TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

D I P L O M A R B E I T

Deformierbare Bildregistrierung mittels Featurelet Algorithmen - Implementierung in die 3D Slicer Software und Validierung

Ausgeführt am
Atominstitut
der Technischen Universität Wien

in Zusammenarbeit mit dem
Christian Doppler Labor für Medizinische Strahlenforschung für die Radioonkologie
der Medizinischen Universität Wien

unter der Anleitung von

Univ.-Prof. Dr. DI Dietmar Georg

durch

Andreas Renner

Gudrunstraße 117/2/27
1100 Wien

Wien, 12. September 2015

Abstract

The typical course of a radiotherapy (RT) treatment is several weeks. In that time organ motion and shape changes can introduce uncertainties in the application of dose to cancerous tissue and organs at risk. Monitoring and quantifying these changes can yield a more precise treatment margin definition and thereby reduce dose delivery to healthy tissue and adjust tumor targeting. Deformable image registration (DIR) has the potential to fulfill this task by calculating a deformation field between a planning CT and a repeated CT of the altered anatomy. Application of the deformation field on the original contours yields new contours that can be used for an adapted plan. DIR is a challenging method and therefore needs careful user interaction and validation. Without a proper graphical user interface (GUI) a misregistration cannot be easily detected by visual inspection and the result cannot be fine-tuned by changing registration parameters. To provide a DIR algorithm with such a GUI available for everyone, we created the extension *Featurelet-Registration* for the open source software platform 3D Slicer. The registration logic is an upgrade of an in-house-developed DIR method, which is a featurelet-based piecewise rigid registration. The so called „featurelets" are equally sized rectangular subvolumes of the image which are rigidly registered to rectangular search regions on the target image. The output is a deformed image and a deformation field. Both can be visualized directly in 3D Slicer facilitating interpretation and quantification of the results. For validation of the registration accuracy two deformable phantoms are used. The performance is benchmarked against the initial in-house-developed algorithm with comparable or better results.

Contents

1. Introduction

Patient anatomy usually changes over the course of radiotherapy (RT) treatments, which can last for several weeks. During this time, patients typically loose weight, tumors can shrink or grow and organs like the bladder or the rectum have different filling at each treatment session. These factors lead to organ motion and shape changes between sessions as illustrated in Figure 1.1. Not taking into account the expected changes can lead to an underdose of the tumor and an overdose of the organs at risk (OAR). One strategy to accommodate for a known tumor motion is enlargement of irradiation margins, but this also leads to an increased irradiation of surrounding healthy tissue. Several previous studies [1, 2, 3, 4, 5, 6] have shown that a strategy accounting for patient-specific changes of organ position can provide both adequate tumor irradiation and a better OAR sparing. Thus, taking into account temporal effects of the tumor position is a benefit for the patient and can reduce side-effects of the RT treatment. However, such an individualized strategy requires image guidance techniques as well as sophisticated treatment planning. Four-dimensional (4D) RT is the explicit inclusion of temporal changes in anatomy during imaging, planning, and dose delivery. It is an intense area of study in radiation oncology [7]. *Yan et al.* [8] proposed *adaptive radiation therapy* (ART) as a tool to minimize the discrepancy between the target volume and the actual dose distribution. It requires

