []
    #y_coor = []

    def findCoords(event):
        Globals.map_dose_isocenter_map_x_coord_scaled.append(event.x*
    scale_vertical)
        Globals.map_dose_isocenter_map_y_coord_scaled.append(event.y*
    scale_horizontal)
        Globals.map_dose_isocenter_map_x_coord_unscaled.append(event.x)
        Globals.map_dose_isocenter_map_y_coord_unscaled.append(event.y)
        canvas.create_oval(event.x-2, event.y-2, event.x+2, event.y+2, fill='red')
        if (len(Globals.map_dose_isocenter_map_x_coord_scaled)==1):
            canvas.config(cursor='sb_down_arrow')
        elif(len(Globals.map_dose_isocenter_map_x_coord_scaled)==2):
            canvas.config(cursor='sb_right_arrow')
        elif(len(Globals.map_dose_isocenter_map_x_coord_scaled)==3):
            canvas.config(cursor='sb_left_arrow')
        else:
            #mark_isocenter_window.destroy()
            draw_image_with_marks(img, scale_horizontal, scale_vertical, mark_isocenter_window, frame)



    canvas.bind("<Button 1>",findCoords)
    frame.pack(fill='both', expand=1)

def UploadAction(type, event=None):
    if(type == "FILM"):
        if(Globals.popt_red[0]==1):
            messagebox.showerror("Error", "No calibration has been found. To a calibration first.")
            return
        Globals.map_dose_film_dataset.set(filedialog.askopenfilename())
        ext = os.path.splitext(Globals.map_dose_film_dataset.get())[-1].lower()
        if(ext==".tif"):
            upload_film_data()
            return
        elif(ext==""):
            Globals.map_dose_film_dataset.set("Error!")
        else:
            messagebox.showerror("Error", "The file must be a .tif file")
            Globals.map_dose_film_dataset.set("Error!")


#laste opp bilde og markere i bildene, egen funksjon
#gammatest, lese opp p  det og implementere
#Eksportere figurer og dataset ut av programmet
#m  lagre siste kalibrering (sp rre hvilken kalibrering bruker vil bruke)
# hvordan er doseplanene lagret.
#Endre geometrien slik at den passer alle skjermer. Kan man bruke skjermst rrelsen i en algoritme?


# Laste opp doseplan (for n  er det en enkel matrise, selvkonstruert.)
# laste opp skannet film, korriger automatisk.
# brukeren spesifiserse posisjon p  film
# gj re film om til dose map (bruke dose response)
# tegne dose plan og dose map fra film
# regne gamma
# tegne gamma pass/fail og variasjoner
# skriv ut all info vi f r fra gammatest
```

FIDORA/Map_Dose.py

A.8 Profile_functions.py

```
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


#Bresenham's line algorithm

def clearAll():
    Globals.profiles_film_orientation.set('–')
    Globals.profiles_film_orientation_menu.config(state=ACTIVE, bg = '#
    ffffff', width=15, relief=FLAT)

    #Globals.profiles_depth.config(state=NORMAL, fg='black')
    #Globals.profiles_depth.delete('1.0', END)
    #Globals.profiles_depth.insert(INSERT, ” ”)

    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 = []


    #if(Globals.profiles_isocenter_check and Globals.profiles_ROI_check):
    #    Globals.profiles_done_button.config(state=DISABLED)
    Globals.profiles_isocenter_check = False
    Globals.profiles_ROI_check = False
    Globals.profiles_reference_point_check = False
    Globals.profiles_ROI_reference_point_check = False

