

Master's thesis

2020

Ane Vigre Håland

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



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 XR-QA2 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 dose 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 dose i behandlingsplanleggingssystemet.

FIDORA ble anvendt til å undersøke oppbyggingsdosen i målbrystet og dose til motsatt bryst. Oppbyggingsdistansen i målbrystet ble målt fra inngangsdosen ved hudens overflate, til dose naddde 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 og 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 dose til motsatte brystet. Denne besparende effekten viste seg dog å være mindre enn antatt. Istedenfor ble det observert en betydelig besparende effekt i dose til motsatt bryst i en filter-fri VMAT behandlingsplan. Denne planen viste seg å redusere dose til motsatt bryst ved minst 38%, sammenlignet med en tangentiell field-in-field plan.

Table of Contents

Preface	1
Abstract	i
Sammendrag	i
Table of Contents	iv
List of Figures	xi
Abbreviations	xii
1 Introduction	1
1.1 External radiotherapy in cancer treatment	1
1.2 Radiochromic film	2
1.3 Aim of project	2
2 Background	3
2.1 Physical principals of radiation therapy	3
2.1.1 Types of ionizing radiation	3
2.1.2 Biological effects of radiation	4
2.2 Interaction of radiation with matter	5
2.2.1 5 R's of radiobiology and fractionation	7
2.3 Dosimetry	9
2.3.1 Dosimetric quantities for photons	9
2.3.2 Depth dose profiles	13
2.3.3 Dosimetric quantities for electrons	14
2.3.4 Measuring dose in tissue	15
2.3.5 Dosimeter	17
2.4 Film dosimetry	19
2.4.1 Radiochromic film	19

2.4.2	GafChromic EBT3 film	19
2.4.3	GafChromic XR-QA2 film	21
2.4.4	Optical density	21
2.4.5	Flat-bed scanner	22
2.4.6	Image processing with Python	22
2.5	Evaluation metrics	23
2.5.1	CT	23
2.5.2	Volume definitions	23
2.5.3	Dose and volume parameters	24
2.6	Treatment planning and delivery	25
2.6.1	Treatment planning system	25
2.6.2	Patient coordinate system in RayStation	25
2.6.3	Delivery of photons	26
2.6.4	IMRT and VMAT	29
2.6.5	Treatment planning of breast cancer	30
2.7	Systems for verification of dose distribution	31
2.7.1	Point dosimeters	31
2.7.2	Film	32
2.7.3	Chemical dosimeters (gel)	32
3	Materials and Methods	33
3.1	Workflow in film dosimetry	33
3.1.1	GafChromic film	33
3.1.2	Calibration	33
3.1.3	Scanning and correction method	35
3.1.4	Image processing	37
3.2	Film dosimetry with FIDORA	37
3.2.1	Correction Method, CoMet	38
3.2.2	Dose-response	39
3.2.3	Map dose	40
3.2.4	Profiles	41
3.3	Experimental setup, treatment planning and field arrangements	42
3.3.1	Phantom	42
3.3.2	Volume definitions	45
3.3.3	Irradiation techniques	45
4	Results	49
4.1	FIDORA	49
4.1.1	CoMet	50
4.1.2	Dose-response	50
4.1.3	Map dose	52
4.1.4	Profiles	53
4.2	Experimental results	56
4.2.1	Correction method	56
4.3	Experimental results obtained with FIDORA	58
4.3.1	Calibration curves	58

4.3.2	Calibration curve for GafChromic EBT3 film	59
4.3.3	Calibration curve for GafChromic XR-QA2 film	62
4.3.4	Validation of calibration curve	62
4.3.5	Build-up in target breast	64
4.3.6	Dose to the contralateral breast	69
5	Discussion	73
5.1	Scanner parameters and correction matrix	73
5.2	FIDORA	73
5.2.1	Calibration curve	74
5.2.2	Dose to the contralateral breast	75
5.2.3	Build-up dose to target breast	76
5.3	Future work	77
5.3.1	GafChromic XR-QA2 film	77
5.3.2	FIDORA	77
6	Conclusion	79
Bibliography		81
Appendix		85
APPENDICES		85
A	FIDORA	86
A.1	notebook.py	87
A.2	Globals.py	111
A.3	gloVar.py	129
A.4	CorrectionFunctions.py	131
A.5	CoMet_functions.py	132
A.6	Dose_response _functions.py	137
A.7	Map_dose.py	151
A.8	Profile_functions.py	159

List of Figures

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].	5
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].	7
2.3	Summary of the repair of sublethal damage as evidenced by a split-dose experiment. Courtesy of [17].	8
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].	10
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]. .	10
2.6	An illustration of the charged particle equilibrium situation, where $E_{in}^n = E_{out}^n$, and thus $\epsilon = E_{tr}^n$. Courtesy of [26].	12

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].	13
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.	14
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].	17
2.10	A principle sketch of a cylindrical ionization chamber. Courtesy of [26].	18
2.11	A drawing of the composition of the EBT3 GafChromic film. It is composed of an active layer (of thickness 25 μm) and two outer polyester layers (both of thickness 125 μm). 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].	20
2.12	The EBT3 film exhibit two absorption bands centered at around 636 and 585 nm. Courtesy of [23].	20
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].	21
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.	24
2.15	Simple schematic diagram of a linear accelerator. Courtesy of [33].	26
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].	27
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].	28

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.	28
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.	29
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].	30
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).	31
3.1 An illustration of standard geometry as defined in TRS-398, with a source-surface-distance of 100 cm. [11]	34
3.2 Orientation in the scanner	35
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.	36
3.4 A schematic overview of the program FIDORA.	38
3.5 A schematic overview of the tab CoMet in FIDORA, responsible for performing a correction method on the scanned image.	39
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.	40
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.	41
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.	42

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.	43
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.	44
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.	44
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.	45
3.13	A standard tangential plan setup, consisting of medial (blue) and lateral (orange) fields with aligned posterior field borders.	46
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.	47
4.1	A print screen of the film dosimetry software, FIDORA, developed during this project.	49
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.	50
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.	51
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.	52
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.	53
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.	53

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. . .	54
4.8	A print screen of the isocenter method in the Profiles tab in FIDORA. . .	55
4.9	A print screen of the reference point method in the Profiles tab in FIDORA.	55
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.	56
4.11	Profiles of the deviation in pixel value compared to the center across the scanner surface in lateral direction for different dose levels.	57
4.12	Profiles of the deviation in pixel value compared to the center across the scanner surface in scanning direction for different dose levels.	57
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. . .	58
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)).	59
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)).	60
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)).	61

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)).	62
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.	64
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.	65
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.	65
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.	66
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.	66
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.	67
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.	67

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.	68
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.	68
4.27 Horizontal, vertical and diagonal profiles drawn through the CLB in the isocentre plane, shown as a red line in the corresponding images.	70
4.28 Horizontal, vertical and diagonal profiles of V1 (tangential FiF). The grey vertical line indicates the position along the profile that was evaluated.	70
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.	71
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.	71
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.	72

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

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 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 follows 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].

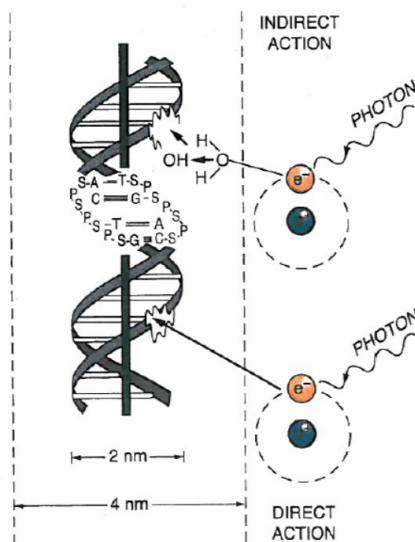


Figure 2.1: Direct and indirect actions of radiation, illustrated with an ejected orbital electron and free radicals. The DNA helix consists of two strands of nucleotides, which are composed of a nitrogenous base, a sugar group and a phosphate group. Nucleotides are linked to their neighboring nucleotides 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 elec-

tron of the absorbing material, for the reaction to occur. The ejected electron obtains the remaining energy, E_e , that the photon carried

$$E_e = h\nu - E_b \quad (2.1)$$

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 incident 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.

$$K = E_\gamma - E'_\gamma = E - mc^2, \quad (2.2)$$

where E_γ and E'_γ is the incident and resultant energy of the photon, respectively. E is the total energy 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, $h\nu \geq 2mc^2$, 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:

$$K_- + K_+ = E_\gamma - 2mc^2 \quad (2.3)$$

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.

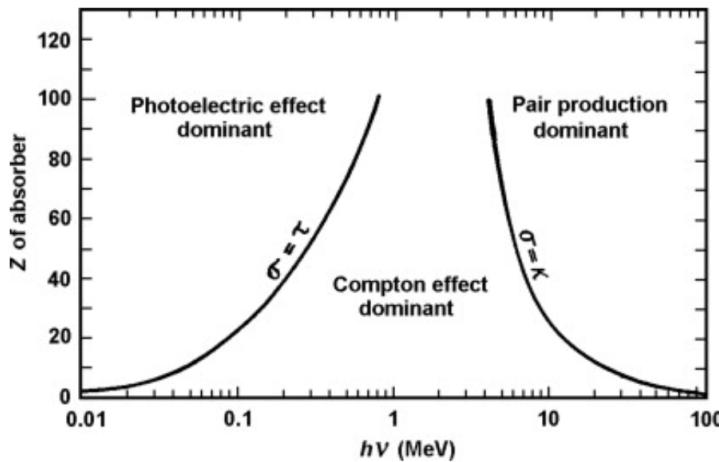


Figure 2.2: The relative importance of various processes of photons interaction with matter. Which process dominates depends on the absorbing material Z , and the photon energy $h\nu$. At the solid lines the cross-sections of two processes are equal. Here σ , τ and κ represents the cross-section of the 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 radiation. Reassortment or redistribution is the progression of cells through the cell cycle. Most of the cells surviving a fraction are 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.

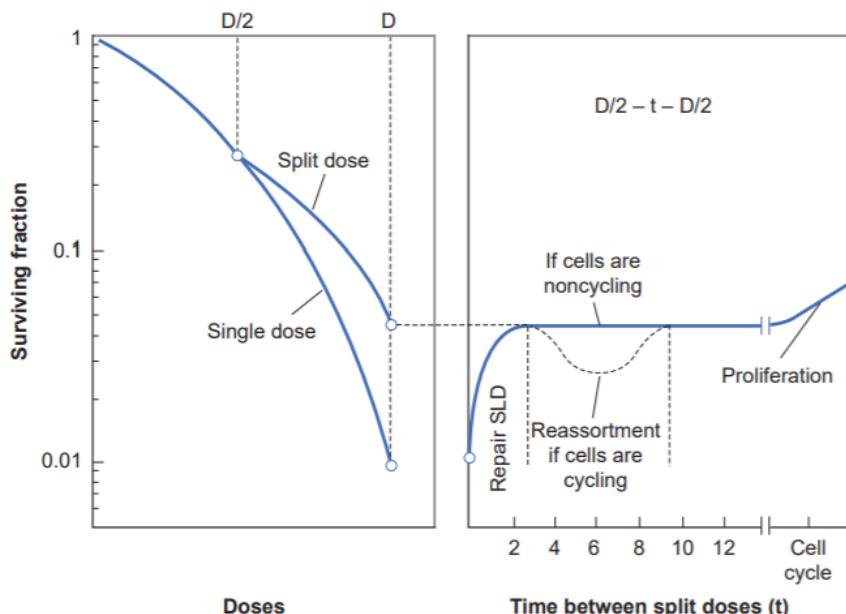


Figure 2.3: Summary of the repair of sublethal damage as evidenced by a split-dose experiment. Courtesy of [17].

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 simplification 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 cho-

sen 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],

$$D = \frac{d\bar{\epsilon}}{dm} = \lim_{m \rightarrow 0} \frac{\bar{\epsilon}}{m} = \lim_{V \rightarrow 0} \frac{1}{\rho V} \bar{\epsilon} \quad (2.4)$$

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/m³), 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

$$\epsilon = R_{in} - R_{out} + \sum Q, \quad (2.5)$$

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.

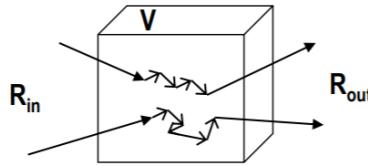


Figure 2.4: An illustration of the particles entering and leaving a volume, V . Note how the energy is not necessarily absorbed at the point of interaction, where it was transferred. Small arrows indicate how the creation of secondary electrons interact and deposit energy along their path, in many steps.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

$$\Phi = \frac{dN}{dA}. \quad (2.6)$$

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,

$$\Phi = \frac{\sum \delta s}{dV}, \quad (2.7)$$

as illustrated in Figure 2.5.

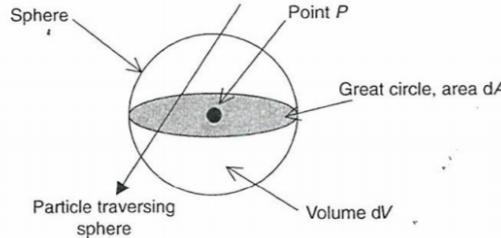


Figure 2.5: Illustration of a particle striking a finite sphere surrounding point P , with the sphere reduced to an infinitesimal one at P with a cross section of dA . The direction of the radiation is not taken into account. Courtesy of [26].

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

$$\frac{\mu_{tr}}{\rho} = \frac{1}{\rho} \frac{dE_{tr}}{ENdl}, \quad (2.8)$$

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,

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

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,

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

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 ,

$$K = \frac{\mu_{tr}}{\rho} E \Phi = \frac{\mu_{tr}}{\rho} \Psi \quad (2.11)$$

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,

$$\epsilon = R_{in} + R_{out}. \quad (2.12)$$

One can also express the mean imparted energy as,

$$\epsilon = E_{tr}^n - E_{out}^n + E_{in}^n, \quad (2.13)$$

where $E_{tr}^n = E_{tr}(1 - g)$. If the electron track that leaves the layer is replaced by an identical track that enters the layer, one has that $E_{in}^n = E_{out}^n$, and thus $\epsilon = E_{tr}^n$, 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} ,

$$D \stackrel{\text{CPE}}{=} K_{col} = \frac{\mu}{\rho} E \Phi, \quad (2.14)$$

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.

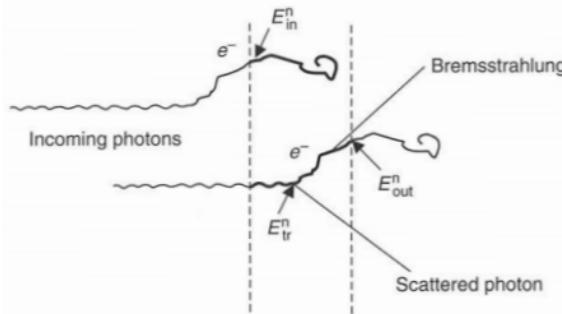


Figure 2.6: An illustration of the charged particle equilibrium situation, where $E_{in}^n = E_{out}^n$, and thus $\epsilon = E_{tr}^n$. Courtesy of [26].

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.

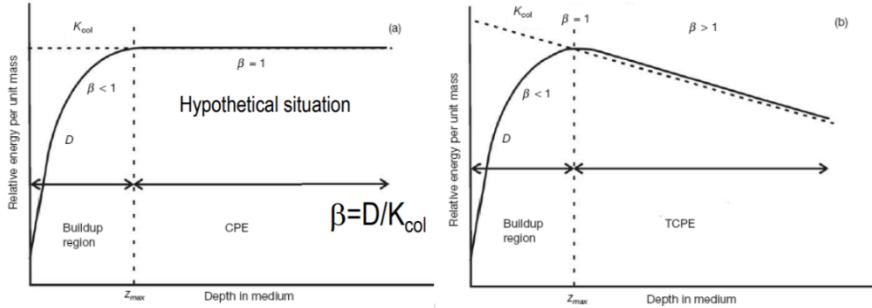


Figure 2.7: Collision kerma and absorbed dose as a function of depth in a medium irradiated by a high energy photon beam for a) the hypothetical case of no photon attenuation or scattering and for b) the realistic case. The x-axis represents the depth in medium, and the build-up region is defined as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

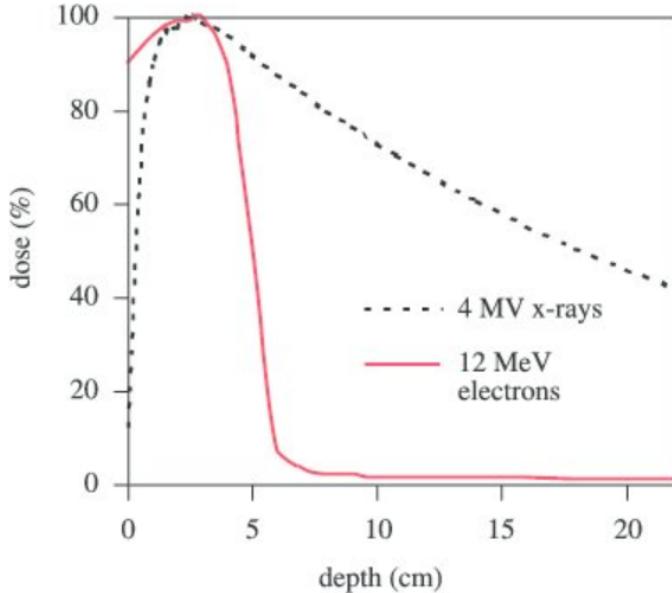


Figure 2.8: Photon and electron depth-dose curves. Courtesy of [30]. The surface dose (at depth 0 cm) is about 75%-95% of the maximum dose for an electron beam and only about 10%-30% for a photon (x-ray) beam.

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 ,

$$S = \frac{dE}{dx}. \quad (2.15)$$

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 .

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

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,

$$dE = S_{col} dl N. \quad (2.17)$$

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,

$$\frac{dE}{dm} = \frac{S_{col}}{\rho} \frac{N dl}{dV} = \frac{S_{col}}{\rho} \Phi. \quad (2.18)$$

This gives us the expression,

$$D = \frac{S_{col}}{\rho} \Phi, \quad (2.19)$$

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,

$$D \stackrel{\text{d-eqm}}{=} \frac{S_{col}}{\rho} \Phi [26], \quad (2.20)$$

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,

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

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],

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

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,

$$D_{med} = f_Q D_{det} \quad (2.23)$$

as illustrated in Figure 2.9.

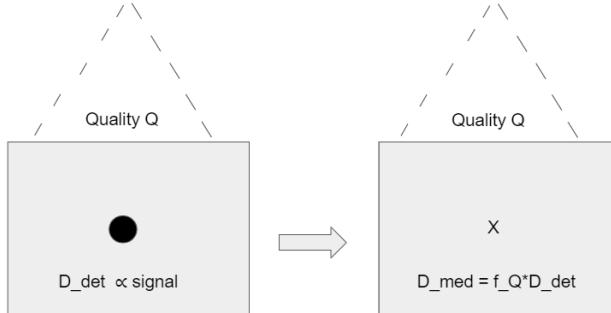


Figure 2.9: Illustration of measurement in cavity, given that the detector is large enough for CPE to exist. When the detected signal, being proportional to the absorbed dose to the detector, D_{det} , is found, one can calculate the absorbed dose to the material, $D_{med} = f_Q D_{det}$, for a given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

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

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].

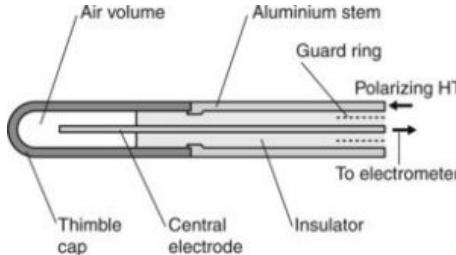


Figure 2.10: A principle sketch of a cylindrical ionization chamber. Courtesy of [26].

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 ionization 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].

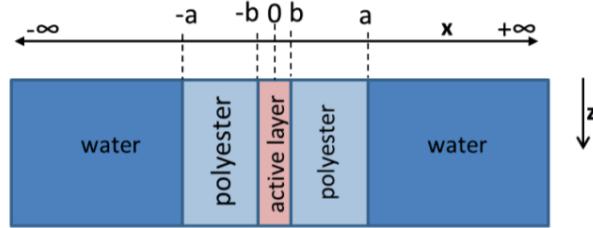


Figure 2.11: A drawing of the composition of the EBT3 GafChromic film. It is composed of an active layer (of thickness 25 μm) and two outer polyester layers (both of thickness 125 μm). The active layer consists of crystals filled with a monomer (diacetylene: Lithium pentacosa-10,12-dynoate (LiPCDA)), 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.

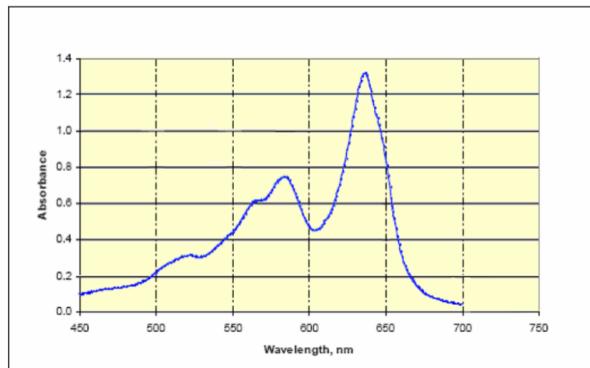


Figure 2.12: The EBT3 film exhibit two absorption bands centered at around 636 and 585 nm. Courtesy of [23].

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.

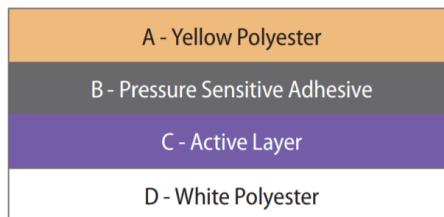


Figure 2.13: GafChromic XR-QA2 film consists of five layers: a 97-m-thick yellow polyester layer, 15-m-thick pressure-sensitive adhesive layer, 25-m-thick active layer, 3-m-thick surface layer, and 97-m-thick opaque white polyester layer [6].

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 medium's 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

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

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:

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

where PV is the pixel value that is read by the scanner. The value 2^{16} refers 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

$$HU = \frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}} \times 1000 \quad (2.27)$$

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

$$TM = \sqrt{IM^2 + SM^2} \quad (2.28)$$

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.

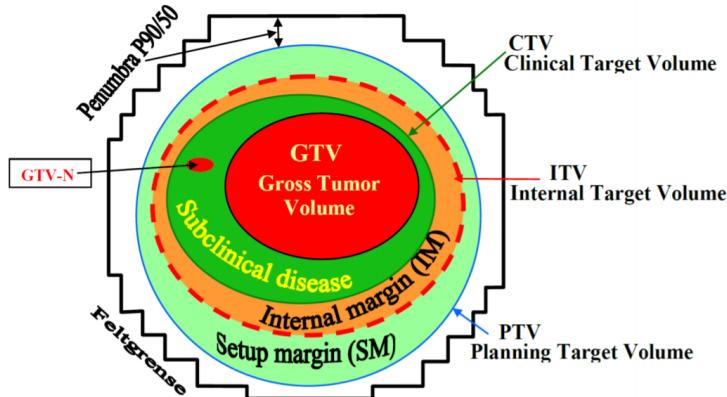


Figure 2.14: Graphical representation of various volume definitions, given by the NRPA [24]. The penumbra is defined as the distance between the 90% and 50% dose levels on a cross-section dose profile that is perpendicular to the central axis at a given depth. The GTV-N is a GTV for a lymph node placed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

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

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):

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

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

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

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 planning 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.

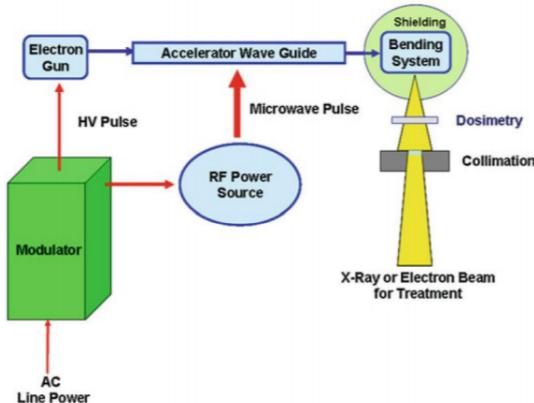


Figure 2.15: Simple schematic diagram of a linear accelerator. Courtesy of [33].

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.

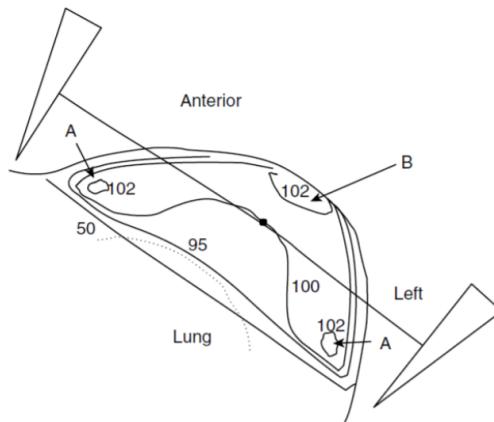


Figure 2.16: Illustration of tangential beams setup for irradiation of breast. The posterior borders of the field are aligned to avoid divergence into the lung, and wedges are used to achieve a more uniform dose distribution. The isodose curves drawn in the figure indicate how much depth dose, relative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

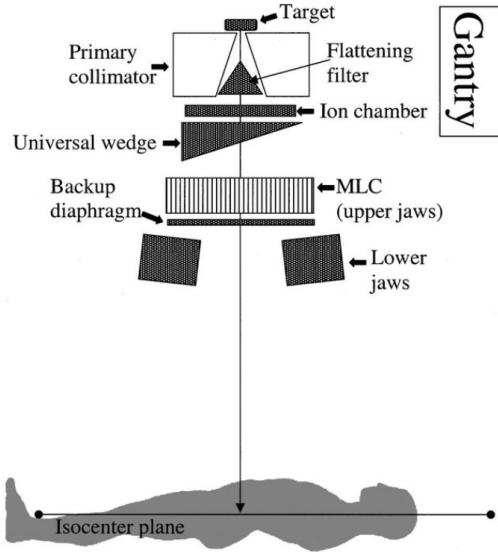


Figure 2.17: A schematic diagram of the different beam shaping features available in the linear accelerator. This is a simplified side view of the gantry head with collimator angle at 0° in relationship to the patient lying on the treatment table. When retracted, the MLC collimator is lateral to the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

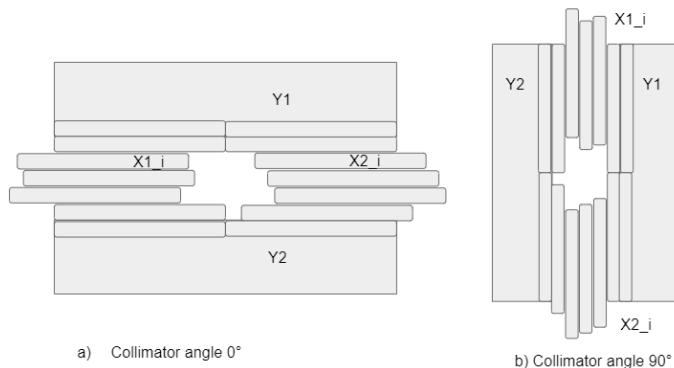


Figure 2.18: An illustration of the collimators in the gantry of the linear accelerator. The multi-leaf collimators (MLCs), $X1_i$ and $X2_i$, and the jaws, $Y1$ and $Y2$, are shown in the Beam's eye view (BEV) at a) 0° collimator angle and b) 90° collimator angle. The MLCs and the jaws enables conformal 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].

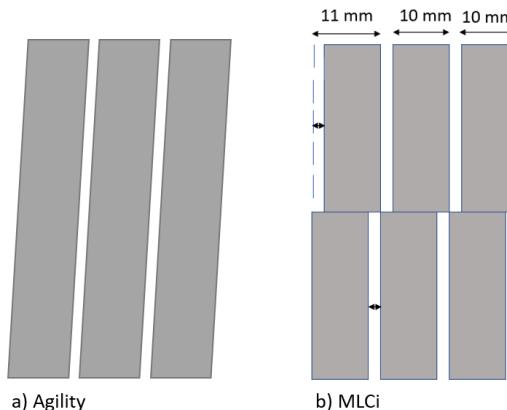


Figure 2.19: An illustration of two different designs of the Elekta multi-leaf collimator. The agility and MLCi model are illustrated. For the MLCi model the tongue width can be characterized by the horizontal distance from the upper left end, to the lower left end on an individual MLC leaf. The groove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

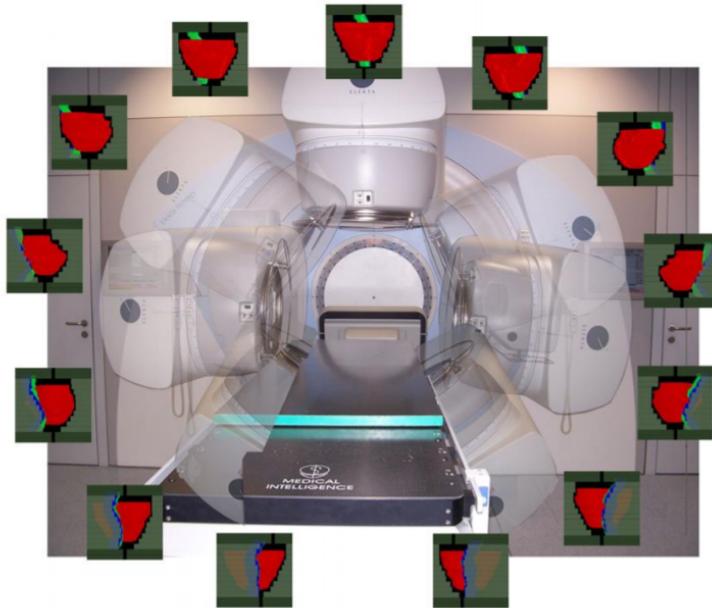


Figure 2.20: An illustration of the VMAT treatment technique. The gantry is moving almost continuously, and the collimators are shifting shape to conform the radiation to the target volume. Courtesy of [10].

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 $V_{18} \text{ Gy} \leq 15\%$ to the lungs, for a fractionation regime yielding $2.67 \text{ Gy} \times 15$ [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.

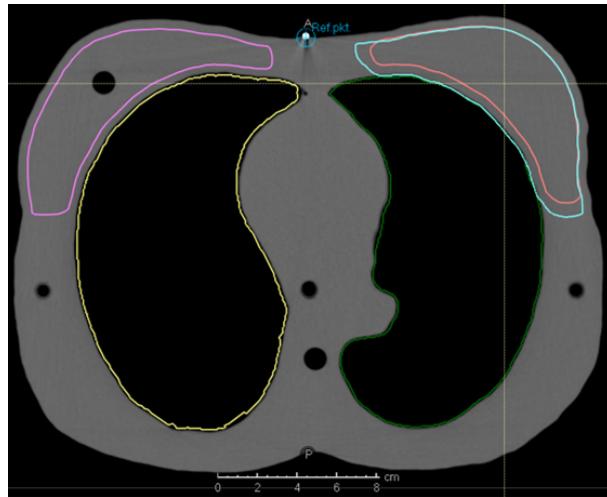


Figure 2.21: A CT image including the delineation of important regions of interest when irradiating a breast. Target volumes: the CTV of the left breast (pink), the PTV of the left breast (blue), and organs at risk: the left lung (green), the right lung (yellow) and the contralateral breast (purple).

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 phantom, 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.

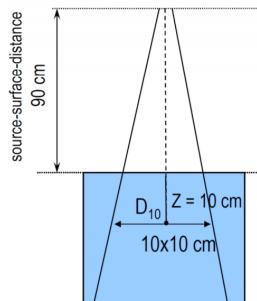


Figure 3.1: An illustration of standard geometry as defined in TRS-398, with a source-surface-distance of 100 cm. [11]

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]:

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

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].

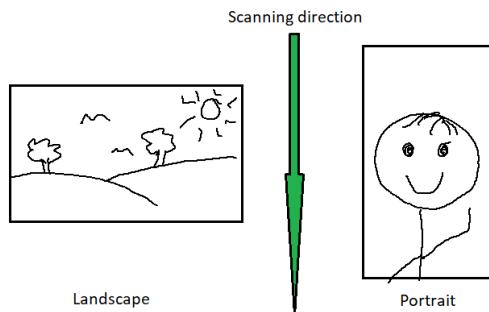


Figure 3.2: Orientation in the scanner

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 were 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 were 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 were 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.

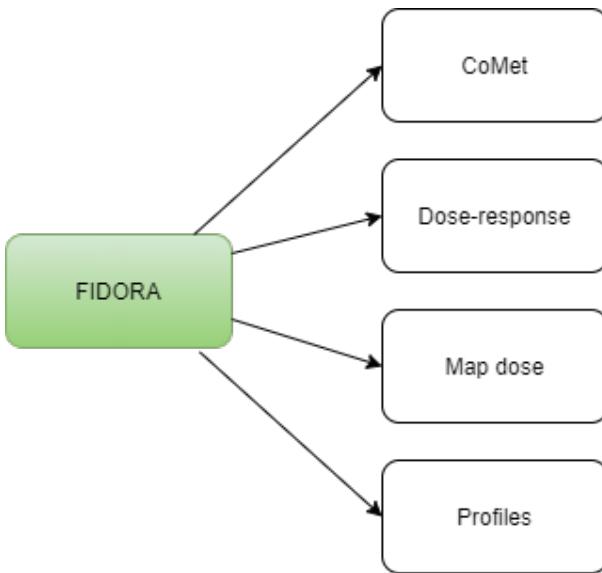


Figure 3.4: A schematic overview of the program FIDORA.

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.

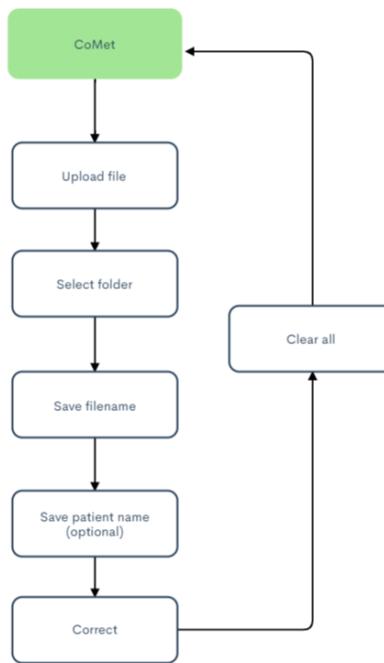


Figure 3.5: A schematic overview of the tab CoMet in FIDORA, responsible for performing a correction method on the scanned image.

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.

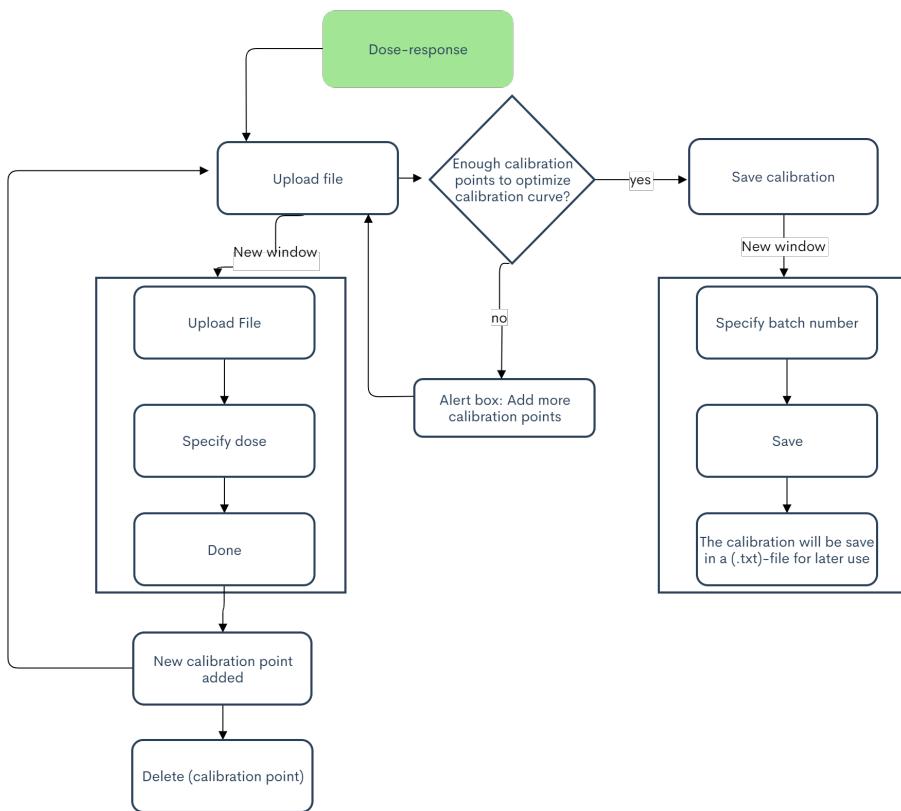


Figure 3.6: A diagram of the tab Dose-response in FIDORA, which is responsible for establishing a calibration curve. The user can upload calibration points, and when there are enough points for the program to optimize a calibration curve, the calibration curve for each color channel will be plotted and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

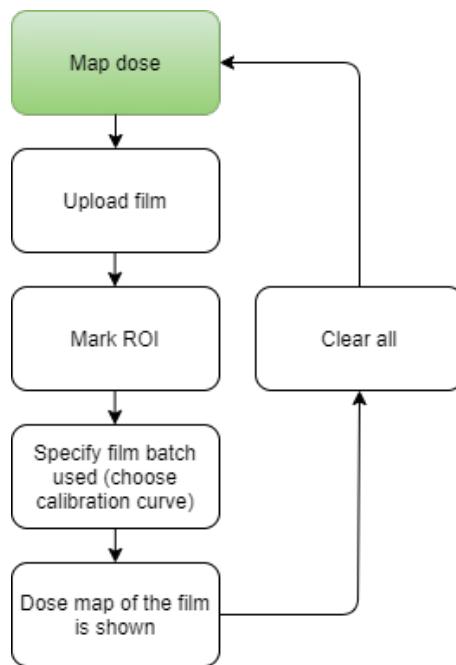


Figure 3.7: A schematic overview of the tab Map dose in FIDORA, which enables the user to upload a scanned film, choose a calibration curve and map the pixel values to dose values in a chosen ROI.

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.

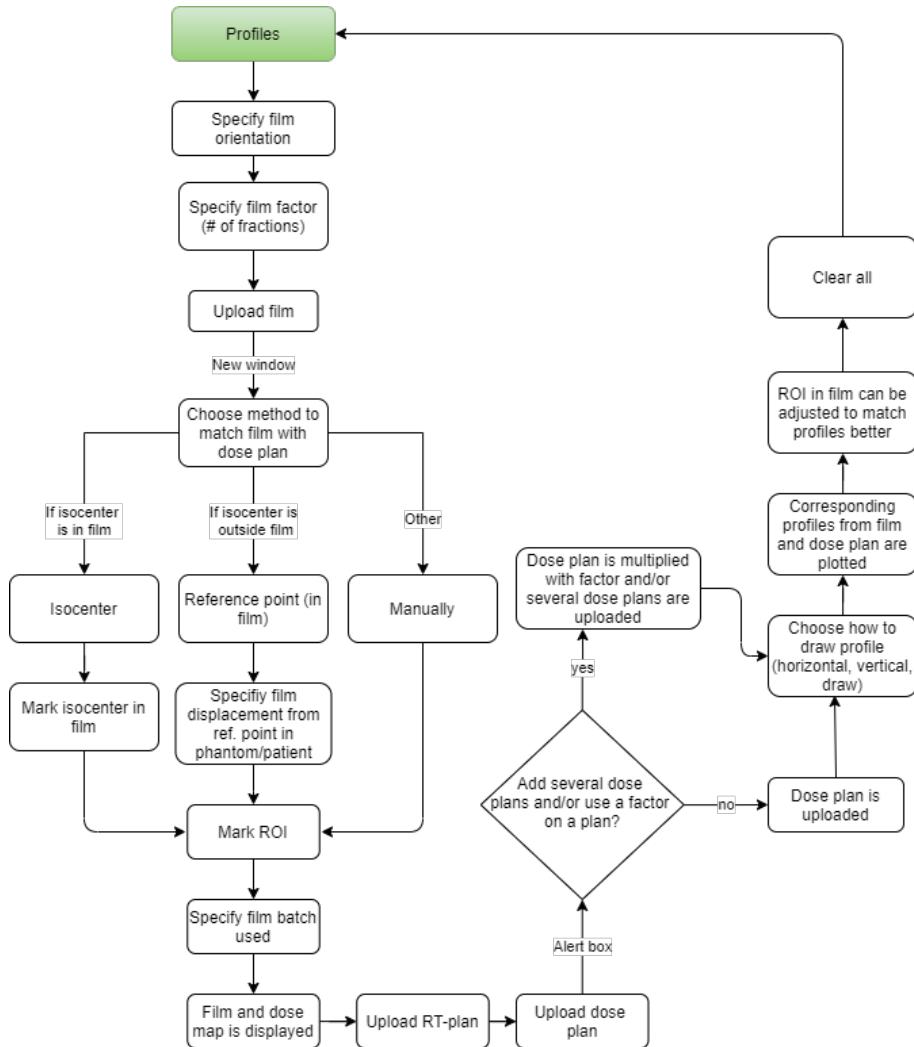


Figure 3.8: A schematic overview of the tab Profiles in FIDORA, which is responsible for plotting profiles that enables comparison of the film and the dose plan matrix along a profile of choice.

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 1.045 g/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.

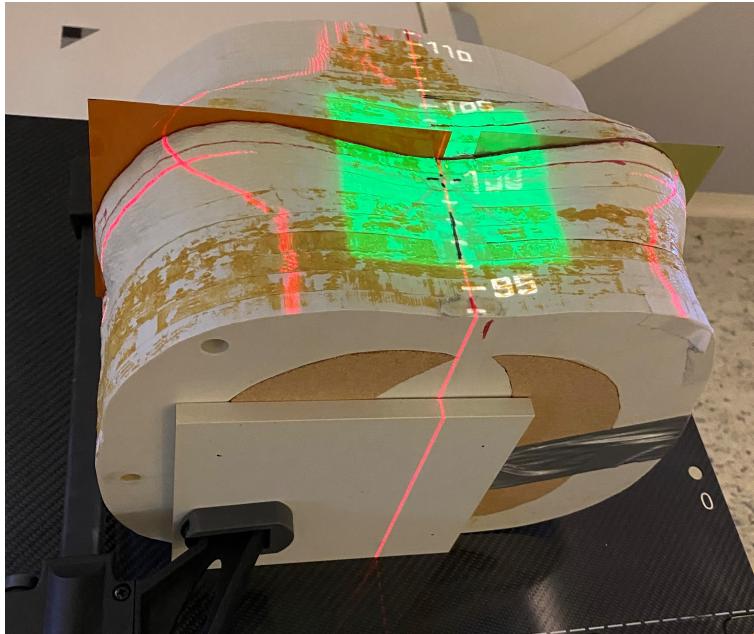


Figure 3.9: An anthropomorphic female thorax phantom is shown. The phantom is made of 18 transversal slices of RW3 with a density of $1.045\text{g}/\text{cm}^3$. Each slice is 10 mm thick, and the lungs were represented by a material with density $0.28\text{ g}/\text{cm}^3$ [1]. GafChromic films, EBT3 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.

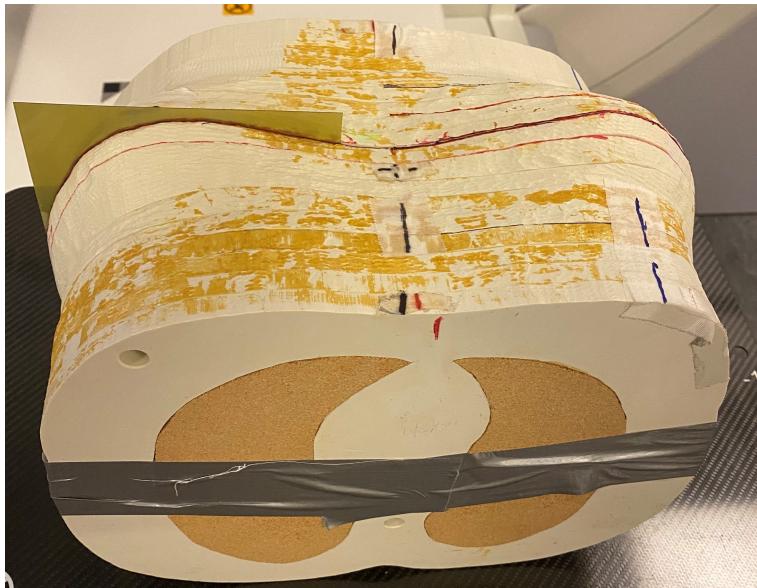


Figure 3.11: A transversal slice of the anthropomorphic female thorax phantom is shown. Here the GafChromic EBT3 film is positioned so that it covers the contralateral (opposite, here right) breast.

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.

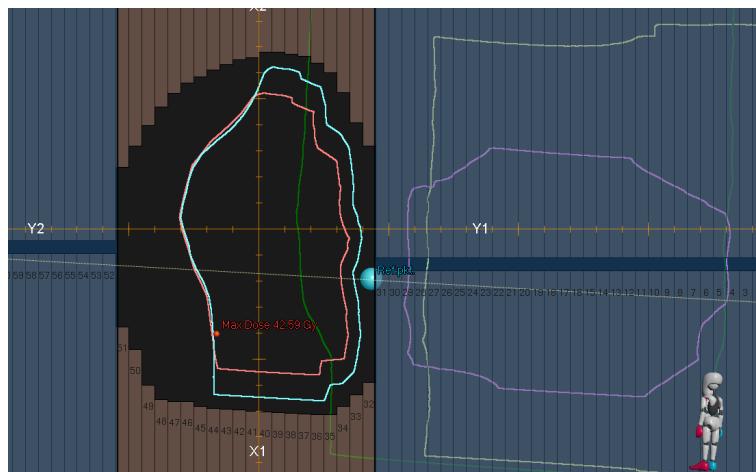


Figure 3.12: Beam's eye view of a lateral treatment field. This is a robust arrangement of an open tangential field extending into air at the superficial regions of the breast.

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.

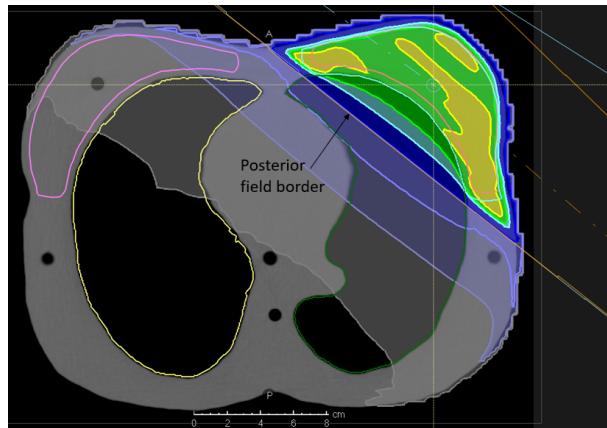
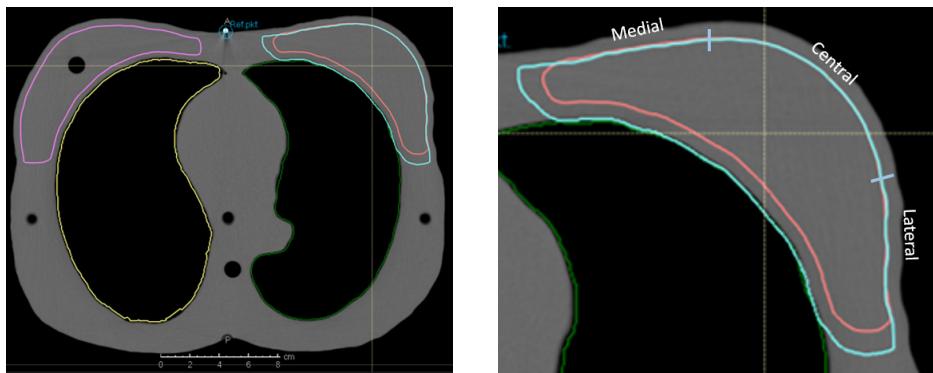


Figure 3.13: A standard tangential plan setup, consisting of medial (blue) and lateral (orange) fields with aligned posterior field borders.

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.



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.

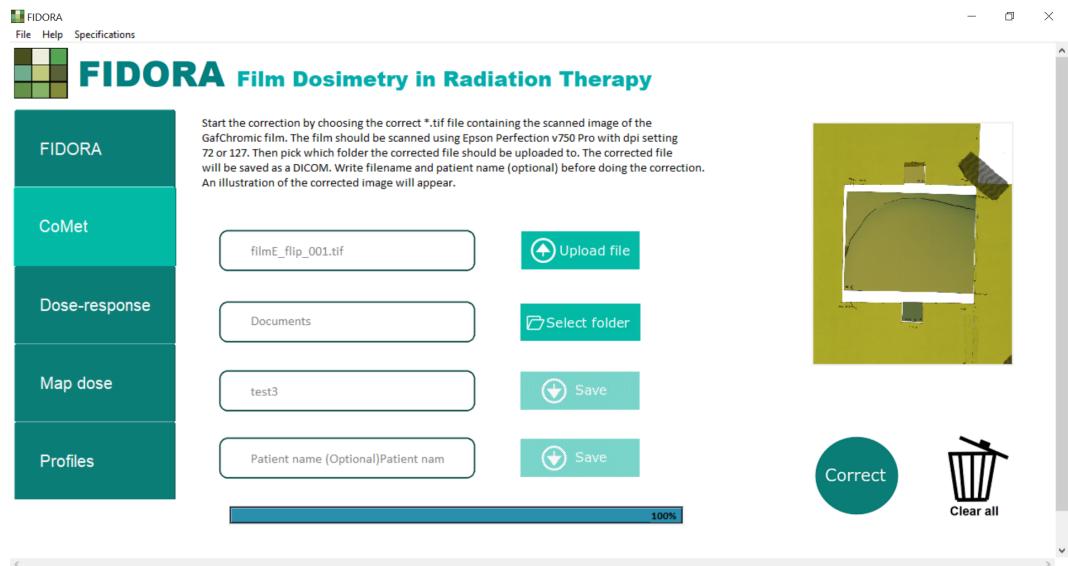


Figure 4.2: A print screen of the CoMet tab in FIDORA. The CoMet tab enables a user to upload an image of a scanned film, and performs a correction on the image.

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.

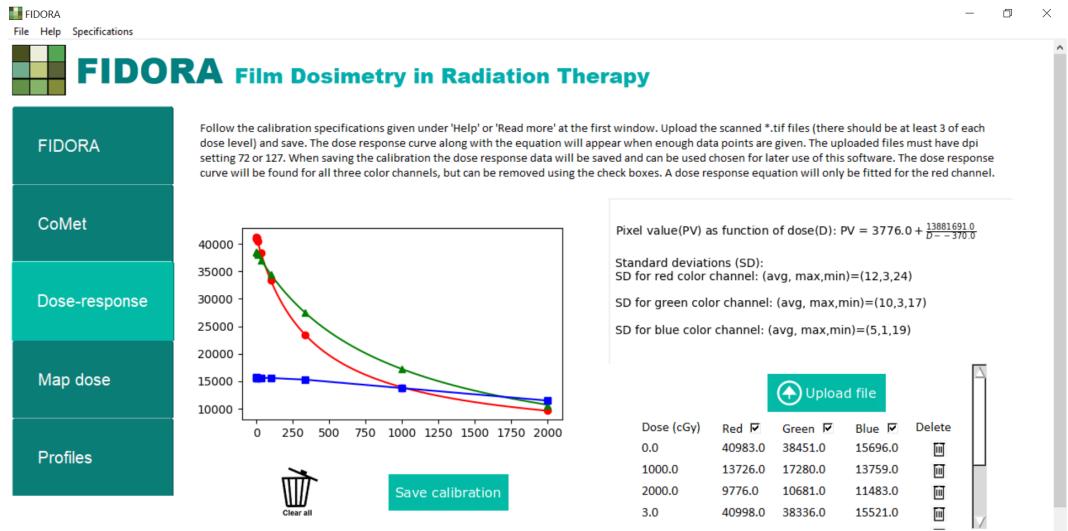


Figure 4.3: A print screen of the dose-response tab in FIDORA. The dose-response tab enables a user to upload known calibration films with known reference doses, scanned in the center of the scanner, and makes a calibration curve based on the formula, $d_x(D) = a + b/(D-c)$. $d_x(D)$ is the optical density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

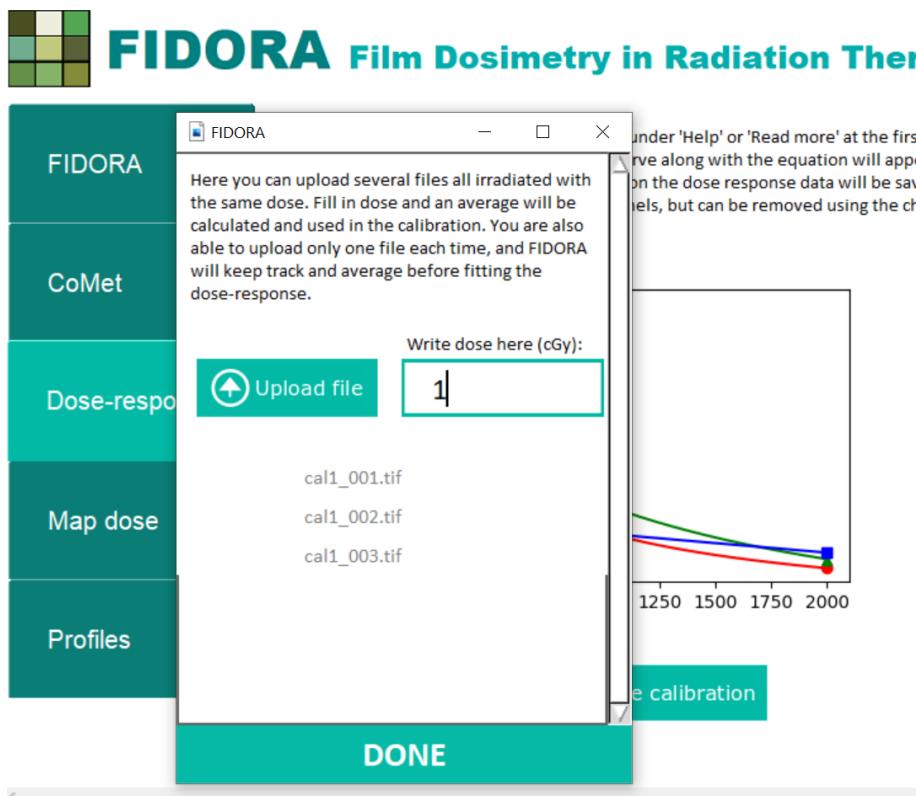


Figure 4.4: A print screen of a tab within the dose-response tab in FIDORA, that enables the user to upload reference doses. The user indicates the reference dose, in units of cGy, and has the option to upload one or more scanned images with the given reference dose. If more than one image is chosen 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.

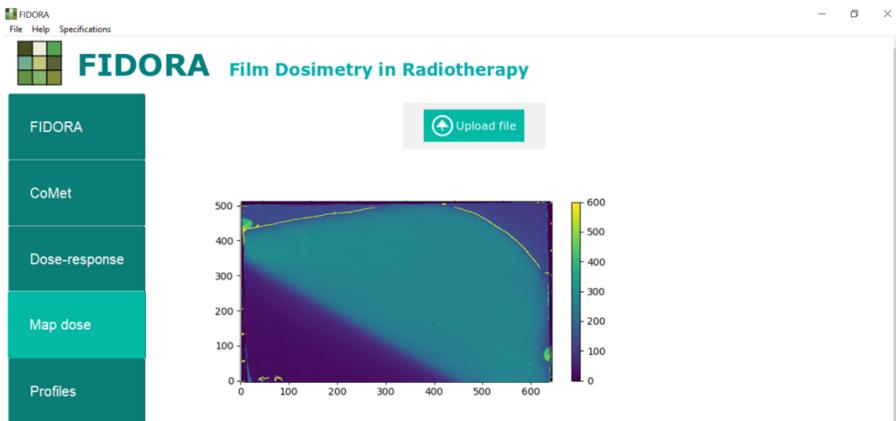


Figure 4.5: A print screen of the Map dose tab in FIDORA. The Map dose tab enables the user to upload an image of a scanned film and will map it to doses, using an available calibration curve made in the tab Dose-response.

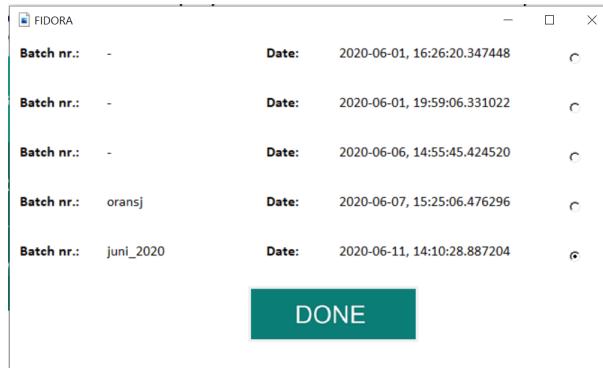


Figure 4.6: A print screen of the pop-up window that enables the user to choose one of the calibration curves made in the Dose-response tab in FIDORA.

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.