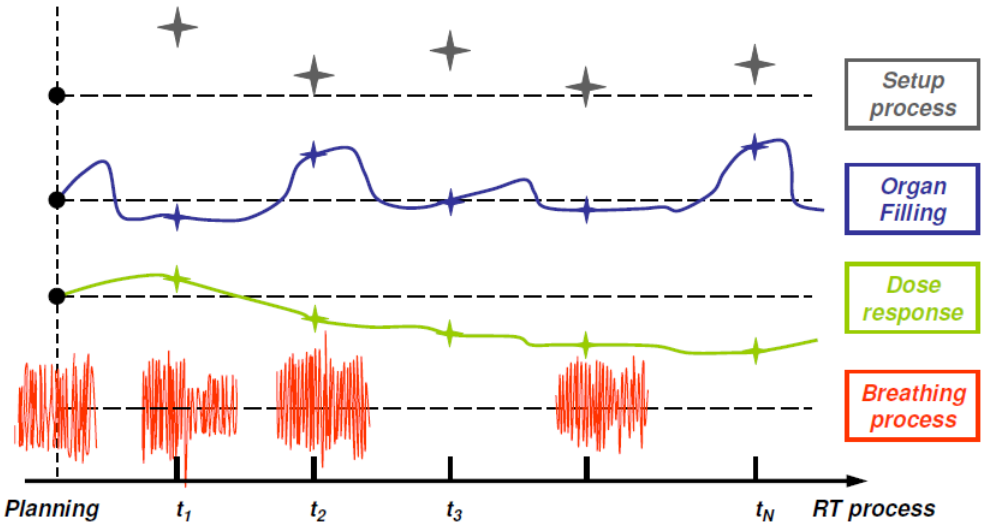

Figure 1.1.: *4D effects in RT: the setup process of each session, differences in organ filling and dose response as well as the breathing process have to be considered for a successful RT treatment (picture from [9]).*

finding the spatial relationship between all images of the same patient for treatment planning and a possible adaption of the plan. This can be a very time-consuming process as often a re-delineation of structures in the updated images is needed.

Deformable image registration (DIR) can solve this problem by deforming the contours from a planning CT to the altered anatomy of the patient represented by a rescanning CT or a cone-beam CT (CBCT). Image registration aims to geometrically align two images. A comprehensive survey of image registration in general and different algorithms available is given in [10, 11]. A systematic overview of DIR and recent advances in that field is given in [12]. One concept of a model-independent, multi-modal registration method is a *featurelet* algorithm based on the approach of *Söhn et al.* [13]. The authors suggest to break down the global problem of DIR to multiple independent rigid registrations of small subvolumes called „featurelets". This idea was picked up by *Fabri* [14] to develop an in-house DIR method. Subsequent validation studies [15, 16] have shown promising results for deformable phantoms, but for clinical datasets in some cases the algorithm did not satisfy.

The aim of this work was to build a graphical user interface (GUI) for the already existing in-house-developed DIR method to facilitate user interaction. It should allow for direct visualization of the output and quick adjustment of registration parameters. Thus, it should pose as a tool for investigation of cases with not satisfying registration results and facilitate further developments of the registration method. In addition we wanted the algorithm to be open source to ensure reproducibility.

To provide a DIR algorithm with such a GUI available for everyone, we created the extension *Featurelet-Registration* for the open source software platform 3D Slicer. It is a freely available software package for visualization and image analysis. 3D Slicer provides a powerful GUI to explore and interact with imaging datasets, and it is designed to be available on multiple platforms, including Windows, Linux and Mac Os X.

In the first part of the thesis an introduction to RT with a focus on external-beam RT is given (Chapter 2). Volume concepts in RT are presented and a short overview of image-guided RT (IGRT) is given as motivation for image registration. Chapter 3 is an introduction into image registration and gives an overview of different existing methods. Also the basic concept of the featurelet algorithm as well as several validation methods for the accuracy of a registration algorithm are presented there. Chapter 4 is about the software development tools that were necessary to create the 3D Slicer extension. In the following Chapter 5 the implementation of the featurelet algorithm is presented. There, the first Section 5.1 is about the software development of *Fabri* [14] which was used as basis for the *Featurelet-Registration* extension of Slicer. In the subsequent Sections details about the developed software are presented. In Chapter 6 validation studies of the featurelet algorithm are summarized. Section 6.1 provides an overview of already existing studies on the basic concept of

the featurelet algorithm and the software development of *Fabri*. Section 6.2 is about the validation of the Slicer implementation. In the final Chapter 7 the conclusion about our software development is presented and an outlook of future extensions to the algorithm is given.

2. Radiation Oncology

During the lifetime one in 3 women and one in 2 men in the United States will develop cancer [17]. Common treatment techniques are surgery, chemotherapy and radiotherapy. The latter one is covered in more detail in the present chapter. A comprehensive overview of radiation oncology in general and the basis of medical physics as well as recent advances in radiotherapy techniques is amongst others given in [18] and [19].

Radiotherapy (RT) or radiation oncology aims to destroy cancerous cells by utilizing ionizing radiation. As main beam delivery system linear accelerators are used. Beside tumor control, minimization of damage to surrounding healthy tissue has to be considered. This selectivity can be achieved by exploiting the difference of tumor control probability (TCP) and normal tissue complication probability (NTCP) as shown in Figure 2.1. Biological response processes depend on spatial and temporal distribution of the applied dose. The temporal contribution is considered by splitting up the total dose in several daily doses, so called *fractions*, which can be delivered over a period of weeks. It was empirically found that this method results in good tumor control and less severe side effects. The spatial contribution of the applied dose is the main topic of the following sections.