    #if(Globals.profiles_film_window_open):
    #    Globals.profiles_film_window.destroy()
    #    Globals.profiles_film_window_open = 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.tab4_canvas.unbind("<Up>")
    Globals.tab4_canvas.unbind("<Down>")


    return

def getCoordsInRandomLine(x1,y1,x2,y2):
    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
    # Reverse the list if the coordinates were reversed
    if rev:
        points.reverse()

    return points


def drawProfiles(even):
    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=(5,3))
        a = fig.add_subplot(111)

        a.axis(ymin=0,ymax=6.2)

        plot_canvas = FigureCanvasTkAgg(fig, master=Globals.profile_plot_canvas)
        plot_canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, sticky=N+E+W+S, padx=(5,0), pady=(0,0))
        #annotation = a.annotate(”HEI”, xy=(0,0), xytext=(0,20))
        #annotation.set_visible(False)
        #txt = tk.Text(Globals.profile_plot_canvas, width=50, height=6)
        #txt.insert(INSERT, ” ”)
        #txt.grid(row=1, column = 1, sticky=N+E+W+S, pady=(5,0), padx=(5,0))
        #txt.config(bg='#ffffff', font=('calibri', '10'), state=DISABLED, relief=FLAT, bd= 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=1, 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))
        #a.text(0,0, ””, fontsize=7, bbox=dict(facecolor='gray', alpha=0.1))
        #txt.set_visible(False)
        v_line = a.axvline(x=0, ymin=0, ymax=50, c='gray')
        #v_line.set_visible(False)

        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]  #Globals.profiles_doseplan_dataset_ROI
            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


            print(dx, dy)
            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


        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
                #if Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]:
                #    idx_doseplan = int(np.round(idx_doseplan/1))
                #elif Globals.profiles_dataset_doseplan.PixelSpacing == [2, 2]:
                #    idx_doseplan = int(np.round(idx_doseplan/2))
                ##else:
                #    idx_doseplan = np.round(idx_doseplan/3)
                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_ylim(y_min,y_max)
                #v_line.set_ymax = y
                #v_line.set_ymax = y_max # =
                #v_line = a.axvline(x=dist, ymin=0, ymax=40, c='gray')
                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_dosemap = Globals.film_dose_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_dosemap)
        Globals.profiles_lines.append(line_film)

        def up_button_pressed(event):
            temp_x = Globals.doseplan_write_image_var_x - 5
            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/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_dose_write_image.coords(line_film_dosemap, 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):
                #Outside the frame
                return
            #Inside the frame
            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_dose_write_image.coords(line_film_dosemap,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_dosemap = Globals.film_dose_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_dosemap)
        Globals.profiles_lines.append(line_film)

        def up_button_pressed(event):
            temp_x = Globals.doseplan_write_image_var_x - 10
            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/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_dose_write_image.coords(line_film_dosemap, 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):
                #Outside the frame
                return
            #Inside the frame
            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_dose_write_image.coords(line_film_dosemap,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_dosemap = Globals.film_dose_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_dosemap)
        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_dose_write_image.coords(line_film_dosemap, 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_dose_write_image.coords(line_film_dosemap,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_dosemap = Globals.film_dose_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_dosemap)
        Globals.profiles_lines.append(line_film)

        def left_button_pressed(event):
            temp_y = Globals.doseplan_write_image_var_y - 5
            if(temp_y < 0):
                #Outside the frame
                return
            #inside the frame
            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_dose_write_image.coords(line_film_dosemap, 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):
                #Outside the frame
                return
            #Inside the frame
            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_dose_write_image.coords(line_film_dosemap,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_dosemap = Globals.film_dose_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_dosemap)
    Globals.profiles_lines.append(line_film)

    def left_button_pressed(event):
        temp_y = Globals.doseplan_write_image_var_y - 10
        if(temp_y < 0):
            #Outside the frame
            return
        #inside the frame
        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_dose_write_image.coords(line_film_dosemap, 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):
            #Outside the frame
            return
        #Inside the frame
        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_dose_write_image.coords(line_film_dosemap ,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_dosemap = Globals.film_dose_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_dosemap)
    Globals.profiles_lines.append(line_film)