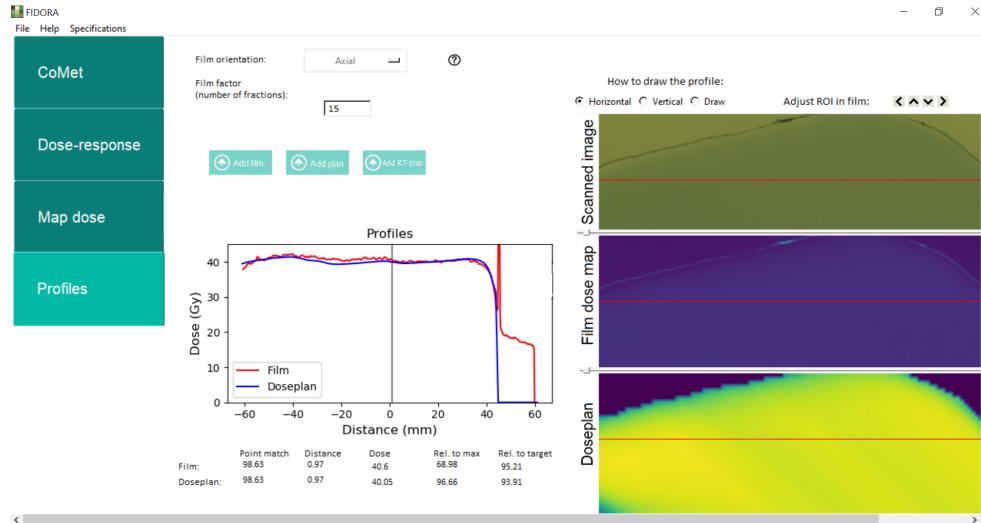
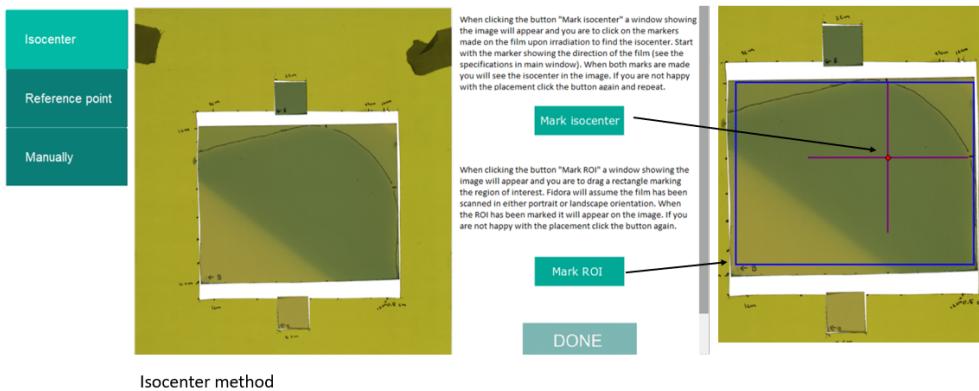
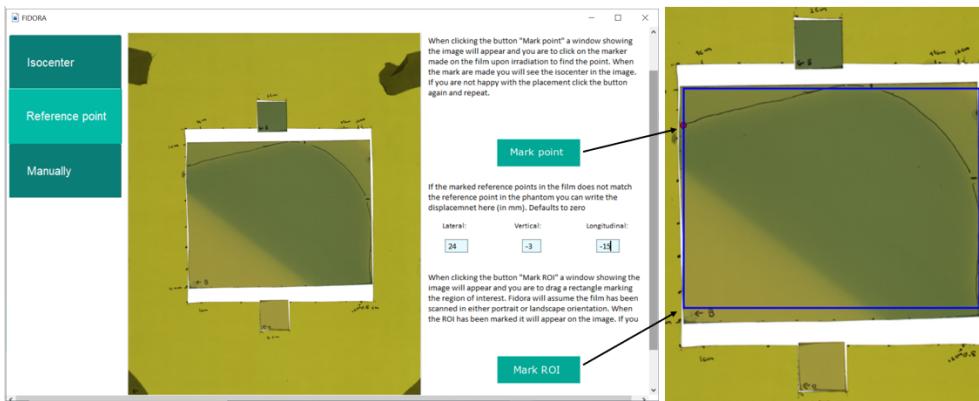


Figure 4.7: A print screen of the Profiles tab in FIDORA. The Profiles tab enables the user to upload a scanned film, and compare it to the corresponding dose plan. The user can chose between horizontal, vertical and user-defined profiles. The profiles are drawn in the dose map of the film, and is mapped 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.



Isocenter method

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

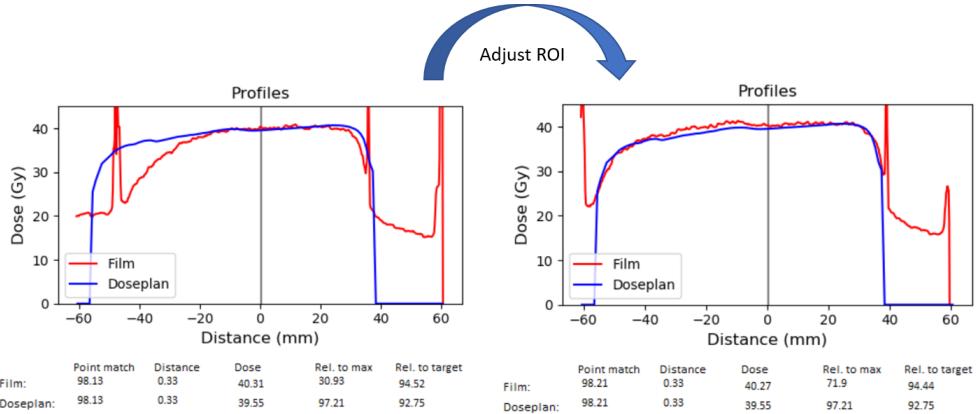


Figure 4.10: A print screen of the plot of the profiles along a user-defined line, with values of interest displayed, in the Profiles tab in FIDORA. When hoovering over the plot, the table below the plot will list values that corresponds to the x-value that is hoovered over. The grey, vertical line, 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 in Figure 4.13.

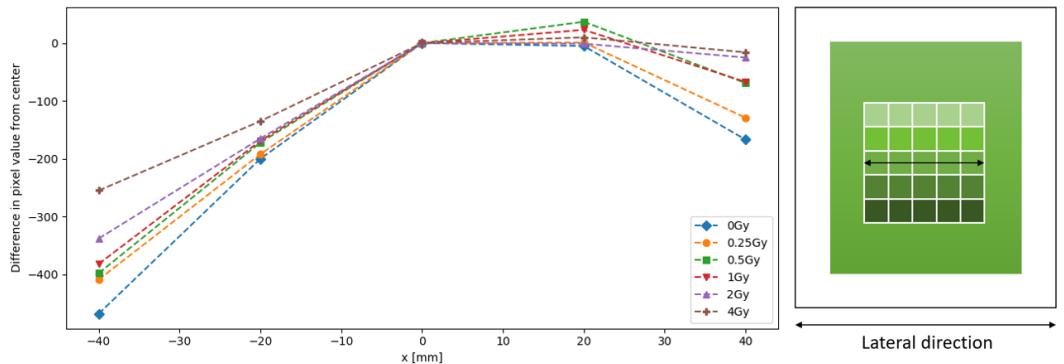


Figure 4.11: Profiles of the deviation in pixel value compared to the center across the scanner surface in lateral direction for different dose levels.

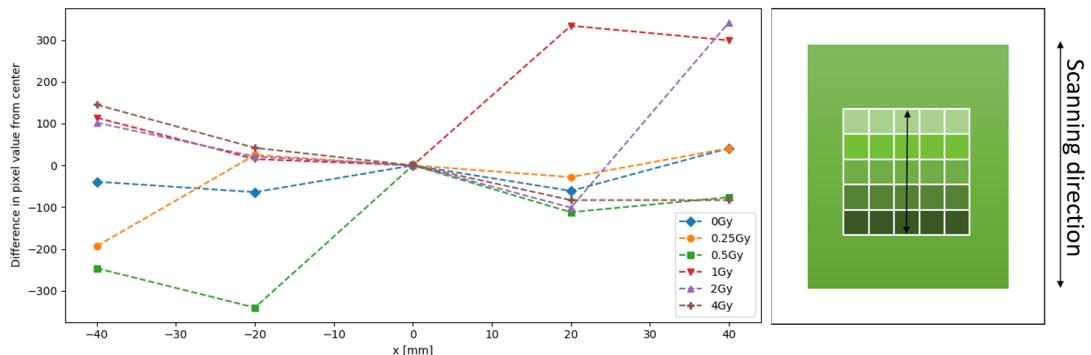


Figure 4.12: Profiles of the deviation in pixel value compared to the center across the scanner surface in scanning direction for different dose levels.

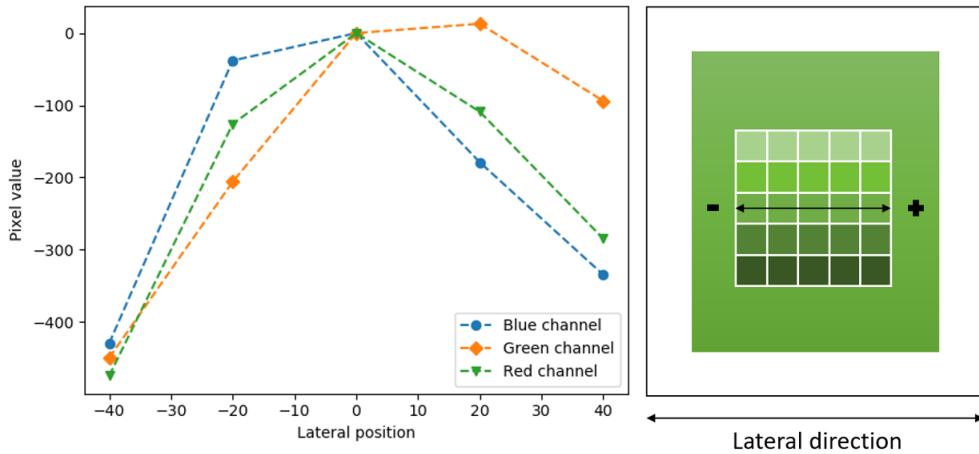


Figure 4.13: A lateral profile, which is the average of all doses obtained by the method described in Section 3.1.3, is shown for all color channels. This profile is shown to illustrate the resulting correction matrix that will be used to correct the investigated non-uniform read-out over the scanning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

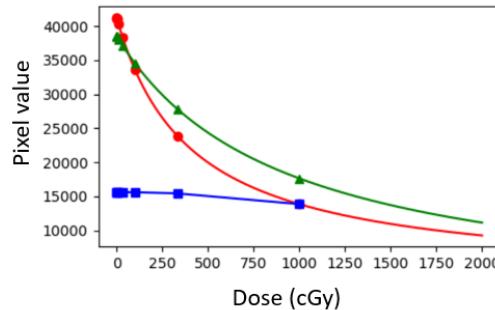


Figure 4.14: Calibration curve (from first day of experiments) obtained from eight reference doses at 0, 1, 3, 10, 33, 100, 333 and 1000 cGy using a filtered radiation beam, and GafChromic EBT3 film. The red, green and blue fitted lines indicates the red, green and blue color channels, respectively.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

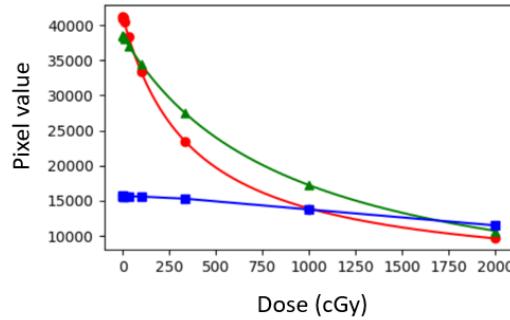


Figure 4.15: Calibration curve (from second day of experiments) obtained from nine reference doses at 0, 1, 3, 10, 33, 100, 333, 1000 and 2000 cGy using a filtered radiation beam, and GafChromic EBT3 film. The red, green and blue fitted lines indicates the red, green and blue color channels, respectively. 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:

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

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

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

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:

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

for the first day of experiments, and

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

for the second day of experiments.

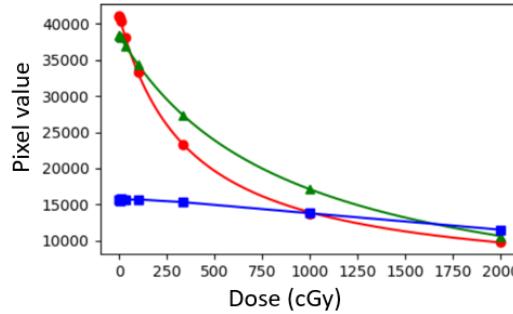


Figure 4.16: Calibration curve obtained from nine reference doses at 0, 1, 3, 10, 33, 100, 333, 1000 and 2000 cGy, each scanned three times, using a filter-free radiation beam, and GafChromic EBT3 film. The red, green and blue fitted lines indicates the red, green and blue color channels, respectively. 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 $PV = 4037 + \frac{13482787}{D - (-363)}$.

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

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

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:

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

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:

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

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:

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

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.

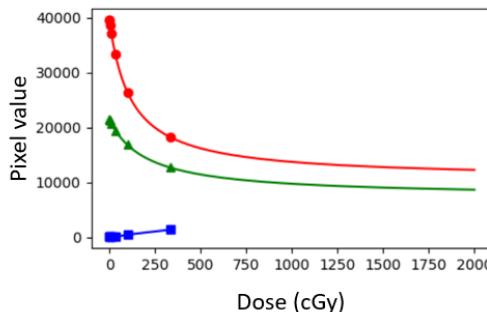


Figure 4.17: Calibration curve obtained from nine reference doses at 0, 1, 3, 10, 33, 100 and 333 cGy using a filtered radiation beam and GafChromic XR-QA2 film. The red, green and blue fitted lines indicates the red, green and blue color channels, respectively. The horizontal axis holds the doses in 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:

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

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:

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

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 fur-

ther 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.

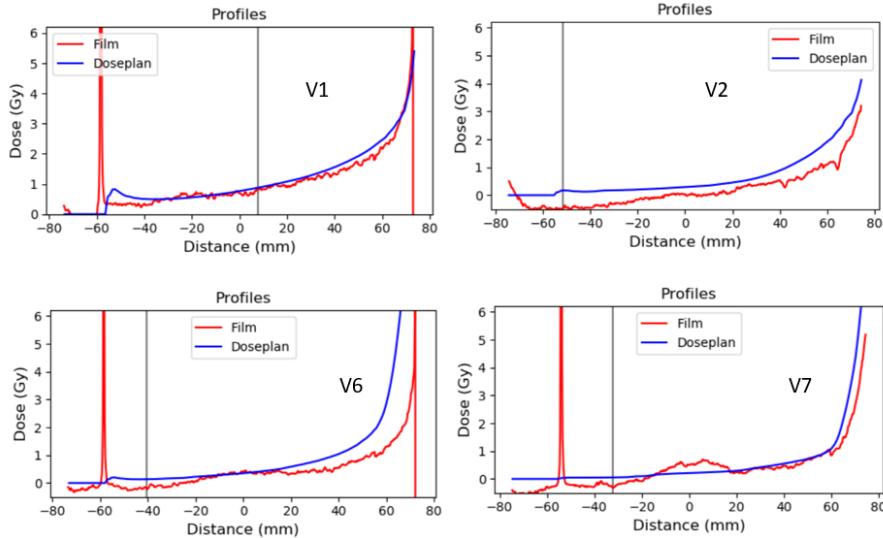


Figure 4.18: Horizontal profiles employing the calibration curve for the GafChromic XR-QA2 film of treatment plans, V1, V2, V6 and V7, as described in Table 3.1. Using GafChromic XR-QA2 film with its associated calibration curve, there is a significant underestimation of dose observed at certain areas. 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 .

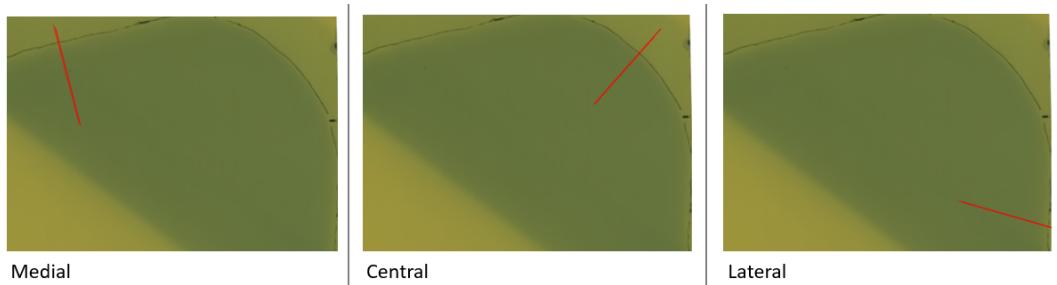


Figure 4.19: Profiles taken at medial, central and lateral incidence on the target breast in the isocentre plane, shown as a red line in the corresponding images.

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.

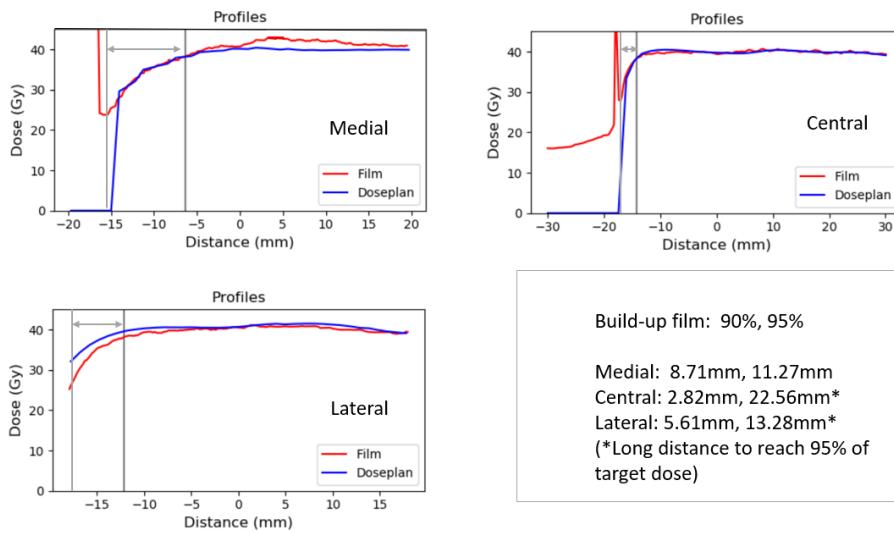


Figure 4.20: Profiles of V1 (tangential FiF) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface.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

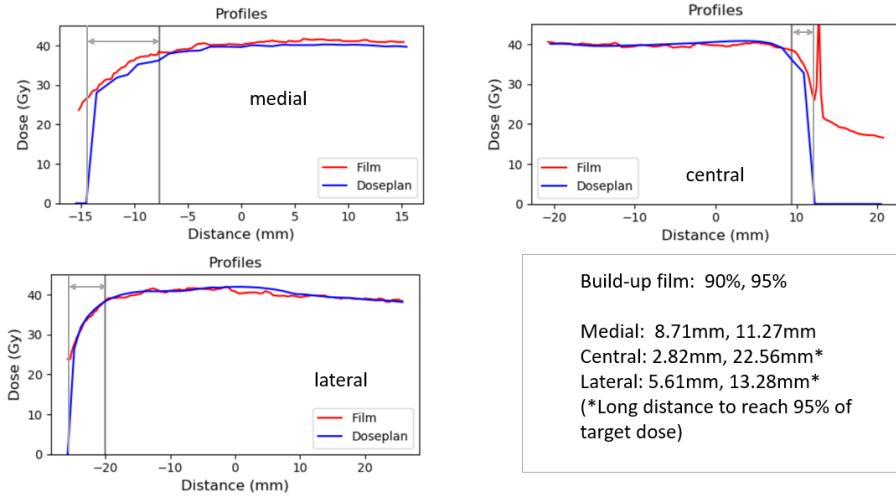


Figure 4.21: Profiles of V2 (tangential FiF 90col) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface.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

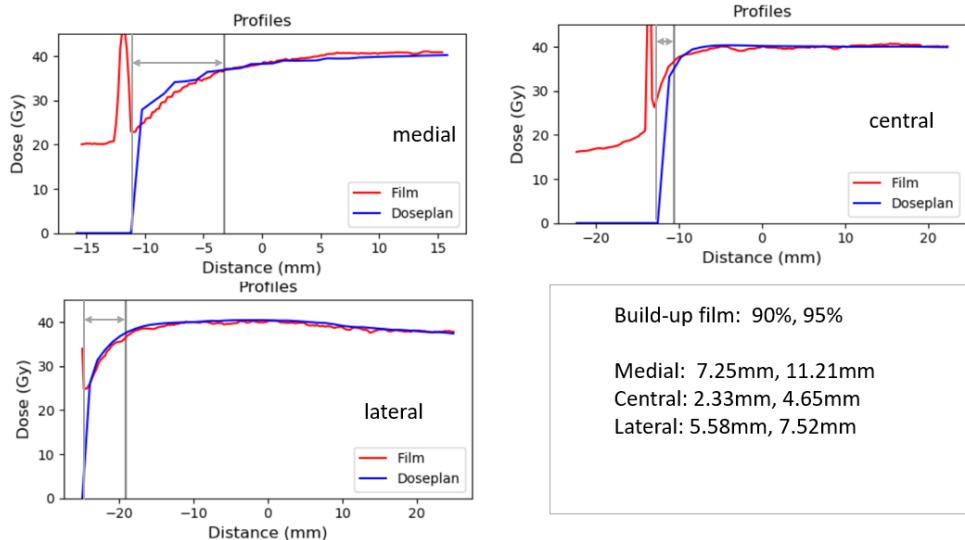


Figure 4.22: Profiles of V3 (hybrid VMAT) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The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

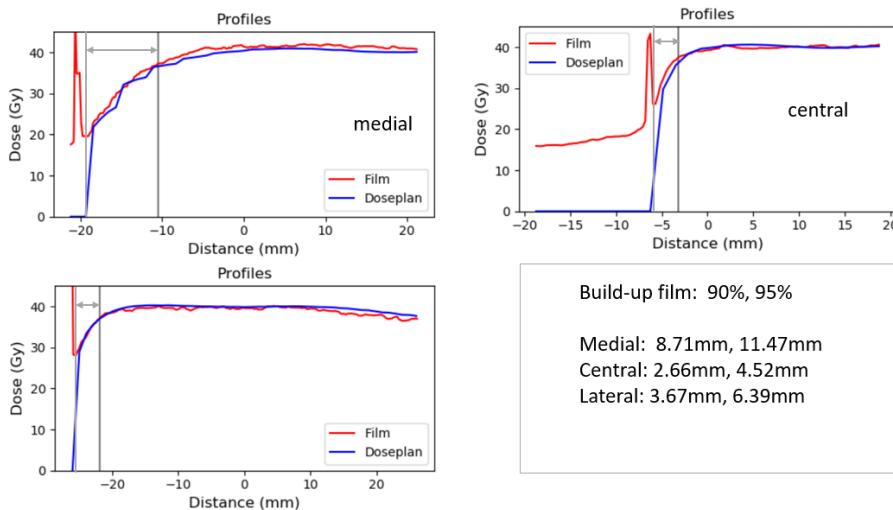


Figure 4.23: Profiles of V5 (VMAT short arcs 0col) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface.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

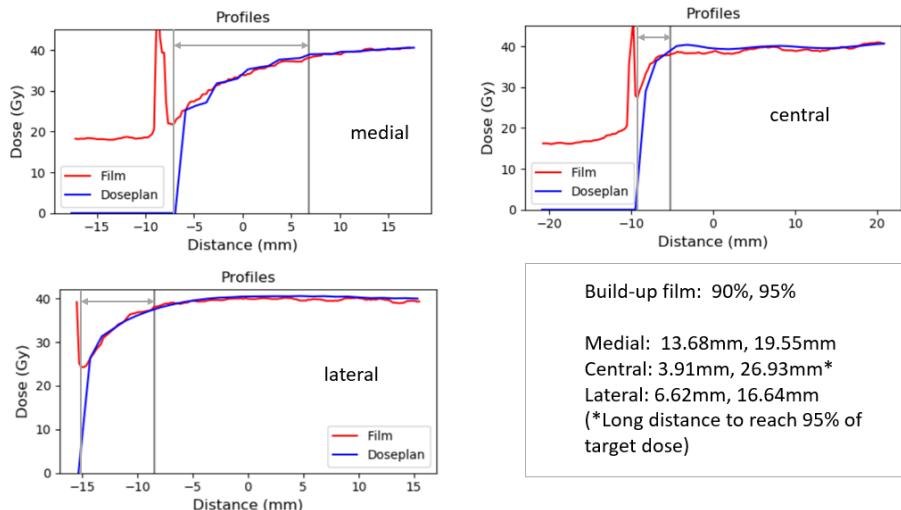


Figure 4.24: Profiles of V6 (VMAT short arcs 90col) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface.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

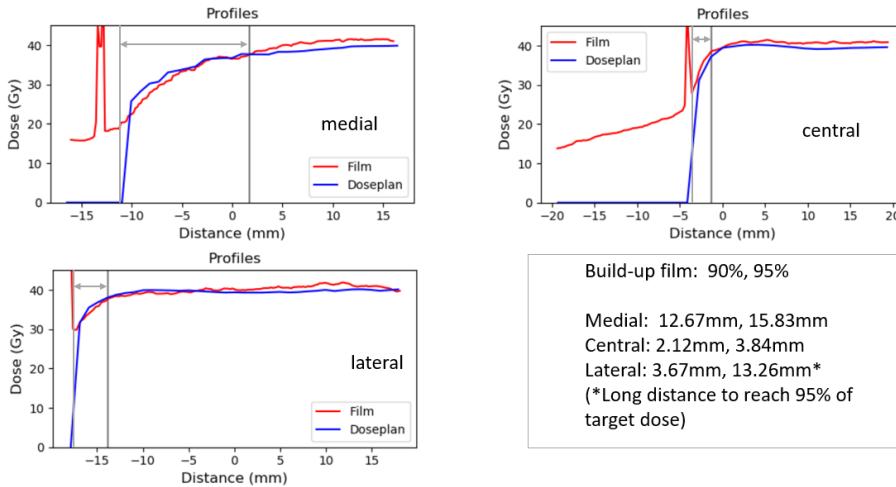


Figure 4.25: Profiles of V7 (VMAT short arcs 90col) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface.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

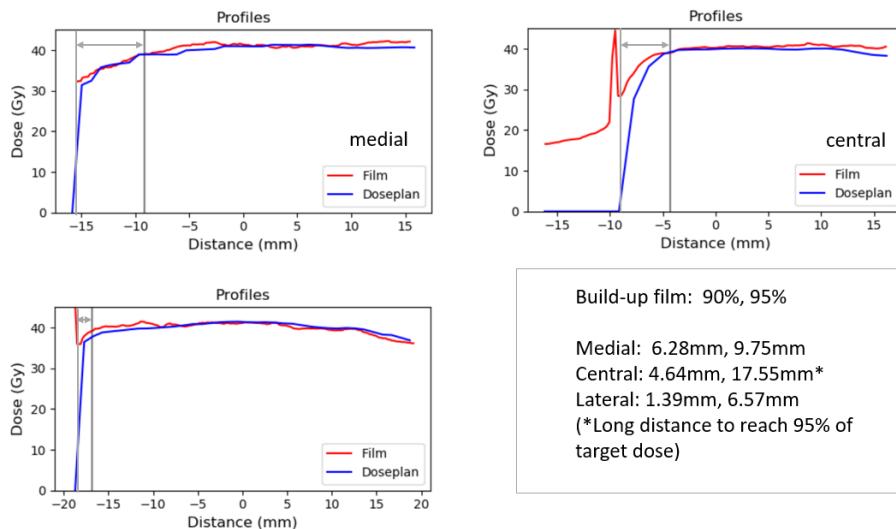


Figure 4.26: Profiles of V8 (medial FFF) at medial, central and lateral incidence to the breast. The build-up distance for the film was quantified by calculating the distance it takes for the depth dose to reach 90% and 95% of the target dose, starting at the entrance dose at the breast surface. The 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%		
Plan name	Medial	Central	Lateral	Medial	Central	Lateral
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.

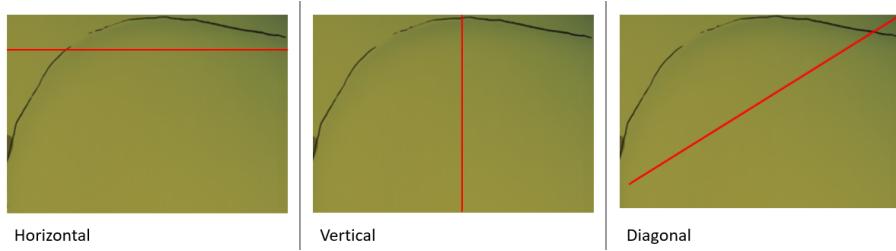


Figure 4.27: Horizontal, vertical and diagonal profiles drawn through the CLB in the isocentre plane, shown as a red line in the corresponding images.

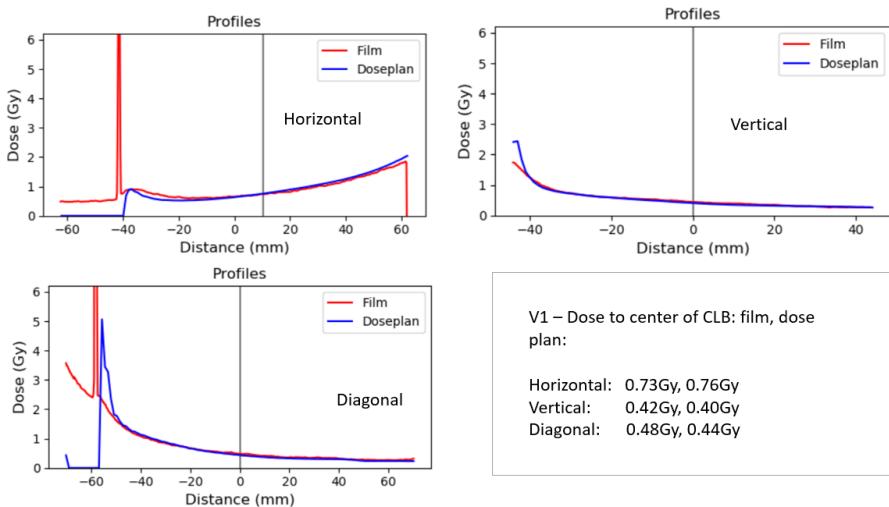


Figure 4.28: Horizontal, vertical and diagonal profiles of V1 (tangential FiF). The grey vertical line indicates the position along the profile that was evaluated.

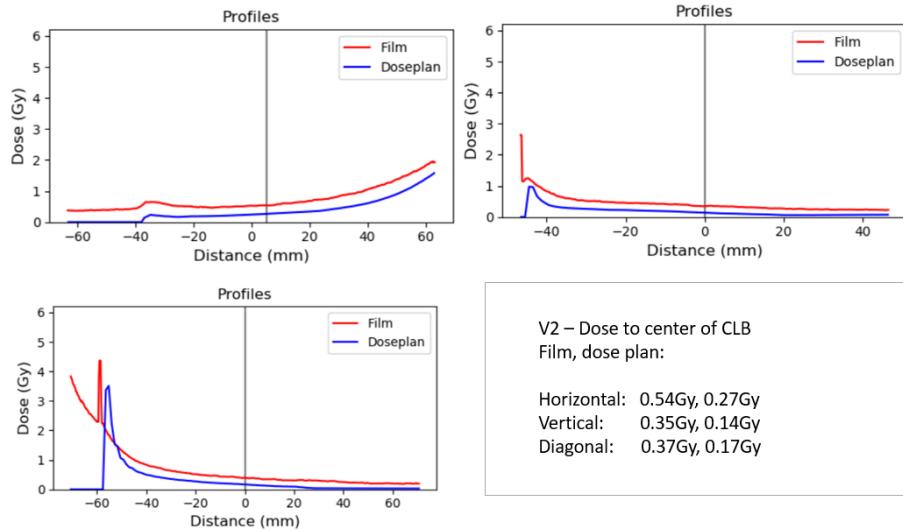


Figure 4.29: Horizontal, vertical and diagonal profiles of V2 (tangential FiF 90col). The grey vertical line indicates the position along the profile that was evaluated.

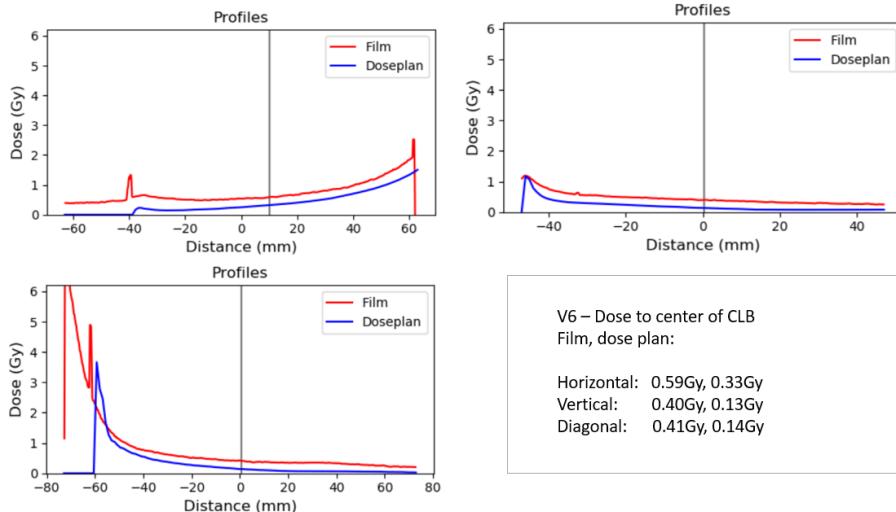


Figure 4.30: Horizontal, vertical and diagonal profiles of V6 (VMAT short arcs 90col). The grey vertical line indicates the position along the profile that was evaluated.

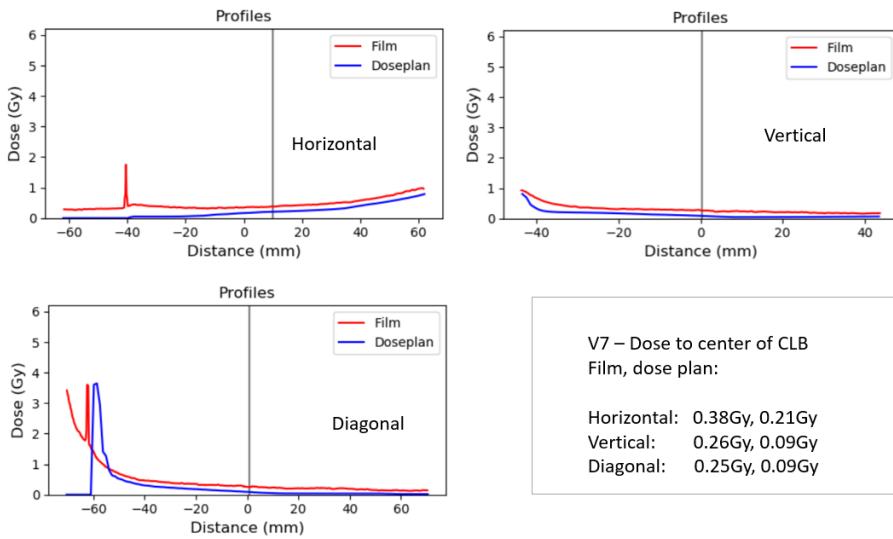


Figure 4.31: Horizontal, vertical and diagonal profiles of V9 (VMAT FFF short arcs 90col). The grey vertical line indicates the position along the profile that was evaluated.

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 ($\pm 1\text{-}3 \text{ cGy}$ 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 25×25 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 measure-

ments 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.

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™ 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™ DOSIMETRY MEDIA, TYPE EBT-3. Retrieved from: http://www.gafchromic.com/documents/EBT3_Specifications.pdf
- [6] Ashland. XR-QA2 specification sheet. GAFCHROMIC™ 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

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

```
1000 import tkinter as tk
1001 from tkinter import ttk, INSERT, DISABLED, GROOVE, CURRENT, Radiobutton, \
1002     NORMAL, ACTIVE, messagebox, Menu, IntVar, Checkbutton, FLAT,
1003     PhotoImage, Label, \
1004     SOLID, N, S, W, E, END, LEFT, Scrollbar, RIGHT, Y, BOTH, TOP,
1005     OptionMenu, SUNKEN, \
1006     RIDGE, BOTTOM, X
1007
1008 import Globals
1009 import re
1010 import CoMet_functions, intro_tab_functions, Map_Dose
1011 import Dose_response_functions, Profile_functions, DVH_functions
1012 from PIL import Image, ImageTk
1013 import os
1014 import sys
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
```

```

1050     }
1051 },
1052 "TNotebook": {
1053     "configure": {
1054         "background": "#ffffff", # color behind the notebook
1055         "tabmargins": [5, 5, 10, 10], # [left margin, upper margin,
1056         #right margin,margin beetwen tab and frames]
1057         "tabposition": 'wn',
1058         "borderwidth": 0,
1059     }
1060 },
1061 "TNotebook.Tab": {
1062     "configure": {
1063         "background": '#0A7D76', # Color of non selected tab-button
1064         "foreground": '#ffffff',
1065         "padding": [30,35, 20,35], # [space beetwen text and
1066         horizontal tab-button
1067             #border, space between text and vertical tab_button border]
1068         "font": ('#FFFFFF', '15'),
1069         "borderwidth": 1,
1070         "equalTabs": True,
1071         "width": 13
1072     },
1073     "map": {
1074         "background": [("selected", '#02B9A5')], # Color of active tab
1075         "expand": [("selected", [1, 1, 1, 0])] # [expanse of text]
1076     }
1077 },
1078 "Treeview": {
1079     "configure": {
1080         "font": ('calibri', '9'),
1081         "highlightthickness": 0,
1082         "relief": FLAT,
1083         "borderwidth": 0
1084     }
1085 },
1086 "Treeview.Heading": {
1087     "configure": {
1088         "font": ('calibri', '9'),
1089         "highlightthickness": 0,
1090         "relief": FLAT,
1091         "borderwidth": 0,
1092         "anchor": W
1093     }
1094 },
1095 },
1096 })
1097 }
1098 })
1099 style.theme_use('MyStyle')
1100
1101 menubar = Menu(Globals.form)
1102 filemenu = Menu(menubar, tearoff=0)

```

```

1106 filmenu.add_command(label="Restart", command=CoMet_functions.
    nothingButton)
1107 filmenu.add_command(label="Open", command=CoMet_functions.nothingButton)
1108 filmenu.add_separator()
1109 filmenu.add_command(label="Exit", command=Globals.form.quit)
1110 menubar.add_cascade(label="File", menu=filmenu)
1111 helpmenu = Menu(menubar, tearoff=0)
1112 helpmenu.add_command(label="Help", command=CoMet_functions.nothingButton)
1113 helpmenu.add_command(label="About", command=CoMet_functions.nothingButton)
1114 menubar.add_cascade(label="Help", menu=helpmenu)

1116 scannermenu=Menu(menubar, tearoff=0)
1117 scannermenu.add_command(label="Scanner settings", \
    command=intro_tab_functions.createScannerSettingsWindow)
1118 scannermenu.add_command(label="Calibration", \
    command=intro_tab_functions.createCalibrationWindow)
1119 scannermenu.add_command(label="Raystation", \
    command=intro_tab_functions.createRaystationWindow)
1120 menubar.add_cascade(label="Specifications", menu=scannermenu)

1124 Globals.form.config(menu=menubar)
1125
1126 upload_button_file = "uploadbutton3.png"
1127 Globals.upload_button_image = ImageTk.PhotoImage(file=upload_button_file)

1130 select_folder_button_file = "select_folder_button2.png"
1131 select_folder_image = ImageTk.PhotoImage(file=select_folder_button_file)
1132
1133 help_button_file = "help_button.png"
1134 Globals.help_button = ImageTk.PhotoImage(file=help_button_file)

1136 done_button_file = "done_button.png"
1137 Globals.done_button_image = ImageTk.PhotoImage(file=done_button_file)
1138
1139 CoMet_border_dark_file = "border.png"
1140 CoMet_border_dark = ImageTk.PhotoImage(file=CoMet_border_dark_file)

1142 CoMet_border_light_file = "border_light.png"
1143 CoMet_border_light = ImageTk.PhotoImage(file=CoMet_border_light_file)
1144
1145 CoMet_save_button_file = "save_button2.png"
1146 CoMet_save_button = ImageTk.PhotoImage(file=CoMet_save_button_file)
1147 Globals.save_button = ImageTk.PhotoImage(file=CoMet_save_button_file)

1148 CoMet_correct_button_file = "icon_correct.png"
1149 CoMet_correct_button.image= ImageTk.PhotoImage(file=
    CoMet_correct_button_file)

1152 CoMet_clear_all_button_file = "icon_clear_all.png"
1153 CoMet_clear_all_button.image = ImageTk.PhotoImage(file=
    CoMet_clear_all_button_file)
1154
1155 dose_response_clear_all_button_file = "icon_clear_all_small.png"
1156 dose_response_clear_all_button.image = \
    ImageTk.PhotoImage(file=dose_response_clear_all_button_file)
1157
1158 CoMet_empty_image_file = "empty_corrected_image.png"

```

```

1160 CoMet.empty_image_image = \
1161     ImageTk.PhotoImage( file=CoMet.empty_image_file )
1162 dose_response_calibration_button_file = "save_calibration_button.png"
1163 dose_response_calibration_button.image = \
1164     ImageTk.PhotoImage( file=dose_response_calibration_button_file )
1165 dose_response_dose_border_file = "dose_border.png"
1166 Globals.dose_response_dose_border = \
1167     ImageTk.PhotoImage( file=dose_response_dose_border_file )
1168 profiles_add_doseplan_button_file = "add_doseplan_button.png"
1169 Globals.profiles_add_doseplan_button.image = \
1170     ImageTk.PhotoImage( file=profiles_add_doseplan_button_file )
1171 profiles_add_film_button_file = "add_film_button.png"
1172 Globals.profiles_add_film_button.image = \
1173     ImageTk.PhotoImage( file=profiles_add_film_button_file )
1174 profiles_add_rtplan_button_file = "add_rtplan_button.png"
1175 Globals.profiles_add_rtplan_button.image = \
1176     ImageTk.PhotoImage( file=profiles_add_rtplan_button_file )
1177 profiles_showPlanes_file = "planes.png"
1178 Globals.profiles_showPlanes.image = \
1179     ImageTk.PhotoImage( file=profiles_showPlanes_file )
1180 profiles_showDirections_file = 'depth_directions.png'
1181 Globals.profiles_showDirections.image = \
1182     ImageTk.PhotoImage( file=profiles_showDirections_file )
1183 profiles_mark_isocenter_button_file = 'mark_isocenter_button.png'
1184 Globals.profiles_mark_isocenter_button.image = \
1185     ImageTk.PhotoImage( file=profiles_mark_isocenter_button_file )
1186 profiles_mark_ROI_button_file = "mark_ROI_button.png"
1187 Globals.profiles_mark_ROI.button.image = \
1188     ImageTk.PhotoImage( file=profiles_mark_ROI_button_file )
1189 profiles_scanned_image_text_image_file = "scanned_image_text_image.png"
1190 Globals.profiles_scanned_image_text_image.image = \
1191     ImageTk.PhotoImage( file=profiles_scanned_image_text_image_file )
1192 profiles_film_dose_map_text_image_file = "film_dose_map_text_image.png"
1193 Globals.profiles_film_dose_map_text_image.image = \
1194     ImageTk.PhotoImage( file=profiles_film_dose_map_text_image_file )
1195 profiles_doseplan_text_image_file = "doseplan_text_image.png"
1196 Globals.profiles_doseplan_text.image = \
1197     ImageTk.PhotoImage( file=profiles_doseplan_text_image_file )
1198 profiles_mark_point_file = "mark_point_button.png"
1199 Globals.profiles_mark_point.button.image = \
1200     ImageTk.PhotoImage( file=profiles_mark_point_file )
1201 profiles_add_doseplans_button_file = "add_doseplan.png"
1202 Globals.profiles_add_doseplans_button.image = \

```

```

1218     ImageTk.PhotoImage(file=profiles.add_doseplans.button_file)
1219
1220 adjust_button_left_file = "adjust_button_left.png"
1221 Globals.adjust_button_left_image = ImageTk.PhotoImage(file=
1222     adjust_button_left_file)
1223
1224 adjust_button_right_file = "adjust_button_right.png"
1225 Globals.adjust_button_right_image = ImageTk.PhotoImage(file=
1226     adjust_button_right_file)
1227
1228 adjust_button_down_file = "adjust_button_down.png"
1229 Globals.adjust_button_down_image = ImageTk.PhotoImage(file=
1230     adjust_button_down_file)
1231
1232 adjust_button_up_file = "adjust_button_up.png"
1233 Globals.adjust_button_up_image = ImageTk.PhotoImage(file=
1234     adjust_button_up_file)
1235 ##### INTRO TAB
1236 #####
1237
1238 #scrollbar = Scrollbar(Globals.intro_tab)
1239 #scrollbar.pack(side=RIGHT, fill=Y)#grid(row=0, column=1, sticky=N+S+E)#
1240     pack(side=RIGHT, fill=Y)
#Globals.intro_tab.grid_columnconfigure(0, weight=0)
#Globals.intro_tab.grid_rowconfigure(0, weight=0)
1241 intro_tab_canvas = tk.Canvas(Globals.intro_tab)#, yscrollcommand=scrollbar
1242     .set)
1243 intro_tab_canvas.config(bg='#ffffff', bd = 0, relief=FLAT,
1244     highlightthickness=0)
1245
1246 tab1_text_box = tk.Frame(intro_tab_canvas, height=230, width=400)
1247 tab1_text_box.grid(row=0, column=0, pady=(30,30), padx=(55,0))
1248 tab1_text_box.config(bd=0, bg='#E5f9ff')
1249
1250 tab1_title_text = tk.Text(tab1_text_box, height=1, width=6)
1251 tab1_title_text.insert(END, "CoMet")
1252 tab1_title_text.grid(in_=tab1_text_box, row=0, column = 0, pady=(15,5),
1253     padx=(10,10))
1254 tab1_title_text.config(state=DISABLED, bd=0, bg ='#E5f9ff', fg='#130e07',
1255     font=('calibri', '25', 'bold'))
1256 tab1_text_box.grid_columnconfigure(0, weight=1)
1257 tab1_text_box.grid_rowconfigure(0, weight=1)
1258
1259 tab1_text = tk.Text(tab1_text_box, height=4, width=43)
1260 tab1_text.grid(in_=tab1_text_box, row=1, column=0, sticky=N+S+W+E, pady
1261     =(0,0), padx=(20,20))
1262 tab1_text.insert(INSERT,"Correct your scanned images using CoMet. A method
1263     \ndeveloped to correct for non-uniformity introduced\n\
1264 by the scanner. The correction is based on absolute \nsubtraction.")
1265 tab1_text.config(state=DISABLED, bd=0, bg ='#E5f9ff', fg='#130E07', font=(
1266     'calibri', '13'))
1267 tab1_text_box.grid_columnconfigure(1, weight=1)
1268 tab1_text_box.grid_rowconfigure(1, weight=1)
1269
```

```

1262 tab1_readmore_text = tk.Text(tab1_text_box, height=1, width=1)
1263 tab1_readmore_text.grid(row=1, column=0, sticky = N+S+W+E, pady=(65,0),
1264     padx = (110,0))
1265 tab1_readmore_text.insert(INSERT,"Read more ...")
1266 tab1_readmore_text.config(state=DISABLED, bd=0, bg="#E5f9ff", fg="#130E07",
1267     , font=('calibri', '12', 'bold'))
1268 tab1_text_box.grid_columnconfigure(2, weight=1)
1269 tab1_text_box.grid_rowconfigure(2, weight=1)

1270 tab1_box_figure = Image.open("icon_comet.png")
1271 tab1_figure = ImageTk.PhotoImage(tab1_box_figure)
1272 tab1_figure_label = Label(tab1_text_box, image=tab1_figure)
1273 tab1_figure_label.image = tab1_figure
1274 tab1_figure_label.grid(row=3, sticky=N+S+W+E, pady=(0,10))
1275 tab1_figure_label.config(bg="#E5f9ff")
1276 tab1_text_box.grid_columnconfigure(3, weight=1)
1277 tab1_text_box.grid_rowconfigure(3, weight=1)
1278 """
1279 tab1_readmore = tk.Button(tab1_text_box, text='Read more', cursor='hand2',
1280     font=('calibri', '12', 'bold'),\
1281     relief=FLAT, state=tk.ACTIVE, width = 15, command=intro_tab_functions.
1282     readMore)
1283 tab1_readmore.place(relwidth=0.25, relheight=0.13, relx=0.27, rely=0.054)
1284 """
1285 tab2_text_box = tk.Frame(intro_tab_canvas, height=230, width=400)
1286 tab2_text_box.grid(row=0, column=1, pady=(30,30), padx=(65,0))
1287 tab2_text_box.config(bd=0, bg="#E5f9ff")

1288 tab2_title = tk.Text(tab2_text_box, height=1, width=12)
1289 tab2_title.grid(in_=tab2_text_box, row=0, column = 0, pady=(15,5), padx
1290     =(10,10))
1291 tab2_title.insert(INSERT, "Dose response")
1292 tab2_title.config(state=DISABLED, bd=0, bg = '#E5f9ff', fg="#130e07", font
1293     =('calibri', '25', 'bold'))
1294 tab2_text_box.grid_columnconfigure(0, weight=1)
1295 tab2_text_box.grid_rowconfigure(0, weight=1)
1296
1297 tab2_text = tk.Text(tab2_text_box, height=4, width=43)
1298 tab2_text.grid(in_=tab2_text_box, row=1, column=0, sticky=N+S+W+E, pady
1299     =(0,0), padx=(20,20))
1300 tab2_text.insert(INSERT,"Make a calibration curve and read the dose
1301     response \nfunction. For every new batch of GafChromic film\
1302     \nthere is a need to update the dose response. All three \nchannels (
1303     RGB) are read and calculated.")
1304 tab2_text.config(state=DISABLED, bd=0, bg="#E5f9ff", fg="#130E07", font=(
1305     'calibri', '13'))
1306 tab2_text_box.grid_columnconfigure(1, weight=1)
1307 tab2_text_box.grid_rowconfigure(1, weight=1)
1308
1309 tab2_readmore_text = tk.Text(tab2_text_box, height=1, width=1)
1310 tab2_readmore_text.grid(row=1, column=0, sticky = N+S+W+E, pady=(65,0),
1311     padx = (300,0))
1312 tab2_readmore_text.insert(INSERT,"Read more ...")
1313 tab2_readmore_text.config(state=DISABLED, bd=0, bg="#E5f9ff", fg="#130E07",
1314     , font=('calibri', '12', 'bold'))
1315 tab2_text_box.grid_columnconfigure(2, weight=1)

```

```

1306 tab2_text_box.grid_rowconfigure(2, weight=1)
1308 tab2_box_figure = Image.open("icon_dose_response.png")
1309 tab2_figure = ImageTk.PhotoImage(tab2_box_figure)
1310 tab2_figure_label = Label(tab2_text_box, image=tab2_figure)
1311 tab2_figure_label.image = tab2_figure
1312 tab2_figure_label.grid(row=3, sticky=N+S+W+E, pady=(0,10))
1313 tab2_figure_label.config(bg="#E5f9ff")
1314 tab2_text_box.grid_columnconfigure(3, weight=1)
1315 tab2_text_box.grid_rowconfigure(3, weight=1)
1316
1317 tab3_text_box = tk.Frame(intro_tab_canvas, height=230, width=400)
1318 tab3_text_box.grid(row=1, column=0, pady=(0,30), padx=(55,0))
1319 tab3_text_box.config(bd=0, bg="#E5f9ff")
1320
1321 tab3_title = tk.Text(tab3_text_box, height=1, width=8)
1322 tab3_title.grid(in_=tab3_text_box, row=0, column = 0, pady=(15,5), padx =(10,10))
1323 tab3_title.insert(INSERT, "Map dose")
1324 tab3_title.config(state=DISABLED, bd=0, bg = "#E5f9ff", fg="#130e07", font =('calibri', '25', 'bold'))
1325 tab3_text_box.grid_columnconfigure(0, weight=1)
1326 tab3_text_box.grid_rowconfigure(0, weight=1)
1327
1328 tab3_text = tk.Text(tab3_text_box, height=4, width=43)
1329 tab3_text.grid(in_=tab3_text_box, row=1, column=0, sticky=N+S+W+E, pady =(0,0), padx=(20,20))
1330 tab3_text.insert(INSERT,"Compare dose distribution in your treatment plan
    \nwith the measures distribution by the Gafchromic \nfilm.\n
    Using the gamma evaluation index a map of \npass/fail and variations is
    visualised.")
1331 tab3_text.config(state=DISABLED, bd=0, bg="#E5f9ff", fg="#130e07", font=('calibri', '13'))
1332 tab3_text_box.grid_columnconfigure(1, weight=1)
1333 tab3_text_box.grid_rowconfigure(1, weight=1)
1334
1335 tab3_readmore_text = tk.Text(tab3_text_box, height=1, width=1)
1336 tab3_readmore_text.grid(row=1, column=0, sticky = N+S+W+E, pady=(65,0),
    padx = (285,0))
1337 tab3_readmore_text.insert(INSERT,"Read more ...")
1338 tab3_readmore_text.config(state=DISABLED, bd=0, bg="#E5f9ff", fg="#130e07",
    font=('calibri', '12', 'bold'))
1339 tab3_text_box.grid_columnconfigure(2, weight=1)
1340 tab3_text_box.grid_rowconfigure(2, weight=1)
1341
1342 tab3_box_figure = Image.open("icon_map_dose.png")
1343 tab3_figure = ImageTk.PhotoImage(tab3_box_figure)
1344 tab3_figure_label = Label(tab3_text_box, image=tab3_figure)
1345 tab3_figure_label.image = tab3_figure
1346 tab3_figure_label.grid(row=3, sticky=N+S+W+E, pady=(0,10))
1347 tab3_figure_label.config(bg="#E5f9ff")
1348 tab3_text_box.grid_columnconfigure(3, weight=1)
1349 tab3_text_box.grid_rowconfigure(3, weight=1)
1350
1351 tab4_text_box = tk.Frame(intro_tab_canvas, height=230, width=400)
1352 tab4_text_box.grid(row=1, column=1, pady=(0,30), padx=(65,0))
1353 tab4_text_box.config(bd=0, bg="#E5f9ff")

```

```

1356 tab4_title = tk.Text(tab4_text_box, height=1, width=7)
1357 tab4_title.grid(in_=tab4_text_box, row=0, column = 0, pady=(15,5), padx
1358 =(10,10))
1359 tab4_title.insert(INSERT, "Profiles")
1360 tab4_title.config(state=DISABLED, bd=0, bg = '#E5f9ff', fg="#130e07", font
1361 =('calibri', '25', 'bold'))
1362 tab4_text_box.grid_columnconfigure(0, weight=1)
1363 tab4_text_box.grid_rowconfigure(0, weight=1)
1364
1365 tab4_text = tk.Text(tab4_text_box, height=4, width=43)
1366 tab4_text.grid(in_=tab4_text_box, row=1, column=0, sticky=N+S+W+E, pady
1367 =(0,0), padx=(20,20))
1368 tab4_text.insert(INSERT,"Investigate the profiles measured using
1369 GafChromic \nfilm and compare with the profiles in your treatment \
1370 nplan.\n")
1371 Using gamma evaluation an acceptance tube \ncan be places over the
1372 profile .")
1373 tab4_text.config(state=DISABLED, bd=0, bg='#E5f9ff', fg="#130E07", font=(
1374 'calibri', '13'))
1375 tab4_text_box.grid_columnconfigure(1, weight=1)
1376 tab4_text_box.grid_rowconfigure(1, weight=1)
1377
1378 tab4_readmore_text = tk.Text(tab4_text_box, height=1, width=1)
1379 tab4_readmore_text.grid(row=1, column=0, sticky = N+S+W+E, pady=(65,0),
1380 padx = (235,0))
1381 tab4_readmore_text.insert(INSERT,"Read more ...")
1382 tab4_readmore_text.config(state=DISABLED, bd=0, bg='#E5f9ff', fg="#130E07",
1383 , font=('calibri', '12', 'bold'))
1384 tab4_text_box.grid_columnconfigure(2, weight=1)
1385 tab4_text_box.grid_rowconfigure(2, weight=1)
1386
1387 tab4_box_figure = Image.open("icon_profiles.png")
1388 tab4_figure = ImageTk.PhotoImage(tab4_box_figure)
1389 tab4_figure_label = Label(tab4_text_box, image=tab4_figure)
1390 tab4_figure_label.image = tab4_figure
1391 tab4_figure_label.grid(row=3, sticky=N+S+W+E, pady=(0,10))
1392 tab4_figure_label.config(bg='#E5f9ff')
1393 tab4_text_box.grid_columnconfigure(3, weight=1)
1394 tab4_text_box.grid_rowconfigure(3, weight=1)
1395
1396 #intro_tab_canvas.configure(scrollregion = intro_tab_canvas.bbox("all"))
1397 intro_tab_canvas.grid(row=0, column=0, sticky=N+S+W)#pack(side=LEFT, fill=
1398 BOTH)
1399 #Globals.intro_tab.grid_columnconfigure(1, weight=2)
1400 #Globals.intro_tab.grid_rowconfigure(1, weight=2)
1401 #scrollbar.config(command=intro_tab_canvas.yview)
1402
1403 ##### TAB 1 – CoMet
1404 #####
1405
1406 Globals.tab1_canvas.config(bg="#ffffff", bd = 0, relief=FLAT,
1407 highlightthickness=0)
1408
1409 CoMet_explained = tk.Text(Globals.tab1_canvas, height=4, width=105)
1410 CoMet_explained.insert(INSERT, \

```

```

1400 " Start the correction by choosing the correct *.tif file containing the
1401 scanned image of the \n\
1402 GafChromic film. The film should be scanned using Epson Perfection v750
1403 Pro with dpi setting \n\
1404 72 or 127. Then pick which folder the corrected file should be uploaded to
1405 . The corrected file\n\
1406 will be saved as a DICOM. Write filename and patient name (optional)
1407 before doing the correction.\n\
1408 An illustration of the corrected image will appear.")
1409 CoMet_explained.grid(row=0, column = 0, columnspan=1, sticky=N+S+E+W, padx
1410 =(20,0), pady=(10,10))
1411 Globals.tab1_canvas.grid_columnconfigure(0, weight=0)
1412 Globals.tab1_canvas.grid_rowconfigure(0, weight=0)
1413 CoMet_explained.config(state=DISABLED, bg='#ffffff', font=('calibri', '11',
1414 ), relief=FLAT)

1415 Globals.CoMet_border_1_label = Label(Globals.tab1_canvas, image =
1416 CoMet_border_dark, width=50)
1417 Globals.CoMet_border_1_label.image=CoMet_border_dark
1418 Globals.CoMet_border_1_label.grid(row=1, column=0, columnspan=2, sticky =
1419 W+E, padx = (0, 190), pady=(10,5))
1420 Globals.tab1_canvas.grid_columnconfigure(1, weight=0)
1421 Globals.tab1_canvas.grid_rowconfigure(1, weight=0)
1422 Globals.CoMet_border_1_label.config(bg='#ffffff', borderwidth=0)

1423 CoMet_upload_button_frame = tk.Frame(Globals.tab1_canvas)
1424 CoMet_upload_button_frame.grid(row=1, column = 0, padx = (200, 0), pady
1425 =(10,5))
1426 Globals.tab1_canvas.grid_columnconfigure(2, weight=0)
1427 Globals.tab1_canvas.grid_rowconfigure(2, weight=0)
1428 CoMet_upload_button_frame.config(bg = '#ffffff')

1429 CoMet_upload_button = tk.Button(CoMet_upload_button_frame, text='Browse',
1430 image = Globals.upload_button_image, \
1431 cursor='hand2',font=('calibri', '14'), relief=FLAT, state=ACTIVE,
1432 command=CoMet_functions.UploadAction)
1433 CoMet_upload_button.pack(expand=True, fill=BOTH)
1434 CoMet_upload_button.config(bg='#ffffff', activebackground='#ffffff',
1435 activeforeground='#ffffff', highlightthickness=0)
1436 CoMet_upload_button.image = Globals.upload_button_image

1437 Globals.CoMet_uploaded_file_text = tk.Text(Globals.CoMet_border_1_label,
1438 height=1, width=31)
1439 Globals.CoMet_uploaded_file_text.grid(row=0, column=0, columnspan=2,
1440 sticky=E+W, pady=(20,20), padx=(80,0))
1441 Globals.CoMet_uploaded_file_text.insert(INSERT, "Upload the image you want
1442 to correct")
1443 Globals.CoMet_uploaded_file_text.config(state=DISABLED, bd=0, font=('
1444 calibri', '12'), fg='gray', bg='#ffffff')

1445 Globals.CoMet_border_2_label = Label(Globals.tab1_canvas, image =
1446 CoMet_border_dark, width=50)
1447 Globals.CoMet_border_2_label.image=CoMet_border_dark
1448 Globals.CoMet_border_2_label.grid(row=2, column=0, columnspan=2, sticky =
1449 N+S+W+E, padx = (0, 190), pady=(0,15))
1450 Globals.tab1_canvas.grid_columnconfigure(3, weight=0)
1451 Globals.tab1_canvas.grid_rowconfigure(3, weight=0)