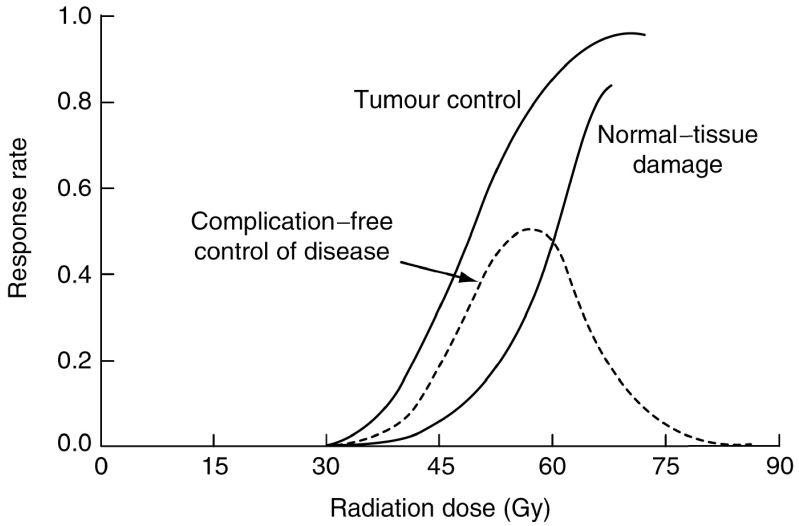

Figure 2.1.: *Therapeutic window: the probability of tumor control and normal-tissue damage shown as function of the radiation dose; between these two graphs is a region where complication-free control of the disease (dashed line) is possible (picture from [18]).*

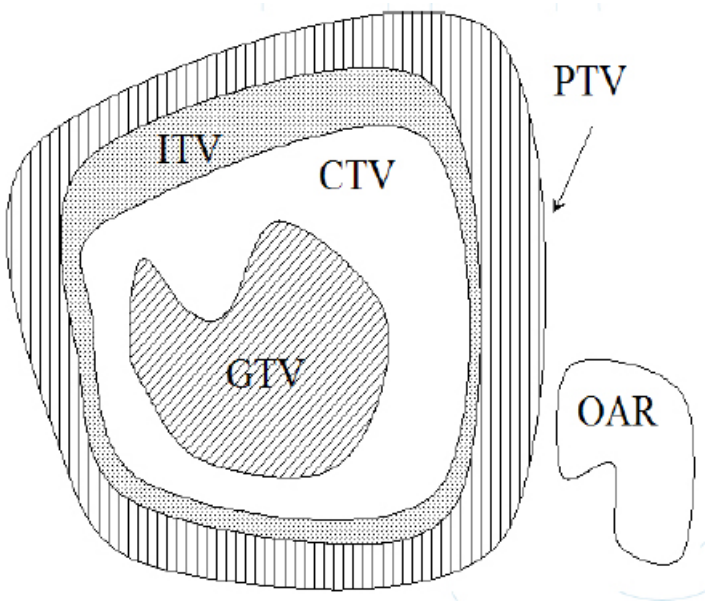

Figure 2.2.: *Volume concepts in RT: graphical illustration of the volumes-of-interest, as defined by the ICRU 50 and 62 reports [20] (picture from [19]).*

2.1. Volume Concepts in Radiotherapy

The basis for the treatment planning process as well as for comparison of treatment outcomes is the definition of certain volumes as a common language for all professional groups involved in RT. This is done in ICRU Report 50, 62 and 83 [20, 21, 22]. The following volume concepts are provided as principal volumes related to 3D treatment planning (compare Figure 2.2) [19]:

GTV Gross Tumor Volume: *„is the gross palpable or visible/demonstrable extent and location of malignant growth."* (ICRU Report 50) The GTV may consist of the primary tumor, metastatic regional nodes, or distant metastasis.

CTV Clinical Target Volume: *„is the tissue volume that contains a demonstrable GTV and/or subclinical microscopic malignant disease, which has to be eliminated. This volume thus has to be treated adequately in order to achieve the aim of therapy, cure or palliation."* (ICRU Report 50)

ITV Internal Target Volume: is a newer concept defined in ICRU Report 62, that attempts to divide treatment inaccuracies into internal patient factors and external factors. The ITV consists of the CTV and an additional internal margin which considers variations in the size and position of the CTV relative to a reference frame of the patient.