    def left_button_pressed(event):
        temp_y = Globals.doseplan_write_image_var_y - 15
        if(temp_y < 0):
            #Outside the frame
            return
        #inside the frame
        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_dose_write_image.coords(line_film_dosemap,
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):
            #Outside the frame
            return
        #Inside the frame
        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_dose_write_image.coords(line_film_dosemap,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_dose_write_image.delete(Globals.profiles_lines[1])
            Globals.film_write_image.delete(Globals.profiles_lines[2])
            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_dosemap = Globals.film_dose_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_dosemap)
        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_dose_write_image.coords(line_film_dosemap, 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_dose_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_dose_write_image.coords(line_film_dosemap, 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_dose_write_image.bind("<ButtonRelease-1>", mouseReleased)
    Globals.film_dose_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_dose_write_image.delete(Globals.profiles_lines[1])
            Globals.film_write_image.delete(Globals.profiles_lines[2])
            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_dosemap = Globals.film_dose_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_dosemap)
        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_dose_write_image.coords(line_film_dosemap, 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_dose_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_dose_write_image.coords(line_film_dosemap,
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_dose_write_image.bind("<ButtonRelease-1>",
mouseReleased)
    Globals.film_dose_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_dose_write_image.delete(Globals.profiles_lines[1])
            Globals.film_write_image.delete(Globals.profiles_lines[2])
            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_dosemap = Globals.film_dose_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_dosemap)
        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_dose_write_image.coords(line_film_dosemap, 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_dose_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_dose_write_image.coords(line_film_dosemap, 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_dose_write_image.bind(”<ButtonRelease−1>”,
    mouseReleased)
        Globals.film_dose_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):
    test_drawProfiles()


def test_drawProfiles():
    if Globals.profiles_dataset_doseplan == None:
        return
    else:
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
        Globals.film_write_image.delete(Globals.profiles_lines[2])
        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):
    if not line_orient == ’d’:
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
        Globals.film_write_image.delete(Globals.profiles_lines[2])
    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):
    if not line_orient == ’d’:
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
        Globals.film_write_image.delete(Globals.profiles_lines[2])
    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):
    if not line_orient == 'd':
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
        Globals.film_write_image.delete(Globals.profiles_lines[2])
    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):
    if not line_orient == ’d’:
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
        Globals.film_write_image.delete(Globals.profiles_lines[2])
    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):
    if not line_orient == ’d’:
        Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
        Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
        Globals.film_write_image.delete(Globals.profiles_lines[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_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):
    ret = c + b/(P-a)
    return ret

def processDoseplan_usingReferencePoint(only_one):

    ################  RT Plan ######################

    #Find each coordinate in mm to isocenter relative to first element in doseplan
    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])
    #Given as [x,y,z] in patient coordinates
    Globals.profiles_isocenter_mm = [iso_1, iso_2, iso_3]

    #Reads input displacement from phantom on reference point in film
    #lateral = Globals.profiles_input_lateral_displacement.get("1.0",'end-1c')
    #vertical = Globals.profiles_input_vertical_displacement.get("1.0", 'end-1c')
    #longit = Globals.profiles_input_longitudinal_displacement.get("1.0",'end-1c')
    #if(lateral==" "):lateral=0
    #if(vertical==" "):vertical=0
    #if(longit==" "):longit=0
    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]):
        #make isocenter coordinates into pixel values
        isocenter_px[0] = np.round(iso_1)
        isocenter_px[1] = np.round(iso_2)
        isocenter_px[2] = np.round(iso_3)

        #find the pixel distance from reference point to ROI corners
        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])
    ])

        #Input to px
        lateral_px = np.round(lateral)
        vertical_px = np.round(vertical)
        longit_px = np.round(longit)

        #displacment to px
        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]):
        #make isocenter coordinates into pixel values
        isocenter_px[0] = np.round(iso_1/2)
        isocenter_px[1] = np.round(iso_2/2)
        isocenter_px[2] = np.round(iso_3/2)

        #find the pixel distance from reference point to ROI corners
        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)])

        #Input to px
        lateral_px = np.round(lateral/2)
        vertical_px = np.round(vertical/2)
        longit_px = np.round(longit/2)