```

```

1440     Globals.CoMet_border_2_label.config(bg='#ffffff', borderwidth=0)
1441
1442     CoMet_folder_button_frame = tk.Frame(Globals.tab1_canvas)
1443     CoMet_folder_button_frame.grid(row=2, column = 0, padx = (200, 0), pady
1444         =(0,15))
1445     Globals.tab1_canvas.grid_columnconfigure(4, weight=0)
1446     Globals.tab1_canvas.grid_rowconfigure(4, weight=0)
1447     CoMet_folder_button_frame.config(bg = '#ffffff')
1448
1449     CoMet_folder_button = tk.Button(CoMet_folder_button_frame, text='Browse',
1450         image = select_folder_image ,cursor='hand2',font=('calibri', '14'),\
1451             relief=FLAT, state=ACTIVE, command=CoMet_functions.
1452                 setCoMet_export_folder)
1453     CoMet_folder_button.pack(expand=True, fill=BOTH)
1454     CoMet_folder_button.config(bg='#ffffff', activebackground='#ffffff',
1455         activeforeground='#ffffff', highlightthickness=0)
1456     CoMet_folder_button.image=select_folder_image
1457
1458     CoMet_save_to_folder = tk.Text(Globals.CoMet_border_2_label, height=1,
1459         width=31)
1460     CoMet_save_to_folder.grid(row=0, column=0, columnspan=2, sticky=E+W, pady
1461         =(25,0), padx=(80,0))
1462     CoMet_save_to_folder.insert(INSERT,"Folder to save the corrected image")
1463     CoMet_save_to_folder.config(state=DISABLED, bd=0, font=('calibri', '12'),
1464         fg='gray', bg='#ffffff')
1465
1466     ## Function to test the filename the user chooses for the corrected image
1467     def testFilename():
1468         Globals.CoMet_corrected_image_filename.set(Globals.CoMet_save_filename
1469             .get("1.0",'end-1c'))
1470         if(Globals.CoMet_corrected_image_filename.get() == " " or Globals.
1471             CoMet_corrected_image_filename.get() == "Filename"):
1472             Globals.CoMet_corrected_image_filename.set("Error!")
1473         elif(len(Globals.CoMet_corrected_image_filename.get()) >21):
1474             messagebox.showerror("Error", "The filename must be under 20
1475 characters")
1476             Globals.CoMet_corrected_image_filename.set("Error!")
1477         elif(re.match("[A-Za-z0-9]*$", (Globals.
1478             CoMet_corrected_image_filename.get()).lstrip())==None):
1479             messagebox.showerror("Error", "Filename can only contain letters
1480 and/or numbers")
1481             Globals.CoMet_corrected_image_filename.set("Error!")
1482         else:
1483             Globals.CoMet_save_button_1.config(state=DISABLED)
1484             Globals.CoMet_save_filename.config(state=DISABLED)
1485             Globals.CoMet_progressbar_counter += 1
1486             Globals.CoMet_progressbar["value"] = Globals.
1487             CoMet_progressbar_counter*25
1488             Globals.CoMet_progressbar_text = tk.Text(Globals.tab1_canvas,
1489                 width = 5, height=1)
1490             Globals.CoMet_progressbar_text.grid(row=5, column=0, columnspan=1,
1491                 sticky=E, padx=(0,158), pady=(27,0))
1492             Globals.CoMet_progressbar_text.insert(INSERT, str(Globals.
1493                 CoMet_progressbar_counter*25) + "%")
1494             if(Globals.CoMet_progressbar_counter*25 == 100):
1495                 Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
1496                     relief=FLAT, bg='#2C8EAD', font=('calibri', '10', 'bold'))

```

```

1480     else:
1481         Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
1482             relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))
1483
1482 Globals.CoMet_border_3_label = Label(Globals.tab1_canvas, image =
1483     CoMet_border_dark)
1484 Globals.CoMet_border_3_label.image=CoMet_border_dark
1484 Globals.CoMet_border_3_label.grid(row=3, column=0, columnspan=2, sticky =
1485     W+E, padx = (0,190), pady=(0,15))
1486 Globals.tab1_canvas.grid_columnconfigure(5, weight=0)
1486 Globals.tab1_canvas.grid_rowconfigure(5, weight=0)
1488 Globals.CoMet_border_3_label.config(bg='#ffffff', borderwidth=0)
1489
1490 Globals.CoMet_save_button_frame_1 = tk.Frame(Globals.tab1_canvas)
1490 Globals.CoMet_save_button_frame_1.grid(row=3, column = 0, padx = (200, 0),
1491     pady=(0,15))
1492 Globals.tab1_canvas.grid_columnconfigure(6, weight=0)
1492 Globals.tab1_canvas.grid_rowconfigure(6, weight=0)
1494 Globals.CoMet_save_button_frame_1.config(bg = '#ffffff')
1495
1496 Globals.CoMet_save.button_1 = tk.Button(Globals.CoMet_save_button_frame_1,
1497     text='Save', image = CoMet_save_button ,cursor='hand2',font=('calibri',
1498     ', '14'),\
1498     relief=FLAT, state=ACTIVE, command=testFilename)
1499 Globals.CoMet_save.button_1.pack(expand=True, fill=BOTH)
1500 Globals.CoMet_save.button_1.config(bg='#ffffff', activebackground='#ffffff',
1500     activeforeground='#ffffff', highlightthickness=0)
1501 Globals.CoMet_save.button_1.image = CoMet_save_button
1502
1504 Globals.CoMet_save_filename = tk.Text(Globals.CoMet_border_3_label, height
1504     =1, width=30)
1505 Globals.CoMet_save_filename.grid(row=0, column=0, columnspan=2, sticky=E+W
1505     , pady=(20,20), padx=(80,0))
1506 Globals.CoMet_save_filename.insert(END,"Filename (will be saved as *.dcm)")
1506 )
1507 Globals.CoMet_save_filename.config(state=NORMAL, bd=0, font=('calibri', '12'),
1507     fg='gray', bg='#ffffff')
1508
1510 def writeFilename(event):
1511     current = Globals.CoMet_save_filename.get("1.0", tk.END)
1512     if(current == "Filename (will be saved as *.dcm)\n"):
1513         Globals.CoMet_save_filename.delete("1.0", tk.END)
1514     else:
1515         Globals.CoMet_save_filename.insert("1.0", "Filename (will be saved
1515             as *.dcm)")
1516
1516 Globals.CoMet_save_filename.bind("<FocusIn>", writeFilename)
1518 Globals.CoMet_save_filename.bind("<FocusOut>", writeFilename)
1519
1520 #Function to validate the patient name written in by the user
1521 def testName():

```

```

1524     Globals.CoMet_patientName.set(CoMet_save_patientName.get("1.0",'end-1c'))
1525
1526 if(Globals.CoMet_patientName.get() == " " or Globals.CoMet_patientName
1527 .get() == "Patient name"):
1528     Globals.CoMet_patientName.set("Error!")
1529 elif(len(Globals.CoMet_patientName.get()) >31):
1530     messagebox.showerror("Error", "The Name must be under 30
1531 characters")
1532     Globals.CoMet_patientName.set("Error!")
1533 elif(re.match("[A-Za-z0-9_]*$", (Globals.CoMet_patientName.get()).
1534 lstrip())==None):
1535     messagebox.showerror("Error","Name can only contain letters (not
1536     , , ) and no spaces")
1537     Globals.CoMet_patientName.set("Error!")
1538 else:
1539     CoMet_save_button_2.config(state=DISABLED)
1540     CoMet_save_patientName.config(state=DISABLED)

1541
1542 Globals.CoMet_border_4_label = Label(Globals.tab1_canvas, image =
1543     CoMet_border_dark)
1544 Globals.CoMet_border_4_label.image=CoMet_border_dark
1545 Globals.CoMet_border_4_label.grid(row=4, column=0, columnspan=2, sticky =
1546     W+E, padx = (0, 190), pady=(5,0))
1547 Globals.tab1_canvas.grid_columnconfigure(7, weight=0)
1548 Globals.tab1_canvas.grid_rowconfigure(7, weight=0)
1549 Globals.CoMet_border_4_label.config(bg='#fffff', borderwidth=0)

1550 CoMet_save_button_frame_2 = tk.Frame(Globals.tab1_canvas)
1551 CoMet_save_button_frame_2.grid(row=4, column = 0, padx = (200, 0), pady
1552     =(5,0))
1553 Globals.tab1_canvas.grid_columnconfigure(8, weight=0)
1554 Globals.tab1_canvas.grid_rowconfigure(8, weight=0)
1555 CoMet_save_button_frame_2.config(bg = '#fffff')

1556 CoMet_save_button_2 = tk.Button(CoMet_save_button_frame_2, text='Save',
1557     image = CoMet_save_button ,cursor='hand2',font=('calibri', '14'),\
1558     relief=FLAT, state=ACTIVE, command=testName)
1559 CoMet_save_button_2.pack(expand=True, fill=BOTH)
1560 CoMet_save_button_2.config(bg='#fffff', activebackground='#fffff',
1561     activeforeground='#fffff', highlightthickness=0)
1562 CoMet_save_button_2.image = CoMet_save_button

1563
1564 CoMet_save_patientName = tk.Text(Globals.CoMet_border_4_label, height=1,
1565     width=30)
1566 CoMet_save_patientName.grid(row=0, column=0, columnspan=2, sticky=E+W,
1567     pady=(20,20), padx=(80,0))
1568 CoMet_save_patientName.insert(END,"Patient name (Optional)")
1569 CoMet_save_patientName.config(state=NORMAL, bd=0, font=('calibri', '12'),
1570     fg='gray', bg='#fffff')

1571 def writePname(event):
1572     current = CoMet_save_patientName.get("1.0", tk.END)
1573     if(current == "Patient name (Optional)\n"):
1574         CoMet_save_patientName.delete("1.0", tk.END)
1575     else:
1576         CoMet_save_patientName.insert("1.0", "Patient name (Optional)")
```

```

1568 CoMet_save_patientName.bind("<FocusIn>", writePname)
1569 CoMet_save_patientName.bind("<FocusOut>", writePname)
1570
1571 CoMet_correct_button_frame = tk.Frame(Globals.tab1_canvas)
1572 CoMet_correct_button_frame.grid(row=4, column=2, rowspan=2, padx = (0, 0),
1573     pady=(0,0), sticky=W)
1574 Globals.tab1_canvas.grid_columnconfigure(9, weight=0)
1575 Globals.tab1_canvas.grid_rowconfigure(9, weight=0)
1576 CoMet_correct_button_frame.config(bg = '#ffffff')
1577
1578 CoMet_correct_button = tk.Button(CoMet_correct_button_frame, text='Correct',
1579     image = CoMet_correct_button.image ,cursor='hand2',font=('calibri',
1580     '14'),\
1581     relief=FLAT, state=ACTIVE, command=CoMet_functions.Correct)
1582 CoMet_correct_button.pack(expand=True, fill=BOTH)
1583 CoMet_correct_button.config(bg='#ffffff', activebackground='#ffffff',
1584     activeforeground='#ffffff', highlightthickness=0)
1585 CoMet_correct_button.image = CoMet_correct_button.image
1586
1587 Globals.CoMet_print_corrected_image = tk.Canvas(Globals.tab1_canvas ,
1588     width=240, height=290)
1589 Globals.CoMet_print_corrected_image.grid(row=0, column=2, rowspan=3,
1590     sticky=N+W+S+E, pady=(20,0), padx=(0,0))
1591 Globals.CoMet_print_corrected_image.config(bg='#ffffff', bd = 0, relief=
1592     FLAT)
1593 Globals.tab1_canvas.grid_columnconfigure(11, weight=0)
1594 Globals.tab1_canvas.grid_rowconfigure(11, weight=0)
1595 Globals.CoMet_print_corrected_image.create_image(123,148,image=
1596     CoMet_empty_image.image)
1597 Globals.CoMet_print_corrected_image.image = CoMet_empty_image.image
1598
1599
1600 def clearAll():
1601     #Clear out the filename
1602     Globals.CoMet_uploaded_file_text = tk.Text(Globals.
1603         CoMet_border_1_label, height=1, width=31)
1604     Globals.CoMet_uploaded_file_text.grid(row=0, column=0, columnspan=2,
1605         sticky=E+W, pady=(20,20), padx=(80,0))
1606     Globals.CoMet_uploaded_file_text.insert(INSERT, "Upload the image you
1607         want to correct")
1608     Globals.CoMet_uploaded_file_text.config(state=DISABLED, bd=0, font=(
1609         'calibri', '12'), fg='gray', bg='#ffffff')
1610     Globals.CoMet_uploaded_filename.set("Error!")
1611
1612     #Clear out folder
1613     CoMet_save_to_folder = tk.Text(Globals.CoMet_border_2_label, height=1,
1614         width=32)
1615     CoMet_save_to_folder.grid(row=0, column=0, columnspan=2, sticky=E+W,
1616         pady=(25,0), padx=(80,0))
1617     CoMet_save_to_folder.insert(INSERT,"Folder to save the corrected image
1618         ")
1619     CoMet_save_to_folder.config(state=DISABLED, bd=0, font=( 'calibri', '12
1620         '), fg='gray', bg='#ffffff')
1621     Globals.CoMet_export_folder.set("Error!")
1622
1623     #Clear filename of corrected file

```

```

1608     Globals . CoMet_save_filename = tk . Text (Globals . CoMet_border_3_label ,
1609         height=1, width=30)
1610     Globals . CoMet_save_filename . grid (row=0, column=0, columnspan=2, sticky
1611         =E+W, pady=(20,20), padx=(80,0))
1612     Globals . CoMet_save_filename . insert (END, "Filename (will be saved as *.dcm)")
1613     Globals . CoMet_save_filename . config (state=NORMAL, bd=0, font=( 'calibri' ,
1614         '12'), fg='gray', bg='#ffffff')
1615     Globals . CoMet_corrected_image_filename . set ("Error!")
1616     Globals . CoMet_save_button_1 . config (state=ACTIVE)
1617
1618     def writeFilename(event):
1619         current = Globals . CoMet_save_filename . get ("1.0", tk .END)
1620         if (current == "Filename (will be saved as *.dcm)\n"):
1621             Globals . CoMet_save_filename . delete ("1.0", tk .END)
1622         else:
1623             Globals . CoMet_save_filename . insert ("1.0", "Filename (will be
1624             saved as *.dcm)")
1625
1626         Globals . CoMet_save_filename . bind ("<FocusIn>", writeFilename)
1627         Globals . CoMet_save_filename . bind ("<FocusOut>", writeFilename)
1628
1629     #Clear patientname
1630     CoMet_save_patientName = tk . Text (Globals . CoMet_border_4_label , height
1631         =1, width=30)
1632     CoMet_save_patientName . grid (row=0, column=0, columnspan=2, sticky=E+W,
1633         pady=(20,20), padx=(80,0))
1634     CoMet_save_patientName . insert (END, "Patient name (Optional)")
1635     CoMet_save_patientName . config (state=NORMAL, bd=0, font=( 'calibri' , '12
1636         ), fg='gray', bg='#ffffff')
1637     Globals . CoMet_patientName . set ("Error!")
1638     CoMet_save_button_2 . config (state=ACTIVE)
1639
1640     def writePname(event):
1641         current = CoMet_save_patientName . get ("1.0", tk .END)
1642         if (current == "Patient name (Optional)\n"):
1643             CoMet_save_patientName . delete ("1.0", tk .END)
1644         else:
1645             CoMet_save_patientName . insert ("1.0", "Patient name (Optional)")
1646
1647
1648     CoMet_save_patientName . bind ("<FocusIn>", writePname)
1649     CoMet_save_patientName . bind ("<FocusOut>", writePname)
1650
1651     #Clear image
1652     Globals . CoMet_print_corrected_image . delete ('all')
1653     Globals . CoMet_print_corrected_image . create_image (123,148,image=
1654         CoMet_empty_image_image)
1655     Globals . CoMet_print_corrected_image . image = CoMet_empty_image_image
1656
1657     #Clear progressbar
1658     Globals . CoMet_progressbar [ "value" ]=0
1659     Globals . CoMet_progressbar_counter = 0
1660     Globals . CoMet_progressbar_check_file = True
1661     Globals . CoMet_progressbar_check_folder = True

```

```

1654     CoMet_progressbar_text = tk.Text(Globals.tab1_canvas, height=1, width=5)
1655     CoMet_progressbar_text.grid(row=5, column=0, columnspan=1, sticky=E, padx=(0,158), pady=(27,0))
1656     CoMet_progressbar_text.insert(INSERT, "0%")
1657     CoMet_progressbar_text.config(state=DISABLED, bd=0, relief=FLAT, bg="#fffff", font=('calibri', '10', 'bold'))
1658
1660
1662 CoMet_clear_all_button_frame = tk.Frame(Globals.tab1_canvas)
1663 CoMet_clear_all_button_frame.grid(row=4, column=2, rowspan=2, padx=(100,0), pady=(0,0), sticky=E)
1664 Globals.tab1_canvas.grid_columnconfigure(13, weight=0)
1665 Globals.tab1_canvas.grid_rowconfigure(13, weight=0)
1666 CoMet_clear_all_button_frame.config(bg="#fffff")
1667
1668 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'),\n    relief=FLAT, state=ACTIVE, command=clearAll)
1669 CoMet_clear_all_button.pack(expand=True, fill=BOTH)
1670 CoMet_clear_all_button.config(bg="#fffff", activebackground="#fffff",\n    activeforeground="#fffff", highlightthickness=0)
1671 CoMet_clear_all_button.image=CoMet_clear_all_button_image
1672
1673 Globals.tab1_canvas.pack(expand=True, fill=BOTH)
1674
1675 ##### TAB 2 – Dose response #####
1676
1677 """To be able to perform an accurate dose caluclations using GafChromic\n    film EBT3 \n#\n#it is necessary to create a dose–respons curve for each batch of film, in\n    addition\n#\n#to a calibration scan before/along every use. The respons of GafChromic\n    film \n#\n#EBT3 is modelled using a rational function,  $X(D,n) = a + b/(D-c)$ , as this\n    has\n#\n#proven to fit well with the film behavior. In the model  $X(D,n)$  is the\n    scanner\n#\n#respons in color channel n and a, b and c are constants. Because of the\n    nature\n#\n#of asymptotic fitting functions a good fit will be achieved by using\n    doses in\n#\n#geomteric progression, D, nD, nnD, etc.. Also, to avoid scanner\n    uncertainties\n#\n#each dose should be scannet three times and uploaded here where an\n    average will be used."""
1681
1682 """Irradiate film piece of size (Bestemt med maske?) with known doses. Place\n    one and one\n#\n#film piece in the center of the scanner and perfom three scans per dose.\n"""
1683
1684

```

```

1691 Globals.tab2.canvas.config(bg='#fffff', bd = 0, relief=FLAT,
1692     highlightthickness=0)
1693 dose_response_explain_text = tk.Text(Globals.tab2_canvas, height=4, width
1694     =140)
1695 dose_response_explain_text.insert(INSERT, "\\\n"
1696     Follow the calibration specifications given under 'Help' or 'Read more' at
1697     the first window. Upload the scanned *.tif files (there should be at
1698     least 3 of each\\\n")
1699 dose level) and save. The dose response curve along with the equation will
1700     appear when enough data points are given. The uploaded files must
1701     have dpi \n\\
1702 setting 72 or 127. When saving the calibration the dose response data will
1703     be saved and can be used chosen for later use of this software. The
1704     dose response \n\\
1705 curve will be found for all three color channels, but can be removed using
1706     the check boxes. A dose response equation will only be fitted for the
1707     red channel. ")
1708 dose_response_explain_text.grid(row=0, column=0, columnspan=5, sticky=N+S+
1709     E+W, pady=(20,20), padx=(20,10))
1710 Globals.tab2_canvas.grid_columnconfigure(0, weight=0)
1711 Globals.tab2_canvas.grid_rowconfigure(0, weight=0)
1712 dose_response_explain_text.config(state=DISABLED, font=('calibri', '11'),
1713     bg ='#fffff', relief=FLAT)

1714 dose_response_upload_button_frame = tk.Frame(Globals.tab2_canvas_files)
1715 dose_response_upload_button_frame.grid(row=0, column = 0, columnspan=8,
1716     padx = (60, 0), pady=(10,5))
1717 Globals.tab2_canvas_files.grid_columnconfigure(0, weight=0)
1718 Globals.tab2_canvas_files.grid_rowconfigure(0, weight=0)
1719 dose_response_upload_button_frame.config(bg = '#fffff')

1720 dose_response_upload_button = tk.Button(dose_response_upload_button_frame,
1721     text='Upload file', image=Globals.upload_button_image, \
1722     cursor='hand2', font=('calibri', '14'), relief=FLAT, state=ACTIVE,
1723     command=Dose_response_functions.create_window)
1724 dose_response_upload_button.pack(expand=True, fill=BOTH)
1725 dose_response_upload_button.config(bg='#fffff', activebackground='#fffff',
1726     activeforeground='#fffff', highlightthickness=0)
1727 dose_response_upload_button.image = Globals.upload_button_image

1728 check1 = Checkbutton(Globals.tab2_canvas_files, variable=Globals.
1729     dose_response_var1, command=Dose_response_functions.plot_dose_response
1730     )
1731 check1.grid(row=1, column=1, sticky=E, padx=(30,15))
1732 Globals.tab2_canvas_files.grid_columnconfigure(5, weight=0)
1733 Globals.tab2_canvas_files.grid_rowconfigure(5, weight=0)
1734 check1.config(bg='#fffff')

1735 check2 = Checkbutton(Globals.tab2_canvas_files, variable=Globals.
1736     dose_response_var2, command=Dose_response_functions.plot_dose_response
1737     )
1738 check2.grid(row=1, column=3, sticky=E, padx=(45,15))
1739 Globals.tab2_canvas_files.grid_columnconfigure(6, weight=0)
1740 Globals.tab2_canvas_files.grid_rowconfigure(6, weight=0)
1741 check2.config(bg='#fffff')

```

```

1728 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')
1730
1732
1734 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'))
1740
1742 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'))
1744
1746 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'))
1750
1752
1754 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'))
1756
1758
1760
1762 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)
1764
1766
1768 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)
1770
1772 Globals.dose_response_save_calibration_button.config(bg='#ffffff',
    activebackground='#ffffff', activeforeground='#ffffff',
    highlightthickness=0)

```

```

1774 Globals.dose_response_save_calibration_button.image =
1775     dose_response_calibration_button_image
1776 dose_response_clear_all_button_frame = tk.Frame(Globals.tab2_canvas)
1777 dose_response_clear_all_button_frame.grid(row=2, column=1, sticky=N+S+E+W,
1778     padx=(0,0), pady=(120,0))
1779 Globals.tab2_canvas.grid_columnconfigure(11, weight=0)
1780 Globals.tab2_canvas.grid_rowconfigure(11, weight=0)
1781 dose_response_clear_all_button_frame.config(bg='#ffffff', height=1, width
1782     =100)
1783 dose_response_clear_all_button_frame.grid_propagate(0)

1784 dose_response_clear_all_button = tk.Button(
1785     dose_response_clear_all_button_frame, text='Clear all', image=
1786     dose_response_clear_all_button_image, \
1787     cursor='hand2', font=('calibri', '12'), relief=FLAT, state=ACTIVE,
1788     command=Dose_response_functions.clear_all)
1789 dose_response_clear_all_button.pack(expand=True, fill=BOTH, side=TOP)
1790 dose_response_clear_all_button.config(bg='#ffffff', activebackground='#
1791     ffffff', activeforeground='#ffffff', highlightthickness=0)
1792 dose_response_clear_all_button.image =
1793     dose_response_clear_all_button_image

1794 delete_text = tk.Text(Globals.tab2_canvas_files, height=1, width=7)
1795 delete_text.insert(INSERT, "Delete")
1796 delete_text.grid(row=1, column=7, sticky=N+S+E+W, padx=(0,0))
1797 Globals.tab2_canvas_files.grid_columnconfigure(4, weight=0)
1798 Globals.tab2_canvas_files.grid_rowconfigure(4, weight=0)
1799 delete_text.config(state=DISABLED, bd=0, font=('calibri', '12'))

1800 Globals.tab2_canvas.pack(expand=True, fill=BOTH)
1801 ##### TAB 3 – Map dose
1802 ##### TAB 3 – Map dose

1803 #path = os.path.dirname(sys.argv[0])
1804 #path= "upload.png"
1805 #upload_button_image = ImageTk.PhotoImage(file=path)

1806 Globals.tab3_canvas.config(bg='#ffffff', bd = 0, relief=FLAT,
1807     highlightthickness=0)

1808 upload_film_data = tk.Button(Globals.tab3_canvas, text='Upload',image=
1809     Globals.upload_button_image, cursor='hand2', font=('calibri', '12'), \
1810     relief=FLAT, state=ACTIVE, width=12, command=lambda: Map_Dose.
1811     UploadAction("FILM"))
1812 upload_film_data.place(relwidth=0.17, relheight=0.11, relx=0.3, rely=0.03)
1813 upload_film_data.image = Globals.upload_button_image

1814 Globals.tab3_canvas.pack(expand=True, fill=BOTH)
1815 ##### TAB 4 – Profiles
1816 ##### TAB 4 – Profiles

1817 Globals.tab4_canvas.config(bg='#ffffff', bd = 0, relief=FLAT,
1818     highlightthickness=0)

```

```

1816 profiles_explain_text = tk.Text(Globals.tab4_canvas, height=4, width=140)
1817 profiles_explain_text.insert(INSERT, "\n")
1818 SliceThickness i plan m v re ['1','1'], ['2','2'] eller ['3','3'],
1819 Filmen m legges i xy, xz eller yz planet (lage figur?), Filmen m
1820 scannes \n\
1821 parallelt med retningene i skanneren (programmet vil anta dette), man m
1822 markere \"opp\" og \"bort\" p filmen. N r man skanner m man legge
1823 \n\
1824 oppmerket oppover og bort merket mot h yre (kan man kreve dette i alle
1825 plan?). \
1826 Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her
1827 kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her
1828 kommer det, \n\
1829 Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her
1830 kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her kommer det tekst, Her
1831 kommer det, " )
1832 profiles_explain_text.grid(row=0, column=0, columnspan=5, sticky=N+S+E+W,
1833 pady=(20,20), padx=(20,10))
1834 Globals.tab4_canvas.grid_columnconfigure(0, weight=0)
1835 Globals.tab4_canvas.grid_rowconfigure(0, weight=0)
1836 profiles_explain_text.config(state=DISABLED, font=('calibri', '11'), bg =
1837 '#E5f9ff', relief=FLAT)

1838 profiles_upload_film_frame = tk.Frame(Globals.tab4_canvas)
1839 profiles_upload_film_frame.grid(row=3, column = 0, padx = (0, 240), pady
1840 =(10,0), sticky=N)
1841 Globals.tab4_canvas.grid_columnconfigure(1, weight=0)
1842 Globals.tab4_canvas.grid_rowconfigure(1, weight=0)
1843 profiles_upload_film_frame.config(bg = '#ffffff')

1844 Globals.profiles_upload_button_film = tk.Button(profiles_upload_film_frame
1845 , text='Browse', image = profiles_add_film_button_image, \
1846 cursor='hand2',font=('calibri', '14'), relief=FLAT, state=ACTIVE,
1847 command=Profile_functions.UploadFilm)
1848 Globals.profiles_upload_button_film.pack(expand=True, fill=BOTH)
1849 Globals.profiles_upload_button_film.config(bg='#ffffff', activebackground=
1850 '#ffffff', activeforeground='#ffffff', highlightthickness=0)
1851 Globals.profiles_upload_button_film.image = profiles_add_film_button_image

1852 profiles_upload_doseplan_frame = tk.Frame(Globals.tab4_canvas)
1853 profiles_upload_doseplan_frame.grid(row=3, column = 0, padx = (0,40), pady
1854 =(10,0), sticky=N)
1855 Globals.tab4_canvas.grid_columnconfigure(3, weight=0)
1856 Globals.tab4_canvas.grid_rowconfigure(3, weight=0)
1857 profiles_upload_film_frame.config(bg = '#ffffff')

1858 Globals.profiles_upload_button_doseplan = tk.Button(
1859     profiles_upload_doseplan_frame, text='Browse', image=Globals.
1860     profiles_add_doseplan_button_image,\ \
1861     cursor='hand2', font=('calibri', '14'), relief=FLAT, state=DISABLED,
1862     command=Profile_functions.UploadDoseplan_button_function)
1863 Globals.profiles_upload_button_doseplan.pack(expand=True, fill=BOTH)
1864 Globals.profiles_upload_button_doseplan.configure(bg='#ffffff',
1865     activebackground='#ffffff', activeforeground='#ffffff',
1866     highlightthickness=0)
1867 Globals.profiles_upload_button_doseplan.image = Globals.
1868     profiles_add_doseplan_button_image

```

```

1852 profiles_upload_rtplan_frame = tk.Frame(Globals.tab4_canvas)
1853 profiles_upload_rtplan_frame.grid(row=3, column=0, padx=(160,0), pady
1854 =(10,0), sticky=N)
1855 Globals.tab4_canvas.grid_columnconfigure(10, weight=0)
1856 Globals.tab4_canvas.grid_rowconfigure(10, weight=0)
1857 profiles_upload_rtplan_frame.config(bg='#ffffff')

1858 Globals.profiles_upload_button_rtplan = tk.Button(
1859     profiles_upload_rtplan_frame, text='Browse', image=
1860     profiles_add_rtplan_button_image, \
1861     cursor='hand2', font=('calibri', '14'), relief=FLAT, state=DISABLED,
1862     command=Profile_functions.UploadRTplan)
1863 Globals.profiles_upload_button_rtplan.pack(expand=True, fill=BOTH)
1864 Globals.profiles_upload_button_rtplan.configure(bg='#ffffff',
1865     activebackground='#ffffff', activeforeground='#ffffff',
1866     highlightthickness=0)
1867 Globals.profiles_upload_button_rtplan.image=
1868     profiles_add_rtplan_button_image

1869 Globals.profiles_film_orientation_menu = OptionMenu(Globals.tab4_canvas,
1870     Globals.profiles_film_orientation, 'Axial', 'Coronal', 'Sagittal')
1871 Globals.profiles_film_orientation_menu.grid(row=1, column=0, sticky=N+S,
1872     padx=(60,0))
1873 Globals.tab4_canvas.grid_columnconfigure(2, weight=0)
1874 Globals.tab4_canvas.grid_rowconfigure(2, weight=0)
1875 Globals.profiles_film_orientation_menu.config(bg = '#ffffff', width=15,
1876     relief=FLAT)

1877 film_orientation_menu_text = tk.Text(Globals.tab4_canvas, width=14, height
1878 =1)
1879 film_orientation_menu_text.insert(INSERT, "Film orientation:")
1880 film_orientation_menu_text.config(state=DISABLED, font=('calibri', '10'),
1881     bd = 0, relief=FLAT)
1882 film_orientation_menu_text.grid(row=1, column=0, sticky=N+S+W, padx=(30,0),
1883     pady=(5,0))
1884 Globals.tab4_canvas.grid_columnconfigure(3, weight=0)
1885 Globals.tab4_canvas.grid_rowconfigure(3, weight=0)

1886 profiles_film_orientation_help_frame = tk.Frame(Globals.tab4_canvas)
1887 profiles_film_orientation_help_frame.grid(row=1, column=0, sticky=N+S+E,
1888     padx=(0,40))
1889 Globals.tab4_canvas.grid_columnconfigure(6, weight=0)
1890 Globals.tab4_canvas.grid_rowconfigure(6, weight=0)
1891 profiles_film_orientation_help_frame.configure(bg='#ffffff')

1892 profiles_help_button_orientation = tk.Button(
1893     profiles_film_orientation_help_frame, text='help', image=Globals.
1894     help_button, \
1895     cursor='hand2', font=('calibri', '14'), relief=FLAT, state=ACTIVE,
1896     command=Profile_functions.help_showPlanes)
1897 profiles_help_button_orientation.pack(expand=True, fill=BOTH)
1898 profiles_help_button_orientation.configure(bg='#ffffff', activebackground='
1899     #ffffff', activeforeground='#ffffff', highlightthickness=0)
1900 profiles_help_button_orientation.image=Globals.help_button

1901 profiles_film_factor = tk.Text(Globals.tab4_canvas, width=20, height=2)

```

```

1890 profiles_film_factor.insert(INSERT, "Film factor \n(number of fractions):")
    )
profiles_film_factor.config(state=DISABLED, font=('calibri', '10'), bd =
    0, relief=FLAT)
1892 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)

1896 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))
1898 Globals.profiles_film_factor_input.insert(INSERT, " ")
Globals.profiles_film_factor_input.config(state=NORMAL, font=('calibri', '10'), bd = 2, bg='#ffffff')
1900 Globals.tab4_canvas.grid_columnconfigure(31, weight=0)
Globals.tab4_canvas.grid_rowconfigure(31, weight=0)
1902

1904 profiles_resetAll_frame = tk.Frame(Globals.tab4_canvas)
profiles_resetAll_frame.grid(row=18,column=0, padx=(0,0), pady=(0,0),
    sticky=S)
1906 Globals.tab4_canvas.grid_columnconfigure(5, weight=0)
Globals.tab4_canvas.grid_rowconfigure(5, weight=0)
1908 profiles_resetAll_frame.config(bg='#ffffff')

1910 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)
1912 profiles_resetAll_button.pack(expand=True, fill=BOTH)
profiles_resetAll_button.configure(bg='#ffffff', activebackground='#ffffff',
    activeforeground='#ffffff', highlightthickness=0)
1914 profiles_resetAll_button.image = dose_response_clear_all_button_image

1916
1917 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()))
1918 Globals.profiles_adjust_button_left.pack(side=LEFT)
1919 Globals.profiles_adjust_button_left.config(bg='#ffffff', activebackground='#ffffff',
    activeforeground='#ffffff', highlightthickness=0)
1920 Globals.profiles_adjust_button_left.image = Globals.
    adjust_button_left_image

1922 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()))
1923 Globals.profiles_adjust_button_up.pack(side=LEFT)

```

```

1926 Globals.profiles_adjust_button_up.config(bg='#ffffff', activebackground='#
1927     ffffff', activeforeground='#ffffff', highlightthickness=0)
1928 Globals.profiles_adjust_button_up.image = Globals.adjust_button_up_image
1929
1930 Globals.profiles_adjust_button_down = tk.Button(Globals.
1931     profiles_redefine_film_ROI_frame, text="left", image=Globals.
1932         adjust_button_down_image,\n
1933         cursor='hand2', font=('calibri', '12'), relief=FLAT, state=DISABLED,
1934         command=lambda: Profile_functions.adjustROIDown(Globals.
1935             profiles_choice_of_profile_line_type.get()))
1936 Globals.profiles_adjust_button_down.pack(side=LEFT)
1937 Globals.profiles_adjust_button_down.config(bg='#ffffff', activebackground=
1938     '#ffffff', activeforeground='#ffffff', highlightthickness=0)
1939 Globals.profiles_adjust_button_down.image = Globals.
1940     adjust_button_down_image
1941
1942 Globals.profiles_adjust_button_right = tk.Button(Globals.
1943     profiles_redefine_film_ROI_frame, text="left", image=Globals.
1944         adjust_button_right_image,\n
1945         cursor='hand2', font=('calibri', '12'), relief=FLAT, state=DISABLED,
1946         command=lambda: Profile_functions.adjustROIRight(Globals.
1947             profiles_choice_of_profile_line_type.get()))
1948 Globals.profiles_adjust_button_right.pack(side=LEFT)
1949 Globals.profiles_adjust_button_right.config(bg='#ffffff', activebackground=
1950     '#ffffff', activeforeground='#ffffff', highlightthickness=0)
1951 Globals.profiles_adjust_button_right.image = Globals.
1952     adjust_button_right_image
1953
1954 Globals.profiles_choice_of_profile_line_type.trace_add('write',
1955     Profile_functions.trace_profileLineType)
1956
1957 Globals.tab4_canvas.pack(expand=True, fill=BOTH)
1958 ###### DVH tab 5
1959 ###### DVH tab 5
1960
1961 Globals.tab5_canvas.config(bg='#ffffff', bd = 0, relief=FLAT,
1962     highlightthickness=0)
1963
1964 DVH_explain_text = tk.Text(Globals.tab5_canvas, height=4, width=140)
1965 DVH_explain_text.insert(INSERT, "\n"
1966     SliceThickness i plan m v re ['1','1'], ['2','2'] eller ['3','3'],
1967     Filmen m legges i xy, xz eller yz planet (lage figur?), Filmen m
1968     scannes \n\
1969 parallelt med retningene i skanneren (programmet vil anta dette), man m
1970     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?). \
1960 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 , " )
1962 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)
1964 Globals.tab5_canvas.grid_rowconfigure(0, weight=0)
DVH.explain_text.config(state=DISABLED, font=('calibri', '11'), bg ='#
E5f9ff', relief=FLAT)

1966 DVH_upload_film_frame = tk.Frame(Globals.tab5_canvas)
1968 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)
1970 Globals.tab5_canvas.grid_rowconfigure(1, weight=0)
DVH_upload_film_frame.config(bg = '#ffffff')

1972 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)
1974 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

1976 DVH_upload_doseplan_frame = tk.Frame(Globals.tab5_canvas)
1978 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)
1980 Globals.tab5_canvas.grid_rowconfigure(3, weight=0)
DVH_upload_film_frame.config(bg = '#ffffff')

1982 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)
1984 Globals.DVH_upload_button_doseplan.pack(expand=True, fill=BOTH)
1986 Globals.DVH_upload_button_doseplan.config(bg='#ffffff',
activebackground='#ffffff', activeforeground='#ffffff',
highlightthickness=0)
Globals.DVH_upload_button_doseplan.image = Globals.
profiles_add_doseplan_button_image

1988 DVH_upload_rtplan_frame = tk.Frame(Globals.tab5_canvas)
1990 DVH_upload_rtplan_frame.grid(row=3, column=0, padx=(160,0), pady=(10,0) ,
sticky=N)
1992 Globals.tab5_canvas.grid_columnconfigure(10, weight=0)
1994 Globals.tab5_canvas.grid_rowconfigure(10, weight=0)
DVH_upload_rtplan_frame.config(bg='#ffffff')
1996

```

```

1998     Globals . DVH_upload_button_rtplan = tk . Button ( DVH_upload_rtplan_frame , text
1999         = 'Browse' , image=profiles_add_rtplan_button_image , \
2000             cursor='hand2' , font=( 'calibri' , '14' ) , relief=FLAT , state=DISABLED ,
2001                 command=DVH_functions . UploadRTplan )
2002     Globals . DVH_upload_button_rtplan . pack ( expand=True , fill=BOTH )
2003     Globals . DVH_upload_button_rtplan . configure ( bg='#ffffff' , activebackground=
2004         '#ffffff' , activeforeground='#ffffff' , highlightthickness=0 )
2005     Globals . DVH_upload_button_rtplan . image=profiles_add_rtplan_button_image
2006
2007     Globals . DVH_film_orientation_menu = OptionMenu ( Globals . tab5_canvas ,
2008         Globals . DVH_film_orientation , 'Axial' , 'Coronal' , 'Sagittal' )
2009     Globals . DVH_film_orientation_menu . grid ( row=1 , column=0 , sticky=N+S , padx
2010         =(60,0) )
2011     Globals . tab5_canvas . grid_columnconfigure ( 2 , weight=0 )
2012     Globals . tab5_canvas . grid_rowconfigure ( 2 , weight=0 )
2013     Globals . DVH_film_orientation_menu . config ( bg = '#ffffff' , width=15 , relief=
2014         FLAT )
2015
2016     film_orientation_menu_text = tk . Text ( Globals . tab5_canvas , width=14 , height
2017         =1 )
2018     film_orientation_menu_text . insert ( INSERT , "Film orientation:" )
2019     film_orientation_menu_text . config ( state=DISABLED , font=( 'calibri' , '10' ) ,
2020         bd = 0 , relief=FLAT )
2021     film_orientation_menu_text . grid ( row=1 , column=0 , sticky=N+S+W , padx=(30,0)
2022         , pady=(5,0) )
2023     Globals . tab5_canvas . grid_columnconfigure ( 3 , weight=0 )
2024     Globals . tab5_canvas . grid_rowconfigure ( 3 , weight=0 )
2025
2026     DVH_film_orientation_help_frame = tk . Frame ( Globals . tab5_canvas )
2027     DVH_film_orientation_help_frame . grid ( row=1 , column=0 , sticky=N+S+E , padx
2028         =(0,40) )
2029     Globals . tab5_canvas . grid_columnconfigure ( 6 , weight=0 )
2030     Globals . tab5_canvas . grid_rowconfigure ( 6 , weight=0 )
2031     DVH_film_orientation_help_frame . configure ( bg='#ffffff' )
2032
2033     DVH_help_button_orientation = tk . Button ( DVH_film_orientation_help_frame , text
2034         = 'help' , image=Globals . help_button , \
2035             cursor='hand2' , font=( 'calibri' , '14' ) , relief=FLAT , state=ACTIVE ,
2036                 command=DVH_functions . help_showPlanes )
2037     DVH_help_button_orientation . pack ( expand=True , fill=BOTH )
2038     DVH_help_button_orientation . configure ( bg='#ffffff' , activebackground='#
2039         ffffff' , activeforeground='#ffffff' , highlightthickness=0 )
2040     DVH_help_button_orientation . image=Globals . help_button
2041
2042     DVH_film_factor = tk . Text ( Globals . tab5_canvas , width=20 , height=2 )
2043     DVH_film_factor . insert ( INSERT , "Film factor \n(number of fractions):" )
2044     DVH_film_factor . config ( state=DISABLED , font=( 'calibri' , '10' ) , bd = 0 ,
2045         relief=FLAT )
2046     DVH_film_factor . grid ( row=2 , column=0 , sticky=N+S+W , padx=(30,0) , pady
2047         =(5,0) )
2048     Globals . tab5_canvas . grid_columnconfigure ( 30 , weight=0 )
2049     Globals . tab5_canvas . grid_rowconfigure ( 30 , weight=0 )
2050
2051     Globals . DVH_film_factor_input = tk . Text ( Globals . tab5_canvas , width=8 ,
2052         height=1 )
2053     Globals . DVH_film_factor_input . grid ( row=2 , column=0 , sticky=E , padx=(0,160)
2054         , pady=(5,0) )

```

```

2038     Globals.DVH_film_factor_input.insert(INSERT, " ")
2039     Globals.DVH_film_factor_input.config(state=NORMAL, font=(‘calibri’, ‘10’),
2040                                         bd = 2, bg=‘#ffffff’)
2041     Globals.tab5_canvas.grid_columnconfigure(31, weight=0)
2042     Globals.tab5_canvas.grid_rowconfigure(31, weight=0)
2043     ””
2044 DVH_resetAll_frame = tk.Frame(Globals.tab5_canvas)
2045 DVH_resetAll_frame.grid(row=15,column=0, padx=(0,0), pady=(0,0), sticky=S)
2046 Globals.tab5_canvas.grid_columnconfigure(5, weight=0)
2047 Globals.tab5_canvas.grid_rowconfigure(5, weight=0)
2048 profiles_resetAll_frame.config(bg=‘#ffffff’)
2049
2050 DVH_resetAll_button = tk.Button(DVH_resetAll_frame, text=‘Reset’, image=
2051     dose_response_clear_all_button_image, \
2052     cursor=‘hand2’, font=(‘calibri’, ‘14’), relief=FLAT, state=ACTIVE,
2053     command=DVH_functions.clearAll)
2054 DVH_resetAll_button.pack(expand=True, fill=BOTH)
2055 DVH_resetAll_button.configure(bg=‘#ffffff’, activebackground=‘#ffffff’,
2056                               activeforeground=‘#ffffff’, highlightthickness=0)
2057 DVH_resetAll_button.image = dose_response_clear_all_button_image
2058 ””
2059 Globals.tab5_canvas.pack(expand=True, fill=BOTH)
2060
2061 ##### End statements
2062 ##### Globals.tab_parent.place(relwidth=1, relheight=0.9, relx=0, rely=0.15)
2063 Globals.form.mainloop()

```

FIDORA/notebook.py

A.2 Globals.py

```

1000 import tkinter as tk
1001 from tkinter import ttk, StringVar, IntVar, Scrollbar, RIGHT, Y, \
1002     HORIZONTAL, E, W, N, S, BOTH, Frame, Canvas, LEFT, FLAT, INSERT,
1003     DISABLED, ALL, X, BOTTOM, \
1004     DoubleVar, PanedWindow, RAISED, TOP, Radiobutton, CENTER, BooleanVar
1005 import numpy as np
1006 import matplotlib.pyplot as plt
1007 from matplotlib.figure import Figure
1008 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
1009
1010 global upload_button_image
1011 global dose_response_dose_border
1012 global save_button
1013 global help_button
1014 global done_button_image
1015 global profiles_add_doseplan_button_image
1016 global profiles_add_doseplans_button_image
1017 global adjust_button_left_image
1018 global adjust_button_right_image
1019 global adjust_button_up_image
1020 global adjust_button_down_image

```

```

1022
1023 global form
1024 form = tk.Tk()
1025
1026 #Main—window
1027 over_all_frame = tk.Frame(form, bd=0, relief=FLAT)
1028 over_all_canvas = Canvas(over_all_frame)
1029
1030 xscrollbar = Scrollbar(over_all_frame, orient=HORIZONTAL, command=
1031     over_all_canvas.xview)
1032 yscrollbar = Scrollbar(over_all_frame, command=over_all_canvas.yview)
1033
1034 scroll_frame = ttk.Frame(over_all_canvas)
1035 scroll_frame.bind("<Configure>", lambda e: over_all_canvas.configure(
1036     scrollregion=over_all_canvas.bbox('all')))
1037
1038 over_all_canvas.create_window((0,0), window=scroll_frame, anchor='nw')
1039 over_all_canvas.configure(xscrollcommand=xscrollbar.set, yscrollcommand=
1040     yscrollbar.set)
1041
1042 over_all_frame.config(highlightthickness=0, bg='#fffff')
1043 over_all_canvas.config(highlightthickness=0, bg='#fffff')
1044 over_all_frame.pack(expand=True, fill=BOTH)
1045 over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
1046 over_all_frame.grid_columnconfigure(0, weight=1)
1047 over_all_frame.grid_rowconfigure(0, weight=1)
1048 xscrollbar.grid(row=1, column=0, sticky=E+W)
1049 over_all_frame.grid_columnconfigure(1, weight=0)
1050 over_all_frame.grid_rowconfigure(1, weight=0)
1051 yscrollbar.grid(row=0, column=1, sticky=N+S)
1052 over_all_frame.grid_columnconfigure(2, weight=0)
1053 over_all_frame.grid_rowconfigure(2, weight=0)
1054
1055 global tab_parent
1056 tab_parent = ttk.Notebook(scroll_frame)
1057 tab_parent.borderWidth=0
1058 tab_parent.grid(row=1, column=0, sticky=E+W+N+S, pady=(0,0), padx =(0,0))
1059
1060 global intro_tab
1061 intro_tab = ttk.Frame(tab_parent)
1062 intro_tab.config(relief=FLAT)
1063 global tab1
1064 tab1 = ttk.Frame(tab_parent)
1065 global tab2
1066 tab2 = ttk.Frame(tab_parent)
1067 global tab3
1068 tab3 = ttk.Frame(tab_parent)
1069 global tab4
1070 tab4 = ttk.Frame(tab_parent)
1071 global tab5
1072 tab5 = ttk.Frame(tab_parent)
1073
1074 global tab1_canvas
1075 tab1_canvas = tk.Canvas(tab1)
1076 global tab2_canvas
1077 tab2_canvas = tk.Canvas(tab2)

```

```

1076 global tab3_canvas
tab3_canvas = tk.Canvas(tab3)
1078 global tab4_canvas
tab4_canvas = tk.Canvas(tab4)
1080 global tab5_canvas
tab5_canvas= tk.Canvas(tab5)

1082 ##### CoMet related #####
global CoMet_progressbar
1084 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))
1086 tab1_canvas.grid_columnconfigure(12, weight=0)
tab1_canvas.grid_rowconfigure(12, weight=0)
1088 CoMet_progressbar["maximum"] = 100
CoMet_progressbar["value"] = 0
1090
global CoMet_progressbar_counter
1092 CoMet_progressbar_counter = 0

1094 global CoMet_progressbar_check_file
CoMet_progressbar_check_file = True
1096
global CoMet_progressbar_check_folder
1098 CoMet_progressbar_check_folder = True

1100 global CoMet_progressbar_text
CoMet_progressbar.text = tk.Text(tab1_canvas, height=1, width=5)
1102 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%")
1106 CoMet_progressbar.text.config(state=DISABLED, bd=0, relief=FLAT, bg="#
    ffffff",font=('calibri', '10', 'bold'))

1108 global CoMet_dpi
CoMet_dpi = StringVar(tab1)
1110 CoMet_dpi.set("127")

1112 global CoMet_saveAs
CoMet_saveAs = tk.StringVar(tab1)
1114 CoMet_saveAs.set(".dem")

1116 global CoMet_uploaded_filename
CoMet_uploaded_filename=StringVar(tab1)
1118 CoMet_uploaded_filename.set("Error!")

1120 global CoMet_export_folder
CoMet_export_folder=StringVar(tab1)
1122 CoMet_export_folder.set("Error!")

1124 global CoMet_image_to_canvas

1126 global CoMet_correcte_image_filename_box

```

```

1128 global CoMet_corrected_image_filename
CoMet_corrected_image_filename=StringVar(tab1)
CoMet_corrected_image_filename.set("Error!")
1130
1132 global CoMet_patientName
CoMet_patientName=StringVar(tab1)
CoMet_patientName.set("Error!")
1134
1136 global CoMet_correctedImage
CoMet_correctedImage=None
1138
1140 global CoMet_border_1_label
CoMet_border_1_label = tk.Label(tab1_canvas)
1142
1144 global CoMet_border_2_label
CoMet_border_2_label = tk.Label(tab1_canvas)
1146
1148 global CoMet_border_3_label
CoMet_border_3_label = tk.Label(tab1_canvas)
1150
1152 global CoMet_save_button_frame_1
CoMet_save_button_frame_1 = tk.Frame(tab1_canvas)
1154
1156 global CoMet_save_button_1
CoMet_save_button_1 = tk.Button(CoMet_save_button_frame_1)
1158
1160 global CoMet_save_filename
CoMet_save_filename = tk.Text(CoMet_border_3_label, height=1, width=30)
1162
1164 global CoMet_print_corrected_image
CoMet_print_corrected_image = tk.Canvas(tab1_canvas)
1166 ##### Dose response related
##### tab2_files_frame = tk.Frame(tab2_canvas)
1168 tab2_files_frame.config(relief=FLAT, bg='#ffffff', highlightthickness=0,
height=200, width=450)
#tab2_files_frame.grid_propagate(0)
1170
1172 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)
1174 scroll = ttk.Scrollbar(tab2_files_frame, command=tab2_scroll_canvas.yview)
1176
scrollable_frame= tk.Frame(tab2_scroll_canvas)
1178 scrollable_frame.bind("<Configure>", lambda e: tab2_scroll_canvas.
configure(scrollregion=tab2_scroll_canvas.bbox('all'))))

```

```

1180 tab2_scroll_canvas.create_window((0,0), window=scrollable_frame, anchor=?  
    nw)  
tab2_scroll_canvas.configure(yscrollcommand=scroll.set)  
1182  
  
1184 global tab2_canvas_files  
tab2_canvas_files = tk.Canvas(scrollable_frame)  
1186 tab2_canvas_files.config(relief=FLAT, bg='#ffffff', highlightthickness=0,  
    bd=0)  
tab2_canvas_files.pack(fill=BOTH, expand=True)  
1188  
tab2_file_frame.grid(row=2, column=4, columnspan=1, rowspan=3, sticky=N)  
1190 tab2_canvas.grid_columnconfigure(1, weight=0)  
tab2_canvas.grid_rowconfigure(1, weight=0)  
1192 tab2_scroll_canvas.pack(side=LEFT, fill=BOTH, expand=True)  
scroll.pack(side=RIGHT, fill=Y)  
1194  
  
1196 global dose_response_save_calibration_button  
  
1198 global doseResponse_dpi  
doseResponse_dpi=StringVar()  
1200 doseResponse_dpi.set("127")  
  
1202  
global dose_response_var1  
1204 dose_response_var1= IntVar()  
dose_response_var1.set(1)  
1206  
global dose_response_var2  
1208 dose_response_var2 = IntVar()  
dose_response_var2.set(1)  
1210  
global dose_response_var3  
1212 dose_response_var3 = IntVar()  
dose_response_var3.set(1)  
1214  
global dose_response_uploaded_filenames  
1216 dose_response_uploaded_filenames = np.array([])  
  
1218 global dose_response_new_window_row_count  
dose_response_new_window_row_count = 4  
1220  
global dose_response_new_window_weight_count  
1222 dose_response_new_window_weight_count = 4  
  
1224 global avg_red_vector  
avg_red_vector = []  
1226  
global avg_green_vector  
1228 avg_green_vector = []  
  
1230 global avg_blue_vector  
avg_blue_vector = []  
1232  
1234 global dose_response_files_row_count

```

```

1236 dose_response_files_row_count = 2
1237
1238 global dose_response_files_weightcount
dose_response_files_weightcount = 8

1240 global dose_response_inOrOut
dose_response_inOrOut = True

1242
1243 global dose_response_delete_buttons
dose_response_delete_buttons = []

1246
1247 global dose_response_red_list
dose_response_red_list = []

1250 global dose_response_green_list
dose_response_green_list = []

1252
1253 global dose_response_blue_list
dose_response_blue_list = []

1256 global dose_response_dose_list
dose_response_dose_list = []

1258
1259 global popt_red
popt_red = np.zeros(3)

1262 global dose_response_batch_number
dose_response_batch_number = "—"

1264
1265 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))
1266 tab2_canvas.grid_columnconfigure(8, weight=0)
tab2_canvas.grid_rowconfigure(8, weight=0)
1267 dose_response_equation_frame.config(bg='#E5f9ff', relief=FLAT,
    highlightthickness=0, width=400, height=200)
dose_response_equation_frame.grid_propagate(0)

1272

1274 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))
1275 tab2_canvas.grid_columnconfigure(9, weight=0)
tab2_canvas.grid_rowconfigure(9, weight=0)
1276 dose_response_plot_frame.config(bg="#ffffff", relief=FLAT,
    highlightthickness=0, height=350, width=500)
dose_response_plot_frame.grid_propagate(0)

1278
1279 dose_response_fig = Figure(figsize=(5,3))
dose_response_a = dose_response_fig.add_subplot(111, ylim=(0,40000), xlim=(0,500))
1280 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))
1286 dose_response_a.set_title("Dose-response", fontsize=12)
dose_response_a.set_ylabel("Pixel value", fontsize=12)
1288 dose_response_a.set_xlabel("Dose", fontsize=12)
dose_response_fig.tight_layout()
1290
1291 global dose_response_sd_list_red
1292 dose_response_sd_list_red = []
1293
1294 global dose_response_sd_list_green
dose_response_sd_list_green = []
1295
1296 global dose_response_sd_list_blue
1298 dose_response_sd_list_blue = []
1299
1300 global dose_response_sd_avg_red
dose_response_sd_avg_red = DoubleVar()
1302 dose_response_sd_avg_red.set(0)
1303
1304 global dose_response_sd_avg_green
dose_response_sd_avg_green = DoubleVar()
1306 dose_response_sd_avg_green.set(0)
1307
1308 global dose_response_sd_avg_blue
dose_response_sd_avg_blue = DoubleVar()
1310 dose_response_sd_avg_blue.set(0)
1311
1312 global dose_response_sd_min_red
dose_response_sd_min_red = DoubleVar()
1314 dose_response_sd_min_red.set(0)
1315
1316 global dose_response_sd_min_red_dose
dose_response_sd_min_red_dose = StringVar()
1318 dose_response_sd_min_red_dose.set('—')
1319
1320 global dose_response_sd_min_green
dose_response_sd_min_green = DoubleVar()
1322 dose_response_sd_min_green.set(0)
1323
1324 global dose_response_sd_min_green_dose
dose_response_sd_min_green_dose = StringVar()
1326 dose_response_sd_min_green_dose.set('—')
1327
1328 global dose_response_sd_min_blue
dose_response_sd_min_blue = DoubleVar()
1330 dose_response_sd_min_blue.set(0)
1331
1332 global dose_response_sd_min_blue_dose
dose_response_sd_min_blue_dose = StringVar()
1334 dose_response_sd_min_blue_dose.set('—')
1335
1336 global dose_response_sd_max_red
dose_response_sd_max_red = DoubleVar()
1338 dose_response_sd_max_red.set(0)
1339
1340 global dose_response_sd_max_red_dose