PTV Planning Target Volume: *„is a geometrical concept, and it is defined to select appropriate beam arrangements, taking into consideration the net effect of all*

possible geometrical variations, in order to ensure that the prescribed dose is actually absorbed in the CTV." (ICRU Report 50) The PTV is an expansion from the ITV to account for external treatment inaccuracies such as set-up uncertainties, machine tolerances and intra-treatment variations. It is linked to a reference frame of the treatment machine. Consideration of internal and external uncertainties together leads to an expansion from the CTV to the PTV.

OAR Organ at Risk: are normal tissues and organs adjacent to the PTV which are non-target. The radiation sensitivity of OARs may significantly influence treatment planning and/or prescribed dose. For many OARs a certain tolerance dose is defined at which a given level of late radiation morbidity can be expected [18]. In general, the aim should be to minimize irradiation of OARs.

PRV Planning Organ at Risk Volume: OAR volumes uncertainties through set-up and organ movement have to be considered in the same way as for the CTV.

2.1.1. Margins in RT

For the success of a treatment it is important that the CTV is irradiated with the prescribed dose. This is ensured by considering possible geometrical uncertainties in the planning process. Occurring errors can be:

- Poor organ/tumor definition: Some tumors have a similar radiographic density compared to the surrounding healthy tissue or they can have diffuse infiltration at the tumor periphery. In these cases delineation of the GTV is not obvious and it is possible that the shape of the tumor and its size will not be completely appreciated by the clinician. This can lead to a considerable inter-observer variation [18].

- Set-up error: Patient set-up is usually performed based on markers placed on the skin. Reproducibility of the set-up is limited by the motion of the skin relative to internal anatomy [23]. Newer approaches reduce this error by the use of image-guidance.

- Internal organ movements: Organ motion with respect to bony anatomy introduces additional uncertainties. Different filling states of hollow organs such as bladder and rectum or respiration can lead to tumor motion and shape changes from day to day and even throughout a single day [18].

Thus, margins are used to deal with errors that occur both relative to treatment planning as well as during the actual treatment delivery. As mentioned above this is done by expanding the CTV or the ITV with a safety margin to obtain the PTV. High dose is delivered to the PTV to ensure that the CTV receives the required dose for tumor control despite the presence of geometrical errors. The downside of considering errors by the use of safety margins is a high-dose irradiation of healthy

tissue. Thus, margins should not be excessive, but sufficient. Detailed information about errors and margins in RT is given in [23].

2.2. Advanced Treatment Techniques

In this part, we cover external beam RT techniques, which do not fit into conventional treatment planning. A tumor volume is usually irregular-shaped. Conventional RT uses simple rectangular treatment fields, which are not sufficient to provide an ideal irradiation to such a volume. Accordingly, normal tissue has to be irradiated as well to guarantee that the CTV receives the prescribed dose. With the aim to deliver a possibly high dose to the tumor while sparring the surrounding tissue and OARs new approaches have been developed to achieve higher conformity of CTV and the delivered dose distribution. Such techniques are called *conformal RT* (CFRT) and can be divided into two broad classes:

- conventional CFRT, involving only geometrical field shaping and
- intensity-modulated RT (IMRT) techniques.

The 3D dose distribution of the first method can only have a convex shape, whereas the second technique is able to produce a concave 3D dose distribution. Because of its high achievable conformity, IMRT is a rapidly expanding subspecialty of CFRT and is presented in more detail in the following section.

The reason CFRT is possible nowadays is a simultaneous development in all different fields involved in the process of RT treatment. The single steps required for a successful therapy are called „chain of RT" and are illustrated in Figure 2.3 [18].