        #displacment to pc
        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:
        #make isocenter coordinates into pixel values
        isocenter_px[0] = np.round(iso_1/3)
        isocenter_px[1] = np.round(iso_2/3)
        isocenter_px[2] = np.round(iso_3/3)

        #find the pixel distance from reference point to ROI corners
        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)])

        #Input to px
        lateral_px = np.round(lateral/3)
        vertical_px = np.round(vertical/3)
        longit_px = np.round(longit/3)

        #displacment to pc
        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)

    #Finding reference point in doseplan
    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 ##################################
    #dataset_swapped is now the dataset entered the same way as expected
    with film (slice, rows, columns)
    #isocenter_px and reference_point is not turned according to the
    doseplan and film orientation.
    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'):
            #number of frames -> rows
            #rows -> number of frames
            #columns -> columns
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,1)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            #column -> number of frames
            #number of frames -> rows
            #rows -> columns
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,2)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            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_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            #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 = 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]),:,:]



    #calculate the coordinates of the Region of Interest in doseplan (marked on the film)
    #and checks if it actualy exists in dosematrix

    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)#cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5)))
        elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
            Globals.profiles_several_img.append(img)#cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10)))
        else:
            Globals.profiles_several_img.append(img)#cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15)))


def processDoseplan_usingIsocenter(only_one):

    ################  RT Plan ######################

    #Find each coordinate in mm to isocenter relative to first element in doseplan
    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])
    #Given as [x,y,z] in patient coordinates
    Globals.profiles_isocenter_mm = [iso_1, iso_2, iso_3]


    #Isocenter in pixel relative to the first element in the doseplan
    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)#np.round(Globals.profiles_isocenter_mm[0])
        isocenter_px[1] = np.round(iso_2)#np.round(Globals.profiles_isocenter_mm[1])
        isocenter_px[2] = np.round(iso_3)#np.round(Globals.profiles_isocenter_mm[2])

        #Change distance in film to pixel in doseplan
        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)#np.round(Globals.
    profiles_isocenter_mm[0]/2)
        isocenter_px[1] = np.round(iso_2/2)#np.round(Globals.
    profiles_isocenter_mm[1]/2)
        isocenter_px[2] = np.round(iso_3/2)#np.round(Globals.
    profiles_isocenter_mm[2]/2)


        #Change distance in film to pixel in doseplan
        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)#np.round(Globals.
    profiles_isocenter_mm[0]/3)
        isocenter_px[1] = np.round(iso_2/3)#np.round(Globals.
    profiles_isocenter_mm[1]/3)
        isocenter_px[2] = np.round(iso_3/3)#np.round(Globals.
    profiles_isocenter_mm[2]/3)

        #Change distance in film to pixel in doseplan
        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 ##################################
    #dataset_swapped is now the dataset entered the same way as expected
    with film (slice, rows, columns)
    #isocenter_px and reference_point is not turned according to the
    doseplan and film orientation.
    if(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[1, 0,
    0, 0, 1, 0]):
        #reference_point[1] = isocenter_px[0]
        #reference_point[2] = isocenter_px[1]
        #reference_point[0] = isocenter_px[2]
        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’):
            #number of frames -> rows
            #rows -> number of frames
            #columns -> columns
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,1)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        elif(Globals.profiles_film_orientation.get()==’Sagittal’):
            #column -> number of frames
            #number of frames -> rows
            #rows -> columns
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,2)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            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_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            #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 = 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[1] = isocenter_px[0]
        #reference_point[2] = isocenter_px[1]
        #reference_point[0] = isocenter_px[2]
        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’):
            #columns -> number of frames
            #number of frames -> columns
            #rows -> rows
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,2)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            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'):
            #rows -> number of frames
            #number of frames -> rows
            #columns -> columns
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,1)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            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[1] = isocenter_px[0]
        #reference_point[2] = isocenter_px[1]
        #reference_point[0] = isocenter_px[2]
        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'):
            #rows -> columns
            #columns -> number of frames
            #number of frames -> rows
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,2)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            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_iso = isocenter_px[1]
            #isocenter_px[1] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            #number -> rows
```