```

```

1342 dose_response_sd_max_red_dose = StringVar()
dose_response_sd_max_red_dose.set('—')

1344 global dose_response_sd_max_green
dose_response_sd_max_green = DoubleVar()
dose_response_sd_max_green.set(0)

1348 global dose_response_sd_max_green_dose
dose_response_sd_max_green_dose = StringVar()
dose_response_sd_max_green_dose.set('—')

1352 global dose_response_sd_max_blue
dose_response_sd_max_blue = DoubleVar()
dose_response_sd_max_blue.set(0)

1356 global dose_response_sd_max_blue_dose
dose_response_sd_max_blue_dose = StringVar()
dose_response_sd_max_blue_dose.set('—')

1360 ###### Map dose related
#####

1362

1364 global map_dose_film_dataset
map_dose_film_dataset=StringVar(tab3)
map_dose_film_dataset.set("Error!")

1366 global map_dose_isocenter_map_x_coord_scaled
map_dose_isocenter_map_x_coord_scaled = []

1370 global map_dose_isocenter_map_x_coord_unscaled
map_dose_isocenter_map_x_coord_unscaled = []

1374 global map_dose_isocenter_map_y_coord_scaled
map_dose_isocenter_map_y_coord_scaled = []

1376 global map_dose_isocenter_map_y_coord_unscaled
map_dose_isocenter_map_y_coord_unscaled = []

1378 global map_dose_icocenter_film #Oppgitt ved [<,v] = [bortover, nedover]

1380 global map_dose_film_batch
map_dose_film_batch = IntVar()
map_dose_film_batch.set(0)

1382 global map_dose_ROI_x_start
map_dose_ROI_x_start = IntVar()
map_dose_ROI_x_start.set(0)

1386 global map_dose_ROI_y_start
map_dose_ROI_y_start = IntVar()
map_dose_ROI_y_start.set(0)

1390 global map_dose_ROI_x_end
map_dose_ROI_x_end = IntVar()
map_dose_ROI_x_end.set(0)

1392 global map_dose_ROI_y_end
map_dose_ROI_y_end = IntVar()
map_dose_ROI_y_end.set(0)

1394 global map_dose_ROI_z_start
map_dose_ROI_z_start = IntVar()
map_dose_ROI_z_start.set(0)

1396 global map_dose_ROI_z_end
map_dose_ROI_z_end = IntVar()
map_dose_ROI_z_end.set(0)

```

```

1446 profiles_a.set_ylabel("Dose (Gy)", fontsize=12)
1447 profiles_a.set_xlabel("Distance (mm)", fontsize=12)
1448 profiles_fig.tight_layout()

1450
1451 global profiles_showPlanes_image
1452 global profiles_showDirections_image

1454 global profiles_depth
1455 global profiles_depth_float

1456 global profiles_film_factor_input

1458 global profiles_mark_isocenter_button_image
1459 global profiles_mark_ROI_button_image
1460 global profiles_mark_point_button_image

1462 global profiles_iscocenter_coords
1463 profiles_iscocenter_coords = []

1466 #Given from top left corner [right , down]
1467 global profiles_film_isocenter
1468 global profiles_film_reference_point

1470 global profiles_distance_isocenter_ROI
1471 profiles_distance_isocenter_ROI = []
1472 global profiles_distance_reference_point_ROI
1473 profiles_distance_reference_point_ROI = []

1474 global profiles_mark_isocenter_up_down_line
1475 profiles_mark_isocenter_up_down_line = []
1476 global profiles_mark_isocenter_right_left_line
1477 profiles_mark_isocenter_right_left_line = []
1478 global profiles_mark_isocenter_oval
1479 profiles_mark_isocenter_oval = []
1480 global profiles_mark_ROI_rectangle
1481 profiles_mark_ROI_rectangle = []

1484 global profiles_mark_reference_point_oval
1485 profiles_mark_reference_point_oval = []

1486 global profiles_ROI_coords
1487 profiles_ROI_coords = []

1490 global profiles_done_button
1491 profiles_done_button = None
1492 global profiles_done_button_reference_point
1493 profiles_done_button_reference_point = None

1494 global profiles_isocenter_check
1495 profiles_isocenter_check=False
1496 global profiles_reference_point_check
1497 profiles_reference_point_check = False

1500 global profiles_ROI_check
1501 profiles_ROI_check = False
1502 global profiles_ROI_reference_point_check

```

```

1504 profiles_ROI_reference_point_check = False
1505
1506 global profiles_film_batch
1507 profiles_film_batch = IntVar()
1508 profiles_film_batch.set(0)
1509
1510 global profiles_popt_red
1511 profiles_popt_red = np.zeros(3)
1512
1513 #global profiles_film_window
1514 #global profiles_film_window_open
1515 #profiles_film_window_open = False
1516
1517 global profiles_upload_button_doseplan
1518 global profiles_upload_button_film
1519 global profiles_upload_button_rtplan
1520
1521 global profiles_dataset_doseplan
1522 profiles_dataset_doseplan = None
1523 global profiles_dataset_rtplan
1524
1525 global profiles_test_if_added_doseplan
1526 global profiles_test_if_added_rtplan
1527 profiles_test_if_added_doseplan = False
1528 profiles_test_if_added_rtplan = False
1529
1530 global profiles_isocenter_mm
1531
1532 global profiles_dose_scaling_doseplan
1533 profiles_dose_scaling_doseplan = []
1534
1535 global profiles_max_dose_film
1536
1537 global profiles_choose_profile_canvas
1538 profiles_choose_profile_canvas = tk.Canvas(profiles_view_film_doseplan_ROI
1539 )
1540 profiles_choose_profile_canvas.pack()
1541 profiles_choose_profile_canvas.config(bg='#ffffff', relief=FLAT,
1542 highlightthickness=0)
1543 global profiles_choice_of_profile_line_type
1544 profiles_choice_of_profile_line_type = StringVar()
1545 profiles_choice_of_profile_line_type.set("h")
1546
1547 profiles_choose_profile_type_text = tk.Text(profiles_choose_profile_canvas
1548 , height=1)
1549 profiles_choose_profile_type_text.insert(INSERT, "How to draw the profile:
1550 ")
1551 profiles_choose_profile_type_text.pack(side=TOP)
1552 profiles_choose_profile_type_text.config(bg='#ffffff', relief=FLAT, \
1553 highlightthickness=0, state=DISABLED, font=(‘calibri’, ‘11’))
1554
1555 Radiobutton(profiles_choose_profile_canvas, text="Horizontal", variable=
1556 profiles_choice_of_profile_line_type, \

```

```

1556     value="h", bg="#ffffff", cursor='hand2').pack(side=LEFT)
1557 Radiobutton(profiles_choose_profile_canvas, text="Vertical", \
1558     variable=profiles_choice_of_profile_line_type, value='v', bg='#ffffff' \
1559     , cursor='hand2').pack(side=LEFT)
1560 Radiobutton(profiles_choose_profile_canvas, text="Draw", \
1561     variable=profiles_choice_of_profile_line_type, value="d", bg='#ffffff' \
1562     , cursor='hand2').pack(side=LEFT)
1563 profiles_adjust_ROI_text = tk.Text(profiles_choose_profile_canvas, width \
1564     =20, height=1)
1565 profiles_adjust_ROI_text.insert(INSERT, "Adjust ROI in film: ")
1566 profiles_adjust_ROI_text.config(state=DISABLED, font=('calibri', '11'), bg \
1567     ='#ffffff', relief=FLAT, bd=0)
1568 profiles_adjust_ROI_text.pack(side=LEFT, padx=(70,0))

1569
1570 global profiles_redefine_film_ROI_frame
1571 profiles_redefine_film_ROI_frame = tk.Frame(profiles_choose_profile_canvas \
1572     )
1573 profiles_redefine_film_ROI_frame.pack(side=LEFT, padx=(0,100))
1574 profiles_redefine_film_ROI_frame.config(bg='#ffffff')
1575 global profiles_adjust_button_left
1576 global profiles_adjust_button_right
1577 global profiles_adjust_button_down
1578 global profiles_adjust_button_up

1579
1580 global profiles_film_panedwindow
1581 profiles_film_panedwindow = PanedWindow(profiles_view_film_doseplan_ROI, \
1582     orient='vertical')
1583 profiles_film_panedwindow.pack()
1584 profiles_film_panedwindow.configure(sashrelief = RAISED, showhandle=True)

1585
1586 #global profiles_film_tab_parent
1587 #profiles_film_tab_parent = ttk.Notebook(profiles_film_notebook_canvas)
1588 #profiles_film_tab_parent.borderWidth=0
1589 #profiles_film_tab_parent.pack()

1590 #global profiles_film_tab_image
1591 #profiles_film_tab_image = tk.Frame(profiles_film_tab_parent)
1592 #profiles_film_tab_image.config(relief=FLAT)

1593
1594 #global profiles_film_tab_dose
1595 #profiles_film_tab_dose = tk.Frame(profiles_film_tab_parent)
1596 #profiles_film_tab_dose.config(relief=FLAT, padding=[0,0,0,0])

1597
1598 #profiles_film_tab_parent.add(profiles_film_tab_image, text='Scanned film \
1599     ')
1600 #profiles_film_tab_parent.add(profiles_film_tab_dose, text='Dose on film')

1601 global profiles_scanned_image_text_image
1602 global profiles_film_dose_map_text_image
1603 global profiles_doseplan_text_image

```

```
1606 global doseplan_write_image
1607 global film_dose_write_image
1608 global film_write_image
1609 global doseplan_write_image_width
1610 global doseplan_write_image_height
1611 global doseplan_write_image_var_x
1612 doseplan_write_image_var_x= 0
1613 global doseplan_write_image_var_y
1614 doseplan_write_image_var_y = 0
1615 global profiles_coordinate_in_dataset
1616 profiles_coordinate_in_dataset = 0

1618 global profiles_first_time_in_drawProfiles
profiles_first_time_in_drawProfiles = True
1620
1622 global new_window_factor_textbox
1624
1625 global profiles_doseplan_lateral_displacement
1626 global profiles_doseplan_vertical_displacement
1627 global profiles_doseplan_longitudinal_displacement
1628 global profiles_doseplan_patient_position

1629 global profiles_reference_point_in_doseplan
1630 global profiles_input_lateral_displacement
1631 global profiles_input_longitudinal_displacement
1632 global profiles_input_vertical_displacement

1633 global profiles_isocenter_or_reference_point
1634
1635 global profiles_lateral
1636 global profiles_vertical
1637 global profiles_longitudinal

1638 global profiles_number_of_doseplans
1639 profiles_number_of_doseplans = 0
1640 global profiles_number_of_doseplans_row_count
1641 profiles_number_of_doseplans_row_count = 4
1642 global profiles_doseplans_grid_config_count
1643 profiles_doseplans_grid_config_count = 6
1644 global profiles_doseplans_filenames
1645 profiles_doseplans_filenames = []
1646 global profiles_doseplans_factor_text
1647 profiles_doseplans_factor_text = []
1648 global profiles_doseplans_factor_input
1649 profiles_doseplans_factor_input = []

1650
1651 global profiles_doseplan_dataset_ROI_several
1652 profiles_doseplan_dataset_ROI_several = []
1653 global profiles_several_img
1654 profiles_several_img = []

1655 global profiles_film_factor
1656
1657 global profiles_lines
```

```
1662 profiles_lines = []
1664 global end_point
end_point = None
1666
1668 global profiles_line_coords_film
global profiles_line_coords_doseplan
1670
1672 global profiles_dataset_film_variable_draw
global profiles_dataset_doesplan_variable_draw
1674 global max_dose_doseplan
1676 global profiles_slice_offset
global profiles_offset
1678 ##### DVH related
#####
1680 global DVH_film_orientation
DVH_film_orientation = StringVar()
1682 DVH_film_orientation.set('-')
1684 global DVH_doseplans_scroll_frame
1686
1688 global DVH_number_of_doseplans
DVH_number_of_doseplans = 0
global DVH_number_of_doseplans_row_count
1690 DVH_number_of_doseplans_row_count = 4
global DVH_doseplans_grid_config_count
1692 DVH_doseplans_grid_config_count = 6
global DVH_doseplans_filenames
1694 DVH_doseplans_filenames = []
global DVH_doseplans_factor_text
1696 DVH_doseplans_factor_text = []
global DVH_doseplans_factor_input
1698 DVH_doseplans_factor_input = []
1700
1702 global DVH_doseplan_dataset_ROI_several
DVH_doseplan_dataset_ROI_several = []
1704 global DVH_several_img
DVH_several_img = []
1706 global profiles_film_factor
1708
1710 #global profiles_lines
#profiles_lines = []
1712
1714 #global end_point
#end_point = None
1716 #global profiles_line_coords_film
#global profiles_line_coords_doseplan
```

```

1718 global DVH_film_orientation_menu
1720 global DVH_film_factor_input
1722 global DVH_film_factor
1724
1726 global DVH_film_dataset
1728 global DVH_film_dataset_red_channel
1730 global DVH_film_dataset_ROI
1732 global DVH_film_dataset_ROI_red_channel
1734 global DVH_doseplan_dataset_ROI
1736 global DVH_film_dataset_ROI_red_channel_dose
1738
1740 global DVH_max_dose_film
1742 global DVH_max_dose_doseplan
1744
1746 global DVH_view_film_doseplan_ROI
1748 DVH_view_film_doseplan_ROI = tk.Canvas(tab5_canvas)
1750 DVH_view_film_doseplan_ROI.grid(row=2, column=3, rowspan=25, sticky=E+W+N,
1752     pady=(0,5), padx=(5,10))
1754 tab5_canvas.grid_columnconfigure(11, weight=0)
1756 tab5_canvas.grid_rowconfigure(11, weight=0)
1758 DVH_view_film_doseplan_ROI.config(bg='#E5f9ff', relief=FLAT,
1760     highlightthickness=0)
1762 """
1764 global DVH_plot_canvas
1766 DVH_plot_canvas = tk.Canvas(tab4_canvas)
1768 DVH_plot_canvas.grid(row=3, column=0, rowspan=10, columnspan=2, sticky=N+E
1770     +W, pady=(0,5), padx=(5,10))
1772 tab5_canvas.grid_columnconfigure(4, weight=0)
1774 tab5_canvas.grid_rowconfigure(4, weight=0)
1776 DVH_plot_canvas.config(bg='#E5f9ff', relief=FLAT, highlightthickness=0)
1778
1780 DVH_fig = Figure(figsize=(5,3))
1782 DVH_a = profiles_fig.add_subplot(111, ylim=(0,40000), xlim=(0,500))
1784 DVH_plot_canvas = FigureCanvasTkAgg(profiles_fig, master=
1786     profile_plot_canvas)
1788 DVH_plot_canvas.get_tk_widget().grid(row=0, column=0, columnspan=4, sticky=N
1790     +E+W, padx=(5,0), pady=(0,0))
1792 DVH_a.set_title("Profiles", fontsize=12)
1794 DVH_a.set_ylabel("Pixel value", fontsize=12)
1796 DVH_a.set_xlabel("Distance (mm)", fontsize=12)
1798 DVH_fig.tight_layout()
1800
1802 global DVH_showPlanes_image
1804 global DVH_showDirections_image
1806
1808 global DVH_depth
1810 global DVH_depth_float

```

```
1770 #global DVH_mark_isocenter_button_image
1771 #global DVH_mark_ROI_button_image
1772 #global DVH_mark_point_button_image
1773 """
1774 global DVH_iscoenter_coords
1775 DVH_iscoenter_coords = []
1776
1777 #Given from top left corner [right , down]
1778 global DVH_film_isocenter
1779
1780 global DVH_film_reference_point
1781
1782 global DVH_distance_isocenter_ROI
1783 DVH_distance_isocenter_ROI = []
1784
1785 global DVH_distance_reference_point_ROI
1786 DVH_distance_reference_point_ROI = []
1787
1788 global DVH_mark_isocenter_up_down_line
1789 DVH_mark_isocenter_up_down_line = []
1790 global DVH_mark_isocenter_right_left_line
1791 DVH_mark_isocenter_right_left_line = []
1792
1793 global DVH_mark_isocenter_oval
1794 DVH_mark_isocenter_oval = []
1795
1796 global DVH_mark_ROI_rectangle
1797 DVH_mark_ROI_rectangle = []
1798
1799 global DVH_mark_reference_point_oval
1800 DVH_mark_reference_point_oval = []
1801
1802 global DVH_ROI_coords
1803 DVH_ROI_coords = []
1804
1805 global DVH_film_variable_ROI_coords
1806
1807 global DVH_done_button
1808 DVH_done_button = None
1809
1810
1811 global DVH_done_button_reference_point
1812 DVH_done_button_reference_point = None
1813
1814 global DVH_isocenter_check
1815 DVH_isocenter_check=False
1816
1817 global DVH_reference_point_check
1818 DVH_reference_point_check = False
1819
1820 global DVH_ROI_check
1821 DVH_ROI_check = False
1822
1823 global DVH_ROI_reference_point_check
1824 DVH_ROI_reference_point_check = False
1825
1826 global DVH_film_batch
```

```

1828 DVH_film_batch = IntVar()
1829 DVH_film_batch.set(0)

1830 global DVH_popt_red
1831 DVH_popt_red = np.zeros(3)

1832 global DVH_upload_button_doseplan
1833
1834 global DVH_upload_button_film
1835
1836 global DVH_upload_button_rtplan
1837
1838 global DVH_dataset_doseplan
1839 global DVH_dataset_rtplan

1840 global DVH_test_if_added_doseplan
1841 global DVH_test_if_added_rtplan
1842 DVH_test_if_added_doseplan = False
1843 DVH_test_if_added_rtplan = False

1844 global DVH_isocenter_mm
1845
1846
1847 global DVH_dose_scaling_doseplan
1848 """
1849 global DVH_max_dose_film

1850 #####
1851 DVH_choose_profile_canvas = tk.Canvas(DVH_view_film_doseplan_ROI)
1852 DVH_choose_profile_canvas.pack()
1853 DVH_choose_profile_canvas.config(bg='#ffffff', relief=FLAT,
1854     highlightthickness=0)
1855 global DVH_choice_of_profile_line_type
1856 DVH_choice_of_profile_line_type = StringVar()
1857 DVH_choice_of_profile_line_type.set("h")

1858 DVH_choose_profile_type_text = tk.Text(DVH_choose_profile_canvas, height
1859     =1)
1860 DVH_choose_profile_type_text.insert(INSERT, "How to draw the profile:")
1861 DVH_choose_profile_type_text.pack(side=TOP)
1862 DVH_choose_profile_type_text.config(bg='#ffffff', relief=FLAT, \
1863     highlightthickness=0, state=DISABLED, font=('calibri', '11'))
1864 Radiobutton(DVH_choose_profile_canvas, text="Horizontal", variable=
1865     DVH_choice_of_profile_line_type, \
1866     value="h", bg='#ffffff', cursor='hand2').pack(side=LEFT)
1867 Radiobutton(DVH_choose_profile_canvas, text="Vertical", \
1868     variable=DVH_choice_of_profile_line_type, value='v', bg='#ffffff',
1869     cursor='hand2').pack(side=LEFT)
1870 Radiobutton(DVH_choose_profile_canvas, text="Draw", \
1871     variable=DVH_choice_of_profile_line_type, value="d", bg='#ffffff',
1872     cursor='hand2').pack(side=LEFT)
1873 #####
1874 """
1875 global DVH_film_panedwindow

```

```

1878 DVH_film_panedwindow = PanedWindow(DVH_view_film_doseplan_ROI, orient='
1879     vertical')
1880 DVH_film_panedwindow.pack()
1881 DVH_film_panedwindow.configure(sashrelief = RAISED, showhandle=True)
1882 """
1883 global DVH_scanned_image_text_image
1884 global DVH_film_dose_map_text_image
1885 global DVH_doseplan_text_image
1886 """
1887 global DVH_doseplan_write_image
1888 global DVH_doseplan_write_image_width
1889 global DVH_doseplan_write_image_height
1890 """
1891 global DVH_doseplan_write_image_var_x
1892 DVH_doseplan_write_image_var_x= 0
1893
1894 global DVH_new_window_factor_textbox ###
1895 """
1896 global DVH_doseplan_lateral_displacement
1897 global DVH_doseplan_vertical_displacement
1898 global DVH_doseplan_longitudinal_displacement
1899 global DVH_doseplan_patient_position
1900
1901 global DVH_reference_point_in_doseplan
1902
1903 global DVH_input_lateral_displacement
1904 global DVH_input_longitudinal_displacement
1905 global DVH_input_vertical_displacement
1906
1907 global DVH_slice_offset
1908 global DVH_offset
1909
1910 global DVH_isocenter_or_reference_point
1911
1912 global DVH_lateral
1913 global DVH_vertical
1914 global DVH_longitudinal
1915
1916 ##### Correction matrix
1917 #####
1918 global correction127_red
1919 with open('red_127.txt', 'r') as f:
1920     correction127_red = [[float(num) for num in line.split(',')] for line
1921                           in f]
1922 correction127_red = np.matrix(correction127_red)
1923 global correction127_green
1924 with open('green_127.txt', 'r') as f:
1925     correction127_green = [[float(num) for num in line.split(',')] for
1926                             line in f]
1927 correction127_green = np.matrix(correction127_green)
1928
1929 global correction127_blue
1930 with open('blue_127.txt', 'r') as f:

```

```

    correction127_blue = [[float(num) for num in line.split(',')] for line
                           in f]
1930 correction127_blue = np.matrix(correction127_blue)

1932 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)

1936 global correction72_green
1938 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)

1942 global correction72_blue
1944 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)

1946

1948 global correctionMatrix127
correctionMatrix127 = np.zeros((1270,1016,3))
1950 correctionMatrix127[:, :, 0] = correction127_blue[:, :]
correctionMatrix127[:, :, 1] = correction127_green[:, :]
1952 correctionMatrix127[:, :, 2] = correction127_red[:, :]

1954 global correctionMatrix72
correctionMatrix72 = np.zeros((720,576,3))
1956 correctionMatrix72[:, :, 0] = correction72_blue[:, :]
correctionMatrix72[:, :, 1] = correction72_green[:, :]
1958 correctionMatrix72[:, :, 2] = correction72_red[:, :]

```

FIDORA/Globals.py

A.3 gloVar.py

```

1000 #import necessary packages
1001 import tkinter as tk
1002 from tkinter import StringVar
1003 import numpy as np

1004

1006 #make GUI–window global
1007 global root
1008 root = tk.Tk()

1010 ##### initialize all global vairables
1011 #####
1012 global method
method="1"

```

```

1014
1015     global filename
1016     filename=StringVar(root)
1017     filename.set("Error!")
1018
1019     global dir_name
1020     dir_name=StringVar(root)
1021     dir_name.set("Error!")
1022
1023     global saveTo
1024     saveTo=StringVar(root)
1025     saveTo.set("Error!")
1026
1027     global pName
1028     pName = StringVar(root)
1029     pName.set("Error!")
1030
1031     global DPI
1032     DPI = tk.StringVar(root)
1033     DPI.set("127")
1034
1035     global comet
1036     comet = tk.StringVar(root)
1037     comet.set("1")
1038
1039     global filetype
1040     filetype = tk.StringVar(root)
1041     filetype.set(".dcm")
1042
1043     global saveAs
1044     saveAs = tk.StringVar(root)
1045     saveAs.set(".dcm")
1046
1047     global savetofolder
1048
1049     global Name
1050
1051
1052 ##### read and globalize the correction
1053     matrices #####
1054     global correction127_red
1055     with open('output_red_127.txt', 'r') as f:
1056         correction127_red = [[float(num) for num in line.split(',')] for line
1057                               in f]
1058     correction127_red = np.matrix(correction127_red)
1059     global correction127_green
1060     with open('output_green_127.txt', 'r') as f:
1061         correction127_green = [[float(num) for num in line.split(',')] for line
1062                               in f]
1063     correction127_green = np.matrix(correction127_green)
1064
1065     global correction127_blue
1066     with open('output_blue_127.txt', 'r') as f:
1067         correction127_blue = [[float(num) for num in line.split(',')] for line
1068                               in f]
1069     correction127_blue = np.matrix(correction127_blue)

```

```

1068 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)

1072 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)

1076 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)

1082 global correctionMatrix127
correctionMatrix127 = np.zeros((1270,1016,3))
correctionMatrix127[:, :, 0] = correction127_blue[:, :]
correctionMatrix127[:, :, 1] = correction127_green[:, :]
correctionMatrix127[:, :, 2] = correction127_red[:, :]

1088 global correctionMatrix72
correctionMatrix72 = np.zeros((720,576,3))
correctionMatrix72[:, :, 0] = correction72_blue[:, :]
correctionMatrix72[:, :, 1] = correction72_green[:, :]
correctionMatrix72[:, :, 2] = correction72_red[:, :]

1094 global correctedImage
correctedImage=None

```

FIDORA/gloVar.py

A.4 CorrectionFunctions.py

```

1000 import numpy as np
1001 import cv2
1002 from cv2 import imread, IMREAD_ANYCOLOR, IMREAD_ANYDEPTH
1003 from os.path import normpath, basename
1004 import os
1005 import gloVar
1006 from tkinter import messagebox
1007 import matplotlib.pyplot as plt

1008 # Function to perform det correction using correction matrix
1009 def correctionMatrix():
1010     dataset = cv2.imread(gloVar.filename.get().strip(), cv2.
1011     IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
1012     if(dataset is None):
1013         current_folder = os.getcwd()
1014         script_path = gloVar.filename.get()

```

```

1016     parent = os.path.dirname(script_path)
1017     os.chdir(parent)
1018     dataset=cv2.imread(basename(normpath(script_path)), cv2.
1019     IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
1020     os.chdir(current_folder)
1021     if(dataset is None):
1022         messagebox.showerror("Error", "Something has happen. Check that
1023         the filename does not contain , , ")
1024         return
1025
1026     if(dataset.shape[2] == 3):
1027         if(gloVar.DPI.get() == "127" and dataset.shape[0]==1270 and dataset.
1028             shape[1]==1016):
1029                 gloVar.correctedImage = abs(dataset-gloVar.correctionMatrix127
1030             )
1031             elif(gloVar.DPI.get() == "72" and dataset.shape[0]==720 and dataset.
1032                 shape[1]==576):
1033                     gloVar.correctedImage = abs(dataset - gloVar.
1034             correctionMatrix72)
1035             else:
1036                 messagebox.showerror("Error","The resolution of the image is
1037                 not consistent with dpi:" + gloVar.DPI.get())
1038
1039     else:
1040         messagebox.showerror("Error","The uploaded image need to be in RGB
1041             -format")

```

FIDORA/CorrectionFunctions.py

A.5 CoMet_functions.py

```

1000 import Globals
1001 import tkinter as tk
1002 from tkinter import filedialog, INSERT, DISABLED, messagebox, NORMAL,
1003     simpledialog, PhotoImage, BOTH, \
1004     E, S, N, W, ACTIVE, FLAT
1005 import os
1006 from os.path import normpath, basename
1007 import cv2
1008 from cv2 import imread, IMREADANYCOLOR, IMREADANYDEPTH, imwrite
1009 import numpy as np
1010 import SimpleITK as sitk
1011 import pydicom
1012 from PIL import Image, ImageTk
1013
1014 ## Function to do nothing (temp)
1015 def nothingButton():
1016     return
1017
1018 ## Function to upload file
1019 def UploadAction(event=None):
1020     Globals.CoMet_uploaded_filename.set(filedialog.askopenfilename())
1021     ext = os.path.splitext(Globals.CoMet_uploaded_filename.get())[-1].
1022         lower()

```

```

1022     if(ext==".tif"):
1023         Globals.CoMet_uploaded_file_text = tk.Text(Globals.
1024             CoMet_border_1_label, height=1, width=32)
1025             Globals.CoMet_uploaded_file_text.grid(row=0, column=0, columnspan
1026             =2, sticky=E+W, pady=(20,20), padx=(80,0))
1027             Globals.CoMet_uploaded_file_text.insert(INSERT, basename(normpath(
1028                 Globals.CoMet_uploaded_filename.get())))
1029             Globals.CoMet_uploaded_file_text.config(state=DISABLED, bd=0, font
1030             =('calibri', '12'), fg='gray', bg='#ffffff')
1031
1032         if (Globals.CoMet_progressbar_check_file):
1033             Globals.CoMet_progressbar_counter +=1
1034             Globals.CoMet_progressbar_check_file = False
1035             Globals.CoMet_progressbar["value"] = Globals.
1036             CoMet_progressbar_counter*25
1037             Globals.CoMet_progressbar_text = tk.Text(Globals.tab1_canvas,
1038                 height = 1, width=5)
1039                 Globals.CoMet_progressbar_text.grid(row=5, column=0, columnspan=1,
1040                 sticky=E, padx=(0,158), pady=(27,0))
1041                 Globals.CoMet_progressbar_text.insert(INSERT, str(Globals.
1042                 CoMet_progressbar_counter*25)+"%")
1043             if(Globals.CoMet_progressbar_counter*25 == 100):
1044                 Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
1045                 relief=FLAT, bg="#2C8EAD", font=('calibri', '10', 'bold'))
1046             else:
1047                 Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
1048                 relief=FLAT, bg='#ffffff', font=('calibri', '10', 'bold'))
1049
1050
1051     elif(ext==""):
1052         Globals.CoMet_uploaded_filename.set("Error!")
1053     else:
1054         messagebox.showerror("Error", "The file must be a .tif file")
1055         Globals.CoMet_uploaded_filename.set("Error!")
1056
1057 ## Function to set dpi
1058 def setCoMet_dpi():
1059     dpi = Globals.CoMet_dpi.get()
1060     print(dpi)
1061     return dpi
1062
1063 ## Function to set the export folder chosen by the user
1064 def setCoMet_export_folder():
1065     Globals.CoMet_export_folder.set(filedialog.askdirectory())
1066     if(Globals.CoMet_export_folder.get() == ""):
1067         #If this: the dialogbox was closed and no folder selected.
1068         Globals.CoMet_export_folder.set("Error!")
1069     else:
1070         current_folder = os.getcwd()
1071         os.chdir(Globals.CoMet_export_folder.get())
1072         save_to_folder=tk.Text(Globals.CoMet_border_2_label, height=1,
1073             width=32)
1074             save_to_folder.grid(row=0, column=0, columnspan=3, sticky=E+W,
1075             pady=(25,0), padx=(80,0))
1076             save_to_folder.insert(INSERT, basename(normpath(Globals.
1077                 CoMet_export_folder.get())))

```

```

1064     save_to_folder.config(state=DISABLED, bd=0, font=(‘calibri’, ‘12’))
1065     , fg=‘gray’, bg=‘#ffffff’)
1066     os.chdir(current_folder)
1067     if(Globals.CoMet_progressbar_check_folder):
1068         Globals.CoMet_progressbar_counter +=1
1069         Globals.CoMet_progressbar_check_folder = False
1070         Globals.CoMet_progressbar[“value”] = Globals.
1071         CoMet_progressbar_counter*25
1072         Globals.CoMet_progressbar_text = tk.Text(Globals.tab1.canvas,
1073         height=1, width=5)
1074         Globals.CoMet_progressbar_text.grid(row=5, column=0, columnspan=1,
1075         sticky=E, padx=(0,158), pady=(27,0))
1076         Globals.CoMet_progressbar_text.insert(INSERT, str(Globals.
1077         CoMet_progressbar_counter*25) + “%”)
1078         if(Globals.CoMet_progressbar_counter*25 == 100):
1079             Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
1080             relief=FLAT, bg=‘#2C8EAD’, font=(‘calibri’, ‘10’, ‘bold’))
1081         else:
1082             Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
1083             relief=FLAT, bg=‘#ffffff’, font=(‘calibri’, ‘10’, ‘bold’))

1084 ## Function to check that user has filled inn everything
1085 def checkAllWidgets(*args):
1086     if(Globals.CoMet_uploaded_filename.get() == “Error!” or Globals.
1087     CoMet_export_folder.get() == “Error!” or Globals.
1088     CoMet_corrected_image_filename.get() == “Error!”):
1089         return False
1090     else:
1091         return True

1092 ## Function to perform det correction using correction matrix
1093 def correctionMatrix():
1094     dataset = cv2.imread(Globals.CoMet_uploaded_filename.get().lstrip(),
1095     cv2.IMREADANYCOLOR | cv2.IMREADANYDEPTH)
1096     if(dataset is None):
1097         current_folder = os.getcwd()
1098         script_path = Globals.CoMet_uploaded_filename.get()
1099         parent = os.path.dirname(script_path)
1100         os.chdir(parent)
1101         dataset=cv2.imread(basename(normpath(script_path)), cv2.
1102         IMREADANYCOLOR | cv2.IMREADANYDEPTH)
1103         os.chdir(current_folder)
1104     if(dataset is None):
1105         messagebox.showerror(“Error”, “Something has happen. Check that
1106         the filename does not contain , , ”)
1107         return

1108     if(dataset.shape[2] == 3):
1109         if(dataset.shape[0]==1270 and dataset.shape[1]==1016):
1110             temp = abs(dataset-Globals.correctionMatrix127)
1111             Globals.CoMet_correctedImage = np.clip(temp, 0, 65535)
1112         elif(dataset.shape[0]==720 and dataset.shape[1]==576):
1113             temp = abs(dataset - Globals.correctionMatrix72)
1114             Globals.CoMet_correctedImage = np.clip(temp, 0, 65535)
1115         else:

```

```

    messagebox.showerror("Error", "The resolution of the image is
not consistent with dpi. Must be either 72 or 127")
1110
else:
    messagebox.showerror("Error", "The uploaded image need to be in RGB
-format")
1112

1114 ## Function to perform the correction on the image
1115 def Correct():
1116     if(checkAllWidgets() is False):
1117         messagebox.showerror("Error", "All boxes must be filled")
1118         return
1119     current_folder = os.getcwd()
1120     os.chdir(Globals.CoMet_export_folder.get())
1121     if(os.path.exists(Globals.CoMet_export_folder.get() + '/' + Globals.
CoMet_corrected_image_filename.get().lstrip() + Globals.CoMet_saveAs.
get()) is True):
1122         os.chdir(current_folder)
1123         messagebox.showerror("Error", "Filename already exists in folder.
Please write a new filename")
1124         Globals.CoMet_progressbar_counter -= 1
1125         Globals.CoMet_progressbar["value"] = Globals.
CoMet_progressbar_counter*25
1126         Globals.CoMet_progressbar_text = tk.Text(Globals.tab1.canvas,
width = 5, height=1)
1127         Globals.CoMet_progressbar_text.grid(row=5, column=0, columnspan=1,
sticky=E, padx=(0,158), pady=(27,0))
1128         Globals.CoMet_progressbar_text.insert(INSERT, str(Globals.
CoMet_progressbar_counter*25) + "%")
1129     if(Globals.CoMet_progressbar_counter*25 == 100):
1130         Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
relief=FLAT, bg="#2C8EAD", font=('calibri', '10', 'bold'))
1131     else:
1132         Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
relief=FLAT, bg="#ffffff", font=('calibri', '10', 'bold'))
1133         Globals.CoMet_save_button_1.config(state=ACTIVE)
1134         Globals.CoMet_save_filename.config(state=NORMAL)
1135     return
1136
os.chdir(current_folder)

1138 correctionMatrix()

1140
1141
1142 if (Globals.CoMet_correctedImage is None):
1143     messagebox.showerror("Error", "The image could not be corrected.
Please check all the specifications and try again.")
1144     Globals.CoMet_progressbar["value"]=0
1145     Globals.CoMet_progressbar_text = tk.Text(Globals.tab1.canvas,
height=1, width=5)
1146     Globals.CoMet_progressbar_text.grid(row=5, column=0, columnspan=1,
sticky=E, padx=(0,158), pady=(27,0))
1147     Globals.CoMet_progressbar_text.insert(INSERT, "0%")
1148     Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0, relief
=FLAT, bg="#ffffff", font=('calibri', '10', 'bold'))
1149 else:

```

```

1152     Globals.CoMet_progressbar_counter +=1
1153     Globals.CoMet_progressbar[ "value" ] = Globals.
1154     CoMet_progressbar_counter*25
1155     Globals.CoMet_progressbar_text = tk.Text(Globals.tab1_canvas,
1156     height=1, width=5)
1157     Globals.CoMet_progressbar_text.grid(row=5, column=0, columnspan=1,
1158     sticky=E, padx=(0,158), pady=(27,0))
1159     Globals.CoMet_progressbar_text.insert(INSERT, str(Globals.
1160     CoMet_progressbar_counter*25) + "%")
1161     if(Globals.CoMet_progressbar_counter*25 == 100):
1162         Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
1163         relief=FLAT, bg="#2C8EAD", font=( 'calibri' , '10' , 'bold' ))
1164     else:
1165         Globals.CoMet_progressbar_text.config(state=DISABLED, bd=0,
1166         relief=FLAT, bg="#ffffff", font=( 'calibri' , '10' , 'bold' ))
1167
1168 R=Globals.CoMet_correctedImage[:, :, 2];G=Globals.CoMet_correctedImage
1169[:, :, 1];B=Globals.CoMet_correctedImage[:, :, 0]
1170 if(Globals.CoMet_dpi.get()=="127"):
1171     corrImg_dicom = np.zeros((1270,1016,3))
1172     corrImg_dicom = corrImg_dicom.astype('uint16')
1173     corrImg_dicom[:, :, 0]=R; corrImg_dicom[:, :, 1]=G; corrImg_dicom
1174    [:, :, 2]=B
1175 elif(Globals.CoMet_dpi.get() == "72"):
1176     corrImg_dicom = np.zeros((720,576,3))
1177     corrImg_dicom = corrImg_dicom.astype('uint16')
1178     corrImg_dicom[:, :, 0]=R; corrImg_dicom[:, :, 1]=G; corrImg_dicom
1179    [:, :, 2]=B
1180 else:
1181     messagebox.showerror("Error", "Wrong DPI in image. No correction.\nPlease check all specifications and try again.")
1182
1183 corrImg_dicom = np.moveaxis(corrImg_dicom,-2,1)
1184 corrImg_dicom = np.rollaxis(corrImg_dicom,2,0)
1185 img_dicom = sitk.GetImageFromArray(corrImg_dicom)
1186 current_folder = os.getcwd()
1187 os.chdir(Globals.CoMet_export_folder.get())
1188 sitk.WriteImage(img_dicom, Globals.CoMet_corrected_image_filename.get()
1189 .lstrip() + Globals.CoMet_saveAs.get())
1190 os.chdir(current_folder)
1191 mod_NameAndModality = pydicom.dcmread(Globals.CoMet_export_folder.get()
1192 () + '/' + Globals.CoMet_corrected_image_filename.get().lstrip() +
1193 Globals.CoMet_saveAs.get())
1194 mod_NameAndModality.Modality = "RTDOSE"
1195 if(Globals.CoMet_patientName.get() != "Error!"):
1196     mod_NameAndModality.PatientName = Globals.CoMet_patientName.get()
1197 else:
1198     mod_NameAndModality.PatientName = "First^Last"
1199
1200 mod_NameAndModality.save_as(Globals.CoMet_export_folder.get() + '/' +
1201 Globals.CoMet_corrected_image_filename.get().lstrip() + Globals.
1202 CoMet_saveAs.get())
1203
1204 ds = pydicom.dcmread(Globals.CoMet_export_folder.get() + '/' + Globals.
1205 .CoMet_corrected_image_filename.get().lstrip() + Globals.CoMet_saveAs.
1206 get() ) # read dicom image
1207 img = ds.pixel_array # get image array

```

```

1192     RGB_image = np.zeros((img.shape[1], img.shape[2], 3))
1193
1194     for i in range(img.shape[0]):
1195         RGB_image[:, :, i] = img[i, :, :]
1196
1197
1198     img8 = (RGB_image/256).astype('uint8')
1199     height, width, channels = img8.shape
1200     img8 = Image.fromarray(img8, 'RGB')
1201
1202     img8 = img8.resize((250, 300))
1203
1204     Globals.CoMet_image_to_canvas = ImageTk.PhotoImage(image=img8)
1205
1206     Globals.CoMet_print_corrected_image.create_image(123,148,image=Globals.
1207     CoMet.image_to_canvas)
1208     Globals.CoMet_print_corrected_image.image = Globals.
1209     CoMet.image_to_canvas

```

FIDORA/CoMet_functions.py

A.6 Dose_response_functions.py

```

1000 import Globals
1001 import tkinter as tk
1002 import tkinter.ttk
1003 from tkinter import filedialog, INSERT, DISABLED, messagebox, NORMAL,
1004     simpledialog, \
1005     PhotoImage, BOTH, Toplevel, GROOVE, ACTIVE, FLAT, N, S, W, E, ALL, ttk
1006     , LEFT, RIGHT, Y, \
1007     Label, X, END, Button, StringVar
1008
1009 #import sympy as sp
1010 #from io import BytesIO
1011
1012 import cv2
1013 import numpy as np
1014 import os
1015 from os.path import normpath, basename
1016 import matplotlib
1017 import matplotlib.pyplot as plt
1018 from matplotlib.figure import Figure
1019 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
1020 #matplotlib.rcParams['text.usetex'] = True #lagt til for kunne skrive
1021     latex i string
1022 from scipy.optimize import curve_fit
1023 from scipy.optimize import curve_fit, OptimizeWarning
1024 from PIL import Image, ImageTk
1025 import sys
1026 from datetime import datetime
1027 import re
1028 import warnings
1029 warnings.filterwarnings("error")

```

```

1028 ## Function to do nothing (temp)
1029 def nothingButton():
1030     return
1031
1032
1033 def saveCalibration():
1034     ask_batch_window = tk.Toplevel(Globals.tab2)
1035     ask_batch_window.geometry("400x180")
1036     ask_batch_window.grab_set()
1037     ask_batch_window_canvas = tk.Canvas(ask_batch_window)
1038     ask_batch_window_canvas.config(bg='#ffffff', bd=0, highlightthickness=0)
1039     ask_batch_window_canvas.pack(expand=True, fill=BOTH)
1040
1041     batch_info = tk.Text(ask_batch_window_canvas, width=50, height=3)
1042     batch_info.grid(row=0, column=0, columnspan=2, sticky=N+S+E+W, padx=(10,10), pady=(30,10))
1043     ask_batch_window_canvas.grid_columnconfigure(0, weight=0)
1044     ask_batch_window_canvas.grid_rowconfigure(0, weight=0)
1045     batch_info.insert(INSERT, 'Write the LOT number of current GafChromic
1046 film:\n
1047     (Defaults to -)')
1048     batch_info.config(state=DISABLED, bd = 0, font=('calibri', '12'))
1049
1050     batch = tk.Text(ask_batch_window_canvas, width=20, height=1)
1051     batch.grid(row=1, column=0, sticky=N+S+W+E, padx=(5,5), pady=(10,10))
1052     ask_batch_window_canvas.grid_columnconfigure(1, weight=0)
1053     ask_batch_window_canvas.grid_rowconfigure(1, weight=0)
1054     batch.insert(INSERT, " ")
1055     batch.config(state=NORMAL, bd = 3, font=('calibri', '12'))
1056
1057     def save_batch():
1058         Globals.dose_response_batch_number= batch.get("1.0",'end-1c')
1059         if(Globals.dose_response_batch_number == " "):
1060             Globals.dose_response_batch_number = "_"
1061             save_batch_button.config(state=DISABLED)
1062             ask_batch_window.destroy()
1063         elif(re.match("[A-Za-z0-9_]*$", (Globals.
1064 dose_response_batch_number).lstrip())==None):
1065             messagebox.showerror("Error","LOT number can only contain
1066 letters and/or numbers")
1067             ask_batch_window.destroy()
1068             saveCalibration()
1069             return
1070         else:
1071             save_batch_button.config(state=DISABLED)
1072             ask_batch_window.destroy()
1073
1073     f = open('calibration.txt', 'r')
1074     lines = f.readlines()
1075     f.close()
1076     string_to_file = str(datetime.now()) + " " + str(Globals.
1077 dose_response_batch_number) + " " +
1078             str(Globals.popt_red[0]) + " " + str(Globals.popt_red[1]) +
1079             " " + str(Globals.popt_red[2]) + "\n"
1080     if(len(lines) < 5):

```

```

1078         f = open('calibration.txt', 'a')
1079         f.write(string_to_file)
1080         f.close()
1081     else:
1082         new_lines = [lines[1], lines[2], lines[3], lines[4],
1083         string_to_file]
1084         f = open('calibration.txt', 'w')
1085         for i in range(len(new_lines)):
1086             f.write(new_lines[i])
1087         f.close()
1088
1089         messagebox.showinfo("Info", "The calibration has been saved")
1090
1091         save_button_frame = tk.Frame(ask_batch_window_canvas)
1092         save_button_frame.grid(row=1, column = 1, padx=(5,5), pady=(10,10))
1093         ask_batch_window_canvas.grid_columnconfigure(2, weight=0)
1094         ask_batch_window_canvas.grid_rowconfigure(2, weight=0)
1095         save_button_frame.config(bg = '#ffffff')
1096
1097         save_batch_button = tk.Button(save_button_frame, text='Save', image=
1098             Globals.save_button, cursor='hand2', font=('calibri', '14'), \
1099             relief=FLAT, state=ACTIVE, command=save_batch)
1100         save_batch_button.pack(fill=BOTH, expand=True)
1101         save_batch_button.image = Globals.save_button
1102
1103 def UploadAction(new_window, event=None):
1104     file = filedialog.askopenfilename()
1105     ext = os.path.splitext(file)[-1].lower()
1106     if(ext==".tif"):
1107         Globals.dose_response_uploaded_filenames = np.append(Globals.
1108         dose_response_uploaded_filenames, file)
1109         uploaded_filename = tk.Text(new_window, height=1, width=1)
1110         uploaded_filename.grid(row=Globals.
1111         dose_response_new_window_row_count, column=0, columnspan=2, sticky=E+W,
1112         , pady=(5,5), padx=(100,0))
1113         new_window.grid_columnconfigure(Globals.
1114         dose_response_new_window_weight_count, weight=0)
1115         new_window.grid_rowconfigure(Globals.
1116         dose_response_new_window_weight_count, weight=0)
1117         uploaded_filename.insert(INSERT, basename(normpath(file)))
1118         uploaded_filename.config(state=DISABLED, bd=0, font=('calibri', '12'),
1119         fg='gray')
1120         Globals.dose_response_new_window_row_count+=1
1121         Globals.dose_response_new_window_weight_count+=1
1122     elif(ext==""):
1123         return
1124     else:
1125         messagebox.showerror("Error", "The file must be a .tif file")
1126
1127 def readImage(filename):
1128     image = cv2.imread(filename, cv2.IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)
1129
1130     if(image is None):
1131         current_folder = os.getcwd()
1132         parent = os.path.dirname(filename)
1133         os.chdir(parent)

```

```

1126     image=cv2.imread(basename(normpath(filename)), cv2.IMREADANYCOLOR
1127     | cv2.IMREAD_ANYDEPTH)
1128     os.chdir(current_folder)
1129     if(image is None):
1130         messagebox.showerror("Error", "Something has happen. Check that
1131         the filename does not contain , , ")
1132         return
1133
1134     if(image.shape[2] == 3):
1135         if(image.shape[0]==1270 and image.shape[1]==1016):
1136             Globals.doseResponse_dpi.set("127")
1137             image = abs(image-Globals.correctionMatrix127)
1138             image = np.clip(image, 0, 65535)
1139         elif(image.shape[0]==720 and image.shape[1]==576):
1140             Globals.doseResponse_dpi.set("72")
1141             image = abs(image - Globals.correctionMatrix72)
1142             image = np.clip(image, 0, 65535)
1143         else:
1144             messagebox.showerror("Error","The resolution of the image is
1145             not consistent with dpi")
1146
1147         else:
1148             messagebox.showerror("Error","The uploaded image need to be in RGB
1149             -format")
1150
1151         sum_red=0;sum_green=0;sum_blue=0
1152         if(Globals.doseResponse_dpi.get() == "127"):
1153             for i in range(622,647):
1154                 for j in range(495, 520):
1155                     sum_red += image[i,j,2]
1156                     sum_green += image[i,j,1]
1157                     sum_blue += image[i,j,0]
1158             sum_red = sum_red/(25*25)
1159             sum_green = sum_green/(25*25)
1160             sum_blue = sum_blue/(25*25)
1161             return sum_red, sum_green, sum_blue
1162         elif(Globals.doseResponse_dpi.get() == "72"):
1163             for i in range(352,367):
1164                 for j in range(280,295):
1165                     sum_red+=image[i,j,2]
1166                     sum_green+=image[i,j,1]
1167                     sum_blue+=image[i,j,0]
1168             sum_red = sum_red/(15*15)
1169             sum_green = sum_green/(15*15)
1170             sum_blue = sum_blue/(15*15)
1171             return sum_red, sum_green, sum_blue
1172         else:
1173             messagebox.showerror("Error", "Something has gone wrong with the
1174             doseResponse_dpi")
1175             return False
1176
1177 def plot_dose_response():
1178     print("sjekk")
1179     ****
1180     sd_red_arr=[];sd_green_arr=[];sd_blue_arr=[]
1181     temp_dose = [item[0] for item in Globals.avg_red_vector]
```

```

1178     temp_avg_red = [item[1] for item in Globals.avg_red_vector]
1179     temp_avg_green = [item[1] for item in Globals.avg_green_vector]
1180     temp_avg_blue = [item[1] for item in Globals.avg_blue_vector]
1181
1182     for i in range(len(temp_dose)):
1183         sd_red_arr.append(np.std(Globals.dose_response_sd_list_red[i]))
1184         sd_green_arr.append(np.std(Globals.dose_response_sd_list_green[i]))
1185     )
1186         sd_blue_arr.append(np.std(Globals.dose_response_sd_list_blue[i]))
1187
1188     if(len(sd_red_arr) > 0):
1189         Globals.dose_response_sd_avg_red.set(sum(sd_red_arr)/len(
1190             sd_red_arr))
1191         Globals.dose_response_sd_avg_green.set(sum(sd_green_arr)/len(
1192             sd_green_arr))
1193         Globals.dose_response_sd_avg_blue.set(sum(sd_blue_arr)/len(
1194             sd_blue_arr))
1195
1196         Globals.dose_response_sd_max_red.set(max(sd_red_arr))
1197         Globals.dose_response_sd_max_red_dose.set(str(temp_dose[
1198             sd_red_arr.index(Globals.dose_response_sd_max_red.get())]))
1199         Globals.dose_response_sd_max_green.set(max(sd_green_arr))
1200         Globals.dose_response_sd_max_green_dose.set(str(temp_dose[
1201             sd_green_arr.index(Globals.dose_response_sd_max_green.get())]))
1202         Globals.dose_response_sd_max_blue.set(max(sd_blue_arr))
1203         Globals.dose_response_sd_max_blue_dose.set(str(temp_dose[
1204             sd_blue_arr.index(Globals.dose_response_sd_max_blue.get())]))
1205
1205
1206     Globals.dose_response_sd_min_red.set(min(sd_red_arr))
1207     Globals.dose_response_sd_min_red_dose.set(str(temp_dose[
1208         sd_red_arr.index(Globals.dose_response_sd_min_red.get())]))
1209     Globals.dose_response_sd_min_green.set(min(sd_green_arr))
1210     Globals.dose_response_sd_min_green_dose.set(str(temp_dose[
1211         sd_green_arr.index(Globals.dose_response_sd_min_green.get())]))
1212     Globals.dose_response_sd_min_blue.set(min(sd_blue_arr))
1213     Globals.dose_response_sd_min_blue_dose.set(str(temp_dose[
1214         sd_blue_arr.index(Globals.dose_response_sd_min_blue.get())]))
1215
1215
1216     else:
1217         Globals.dose_response_sd_avg_red.set(0)
1218         Globals.dose_response_sd_avg_green.set(0)
1219         Globals.dose_response_sd_avg_blue.set(0)
1220         Globals.dose_response_sd_max_red.set(0)
1221         Globals.dose_response_sd_max_red_dose.set('—')
1222         Globals.dose_response_sd_max_green.set(0)
1223         Globals.dose_response_sd_max_green_dose.set('—')
1224         Globals.dose_response_sd_max_blue.set(0)
1225         Globals.dose_response_sd_max_blue_dose.set('—')
1226         Globals.dose_response_sd_min_red.set(0)
1227         Globals.dose_response_sd_min_red_dose.set('—')
1228         Globals.dose_response_sd_min_green.set(0)
1229         Globals.dose_response_sd_min_green_dose.set('—')
1230         Globals.dose_response_sd_min_blue.set(0)
1231         Globals.dose_response_sd_min_blue_dose.set('—')
1232
1233     print("sjekk2")
1234     ****

```

```

1224     fig = Figure(figsize=(5,3))
1225     a = fig.add_subplot(111)
1226     canvas = FigureCanvasTkAgg(fig, master=Globals.
1227         dose_response_plot_frame)
1228     canvas.get_tk_widget().grid(row=0,column=0,columnspan=4, sticky=N+S+E+
1229     W, padx=(5,0), pady=(0,0))
1230     if(Globals.dose_response_var1.get()):
1231         a.errorbar(temp_dose,temp_avg_red,yerr=sd_red_arr, fmt='ro')
1232     if(Globals.dose_response_var2.get()):
1233         a.errorbar(temp_dose, temp_avg_green, yerr=sd_green_arr, fmt='g^')
1234     if(Globals.dose_response_var3.get()):
1235         a.errorbar(temp_dose, temp_avg_blue, yerr=sd_blue_arr, fmt='bs')

1236     if(len(temp_avg_red) > 3):
1237         sorted_temp_red = sorted(Globals.avg_red_vector, key=lambda l:l[0])
1238         sorted_temp_avg_red = [item[1] for item in sorted_temp_red]
1239         sorted_temp_dose = [item[0] for item in sorted_temp_red]

1240         sorted_temp_green = sorted(Globals.avg_green_vector, key=lambda l:l[0])
1241         sorted_temp_avg_green = [item[1] for item in sorted_temp_green]

1242         sorted_temp_blue = sorted(Globals.avg_blue_vector, key=lambda l:l[0])
1243         sorted_temp_avg_blue = [item[1] for item in sorted_temp_blue]

1244     try:
1245         Globals.popt_red, pcov_red = curve_fit(fitted_dose_response,
1246         sorted_temp_dose, sorted_temp_avg_red, p0=[1700, 15172069, -390],
1247         maxfev=10000)
1248         popt_green, pcov_green = curve_fit(fitted_dose_response,
1249         sorted_temp_dose, sorted_temp_avg_green, p0=[1700, 15172069, -390],
1250         maxfev=10000)

1251         xdata = np.linspace(0,2000,1001)
1252         ydata_red = np.zeros(len(xdata));ydata_green=np.zeros(len(
1253         xdata))
1254         for i in range(len(xdata)):
1255             ydata_red[i] = fitted_dose_response(xdata[i], Globals.
1256             popt_red[0], Globals.popt_red[1], Globals.popt_red[2])
1257             ydata_green[i] = fitted_dose_response(xdata[i], popt_green
1258             [0], popt_green[1], popt_green[2])
1259             if(Globals.dose_response_var1.get()):
1260                 a.plot(xdata, ydata_red, color='red')
1261             if(Globals.dose_response_var2.get()):
1262                 a.plot(xdata, ydata_green, color='green')
1263             if(Globals.dose_response_var3.get()):
1264                 a.plot(sorted_temp_dose, sorted_temp_avg_blue , color='
1265 blue')

1266             out_text_function = "Pixel value = " + str(round(Globals.
1267             popt_red[0])) + " + " + str(round(Globals.popt_red[1])) + "/(dose - (
1268             + str(round(Globals.popt_red[2])) + ))"
1269             standarddavvik_rgb = "Standard deviation red = " + str(round(
1270             Globals.dose_response_sd_avg_red.get()))
1271             #write_out_respons_function = tk.Text(Globals.
1272             dose_response_equation_frame)#
1273             height=1, width=10)

```

```

1264     #write_out_respons_function.insert(INSERT, out_text_function )
1265     ##ekstra linje med standardavvik, pr ver    inserter de ogs
1266     #write_out_respons_function.insert(INSERT, standardavvik_rgb)
1267     def clickFunction(a,b,c):
1268         tmpText = StringVar()
1269         text = "Pixel value(PV) as function of dose(D): "
1270         a=str(a)  #str(round(Globals.popt_red[0]))
1271         b=str(b) #str(round(Globals.popt_red[1]))
1272         c=str(c) #str(round(Globals.popt_red[2]))
1273         latex= a + " " + "\frac {" + f"{b}" + "}{" + "D" + "—" +
1274             f"{c}" + "}"
1275         avgR=str(round(Globals.dose_response_sd_avg_red.get()));
1276         minR=str(round(Globals.dose_response_sd_min_red.get())); maxR=str(
1277             round(Globals.dose_response_sd_max_red.get()))
1278         latexR=(avgR+"," +minR+"," +maxR+"); textR="\n\nStandard
1279         deviations (SD): \nSD for red color channel: (avg, max,min)="
1280         avgG=str(round(Globals.dose_response_sd_avg_green.get()));
1281         minG=str(round(Globals.dose_response_sd_min_green.get())); maxG=str(
1282             round(Globals.dose_response_sd_max_green.get()))
1283         latexG=(avgG+"," +minG+"," +maxG+"); textG="\n\nSD for
1284         green color channel: (avg, max,min)="
1285         avgB=str(round(Globals.dose_response_sd_avg_blue.get()));
1286         minB=str(round(Globals.dose_response_sd_min_blue.get())); maxB=str(
1287             round(Globals.dose_response_sd_max_blue.get()))
1288         latexB=(avgB+"," +minB+"," +maxB+"); textB="\n\nSD for
1289         blue color channel: (avg, max,min)="
1290         tmpText.set(latex)
1291
1292         #tmpText = entry.get()
1293         tmpText = "$"+tmpText.get()+"$"
1294
1295         axLatex.clear()
1296         axLatex.text(0.01, 0.3, text+PV = "+tmpText+textR+latexR+
1297             textG+latexG+textB+latexB, fontsize = 4) #this is where the text is
1298             added to the axis
1299             canvasLatex.draw()
1300
1301             #root = tk.Tk()
1302             #make a frame and place it with grid
1303             #mainframe = Frame(root)
1304             #mainframe.grid(row=0,column=0)
1305
1306             #make a label and place it with grid
1307             labelLatex = Label(Globals.dose_response_equation_frame)
1308             labelLatex.grid(row=0,column=0)
1309
1310             figLatex = matplotlib.figure.Figure(figsize=(2.4, 1), dpi=250)
1311             figLatex.subplots_adjust(bottom=-0.01, top=1.2, left=-0.01,
1312                 right=2)
1313             axLatex = figLatex.add_subplot(111)
1314
1315             canvasLatex = FigureCanvasTkAgg(figLatex, master=labelLatex)
1316             canvasLatex.get_tk_widget().grid(row=0, column=0, sticky="N")
1317             canvasLatex._tkcanvas.grid(row=0, column=0, sticky="N") # (
1318                 side=TOP, fill=BOTH, expand=1)