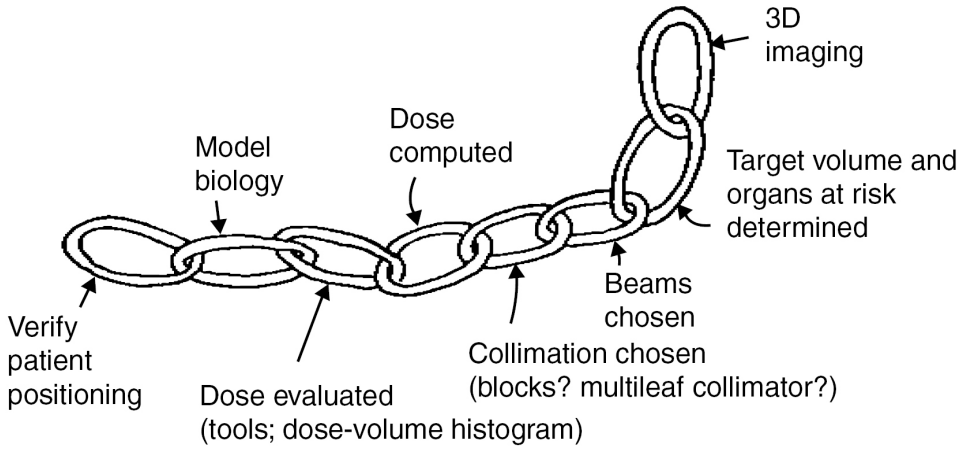

Figure 2.3.: *Chain of RT: to achieve the aim of higher conformity of dose distribution and CTV a whole chain of processes is involved. At each stage good performance is required (picture from [18]).*

2.2.1. IMRT

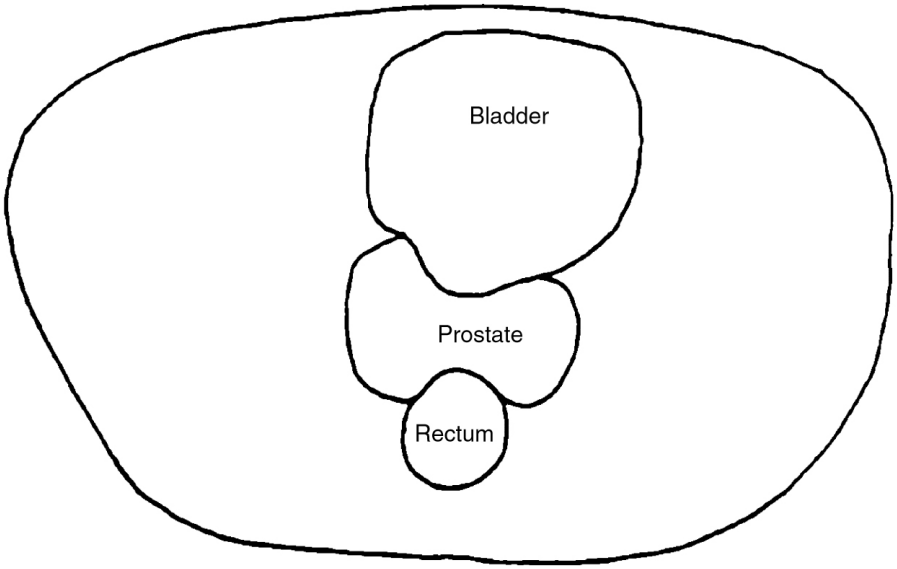

Figure 2.4.: *Illustration why IMRT is needed for some cases; the prostate has a concave shape and both bladder and rectum are in its proximity. An OAR in the concavity of the PTV limits dose escalation (picture from [18]).*

Intensity-modulated RT (IMRT) offers an enhancement of conformal high-dose irradiation. This is achieved by the use of several treatment fields with nonuniform photon fluence. A single field is divided into subfields of uniform photon fluence. The latter one can be varied independently. Thus, the dose of one or several subfields from one direction can be reduced when there is an OAR within the treatment field. The missing dose can be compensated from another beam angle. Thus, IMRT is able to create concavities in treatment volumes as it is the case for prostate shown in Figure 2.4. In addition this approach reduces high dose areas outside the CTV and therefore supports dose escalation for the tumor [18].

Internal tumor and organ movement as well as uncertainties in patient set-up are important when considering IMRT, because of tight margins and steep dose gradients. Thus, taking advantage of the progress in imaging techniques is needed to improve the accuracy of treatment delivery. Daily imaging before each fraction is required to capitalize on the potential benefits of high-precision RT [24, 18].