```
                #colums -> colums
                #rows -> number of frames
                dataset_swapped = np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
                #temp_iso = isocenter_px[0]
                #isocenter_px[0] = isocenter_px[1]
                #isocenter_px[1] = temp_iso
                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'):
                #column -> rows
                #rows -> column
                #number of frames -> number of frames
                dataset_swapped = np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 1,2)
                #temp_iso = isocenter_px[1]
                #isocenter_px[1] = isocenter_px[2]
                #isocenter_px[2] = temp_iso
                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[1] = isocenter_px[0]
            #reference_point[2] = isocenter_px[1]
            #reference_point[0] = isocenter_px[2]
            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'):
                #rows -> rows
                #columns -> number of frames
                #number of frames ->columns
                dataset_swapped = np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,2)
                #temp_iso = isocenter_px[0]
                #isocenter_px[0] = isocenter_px[2]
                #isocenter_px[2] = temp_iso
                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'):
```

```
            #number of frames -> columns
            #columns -> rows
            #rows -> number of frames
            dataset_swapped = np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            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_iso = isocenter_px[1]
            #isocenter_px[1] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            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[1] = isocenter_px[0]
        #reference_point[2] = isocenter_px[1]
        #reference_point[0] = isocenter_px[2]
        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'):
            #rows -> columns
            #columns -> rows
            #number of frames -> number of frames
            dataset_swapped = np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 1,2)
            #temp_iso = isocenter_px[1]
            #isocenter_px[1] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            temp_ref = reference_point[1]
            reference_point[1] = reference_point[2]
            reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            #rows -> number of frames
            #columns -> rows
            #number of frames -> columns
            dataset_swapped = np.swapaxes(Globals.profiles_dataset_doseplan.pixel_array, 0,1)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            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_iso = isocenter_px[1]
            #isocenter_px[1] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            #temp_ref = reference_point[1]
            #reference_point[1] = reference_point[2]
            #reference_point[2] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Axial'):
            #rows -> columns
            #colums -> number of frames
            #number of frames -> rows
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,1)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            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_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            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[1] = isocenter_px[0]
        #reference_point[2] = isocenter_px[1]
        #reference_point[0] = isocenter_px[2]
        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'):
            #rows -> number of frames
            #columns ->rows
            #number of frames -> columns
            dataset_swapped = np.swapaxes(Globals.
    profiles_dataset_doseplan.pixel_array, 0,2)
            #temp_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[2]
            #isocenter_px[2] = temp_iso
            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_iso = isocenter_px[0]
            #isocenter_px[0] = isocenter_px[1]
            #isocenter_px[1] = temp_iso
            temp_ref = reference_point[0]
            reference_point[0] = reference_point[1]
            reference_point[1] = temp_ref
        elif(Globals.profiles_film_orientation.get()=='Sagittal'):
            #rows -> columns
            #columns -> rows
            #number of frames -> number of frames
            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
    ###############################

    #Pick the slice where the reference point is (this is the slice-
    position of the film)

    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)]



    #calculate the coordinates of the Region of Interest in doseplan (
    marked on the film)
    #and checks if it actualy exists in dosematrix

    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])

    #dose_slice = cv2.flip(dose_slice, 1)
    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)]
        """
        if(Globals.profiles_number_of_doseplans == 1):
            Globals.profiles_doseplan_dataset_ROI_several = img
            Globals.profiles_number_of_doseplans+=1

            if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
                Globals.profiles_several_img = cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5))
            elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
                Globals.profiles_several_img = cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10))
            else:
                Globals.profiles_several_img = cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15))

        else:
            Globals.profiles_doseplan_dataset_ROI_several += img
            Globals.profiles_number_of_doseplans+=1

            if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
                Globals.profiles_several_img += cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5))
            elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
                Globals.profiles_several_img += cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10))
            else:
                Globals.profiles_several_img += cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15))
        """
        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)#cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5)))
        elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
            Globals.profiles_several_img.append(img)#cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10)))
        else:
            Globals.profiles_several_img.append(img)#cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15)))
```