```

```

1308     axLatex.get_xaxis().set_visible(False)
1309     axLatex.get_yaxis().set_visible(False)
1310     a_=round(Globals.popt_red[0])
1311     b_=round(Globals.popt_red[1])
1312     c_=round(Globals.popt_red[2])
1313     clickFunction(a_,b_,c_)

1314     #displayButton = Button(Globals.dose_response_equation_frame,
1315     text="display equation",width=15,command=lambda: clickFunction(12,3,4))
1316     #displayButton.grid(row=1,column=0,sticky="N")

1317     #write_out_respons_function.grid(row=0, column=0, sticky=N+S+W+E, pady=(5,5), padx=(5,5))
1318     #Globals.dose_response_equation_frame.grid_columnconfigure(0, weight=0)
1319     #Globals.dose_response_equation_frame.grid_rowconfigure(0, weight=0)
1320     #write_out_respons_function.config(state=DISABLED, bd=0, font=('calibri', '12'), bg="#ffffff")
1321     Globals.dose_response_save_calibration_button.config(state=ACTIVE)
1322     except OptimizeWarning:
1323         messagebox.showwarning("Warning", "It appears that you have optimization problems. \
1324 Try adding more data points to improve the optimization.\ \
1325 Or, check that your specified dose matches your uploaded files.")
1326     except RuntimeError:
1327         messagebox.showwarning("Warning", "It appears that you have optimization problems. \
1328 Try adding more data points to improve the optimization. \
1329 Or, check that your specified dose matches your uploaded files.")
1330     #####
1331     a.set_title("Dose-response", fontsize=12)
1332     a.set_ylabel("Pixel value", fontsize=12)
1333     a.set_xlabel("Dose", fontsize=12)
1334     fig.tight_layout()

1336 def delete_line(delete_button):
1337     #The button index equals the index in Globals.avg_red_vector etc.
1338     button_index = Globals.dose_response_delete_buttons.index(delete_button)
1339     Globals.dose_response_red_list[button_index].destroy()
1340     Globals.dose_response_green_list[button_index].destroy()
1341     Globals.dose_response_blue_list[button_index].destroy()
1342     Globals.dose_response_dose_list[button_index].destroy()
1343     Globals.dose_response_delete_buttons[button_index].destroy()
1344     del(Globals.dose_response_red_list[button_index])
1345     del(Globals.dose_response_green_list[button_index])
1346     del(Globals.dose_response_blue_list[button_index])
1347     del(Globals.dose_response_dose_list[button_index])

1349 if(len(Globals.dose_response_delete_buttons) > 1):
1350     del(Globals.avg_red_vector[button_index])
1351     del(Globals.avg_green_vector[button_index])
1352     del(Globals.avg_blue_vector[button_index])

```

```

1354     del(Globals.dose_response_delete_buttons[button_index])
1355     del(Globals.dose_response_sd_list_red[button_index])
1356     del(Globals.dose_response_sd_list_green[button_index])
1357     del(Globals.dose_response_sd_list_blue[button_index])
1358 else:
1359     Globals.avg_red_vector = []
1360     Globals.avg_green_vector = []
1361     Globals.avg_blue_vector = []
1362     Globals.dose_response_delete_buttons = []
1363     Globals.dose_response_sd_list_red = []
1364     Globals.dose_response_sd_list_green = []
1365     Globals.dose_response_sd_list_blue = []
1366
1367 Globals.dose_response_files_row_count = 2
1368 for i in range(len(Globals.dose_response_delete_buttons)):
1369     Globals.dose_response_red_list[i].grid(row=Globals.
1370 dose_response_files_row_count, column=1, sticky=N+S+W+E, padx=(0,0))
1371     Globals.dose_response_green_list[i].grid(row=Globals.
1372 dose_response_files_row_count, column=3, sticky=N+S+W+E, padx=(0,0))
1373     Globals.dose_response_blue_list[i].grid(row=Globals.
1374 dose_response_files_row_count, column=5, sticky=N+S+W+E, padx=(0,5))
1375     Globals.dose_response_dose_list[i].grid(row=Globals.
1376 dose_response_files_row_count, column=0, sticky=N+S+W+E, padx=(0,15))
1377     Globals.dose_response_delete_buttons[i].grid(row=Globals.
1378 dose_response_files_row_count, column=7, sticky=N+S+W+E, padx=(5,5))
1379     Globals.dose_response_files_row_count+=1
1380
1381 if(len(Globals.dose_response_delete_buttons) < 4):
1382     Globals.dose_response_save_calibration_button.config(state=
1383 DISABLED)
1384
1385 plot_dose_response()
1386
1387 def fitted_dose_response(D, a, b, c):
1388     return a + b/(D-c)
1389
1390
1391
1392 ## Function to find mean of uploaded images with same dose.
1393 def avgAllFiles(write_dose_box, new_window):
1394     #First block is to test that everything is filled in and as expected.
1395     dose_input = write_dose_box.get("1.0",'end-1c')
1396     if (dose_input == ""):
1397         messagebox.showerror("Error", "Input dose")
1398         return
1399     try:
1400         dose_input = float(dose_input)
1401     except:
1402         messagebox.showerror("Error", "The dose must be a number")
1403         return
1404     if(len(Globals.dose_response_uploaded_filenames) == 0):
1405         messagebox.showerror("Error", "No files uploaded")
1406         return
1407
1408     #Calculates the mean in each color channel
1409     avg_red=0;avg_green=0;avg_blue=0

```

```

1406     red_temp_sd_list = []; green_temp_sd_list = []; blue_temp_sd_list = []
1407     for i in range(0, len(Globals.dose_response_uploaded_filenames)):
1408         if(readImage(Globals.dose_response_uploaded_filenames[i])==False):
1409             messagebox.showerror("Error", "A mistake has happened in
1410             readImage()")
1411             return
1412         red, green, blue = readImage(Globals.
1413             dose_response_uploaded_filenames[i])
1414         avg_red+=red
1415         avg_green+=green
1416         avg_blue+=blue
1417
1418
1420     red_temp_sd_list.append(red)
1421     green_temp_sd_list.append(green)
1422     blue_temp_sd_list.append(blue)
1423
1424
1426     avg_red = avg_red/len(Globals.dose_response_uploaded_filenames)
1427     avg_green = avg_green/len(Globals.dose_response_uploaded_filenames)
1428     avg_blue = avg_blue/len(Globals.dose_response_uploaded_filenames)
1429     temp_dose = [item[0] for item in Globals.avg_red_vector]
1430     isTest = False
1431     try:
1432         indx = temp_dose.index(dose_input)
1433         Globals.avg_red_vector[indx][1] = (avg_red + Globals.
1434             avg_red_vector[indx][1])/2
1435         Globals.avg_green_vector[indx][1] = (avg_green + Globals.
1436             avg_green_vector[indx][1])/2
1437         Globals.avg_blue_vector[indx][1] = (avg_blue + Globals.
1438             avg_blue_vector[indx][1])/2
1439
1440         for i in range(0, len(red_temp_sd_list)):
1441             Globals.dose_response_sd_list_red[indx].append(
1442                 red_temp_sd_list[i])
1443             Globals.dose_response_sd_list_green[indx].append(
1444                 green_temp_sd_list[i])
1445             Globals.dose_response_sd_list_blue[indx].append(
1446                 blue_temp_sd_list[i])
1447
1448     except:
1449         Globals.avg_red_vector.append([dose_input, avg_red])
1450         Globals.avg_green_vector.append([dose_input, avg_green])
1451         Globals.avg_blue_vector.append([dose_input, avg_blue])
1452
1453         Globals.dose_response_sd_list_red.append(red_temp_sd_list)
1454         Globals.dose_response_sd_list_green.append(green_temp_sd_list)
1455         Globals.dose_response_sd_list_blue.append(blue_temp_sd_list)
1456
1457     isTest = True
1458
1459     temp_dose = [item[0] for item in Globals.avg_red_vector]
1460
1461     if(isTest):
1462         result_red = tk.Text(Globals.tab2_canvas_files, height=1, width=7)
1463         result_red.insert(INSERT, round(avg_red))
1464         result_red.grid(row=Globals.dose_response_files_row_count, column
1465 =1, sticky=N+S+W+E, padx=(0,0))

```

```

1454     Globals.tab2_canvas_files.grid_columnconfigure(Globals.
1455 dose_response_files_weightcount, weight=0)
1456     Globals.tab2_canvas_files.grid_rowconfigure(Globals.
1457 dose_response_files_weightcount, weight=0)
1458         result_red.config(state=DISABLED, bd=0, font=(‘calibri’, ‘12’))
1459         Globals.dose_response_red_list.append(result_red)
1460         Globals.dose_response_files_weightcount+=1
1461
1462     result_green = tk.Text(Globals.tab2_canvas_files, height=1, width
1463 =7)
1464         result_green.insert(INSERT, round(avg_green))
1465         result_green.grid(row=Globals.dose_response_files_row_count,
1466 column=3, sticky=N+S+W+E, padx=(0,0))
1467         Globals.tab2_canvas_files.grid_columnconfigure(Globals.
1468 dose_response_files_weightcount, weight=0)
1469         Globals.tab2_canvas_files.grid_rowconfigure(Globals.
1470 dose_response_files_weightcount, weight=0)
1471         result_green.config(state=DISABLED, bd=0, font=(‘calibri’, ‘12’))
1472         Globals.dose_response_green_list.append(result_green)
1473         Globals.dose_response_files_weightcount+=1
1474
1475     result_blue = tk.Text(Globals.tab2_canvas_files, height=1, width
1476 =7)
1477         result_blue.insert(INSERT, round(avg_blue))
1478         result_blue.grid(row=Globals.dose_response_files_row_count, column
1479 =5, sticky=N+S+W+E, padx=(0,5))
1480         Globals.tab2_canvas_files.grid_columnconfigure(Globals.
1481 dose_response_files_weightcount, weight=0)
1482         Globals.tab2_canvas_files.grid_rowconfigure(Globals.
1483 dose_response_files_weightcount, weight=0)
1484         result_blue.config(state=DISABLED, bd=0, font=(‘calibri’, ‘12’))
1485         Globals.dose_response_blue_list.append(result_blue)
1486         Globals.dose_response_files_weightcount+=1
1487
1488     dose_print = tk.Text(Globals.tab2_canvas_files, height=1, width
1489 =10)
1490         dose_print.insert(INSERT, dose_input)
1491         dose_print.grid(row=Globals.dose_response_files_row_count, column
1492 =0, sticky=N+S+W+E, padx=(0,15))
1493         Globals.tab2_canvas_files.grid_columnconfigure(Globals.
1494 dose_response_files_weightcount, weight=0)
1495         Globals.tab2_canvas_files.grid_rowconfigure(Globals.
1496 dose_response_files_weightcount, weight=0)
1497         dose_print.config(state=DISABLED, bd=0, font=(‘calibri’, ‘12’))
1498         Globals.dose_response_dose_list.append(dose_print)
1499         Globals.dose_response_files_weightcount+=1
1500
1501     path = os.path.dirname(sys.argv[0])
1502     path = path + r”\delete.png”
1503     img = ImageTk.PhotoImage(file=path)
1504
1505     delete_button = tk.Button(Globals.tab2_canvas_files, text=‘Remove’,
1506 , image=img, cursor=‘hand2’, font=(‘calibri’, ‘18’), \
1507         highlightthickness= 0, relief=FLAT, state=ACTIVE, width = 15)
1508         delete_button.image = img
1509         Globals.dose_response_delete_buttons.append(delete_button)
1510         delete_button.config(command=lambda: delete_line(delete_button)))

```

```

1496         delete_button.grid(row=Globals.dose_response_files_row_count,
1497                               column=7, sticky=N+S+W+E, padx=(5,5))
1498         Globals.tab2_canvas_files.grid_columnconfigure(Globals.
1499                                         dose_response_files_weightcount, weight=0)
1500         Globals.tab2_canvas_files.grid_rowconfigure(Globals.
1501                                         dose_response_files_weightcount, weight=0)
1502         delete_button.config(bg='#ffffff', activebackground='#ffffff',
1503                               activeforeground='#ffffff', highlightthickness=0)
1504         Globals.dose_response_files_row_count+=1
1505         Globals.dose_response_files_weightcount+=1
1506
1507     else:
1508         Globals.dose_response_red_list[indx].config(state=NORMAL)
1509         Globals.dose_response_red_list[indx].delete('1.0', END)
1510         Globals.dose_response_red_list[indx].insert(INSERT, round(Globals.
1511                                         avg_red_vector[indx][1]))
1512         Globals.dose_response_red_list[indx].config(state=DISABLED)
1513
1514         Globals.dose_response_green_list[indx].config(state=NORMAL)
1515         Globals.dose_response_green_list[indx].delete('1.0', END)
1516         Globals.dose_response_green_list[indx].insert(INSERT, round(
1517                                         Globals.avg_green_vector[indx][1]))
1518         Globals.dose_response_green_list[indx].config(state=DISABLED)
1519
1520         Globals.dose_response_blue_list[indx].config(state=NORMAL)
1521         Globals.dose_response_blue_list[indx].delete('1.0', END)
1522         Globals.dose_response_blue_list[indx].insert(INSERT, round(Globals.
1523                                         avg_blue_vector[indx][1]))
1524         Globals.dose_response_blue_list[indx].config(state=DISABLED)
1525
1526     plot_dose_response()
1527     new_window.destroy()
1528
1529 def create_window():
1530     new_window = tk.Toplevel(Globals.tab2)
1531     new_window.geometry("360x500")
1532     new_window.grab_set()
1533
1534     new_window_frame = tk.Frame(new_window)
1535     new_window_frame.config(relief=FLAT, bg='#ffffff', highlightthickness
1536                               =0)
1537
1538     new_window_scroll_canvas = tk.Canvas(new_window_frame)
1539     new_window_scroll_canvas.config(bg='#ffffff', height=450, width=200)
1540     new_window_scroll_canvas.grid_propagate(0)
1541
1542     new_window_scroll = ttk.Scrollbar(new_window_frame, command=
1543                                         new_window_scroll_canvas.yview)
1544
1545     scrollable_frame= tk.Frame(new_window_scroll_canvas)
1546
1547     scrollable_frame.bind("<Configure>", lambda e:
1548                           new_window_scroll_canvas.configure(scrollregion=
1549                                         new_window_scroll_canvas.bbox('all')))
1550     new_window_scroll_canvas.create_window((0,0), window=scrollable_frame,
1551                                         anchor='nw')

```

```

1540    new_window_scroll_canvas.configure(yscrollcommand=new_window_scroll.
1541                                         set)
1542
1543    new_window_canvas = tk.Canvas(scrollable_frame)
1544    new_window_canvas.config(relief=FLAT, bg='#ffffff', highlightthickness
1545                               =0)
1546    new_window_canvas.pack(fill=BOTH, expand=True)
1547
1548    new_window_frame.pack(expand=True, fill = BOTH)
1549    new_window_scroll_canvas.pack(side=LEFT, fill=BOTH, expand=True)
1550    new_window_scroll.pack(side=RIGHT, fill=Y)

1551
1552    Globals.dose_response_uploaded_filenames = []
1553
1554    explain_text = tk.Text(new_window_canvas, height=11, width = 47)
1555    explain_text.grid(row=0, column = 0, rowspan = 3, columnspan=2, sticky
1556                      =N+S+W+E, pady=(10,10), padx=(10,10))
1557    new_window_canvas.grid_columnconfigure(0, weight=0)
1558    new_window_canvas.grid_rowconfigure(0, weight=0)
1559    explain_text.insert(INSERT, "\nHere you can upload several files all irradiated with \nthe same dose. \
1560    Fill in dose and an average will be \ncalculated and used in the \
1561    calibration. You are also \nable to upload \
1562    only one file each time, and FIDORA \nwill keep track and average before \
1563    fitting the \ndose-response.")
1564    explain_text.config(state=DISABLED, bd=0, font=('calibri', '11'))
1565
1566    write_dose_box_frame = tk.Frame(new_window_canvas)
1567    write_dose_box_frame.grid(row=2, column=1, sticky=N+S+E+W, pady=(0,30)
1568                             , padx=(0,10))
1569    new_window_canvas.grid_columnconfigure(1, weight=0)
1570    new_window_canvas.grid_rowconfigure(1, weight=0)
1571    write_dose_box_frame.config(bg='#ffffff')
1572
1573    dose_border_label = Label(write_dose_box_frame, image = Globals.
1574                               dose_response_dose_border)
1575    dose_border_label.image=Globals.dose_response_dose_border
1576    dose_border_label.config(bg='#ffffff', borderwidth=0)
1577    dose_border_label.pack(expand=True, fill=BOTH)
1578
1579    write_dose_text = tk.Text(new_window_canvas, height=1, width=19)
1580    write_dose_text.insert(INSERT, "Write dose here (cGy):")
1581    write_dose_text.config(state=DISABLED, bd=0, font=('calibri', '11'),
1582                           bg='#ffffff')
1583    write_dose_text.grid(row=1, column=1, sticky=E+W, pady=(140,0) , padx
1584                          =(5,5))
1585    new_window_canvas.grid_columnconfigure(3, weight=0)
1586    new_window_canvas.grid_rowconfigure(3, weight=0)
1587
1588    write_dose_box = tk.Text(dose_border_label, height=1, width=8)
1589    write_dose_box.grid(row=0,column=0, sticky=N+S+W+E, pady=(10,0) , padx
1590                          =(20,5))
1591    write_dose_box.insert(INSERT, " ")
1592    write_dose_box.config(state=NORMAL, bd=0, font=('calibri', '18'), bg=
1593                          '#ffffff')

```

```

1586     upload_button_frame = tk.Frame(new_window_canvas)
1587     upload_button_frame.grid(row=2, column=0, sticky=N+S+W+E, pady=(0,30))
1588     new_window_canvas.grid_columnconfigure(2, weight=0)
1589     new_window_canvas.grid_rowconfigure(2, weight=0)
1590     upload_button_frame.config(bg='#fffff')
1591
1592     upload_button = tk.Button(upload_button_frame, text='Upload file',
1593                               image=Globals.upload_button_image, \
1594                               cursor='hand2', font=('calibri', '14'), relief=FLAT, state=ACTIVE,
1595                               command=lambda: UploadAction(new_window_canvas))
1596     upload_button.pack(expand=True, fill=BOTH)
1597     upload_button.config(bg='#fffff', activebackground='#fffff',
1598                           activeforeground='#fffff', highlightthickness=0)
1599     upload_button.image=Globals.upload_button_image
1600
1601     Globals.dose_response_inOrOut = True
1602     done_button = tk.Button(new_window, text='DONE', cursor='hand2', font
1603                             =('calibri', '20', 'bold'), \
1604                             relief=FLAT, state=ACTIVE, command=lambda: avgAllFiles(
1605                             write_dose_box, new_window))
1606     done_button.config(activebackground="#04BAA6", bg= '#04BAA6',
1607                         activeforeground='#fffff', fg= '#fffff', height=1)
1608     done_button.pack(expand=True, fill=X)
1609
1610
1611
1612 def clear_all():
1613     for i in range(len(Globals.dose_response_delete_buttons)):
1614         Globals.dose_response_red_list[i].destroy()
1615         Globals.dose_response_green_list[i].destroy()
1616         Globals.dose_response_blue_list[i].destroy()
1617         Globals.dose_response_delete_buttons[i].destroy()
1618         Globals.dose_response_dose_list[i].destroy()
1619
1620     Globals.dose_response_dose_list = []
1621     Globals.dose_response_red_list = []
1622     Globals.dose_response_green_list = []
1623     Globals.dose_response_blue_list = []
1624     Globals.dose_response_delete_buttons = []
1625
1626     Globals.dose_response_sd_list_red = []
1627     Globals.dose_response_sd_list_green = []
1628     Globals.dose_response_sd_list_blue = []
1629
1630     Globals.dose_response_var1.set(1)
1631     Globals.dose_response_var2.set(1)
1632     Globals.dose_response_var3.set(1)
1633
1634     Globals.avg_red_vector = []
1635     Globals.avg_green_vector = []
1636     Globals.avg_blue_vector = []
1637
1638     Globals.dose_response_batch_number = "—"
1639     Globals.popt_red = np.zeros(3)
1640     Globals.dose_response_inOrOut = True
1641     Globals.dose_response_files_weightcount = 8

```

```

1636     Globals.dose_response_files_row_count = 2
1638     Globals.dose_response_save_calibration_button.config(state=DISABLED)
1640
1641     plot_dose_response()

```

FIDORA/Dose_response_functions.py

A.7 Map_dose.py

```

1000 ##### Map dose #####
1001 import Globals
1002 import tkinter as tk
1003 from tkinter import filedialog, INSERT, DISABLED, messagebox, NORMAL,
1004     simpledialog,\_
1005     PhotoImage, BOTH, Canvas, N, S, W, E, ALL, Frame, SUNKEN, Radiobutton
1006     , GROOVE
1007 import os
1008 from os.path import normpath, basename
1009 import cv2
1010 from cv2 import imread, IMREADANYCOLOR, IMREADANYDEPTH, imwrite
1011 import numpy as np
1012 import SimpleITK as sitk
1013 import pydicom
1014 from PIL import Image, ImageTk
1015 import os
1016 import sys
1017 import matplotlib
1018 import matplotlib.pyplot as plt
1019 from matplotlib.figure import Figure
1020 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
1021
1022 def dose_to_pixel(D,a,b,c):
1023     return a + b/(D-c)
1024
1025 def pixel_to_dose(P,a,b,c):
1026     return c + b/(P-a)
1027
1028 ##### LEgg til medianfilter
1029 def calculate_dose_map(cv2Img):
1030     wid = Globals.map_dose_ROI_x_end.get() - Globals.map_dose_ROI_x_start.
1031     get()
1032     heig = Globals.map_dose_ROI_y_end.get() - Globals.map_dose_ROI_y_start
1033     .get()
1034     print(wid, heig)
1035     doseMap_film = np.zeros((heig,wid))
1036     for i in range(heig):
1037         for j in range(wid):
1038             doseMap_film[i,j] = pixel_to_dose(cv2Img[Globals.
1039             map_dose_ROI_y_start.get()+i,Globals.map_dose_ROI_x_start.get()+j,2],
1040             \
1041             Globals.popt_red[0], Globals.popt_red[1], Globals.popt_red
1042             [2])

```

```

1038
1040     fig = Figure(figsize=(0.8,0.8))
1041     a = fig.add_subplot(111)
1042     #test ane:
1043     #plot.image = cv2.flip(doseMap_film,-1) #fjern test etterp
1044     plot.image = a.pcolormesh(doseMap_film, cmap='viridis', rasterized=
1045     True, vmin=0, vmax=600)
1046     fig.colorbar(plot.image, ax=a)
1047     canvas_dosemap_film = FigureCanvasTkAgg(fig, master = Globals.tab3)
1048     canvas_dosemap_film.get_tk_widget().place(relwidth=0.6, relheight=
1049     0.55, relx = 0.03, rely=0.2)#relwidth=0.3, rely=0.1
1050     canvas_dosemap_film.draw()
1051     #plotte dosekartet (dette m v re krympet (408,508))
1052
1053 def prepare_Image():
1054     cv2Img = cv2.imread(Globals.map_dose_film_dataset.get(), cv2.
1055     IMREADANYCOLOR | cv2.IMREADANYDEPTH)
1056     if(cv2Img is None):
1057         current_folder = os.getcwd()
1058         parent = os.path.dirname(Globals.map_dose_film_dataset.get())
1059         os.chdir(parent)
1060         cv2Img=cv2.imread(basename(normpath(Globals.map_dose_film_dataset.
1061     get())), cv2.IMREADANYCOLOR | cv2.IMREADANYDEPTH)
1062         os.chdir(current_folder)
1063     if(cv2Img is None):
1064         messagebox.showerror("Error", "Something has happen. Check that
1065         the filename does not contain , , ")
1066         return
1067
1068     if(cv2Img.shape[2] == 3):
1069         if(cv2Img.shape[0]==1270 and cv2Img.shape[1]==1016):
1070             cv2Img = abs(cv2Img-Globals.correctionMatrix127)
1071             cv2Img = np.clip(cv2Img, 0, 65535)
1072         elif(cv2Img.shape[0]==720 and cv2Img.shape[1]==576):
1073             cv2Img = abs(cv2Img - Globals.correctionMatrix72)
1074             cv2Img = np.clip(cv2Img, 0, 65535)
1075         else:
1076             messagebox.showerror("Error","The resolution of the image is
1077             not consistent with dpi")
1078
1079     else:
1080         messagebox.showerror("Error","The uploaded image need to be in RGB
1081             -format")
1082         return
1083
1084 #Read last calibration done, or ask if one wish to change
1085 choose_batch_window = tk.Toplevel(Globals.tab3)
1086 choose_batch_window.geometry("800x400")
1087 choose_batch_window.grab_set()
1088
1089 def set_batch():
1090     choose_batch_window.destroy()
1091     f = open('calibration.txt', 'r')
1092     lines = f.readlines()

```

```

1088     words = lines[Globals.map_dose_film_batch.get()].split()
1089     Globals.popt_red[0] = float(words[3])
1090     Globals.popt_red[1] = float(words[4])
1091     Globals.popt_red[2] = float(words[5])
1092     f.close()
1093     calculate_dose_map(cv2Img)

1094     batch_cnt = 0
1095     r = open('calibration.txt', 'r')
1096     lines = r.readlines()
1097     write_batch_y_coord = 0.3
1098     for l in lines:
1099         words = l.split()
1100         line = "Batch nr. : " + words[2] + ".      Date:    " + words[0] + "
1101         " + words[1] + "."
1102         write_batch = tk.Text(choose_batch_window, width=1, height=1)
1103         write_batch.place(relwidth=0.7, relheight=0.1, relx = 0.1, rely=
1104         write_batch_y_coord)
1105         write_batch.insert(INSERT, line)
1106         write_batch.config(state=DISABLED, bd = 0, font=(‘calibri’, ‘12’))

1107         Radiobutton(choose_batch_window, text=' ', cursor='hand2', font=(‘
1108         calibri’, ‘14’), \
1109         variable=Globals.map_dose_film_batch, value=batch_cnt).place(
1110         relwidth=0.08, \
1111         relheight=0.1, relx=0.8, rely=write_batch_y_coord)

1112         write_batch_y_coord+=0.1; batch_cnt+=1

1113         ok_batch_button = tk.Button(choose_batch_window, text='OK', cursor='
1114         hand2', \
1115         font=(‘calibri’, ‘14’), overrelief=GROOVE, state=tk.ACTIVE, width
1116         = 15, command=set_batch)
1117         ok_batch_button.place(relwidth=0.2, relheight=0.2, relx=0.4, rely=0.9)
1118         r.close()

1119     def draw_ROI(img, scale_horizontal, scale_vertical):
1120         draw_ROI_window = tk.Toplevel(Globals.tab3)
1121         draw_ROI_window.grab_set()
1122         local_frame= Frame(draw_ROI_window, bd = 2, relief=SUNKEN)
1123         local_frame.grid_rowconfigure(0, weight=1)
1124         local_frame.grid_columnconfigure(0, weight=1)

1125         local_canvas = Canvas(local_frame, bd=0)
1126         local_canvas.grid(row=0,column=0, sticky=N+S+E+W)

1127         w = 10 + img.width()
1128         h = 10 + img.height()
1129         draw_ROI_window.geometry("%dx%d+0+0" % (w, h))

1130         local_canvas.create_image(0,0,image=img, anchor="nw")
1131         local_canvas.config(scrollregion=local_canvas.bbox(ALL), cursor='arrow
1132         ')
1133         local_canvas.image= img

1134         rectangle = local_canvas.create_rectangle(0,0,0,0, outline='green')

```

```

1138     def buttonPushed(event):
1139         Globals.map_dose_ROI_x_start.set(event.x)
1140         Globals.map_dose_ROI_y_start.set(event.y)
1141
1142     def buttonMoving(event):
1143         local_canvas.coords(rectangle, Globals.map_dose_ROI_x_start.get(),
1144                             Globals.map_dose_ROI_y_start.get(), \
1145                             event.x, event.y)
1146
1147     def buttonReleased(event):
1148         Globals.map_dose_ROI_x_end.set(event.x)
1149         Globals.map_dose_ROI_y_end.set(event.y)
1150         local_canvas.coords(rectangle, Globals.map_dose_ROI_x_start.get(),
1151                             Globals.map_dose_ROI_y_start.get(), \
1152                             Globals.map_dose_ROI_x_end.get(), Globals.map_dose_ROI_y_end.
1153                             get())
1154         local_canvas.itemconfig(rectangle, outline='Blue')
1155         answer = messagebox.askquestion("Question", "Happy with placement?")
1156         if(answer=='yes'):
1157             Globals.map_dose_ROI_x_start.set(Globals.map_dose_ROI_x_start.
1158                 get()*scale_horizontal)
1159             Globals.map_dose_ROI_y_start.set(Globals.map_dose_ROI_y_start.
1160                 get()*scale_vertical)
1161             Globals.map_dose_ROI_x_end.set(Globals.map_dose_ROI_x_end.get(
1162                 )*scale_horizontal)
1163             Globals.map_dose_ROI_y_end.set(Globals.map_dose_ROI_y_end.get(
1164                 )*scale_vertical)
1165             prepare_Image()
1166             draw_ROI_window.destroy()
1167
1168         local_canvas.bind("<B1-Motion>", buttonMoving)
1169         local_canvas.bind("<Button-1>", buttonPushed)
1170         local_canvas.bind("<ButtonRelease-1>", buttonReleased)
1171
1172         local_frame.pack(fill='both', expand=1)
1173
1174     def draw_image_with_marks(img, scale_horizontal, scale_vertical,
1175                             mark_isocenter_window, frame):
1176         #check_isocenter_window = tk.Toplevel(Globals.tab3)
1177         #check_isocenter_window.grab_set()
1178         #frame_local = Frame(mark_isocenter_window, bd=2, relief=SUNKEN) #
1179         #check_isocenter_window, bd=2, relief=SUNKEN)
1180         #frame_local.grid_rowconfigure(0, weight=1)
1181         #frame_local.grid_columnconfigure(0, weight=1)
1182         canvas_local = Canvas(frame, bd=0)
1183         canvas_local.grid(row=0, column=0, sticky=N+S+E+W)
1184
1185         #w = 10 + img.width()
1186         #h = 10 + img.height()
1187         #check_isocenter_window.geometry("%dx%d+0+0" % (w, h))
1188
1189         canvas_local.create_image(0,0,image=img, anchor="nw")
1190         canvas_local.config(scrollregion=canvas_local.bbox(ALL), cursor='arrow')
1191         canvas_local.image= img

```

```

    canvas_local.create_oval(Globals.
1184 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.
1185 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.
1186 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.
1187 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')

1190 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)

1194 x1 = Globals.map_dose_isocenter_map_x_coord_unscaled[0]
1195 x2 = Globals.map_dose_isocenter_map_x_coord_unscaled[1]
1196 x3 = Globals.map_dose_isocenter_map_x_coord_unscaled[2]
1197 x4 = Globals.map_dose_isocenter_map_x_coord_unscaled[3]
1198 y1 = Globals.map_dose_isocenter_map_y_coord_unscaled[0]
1199 y2 = Globals.map_dose_isocenter_map_y_coord_unscaled[1]
1200 y3 = Globals.map_dose_isocenter_map_y_coord_unscaled[2]
1201 y4 = Globals.map_dose_isocenter_map_y_coord_unscaled[3]

1204
1205
1206
1207
1208
1209
1210 if(y1==y2 and y3==y4):
    messagebox.showerror("Error", "Reference points are not correct.
Try again.")
1211 check_isocenter_window.destroy()
upload_film_data()
1212 elif(y1==y2):
    if(x1==x2):
        messagebox.showerror("Error", "Reference points are not
correct. Try again.")
        check_isocenter_window.destroy()
        upload_film_data()
1213 else:

```

```

1220         a = 0; b=y1
1221         if(x3==x4):
1222             isocenter = [x3,y1]
1223         else:
1224             c=(y3-y4)/(x3-x4); d = y3 - c*x3
1225             isocenter = [(d-b)/(a-c), b]
1226     elif(y3==y4):
1227         if(x3==x4):
1228             messagebox.showerror("Error", "Reference points are not
correct. Try again.")
1229             check_isocenter_window.destroy()
1230             upload_film_data()
1231         else:
1232             c = 0; d = y3
1233             if(x1==x2):
1234                 isocenter = [x1,y3]
1235             else:
1236                 a = (y1-y2)/(x1-x2); b = y1 - a*x1
1237                 isocenter = [(d-b)/(a-c), d]
1238     else:
1239         if(x1==x2 and x3==x4):
1240             messagebox.showerror("Error", "Reference points are not
correct. Try again.")
1241             check_isocenter_window.destroy()
1242             upload_film_data()
1243         elif(x1==x2):
1244             c = (y3-y4)/(x3-x4); d = y3 - c*x3
1245             isocenter = [x1, c*x1+d]
1246         elif(x3==x4):
1247             a = (y1-y2)/(x1-x2); b = y1 - a*x1
1248             isocenter = [x3, a*x3+d]
1249         else:
1250             a = (y1-y2)/(x1-x2)
1251             b = y1 - a*x1
1252             c = (y3-y4)/(x3-x4)
1253             d = y3 - c*x3
1254             isocenter = [(d-b)/(a-c), a*(d-b)/(a-c) + b]

1255 #frame.pack(fill='both', expand=1)
1256 if(isocenter[0] < 0 or isocenter[1] < 0 or isocenter[0] > 408 or
isocenter[1] > 508):
1257     messagebox.showerror("Error", "Reference points are not correct.
Try again.")
1258     mark_isocenter_window.destroy() #check_isocenter_window.destroy()
1259     upload_film_data()
1260 else:
1261     canvas_local.create_oval(isocenter[0]-6, isocenter[1]-6, isocenter
[0]+6,isocenter[1]+6, outline="pink")
1262     answer = messagebox.askquestion("Question", "Happy with placement?"
, parent=mark_isocenter_window)#check_isocenter_window)
1263     if(answer=="yes"):
1264         Globals.map_dose_isocenter_film = [isocenter[0]*
scale_horizontal, isocenter[1]*scale_vertical]
1265         mark_isocenter_window.destroy() #check_isocenter_window.
destroy()
1266         draw_ROI(img, scale_horizontal, scale_vertical)

```

```

1270     else :
1271         mark_isocenter_window . destroy () #check_isocenter_window .
1272         destroy ()
1273             upload_film_data ()
1274             return
1275
1276
1277
1278
1279 def upload_film_data () :
1280     current_folder = os.getcwd()
1281     os.chdir(os.path.dirname(sys.argv[0]))
1282     img = Image.open(Globals.map_dose_film_dataset.get())
1283     if(not (img.width == 1016 or img.width == 576)):
1284         messagebox.showerror("Error", "Dpi in image has to be 127 or 72")
1285         return
1286
1287     Globals.map_dose_isocenter_map_x_coord_scaled = []
1288     Globals.map_dose_isocenter_map_x_coord_unscaled = []
1289     Globals.map_dose_isocenter_map_y_coord_scaled = []
1290     Globals.map_dose_isocenter_map_y_coord_unscaled = []
1291
1292     mark_isocenter_window = tk.Toplevel(Globals.tab3)
1293     mark_isocenter_window . grab_set()
1294     frame = Frame(mark_isocenter_window , bd=2, relief=SUNKEN)
1295     frame . grid_rowconfigure(0, weight=1)
1296     frame . grid_columnconfigure(0, weight=1)
1297     canvas = Canvas(frame , bd=0)
1298     canvas . grid(row=0, column=0, sticky=N+S+E+W)
1299
1300
1301     scale_horizontal = img . width/408
1302     scale_vertical = img . height/508
1303     img = img . resize((408,508))
1304     img = ImageTk.PhotoImage(image=img)
1305     os.chdir(current_folder)
1306     canvas . image = img
1307
1308     w = 10 + img . width()
1309     h = 10 + img . height()
1310     mark_isocenter_window . geometry("%dx%d+0+0" % (w, h))
1311     canvas . create_image(0,0,image=img, anchor="nw")
1312     canvas . config(scrollregion=canvas . bbox(ALL), cursor='sb_up_arrow')
1313     #x_coor = []
1314     #y_coor = []
1315
1316     def findCoords(event):
1317         Globals.map_dose_isocenter_map_x_coord_scaled.append(event.x*
1318             scale_vertical)
1319         Globals.map_dose_isocenter_map_y_coord_scaled.append(event.y*
1320             scale_horizontal)
1321         Globals.map_dose_isocenter_map_x_coord_unscaled.append(event.x)
1322         Globals.map_dose_isocenter_map_y_coord_unscaled.append(event.y)

```

```

1322         canvas.create_oval(event.x-2, event.y-2, event.x+2, event.y+2,
1323             fill='red')
1324             if (len(Globals.map_dose_isocenter_map_x_coord_scaled)==1):
1325                 canvas.config(cursor='sb_down_arrow')
1326             elif(len(Globals.map_dose_isocenter_map_x_coord_scaled)==2):
1327                 canvas.config(cursor='sb_right_arrow')
1328             elif(len(Globals.map_dose_isocenter_map_x_coord_scaled)==3):
1329                 canvas.config(cursor='sb_left_arrow')
1330             else:
1331                 #mark_isocenter_window.destroy()
1332                 draw_image_with_marks(img, scale_horizontal, scale_vertical,
1333                     mark_isocenter_window, frame)
1334
1335
1336         canvas.bind("<Button 1>",findCoords)
1337         frame.pack(fill='both', expand=1)
1338
1339     def UploadAction(type, event=None):
1340         if(type == "FILM"):
1341             if(Globals.popt_red[0]==1):
1342                 messagebox.showerror("Error", "No calibration has been found.
1343 To a calibration first.")
1344                 return
1345             Globals.map_dose_film_dataset.set(filedialog.askopenfilename())
1346             ext = os.path.splitext(Globals.map_dose_film_dataset.get())[-1].lower()
1347             if(ext==".tif"):
1348                 upload_film_data()
1349                 return
1350             elif(ext==""):
1351                 Globals.map_dose_film_dataset.set("Error!")
1352             else:
1353                 messagebox.showerror("Error", "The file must be a .tif file")
1354                 Globals.map_dose_film_dataset.set("Error!")
1355
1356 #laste opp bilde og markere i bildene, egen funksjon
1357 #gammatest, lese opp p det og implementere
1358 #Eksportere figurer og dataset ut av programmet
1359 #m lagre siste kalibrering (sp rre hvilken kalibrering bruker vil bruke
1360 #)
1361 # hvordan er doseplanene lagret.
1362 #Endre geometrien slik at den passer alle skjermer. Kan man bruke
1363 #skjermst rrelsen i en algoritme?
1364
1365 # Laste opp doseplan (for n er det en enkel matrise, selvkonstruert.)
1366 # laste opp skannet film, korriger automatisk
1367 # brukeren spesifiserer posisjon p film
1368 # gj re film om til dose map (bruke dose response)
1369 # tegne dose plan og dose map fra film
1370 # regne gamma
1371 # tegne gamma pass/fail og variasjoner
1372 # skriv ut all info vi f r fra gammatest

```

FIDORA/Map_Dose.py

A.8 Profile_functions.py

```
1000 import Globals
1001 import tkinter as tk
1002 from tkinter import filedialog, INSERT, DISABLED, messagebox, NORMAL,
1003     simpledialog,\ 
1004     PhotoImage, BOTH, Canvas, N, S, W, E, ALL, Frame, SUNKEN, Radiobutton,
1005     GROOVE, ACTIVE, \
1006     FLAT, END, Scrollbar, HORIZONTAL, VERTICAL, ttk, TOP, RIGHT, LEFT, ttk
1007 import os
1008 from os.path import normpath, basename
1009 from PIL import Image, ImageTk
1010 import cv2
1011 from cv2 import imread, IMREADANYCOLOR, IMREADANYDEPTH, imwrite
1012 import pydicom
1013 from matplotlib.figure import Figure
1014 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
1015 import matplotlib as mpl
1016 from matplotlib import cm
1017 import matplotlib.pyplot as plt
1018 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
1019     NavigationToolbar2Tk
1020 import numpy as np
1021
1022 #Bresenham's line algorithm
1023
1024 def clearAll():
1025     Globals.profiles_film_orientation.set('-')
1026     Globals.profiles_film_orientation.menu.config(state=ACTIVE, bg = '#'
1027         'ffffff', width=15, relief=FLAT)
1028
1029     #Globals.profiles_depth.config(state=NORMAL, fg='black')
1030     #Globals.profiles_depth.delete('1.0', END)
1031     #Globals.profiles_depth.insert(INSERT, " ")
1032
1033     Globals.profiles_iscoenter_coords = []
1034     Globals.profiles_film_isocenter = None
1035     Globals.profiles_film_reference_point = None
1036     Globals.profiles_mark_isocenter_up_down_line = []
1037     Globals.profiles_mark_isocenter_right_left_line = []
1038     Globals.profiles_mark_isocenter_oval = []
1039     Globals.profiles_mark_reference_point_oval = []
1040     Globals.profiles_mark_ROI_rectangle = []
1041     Globals.profiles_ROI_coords = []
1042
1043     #if(Globals.profiles_isocenter_check and Globals.profiles_ROI_check):
1044     #    Globals.profiles_done_button.config(state=DISABLED)
1045     Globals.profiles_isocenter_check = False
1046     Globals.profiles_ROI_check = False
1047     Globals.profiles_reference_point_check = False
1048     Globals.profiles_ROI_reference_point_check = False
1049
1050     #if(Globals.profiles_film_window_open):
1051     #    Globals.profiles_film_window.destroy()
1052     #    Globals.profiles_film_window_open = False
```

```

1052     Globals.profiles_upload_button_film.config(state=ACTIVE)
1053     Globals.profiles_upload_button_doseplan.config(state=DISABLED)
1054     Globals.profiles_upload_button_rtplan.config(state=DISABLED)

1056     Globals.profiles_distance_isocenter_ROI = []

1058     Globals.profiles_film_dataset = None
1059     Globals.profiles_film_dataset_red_channel = None
1060     Globals.profiles_film_dataset_ROI = None
1061     Globals.profiles_film_dataset_ROI_red_channel = None

1062     Globals.profiles_film_match_isocenter_dataset = np.zeros((7,7))

1064
1065     Globals.profiles_dataset_doseplan = None
1066     Globals.profiles_dataset_rtplan = None
1067     Globals.profiles_isocenter_mm = None
1068     Globals.profiles_test_if_added_rtplan = False
1069     Globals.profiles_test_if_added_doseplan = False

1070     Globals.tab4_canvas.unbind("<Up>")
1071     Globals.tab4_canvas.unbind("<Down>")

1074
1075     return

1076 def getCoordsInRandomLine(x1,y1,x2,y2):
1077     points = []
1078     issteep = abs(y2-y1) - abs(x2-x1)
1079     if issteep > 0:
1080         x1, y1 = y1, x1
1081         x2, y2 = y2, x2
1082     rev = False
1083     if x1 > x2:
1084         x1, x2 = x2, x1
1085         y1, y2 = y2, y1
1086         rev = True
1087     deltax = x2 - x1
1088     deltay = abs(y2-y1)
1089     error = int(deltax / 2)
1090     y = y1
1091     ystep = None
1092     if y1 < y2:
1093         ystep = 1
1094     else:
1095         ystep = -1
1096     for x in range(x1, x2 + 1):
1097         if issteep:
1098             points.append((y, x))
1099         else:
1100             points.append((x, y))
1101         error -= deltay
1102         if error < 0:
1103             y += ystep
1104             error += deltax
1105     # Reverse the list if the coordinates were reversed
1106     if rev:

```

```

1108     points . reverse ()
1109
1110     return points
1111
1112 def drawProfiles(even):
1113     if Globals . profiles_choice_of_profile_line_type . get () == 'h' or
1114         Globals . profiles_choice_of_profile_line_type . get () == 'v':
1115             Globals . profiles_lines = []
1116
1117     if Globals . profiles_dataset_doseplan == None:
1118         return
1119
1120     Globals . profiles_adjust_button_right . config ( state=ACTIVE )
1121     Globals . profiles_adjust_button_left . config ( state=ACTIVE )
1122     Globals . profiles_adjust_button_down . config ( state=ACTIVE )
1123     Globals . profiles_adjust_button_up . config ( state=ACTIVE )
1124     Globals . profiles_adjust_button_return . config ( state=ACTIVE )
1125
1126
1127     def draw(line_orient , dataset_film , dataset_doseplan):
1128         Globals . profile_plot_canvas . delete ('all')
1129         fig= Figure(figsize=(5,3))
1130         a = fig . add_subplot(111)
1131
1132         a . axis (ymin=0,ymax=6.2)
1133
1134         plot_canvas = FigureCanvasTkAgg(fig , master=Globals .
1135         profile_plot_canvas)
1136         plot_canvas . get_tk_widget () . grid (row=0,column=0,columnspan=4,
1137         sticky=N+E+W+S , padx=(5,0) , pady=(0,0))
1138         #annotation = a . annotate ("HEI" , xy=(0,0) , xytext=(0,20))
1139         #annotation . set_visible (False)
1140         #txt = tk . Text(Globals . profile_plot_canvas , width=50, height=6)
1141         #txt . insert (INSERT , " ")
1142         #txt . grid (row=1, column = 1, sticky=N+E+W+S , pady=(5,0) , padx
1143         =(5,0))
1144         #txt . config (bg='#ffffff' , font=('calibri' , '10') , state=DISABLED,
1145         relief=FLAT, bd= 0)
1146         cols = (' ', 'Point match' , 'Distance' , 'Dose' , 'Rel. to max' ,
1147         'Rel. to target')
1148         listBox = ttk . Treeview(Globals . profile_plot_canvas , columns=cols ,
1149         show='headings')
1150         for col in cols:
1151             listBox . heading (col , text=col , anchor=W)
1152             listBox . column (col , width=84, stretch=False , anchor=W)
1153             listBox . grid (row=1, column=0, columnspan=4)
1154             lst = [[ 'Film: ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' ], \
1155                 [ 'Doseplan: ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' ]]
1156             for i , (name , m , dis , d , rdROI , rdTarget) in enumerate (lst):
1157                 listBox . insert ("", "end" , values=(name , m , dis , d , rdROI ,
1158                 rdTarget))
1159                 #a . text (0,0 , "" , fontsize=7, bbox=dict (facecolor='gray' ,
1160                 alpha=0.1))
1161                 #txt . set_visible (False)
1162                 v_line = a . axvline (x=0, ymin=0, ymax=50, c='gray')

```

```

1156     #v_line.set_visible(False)
1158
1159     if line_orient == 'h':
1160         if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
1161             dy = Globals.profiles_doseplan_dataset_ROI.shape[1]/2
1162         elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
1163             dy = Globals.profiles_doseplan_dataset_ROI.shape[1]*2/2
1164         else:
1165             dy = Globals.profiles_doseplan_dataset_ROI.shape[1]*3/2
1166         dx = dataset_film.shape[1]*0.2/2
1167         x = np.linspace(-dx,dx, dataset_film.shape[1])
1168         y = np.linspace(-dy,dy, Globals.profiles_doseplan_dataset_ROI.
1169                         shape[1])
1170         plot_film = dataset_film[Globals.
1171         profiles_coordinate_in_dataset,:]/100
1172         plot_doseplan = dataset_doseplan[Globals.
1173         profiles_coordinate_in_dataset, :]
1174         film = a.plot(x,plot_film, color='r', label='Film')
1175         dose = a.plot(y,plot_doseplan, color='b', label='Doseplan')
1176     elif line_orient == 'v':
1177         if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
1178             dy = Globals.profiles_doseplan_dataset_ROI.shape[0]/2
1179         elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
1180             dy = Globals.profiles_doseplan_dataset_ROI.shape[0]*2/2
1181         else:
1182             dy = Globals.profiles_doseplan_dataset_ROI.shape[0]*3/2
1183         dx = dataset_film.shape[0]*0.2/2
1184         x = np.linspace(-dx,dx, dataset_film.shape[0])
1185         y = np.linspace(-dy,dy, Globals.profiles_doseplan_dataset_ROI.
1186                         shape[0])
1187         plot_film = dataset_film[:,Globals.
1188         profiles_coordinate_in_dataset]/100
1189         plot_doseplan = dataset_doseplan[:, Globals.
1190         profiles_coordinate_in_dataset] #Globals.
1191         profiles_doseplan_dataset_ROI
1192         film=a.plot(x,plot_film, color='r', label='Film')
1193         dose=a.plot(y,plot_doseplan, color='b', label='Doseplan')
1194     elif line_orient == 'd':
1195         start_f_x, start_f_y = Globals.profiles_line_coords_film[0]
1196         end_f_x, end_f_y = Globals.end_point
1197         dx=np.sqrt(((end_f_x-start_f_x)*0.2)**2 + ((end_f_y-start_f_y)
1198 *0.2)**2)/2
1199         if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
1200             start_d_x, start_d_y = Globals.
1201             profiles_line_coords_doseplan[0]
1202             end_d_x, end_d_y = Globals.end_point
1203             end_d_x=end_d_x/5; end_d_y=end_d_y/5
1204             dy=np.sqrt(((end_d_x-start_d_x))**2 + ((end_d_y-start_d_y)
1205 )**2)/2
1206         elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
1207             start_d_x, start_d_y = Globals.
1208             profiles_line_coords_doseplan[0]
1209             end_d_x, end_d_y = Globals.end_point
1210             end_d_x=end_d_x/10; end_d_y=end_d_y/10
1211             dy=np.sqrt(((end_d_x-start_d_x)*2)**2 + ((end_d_y-
1212             start_d_y)*2)**2)/2
1213     else:

```

```

1201         start_d_x , start_d_y = Globals.
1202         profiles_line_coords_doseplan[0]
1203             end_d_x , end_d_y = Globals.end_point
1204             end_d_x=end_d_x/15; end_d_y=end_d_y/15
1205             dy=np.sqrt(((end_d_x-start_d_x)*3)**2 + ((end_d_y-
1206             start_d_y)*3)**2)/2

1207
1208     print(dx, dy)
1209     x = np.linspace(-dx,dx, len(dataset_film))
1210     y = np.linspace(-dy,dy, len(dataset_doseplan))
1211     plot_film=dataset_film/100
1212     plot_doseplan=dataset_doseplan
1213     film = a.plot(x,plot_film, color='r', label='Film')
1214     dose= a.plot(y,plot_doseplan, 'b', label='Doseplan')

1215 else:
1216     messagebox.showerror("Error", "Fatal error. Something has gone
1217 wrong, try again \n(Code: draw")
1218     return

1219
1220     a.legend()
1221     a.set_title("Profiles", fontsize=12)
1222     a.set_ylabel("Dose (Gy)", fontsize=12)
1223     a.set_xlabel("Distance (mm)", fontsize=12)

1224 def mouseMove(event):
1225     if event.inaxes == a:
1226         dist = event.xdata
1227         idx_film = np.searchsorted(x, dist)
1228         idx_doseplan = np.searchsorted(y, dist)
1229         if idx_film == 0:
1230             idx_film = 0
1231         elif idx_film == len(x):
1232             idx_film = len(x)-1
1233         else:
1234             if abs(x[idx_film-1]-dist) < abs(x[idx_film]-dist):
1235                 idx_film = idx_film-1
1236             else:
1237                 idx_film = idx_film
1238         if idx_doseplan == 0:
1239             idx_doseplan = 0
1240         elif idx_doseplan == len(y):
1241             idx_doseplan = len(y)-1
1242         else:
1243             if abs(y[idx_doseplan-1]-dist) < abs(y[idx_doseplan]-
1244             dist):
1245                 idx_doseplan = idx_doseplan-1
1246             else:
1247                 idx_doseplan = idx_doseplan

1248     idx_film = int(np.round(idx_film))
1249     if idx_film < 0:
1250         idx_film = 0
1251     if idx_film >= len(plot_film):
1252         idx_film = len(plot_film) - 1

```

```

1254         #if Globals.profiles_dataset.doseplan.PixelSpacing == [1,
1255             1]:
1256             #    idx_doseplan = int(np.round(idx_doseplan/1))
1257             #elif Globals.profiles_dataset.doseplan.PixelSpacing ==
1258             [2, 2]:
1259                 #    idx_doseplan = int(np.round(idx_doseplan/2))
1260                 ##else:
1261                 #    idx_doseplan = np.round(idx_doseplan/3)
1262                 idx_doseplan = int(np.round(idx_doseplan))
1263                 if idx_doseplan < 0:
1264                     idx_doseplan = 0
1265                 if idx_doseplan >= len(plot_doseplan):
1266                     idx_doseplan = len(plot_doseplan) - 1
1267
1268             match_text = "\tGraph match: \t"
1269             match = str(np.round(min(plot_film[idx_film],
1270             plot_doseplan[idx_doseplan])/max(plot_film[idx_film], plot_doseplan[
1271             idx_doseplan])*100, 2)) + "\n"
1272             distance_text = "Distance:\t"
1273             dose_text = "Dose: \t"
1274             rel_target_dose_text = "Relative to target dose: \t"
1275             rel_mx_dose_ROI_text = "Relative to max dose in ROI: \n"
1276             distance = str(np.round(dist,2)) + "\n"
1277             film = "FILM: \t"
1278             dose_film = str(np.round(plot_film[idx_film],2)) + "\t"
1279             rel_target_dose_film = str(np.round(100*plot_film[idx_film]/
1280             Globals.max_dose_doseplan,2)) + "\t\t\t"
1281             rel_mx_dose_ROI_film = str(np.round(100*plot_film[idx_film]/
1282             np.max(plot_film),2)) + "\n"
1283             doseplan = "DOSEPLAN: \t"
1284             dose_doseplan = str(np.round(plot_doseplan[idx_doseplan],
1285             2)) + "\t"
1286             rel_target_dose_doseplan = str(np.round(100*plot_doseplan[
1287             idx_doseplan]/Globals.max_dose_doseplan,2)) + "\t\t\t"
1288             rel_mx_dose_ROI_doseplan = str(np.round(100*plot_doseplan[
1289             idx_doseplan]/np.max(plot_doseplan),2))
1290             notation = match_text + distance_text + dose_text,
1291             rel_target_dose_text + rel_mx_dose_ROI_text +
1292                 film + dose_film + rel_target_dose_film +
1293                 rel_mx_dose_ROI_film +
1294                 doseplan + dose_doseplan +
1295             rel_target_dose_doseplan + rel_mx_dose_ROI_doseplan
1296
1297             children = listBox.get_children()
1298             for item in children:
1299                 listBox.delete(item)
1300             lst = [['Film: ', match, distance, dose_film,
1301             rel_mx_dose_ROI_film, rel_target_dose_film], \
1302                 ['Doseplan: ', match, distance, dose_doseplan,
1303                 rel_mx_dose_ROI_doseplan, rel_target_dose_doseplan]]
1304             for i, (name, m, dis, d, rdROI, rdTarget) in enumerate(lst):
1305                 listBox.insert("", "end", values=(name, m, dis, d,
1306                 rdROI, rdTarget))
1307                 y_min = max(plot_film[idx_film], plot_doseplan[
1308                 idx_doseplan]) - 0.3 * max(np.max(plot_film), np.max(plot_doseplan))
1309                 if y_min < 0:

```

```

1294             y_min = 0
1295             y_max = max(plot_film[idx_film], plot_doseplan[
1296                 idx_doseplan])+0.3*max(np.max(plot_film), np.max(plot_doseplan))
1297             if y_max > max(np.max(plot_film), np.max(plot_doseplan)):
1298                 y_max = max(np.max(plot_film), np.max(plot_doseplan))
1299             v_line.set_xdata(dist)
1300             #v_line.set_xlim(y_min,y_max)
1301             #v_line.set_ymax = y
1302             #v_line.set_ymax = y_max # =
1303             #v_line = a.axvline(x=dist, ymin=0, ymax=40, c='gray')
1304             v_line.set_visible(True)
1305             fig.canvas.draw_idle()

1306     def freezeData(event):
1307         fig.canvas.mpl_disconnect(cid)
1308         v_line.set_visible(False)
1309         fig.canvas.draw_idle()
1310     def startData(event):
1311         fig.canvas.mpl_disconnect(cid2)
1312         fig.canvas.mpl_disconnect(cid3)
1313         draw(line_orient, dataset_film, dataset_doseplan)
1314

1316         cid3 = fig.canvas.mpl_connect('button_press_event',
1317             startData)

1318         cid2 = fig.canvas.mpl_connect('button_press_event',
1319             freezeData)
1320         else:
1321             return

1322         cid3 = None
1323         cid = fig.canvas.mpl_connect('motion_notify_event', mouseMove)
1324         fig.tight_layout()

1326     if even:
1327         draw('d', Globals.profiles_dataset_film_variable_draw, Globals.
1328 profiles_dataset_doseplan_variable_draw)
1329         return

1332     if(Globals.profiles_choice_of_profile_line_type.get() == 'h' and
1333     Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]):
1334         dataset_film = np.zeros(
1335             (Globals.profiles_doseplan_dataset_ROI.shape[0], Globals.
1336 profiles_film_dataset_ROI_red_channel_dose.shape[1]))
1337         for i in range(dataset_film.shape[0]-1):
1338             dataset_film[i,:] = Globals.
1339             profiles_film_dataset_ROI_red_channel_dose[int((i*5)+2),:]
1340             try:
1341                 dataset_film[dataset_film.shape[0]-1,:] = Globals.
1342                 profiles_film_dataset_ROI_red_channel_dose[int((dataset_film.shape
1343 [0]-1)*5+2), :]
1344             except:
1345                 dataset_film[dataset_film.shape[0]-1,:] = \

```

```

1342                               Globals.profiles.film_dataset_ROI_red_channel_dose[Globals
1343                               .profiles.film_dataset_ROI_red_channel_dose.shape[0]-1,:]

1344           line_doseplan = Globals.doseplan_write_image.create_line(0,Globals
1345           .doseplan_write_image_var_x,\n
1346           Globals.doseplan_write_image_width,Globals.
1347           doseplan_write_image_var_x, fill='red')
1348           line_film_dosemap = Globals.film_dose_write_image.create_line(0,
1349           Globals.doseplan_write_image_var_x,\n
1350           Globals.doseplan_write_image_width,Globals.
1351           doseplan_write_image_var_x, fill='red')
1352           line_film = Globals.film_write_image.create_line(0,Globals.
1353           doseplan_write_image_var_x,\n
1354           Globals.doseplan_write_image_width,Globals.
1355           doseplan_write_image_var_x, fill='red')

1356           Globals.profiles_lines.append(line_doseplan)
1357           Globals.profiles_lines.append(line_film_dosemap)
1358           Globals.profiles_lines.append(line_film)

1359       def up_button_pressed(event):
1360           temp_x = Globals.doseplan_write_image_var_x - 5
1361           if(temp_x < 0):
1362               #Outside the frame
1363               return
1364           #inside the frame
1365           Globals.doseplan_write_image_var_x = temp_x
1366           Globals.profiles_coordinate_in_dataset = int(temp_x/5)
1367           Globals.doseplan_write_image.coords(line_doseplan,0,Globals.
1368           doseplan_write_image_var_x,\n
1369               Globals.doseplan_write_image_width,Globals.
1370           doseplan_write_image_var_x)
1371           Globals.film_dose_write_image.coords(line_film_dosemap , 0,
1372           Globals.doseplan_write_image_var_x,\n
1373               Globals.doseplan_write_image_width,Globals.
1374           doseplan_write_image_var_x)
1375           Globals.film_write_image.coords(line_film , 0,Globals.
1376           doseplan_write_image_var_x,\n
1377               Globals.doseplan_write_image_width,Globals.
1378           doseplan_write_image_var_x)
1379           draw('h', dataset_film , Globals.profiles_doseplan_dataset_ROI)

1380       def down_button_pressed(event):
1381           temp_x = Globals.doseplan_write_image_var_x + 5
1382           if(temp_x >= Globals.doseplan_write_image_height):
1383               #Outside the frame
1384               return
1385           #Inside the frame
1386           Globals.profiles_coordinate_in_dataset = int(temp_x/5)
1387           Globals.doseplan_write_image_var_x = temp_x
1388           Globals.doseplan_write_image.coords(line_doseplan,0,Globals.
1389           doseplan_write_image_var_x,\n
1390               Globals.doseplan_write_image_width,Globals.
1391           doseplan_write_image_var_x)
1392           Globals.film_dose_write_image.coords(line_film_dosemap , 0,
1393           Globals.doseplan_write_image_var_x,\n

```

```

1382             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)

1386

1388     Globals.form.bind("<Up>", up_button_pressed)
    Globals.form.bind("<Down>", down_button_pressed)

1390
1392     if Globals.profiles_first_time_in_drawProfiles:
        Globals.profiles_first_time_in_drawProfiles = False
        draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)
1394     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,:]

1404

1406     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')
1408     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')
1410     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')

1412     Globals.profiles_lines.append(line_doseplan)
1414     Globals.profiles_lines.append(line_film_dosemap)
    Globals.profiles_lines.append(line_film)

1416     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

```

```

1424     Globals.profiles_coordinate_in_dataset = int(temp_x/10)
1425     Globals.doseplan_write_image.coords(line_doseplan,0,Globals.
1426 doseplan_write_image_var_x,\_
1427     Globals.doseplan_write_image_width,Globals.
1428 doseplan_write_image_var_x)
1429     Globals.film_dose_write_image.coords(line_film_dosemap,0,
1430 Globals.doseplan_write_image_var_x,\_
1431     Globals.doseplan_write_image_width,Globals.
1432 doseplan_write_image_var_x)
1433     Globals.film_write_image.coords(line_film,0,Globals.
1434 doseplan_write_image_var_x,\_
1435     Globals.doseplan_write_image_width,Globals.
1436 doseplan_write_image_var_x)
1437     draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)
1438
1439     def down_button_pressed(event):
1440         temp_x = Globals.doseplan_write_image_var_x + 10
1441         if(temp_x >= Globals.doseplan_write_image_height):
1442             #Outside the frame
1443             return
1444         #Inside the frame
1445         Globals.profiles_coordinate_in_dataset = int(temp_x/10)
1446         Globals.doseplan_write_image_var_x = temp_x
1447         Globals.doseplan_write_image.coords(line_doseplan,0,Globals.
1448 doseplan_write_image_var_x,\_
1449     Globals.doseplan_write_image_width,Globals.
1450 doseplan_write_image_var_x)
1451         Globals.film_dose_write_image.coords(line_film_dosemap,0,
1452 Globals.doseplan_write_image_var_x,\_
1453     Globals.doseplan_write_image_width,Globals.
1454 doseplan_write_image_var_x)
1455         Globals.film_write_image.coords(line_film,0,Globals.
1456 doseplan_write_image_var_x,\_
1457     Globals.doseplan_write_image_width,Globals.
1458 doseplan_write_image_var_x)
1459         draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)
1460
1461         Globals.form.bind("<Up>", up_button_pressed)
1462         Globals.form.bind("<Down>", down_button_pressed)
1463
1464         if Globals.profiles_first_time_in_drawProfiles:
1465             Globals.profiles_first_time_in_drawProfiles = False
1466             draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)
1467
1468         elif(Globals.profiles_choice_of_profile_line_type.get() == 'h' and
1469 Globals.profiles_dataset_doseplan.PixelSpacing==[3, 3]):
1470             dataset_film = np.zeros(\_
1471             (Globals.profiles_doseplan_dataset_ROI.shape[0], Globals.
1472 profiles_film_dataset_ROI_red_channel_dose.shape[1]))
1473             for i in range(dataset_film.shape[0]-1):
1474                 dataset_film[i,:] = Globals.
1475 profiles_film_dataset_ROI_red_channel_dose[int((i*15)+7),:]
1476             try:
1477                 dataset_film[dataset_film.shape[0]-1,:] = Globals.
1478 profiles_film_dataset_ROI_red_channel_dose[int((dataset_film.shape
1479 [0]-1)*15+7), :]
1480             except:

```

```

1464         dataset_film[dataset_film.shape[0]-1,:] = \
1465             Globals.profiles_film_dataset_ROI_red_channel_dose[Globals.
1466             .profiles_film_dataset_ROI_red_channel_dose.shape[0]-1,:]
1467
1468     line_doseplan = Globals.doseplan_write_image.create_line(0,Globals
1469     .doseplan_write_image_var_x,\n
1470         Globals.doseplan_write_image_width,Globals.
1471         doseplan_write_image_var_x, fill='red')
1472     line_film_dosemap = Globals.film_dose_write_image.create_line(0,
1473     Globals.doseplan_write_image_var_x,\n
1474         Globals.doseplan_write_image_width,Globals.
1475         doseplan_write_image_var_x, fill='red')
1476     line_film = Globals.film_write_image.create_line(0,Globals.
1477     doseplan_write_image_var_x,\n
1478         Globals.doseplan_write_image_width,Globals.
1479         doseplan_write_image_var_x, fill='red')
1480
1481     Globals.profiles_lines.append(line_doseplan)
1482     Globals.profiles_lines.append(line_film_dosemap)
1483     Globals.profiles_lines.append(line_film)
1484
1485     def up_button_pressed(event):
1486         temp_x = Globals.doseplan_write_image_var_x - 15
1487         if(temp_x < 0):
1488             #Outside the frame
1489             return
1490         #inside the frame
1491         Globals.doseplan_write_image_var_x = temp_x
1492         Globals.profiles_coordinate_in_dataset = int(temp_x/15)
1493         Globals.doseplan_write_image.coords(line_doseplan,0,Globals.
1494         doseplan_write_image_var_x,\n
1495             Globals.doseplan_write_image_width,Globals.
1496             doseplan_write_image_var_x)
1497         Globals.film_dose_write_image.coords(line_film_dosemap, 0,
1498         Globals.doseplan_write_image_var_x,\n
1499             Globals.doseplan_write_image_width,Globals.
1500             doseplan_write_image_var_x)
1501         Globals.film_write_image.coords(line_film, 0,Globals.
1502             doseplan_write_image_var_x,\n
1503                 Globals.doseplan_write_image_width,Globals.
1504                 doseplan_write_image_var_x)
1505
1506     def down_button_pressed(event):
1507         temp_x = Globals.doseplan_write_image_var_x + 15
1508         if(temp_x >= Globals.doseplan_write_image_height):
1509             #Outside the frame
1510             return
1511         #Inside the frame
1512         Globals.profiles_coordinate_in_dataset = int(temp_x/15)
1513         Globals.doseplan_write_image_var_x = temp_x
1514         Globals.doseplan_write_image.coords(line_doseplan,0,Globals.
1515         doseplan_write_image_var_x,\n
1516             Globals.doseplan_write_image_width,Globals.
1517             doseplan_write_image_var_x)

```

```

1506     Globals.film_dose_write_image.coords(line_film_dosemap,0,
1507     Globals.doseplan_write_image_var_x,\n
1508     Globals.doseplan_write_image_width,Globals.
doseplan_write_image_var_x)
1509     Globals.film_write_image.coords(line_film,0,Globals.
doseplan_write_image_var_x,\n
1510     Globals.doseplan_write_image_width,Globals.
doseplan_write_image_var_x)
1511     draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)
1512
1513     Globals.form.bind("<Up>", up_button_pressed)
1514     Globals.form.bind("<Down>", down_button_pressed)
1515
1516     if Globals.profiles_first_time_in_drawProfiles:
1517         Globals.profiles_first_time_in_drawProfiles = False
1518         draw('h', dataset_film, Globals.profiles_doseplan_dataset_ROI)
1519
1520     elif Globals.profiles_choice_of_profile_line_type.get() == 'v' and
1521     Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]:
1522         dataset_film = np.zeros(\n
1523             (Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0],
1524             Globals.profiles_doseplan_dataset_ROI.shape[1]))
1525         for i in range(dataset_film.shape[1]-1):
1526             dataset_film[:,i] = Globals.
1527             profiles_film_dataset_ROI_red_channel_dose[:, int((i*5)+2)]
1528             try:
1529                 dataset_film[:,dataset_film.shape[1]-1] = Globals.
1530                 profiles_film_dataset_ROI_red_channel_dose[:, int((dataset_film.shape
1531                 [1]-1)*5+2)]
1532             except:
1533                 dataset_film[:,dataset_film.shape[1]-1] = \
1534                     Globals.profiles_film_dataset_ROI_red_channel_dose[:,,
1535                     Globals.profiles_film_dataset_ROI_red_channel_dose.shape[1]-1]
1536
1537
1538     line_doseplan = Globals.doseplan_write_image.create_line(Globals.
doseplan_write_image_var_y, 0,\n
1539     Globals.doseplan_write_image_var_y, Globals.
doseplan_write_image_height, fill='red')
1540     line_film_dosemap = Globals.film_dose_write_image.create_line(
1541     Globals.doseplan_write_image_var_y,0,\n
1542     Globals.doseplan_write_image_var_y, Globals.
doseplan_write_image_height, fill='red')
1543     line_film = Globals.film_write_image.create_line(Globals.
doseplan_write_image_var_y,0,\n
1544     Globals.doseplan_write_image_var_y, Globals.
doseplan_write_image_height, fill='red')
1545
1546     Globals.profiles_lines.append(line_doseplan)
1547     Globals.profiles_lines.append(line_film_dosemap)
1548     Globals.profiles_lines.append(line_film)
1549
1550     def left_button_pressed(event):
1551         temp_y = Globals.doseplan_write_image_var_y - 5
1552         if(temp_y < 0):
1553             #Outside the frame
1554             return

```

```

1546     #inside the frame
1547     Globals.doseplan_write_image_var_y = temp_y
1548     Globals.profiles_coordinate_in_dataset = int(temp_y/5)
1549     Globals.doseplan_write_image.coords(line_doseplan ,Globals.
1550     doseplan_write_image_var_y , 0,\_
1551         Globals.doseplan_write_image_var_y , Globals.
1552         doseplan_write_image_height)
1553         Globals.film_dose_write_image.coords(line_film_dosemap ,
1554     Globals.doseplan_write_image_var_y , 0,\_
1555         Globals.doseplan_write_image_var_y , Globals.
1556         doseplan_write_image_height)
1557         Globals.film_write_image.coords(line_film , Globals.
1558     doseplan_write_image_var_y , 0,\_
1559         Globals.doseplan_write_image_var_y , Globals.
1560         doseplan_write_image_height)
1561         draw('v' , dataset_film , Globals.profiles_doseplan_dataset_ROI)

1562     def right_button_pressed(event):
1563         temp_y = Globals.doseplan_write_image_var_y + 5
1564         if(temp_y >= Globals.doseplan_write_image_width):
1565             #Outside the frame
1566             return
1567         #Inside the frame
1568         Globals.profiles_coordinate_in_dataset = int(temp_y/5)
1569         Globals.doseplan_write_image_var_y = temp_y
1570         Globals.doseplan_write_image.coords(line_doseplan ,Globals.
1571         doseplan_write_image_var_y , 0,\_
1572             Globals.doseplan_write_image_var_y , Globals.
1573             doseplan_write_image_height)
1574             Globals.film_dose_write_image.coords(line_film_dosemap ,Globals.
1575         .doseplan_write_image_var_y , 0,\_
1576             Globals.doseplan_write_image_var_y , Globals.
1577             doseplan_write_image_height)
1578             Globals.film_write_image.coords(line_film , Globals.
1579         doseplan_write_image_var_y , 0,\_
1580             Globals.doseplan_write_image_var_y , Globals.
1581             doseplan_write_image_height)
1582             draw('v' , dataset_film , Globals.profiles_doseplan_dataset_ROI)

1583
1584     Globals.form.bind("<Left>" , left_button_pressed)
1585     Globals.form.bind("<Right>" , right_button_pressed)

1586     if Globals.profiles_first_time_in_drawProfiles :
1587         Globals.profiles_first_time_in_drawProfiles = False
1588         draw('v' , dataset_film , Globals.profiles_doseplan_dataset_ROI)

1589     elif(Globals.profiles_choice_of_profile_line_type.get() == 'v' and
1590     Globals.profiles_dataset_doseplan.PixelSpacing == [2, 2]):
1591         dataset_film = np.zeros(\_
1592             (Globals.profiles_film_dataset_ROI_red_channel_dose.shape[0],
1593             Globals.profiles_doseplan_dataset_ROI.shape[1]))
1594         for i in range(dataset_film.shape[1]-1):
1595             dataset_film[:,i] = Globals.
1596             profiles_film_dataset_ROI_red_channel_dose[:,int((i*10)+5)]
1597             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]

1592
    line_doseplan = Globals.doseplan_write_image.create_line(Globals.
doseplan_write_image_var_y , 0,\n
        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,\n
        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,\n
        Globals.doseplan_write_image_var_y , Globals.
doseplan_write_image_height , fill='red')

1600    Globals.profiles_lines.append(line_doseplan)
    Globals.profiles_lines.append(line_film_dosemap)
    Globals.profiles_lines.append(line_film)

1604    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,\n
            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,\n
            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,\n
            Globals.doseplan_write_image_var_y , Globals.
doseplan_write_image_height)
        draw('v' , dataset_film , Globals.profiles_doseplan_dataset_ROI)

1620    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

```

```

1628     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)
1630     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)
1632     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)
1634     draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)

1636
1638     Globals.form.bind("<Left>", left_button_pressed)
     Globals.form.bind("<Right>", right_button_pressed)

1640     if Globals.profiles_first_time_in_drawProfiles:
     Globals.profiles_first_time_in_drawProfiles = False
     draw('v', dataset_film, Globals.profiles_doseplan_dataset_ROI)

1642
1644     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]

1654
1656     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')
1658     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')
1660     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')

1662     Globals.profiles_lines.append(line_doseplan)
1664     Globals.profiles_lines.append(line_film_dosemap)
     Globals.profiles_lines.append(line_film)

```

```

1668     def left_button_pressed(event):
1669         temp_y = Globals.doseplan_write_image_var_y - 15
1670         if(temp_y < 0):
1671             #Outside the frame
1672             return
1673         #inside the frame
1674         Globals.doseplan_write_image_var_y = temp_y
1675         Globals.profiles_coordinate_in_dataset = int(temp_y/15)
1676         Globals.doseplan_write_image.coords(line_doseplan ,Globals.
doseplan_write_image_var_y , 0,\n
1677             Globals.doseplan_write_image_var_y , Globals.
doseplan_write_image_height)
1678         Globals.film_dose_write_image.coords(line_film_dosemap ,
1679         Globals.doseplan_write_image_var_y , 0,\n
1680             Globals.doseplan_write_image_var_y , Globals.
doseplan_write_image_height)
1681         Globals.film_write_image.coords(line_film , Globals.
doseplan_write_image_var_y , 0,\n
1682             Globals.doseplan_write_image_var_y , Globals.
doseplan_write_image_height)
1683         draw('v' , dataset_film , Globals.profiles_doseplan_dataset_ROI)

1684     def right_button_pressed(event):
1685         temp_y = Globals.doseplan_write_image_var_y + 15
1686         if(temp_y >= Globals.doseplan_write_image_width):
1687             #Outside the frame
1688             return
1689         #Inside the frame
1690         Globals.profiles_coordinate_in_dataset = int(temp_y/15)
1691         Globals.doseplan_write_image_var_y = temp_y
1692         Globals.doseplan_write_image.coords(line_doseplan ,Globals.
doseplan_write_image_var_y , 0,\n
1693             Globals.doseplan_write_image_var_y , Globals.
doseplan_write_image_height)
1694         Globals.film_dose_write_image.coords(line_film_dosemap ,Globals.
doseplan_write_image_var_y , 0,\n
1695             Globals.doseplan_write_image_var_y , Globals.
doseplan_write_image_height)
1696         Globals.film_write_image.coords(line_film , Globals.
doseplan_write_image_var_y , 0,\n
1697             Globals.doseplan_write_image_var_y , Globals.
doseplan_write_image_height)
1698         draw('v' , dataset_film , Globals.profiles_doseplan_dataset_ROI)

1699

1700     Globals.form.bind("<Left>" , left_button_pressed)
1701     Globals.form.bind("<Right>" , right_button_pressed)

1702     if Globals.profiles_first_time_in_drawProfiles:
1703         Globals.profiles_first_time_in_drawProfiles = False
1704         draw('v' , dataset_film , Globals.profiles_doseplan_dataset_ROI)
1705     elif(Globals.profiles_choice_of_profile_line_type.get() == 'd' and
1706     Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]):
1707         start_point = [0,0]
1708         def mousePushed(event):
1709             start_point = [event.y, event.x]
1710             if not len(Globals.profiles_lines)==0:

```

```

1712             Globals.doseplan_write_image.delete(Globals.profiles_lines
1713                 [0])
1714             Globals.film_dose_write_image.delete(Globals.
1715                 profiles_lines[1])
1716             Globals.film_write_image.delete(Globals.profiles_lines[2])
1717             Globals.profiles_lines = []
1718
1719             line_doseplan = Globals.doseplan_write_image.create_line(
1720                 start_point[1], start_point[0], start_point[1], start_point[0], fill='
1721                     red')
1722             line_film_dosemap = Globals.film_dose_write_image.create_line(
1723                 start_point[1], start_point[0], start_point[1], start_point[0], fill='
1724                     red')
1725             line_film = Globals.film_write_image.create_line(start_point
1726                 [1], start_point[0], start_point[1], start_point[0], fill='red')
1727
1728             Globals.profiles_lines.append(line_doseplan)
1729             Globals.profiles_lines.append(line_film_dosemap)
1730             Globals.profiles_lines.append(line_film)
1731
1732             def mouseMoving(event):
1733                 Globals.doseplan_write_image.coords(line_doseplan ,
1734                     start_point[1], start_point[0], event.x, event.y)
1735                 Globals.film_dose_write_image.coords(line_film_dosemap ,
1736                     start_point[1], start_point[0], event.x, event.y)
1737                 Globals.film_write_image.coords(line_film , start_point[1],
1738                     start_point[0], event.x, event.y)
1739
1740
1741             Globals.film_dose_write_image.bind("<B1-Motion>", mouseMoving)
1742
1743             def mouseReleased(event):
1744                 Globals.end_point = [event.y, event.x]
1745                 Globals.doseplan_write_image.coords(line_doseplan ,
1746                     start_point[1], start_point[0], event.x, event.y)
1747                 Globals.film_dose_write_image.coords(line_film_dosemap ,
1748                     start_point[1], start_point[0], event.x, event.y)
1749                 Globals.film_write_image.coords(line_film , start_point[1],
1750                     start_point[0], event.x, event.y)
1751                 Globals.profiles_line_coords_film = getCoordsInRandomLine(
1752                     start_point[1], start_point[0], Globals.end_point[1], Globals.
1753                     end_point[0])
1754                 Globals.profiles_line_coords_doseplan =
1755                     getCoordsInRandomLine(int(start_point[1]/5), int(start_point[0]/5) , \
1756                         int(Globals.end_point[1]/5), int(Globals.end_point
1757                             [0]/5))
1758                 Globals.profiles_dataset_film_variable_draw = np.zeros(len
1759                     (Globals.profiles_line_coords_film))
1760                 Globals.profiles_dataset_doseplan_variable_draw=np.zeros(
1761                     len(Globals.profiles_line_coords_doseplan))
1762
1763
1764                 for i in range(len(Globals.
1765                     profiles_dataset_film_variable_draw)):
1766                     coord = Globals.profiles_line_coords_film[i]
1767                     try:

```

```

1748         Globals.profiles_dataset_film_variable_draw[i] =
1749             Globals.profiles_film_dataset_ROI_red_channel_dose[coord[0]-1, coord
1750                 [1]-1]
1751             except:
1752                 return
1753             for i in range(len(Globals.
1754     profiles_dataset_doesplan_variable_draw)):
1755                 coord = Globals.profiles_line_coords_doseplan[i]
1756                 try:
1757                     Globals.profiles_dataset_doesplan_variable_draw[i]
1758 = Globals.profiles_doseplan_dataset_ROI[coord[0]-1, coord[1]-1]
1759                 except:
1760                     return
1761                 draw('d', Globals.profiles_dataset_film_variable_draw,
1762     Globals.profiles_dataset_doesplan_variable_draw)
1763
1764         Globals.film_dose_write_image.bind("<ButtonRelease-1>",
1765     mouseReleased)
1766         Globals.film_dose_write_image.bind("<Button-1>", mousePushed)
1767
1768     elif(Globals.profiles_choice_of_profile_line_type.get() == 'd' and
1769     Globals.profiles_dataset_doseplan.PixelSpacing == [2, 2]):
1770         start_point = [0,0]
1771         def mousePushed(event):
1772             start_point = [event.y, event.x]
1773             if not len(Globals.profiles_lines)==0:
1774                 Globals.doseplan_write_image.delete(Globals.profiles_lines
1775 [0])
1776                 Globals.film_dose_write_image.delete(Globals.
1777 profiles_lines[1])
1778                 Globals.film_write_image.delete(Globals.profiles_lines[2])
1779                 Globals.profiles_lines = []
1780
1781             line_doseplan = Globals.doseplan_write_image.create_line(
1782     start_point[1], start_point[0], start_point[1], start_point[0], fill='
1783 red')
1784             line_film_dosemap = Globals.film_dose_write_image.create_line(
1785     start_point[1], start_point[0], start_point[1], start_point[0], fill='
1786 red')
1787             line_film = Globals.film_write_image.create_line(start_point
1788 [1], start_point[0], start_point[1], start_point[0], fill='red')
1789
1790             Globals.profiles_lines.append(line_doseplan)
1791             Globals.profiles_lines.append(line_film_dosemap)
1792             Globals.profiles_lines.append(line_film)
1793
1794         def mouseMoving(event):
1795             Globals.doseplan_write_image.coords(line_doseplan,
1796     start_point[1], start_point[0], event.x, event.y)
1797             Globals.film_dose_write_image.coords(line_film_dosemap,
1798     start_point[1], start_point[0], event.x, event.y)
1799             Globals.film_write_image.coords(line_film, start_point[1],
1800     start_point[0], event.x, event.y)

```

```

1788     Globals.film_dose_write_image.bind("<B1-Motion>", mouseMoving)
1790
1791     def mouseReleased(event):
1792         Globals.end_point = [event.y, event.x]
1793         Globals.doseplan_write_image.coords(line_doseplan,
1794             start_point[1], start_point[0], event.x, event.y)
1795         Globals.film_dose_write_image.coords(line_film_dosemap,
1796             start_point[1], start_point[0], event.x, event.y)
1797         Globals.film_write_image.coords(line_film, start_point[1],
1798             start_point[0], event.x, event.y)
1799         Globals.profiles_line_coords_film = getCoordsInRandomLine(
1800             start_point[1], start_point[0], Globals.end_point[1], Globals.
1801             end_point[0])
1802         Globals.profiles_line_coords_doseplan =
1803             getCoordsInRandomLine(int(start_point[1]/10), int(start_point[0]/10),
1804             \
1805                 int(Globals.end_point[1]/10), int(Globals.end_point
1806                 [0]/10))
1807         Globals.profiles_dataset_film_variable_draw = np.zeros(len
1808             (Globals.profiles_line_coords_film))
1809         Globals.profiles_dataset_doseplan_variable_draw=np.zeros(
1810             len(Globals.profiles_line_coords_doseplan))
1811
1812         for i in range(len(Globals.
1813             profiles_dataset_film_variable_draw)):
1814             coord = Globals.profiles_line_coords_film[i]
1815             try:
1816                 Globals.profiles_dataset_film_variable_draw[i] = \
1817                     Globals.
1818             profiles_film_dataset_ROI_red_channel_dose[coord[0]-1, coord[1]-1]
1819             except:
1820                 return
1821
1822         for i in range(len(Globals.
1823             profiles_dataset_doseplan_variable_draw)):
1824             try:
1825                 Globals.profiles_dataset_doseplan_variable_draw[i] =
1826                     \
1827                         Globals.profiles_doseplan_dataset_ROI[int(
1828                             Globals.profiles_line_coords_doseplan[i][1])-1, int(Globals.
1829                             profiles_line_coords_doseplan[i][0])-1]
1830             except:
1831                 return
1832             draw('d', Globals.profiles_dataset_film_variable_draw,
1833                 Globals.profiles_dataset_doseplan_variable_draw)
1834
1835             Globals.film_dose_write_image.bind("<ButtonRelease-1>",
1836                 mouseReleased)
1837             Globals.film_dose_write_image.bind("<Button-1>", mousePushed)
1838             elif(Globals.profiles_choice_of_profile_line_type.get() == 'd' and
1839                 Globals.profiles_dataset_doseplan.PixelSpacing == [3, 3]):
1840                 start_point = [0,0]
1841                 def mousePushed(event):
1842                     start_point = [event.y, event.x]
1843                     if not len(Globals.profiles_lines)==0:
1844                         Globals.doseplan_write_image.delete(Globals.profiles_lines
1845                         [0])

```

```

1826         Globals.film_dose_write_image.delete(Globals.
profiles_lines[1])
1827         Globals.film_write_image.delete(Globals.profiles_lines[2])
1828         Globals.profiles_lines = []
1829
1830         line_doseplan = Globals.doseplan_write_image.create_line(
start_point[1], start_point[0], start_point[1], start_point[0], fill='
red')
1831         line_film_dosemap = Globals.film_dose_write_image.create_line(
start_point[1], start_point[0], start_point[1], start_point[0], fill='
red')
1832         line_film = Globals.film_write_image.create_line(start_point
[1], start_point[0], start_point[1], start_point[0], fill='red')
1833
1834         Globals.profiles_lines.append(line_doseplan)
1835         Globals.profiles_lines.append(line_film_dosemap)
1836         Globals.profiles_lines.append(line_film)
1837
1838     def mouseMoving(event):
1839         Globals.doseplan_write_image.coords(line_doseplan,
1840         start_point[1], start_point[0], event.x, event.y)
1841         Globals.film_dose_write_image.coords(line_film_dosemap,
1842         start_point[1], start_point[0], event.x, event.y)
1843         Globals.film_write_image.coords(line_film, start_point[1],
1844         start_point[0], event.x, event.y)
1845
1846
1847     Globals.film_dose_write_image.bind("<B1-Motion>", mouseMoving)
1848
1849     def mouseReleased(event):
1850         Globals.end_point = [event.y, event.x]
1851         Globals.doseplan_write_image.coords(line_doseplan,
1852         start_point[1], start_point[0], event.x, event.y)
1853         Globals.film_dose_write_image.coords(line_film_dosemap,
1854         start_point[1], start_point[0], event.x, event.y)
1855         Globals.film_write_image.coords(line_film, start_point[1],
1856         start_point[0], event.x, event.y)
1857         Globals.profiles_line_coords_film = getCoordsInRandomLine(
1858         start_point[1], start_point[0], Globals.end_point[1], Globals.
1859         end_point[0])
1860
1861         Globals.profiles_line_coords_doseplan =
1862         getCoordsInRandomLine(int(start_point[1]/15), int(start_point[0]/15),
1863         \
1864             int(Globals.end_point[1]/15), int(Globals.end_point
1865             [0]/15))
1866         Globals.profiles_dataset_film_variable_draw = np.zeros(len
1867         (Globals.profiles_line_coords_film))
1868         Globals.profiles_dataset_doseplan_variable_draw=np.zeros(
1869         len(Globals.profiles_line_coords_doseplan))
1870
1871         for i in range(len(Globals.
1872 profiles_dataset_film_variable_draw)):
1873             coord = Globals.profiles_line_coords_film[i]
1874             try:
1875                 Globals.profiles_dataset_film_variable_draw[i] =
1876                 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("<Right>")
        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], \

```

```

1908     Globals . profiles_film_variable_ROI_coords[2]-1, Globals .
profiles_film_variable_ROI_coords[3]-1]
1910     Globals . profiles_film_dataset_ROI_red_channel_dose = \
        Globals . profiles_film_dataset_red_channel_dose \
        [ Globals . profiles_film_variable_ROI_coords[0]:Globals .
1912     profiles_film_variable_ROI_coords[1], \
        Globals . profiles_film_variable_ROI_coords[2]:Globals .
profiles_film_variable_ROI_coords[3]]
1914     Globals . profiles_first_time_in_drawProfiles = True
1916     if line_orient == 'd':
1918         for i in range(len(Globals . profiles_dataset_film_variable_draw)):
1920             coord = Globals . profiles_line_coords_film[i]
1922             try:
1924                 Globals . profiles_dataset_film_variable_draw[i] = Globals .
profiles_film_dataset_ROI_red_channel_dose[coord[0]-1, coord[1]-1]
1926             except:
1928                 return
1930             for i in range(len(Globals . profiles_dataset_doesplan_variable_draw
)):
1932                 try:
1934                     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]
1936                 except:
1938                     return
1940             drawProfiles(True)
1942     else:
1944         drawProfiles(False)

1948 def adjustROIRight(line_orient):
1950     if not line_orient == 'd':
1952         Globals . doseplan_write_image . delete(Globals . profiles_lines[0])
1954         Globals . film_dose_write_image . delete(Globals . profiles_lines[1])
1956         Globals . film_write_image . delete(Globals . profiles_lines[2])
1958     if(Globals . profiles_film_variable_ROI_coords[3]+1 > Globals .
1960     profiles_film_dataset_red_channel_dose . shape[1]):
1962         messagebox.showwarning("Warning", "Reached end of film \n(Code:
1964     adjustROIRight)")
1966     return
1968     Globals . profiles_film_variable_ROI_coords = \
1970         [ Globals . profiles_film_variable_ROI_coords[0], Globals .
1972         profiles_film_variable_ROI_coords[1], \
1974             Globals . profiles_film_variable_ROI_coords[2]+1, Globals .
1976         profiles_film_variable_ROI_coords[3]+1]
1978     Globals . profiles_film_dataset_ROI_red_channel_dose = \
1980         Globals . profiles_film_dataset_red_channel_dose \
1982         [ Globals . profiles_film_variable_ROI_coords[0]:Globals .
1984         profiles_film_variable_ROI_coords[1], \
1986             Globals . profiles_film_variable_ROI_coords[2]:Globals .
1988         profiles_film_variable_ROI_coords[3]]
1990     Globals . profiles_first_time_in_drawProfiles = True
1992     if line_orient == 'd':
1994         for i in range(len(Globals . profiles_dataset_film_variable_draw)):
1996             coord = Globals . profiles_line_coords_film[i]
1998             try:

```

```

1950           Globals.profiles_dataset_film_variable_draw[i] = Globals.
1951   profiles_film_dataset_ROI_red_channel_dose[coord[0]-1, coord[1]-1]
1952       except:
1953           return
1954   for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
1955       try:
1956           Globals.profiles_dataset_doesplan_variable_draw[i] =
1957   Globals.profiles_doseplan_dataset_ROI[int(Globals.
1958   profiles_line_coords_doseplan[i][0])-1, int(Globals.
1959   profiles_line_coords_doseplan[i][1])-1]
1960       except:
1961           return
1962   drawProfiles(True)
1963 else:
1964     drawProfiles(False)
1965
1966 def adjustROIUp(line_orient):
1967 if not line_orient == 'd':
1968     Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
1969     Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
1970     Globals.film_write_image.delete(Globals.profiles_lines[2])
1971 if (Globals.profiles_film_variable_ROI_coords[0]-1 < 0):
1972     messagebox.showwarning("Warning", "Reached end of film \n(Code:
1973 adjustROIUp)")
1974     return
1975 Globals.profiles_film_variable_ROI_coords = \
1976     [Globals.profiles_film_variable_ROI_coords[0]-1, Globals.
1977     profiles_film_variable_ROI_coords[1]-1, \
1978     Globals.profiles_film_variable_ROI_coords[2], Globals.
1979     profiles_film_variable_ROI_coords[3]]
1980 Globals.profiles_film_dataset_ROI_red_channel_dose = \
1981     Globals.profiles_film_dataset_red_channel_dose \
1982     [Globals.profiles_film_variable_ROI_coords[0]:Globals.
1983     profiles_film_variable_ROI_coords[1], \
1984     Globals.profiles_film_variable_ROI_coords[2]:Globals.
1985     profiles_film_variable_ROI_coords[3]]
1986 Globals.profiles_first_time_in_drawProfiles = True
1987 if line_orient == 'd':
1988     for i in range(len(Globals.profiles_dataset_film_variable_draw)):
1989         coord = Globals.profiles_line_coords_film[i]
1990         try:
1991             Globals.profiles_dataset_film_variable_draw[i] = Globals.
1992   profiles_film_dataset_ROI_red_channel_dose[coord[0]-1, coord[1]-1]
1993         except:
1994             return
1995     for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
1996         try:
1997             Globals.profiles_dataset_doesplan_variable_draw[i] =
1998   Globals.profiles_doseplan_dataset_ROI[int(Globals.
1999   profiles_line_coords_doseplan[i][0])-1, int(Globals.
2000   profiles_line_coords_doseplan[i][1])-1]
2001         except:
2002             return
2003     drawProfiles(True)
2004 else:

```

```

1992     drawProfiles(False)
1993
1994 def adjustROIDown(line_orient):
1995     if not line_orient == 'd':
1996         Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
1997         Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
1998         Globals.film_write_image.delete(Globals.profiles_lines[2])
1999     if(Globals.profiles_film_variable_ROI_coords[1]+1 > Globals.
2000         profiles_film_dataset_red_channel_dose.shape[0]):
2001         messagebox.showwarning("Warning", "Reached end of film \n(Code:
2002         adjustROIDown)")
2003         return
2004     Globals.profiles_film_variable_ROI_coords = \
2005         [Globals.profiles_film_variable_ROI_coords[0]+1, Globals.
2006         profiles_film_variable_ROI_coords[1]+1, \
2007             Globals.profiles_film_variable_ROI_coords[2], Globals.
2008         profiles_film_variable_ROI_coords[3]]
2009     Globals.profiles_film_dataset_ROI_red_channel_dose = \
2010         [Globals.profiles_film_dataset_red_channel_dose\ \
2011             [Globals.profiles_film_variable_ROI_coords[0]:Globals.
2012             profiles_film_variable_ROI_coords[1], \
2013                 Globals.profiles_film_variable_ROI_coords[2]:Globals.
2014             profiles_film_variable_ROI_coords[3]]]
2015     Globals.profiles_first_time_in_drawProfiles = True
2016     if line_orient == 'd':
2017         for i in range(len(Globals.profiles_dataset_film_variable_draw)):
2018             coord = Globals.profiles_line_coords_film[i]
2019             try:
2020                 Globals.profiles_dataset_film_variable_draw[i] = Globals.
2021             profiles_film_dataset_ROI_red_channel_dose[coord[0]-1, coord[1]-1]
2022             except:
2023                 return
2024         for i in range(len(Globals.profiles_dataset_doesplan_variable_draw
2025 )):
2026             try:
2027                 Globals.profiles_dataset_doesplan_variable_draw[i] =
2028             Globals.doseplan_dataset_ROI[int(Globals.
2029             profiles_line_coords_doseplan[i][0])-1, int(Globals.
2030             profiles_line_coords_doseplan[i][1])-1]
2031             except:
2032                 return
2033         drawProfiles(True)
2034     else:
2035         drawProfiles(False)
2036
2037 def returnToOriginalROICoordinates(line_orient):
2038     if not line_orient == 'd':
2039         Globals.doseplan_write_image.delete(Globals.profiles_lines[0])
2040         Globals.film_dose_write_image.delete(Globals.profiles_lines[1])
2041         Globals.film_write_image.delete(Globals.profiles_lines[2])
2042     Globals.profiles_film_variable_ROI_coords = \
2043         [Globals.profiles_ROI_coords[0][1], Globals.profiles_ROI_coords
2044         [2][1], \
2045             Globals.profiles_ROI_coords[0][0], Globals.
2046             profiles_ROI_coords[1][0]]
2047
2048     Globals.profiles_film_dataset_ROI_red_channel_dose = \

```

```

2036     Globals.profiles_film_dataset_red_channel_dose \
2037         [Globals.profiles_film_variable_ROI_coords[0]:Globals.
2038             profiles_film_variable_ROI_coords[1], \
2039                 Globals.profiles_film_variable_ROI_coords[2]:Globals.
2040                     profiles_film_variable_ROI_coords[3]]]
2041             Globals.profiles_first_time_in_drawProfiles = True
2042             if line_orient == 'd':
2043                 for i in range(len(Globals.profiles_dataset_film_variable_draw)):
2044                     coord = Globals.profiles_line_coords_film[i]
2045                     try:
2046                         Globals.profiles_dataset_film_variable_draw[i] = Globals.
2047                             profiles_film_dataset_ROI_red_channel_dose[coord[0]-1, coord[1]-1]
2048                     except:
2049                         return
2050                     for i in range(len(Globals.profiles_dataset_doesplan_variable_draw)):
2051                         Globals.profiles_dataset_doesplan_variable_draw[i] = Globals.
2052                             profiles_doseplan_dataset_ROI[int(Globals.
2053                                 profiles_line_coords_doseplan[i][0])-1, int(Globals.
2054                                     profiles_line_coords_doseplan[i][1])-1]
2055                             drawProfiles(True)
2056             else:
2057                 drawProfiles(False)

2058 def pixel_to_dose(P,a,b,c):
2059     ret = c + b/(P-a)
2060     return ret

2061 def processDoseplan_usingReferencePoint(only_one):
2062     ##### RT Plan #####
2063
2064     #Find each coordinate in mm to isocenter relative to first element in
2065     #doseplan
2066     iso_1 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[0]
2067     - Globals.profiles_dataset_rtplan.BeamSequence[0].ControlPointSequence
2068     [0].IsocenterPosition[0])
2069     iso_2 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[1]
2070     - Globals.profiles_dataset_rtplan.BeamSequence[0].ControlPointSequence
2071     [0].IsocenterPosition[1])
2072     iso_3 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[2]
2073     - Globals.profiles_dataset_rtplan.BeamSequence[0].ControlPointSequence
2074     [0].IsocenterPosition[2])
2075     #Given as [x,y,z] in patient coordinates
2076     Globals.profiles_isocenter_mm = [iso_1, iso_2, iso_3]

2077     #Reads input displacement from phantom on reference point in film
2078     #lateral = Globals.profiles_input_lateral_displacement.get("1.0", 'end
2079     #    -lc')
2080     #vertical = Globals.profiles_input_vertical_displacement.get("1.0", 'end-lc')
2081     #longit = Globals.profiles_input_longitudinal_displacement.get("1.0",
2082     #    'end-lc')
2083     #if(lateral==" "):lateral=0
2084     #if(vertical==" "):vertical=0
2085     #if(longit==" "):longit=0
2086     try:

```

```

2076     Globals.profiles_vertical = int(Globals.profiles_vertical)
2077 except:
2078     messagebox.showerror("Error", "Could not read the vertical
2079     displacements\n (Code: displacements to integer)")
2080     return
2081 try:
2082     Globals.profiles_lateral = int(Globals.profiles_lateral)
2083 except:
2084     messagebox.showerror("Error", "Could not read the lateral
2085     displacements\n (Code: displacements to integer)")
2086     return
2087 try:
2088     Globals.profiles_longitudinal = int(Globals.profiles_longitudinal)
2089 except:
2090     messagebox.showerror("Error", "Could not read the longitudinal
2091     displacements\n (Code: displacements to integer)")
2092     return
2093
2094 lateral = Globals.profiles_lateral
2095 longit = Globals.profiles_longitudinal
2096 vertical = Globals.profiles_vertical
2097 isocenter_px = np.zeros(3)
2098 distance_in_doseplan_ROI_reference_point_px = []
2099 if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
2100     #make isocenter coordinates into pixel values
2101     isocenter_px[0] = np.round(iso_1)
2102     isocenter_px[1] = np.round(iso_2)
2103     isocenter_px[2] = np.round(iso_3)
2104
2105     #find the pixel distance from reference point to ROI corners
2106     distance_in_doseplan_ROI_reference_point_px.append([np.round(
2107         Globals.profiles_distance_reference_point_ROI[0][0]), \
2108             np.round(Globals.profiles_distance_reference_point_ROI[0][1])])
2109     distance_in_doseplan_ROI_reference_point_px.append([np.round(
2110         Globals.profiles_distance_reference_point_ROI[1][0]), \
2111             np.round(Globals.profiles_distance_reference_point_ROI[1][1])])
2112     distance_in_doseplan_ROI_reference_point_px.append([np.round(
2113         Globals.profiles_distance_reference_point_ROI[2][0]), \
2114             np.round(Globals.profiles_distance_reference_point_ROI[2][1])])
2115     distance_in_doseplan_ROI_reference_point_px.append([np.round(
2116         Globals.profiles_distance_reference_point_ROI[3][0]), \
2117             np.round(Globals.profiles_distance_reference_point_ROI[3][1])])
2118
2119     #Input to px
2120     lateral_px = np.round(lateral)
2121     vertical_px = np.round(vertical)
2122     longit_px = np.round(longit)
2123
2124     #displacement to px
2125     doseplan_lateral_displacement_px = np.round(Globals.
2126 profiles_doseplan_lateral_displacement)
2127     doseplan_vertical_displacement_px = np.round(Globals.
2128 profiles_doseplan_vertical_displacement)

```

```

2120     doseplan_longitudinal_displacement_px = np.round(Globals.
2121 profiles_doseplan_longitudianl_displacement)
2122
2123     elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
2124         #make isocenter coordinates into pixel values
2125         isocenter_px[0] = np.round(iso_1/2)
2126         isocenter_px[1] = np.round(iso_2/2)
2127         isocenter_px[2] = np.round(iso_3/2)
2128
2129         #find the pixel distance from reference point to ROI corners
2130         distance_in_doseplan_ROI_reference_point_px.append([np.round((
2131             Globals.profiles_distance_reference_point_ROI[0][0])/2), \
2132             np.round((Globals.profiles_distance_reference_point_ROI[0][1])/2)])
2133
2134         distance_in_doseplan_ROI_reference_point_px.append([np.round((
2135             Globals.profiles_distance_reference_point_ROI[1][0])/2), \
2136             np.round((Globals.profiles_distance_reference_point_ROI[1][1])/2)])
2137
2138         distance_in_doseplan_ROI_reference_point_px.append([np.round((
2139             Globals.profiles_distance_reference_point_ROI[2][0])/2), \
2140             np.round((Globals.profiles_distance_reference_point_ROI[2][1])/2)])
2141
2142         distance_in_doseplan_ROI_reference_point_px.append([np.round((
2143             Globals.profiles_distance_reference_point_ROI[3][0])/2), \
2144             np.round((Globals.profiles_distance_reference_point_ROI[3][1])/2)])
2145
2146
2147         #Input to px
2148         lateral_px = np.round(lateral/2)
2149         vertical_px = np.round(vertical/2)
2150         longit_px = np.round(longit/2)
2151
2152         #displacement to pc
2153         doseplan_lateral_displacement_px = np.round((Globals.
2154             profiles_doseplan_lateral_displacement)/2)
2155         doseplan_vertical_displacement_px = np.round((Globals.
2156             profiles_doseplan_vertical_displacement)/2)
2157         doseplan_longitudinal_displacement_px = np.round((Globals.
2158             profiles_doseplan_longitudianl_displacement)/2)
2159
2160     else:
2161         #make isocenter coordinates into pixel values
2162         isocenter_px[0] = np.round(iso_1/3)
2163         isocenter_px[1] = np.round(iso_2/3)
2164         isocenter_px[2] = np.round(iso_3/3)
2165
2166         #find the pixel distance from reference point to ROI corners
2167         distance_in_doseplan_ROI_reference_point_px.append([np.round((
2168             Globals.profiles_distance_reference_point_ROI[0][0])/3), \
2169             np.round((Globals.profiles_distance_reference_point_ROI[0][1])/3)])
2170
2171         distance_in_doseplan_ROI_reference_point_px.append([np.round((
2172             Globals.profiles_distance_reference_point_ROI[1][0])/3), \
2173             np.round((Globals.profiles_distance_reference_point_ROI[1][1])/3)])
2174
2175         distance_in_doseplan_ROI_reference_point_px.append([np.round((
2176             Globals.profiles_distance_reference_point_ROI[2][0])/3), \

```

```

2160           np.round((Globals.profiles_distance_reference_point_ROI[2][1])\\
2161 /3))
2162     distance_in_doseplan_ROI_reference_point_px.append([np.round((\\
2163 Globals.profiles_distance_reference_point_ROI[3][0])/3),\\
2164   np.round((Globals.profiles_distance_reference_point_ROI[3][1])\\
2165 /3)])
2166
2167     #Input to px
2168     lateral_px = np.round(lateral/3)
2169     vertical_px = np.round(vertical/3)
2170     longit_px = np.round(longit/3)
2171
2172     #displacement to px
2173     doseplan_lateral_displacement_px = np.round((Globals.\\
2174 profiles_doseplan_lateral_displacement)/3)
2175     doseplan_vertical_displacement_px = np.round((Globals.\\
2176 profiles_doseplan_vertical_displacement)/3)
2177     doseplan_longitudinal_displacement_px = np.round((Globals.\\
2178 profiles_doseplan_longitudinal_displacement)/3)
2179
2180     temp_ref_point_doseplan = np.zeros(3)
2181
2182     #Finding reference point in doseplan
2183     if(Globals.profiles_doseplan_patient_position=='HFS'):
2184       temp_ref_point_doseplan[0] = int(isocenter_px[0]+\\
2185 doseplan_lateral_displacement_px - lateral_px)
2186       temp_ref_point_doseplan[1] = int(isocenter_px[1]-\\
2187 doseplan_vertical_displacement_px + vertical_px)
2188       temp_ref_point_doseplan[2] = int(isocenter_px[2]+\\
2189 doseplan_longitudinal_displacement_px - longit_px)
2190     elif(Globals.profiles_doseplan_patient_position=='HFP'):
2191       temp_ref_point_doseplan[0] = isocenter_px[0]-\\
2192 doseplan_lateral_displacement_px+ lateral_px
2193       temp_ref_point_doseplan[1] = isocenter_px[1]+\\
2194 doseplan_vertical_displacement_px - vertical_px
2195       temp_ref_point_doseplan[2] = isocenter_px[2]+\\
2196 doseplan_longitudinal_displacement_px - longit_px
2197     elif(Globals.profiles_doseplan_patient_position=='HFDR'):
2198       temp_ref_point_doseplan[0] = isocenter_px[0]-\\
2199 doseplan_vertical_displacement_px + vertical_px
2200       temp_ref_point_doseplan[1] = isocenter_px[1]+\\
2201 doseplan_lateral_displacement_px - lateral_px
2202       temp_ref_point_doseplan[2] = isocenter_px[2]+\\
2203 doseplan_longitudinal_displacement_px - longit_px
2204     elif(Globals.profiles_doseplan_patient_position=='HFDL'):
2205       temp_ref_point_doseplan[0] = isocenter_px[0]+\\
2206 doseplan_vertical_displacement_px - vertical_px
2207       temp_ref_point_doseplan[1] = isocenter_px[1]-\\
2208 doseplan_lateral_displacement_px + lateral_px
2209       temp_ref_point_doseplan[2] = isocenter_px[2]+\\
2210 doseplan_longitudinal_displacement_px - longit_px
2211     elif(Globals.profiles_doseplan_patient_position=='FFS'):
2212       temp_ref_point_doseplan[0] = isocenter_px[0]-\\
2213 doseplan_lateral_displacement_px + lateral_px
2214       temp_ref_point_doseplan[1] = isocenter_px[1]+\\
2215 doseplan_vertical_displacement_px - vertical_px

```

```

2196     temp_ref_point_doseplan[2] = isocenter_px[2]-
doseplan_longitudinal_displacement_px + longit_px
2197 elif(Globals.profiles_doseplan_patient_position=='FFP'):
2198     temp_ref_point_doseplan[0] = isocenter_px[0]+
doseplan_lateral_displacement_px - lateral_px
2199     temp_ref_point_doseplan[1] = isocenter_px[1]-+
doseplan_vertical_displacement_px + vertical_px
2200     temp_ref_point_doseplan[2] = isocenter_px[2]-+
doseplan_longitudinal_displacement_px + longit_px
2201 elif(Globals.profiles_doseplan_patient_position=='FFDR'):
2202     temp_ref_point_doseplan[0] = isocenter_px[0]-+
doseplan_vertical_displacement_px + vertical_px
2203     temp_ref_point_doseplan[1] = isocenter_px[1]-+
doseplan_lateral_displacement_px + lateral_px
2204     temp_ref_point_doseplan[2] = isocenter_px[2]-+
doseplan_longitudinal_displacement_px + longit_px
2205 else:
2206     temp_ref_point_doseplan[0] = isocenter_px[0] +
doseplan_vertical_displacement_px - vertical_px
2207     temp_ref_point_doseplan[1] = isocenter_px[1] +
doseplan_lateral_displacement_px - lateral_px
2208     temp_ref_point_doseplan[2] = isocenter_px[2]-+
doseplan_longitudinal_displacement_px + longit_px
2209
2210 Globals.profiles_reference_point_in_doseplan = temp_ref_point_doseplan
2211 reference_point = np.zeros(3)
2212 ##### 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.
2214 if(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[1, 0,
0, 0, 1, 0]):
2215     reference_point[0] = temp_ref_point_doseplan[2]
2216     reference_point[1] = temp_ref_point_doseplan[1]
2217     reference_point[2] = temp_ref_point_doseplan[0]
2218     if(Globals.profiles_film_orientation.get()=='Coronal'):
2219         #number of frames -> rows
2220         #rows -> number of frames
2221         #columns -> columns
2222         dataset_swapped = np.swapaxes(Globals.
profiles_dataset_doseplan.pixel_array , 0,1)
2223         #temp_iso = isocenter_px[0]
2224         #isocenter_px[0] = isocenter_px[1]
2225         #isocenter_px[1] = temp_iso
2226         temp_ref = reference_point[0]
2227         reference_point[0] = reference_point[1]
2228         reference_point[1] = temp_ref
2229     elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2230         #column -> number of frames
2231         #number of frames -> rows
2232         #rows -> columns
2233         dataset_swapped = np.swapaxes(Globals.
profiles_dataset_doseplan.pixel_array , 0,2)
2234         #temp_iso = isocenter_px[0]
2235         #isocenter_px[0] = isocenter_px[2]

```

```

2238     #isocenter_px[2] = temp_iso
2239     temp_ref = reference_point[0]
2240     reference_point[0] = reference_point[2]
2241     reference_point[2] = temp_ref
2242     #dataset_swapped = np.swapaxes(dataset_swapped, 0,1)
2243     #temp_iso = isocenter_px[0]
2244     #isocenter_px[0] = isocenter_px[1]
2245     #isocenter_px[1] = temp_iso
2246     #temp_ref = reference_point[0]
2247     #reference_point[0] = reference_point[1]
2248     #reference_point[1] = temp_ref
2249     elif(Globals.profiles_film_orientation.get()=='Axial'):
2250         dataset_swapped = Globals.profiles_dataset_doseplan.
2251         pixel_array
2252     else:
2253         messagebox.showerror("Error", "Something has gone wrong here.")
2254     )
2255     clearAll()
2256     return
2257 elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[1, 0,
2258 0, 0, 0, 1]):
2259     reference_point[0] = temp_ref_point_doseplan[1]
2260     reference_point[1] = temp_ref_point_doseplan[2]
2261     reference_point[2] = temp_ref_point_doseplan[0]
2262     if(Globals.profiles_film_orientation.get()=='Coronal'):
2263         dataset_swapped = Globals.profiles_dataset_doseplan.
2264         pixel_array
2265     elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2266         dataset_swapped = np.swapaxes(Globals.
2267         profiles_dataset_doseplan.pixel_array, 0,2)
2268     temp_ref = reference_point[0]
2269     reference_point[0] = reference_point[2]
2270     reference_point[2] = temp_ref
2271     dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
2272     temp_ref = reference_point[1]
2273     reference_point[1] = reference_point[2]
2274     reference_point[2] = temp_ref
2275     elif(Globals.profiles_film_orientation.get()=='Axial'):
2276         dataset_swapped = np.swapaxes(Globals.
2277         profiles_dataset_doseplan.pixel_array, 0,1)
2278     temp_ref = reference_point[0]
2279     reference_point[0] = reference_point[1]
2280     reference_point[1] = temp_ref
2281     else:
2282         messagebox.showerror("Error", "Something has gone wrong.")
2283     clearAll()
2284     return
2285 elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[0, 1,
2286 0, 1, 0, 0]):
2287     reference_point[0] = temp_ref_point_doseplan[2]
2288     reference_point[1] = temp_ref_point_doseplan[0]
2289     reference_point[2] = temp_ref_point_doseplan[1]
2290     if(Globals.profiles_film_orientation.get()=='Coronal'):
2291         dataset_swapped = np.swapaxes(Globals.
2292         profiles_dataset_doseplan.pixel_array, 0,2)
2293     temp_ref = reference_point[0]
2294     reference_point[0] = reference_point[2]

```

```

2286     reference_point[2] = temp_ref
2287     dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
2288     temp_ref = reference_point[1]
2289     reference_point[1] = reference_point[2]
2290     reference_point[2] = temp_ref
2291     elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2292         dataset_swapped = np.swapaxes(Globals.
2293         profiles_dataset_doseplan.pixel_array, 0,1)
2294         temp_ref = reference_point[0]
2295         reference_point[0] = reference_point[1]
2296         reference_point[1] = temp_ref
2297         dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
2298         temp_ref = reference_point[1]
2299         reference_point[1] = reference_point[2]
2300         reference_point[2] = temp_ref
2301     elif(Globals.profiles_film_orientation.get()=='Axial'):
2302         dataset_swapped = np.swapaxes(Globals.
2303         profiles_dataset_doseplan.pixel_array, 1,2)
2304         temp_ref = reference_point[1]
2305         reference_point[1] = reference_point[2]
2306         reference_point[2] = temp_ref
2307     else:
2308         messagebox.showerror("Error", "Something has gone wrong.")
2309         clearAll()
2310     return
2311 elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[0, 1,
2312 0, 0, 0, 1]):
2313     reference_point[0] = temp_ref_point_doseplan[0]
2314     reference_point[1] = temp_ref_point_doseplan[2]
2315     reference_point[2] = temp_ref_point_doseplan[1]
2316     if(Globals.profiles_film_orientation.get()=='Coronal'):
2317         dataset_swapped = np.swapaxes(Globals.
2318         profiles_dataset_doseplan.pixel_array, 0,2)
2319         temp_ref = reference_point[0]
2320         reference_point[0] = reference_point[2]
2321         reference_point[2] = temp_ref
2322     elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2323         dataset_swapped = np.swapaxes(Globals.
2324         profiles_dataset_doseplan.pixel_array, 1,2)
2325         temp_ref = reference_point[1]
2326         reference_point[1] = reference_point[2]
2327         reference_point[2] = temp_ref
2328     elif(Globals.profiles_film_orientation.get()=='Axial'):
2329         dataset_swapped = np.swapaxes(Globals.
2330         profiles_dataset_doseplan.pixel_array, 0,1)
2331         temp_ref = reference_point[0]
2332         reference_point[0] = reference_point[1]
2333         reference_point[1] = temp_ref
2334         dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
2335         temp_ref = reference_point[1]
2336         reference_point[1] = reference_point[2]
2337         reference_point[2] = temp_ref
2338     else:
2339         messagebox.showerror("Error", "Something has gone wrong.")
2340         clearAll()
2341     return

```

```

2336     elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[0, 0,
2337         1, 1, 0, 0]):
2338         reference_point[0] = temp_ref_point_doseplan[1]
2339         reference_point[1] = temp_ref_point_doseplan[0]
2340         reference_point[2] = temp_ref_point_doseplan[2]
2341         if(Globals.profiles_film_orientation.get()=='Coronal'):
2342             dataset_swapped = np.swapaxes(Globals.
2343                 profiles_dataset_doseplan.pixel_array , 1,2)
2344             temp_ref = reference_point[1]
2345             reference_point[1] = reference_point[2]
2346             reference_point[2] = temp_ref
2347             elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2348                 dataset_swapped = np.swapaxes(Globals.
2349                     profiles_dataset_doseplan.pixel_array , 0,1)
2350                     temp_ref = reference_point[0]
2351                     reference_point[0] = reference_point[1]
2352                     reference_point[1] = temp_ref
2353                     elif(Globals.profiles_film_orientation.get()=='Axial'):
2354                         dataset_swapped = np.swapaxes(Globals.
2355                             profiles_dataset_doseplan.pixel_array , 0,1)
2356                             temp_ref = reference_point[0]
2357                             reference_point[0] = reference_point[1]
2358                             reference_point[1] = temp_ref
2359                             else:
2360                                 messagebox.showerror("Error", "Something has gone wrong.")
2361                                 clearAll()
2362                                 return
2363             elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[0, 0,
2364                 1, 0, 1, 0]):
2365                 reference_point[0] = temp_ref_point_doseplan[0]
2366                 reference_point[1] = temp_ref_point_doseplan[1]
2367                 reference_point[2] = temp_ref_point_doseplan[2]
2368                 if(Globals.profiles_film_orientation.get()=='Coronal'):
2369                     dataset_swapped = np.swapaxes(Globals.
2370                         profiles_dataset_doseplan.pixel_array , 0,2)
2371                         temp_ref = reference_point[0]
2372                         reference_point[0] = reference_point[2]
2373                         reference_point[2] = temp_ref
2374                         dataset_swapped = np.swapaxes(dataset_swapped , 0,1)
2375                         temp_ref = reference_point[0]
2376                         reference_point[0] = reference_point[1]
2377                         reference_point[1] = temp_ref
2378                         elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2379                             dataset_swapped = Globals.profiles_dataset_doseplan.
2380                             pixel_array
2381                             elif(Globals.profiles_film_orientation.get()=='Axial'):
2382                                 dataset_swapped = np.swapaxes(Globals.
2383                                     profiles_dataset_doseplan.pixel_array , 0,2)
2384                                     temp_ref = reference_point[0]
2385                                     reference_point[0] = reference_point[2]
2386                                     reference_point[2] = temp_ref
2387                                     else:
2388                                         messagebox.showerror("Error", "Something has gone wrong.")