```
def UploadRTplan():
    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

    #Isocenter given in mm from origo in patient coordinate system
    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
    #if(Globals.profiles_test_if_added_doseplan):
    #    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
    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 \non
    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):
    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
    #Check that the resolution is either 1x1x1, 2x2x2 or 3x3x3
    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
    #Check that the datamatrix is in right angles to the coordinate system
    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;


####################################################### F I L M #######################################################
def markIsocenter(img, new_window_isocenter_tab, image_canvas, cv2Img):
    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):

    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):
    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)
```

```python
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')
```

```
        ### Husk at koordinatene g r bortover s  nedover! Top left - top right - bottom left - bottom right
        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 does not contain  ,  ,  ”)
            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
            img_film = Globals.profiles_film_dataset_ROI_red_channel_dose
            img_film = img_film/mx_film
            PIL_img_film = Image.fromarray(np.uint8(cm.viridis(img_film)*255))

            scaled_image_visual_film = ImageTk.PhotoImage(image=PIL_img_film)
            Globals.film_dose_write_image.create_image(0,0,image=scaled_image_visual_film, anchor=”nw”)
            Globals.film_dose_write_image.image =
scaled_image_visual_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
            film_dose_map_image_text_canvas.create_image(0,0, image=
Globals.profiles_film_dose_map_text_image, anchor="nw")
            film_dose_map_image_text_canvas.image=Globals.
profiles_film_dose_map_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')
        #PIL_img_ROI = Image.fromarray((img_ROI_RGB * 255).astype(np.
uint8), 'RGB')
        wid = PIL_img_ROI.width;heig = PIL_img_ROI.height
        #film_window_write_image = tk.Canvas(film_window_scroll_frame)

        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())

        film_dose_canvas = tk.Canvas(Globals.profiles_film_panedwindow)
        film_dose_canvas.grid(row=1,column=0, sticky=N+S+W+E)
        Globals.profiles_film_panedwindow.add(film_dose_canvas, \
            height=max(heig,Globals.profiles_film_dose_map_text_image.height()), \
                width=wid + Globals.profiles_film_dose_map_text_image.width())
        film_dose_canvas.config(bg='#ffffff', relief=FLAT, highlightthickness=0, \
            height=max(heig,Globals.profiles_film_dose_map_text_image.height()), \
                width=wid + Globals.profiles_film_dose_map_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)

        Globals.film_dose_write_image = tk.Canvas(film_dose_canvas)
        Globals.film_dose_write_image.grid(row=0,column=1,sticky=N+S+W+E)
        Globals.film_dose_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())

        film_dose_map_image_text_canvas=tk.Canvas(film_dose_canvas)
        film_dose_map_image_text_canvas.grid(row=0,column=0,sticky=N+S+W+E)
        film_dose_map_image_text_canvas.config(bg='#ffffff', relief=FLAT, highlightthickness=0, \
            height=Globals.profiles_film_dose_map_text_image.height(), width=Globals.profiles_film_dose_map_text_image.width())

        scaled_image_visual = PIL_img_ROI
        scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual)

        #film_window_write_image.create_image(0,0,image=scaled_image_visual,anchor=”nw”)
        #film_window_write_image.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)

        #Beregne avstand mellom ROI og isocenter gitt i mm
        # [top left[mot venstre, oppover], top right[mot venstre (h yre blir negativ), oppover], bottom left, bottom right]
        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 plot_profiles():

    return


def help_showPlanes():
    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)



def help_showDepth():
    new_window = tk.Toplevel(Globals.tab4)
    w = Globals.profiles_showDirections_image.width()
    h = Globals.profiles_showDirections_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_showDirections_image, anchor='nw')
    canvas.pack(expand=True, fill=BOTH)
```

FIDORA/Profile_functions.py