```

```

2386     clearAll()
2387     return
2388 else:
2389     messagebox.showerror("Error", "Something has gone wrong.")
2390     clearAll()
2391     return
2392
2393 if(reference_point[0]<0 or reference_point[0]>dataset_swapped.shape[0]):
2394     messagebox.showerror("Error", "Reference point is outside of
2395 dosematrix\n"
2396     (Code: first dimension, number of frames in dosematrix))
2397     return
2398 if(reference_point[1]<0 or reference_point[1]>dataset_swapped.shape[1]):
2399     messagebox.showerror("Error", "Reference point is outside of
2400 dosematrix\n"
2401     (Code: second dimension, rows in dosematrix))
2402     return
2403 if(reference_point[2]<0 or reference_point[2]>dataset_swapped.shape[2]):
2404     messagebox.showerror("Error", "Reference point is outside of
2405 dosematrix\n"
2406     (Code: third dimension, columns in dosematrix))
2407     return
2408
2409 dose_slice = dataset_swapped[int(reference_point[0]), :, :]
2410
2411
2412 #calculate the coordinates of the Region of Interest in doseplan (
2413 #marked on the film)
2414 #and checks if it actualy exists in dosematrix
2415
2416 doseplan_ROI_coords = []
2417 top_left_test_side = False; top_left_test_down = False
2418 top_right_test_side = False; top_right_test_down = False
2419 bottom_left_test_side = False; bottom_left_test_down = False
2420 bottom_right_test_side = False; bottom_right_test_down = False
2421 top_left_side_corr = 0; top_left_down_corr = 0
2422 top_right_side_corr = 0; top_right_down_corr = 0
2423 bottom_left_side_corr = 0; bottom_left_down_corr = 0
2424 bottom_right_side_corr = 0; bottom_right_down_corr = 0
2425
2426
2427 top_left_to_side = reference_point[2] -
2428 distance_in_doseplan_ROI_reference_point_px[0][0]
2429 top_left_down = reference_point[1] -
2430 distance_in_doseplan_ROI_reference_point_px[0][1]
2431 if(top_left_to_side < 0):
2432     top_left_test_side = True
2433     top_left_side_corr = abs(top_left_to_side)
2434     top_left_to_side = 0
2435 if(top_left_to_side > dose_slice.shape[1]):
2436     messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2437 out of range in doseplan. Try again")
2438     clearAll()

```

```

2432     return
2433     if(top_left_down < 0):
2434         top_left_test_down = True
2435         top_left_down_corr = abs(top_left_down)
2436         top_left_down = 0
2437     if(top_left_down > dose_slice.shape[0]):
2438         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2439             out of range in doseplan. Try again")
2440         clearAll()
2441         return
2442
2443     top_right_to_side = reference_point[2] -
2444         distance_in_doseplan_ROI_reference_point_px[1][0]
2445     top_right_down = reference_point[1] -
2446         distance_in_doseplan_ROI_reference_point_px[1][1]
2447     if(top_right_to_side < 0):
2448         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2449             out of range in doseplan. Try again")
2450         clearAll()
2451         return
2452     if(top_right_to_side > dose_slice.shape[1]):
2453         top_right_test_side = True
2454         top_right_side_corr = top_right_to_side - dose_slice.shape[1]
2455         top_right_to_side = dose_slice.shape[1]
2456     if(top_right_down < 0):
2457         top_right_test_down = True
2458         top_right_down_corr = abs(top_right_down)
2459         top_right_down = 0
2460     if(top_right_down > dose_slice.shape[0]):
2461         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2462             out of range in doseplan. Try again")
2463         clearAll()
2464         return
2465
2466     bottom_left_to_side = reference_point[2] -
2467         distance_in_doseplan_ROI_reference_point_px[2][0]
2468     bottom_left_down = reference_point[1] -
2469         distance_in_doseplan_ROI_reference_point_px[2][1]
2470     if(bottom_left_to_side < 0):
2471         bottom_left_test_side = True
2472         bottom_left_side_corr = abs(bottom_left_to_side)
2473         bottom_left_to_side = 0
2474     if(bottom_left_to_side > dose_slice.shape[1]):
2475         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2476             out of range in doseplan. Try again")
2477         clearAll()
2478         return
2479     if(bottom_left_down < 0):
2480         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2481             out of range in doseplan. Try again")
2482         clearAll()
2483         return
2484     if(bottom_left_down > dose_slice.shape[0]):
2485         bottom_left_down_corr = bottom_left_down - dose_slice.shape[0]
2486         bottom_left_down = dose_slice.shape[0]
2487         bottom_left_test_down = True

```

```

2480     bottom_right_to_side = reference_point[2] -
2481         distance_in_doseplan_ROI_reference_point_px[3][0]
2482     bottom_right_down = reference_point[1] -
2483         distance_in_doseplan_ROI_reference_point_px[3][1]
2484     if(bottom_right_to_side < 0):
2485         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2486             out of range in doseplan. Try again")
2487         clearAll()
2488         return
2489     if(bottom_right_to_side > dose_slice.shape[1]):
2490         bottom_right_side_corr = bottom_right_to_side - dose_slice.shape
2491             [1]
2492         bottom_right_to_side = dose_slice.shape[1]
2493         bottom_right_test_side = True
2494     if(bottom_right_down < 0):
2495         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2496             out of range in doseplan. Try again")
2497         clearAll()
2498         return
2499     if(bottom_right_down > dose_slice.shape[0]):
2500         bottom_right_down_corr = bottom_right_down - dose_slice.shape[0]
2501         bottom_right_down = dose_slice.shape[0]
2502         bottom_right_test_down = True
2503
2504
2505     if(top_right_test_side or top_right_test_down or top_left_test_side or
2506         top_left_test_down \
2507         or bottom_right_test_side or bottom_right_test_down or
2508         bottom_left_test_side or bottom_left_test_down):
2509         ROI_info = "Left side: " + str(max(top_left_side_corr,
2510             bottom_left_side_corr)) + " pixels.\n" \
2511             + "Right side: " + str(max(top_right_side_corr,
2512                 bottom_right_side_corr)) + " pixels.\n" \
2513                 + "Top side: " + str(max(top_left_down_corr,
2514                     top_right_down_corr)) + " pixels.\n" \
2515                     + "Bottom side: " + str(max(bottom_left_down_corr,
2516                         bottom_right_down_corr)) + " pixels."
2517         messagebox.showinfo("ROI info", "The ROI marked on the film did
2518             not fit with the size of the doseplan and had to \
2519                 be cut.\n" + ROI_info )
2520
2521     doseplan_ROI_coords.append([top_left_to_side, top_left_down])
2522     doseplan_ROI_coords.append([top_right_to_side, top_right_down])
2523     doseplan_ROI_coords.append([bottom_left_to_side, bottom_left_down])
2524     doseplan_ROI_coords.append([bottom_right_to_side, bottom_right_down])
2525
2526     if only_one:
2527         Globals.profiles_doseplan_dataset_ROI = \
2528             dose_slice[int(top_left_down):int(bottom_left_down), int(
2529                 top_left_to_side):int(top_right_to_side)]*Globals.
2530                 profiles_dataset_doseplan.DoseGridScaling
2531
2532         img=Globals.profiles_doseplan_dataset_ROI
2533         if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
2534             img = cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5))
2535         elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
2536             img = cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10))

```

```

2524     else :
2525         img = cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15))
2526
2527         mx=np.max(img)
2528         Globals.max_dose_doseplan = mx
2529         img = img/mx
2530         PIL_img_doseplan_ROI = Image.fromarray(np.uint8(cm.viridis(img)
2531 *255))
2532
2533         wid = PIL_img_doseplan_ROI.width; heig = PIL_img_doseplan_ROI.
2534 height
2535         doseplan_canvas = tk.Canvas(Globals.profiles_film_panedwindow)
2536         doseplan_canvas.grid(row=2, column=0, sticky=N+S+W+E)
2537         Globals.profiles_film_panedwindow.add(doseplan_canvas, \
2538             height=max(heig, Globals.profiles_doseplan_text_image.height())
2539         ), \
2540             width=wid + Globals.profiles_doseplan_text_image.width())
2541         doseplan_canvas.config(bg='#ffffff', relief=FLAT,
2542         highlightthickness=0, \
2543             height=max(heig, Globals.profiles_doseplan_text_image.height())
2544         ), \
2545             width=wid + Globals.profiles_doseplan_text_image.width())
2546
2547
2548         Globals.doseplan_write_image = tk.Canvas(doseplan_canvas)
2549         Globals.doseplan_write_image.grid(row=0,column=1,sticky=N+S+W+E)
2550         Globals.doseplan_write_image.config(bg='#ffffff', relief=FLAT,
2551         highlightthickness=0, width=wid, height=heig)
2552
2553         doseplan_text_image_canvas = tk.Canvas(doseplan_canvas)
2554         doseplan_text_image_canvas.grid(row=0,column=0,sticky=N+S+W+E)
2555         doseplan_text_image_canvas.config(bg='#ffffff', relief=FLAT,
2556         highlightthickness=0, \
2557             width=Globals.profiles_doseplan_text_image.width(), height=
2558             Globals.profiles_doseplan_text_image.height())
2559
2560         scaled_image_visual = PIL_img_doseplan_ROI
2561         scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual
2562     )
2563
2564         Globals.doseplan_write_image_width = scaled_image_visual.width()
2565         Globals.doseplan_write_image_height = scaled_image_visual.height()
2566         Globals.doseplan_write_image.create_image(0,0,image=
2567             scaled_image_visual, anchor="nw")
2568
2569         Globals.doseplan_write_image.image = scaled_image_visual
2570         doseplan_text_image_canvas.create_image(0,0,image=Globals.
2571 profiles_doseplan_text_image, anchor="nw")
2572
2573         doseplan_text_image_canvas.image=Globals.
2574 profiles_doseplan_text_image
2575
2576         drawProfiles(False)
2577
2578     else :
2579         img=dose_slice[int(top_left_down):int(bottom_left_down), int(
2580 top_left_to_side):int(top_right_to_side)]
2581
2582         Globals.profiles_doseplan_dataset_ROI_several.append(img)
2583         Globals.number_of_doseplans+=1

```

```

2568     if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
2569         Globals.profiles_several_img.append(img)#cv2.resize(img, dsize
2570         =(img.shape[1]*5,img.shape[0]*5)))
2571     elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
2572         Globals.profiles_several_img.append(img)#cv2.resize(img, dsize
2573         =(img.shape[1]*10,img.shape[0]*10)))
2574     else:
2575         Globals.profiles_several_img.append(img)#cv2.resize(img, dsize
2576         =(img.shape[1]*15,img.shape[0]*15)))
2577
2578
2579 def processDoseplan_usingIsocenter(only_one):
2580     ##### RT Plan #####
2581
2582     #Find each coordinate in mm to isocenter relative to first element in
2583     #doseplan
2584     iso_1 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[0]
2585     - Globals.profiles_dataset_rtplan.BeamSequence[0].ControlPointSequence
2586     [0].IsocenterPosition[0])
2587     iso_2 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[1]
2588     - Globals.profiles_dataset_rtplan.BeamSequence[0].ControlPointSequence
2589     [0].IsocenterPosition[1])
2590     iso_3 = abs(Globals.profiles_dataset_doseplan.ImagePositionPatient[2]
2591     - Globals.profiles_dataset_rtplan.BeamSequence[0].ControlPointSequence
2592     [0].IsocenterPosition[2])
2593     #Given as [x,y,z] in patient coordinates
2594     Globals.profiles_isocenter_mm = [iso_1, iso_2, iso_3]
2595
2596
2597     #Isocenter in pixel relative to the first element in the doseplan
2598     isocenter_px = np.zeros(3)
2599     distance_in_doseplan_ROI_reference_point_px = []
2600     if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
2601         isocenter_px[0] = np.round(iso_1)#np.round(Globals.
2602         profiles_isocenter_mm[0])
2603         isocenter_px[1] = np.round(iso_2)#np.round(Globals.
2604         profiles_isocenter_mm[1])
2605         isocenter_px[2] = np.round(iso_3)#np.round(Globals.
2606         profiles_isocenter_mm[2])
2607
2608         #Change distance in film to pixel in doseplan
2609         distance_in_doseplan_ROI_reference_point_px.append([np.round(
2610             Globals.profiles_distance_isocenter_ROI[0][0]), \
2611             np.round(Globals.profiles_distance_isocenter_ROI[0][1])])
2612         distance_in_doseplan_ROI_reference_point_px.append([np.round(
2613             Globals.profiles_distance_isocenter_ROI[1][0]), \
2614             np.round(Globals.profiles_distance_isocenter_ROI[1][1])])
2615         distance_in_doseplan_ROI_reference_point_px.append([np.round(
2616             Globals.profiles_distance_isocenter_ROI[2][0]), \
2617             np.round(Globals.profiles_distance_isocenter_ROI[2][1])])
2618         distance_in_doseplan_ROI_reference_point_px.append([np.round(
2619             Globals.profiles_distance_isocenter_ROI[3][0]), \
2620             np.round(Globals.profiles_distance_isocenter_ROI[3][1])])
2621
2622     elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):

```

```

    isocenter_px[0] = np.round(iso_1/2)#np.round(Globals.
2608 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)

2610

2612     #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), \
2614         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), \
2616         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), \
2618         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), \
2620         np.round((Globals.profiles_distance_isocenter_ROI[3][1])/2)))

2622 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)

2626     #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), \
2628         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), \
2630         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), \
2632         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), \
2634         np.round((Globals.profiles_distance_isocenter_ROI[3][1])/3)])

2636 reference_point = np.zeros(3)

2638 ##### Doseplan #####
2640 #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.
2642 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]

```

```

2648     reference_point[1] = isocenter_px[1]
2649     reference_point[2] = isocenter_px[0]
2650     if(Globals.profiles_film_orientation.get()=='Coronal'):
2651         #number of frames -> rows
2652         #rows -> number of frames
2653         #columns -> columns
2654         dataset_swapped = np.swapaxes(Globals.
2655             profiles_dataset_doseplan.pixel_array, 0,1)
2656         #temp_iso = isocenter_px[0]
2657         #isocenter_px[0] = isocenter_px[1]
2658         #isocenter_px[1] = temp_iso
2659         temp_ref = reference_point[0]
2660         reference_point[0] = reference_point[1]
2661         reference_point[1] = temp_ref
2662     elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2663         #column -> number of frames
2664         #number of frames -> rows
2665         #rows -> columns
2666         dataset_swapped = np.swapaxes(Globals.
2667             profiles_dataset_doseplan.pixel_array, 0,2)
2668         #temp_iso = isocenter_px[0]
2669         #isocenter_px[0] = isocenter_px[2]
2670         #isocenter_px[2] = temp_iso
2671         temp_ref = reference_point[0]
2672         reference_point[0] = reference_point[2]
2673         reference_point[2] = temp_ref
2674         #dataset_swapped = np.swapaxes(dataset_swapped, 0,1)
2675         #temp_iso = isocenter_px[0]
2676         #isocenter_px[0] = isocenter_px[1]
2677         #isocenter_px[1] = temp_iso
2678         #temp_ref = reference_point[0]
2679         #reference_point[0] = reference_point[1]
2680         #reference_point[1] = temp_ref
2681     elif(Globals.profiles_film_orientation.get()=='Axial'):
2682         dataset_swapped = Globals.profiles_dataset_doseplan.
2683             pixel_array
2684     else:
2685         messagebox.showerror("Error", "Something has gone wrong here.")
2686     )
2687     clearAll()
2688     return
2689 elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[1, 0,
2690 0, 0, 0, 1]):
2691     #reference_point[1] = isocenter_px[0]
2692     #reference_point[2] = isocenter_px[1]
2693     #reference_point[0] = isocenter_px[2]
2694     reference_point[0] = isocenter_px[1]
2695     reference_point[1] = isocenter_px[2]
2696     reference_point[2] = isocenter_px[0]
2697     if(Globals.profiles_film_orientation.get()=='Coronal'):
2698         dataset_swapped = Globals.profiles_dataset_doseplan.
2699             pixel_array
2700     elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2701         #columns -> number of frames
2702         #number of frames -> columns
2703         #rows -> rows

```

```

        dataset_swapped = np.swapaxes(Globals.
2698 profiles_dataset_doseplan.pixel_array, 0,2)
        #temp_iso = isocenter_px[0]
        #isocenter_px[0] = isocenter_px[2]
2700 #isocenter_px[2] = temp_iso
        temp_ref = reference_point[0]
2702 reference_point[0] = reference_point[2]
        reference_point[2] = temp_ref
2704 dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
        temp_ref = reference_point[1]
2706 reference_point[1] = reference_point[2]
        reference_point[2] = temp_ref
2708 elif(Globals.profiles_film_orientation.get()=='Axial'):
#rows -> number of frames
2710 #number of frames -> rows
#columns -> columns
2712 dataset_swapped = np.swapaxes(Globals.
profiles_dataset_doseplan.pixel_array, 0,1)
        #temp_iso = isocenter_px[0]
        #isocenter_px[0] = isocenter_px[1]
2714 #isocenter_px[1] = temp_iso
        temp_ref = reference_point[0]
2716 reference_point[0] = reference_point[1]
        reference_point[1] = temp_ref
2718 else:
    messagebox.showerror("Error", "Something has gone wrong.")
    clearAll()
    return
2722 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]
2724 reference_point[0] = isocenter_px[2]
reference_point[1] = isocenter_px[0]
reference_point[2] = isocenter_px[1]
2728 if(Globals.profiles_film_orientation.get()=='Coronal'):
#rows -> columns
#columns -> number of frames
#number of frames -> rows
2732 dataset_swapped = np.swapaxes(Globals.
profiles_dataset_doseplan.pixel_array, 0,2)
        #temp_iso = isocenter_px[0]
        #isocenter_px[0] = isocenter_px[2]
2734 #isocenter_px[2] = temp_iso
        temp_ref = reference_point[0]
2736 reference_point[0] = reference_point[2]
        reference_point[2] = temp_ref
2738 dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
        #temp_iso = isocenter_px[1]
        #isocenter_px[1] = isocenter_px[2]
2740 #isocenter_px[2] = temp_iso
        temp_ref = reference_point[1]
2742 reference_point[1] = reference_point[2]
        reference_point[2] = temp_ref
2744 elif(Globals.profiles_film_orientation.get()=='Sagittal'):
#number -> rows

```

```

2750         #columns -> columns
2751         #rows -> number of frames
2752         dataset_swapped = np.swapaxes(Globals.
2753             profiles_dataset_doseplan.pixel_array , 0,1)
2754         #temp_iso = isocenter_px[0]
2755         #isocenter_px[0] = isocenter_px[1]
2756         #isocenter_px[1] = temp_iso
2757         temp_ref = reference_point[0]
2758         reference_point[0] = reference_point[1]
2759         reference_point[1] = temp_ref
2760         dataset_swapped = np.swapaxes(dataset_swapped , 1,2)
2761         temp_ref = reference_point[1]
2762         reference_point[1] = reference_point[2]
2763         reference_point[2] = temp_ref
2764     elif(Globals.profiles_film_orientation.get()=='Axial'):
2765         #column -> rows
2766         #rows -> column
2767         #number of frames -> number of frames
2768         dataset_swapped = np.swapaxes(Globals.
2769             profiles_dataset_doseplan.pixel_array , 1,2)
2770         #temp_iso = isocenter_px[1]
2771         #isocenter_px[1] = isocenter_px[2]
2772         #isocenter_px[2] = temp_iso
2773         temp_ref = reference_point[1]
2774         reference_point[1] = reference_point[2]
2775         reference_point[2] = temp_ref
2776     else:
2777         messagebox.showerror("Error" , "Something has gone wrong.")
2778         clearAll()
2779     return
2780 elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[0, 1,
2781 0, 0, 0, 1]):
2782     #reference_point[1] = isocenter_px[0]
2783     #reference_point[2] = isocenter_px[1]
2784     #reference_point[0] = isocenter_px[2]
2785     reference_point[0] = isocenter_px[0]
2786     reference_point[1] = isocenter_px[2]
2787     reference_point[2] = isocenter_px[1]
2788     if(Globals.profiles_film_orientation.get()=='Coronal'):
2789         #rows -> rows
2790         #columns -> number of frames
2791         #number of frames ->columns
2792         dataset_swapped = np.swapaxes(Globals.
2793             profiles_dataset_doseplan.pixel_array , 0,2)
2794         #temp_iso = isocenter_px[0]
2795         #isocenter_px[0] = isocenter_px[2]
2796         #isocenter_px[2] = temp_iso
2797         temp_ref = reference_point[0]
2798         reference_point[0] = reference_point[2]
2799         reference_point[2] = temp_ref
2800     elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2801         dataset_swapped = np.swapaxes(Globals.
2802             profiles_dataset_doseplan.pixel_array , 1,2)
2803         temp_ref = reference_point[1]
2804         reference_point[1] = reference_point[2]
2805         reference_point[2] = temp_ref
2806     elif(Globals.profiles_film_orientation.get()=='Axial'):

```

```

2802         #number of frames -> columns
2803         #columns -> rows
2804         #rows -> number of frames
2805         dataset_swapped = np.swapaxes(Globals.
2806             profiles_dataset_doseplan.pixel_array, 0,1)
2807             #temp_iso = isocenter_px[0]
2808             #isocenter_px[0] = isocenter_px[1]
2809             #isocenter_px[1] = temp_iso
2810             temp_ref = reference_point[0]
2811             reference_point[0] = reference_point[1]
2812             reference_point[1] = temp_ref
2813             dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
2814             #temp_iso = isocenter_px[1]
2815             #isocenter_px[1] = isocenter_px[2]
2816             #isocenter_px[2] = temp_iso
2817             temp_ref = reference_point[1]
2818             reference_point[1] = reference_point[2]
2819             reference_point[2] = temp_ref
2820     else:
2821         messagebox.showerror("Error", "Something has gone wrong.")
2822         clearAll()
2823         return
2824     elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[0, 0,
2825           1, 1, 0, 0]):
2826         #reference_point[1] = isocenter_px[0]
2827         #reference_point[2] = isocenter_px[1]
2828         #reference_point[0] = isocenter_px[2]
2829         reference_point[0] = isocenter_px[1]
2830         reference_point[1] = isocenter_px[0]
2831         reference_point[2] = isocenter_px[2]
2832         if(Globals.profiles_film_orientation.get()=='Coronal'):
2833             #rows -> columns
2834             #columns -> rows
2835             #number of frames -> number of frames
2836             dataset_swapped = np.swapaxes(Globals.
2837                 profiles_dataset_doseplan.pixel_array, 1,2)
2838                 #temp_iso = isocenter_px[1]
2839                 #isocenter_px[1] = isocenter_px[2]
2840                 #isocenter_px[2] = temp_iso
2841                 temp_ref = reference_point[1]
2842                 reference_point[1] = reference_point[2]
2843                 reference_point[2] = temp_ref
2844             elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2845                 #rows -> number of frames
2846                 #columns -> rows
2847                 #number of frames -> columns
2848                 dataset_swapped = np.swapaxes(Globals.
2849                     profiles_dataset_doseplan.pixel_array, 0,1)
2850                     #temp_iso = isocenter_px[0]
2851                     #isocenter_px[0] = isocenter_px[1]
2852                     #isocenter_px[1] = temp_iso
2853                     temp_ref = reference_point[0]
2854                     reference_point[0] = reference_point[1]
2855                     reference_point[1] = temp_ref
2856                     #dataset_swapped = np.swapaxes(dataset_swapped, 1,2)
2857                     #temp_iso = isocenter_px[1]
2858                     #isocenter_px[1] = isocenter_px[2]

```

```

2856     #isocenter_px[2] = temp_iso
2857     #temp_ref = reference_point[1]
2858     #reference_point[1] = reference_point[2]
2859     #reference_point[2] = temp_ref
2860 elif(Globals.profiles_film_orientation.get()=='Axial'):
2861     #rows -> columns
2862     #columns -> number of frames
2863     #number of frames -> rows
2864     dataset_swapped = np.swapaxes(Globals.
2865         profiles_dataset_doseplan.pixel_array, 0,1)
2866     #temp_iso = isocenter_px[0]
2867     #isocenter_px[0] = isocenter_px[1]
2868     #isocenter_px[1] = temp_iso
2869     temp_ref = reference_point[0]
2870     reference_point[0] = reference_point[1]
2871     reference_point[1] = temp_ref
2872     dataset_swapped = np.swapaxes(dataset_swapped, 0,2)
2873     #temp_iso = isocenter_px[0]
2874     #isocenter_px[0] = isocenter_px[2]
2875     #isocenter_px[2] = temp_iso
2876     temp_ref = reference_point[0]
2877     reference_point[0] = reference_point[2]
2878     reference_point[2] = temp_ref
2879 else:
2880     messagebox.showerror("Error", "Something has gone wrong.")
2881     clearAll()
2882     return
2883 elif(Globals.profiles_dataset_doseplan.ImageOrientationPatient==[0, 0,
2884     1, 0, 1, 0]):
2885     #reference_point[1] = isocenter_px[0]
2886     #reference_point[2] = isocenter_px[1]
2887     #reference_point[0] = isocenter_px[2]
2888     reference_point[0] = isocenter_px[0]
2889     reference_point[1] = isocenter_px[1]
2890     reference_point[2] = isocenter_px[2]
2891     if(Globals.profiles_film_orientation.get()=='Coronal'):
2892         #rows -> number of frames
2893         #columns ->rows
2894         #number of frames -> columns
2895         dataset_swapped = np.swapaxes(Globals.
2896             profiles_dataset_doseplan.pixel_array, 0,2)
2897         #temp_iso = isocenter_px[0]
2898         #isocenter_px[0] = isocenter_px[2]
2899         #isocenter_px[2] = temp_iso
2900         temp_ref = reference_point[0]
2901         reference_point[0] = reference_point[2]
2902         reference_point[2] = temp_ref
2903         dataset_swapped = np.swapaxes(dataset_swapped, 0,1)
2904         #temp_iso = isocenter_px[0]
2905         #isocenter_px[0] = isocenter_px[1]
2906         #isocenter_px[1] = temp_iso
2907         temp_ref = reference_point[0]
2908         reference_point[0] = reference_point[1]
2909         reference_point[1] = temp_ref
2910     elif(Globals.profiles_film_orientation.get()=='Sagittal'):
2911         #rows -> columns
2912         #columns -> rows

```

```

2910         #number of frames -> number of frames
2911         dataset_swapped = Globals.profiles_dataset_doseplan.
2912         pixel_array
2913         elif (Globals.profiles_film_orientation.get() == 'Axial'):
2914             dataset_swapped = np.swapaxes(Globals.
2915             profiles_dataset_doseplan.pixel_array, 0,2)
2916             temp_ref = reference_point[0]
2917             reference_point[0] = reference_point[2]
2918             reference_point[2] = temp_ref
2919         else:
2920             messagebox.showerror("Error", "Something has gone wrong.")
2921             clearAll()
2922             return
2923
2924
2925
2926 ##### Match film and doseplan
2927 #####
2928 #Pick the slice where the reference point is (this is the slice-
2929 #position of the film)
2930
2931 if Globals.profiles_dataset_doseplan.PixelSpacing == [1, 1]:
2932     offset = int(np.round(Globals.profiles_offset))
2933     dose_slice = dataset_swapped[int(reference_point[0] + offset)]
2934 elif Globals.profiles_dataset_doseplan.PixelSpacing == [2, 2]:
2935     offset = int(np.round(Globals.profiles_offset/2))
2936     dose_slice = dataset_swapped[int(reference_point[0] + offset)]
2937 else:
2938     offset = int(np.round(Globals.profiles_offset/3))
2939     dose_slice = dataset_swapped[int(reference_point[0]+ offset)]
2940
2941
2942 #calculate the coordinates of the Region of Interest in doseplan (
2943 #marked on the film)
2944 #and checks if it actually exists in dosematrix
2945
2946 doseplan_ROI_coords = []
2947 top_left_test_side = False; top_left_test_down = False
2948 top_right_test_side = False; top_right_test_down = False
2949 bottom_left_test_side = False; bottom_left_test_down = False
2950 bottom_right_test_side = False; bottom_right_test_down = False
2951 top_left_side_corr = 0; top_left_down_corr = 0
2952 top_right_side_corr = 0; top_right_down_corr = 0
2953 bottom_left_side_corr = 0; bottom_left_down_corr = 0
2954 bottom_right_side_corr = 0; bottom_right_down_corr = 0
2955
2956 top_left_to_side = reference_point[2] -
2957 distance_in_doseplan_ROI_reference_point_px[0][0]
2958 top_left_down = reference_point[1] -
distance_in_doseplan_ROI_reference_point_px[0][1]
```

```

2960     if(top_left_to_side < 0):
2961         top_left_test_side = True
2962         top_left_side_corr = abs(top_left_to_side)
2963         top_left_to_side = 0
2964     if(top_left_to_side > dose_slice.shape[1]):
2965         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2966         out of range in doseplan. Try again")
2967         clearAll()
2968     return
2969
2970     if(top_left_down < 0):
2971         top_left_test_down = True
2972         top_left_down_corr = abs(top_left_down)
2973         top_left_down = 0
2974     if(top_left_down > dose_slice.shape[0]):
2975         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2976         out of range in doseplan. Try again")
2977         clearAll()
2978     return
2979
2980     top_right_to_side = reference_point[2] -
2981     distance_in_doseplan_ROI_reference_point_px[1][0]
2982     top_right_down = reference_point[1] -
2983     distance_in_doseplan_ROI_reference_point_px[1][1]
2984     if(top_right_to_side < 0):
2985         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2986         out of range in doseplan. Try again")
2987         clearAll()
2988     return
2989     if(top_right_to_side > dose_slice.shape[1]):
2990         top_right_test_side = True
2991         top_right_side_corr = top_right_to_side - dose_slice.shape[1]
2992         top_right_to_side = dose_slice.shape[1]
2993     if(top_right_down < 0):
2994         top_right_test_down = True
2995         top_right_down_corr = abs(top_right_down)
2996         top_right_down = 0
2997     if(top_right_down > dose_slice.shape[0]):
2998         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
2999         out of range in doseplan. Try again")
3000         clearAll()
3001     return
3002
3003     bottom_left_to_side = reference_point[2] -
3004     distance_in_doseplan_ROI_reference_point_px[2][0]
3005     bottom_left_down = reference_point[1] -
3006     distance_in_doseplan_ROI_reference_point_px[2][1]
3007     if(bottom_left_to_side < 0):
3008         bottom_left_test_side = True
3009         bottom_left_side_corr = abs(bottom_left_to_side)
3010         bottom_left_to_side = 0
3011     if(bottom_left_to_side > dose_slice.shape[1]):
3012         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
3013         out of range in doseplan. Try again")
3014         clearAll()
3015     return
3016     if(bottom_left_down < 0):

```

```

3006     messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
3007         out of range in doseplan. Try again")
3008         clearAll()
3009         return
3010     if(bottom_left_down > dose_slice.shape[0]):
3011         bottom_left_down_corr = bottom_left_down - dose_slice.shape[0]
3012         bottom_left_down = dose_slice.shape[0]
3013         bottom_left_test_down = True
3014
3015     bottom_right_to_side = reference_point[2] -
3016         distance.in_doseplan_ROI_reference_point_px[3][0]
3017     bottom_right_down = reference_point[1] -
3018         distance.in_doseplan_ROI_reference_point_px[3][1]
3019     if(bottom_right_to_side < 0):
3020         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
3021             out of range in doseplan. Try again")
3022         clearAll()
3023         return
3024     if(bottom_right_to_side > dose_slice.shape[1]):
3025         bottom_right_side_corr = bottom_right_to_side - dose_slice.shape
3026             [1]
3027         bottom_right_to_side = dose_slice.shape[1]
3028         bottom_right_test_side = True
3029     if(bottom_right_down < 0):
3030         messagebox.showerror("Fatal Error", "Fatal error: marked ROI is
3031             out of range in doseplan. Try again")
3032         clearAll()
3033         return
3034     if(bottom_right_down > dose_slice.shape[0]):
3035         bottom_right_down_corr = bottom_right_down - dose_slice.shape[0]
3036         bottom_right_down = dose_slice.shape[0]
3037         bottom_right_test_down = True
3038
3039
3040     if(top_right_test_side or top_right_test_down or top_left_test_side or
3041         top_left_test_down \
3042         or bottom_right_test_side or bottom_right_test_down or
3043         bottom_left_test_side or bottom_left_test_down):
3044         ROI_info = "Left side: " + str(max(top_left_side_corr,
3045             bottom_left_side_corr)) + " pixels.\n" \
3046             + "Right side: " + str(max(top_right_side_corr,
3047                 bottom_right_side_corr)) + " pixels.\n" \
3048                 + "Top side: " + str(max(top_left_down_corr,
3049                     top_right_down_corr)) + " pixels.\n" \
3050                         + "Bottom side: " + str(max(bottom_left_down_corr,
3051                             bottom_right_down_corr)) + " pixels."
3052         messagebox.showinfo("ROI info", "The ROI marked on the film did
3053             not fit with the size of the doseplan and had to \
3054                 be cut.\n" + ROI_info)
3055
3056 doseplan_ROI_coords.append([top_left_to_side, top_left_down])
3057 doseplan_ROI_coords.append([top_right_to_side, top_right_down])
3058 doseplan_ROI_coords.append([bottom_left_to_side, bottom_left_down])
3059 doseplan_ROI_coords.append([bottom_right_to_side, bottom_right_down])
3060
3061 #dose_slice = cv2.flip(dose_slice, 1)
3062 if(only_one):

```

```

3050     Globals.profiles_doseplan_dataset_ROI = \
3051         dose_slice[int(top_left_down):int(bottom_left_down), int(
3052             top_left_to_side):int(top_right_to_side)]*Globals.
3053             profiles_dataset_doseplan.DoseGridScaling

3054
3055     img=Globals.profiles_doseplan_dataset_ROI
3056     if(Globals.profiles_dataset_doseplan.PixelSpacing==[1, 1]):
3057         img = cv2.resize(img, dsize=(img.shape[1]*5,img.shape[0]*5))
3058     elif(Globals.profiles_dataset_doseplan.PixelSpacing==[2, 2]):
3059         img = cv2.resize(img, dsize=(img.shape[1]*10,img.shape[0]*10))
3060     else:
3061         img = cv2.resize(img, dsize=(img.shape[1]*15,img.shape[0]*15))

3062     mx=np.max(img)
3063     Globals.max_dose_doseplan = mx
3064     max_dose = mx
3065     img = img/mx
3066     PIL_img_doseplan_ROI = Image.fromarray(np.uint8(cm.viridis(img)
3067 *255))

3068     wid = PIL_img_doseplan_ROI.width;heig = PIL_img_doseplan_ROI.
3069     height
3070     doseplan_canvas = tk.Canvas(Globals.profiles_film_panedwindow)
3071     doseplan_canvas.grid(row=2, column=0, sticky=N+S+W+E)
3072     Globals.profiles_film_panedwindow.add(doseplan_canvas, \
3073         height=max(heig, Globals.profiles_doseplan_text_image.height())
3074     ), \
3075         width=wid + Globals.profiles_doseplan_text_image.width())
3076     doseplan_canvas.config(bg='#ffffff', relief=FLAT,
3077     highlightthickness=0, \
3078         height=max(heig, Globals.profiles_doseplan_text_image.height())
3079     ), \
3080         width=wid + Globals.profiles_doseplan_text_image.width())

3081
3082     Globals.doseplan_write_image = tk.Canvas(doseplan_canvas)
3083     Globals.doseplan_write_image.grid(row=0,column=1,sticky=N+S+W+E)
3084     Globals.doseplan_write_image.config(bg='#ffffff', relief=FLAT,
3085     highlightthickness=0, width=wid, height=heig)

3086     doseplan_text_image_canvas = tk.Canvas(doseplan_canvas)
3087     doseplan_text_image_canvas.grid(row=0,column=0,sticky=N+S+W+E)
3088     doseplan_text_image_canvas.config(bg='#ffffff', relief=FLAT,
3089     highlightthickness=0, \
3090         width=Globals.profiles_doseplan_text_image.width(), height=
3091     Globals.profiles_doseplan_text_image.height())

3092     scaled_image_visual = PIL_img_doseplan_ROI
3093     scaled_image_visual = ImageTk.PhotoImage(image=scaled_image_visual
3094 )
3095     Globals.doseplan_write_image_width = scaled_image_visual.width()
3096     Globals.doseplan_write_image_height = scaled_image_visual.height()
3097     Globals.doseplan_write_image.create_image(0,0,image=
3098     scaled_image_visual, anchor="nw")
3099     Globals.doseplan_write_image.image = scaled_image_visual

```

```

3094     doseplan_text_image_canvas.create_image(0,0,image=Globals .
3095 profiles_doseplan_text_image , anchor="nw")
3096     doseplan_text_image_canvas.image=Globals .
3097 profiles_doseplan_text_image
3098
3099     drawProfiles(False)
3100
3101 else :
3102     img=dose_slice[ int( top_left_down ):int( bottom_left_down ), int(
3103 top_left_to_side ):int( top_right_to_side ) ]
3104     """
3105     if(Globals . profiles_number_of_doseplans == 1):
3106         Globals . profiles_doseplan_dataset_ROI_several = img
3107         Globals . profiles_number_of_doseplans +=1
3108
3109         if( Globals . profiles_dataset_doseplan . PixelSpacing == [1, 1]):
3110             Globals . profiles_several_img = cv2.resize(img, dsize=(img .
3111 shape[1]*5,img . shape[0]*5))
3112             elif( Globals . profiles_dataset_doseplan . PixelSpacing == [2, 2]):
3113                 Globals . profiles_several_img = cv2.resize(img, dsize=(img .
3114 shape[1]*10,img . shape[0]*10))
3115                 else :
3116                     Globals . profiles_several_img = cv2.resize(img, dsize=(img .
3117 shape[1]*15,img . shape[0]*15))
3118
3119             else :
3120                 Globals . profiles_doseplan_dataset_ROI_several += img
3121                 Globals . profiles_number_of_doseplans +=1
3122
3123                 if( Globals . profiles_dataset_doseplan . PixelSpacing == [1, 1]):
3124                     Globals . profiles_several_img += cv2.resize(img, dsize=(img .
3125 shape[1]*5,img . shape[0]*5))
3126                     elif( Globals . profiles_dataset_doseplan . PixelSpacing == [2, 2]):
3127                         Globals . profiles_several_img += cv2.resize(img, dsize=(img .
3128 shape[1]*10,img . shape[0]*10))
3129                         else :
3130                             Globals . profiles_several_img += cv2.resize(img, dsize=(img .
3131 shape[1]*15,img . shape[0]*15))
3132
3133                     Globals . profiles_doseplan_dataset_ROI_several.append(img)
3134                     Globals . profiles_number_of_doseplans +=1
3135
3136
3137
3138

```

```

3140     def UploadRTplan():
3141         file = filedialog.askopenfilename()
3142         ext = os.path.splitext(file)[-1].lower()
3143         if(not(ext == '.dcm')):
3144             if(ext == ""):
3145                 return
3146             else:
3147                 messagebox.showerror("Error", "The file must be a *.dcm file")
3148                 return
3149
3150         current_folder = os.getcwd()
3151         parent = os.path.dirname(file)
3152         os.chdir(parent)
3153         dataset = pydicom.dcmread(file)
3154         os.chdir(current_folder)
3155         Globals.profiles_dataset_rtplan = dataset
3156
3157         #Isocenter given in mm from origo in patient coordinate system
3158         try:
3159             isocenter_mm = dataset.BeamSequence[0].ControlPointSequence[0].
3160             IsocenterPosition
3161             Globals.profiles_isocenter_mm = isocenter_mm
3162
3163         except:
3164             messagebox.showerror("Error", "Could not read the RT plan file.
3165             Try again or try another file.\n\
3166             (Code: isocenter reading)")
3167             return
3168
3169         try:
3170             Globals.profiles_doseplan_vertical_displacement = dataset.
3171             PatientSetupSequence[0].TableTopVerticalSetupDisplacement
3172         except:
3173             messagebox.showerror("Error", "Could not read the RT plan file.
3174             Try again or try another file.\n\
3175             (Code: vertical table displacement)")
3176
3177         try:
3178             Globals.profiles_doseplan_lateral_displacement = dataset.
3179             PatientSetupSequence[0].TableTopLateralSetupDisplacement
3180         except:
3181             messagebox.showerror("Error", "Could not read the RT plan file.
3182             Try again or try another file.\n\
3183             (Code: lateral table displacement)")
3184
3185         try:
3186             Globals.profiles_doseplan_longitudinal_displacement = dataset.
3187             PatientSetupSequence[0].TableTopLongitudinalSetupDisplacement
3188         except:
3189             messagebox.showerror("Error", "Could not read the RT plan file.
3190             Try again or try another file.\n\
3191             (Code: longitudinal table displacement)")
3192
3193         try:
3194             patient_position = dataset.PatientSetupSequence[0].PatientPosition
3195             Globals.profiles_doseplan_patient_position = patient_position
3196         except:

```

```

3188     messagebox.showerror("Error", "Could not read the RT plan file.\nTry again or try another file\\n\\\n(Code: Patient position)")
3190
3191     if(not(patient_position=='HFS' or patient_position=='HFP' or
3192     patient_position=='HFDR' or patient_position == 'HFDL'\\
3193         or patient_position=='FFDR' or patient_position== 'FFDL' or
3194     patient_position=='FFP' or patient_position== 'FFS')):
3195         messagebox.showerror("Error", "Fidora does only support patient
3196         positions:\\n\\\nHFS, HFP, HFDR, HFDL, FFP, FFS, FFDR, FFDL")
3197         return
3198
3199     Globals.profiles_test_if_added_rtplan = True
3200     #if(Globals.profiles_test_if_added_doseplan):
3201     #    if(Globals.profiles_isocenter_or_reference_point == "Isocenter"):
3202     #        processDoseplan_usingIsocenter(only_one)
3203     #    elif(Globals.profiles_isocenter_or_reference_point == "Ref_point"
3204     #    ):
3205     #        processDoseplan_usingReferencePoint(only_one)
3206     #    else:
3207     #        messagebox.showerror("Error", "Something went wrong. Try
3208     #again.\\n\\\n(Code: processDoseplan)")
3209     #        return
3210     Globals.upload_button_doseplan.config(state=ACTIVE)
3211     Globals.upload_button_rtplan.config(state=DISABLED)
3212
3213     def UploadDoseplan_button_function():
3214         yes = messagebox.askyesno("Question", "Are you going to upload several
3215         doseplans and/or use a factor on a plan?")
3216         if not yes:
3217             UploadDoseplan(True)
3218             return
3219
3220         several_doseplans_window = tk.Toplevel(Globals.tab4_canvas)
3221         several_doseplans_window.geometry("600x500+10+10")
3222         several_doseplans_window.grab_set()
3223
3224         doseplans_over_all_frame = tk.Frame(several_doseplans_window, bd=0,
3225             relief=FLAT)
3226         doseplans_over_all_canvas = Canvas(doseplans_over_all_frame)
3227
3228         doseplans_xscrollbar = Scrollbar(doseplans_over_all_frame, orient=
3229             HORIZONTAL, command=doseplans_over_all_canvas.xview)
3230         doseplans_yscrollbar = Scrollbar(doseplans_over_all_frame, command=
3231             doseplans_over_all_canvas.yview)
3232
3233         Globals.doseplans_scroll_frame = ttk.Frame(doseplans_over_all_canvas)
3234         Globals.doseplans_scroll_frame.bind("<Configure>", lambda e:
3235             doseplans_over_all_canvas.configure(scrollregion=
3236                 doseplans_over_all_canvas.bbox('all')))
3237
3238         doseplans_over_all_canvas.create_window((0,0), window=Globals.
3239             doseplans_scroll_frame, anchor='nw')
3240         doseplans_over_all_canvas.configure(xscrollcommand=
3241             doseplans_xscrollbar.set, yscrollcommand=doseplans_yscrollbar.set)

```

```

3232 doseplans_over_all_frame.config(highlightthickness=0, bg='#ffffff')
3233 doseplans_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
3234 doseplans_over_all_frame.pack(expand=True, fill=BOTH)
3235 doseplans_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
3236 doseplans_over_all_frame.grid_columnconfigure(0, weight=1)
3237 doseplans_over_all_frame.grid_rowconfigure(0, weight=1)
3238 doseplans_xscrollbar.grid(row=1, column=0, sticky=E+W)
3239 doseplans_over_all_frame.grid_columnconfigure(1, weight=0)
3240 doseplans_over_all_frame.grid_rowconfigure(1, weight=0)
3241 doseplans_yscrollbar.grid(row=0, column=1, sticky=N+S)
3242 doseplans_over_all_frame.grid_columnconfigure(2, weight=0)
3243 doseplans_over_all_frame.grid_rowconfigure(2, weight=0)
3244
3245 upload_doseplan_frame = tk.Frame(Globals.doseplans_scroll_frame)
3246 upload_doseplan_frame.grid(row=0, column = 0, padx = (30,30), pady
=(30,0), sticky=N+S+E+W)
3247 Globals.doseplans_scroll_frame.grid_columnconfigure(0, weight=0)
3248 Globals.doseplans_scroll_frame.grid_rowconfigure(0, weight=0)
3249 upload_doseplan_frame.config(bg = '#ffffff')
3250
3251 upload_button_doseplan = tk.Button(upload_doseplan_frame, text='Browse',
3252 image=Globals.profiles_add_doseplans_button_image, \
3253 cursor='hand2', font=('calibri', '14'), relief=FLAT, state=ACTIVE,
3254 command=lambda: UploadDoseplan(False))
3255 upload_button_doseplan.pack(expand=True, fill=BOTH)
3256 upload_button_doseplan.configure(bg='#ffffff', activebackground="#
ffffff", activeforeground='#ffffff', highlightthickness=0)
3257 upload_button_doseplan.image = Globals.
3258 profiles_add_doseplans_button_image
3259
3260 def closeUploadDoseplans():
3261     if(len(Globals.profiles_doseplan_dataset_ROI_several) == 0):
3262         messagebox.showinfo("INFO", "No doseplan has been uploaded")
3263         return
3264     for i in range(len(Globals.profiles_doseplan_dataset_ROI_several)):
3265         :
3266             if Globals.profiles_doseplans_factor_input[i].get("1.0", 'end
-1c') == " ":
3267                 factor = 1
3268             else:
3269                 try:
3270                     factor = float(Globals.profiles_doseplans_factor_input
[i].get("1.0", 'end-1c'))
3271                 except:
3272                     messagebox.showerror("Error", "Invalid factor. Must be
number.\n (Code: closeUploadDoseplans)")
3273                     return
3274             if i == 0:
3275                 doseplan_ROI = Globals.
3276 profiles_doseplan_dataset_ROI_several[i]*Globals.
3277 profiles_dose_scaling_doseplan[i]
3278                 doseplan_ROI=doseplan_ROI*factor
3279
3280                 img_ROI = Globals.profiles_several_img[i]*Globals.
3281 profiles_dose_scaling_doseplan[i]
3282                 img_ROI = img_ROI*factor

```

```

3276     else :
3277         doseplan_ROI+= factor*Globals .
3278     profiles_doseplan_dataset_ROI_several[i]*Globals .
3279     profiles_dose_scaling_doseplan[i]
3280         img_ROI+= factor*Globals . profiles_several_img [ i ]*Globals .
3281     profiles_dose_scaling_doseplan[i]

3282
3283     img_ROI = cv2 . resize (img_ROI , dsize=(img_ROI . shape [1]*5 ,img_ROI .
3284     shape [0]*5))
3285     Globals . profiles_doseplan_dataset_ROI = doseplan_ROI
3286     mx=np . max (img_ROI)
3287     Globals . max_dose_doseplan = mx
3288     img_ROI = img_ROI/mx
3289     PIL_img_doseplan_ROI = Image . fromarray (np . uint8 (cm . viridis (img_ROI
3290     )*255))

3291     wid = PIL_img_doseplan_ROI . width ;heig = PIL_img_doseplan_ROI .
3292     height
3293     doseplan_canvas = tk . Canvas (Globals . profiles_film_panedwindow)
3294     doseplan_canvas . grid (row=2 , column=0 , sticky=N+S+W+E)
3295     Globals . profiles_film_panedwindow . add (doseplan_canvas , \
3296         height=max (heig , Globals . profiles_doseplan_text_image . height ())
3297     , \
3298         width=wid + Globals . profiles_doseplan_text_image . width ())
3299     doseplan_canvas . config (bg='#ffffff' , relief=FLAT,
3300     highlightthickness=0 , \
3301         height=max (heig , Globals . profiles_doseplan_text_image . height ())
3302     , \
3303         width=wid + Globals . profiles_doseplan_text_image . width ())

3304
3305     Globals . doseplan_write_image = tk . Canvas (doseplan_canvas)
3306     Globals . doseplan_write_image . grid (row=0 , column=1 , sticky=N+S+W+E)
3307     Globals . doseplan_write_image . config (bg='#ffffff' , relief=FLAT,
3308     highlightthickness=0 , width=wid , height=heig)

3309     doseplan_text_image_canvas = tk . Canvas (doseplan_canvas)
3310     doseplan_text_image_canvas . grid (row=0 , column=0 , sticky=N+S+W+E)
3311     doseplan_text_image_canvas . config (bg='#ffffff' , relief=FLAT,
3312     highlightthickness=0 , \
3313         width=Globals . profiles_doseplan_text_image . width () , height=
3314         Globals . profiles_doseplan_text_image . height ())

3315
3316     scaled_image_visual = PIL_img_doseplan_ROI
3317     scaled_image_visual = ImageTk . PhotoImage (image=scaled_image_visual
3318 )
3319     Globals . doseplan_write_image_width = scaled_image_visual . width ()
3320     Globals . doseplan_write_image_height = scaled_image_visual . height ()
3321     Globals . doseplan_write_image . create_image (0 ,0 ,image=
3322         scaled_image_visual , anchor="nw")
3323     Globals . doseplan_write_image . image = scaled_image_visual
3324     doseplan_text_image_canvas . create_image (0 ,0 ,image=Globals .
3325     profiles_doseplan_text_image , anchor="nw")
3326         doseplan_text_image_canvas . image=Globals .
3327     profiles_doseplan_text_image

```

```

3318     Globals.profiles_doseplan_dataset_ROI = doseplan_ROI
3319
3320     Globals.profiles_upload_button_doseplan.config(state=DISABLED)
3321
3322     several_doseplans_window.after(500, lambda:
3323         several_doseplans_window.destroy())
3324     drawProfiles(False)
3325
3326     doseplans_done_button_frame = tk.Frame(Globals.doseplans_scroll_frame)
3327     doseplans_done_button_frame.grid(row=0, column = 1, padx=(0,40), pady
3328     =(30,0), sticky=N+S+W+E)
3329     doseplans_done_button_frame.config(bg='#ffffffff')
3330     Globals.doseplans_scroll_frame.grid_rowconfigure(3, weight=0)
3331     Globals.doseplans_scroll_frame.grid_columnconfigure(3, weight=0)
3332
3333     doseplans_done_button = tk.Button(doseplans_done_button_frame, text='
3334     Done', image=Globals.done_button.image,
3335         cursor='hand2', font=('calibri', '14'), relief=FLAT, state=ACTIVE,
3336         command=closeUploadDoseplans)
3337     doseplans_done_button.pack(expand=True, fill=BOTH)
3338     doseplans_done_button.configure(bg='#ffffffff', activebackground='#
3339     ffffff', activeforeground='#ffffff', highlightthickness=0)
3340     doseplans_done_button.image = Globals.done_button.image
3341
3342
3343     filename_title = tk.Text(Globals.doseplans_scroll_frame, width = 15,
3344     height= 1)
3345     filename_title.insert(INSERT, "Filename")
3346     filename_title.grid(row=2, column=0, sticky=N+S+E+W, pady=(40,0), padx
3347     =(45,15))
3348     filename_title.config(bg='#ffffffff', relief=FLAT, state=DISABLED, font
3349     =('calibri', '15', 'bold'))
3350     Globals.doseplans_scroll_frame.grid_rowconfigure(1, weight=0)
3351     Globals.doseplans_scroll_frame.grid_columnconfigure(1, weight=0)
3352
3353
3354     factor_title = tk.Text(Globals.doseplans_scroll_frame, width=30,
3355     height=2)
3356     factor_title.insert(INSERT, "Here you can write a factor to use \non
3357     the doseplan. Defaults to 1.")
3358     factor_title.grid(row=2, column=1, sticky=N+W+S+E, pady=(37,10), padx
3359     =(15,25))
3360     factor_title.config(bg='#ffffffff', relief=FLAT, state=DISABLED, font=(

3361     'calibri', '15', 'bold'))
3362     Globals.doseplans_scroll_frame.grid_columnconfigure(2, weight=0)
3363     Globals.doseplans_scroll_frame.grid_rowconfigure(2, weight=0)
3364
3365
3366 def UploadDoseplan(only_one):
3367     file = filedialog.askopenfilename()
3368     ext = os.path.splitext(file)[-1].lower()
3369     if(not(ext == '.dcm')):
3370         if(ext == ""):
3371             return
3372         else:
3373             messagebox.showerror("Error", "The file must be a *.dcm file")
3374             return

```

```

3362     current_folder = os.getcwd()
3364     parent = os.path.dirname(file)
3365     os.chdir(parent)
3366     dataset = pydicom.dcmread(file)
3367     try:
3368         dose_summation_type = dataset.DoseSummationType
3369     except:
3370         messagebox.showerror("Error", "Could not upload the doseplan\n"
3371                             "correctly. Try again or another file.\n(Code: dose summation)")
3372         return
3373
3374     if(not(dose_summation_type == "PLAN")):
3375         ok = messagebox.askokcancel("Dose summation", "You did not upload\n"
3376                                   "the full doseplan. Do you want to continue?")
3377         if not ok:
3378             return
3379     os.chdir(current_folder)
3380     doseplan_dataset = dataset.pixel_array
3381     #Check that the resolution is either 1x1x1, 2x2x2 or 3x3x3
3382     if(not((dataset.PixelSpacing==[1, 1] and dataset.SliceThickness==1) \
3383            or (dataset.PixelSpacing==[2, 2] and dataset.SliceThickness==2) \
3384            or (dataset.PixelSpacing==[3, 3] and dataset.SliceThickness==3))):
3385         messagebox.showerror("Error", "The resolution in doseplan must be\n"
3386                           "1x1x1, 2x2x2 or 3x3x3")
3387         return
3388     #Check that the datamatrix is in right angles to the coordinate system
3389     if(not(dataset.ImageOrientationPatient==[1, 0, 0, 0, 1, 0] or \
3390            dataset.ImageOrientationPatient==[1, 0, 0, 0, 0, 1] or \
3391            dataset.ImageOrientationPatient==[0, 1, 0, 1, 0, 0] or \
3392            dataset.ImageOrientationPatient==[0, 1, 0, 0, 0, 1] or \
3393            dataset.ImageOrientationPatient==[0, 0, 1, 1, 0, 0] or \
3394            dataset.ImageOrientationPatient==[0, 0, 1, 0, 1, 0])):
3395         messagebox.showerror("Error", "The Image Orientation (Patient)\n"
3396                           "must be parallel to one of the main axis and perpendicular to the two\n"
3397                           "others.")
3398         return
3399
3400     if not only_one and Globals.profiles_number_of_doseplans > 1:
3401         if(not (Globals.profiles_dataset_doseplan.PixelSpacing==dataset.PixelSpacing)):
3402             messagebox.showerror("Error", "Resolution of the doseplans\n"
3403                               "must be equal. \n(Code: UploadDoseplan)")
3404             return
3405         if(not (Globals.profiles_dataset_doseplan.DoseGridScaling ==
3406                dataset.DoseGridScaling)):
3407             messagebox.showerror("Error", "Dose grid scaling of the\n"
3408                               "doseplans must be equal. \n(Code: UploadDoseplan)")
3409             return
3410     Globals.profiles_dataset_doseplan = dataset
3411     Globals.profiles_dose_scaling_doseplan.append(dataset.DoseGridScaling)
3412     Globals.profiles_test_if_added_doseplan = True
3413     if(Globals.profiles_test_if_added_rtplan):
3414         if(Globals.profiles_isocenter_or_reference_point == "Isocenter"):
3415             processDoseplan_usingIsocenter(only_one)
3416         elif(Globals.profiles_isocenter_or_reference_point == "Ref_point"):
3417             :

```

```

    processDoseplan_usingReferencePoint(only_one)
3410   else:
3411     messagebox.showerror("Error", "Something went wrong. Try again
3412     .\n (Code: processDoseplan)")
3413     return
3414
3415   if only_one:
3416     Globals.profiles_upload_button_doseplan.config(state=DISABLED)
3417
3418   if not only_one:
3419     filename = basename(normpath(file))
3420     textbox_filename = tk.Text(Globals.doseplans_scroll_frame, width =
3421       30, height = 1)
3422     textbox_filename.insert(INSERT, filename)
3423     textbox_filename.config(bg='#ffffff', font=('calibri', '12'),
3424       state=DISABLED, relief=FLAT)
3425     textbox_filename.grid(row = Globals.
3426     profiles_number_of_doseplans_row_count, column = 0, sticky=N+S+W+E,
3427     pady=(10,10), padx=(10,10))
3428     Globals.doseplans_scroll_frame.grid_columnconfigure(Globals.
3429     profiles_doseplans_grid_config_count, weight=0)
3430     Globals.doseplans_scroll_frame.grid_rowconfigure(Globals.
3431     profiles_doseplans_grid_config_count, weight=0)
3432     Globals.profiles_doseplans_filenames.append(textbox_filename)
3433
3434     Globals.profiles_doseplans_grid_config_count+=1;
3435
3436     textbox_factor = tk.Text(Globals.doseplans_scroll_frame, width =
3437       6, height = 1)
3438     textbox_factor.insert(INSERT, "Factor: ")
3439     textbox_factor.config(bg='#ffffff', font=('calibri', '12'), state=
3440       DISABLED, relief=FLAT)
3441     textbox_factor.grid(row = Globals.
3442     profiles_number_of_doseplans_row_count, column = 1, sticky=N+S+W+E,
3443     pady=(10,10), padx=(10,10))
3444     Globals.doseplans_scroll_frame.grid_columnconfigure(Globals.
3445     profiles_doseplans_grid_config_count, weight=0)
3446     Globals.doseplans_scroll_frame.grid_rowconfigure(Globals.
3447     profiles_doseplans_grid_config_count, weight=0)
3448     Globals.profiles_doseplans_factor_text.append(textbox_factor)
3449
3450     Globals.profiles_doseplans_grid_config_count+=1;
3451
3452     textbox_factor_input = tk.Text(Globals.doseplans_scroll_frame,
3453       width=3, height=1)
3454     textbox_factor_input.insert(INSERT, " ")
3455     textbox_factor_input.config(bg='#E5f9ff', font=('calibri', '12'),
3456       state=NORMAL, bd = 2)
3457     textbox_factor_input.grid(row = Globals.
3458     profiles_number_of_doseplans_row_count, column = 1, sticky=N+S, pady
3459       =(10,10), padx=(40,10))
3460     Globals.doseplans_scroll_frame.grid_columnconfigure(Globals.
3461     profiles_doseplans_grid_config_count, weight=0)
3462     Globals.doseplans_scroll_frame.grid_rowconfigure(Globals.
3463     profiles_doseplans_grid_config_count, weight=0)
3464     Globals.profiles_doseplans_factor_input.append(
3465       textbox_factor_input)

```

```

3446     Globals . profiles_number_of_doseplans_row_count+=1
3448     Globals . profiles_doseplans_grid_config_count+=1;
3450
3451 ##### F I L M #####
3452 def markIsocenter(img , new_window_isocenter_tab , image_canvas , cv2Img):
3453     if(len(Globals . profiles_mark_isocenter_oval)>0):
3454         image_canvas.delete(Globals . profiles_mark_isocenter_up_down_line
3455 [0])
3456         image_canvas.delete(Globals .
3457 profiles_mark_isocenter_right_left_line [0])
3458         image_canvas.delete(Globals . profiles_mark_isocenter_oval [0])
3459
3460     Globals . profiles_mark_isocenter_oval=[]
3461     Globals . profiles_mark_isocenter_right_left_line []
3462     Globals . profiles_mark_isocenter_up_down_line []
3463
3464     Globals . profiles_iscoenter_coords = []
3465     img_mark_isocenter = ImageTk . PhotoImage(image=img)
3466     mark_isocenter_window = tk . Toplevel(new_window_isocenter_tab)
3467     mark_isocenter_window.geometry("1035x620+10+10")
3468     mark_isocenter_window.grab_set()
3469
3470     mark_isocenter_over_all_frame = tk . Frame(mark_isocenter_window , bd=0,
3471         relief=FLAT)
3472     mark_isocenter_over_all_canvas = Canvas(mark_isocenter_over_all_frame)
3473
3474     mark_isocenter_xscrollbar = Scrollbar(mark_isocenter_over_all_frame ,
3475         orient=HORIZONTAL, command=mark_isocenter_over_all_canvas . xview)
3476     mark_isocenter_yscrollbar = Scrollbar(mark_isocenter_over_all_frame ,
3477         command=mark_isocenter_over_all_canvas . yview)
3478
3479     mark_isocenter_scroll_frame = ttk . Frame(mark_isocenter_over_all_canvas)
3480     mark_isocenter_scroll_frame.bind("<Configure>" , lambda e:
3481         mark_isocenter_over_all_canvas . configure(scrollregion=
3482             mark_isocenter_over_all_canvas . bbox('all')))
3483
3484     mark_isocenter_over_all_canvas.create_window((0,0) , window=
3485         mark_isocenter_scroll_frame , anchor='nw')
3486     mark_isocenter_over_all_canvas.configure(xscrollcommand=
3487         mark_isocenter_xscrollbar . set , yscrollcommand=
3488         mark_isocenter_yscrollbar . set)
3489
3490     mark_isocenter_over_all_frame.config(highlightthickness=0, bg='#ffffff')
3491     mark_isocenter_over_all_canvas.config(highlightthickness=0, bg='#
3492         ffffff')
3493     mark_isocenter_over_all_frame.pack(expand=True , fill=BOTH)
3494     mark_isocenter_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
3495     mark_isocenter_over_all_frame.grid_columnconfigure(0, weight=1)
3496     mark_isocenter_over_all_frame.grid_rowconfigure(0, weight=1)
3497     mark_isocenter_xscrollbar.grid(row=1, column=0, sticky=E+W)
3498     mark_isocenter_over_all_frame.grid_columnconfigure(1, weight=0)
3499     mark_isocenter_over_all_frame.grid_rowconfigure(1, weight=0)

```

```

3490     mark_isocenter_yscrollbar.grid(row=0, column=1, sticky=N+S)
3491     mark_isocenter_over_all_frame.grid_columnconfigure(2, weight=0)
3492     mark_isocenter_over_all_frame.grid_rowconfigure(2, weight=0)
3493
3494     mark_isocenter_image_canvas = tk.Canvas(mark_isocenter_scroll_frame)
3495     mark_isocenter_image_canvas.grid(row=0, column=0, rowspan=10,
3496         columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
3497     mark_isocenter_scroll_frame.grid_columnconfigure(0, weight=0)
3498     mark_isocenter_scroll_frame.grid_rowconfigure(0, weight=0)
3499
3500     mark_isocenter_image_canvas.create_image(0,0,image=img_mark_isocenter,
3501         anchor="nw")
3502     mark_isocenter_image_canvas.image = img_mark_isocenter
3503     mark_isocenter_image_canvas.config(cursor='hand2', bg='#ffffff',
3504         relief=FLAT, bd=0, \
3505             scrollregion=mark_isocenter_image_canvas.bbox(ALL), height=
3506             img_mark_isocenter.height(), width=img_mark_isocenter.width())
3507     mark_isocenter_image_canvas.grid_propagate(0)
3508
3509     def findCoords(event):
3510         mark_isocenter.image_canvas.create_oval(event.x-2, event.y-2,
3511             event.x+2, event.y+2, fill='red')
3512         if(Globals.profiles_iscoenter_coords==[]):
3513             Globals.profiles_iscoenter_coords.append([event.x, event.y])
3514             mark_isocenter_image_canvas.config(cursor='hand2')
3515
3516         elif(len(Globals.profiles_iscoenter_coords)==1):
3517             Globals.profiles_iscoenter_coords.append([event.x, event.y])
3518             Globals.profiles_film_isocenter = [Globals.
3519             profiles_iscoenter_coords[0][0], Globals.profiles_iscoenter_coords
3520             [1][1]]
3521             x1,y1 = Globals.profiles_iscoenter_coords[0]
3522             x4,y4 = Globals.profiles_iscoenter_coords[1]
3523             x2 = x1;y3=y4
3524             y2=2*Globals.profiles_film_isocenter[1]-y1
3525             x3=2*Globals.profiles_film_isocenter[0]-x4
3526             up_down_line = image_canvas.create_line(int(x1/2), int(y1/2),
3527                 int(x2/2), int(y2/2), fill='purple', smooth=1, width=2)
3528             right_left_line = image_canvas.create_line(int(x3/2), int(y3/2),
3529                 int(x4/2), int(y4/2), fill='purple', smooth=1, width=2)
3530             oval = image_canvas.create_oval(int(Globals.
3531             profiles_film_isocenter[0]/2)-3, int(Globals.profiles_film_isocenter
3532             [1]/2)-3,\n                int(Globals.profiles_film_isocenter[0]/2)+3, int(Globals.
3533             profiles_film_isocenter[1]/2)+3, fill='red')
3534
3535             Globals.profiles_mark_isocenter_up_down_line.append(
3536                 up_down_line)
3537             Globals.profiles_mark_isocenter_right_left_line.append(
3538                 right_left_line)
3539             Globals.profiles_mark_isocenter_oval.append(oval)
3540
3541             mark_isocenter_window.after(500, lambda: mark_isocenter_window
3542                 .destroy())
3543             Globals.profiles_isocenter_check = True
3544             if(Globals.profiles_ROI_check):
3545                 Globals.profiles_done_button.config(state=ACTIVE)

```

```

3532     mark_isocenter_image_canvas.bind("<Button 1>", findCoords)

3534 def markReferencePoint(img, new_window_reference_point_tab,
3535     image_canvas_reference_tab, cv2Img):
3536
3537     if(len(Globals.profiles_mark_reference_point_oval)>0):
3538         image_canvas_reference_tab.delete(Globals.
3539             profiles_mark_reference_point_oval[0])
3540         Globals.profiles_mark_reference_point_oval=[]
3541
3542     img_mark_reference_point = ImageTk.PhotoImage(image=img)
3543     mark_reference_point_window = tk.Toplevel(
3544         new_window_reference_point_tab)
3545     mark_reference_point_window.geometry("1035x620+10+10")
3546     mark_reference_point_window.grab_set()
3547
3548     mark_reference_point_over_all_frame = tk.Frame(
3549         mark_reference_point_window, bd=0, relief=FLAT)
3550     mark_reference_point_over_all_canvas = Canvas(
3551         mark_reference_point_over_all_frame)
3552
3553     mark_reference_point_xscrollbar = Scrollbar(
3554         mark_reference_point_over_all_frame, orient=HORIZONTAL, command=
3555         mark_reference_point_over_all_canvas.xview)
3556     mark_reference_point_yscrollbar = Scrollbar(
3557         mark_reference_point_over_all_frame, command=
3558         mark_reference_point_over_all_canvas.yview)
3559
3560     mark_reference_point_scroll_frame = ttk.Frame(
3561         mark_reference_point_over_all_canvas)
3562     mark_reference_point_scroll_frame.bind("<Configure>", lambda e:
3563         mark_reference_point_over_all_canvas.configure(scrollregion=
3564             mark_reference_point_over_all_canvas.bbox('all')))
3565
3566     mark_reference_point_over_all_canvas.create_window((0,0), window=
3567         mark_reference_point_scroll_frame, anchor='nw')
3568     mark_reference_point_over_all_canvas.configure(xscrollcommand=
3569         mark_reference_point_xscrollbar.set, yscrollcommand=
3570         mark_reference_point_yscrollbar.set)
3571
3572     mark_reference_point_over_all_frame.config(highlightthickness=0, bg="#
3573         ffffff")
3574     mark_reference_point_over_all_canvas.config(highlightthickness=0, bg=
3575         "#ffffff")
3576     mark_reference_point_over_all_frame.pack(expand=True, fill=BOTH)
3577     mark_reference_point_over_all_canvas.grid(row=0, column=0, sticky=N+S+
3578         E+W)
3579     mark_reference_point_over_all_frame.grid_columnconfigure(0, weight=1)
3580     mark_reference_point_over_all_frame.grid_rowconfigure(0, weight=1)
3581     mark_reference_point_xscrollbar.grid(row=1, column=0, sticky=E+W)
3582     mark_reference_point_over_all_frame.grid_columnconfigure(1, weight=0)
3583     mark_reference_point_over_all_frame.grid_rowconfigure(1, weight=0)
3584     mark_reference_point_yscrollbar.grid(row=0, column=1, sticky=N+S)
3585     mark_reference_point_over_all_frame.grid_columnconfigure(2, weight=0)
3586     mark_reference_point_over_all_frame.grid_rowconfigure(2, weight=0)

```

```

3570 mark_reference_point_image_canvas = tk.Canvas(
3571     mark_reference_point_scroll_frame)
3572     mark_reference_point_image_canvas.grid(row=0, column=0, rowspan=10,
3573     columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
3574     mark_reference_point_scroll_frame.grid_columnconfigure(0, weight=0)
3575     mark_reference_point_scroll_frame.grid_rowconfigure(0, weight=0)

3576     mark_reference_point_image_canvas.create_image(0,0,image=
3577         img_mark_reference_point, anchor="nw")
3578     mark_reference_point_image_canvas.image = img_mark_reference_point
3579     mark_reference_point_image_canvas.config(cursor='hand2', bg='#ffffff',
3580     relief=FLAT, bd=0, \
3581         scrollregion=mark_reference_point_image_canvas.bbox(ALL), height=
3582         img_mark_reference_point.height(), width=img_mark_reference_point.
3583         width())
3584     mark_reference_point_image_canvas.grid_propagate(0)

3585
3586     def findCoords(event):
3587         mark_reference_point_image_canvas.create_oval(event.x-2, event.y
3588             -2, event.x+2, event.y+2, fill='red')
3589         Globals.profiles_film_reference_point = [event.x, event.y]
3590         oval = image_canvas_reference_tab.create_oval(int(Globals.
3591         profiles_film_reference_point[0]/2)-3, \
3592             int(Globals.profiles_film_reference_point[1]/2)-3, int(Globals
3593             .profiles_film_reference_point[0]/2)+3, \
3594             int(Globals.profiles_film_reference_point[1]/2)+3, fill='red')
3595         Globals.profiles_mark_reference_point_oval.append(oval)

3596
3597         mark_reference_point_window.after(500, lambda:
3598             mark_reference_point_window.destroy())
3599             Globals.profiles_reference_point_check = True
3600             if(Globals.profiles_ROI_reference_point_check):
3601                 Globals.profiles_done_button_reference_point.config(state=
3602                     ACTIVE)

3603             mark_reference_point_image_canvas.bind("<Button 1>",findCoords)

3604     def markROI(img, tab, canvas, ref_point_test):
3605         if(len(Globals.profiles_mark_ROI_rectangle)>0):
3606             canvas.delete(Globals.profiles_mark_ROI_rectangle[0])
3607             Globals.profiles_mark_ROI_rectangle = []
3608
3609             Globals.profiles_ROI_coords = []

3610             img_mark_ROI = ImageTk.PhotoImage(image=img)
3611             mark_ROI_window = tk.Toplevel(tab)
3612             mark_ROI_window.geometry("1035x620+10+10")
3613             mark_ROI_window.grab_set()

3614             mark_ROI_over_all_frame = tk.Frame(mark_ROI_window, bd=0, relief=FLAT)
3615             mark_ROI_over_all_canvas = Canvas(mark_ROI_over_all_frame)

3616             mark_ROI_xscrollbar = Scrollbar(mark_ROI_over_all_frame, orient=
3617                 HORIZONTAL, command=mark_ROI_over_all_canvas.xview)

```

```

3614     mark_ROI_yscrollbar = Scrollbar(mark_ROI_over_all_frame , command=
3615         mark_ROI_over_all_canvas.yview)

3616     mark_ROI_scroll_frame = ttk.Frame(mark_ROI_over_all_canvas)
3617     mark_ROI_scroll_frame.bind("<Configure>" , lambda e:
3618         mark_ROI_over_all_canvas.configure(scrollregion=
3619             mark_ROI_over_all_canvas.bbox('all')))

3620     mark_ROI_over_all_canvas.create_window((0,0) , window=
3621         mark_ROI_scroll_frame , anchor='nw')
3622     mark_ROI_over_all_canvas.configure(xscrollcommand=mark_ROI_xscrollbar.
3623         set , yscrollcommand=mark_ROI_yscrollbar.set)

3624     mark_ROI_over_all_frame.config(highlightthickness=0, bg='#ffffff')
3625     mark_ROI_over_all_canvas.config(highlightthickness=0, bg='#ffffff')
3626     mark_ROI_over_all_frame.pack(expand=True, fill=BOTH)
3627     mark_ROI_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
3628     mark_ROI_over_all_frame.grid_columnconfigure(0, weight=1)
3629     mark_ROI_over_all_frame.grid_rowconfigure(0, weight=1)
3630     mark_ROI_xscrollbar.grid(row=1, column=0, sticky=E+W)
3631     mark_ROI_over_all_frame.grid_columnconfigure(1, weight=0)
3632     mark_ROI_over_all_frame.grid_rowconfigure(1, weight=0)
3633     mark_ROI_yscrollbar.grid(row=0, column=1, sticky=N+S)
3634     mark_ROI_over_all_frame.grid_columnconfigure(2, weight=0)
3635     mark_ROI_over_all_frame.grid_rowconfigure(2, weight=0)

3636     mark_ROI_image_canvas = tk.Canvas(mark_ROI_scroll_frame)
3637     mark_ROI_image_canvas.grid(row=0,column=0, rowspan=10, columnspan=3,
3638         sticky=N+S+E+W, padx=(0,0), pady=(0,0))
3639     mark_ROI_scroll_frame.grid_columnconfigure(0, weight=0)
3640     mark_ROI_scroll_frame.grid_rowconfigure(0, weight=0)
3641     mark_ROI_image_canvas.create_image(0,0,image=img_mark_ROI, anchor="nw")
3642     mark_ROI_image_canvas.image = img_mark_ROI
3643     mark_ROI_image_canvas.config(bg='#E5f9ff' , relief=FLAT, bd=0, \
3644         scrollregion=mark_ROI_image_canvas.bbox(ALL) , height=img_mark_ROI.
3645         height() , width=img_mark_ROI.width())
3646     mark_ROI_image_canvas.grid_propagate(0)

3647     rectangle = mark_ROI_image_canvas.create_rectangle(0,0,0,0, outline='
3648         green')
3649     rectangle_top_corner = []
3650     rectangle_bottom_corner = []
3651     def buttonPushed(event):
3652         rectangle_top_corner.append([event.x, event.y])

3653     def buttonMoving(event):
3654         mark_ROI_image_canvas.coords(rectangle , rectangle_top_corner
3655             [0][0], rectangle_top_corner[0][1], \
3656             event.x, event.y)

3657     def buttonReleased(event):
3658         rectangle_bottom_corner.append([event.x, event.y])
3659         mark_ROI_image_canvas.coords(rectangle , rectangle_top_corner
3660             [0][0], rectangle_top_corner[0][1], \
3661             rectangle_bottom_corner[0][0], rectangle_bottom_corner[0][1])
3662         mark_ROI_image_canvas.itemconfig(rectangle , outline='Blue')

```

```

3660     ### Husk at koordinatene g r bortover s nedover! Top left - top
right - bottom left - bottom right
3661     Globals.profiles_ROI_coords.append([rectangle_top_corner[0][0],
rectangle_top_corner[0][1]])
3662     Globals.profiles_ROI_coords.append([rectangle_bottom_corner[0][0],
rectangle_top_corner[0][1]])
3663     Globals.profiles_ROI_coords.append([rectangle_top_corner[0][0],
rectangle_bottom_corner[0][1]])
3664     Globals.profiles_ROI_coords.append([rectangle_bottom_corner[0][0],
rectangle_bottom_corner[0][1]])
3665
3666     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)
3667     Globals.profiles_mark_ROI_rectangle.append(rect)
3668
3669     if(ref_point_test):
3670         Globals.profiles_ROI_reference_point_check = True
3671     if(Globals.profiles_reference_point_check):
3672         Globals.profiles_done_button_reference_point.config(state=ACTIVE)
3673     else:
3674         Globals.profiles_ROI_check = True
3675     if(Globals.profiles_isocenter_check):
3676         Globals.profiles_done_button.config(state=ACTIVE)
3677
3678
3679     mark_ROI_window.after(500, lambda: mark_ROI_window.destroy())
3680
3681 mark_ROI_image_canvas.bind("<B1-Motion>", buttonMoving)
3682 mark_ROI_image_canvas.bind("<Button-1>", buttonPushed)
3683 mark_ROI_image_canvas.bind("<ButtonRelease-1>", buttonReleased)
3684
3685
3686 def UploadFilm():
3687     if(Globals.profiles_film_orientation.get() == '-'):
3688         messagebox.showerror("Missing parameter", "Film orientation
missing \n (Code: UploadFilm)")
3689         return
3690     if Globals.profiles_film_factor_input.get("1.0", 'end-1c') == " ":
3691         Globals.profiles_film_factor = 1
3692     else:
3693         try:
3694             Globals.profiles_film_factor = float(Globals.
3695 profiles_film_factor_input.get("1.0", 'end-1c'))
3696         except:
3697             messagebox.showerror("Missing parameter", "Film factor invalid
format. \n (Code: UploadFilm)")
3698             return
3699
3700     file = filedialog.askopenfilename()
3701     ext = os.path.splitext(file)[-1].lower()
3702     if(ext == '.tif'):
3703         current_folder = os.getcwd()
3704         parent = os.path.dirname(file)
3705         os.chdir(parent)

```

```

3706     img = Image.open(file)
3707     img = img.transpose(Image.FLIP_LEFT_RIGHT)
3708     cv2Img = cv2.imread(basename(normpath(file)), cv2.IMREAD_ANYCOLOR
| cv2.IMREAD_ANYDEPTH)
3709     cv2Img = cv2.medianBlur(cv2Img, 5)
3710     if(cv2Img is None):
3711         messagebox.showerror("Error", "Something has gone wrong. Check
that the filename does not contain , , ")
3712         return
3713     if(cv2Img.shape[2] == 3):
3714         if(cv2Img.shape[0]==1270 and cv2Img.shape[1]==1016):
3715             cv2Img = abs(cv2Img-Globals.correctionMatrix127)
3716             cv2Img = np.clip(cv2Img, 0, 65535)
3717             cv2Img = cv2.flip(cv2Img,1)
3718             img_scaled = img.resize((508, 635), Image.ANTIALIAS)
3719             img_scaled = ImageTk.PhotoImage(image=img_scaled)
3720
3721
3722             Globals.profiles_film_dataset = cv2Img
3723             Globals.profiles_film_dataset_red_channel = cv2Img[:, :, 2]
3724         else:
3725             messagebox.showerror("Error", "The resolution of the image
is not consistent with dpi")
3726             return
3727         else:
3728             messagebox.showerror("Error", "The uploaded image need to be in
RGB-format")
3729             return
3730
3731     os.chdir(current_folder)
3732
3733     if(not (img.width == 1016)):
3734         messagebox.showerror("Error", "Dpi in image has to be 127")
3735         return
3736
3737     Globals.profiles_film_orientation_menu.configure(state=DISABLED)
3738     Globals.profiles_film_factor_input.config(state=DISABLED)
3739
3740     h = 635 + 20
3741     w = 508 + 625
3742     new_window = tk.Toplevel(Globals.tab4)
3743     new_window.geometry("%dx%d+0+0" % (w, h))
3744     new_window.grab_set()
3745
3746     new_window_over_all_frame = tk.Frame(new_window, bd=0, relief=FLAT
)
3747     new_window_over_all_canvas = Canvas(new_window_over_all_frame)
3748
3749     new_window_xscrollbar = Scrollbar(new_window_over_all_frame,
orient=HORIZONTAL, command=new_window_over_all_canvas.xview)
3750     new_window_yscrollbar = Scrollbar(new_window_over_all_frame,
command=new_window_over_all_canvas.yview)
3751
3752     new_window_scroll_frame = ttk.Frame(new_window_over_all_canvas)
3753     new_window_scroll_frame.bind("<Configure>", lambda e:
new_window_over_all_canvas.configure(scrollregion=
new_window_over_all_canvas.bbox('all'))))

```

```

3754     new_window_over_all_canvas.create_window((0,0), window=
new_window_scroll_frame, anchor='nw')
3755     new_window_over_all_canvas.configure(xscrollcommand=
new_window_xscrollbar.set, yscrollcommand=new_window_yscrollbar.set)
3756
3757     new_window_over_all_frame.config(highlightthickness=0, bg='#ffffff')
3758     new_window_over_all_canvas.config(highlightthickness=0, bg='#
ffffff')
3759     new_window_over_all_frame.pack(expand=True, fill=BOTH)
3760     new_window_over_all_canvas.grid(row=0, column=0, sticky=N+S+E+W)
3761     new_window_over_all_frame.grid_columnconfigure(0, weight=1)
3762     new_window_over_all_frame.grid_rowconfigure(0, weight=1)
3763     new_window_xscrollbar.grid(row=1, column=0, sticky=E+W)
3764     new_window_over_all_frame.grid_columnconfigure(1, weight=0)
3765     new_window_over_all_frame.grid_rowconfigure(1, weight=0)
3766     new_window_yscrollbar.grid(row=0, column=1, sticky=N+S)
3767     new_window_over_all_frame.grid_columnconfigure(2, weight=0)
3768     new_window_over_all_frame.grid_rowconfigure(2, weight=0)
3769
3770     new_window_explain_text = tk.Text(new_window_scroll_frame, height=
3, width=120)
3771     new_window_explain_text.insert(INSERT, \
3772 "To match the film with the doseplan you have to mark either isocenter or
3773 a reference point\
3774 on the film of your choice. In the case of the reference point you \nwill
3775 be asked to input the \
3776 lenght in lateral, longitudinal and vertical to a reference point used in
3777 the linac. If the \
3778 reference point in the film is the same as \nthe one in the phantom/linac
3779 you can input all zeros,\
3780 in other cases your input is in mm. Later you will have the opportunity to
3781 make small\
3782 adjustments \nto the placement of either the reference point or isocenter.
3783 ")
3784     new_window_explain_text.config(state=DISABLED, font=(`calibri`, '
3785 13', 'bold'), bg = '#ffffff', relief=FLAT)
3786     new_window_explain_text.grid(row=0, column=0, columnspan=5, sticky=
N+S+W+E, pady=(15,5), padx=(10,10))
3787     new_window_scroll_frame.grid_rowconfigure(0, weight=0)
3788     new_window_scroll_frame.grid_columnconfigure(0, weight=0)
3789
3790     new_window_notebook = ttk.Notebook(new_window_scroll_frame)
3791     new_window_notebook.borderWidth=0
3792     new_window_notebook.grid(row=2, column=0, columnspan=5, sticky=E+W
+N+S, pady=(0,0), padx =(0,0))
3793     new_window_scroll_frame.grid_rowconfigure(4, weight=0)
3794     new_window_scroll_frame.grid_columnconfigure(4, weight=0)
3795
3796     new_window_isocenter_tab = ttk.Frame(new_window_notebook)
3797     new_window_notebook.add(new_window_isocenter_tab, text='Isocenter'
3798 )
3799     new_window_reference_point_tab = ttk.Frame(new_window_notebook)
3800     new_window_notebook.add(new_window_reference_point_tab, text='
Reference point')
3801     new_window_manually_tab = ttk.Frame(new_window_notebook)

```

```

    new_window_notebook.add(new_window_manually_tab, text='Manually')

3796
3798     image_canvas = tk.Canvas(new_window_isocenter_tab)
3799     image_canvas.grid(row=0, column=0, rowspan=12, columnspan=3, sticky
3800     =N+S+E+W, padx=(0,0), pady=(0,0))
3801     new_window_isocenter_tab.grid_rowconfigure(1, weight=0)
3802     new_window_isocenter_tab.grid_columnconfigure(1, weight=0)
3803     image_canvas.create_image(0,0,image=img_scaled, anchor="nw")
3804     image_canvas.image = img_scaled
3805     image_canvas.config(bg='#ffffff', relief=FLAT, bd=0, scrollregion=
3806     image_canvas.bbox(ALL), \
3807         height=img_scaled.height(), width=img_scaled.width())
3808     image_canvas.grid_propagate(0)

3809     image_canvas_reference_tab = tk.Canvas(
3810     new_window_reference_point_tab)
3811     image_canvas_reference_tab.grid(row=0, column=0, rowspan=10,
3812     columnspan=3, sticky=N+S+E+W, padx=(0,0), pady=(0,0))
3813     new_window_reference_point_tab.grid_rowconfigure(1, weight=0)
3814     new_window_reference_point_tab.grid_columnconfigure(1, weight=0)
3815     image_canvas_reference_tab.create_image(0,0,image=img_scaled,
3816     anchor="nw")
3817     image_canvas_reference_tab.image = img_scaled
3818     image_canvas_reference_tab.config(bg='#ffffff', relief=FLAT, bd=0,
3819     scrollregion=image_canvas.bbox(ALL), \
3820         height=img_scaled.height(), width=img_scaled.width())
3821     image_canvas_reference_tab.grid_propagate(0)

3822     film_window_mark_isocenter_text = tk.Text(new_window_isocenter_tab
3823     , width=55, height=7)
3824     film_window_mark_isocenter_text.insert(INSERT, \
3825 "When clicking the button \"Mark isocenter\" a window showing \n\
3826 the image will appear and you are to click on the markers \n\
3827 made on the film upon irradiation to find the isocenter. Start \n\
3828 with the marker showing the direction of the film (see the \n\
3829 specifications in main window). When both marks are made \n\
3830 you will see the isocenter in the image. If you are not happy \n\
3831 with the placement click the button again and repeat.")
3832     film_window_mark_isocenter_text.config(bg='#ffffff', relief=FLAT,
3833     bd=0, state=DISABLED, font=('calibri', '11'))
3834     film_window_mark_isocenter_text.grid(row=0, column=3, rowspan=3,
3835     sticky=N+S+E+W, padx=(10,10), pady=(10,0))
3836     new_window_isocenter_tab.columnconfigure(2, weight=0)
3837     new_window_isocenter_tab.rowconfigure(2, weight=0)

3838     film_window_mark_reference_point_text = tk.Text(
3839     new_window_reference_point_tab, width=55, height=5)
3840     film_window_mark_reference_point_text.insert(INSERT, \
3841 "When clicking the button \"Mark point\" a window showing \n\
3842 the image will appear and you are to click on the marker \n\
3843 made on the film upon irradiation to find the point. When\n\
3844 the mark are made you will see the isocenter in the image.\n\
3845 If you are not happy with the placement click the button \n\
3846 again and repeat.")
3847     film_window_mark_reference_point_text.config(bg='#ffffff', relief=
3848     FLAT, bd=0, state=DISABLED, font=('calibri', '11'))

```

```

    film_window_mark_reference_point_text.grid(row=0, column=3,
3842      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)

3844
3846      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)

3850
3852      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

3856
3858      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)

3862
3864      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

3870
3872      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, "\n
If the marked reference points in the film does not match\n\
the reference point in the phantom you can write the\n\
displacement here (in mm). Defaults to zero ")
3874      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)

3878
3880      input_lateral_text = tk.Text(new_window_reference_point.tab, width
=12, height=1)
        input_lateral_text.insert(INSERT, "Lateral:")

```

```

3882     input_lateral_text.config(bg='ffffff', relief=FLAT, bd=0, state=
3883 DISABLED, font=('calibri', '10'))
3884     input_lateral_text.grid(row=5, column=3, sticky=N+S, padx=(0,250),
3885 pady=(25,0))
3886     new_window_reference_point_tab.grid_rowconfigure(10, weight=0)
3887     new_window_reference_point_tab.grid_rowconfigure(10, weight=0)
3888
3889     Globals.profiles_input_lateral_displacement = tk.Text(
3890 new_window_reference_point_tab, width=5, height=1)
3891     Globals.profiles_input_lateral_displacement.insert(INSERT, " ")
3892     Globals.profiles_input_lateral_displacement.config(bg='#E5f9ff',
3893 relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
3894     Globals.profiles_input_lateral_displacement.grid(row=5, column=3,
3895 padx=(0,285), pady=(35,0))
3896     new_window_reference_point_tab.grid_rowconfigure(7, weight=0)
3897     new_window_reference_point_tab.grid_columnconfigure(7, weight=0)
3898
3899     input_vertical_text = tk.Text(new_window_reference_point_tab,
3900 width=12, height=1)
3901     input_vertical_text.insert(INSERT, "Vertical:")
3902     input_vertical_text.config(bg='ffffff', relief=FLAT, bd=0, state=
3903 DISABLED, font=('calibri', '10'))
3904     input_vertical_text.grid(row=5, column=3, sticky=N+S, padx=(0,0),
3905 pady=(25,0))
3906     new_window_reference_point_tab.grid_rowconfigure(11, weight=0)
3907     new_window_reference_point_tab.grid_rowconfigure(11, weight=0)
3908
3909     Globals.profiles_input_vertical_displacement = tk.Text(
3910 new_window_reference_point_tab, width=4, height=1)
3911     Globals.profiles_input_vertical_displacement.insert(INSERT, " ")
3912     Globals.profiles_input_vertical_displacement.config(bg='#E5f9ff',
3913 relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
3914     Globals.profiles_input_vertical_displacement.grid(row=5, column=3,
3915 padx=(0,25), pady=(35,0))
3916     new_window_reference_point_tab.grid_rowconfigure(8, weight=0)
3917     new_window_reference_point_tab.grid_columnconfigure(8, weight=0)
3918
3919     input_long_text = tk.Text(new_window_reference_point_tab, width
3920 =12, height=1)
3921     input_long_text.insert(INSERT, "Longitudinal:")
3922     input_long_text.config(bg='ffffff', relief=FLAT, bd=0, state=
3923 DISABLED, font=('calibri', '10'))
3924     input_long_text.grid(row=5, column=3, sticky=N+S, padx=(250,0),
3925 pady=(25,0))
3926     new_window_reference_point_tab.grid_rowconfigure(12, weight=0)
3927     new_window_reference_point_tab.grid_rowconfigure(12, weight=0)
3928
3929     Globals.profiles_input_longitudinal_displacement = tk.Text(
3930 new_window_reference_point_tab, width=5, height=1)
3931     Globals.profiles_input_longitudinal_displacement.insert(INSERT, " ")
3932     Globals.profiles_input_longitudinal_displacement.config(bg='#
3933 E5f9ff', relief=GROOVE, bd=2, state=NORMAL, font=('calibri', '11'))
3934     Globals.profiles_input_longitudinal_displacement.grid(row=5,
3935 column=3, padx=(240,0), pady=(35,0))
3936     new_window_reference_point_tab.grid_rowconfigure(9, weight=0)
3937     new_window_reference_point_tab.grid_columnconfigure(9, weight=0)

```

```

3922     film_window_mark_ROI_text = tk.Text(new_window_isocenter_tab ,
3923     width=55, height=7)
3924     film_window_mark_ROI_text.insert(INSERT, \
3925 "When clicking the button \"Mark ROI\" a window showing the\n\
3926 image will appear and you are to drag a rectangle marking \n\
3927 the region of interest. Fidora will assume the film has been\n\
3928 scanned in either portrait or landscape orientation. When\n\
3929 the ROI has been marked it will appear on the image. If you\n\
3930 are not happy with the placement click the button again.")
3931     film_window_mark_ROI_text.config(bg='#ffffff', relief=FLAT, bd=0,
3932     state=DISABLED, font=('calibri', '11'))
3933     film_window_mark_ROI_text.grid(row=5, column=3, rowspan=4, sticky=
3934 N+S+E+W, padx=(10,10), pady=(0,0))
3935     new_window_isocenter_tab.grid_columnconfigure(4, weight=0)
3936     new_window_isocenter_tab.grid_rowconfigure(4, weight=0)
3937
3938     film_window_mark_ROI_reference_point_text = tk.Text(
3939     new_window_reference_point_tab, width=55, height=5)
3940     film_window_mark_ROI_reference_point_text.insert(INSERT, \
3941 "When clicking the button \"Mark ROI\" a window showing the\n\
3942 image will appear and you are to drag a rectangle marking \n\
3943 the region of interest. Fidora will assume the film has been\n\
3944 scanned in either portrait or landscape orientation. When\n\
3945 the ROI has been marked it will appear on the image. If you\n\
3946 are not happy with the placement click the button again.")
3947     film_window_mark_ROI_reference_point_text.config(bg='#ffffff',
3948     relief=FLAT, bd=0, state=DISABLED, font=('calibri', '11'))
3949     film_window_mark_ROI_reference_point_text.grid(row=6, column=3,
3950     rowspan=3, sticky=N+E+W, padx=(10,10), pady=(10,0))
3951     new_window_reference_point_tab.grid_columnconfigure(4, weight=0)
3952     new_window_reference_point_tab.grid_rowconfigure(4, weight=0)
3953
3954     mark_ROI_button_frame = tk.Frame(new_window_isocenter_tab)
3955     mark_ROI_button_frame.grid(row=8, column=3, padx=(10,0), pady
3956 =(0,5))
3957     mark_ROI_button_frame.configure(bg='#ffffff')
3958     new_window_isocenter_tab.grid_columnconfigure(5, weight=0)
3959     new_window_isocenter_tab.grid_rowconfigure(5, weight=0)
3960
3961     mark_ROI_button = tk.Button(mark_ROI_button_frame, text='Browse',
3962     image=Globals.profiles_mark_ROI_button_image, \
3963     cursor='hand2', font=('calibri', '14'), relief=FLAT, state=
3964 ACTIVE, command=lambda: markROI(img, new_window_isocenter_tab,
3965     image_canvas, False))
3966     mark_ROI_button.pack(expand=True, fill=BOTH)
3967     mark_ROI_button.config(bg='#ffffff', activebackground='#ffffff',
3968     activeforeground='#ffffff', highlightthickness=0)
3969     mark_ROI_button.image=Globals.profiles_mark_ROI_button_image
3970
3971     slice_offset_text = tk.Text(new_window_isocenter_tab, width=25,
3972     height=1)
3973     slice_offset_text.insert(INSERT, "Slice offset, mm (default 0):")
3974     slice_offset_text.config(state=DISABLED, font=('calibri', '10'),
3975     bd = 0, relief=FLAT)
3976     slice_offset_text.grid(row=9, column=3, padx=(5,110), pady=(0,0))
3977     new_window_isocenter_tab.grid_columnconfigure(6, weight=0)

```

```

        new_window_isocenter_tab.grid_rowconfigure(6, weight=0)

3966     Globals.profiles_slice_offset = tk.Text(new_window_isocenter_tab,
3967         width=8, height=1)
3968     Globals.profiles_slice_offset.grid(row=9, column=3, padx=(110,10),
3969         pady=(0,0))
3970     Globals.profiles_slice_offset.insert(INSERT, " ")
3971     Globals.profiles_slice_offset.config(state=NORMAL, font=('calibri',
3972         '10'), bd = 2, bg='#ffffff')
3973     new_window_isocenter_tab.grid_columnconfigure(7, weight=0)
3974     new_window_isocenter_tab.grid_rowconfigure(7, weight=0)

3975     mark_ROI_button_reference_point_frame = tk.Frame(
3976         new_window_reference_point_tab)
3977     mark_ROI_button_reference_point_frame.grid(row=9, column=3, padx=
3978         (10,10), pady=(0,5))
3979     mark_ROI_button_reference_point_frame.configure(bg='#ffffff')
3980     new_window_reference_point_tab.grid_columnconfigure(5, weight=0)
3981     new_window_reference_point_tab.grid_rowconfigure(5, weight=0)

3982     mark_ROI_reference_point_button = tk.Button(
3983         mark_ROI_button_reference_point_frame, text='Browse', image=Globals.
3984         profiles_mark_ROI_button_image,\n            cursor='hand2', font=('calibri', '14'), relief=FLAT, state=
3985             ACTIVE, command=lambda: markROI(img, new_window_reference_point_tab,
3986                 image_canvas_reference_tab, True))
3987     mark_ROI_reference_point_button.pack(expand=True, fill=BOTH)
3988     mark_ROI_reference_point_button.config(bg='#ffffff',
3989         activebackground='#ffffff', activeforeground='#ffff00',
3990         highlightthickness=0)
3991     mark_ROI_reference_point_button.image=Globals.
3992     profiles_mark_ROI_button_image

3993     def finishFilmMarkers(ref_test):
3994         Globals.profiles_slice_offset.config(state=DISABLED)
3995         if(ref_test):
3996             if(not(Globals.profiles_input_lateral_displacement.get("1.0",
3997                 'end-1c')==" ")):
3998                 try:
3999                     test = float(Globals.
4000                     profiles_input_lateral_displacement.get("1.0",
4001                         'end-1c'))
4002                     Globals.profiles_lateral = test
4003                 except:
4004                     messagebox.showerror("Error", "The displacements
4005             must be numbers\n (Code: lateral displacement)")
4006             return
4007         else:
4008             Globals.profiles_lateral = 0
4009             if(not(Globals.profiles_input_longitudinal_displacement.
4010                 get("1.0", 'end-1c')==" ")):
4011                 try:
4012                     test = float(Globals.
4013                     profiles_input_longitudinal_displacement.get("1.0",
4014                         'end-1c'))
4015                     Globals.profiles_longitudinal = test
4016                 except:
4017                     messagebox.showerror("Error", "The displacements
4018             must be numbers\n (Code: longitudinal displacement)")

```

```

4004                     return
4006             else:
4007                 Globals.profiles_longitudinal = 0
4008                 if(not(Globals.profiles_input_vertical_displacement.get("1.0",'end-1c')=="")):
4009                     try:
4010                         test = float(Globals.
4011                         profiles_input_vertical_displacement.get("1.0",'end-1c'))
4012                         Globals.profiles_vertical = test
4013                     except:
4014                         messagebox.showerror("Error", "The displacements
4015 must be numbers\n (Code: vertical displacement)")
4016                         return
4017                     else:
4018                         Globals.profiles_vertical = 0
4019                         Globals.profiles_input_vertical_displacement.config(state=
4020 DISABLED)
4021                         Globals.profiles_input_longitudinal_displacement.config(
4022 state=DISABLED)
4023                         Globals.profiles_input_lateral_displacement.config(state=
4024 DISABLED)
4025                     else:
4026                         if not Globals.profiles_slice_offset.get("1.0",'end-1c')==
4027 " ":
4028                             try:
4029                                 offset = float(Globals.profiles_slice_offset.get("1.0",'end-1c'))
4030                                 Globals.profiles_offset = offset
4031                             except:
4032                                 messagebox.showerror("Error", "Slice offset must
4033 be a number \n(Code: finishFilmMarkers(false))")
4034                                 return
4035                             else:
4036                                 Globals.profiles_offset = 0
4037                                 if(ref_test):
4038                                     choose_batch_window = tk.Toplevel(
4039                                         new_window_reference_point_tab)
4040                                 else:
4041                                     choose_batch_window = tk.Toplevel(new_window_isocenter_tab
4042 )
4043
4044         choose_batch_window.geometry("670x380+50+50")
4045         choose_batch_window.grab_set()
4046
4047         choose_batch_frame = tk.Frame(choose_batch_window)
4048         choose_batch_frame.pack(expand=True, fill=BOTH)
4049         choose_batch_frame.configure(bg="#ffffff")
4050
4051         batch_cnt = 0
4052         weight_cnt = 0
4053         read = open('calibration.txt', 'r')
4054         lines = read.readlines()
4055         read.close()
4056         row_cnt=0
4057         for l in lines:
4058             words = l.split()

```

```

4050           line = "Batch nr. : " + words[2] + ".      Date:    " +
words[0] + " " + words[1] + "."
4051           write_batch_nr = tk.Text(choose_batch_frame, width=10,
height=1)
4052           write_batch_nr.grid(row=row_cnt, column=0, sticky=N+S+W+E,
padx=(10,5), pady=(10,10))
4053           choose_batch_frame.grid_columnconfigure(weight_cnt, weight
=0)
4054           choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
4055           write_batch_nr.insert(INSERT, "Batch nr.: ")
4056           write_batch_nr.config(state=DISABLED, bd = 0, font=(`calibri',
4057           '12', 'bold')))
4058           weight_cnt+=1
4059           write_batch = tk.Text(choose_batch_frame, width=20, height
=1)
4060           write_batch.grid(row=row_cnt, column=1, sticky=N+S+W+E,
padx=(10,5), pady=(10,10))
4061           choose_batch_frame.grid_columnconfigure(weight_cnt, weight
=0)
4062           choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
4063           write_batch.insert(INSERT, words[2])
4064           write_batch.config(state=DISABLED, bd = 0, font=(`calibri'
4065           , '12'))
4066           weight_cnt+=1
4067           write_batch_date = tk.Text(choose_batch_frame, width=8,
height=1)
4068           write_batch_date.grid(row=row_cnt, column=2, sticky=N+S+W+E,
padx=(10,5), pady=(10,10))
4069           choose_batch_frame.grid_columnconfigure(weight_cnt, weight
=0)
4070           choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
4071           write_batch_date.insert(INSERT, "Date: ")
4072           write_batch_date.config(state=DISABLED, bd = 0, font=(`calibri',
4073           '12', 'bold')))
4074           weight_cnt+=1
4075           write_date = tk.Text(choose_batch_frame, width=30, height
=1)
4076           write_date.grid(row=row_cnt, column=3, sticky=N+S+W+E,
padx=(10,5), pady=(10,10))
4077           choose_batch_frame.grid_columnconfigure(weight_cnt, weight
=0)
4078           choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
4079           Radiobutton(choose_batch_frame, text=' ',bg='#ffffff',
4080           cursor='hand2',font=(`calibri', '14'), \
4081           variable=Globals.profiles.film_batch, value=batch_cnt)
4082           .grid(row=row_cnt, \
4083           column=4, sticky=N+S+W+E, padx=(5,5), pady=(10,10))
4084           choose_batch_frame.grid_columnconfigure(weight_cnt, weight
=0)
4085           choose_batch_frame.grid_rowconfigure(weight_cnt, weight=0)
4086           weight_cnt+=1;row_cnt+=1;batch_cnt+=1

```

```

4086     def set_batch():
4087         choose_batch_window.destroy()
4088         f = open('calibration.txt', 'r')
4089         lines = f.readlines()
4090         words = lines[Globals.profiles_film_batch.get()].split()
4091         Globals.profiles_popt_red[0] = float(words[3])
4092         Globals.profiles_popt_red[1] = float(words[4])
4093         Globals.profiles_popt_red[2] = float(words[5])
4094         f.close()
4095
4096         Globals.profiles_film_dataset_ROI_red_channel_dose = np.
4097         zeros((Globals.profiles_film_dataset_ROI_red_channel.shape[0], \
4098               Globals.profiles_film_dataset_ROI_red_channel.shape
4099               [1]))
4100             for i in range(Globals.
4101                 profiles_film_dataset_ROI_red_channel_dose.shape[0]):
4102                   for j in range(Globals.
4103                     profiles_film_dataset_ROI_red_channel_dose.shape[1]):
4104                         Globals.profiles_film_dataset_ROI_red_channel_dose
4105                         [i,j] = Globals.profiles_film_factor*\\
4106                           pixel_to_dose(Globals.
4107                           profiles_film_dataset_ROI_red_channel[i,j], \
4108                           Globals.profiles_popt_red[0], Globals.
4109                           profiles_popt_red[1], Globals.profiles_popt_red[2])
4110
4111         Globals.profiles_film_dataset_red_channel_dose = np.zeros
4112         ((Globals.profiles_film_dataset_red_channel.shape[0], \
4113           Globals.profiles_film_dataset_red_channel.shape[1]))
4114             for i in range(Globals.
4115               profiles_film_dataset_red_channel_dose.shape[0]):
4116                   for j in range(Globals.
4117                     profiles_film_dataset_red_channel_dose.shape[1]):
4118                         Globals.profiles_film_dataset_red_channel_dose[i,j]
4119                         = Globals.profiles_film_factor*\\
4120                           pixel_to_dose(Globals.
4121                           profiles_film_dataset_red_channel[i,j], \
4122                           Globals.profiles_popt_red[0], Globals.
4123                           profiles_popt_red[1], Globals.profiles_popt_red[2])
4124
4125             Globals.film_write_image.create_image(0,0,image=
4126               scaled_image_visual, anchor="nw")
4127             Globals.film_write_image.image = scaled_image_visual
4128
4129             mx_film=np.max(Globals.
4130               profiles_film_dataset_ROI_red_channel_dose)
4131               Globals.profiles_max_dose_film = mx_film
4132               img_film = Globals.
4133               profiles_film_dataset_ROI_red_channel_dose
4134                 img_film = img_film/mx_film
4135                 PIL_img_film = Image.fromarray(np.uint8(cm.viridis(
4136                   img_film)*255))
4137
4138             scaled_image_visual_film = ImageTk.PhotoImage(image=
4139               PIL_img_film)
4140               Globals.film_dose_write_image.create_image(0,0,image=
4141                 scaled_image_visual_film, anchor="nw")

```

```

4124         Globals.film_dose_write_image.image =
4125             scaled_image_visual_film
4126
4127             film_scanned_image_text_canvas.create_image(0,0,image=
4128                 Globals.profiles_scanned_image_text_image , anchor="nw")
4129                 film_scanned_image_text_canvas.image = Globals.
4130                 profiles_scanned_image_text_image
4131                     film_dose_map_image_text_canvas.create_image(0,0, image=
4132                         Globals.profiles_film_dose_map_text_image , anchor="nw")
4133                             film_dose_map_image_text_canvas.image=Globals.
4134                             profiles_film_dose_map_text_image
4135
4136             new_window.destroy()
4137
4138             set_batch_button_frame = tk.Frame(choose_batch_frame)
4139                 set_batch_button_frame.grid(row=row_cnt , column=1, columnspan
4140 =3, padx=(10,0) , pady=(5,5))
4141                 set_batch_button_frame.configure(bg='#ffffff')
4142                 choose_batch_frame.grid_columnconfigure(weight_cnt , weight=0)
4143                 choose_batch_frame.grid_rowconfigure(weight_cnt , weight=0)
4144
4145             set_batch_button = tk.Button(set_batch_button_frame , text='OK'
4146 , image=Globals.done_button_image , cursor='hand2' ,\
4147                 font=('calibri' , '14') , relief=FLAT, state=ACTIVE, command
4148 =set_batch)
4149                 set_batch_button.pack(expand=True , fill=BOTH)
4150                 set_batch_button.image=Globals.done_button_image
4151
4152
4153             img_ROI = Globals.profiles_film_dataset[Globals.
4154                 profiles_ROI_coords[0][1]:Globals.profiles_ROI_coords[2][1],\
4155                     Globals.profiles_ROI_coords[0][0]:Globals.
4156                     profiles_ROI_coords[1][0] , :]
4157                 img_ROI_red_channel = img_ROI[:, :, 2]
4158                 Globals.profiles_film_variable_ROI_coords = [Globals.
4159                     profiles_ROI_coords[0][1] , Globals.profiles_ROI_coords[2][1], \
4160                         Globals.profiles_ROI_coords[0][0] , Globals.
4161                         profiles_ROI_coords[1][0]]
4162                 Globals.profiles_film_dataset_ROI = img_ROI
4163                 Globals.profiles_film_dataset_ROI_red_channel =
4164                     img_ROI_red_channel
4165                     R = img_ROI[:, :, 2];B = img_ROI[:, :, 0]; G = img_ROI[:, :, 1]
4166                     img_ROI_RGB = np.zeros(img_ROI.shape)
4167                     img_ROI_RGB[:, :, 0]=R; img_ROI_RGB[:, :, 1]=G; img_ROI_RGB
4168                         [:,:,2]=B
4169                         PIL_img_ROI = (img_ROI_RGB/256).astype('uint8')
4170                         PIL_img_ROI = Image.fromarray(PIL_img_ROI , 'RGB')
4171                         #PIL_img_ROI = Image.fromarray((img_ROI_RGB * 255).astype(np.
4172                             uint8) , 'RGB')
4173                         wid = PIL_img_ROI.width;heig = PIL_img_ROI.height
4174                         #film_window_write_image = tk.Canvas(film_window_scroll_frame)
4175
4176             film_image_canvas = tk.Canvas(Globals.
4177                 profiles_film_panedwindow)
4178                 film_image_canvas.grid(row=0,column=0, sticky=N+S+W+E)
4179                 Globals.profiles_film_panedwindow.add(film_image_canvas , \

```

```

4164                               height=max(heig , Globals . profiles_scanned_image_text_image .
height()) , \
4165                               width=wid + Globals . profiles_scanned_image_text_image .
width())
4166                               film_image_canvas.config(bg='#ffffff' , relief=FLAT,
highlightthickness=0 , \
4167                               height=max(heig , Globals . profiles_scanned_image_text_image .
height()) , \
4168                               width=wid + Globals . profiles_scanned_image_text_image .
width())
4169
4170             film_dose_canvas = tk.Canvas(Globals . profiles_film_panedwindow
)
4171             film_dose_canvas.grid(row=1,column=0, sticky=N+S+W+E)
4172             Globals . profiles_film_panedwindow.add(film_dose_canvas , \
4173                               height=max(heig , Globals . profiles_film_dose_map_text_image .
height()) , \
4174                               width=wid + Globals . profiles_film_dose_map_text_image .
width())
4175             film_dose_canvas.config(bg='#ffffff' , relief=FLAT,
highlightthickness=0 , \
4176                               height=max(heig , Globals . profiles_film_dose_map_text_image .
height()) , \
4177                               width=wid + Globals . profiles_film_dose_map_text_image .
width())
4178
4179             Globals . film_write_image = tk.Canvas(film_image_canvas)
4180             Globals . film_write_image.grid(row=0,column=1,sticky=N+S+W+E)
4181             Globals . film_write_image.config(bg='#ffffff' , relief=FLAT,
highlightthickness=0, width=wid , height=heig)
4182
4183             Globals . film_dose_write_image = tk.Canvas(film_dose_canvas)
4184             Globals . film_dose_write_image.grid(row=0,column=1,sticky=N+S+W+E)
4185             Globals . film_dose_write_image.config(bg='#ffffff' , relief=FLAT
, highlightthickness=0, width=wid , height=heig)
4186
4187             film_scanned_image_text_canvas=tk.Canvas(film_image_canvas)
4188             film_scanned_image_text_canvas.grid(row=0,column=0,sticky=N+S+W+E)
4189             film_scanned_image_text_canvas.config(bg='#ffffff' , relief=
FLAT, highlightthickness=0 , \
4190                               height=Globals . profiles_scanned_image_text_image .height() ,
width=Globals . profiles_scanned_image_text_image .width())
4191
4192             film_dose_map_image_text_canvas=tk.Canvas(film_dose_canvas)
4193             film_dose_map_image_text_canvas.grid(row=0,column=0,sticky=N+S+W+E)
4194             film_dose_map_image_text_canvas.config(bg='#ffffff' , relief=
FLAT, highlightthickness=0 , \
4195                               height=Globals . profiles_film_dose_map_text_image .height() ,
width=Globals . profiles_film_dose_map_text_image .width())
4196
4197             scaled_image_visual = PILimg_ROI
4198             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

4202    Globals.profiles_upload_button_doseplan.config(state=DISABLED)
4204    Globals.profiles_upload_button_rtplan.config(state=ACTIVE)
4205    Globals.profiles_upload_button_film.config(state=DISABLED)

4206    #Beregne avstand mellom ROI og isocenter gitt i mm
4207    # [top left[mot venstre , oppover], top right[mot venstre (h yre blir negativ), oppover], bottom left , bottom right]
4208    if(ref_test):
4209        Globals.profiles_distance_reference_point_ROI.append([((
4210            Globals.profiles_film_reference_point[0]-Globals.profiles_ROI_coords
4211            [0][0])*0.2,\n
4212                (Globals.profiles_film_reference_point[1]-Globals.
4213                    profiles_ROI_coords[0][1])*0.2])
4213        Globals.profiles_distance_reference_point_ROI.append([((
4214            Globals.profiles_film_reference_point[0]-Globals.profiles_ROI_coords
4215            [1][0])*0.2,\n
4216                (Globals.profiles_film_reference_point[1]-Globals.
4217                    profiles_ROI_coords[1][1])*0.2])
4217        Globals.profiles_distance_reference_point_ROI.append([((
4218            Globals.profiles_film_reference_point[0]-Globals.profiles_ROI_coords
4219            [2][0])*0.2,\n
4220                (Globals.profiles_film_reference_point[1]-Globals.
4221                    profiles_ROI_coords[2][1])*0.2])
4221        Globals.profiles_distance_reference_point_ROI.append([((
4222            Globals.profiles_film_isocenter[0]-Globals.profiles_ROI_coords[0][0])*0.2,\n
4223                (Globals.profiles_film_isocenter[1]-Globals.
4224                    profiles_ROI_coords[0][1])*0.2])
4225        Globals.profiles_distance_isocenter_ROI.append([((
4226            Globals.profiles_film_isocenter[0]-Globals.profiles_ROI_coords[1][0])*0.2,\n
4227                (Globals.profiles_film_isocenter[1]-Globals.
4228                    profiles_ROI_coords[1][1])*0.2])
4229        Globals.profiles_distance_isocenter_ROI.append([((
4230            Globals.profiles_film_isocenter[0]-Globals.profiles_ROI_coords[2][0])*0.2,\n
4231                (Globals.profiles_film_isocenter[1]-Globals.
4232                    profiles_ROI_coords[2][1])*0.2])
4233        Globals.profiles_distance_isocenter_ROI.append([((
4234            Globals.profiles_film_isocenter[0]-Globals.profiles_ROI_coords[3][0])*0.2,\n
4235                (Globals.profiles_film_isocenter[1]-Globals.
4236                    profiles_ROI_coords[3][1])*0.2)])
4237
4238        Globals.profiles_isocenter_or_reference_point = "Ref_point"
4239
4240    else:
4241        Globals.profiles_distance_isocenter_ROI.append([((
4242            Globals.profiles_film_isocenter[0]-Globals.profiles_ROI_coords[0][0])*0.2,\n
4243                (Globals.profiles_film_isocenter[1]-Globals.
4244                    profiles_ROI_coords[0][1])*0.2])
4245        Globals.profiles_distance_isocenter_ROI.append([((
4246            Globals.profiles_film_isocenter[0]-Globals.profiles_ROI_coords[1][0])*0.2,\n
4247                (Globals.profiles_film_isocenter[1]-Globals.
4248                    profiles_ROI_coords[1][1])*0.2])
4249        Globals.profiles_distance_isocenter_ROI.append([((
4250            Globals.profiles_film_isocenter[0]-Globals.profiles_ROI_coords[2][0])*0.2,\n
4251                (Globals.profiles_film_isocenter[1]-Globals.
4252                    profiles_ROI_coords[2][1])*0.2])
4253        Globals.profiles_distance_isocenter_ROI.append([((
4254            Globals.profiles_film_isocenter[0]-Globals.profiles_ROI_coords[3][0])*0.2,\n
4255                (Globals.profiles_film_isocenter[1]-Globals.
4256                    profiles_ROI_coords[3][1])*0.2)])
4257
4258        Globals.profiles_isocenter_or_reference_point = "Isocenter"
4259

```

```

4232     done_button_frame = tk.Frame(new_window_isocenter_tab)
4233     done_button_frame.grid(row=10, column=3, padx=(10,10), pady=(5,5),
4234                             sticky=N+S+W+E)
4235     done_button_frame.configure(bg='#ffffff')
4236     new_window_isocenter_tab.grid_columnconfigure(5, weight=0)
4237     new_window_isocenter_tab.grid_rowconfigure(5, weight=0)

4238     Globals.profiles_done_button = tk.Button(done_button_frame, text='
4239         Done', image=Globals.done_button_image,\n            cursor='hand2', font=('calibri', '14'), relief=FLAT, state=
4240             DISABLED, command=lambda: finishFilmMarkers(False))
4241     Globals.profiles_done_button.pack(expand=True, fill=BOTH)
4242     Globals.profiles_done_button.config(bg='#ffffff', activebackground
4243 = '#ffffff', activeforeground='#ffffff', highlightthickness=0)
4244     Globals.profiles_done_button.image=Globals.done_button_image

4245     done_button_reference_point_frame = tk.Frame(
4246         new_window_reference_point_tab)
4247     done_button_reference_point_frame.grid(row=10, column=3, padx
4248 =(10,10), pady=(5,5), sticky=N+S+W+E)
4249     done_button_reference_point_frame.configure(bg='#ffffff')
4250     new_window_reference_point_tab.grid_columnconfigure(5, weight=0)
4251     new_window_reference_point_tab.grid_rowconfigure(5, weight=0)

4252     Globals.profiles_done_button_reference_point= tk.Button(
4253         done_button_reference_point_frame, text='Done', image=Globals.
4254         done_button_image,\n            cursor='hand2', font=('calibri', '14'), relief=FLAT, state=
4255             DISABLED, command=lambda: finishFilmMarkers(True))
4256     Globals.profiles_done_button_reference_point.pack(expand=True,
4257             fill=BOTH)
4258     Globals.profiles_done_button_reference_point.config(bg='#ffffff',
4259             activebackground='#ffffff', activeforeground='#ffffff',
4260             highlightthickness=0)
4261     Globals.profiles_done_button_reference_point.image=Globals.
4262         done_button_image

4263
4264     elif(ext== ""):
4265         return
4266     else:
4267         messagebox.showerror("Error", "The file must be a *.tif file")

4268 def plot_profiles():
4269
4270     return

4271
4272 def help_showPlanes():
4273     new_window = tk.Toplevel(Globals.tab4)
4274     w = Globals.profiles_showPlanes_image.width()
4275     h = Globals.profiles_showPlanes_image.height()
4276     new_window.geometry("%dx%d+0+0" % (w, h))
4277     new_window.grab_set()

4278     canvas = tk.Canvas(new_window)
4279     canvas.config(relief=FLAT, bg='#ffffff', highlightthickness=0)

```

```
4276     canvas.create_image(0, 0, image=Globals.profiles_showPlanes_image,
4277                         anchor='nw')
4278     canvas.pack(expand=True, fill=BOTH)
4279
4280
4281     def help_showDepth():
4282         new_window = tk.Toplevel(Globals.tab4)
4283         w = Globals.profiles_showDirections_image.width()
4284         h = Globals.profiles_showDirections_image.height()
4285         new_window.geometry("%dx%d+0+0" % (w, h))
4286         new_window.grab_set()
4287
4288         canvas = tk.Canvas(new_window)
4289         canvas.config(relief=FLAT, bg='#ffffff', highlightthickness=0)
4290         canvas.create_image(0,0, image=Globals.profiles_showDirections_image,
4291                             anchor='nw')
4292         canvas.pack(expand=True, fill=BOTH)
```

FIDORA/Profile_functions.py