2.2.2. Image-guided Radiotherapy

The accuracy of a treatment delivery can be improved by the use of advanced imaging methods throughout the whole process of RT. To describe this, the term Image-guided RT (IGRT) is used (Mackie et al. 2003; Bortfeld et al. 2006). However,

IGRT is not a precise concept. Most RT treatments in recent years could be described as image guided. The term IGRT as presently used implies the use of a 3D imaging system on the treatment machine, allowing for accurate assessment of the patient setup and the tumor in relation to the beam before each individual fraction [18]. The traditional approach to implement IGRT is to use the megavoltage treatment beam and a so called electronic portal imaging device (EPID). The downside of using the treatment beam for imaging is a lack of soft tissue contrast to show organ motion. Therefore, kilovoltage imaging modalities like cone-beam computed tomography (CBCT) are placed inside the treatment room or even within the treament machine to solve this problem.

Target motion results in smearing of gradients of the dose distribution, which leads to smoothing of dose inhomogeneities within the target and dose washout beyond the target edges. This is usually considered with increased planning margins to ensure desired irradiation of the target [23]. Using the information of IGRT regarding the tumor motion has the potential to reduce margins and possibly lead to an adaption of the treatment plan. The availability of on-line image data from IGRT presents a challenge to the treatment staff. They are required to make instant interpretations of the data in order to decide whether to correct the position of the patient or not before proceeding to treat the patient. In addition, techniques such as target of the day and adaptive RT (Yan et al. 1997; Hugo et al. 2007) are currently being evaluated [18].

2.3. Adaptive Radiotherapy

If anatomical changes such as weight loss or tumor shrinkage as well as organ motion cannot be accounted for by adjusting the position of the patient, an adaptation of the treatment plan may be necessary [25]. For these cases *Yan et al.* [8] proposed *adaptive radiation therapy* (ART) as a tool to minimize the discrepancy between CTV and the actual dose distribution through the concept of a closed feedback loop control. It consists of four sub-processes [26]:

- assessment of treatment dose
- identification/evaluation of possible treatment variation
- decisions if treatment modification is necessary and
- adaptive treatment modification.

On-line imaging introduced in the course of IGRT enables to identify geometric uncertainties. If a numerical model of tumor motion is available, which predicts the behavior of the system, the last step of the feedback loop can be automatically applied. Re-optimization of the treatment plan is triggered only after an unacceptable error is acknowledged based on updated patient images. In practice, the feedback

loop of ART is not completely closed but manually triggered [26, 27].

A very time-consuming process in ART and re-planning in general is finding the spatial relationship between all the images of the same patient, as it often requires manual re-delineation of structures in the updated image. Deformable image registration (DIR) can solve this problem by deforming the contours from a planning CT to the altered anatomy of the patient represented by a rescanning CT or a CBCT [25].

3. Image Registration

> *Medical imaging modalities do not provide images of the patients anatomy and physiology. They do record certain physical properties of tissue.* Birkfellner, 2011 [28]

Different modalities yield different information about the anatomy. This fact can be used to get a more comprehensive „insight" into the patient. Also sometimes changes of the anatomy over time are useful to know. They are usually not visible on a single image. Thus, a series of images of the same patient is needed. To be able to evaluate such results of different modalities or different points in time, there has to be some information about which point in one image corresponds to a certain point in the other image. To get this information image registration is used.

The process of image registration can be described as the overlaying of two or more images of the same scene, but taken with different imaging conditions such as different viewpoints or different time points. In addition, different imaging modalities show different anatomy. Image registration aims to geometrically align two images. In this thesis, these two images are referred to as *fixed image* and *moving image*. Unfortunately, there is no single method which can achieve the alignment for every possible setting. A registration method has to take into account the diversity of possible images to be registered as well as various types of degradations (e.g. noise corruption) [11].

A comprehensive overview of image registration in general and different algorithms available is amongst others given in [10] or [11].

3.1. Classification of Registration Methods

As there is a great number of different, sometimes very specialized applications, there is also a variety of different registration algorithms. To get a better overview of existing methods they can be subdivided into different categories according to several possible criteria. *Maintz et al.* try *„to paint a comprehensive picture of current medical image registration methods"* and use the following nine criteria to classify different registration methods [10]:

- Dimensionality: e.g. 2D-2D, 2D-3D or 3D-3D

- Nature of registration basis
  - Extrinsic: based on external objects introduced into the image space (e.g. gold markers)
  - Intrinsic: based on the image information of the patient itself
  - Non-image based: possible in the case of two scanners which have coordinate systems calibrated to each other
- Nature of transformation
  - Rigid
  - Affine
  - Non-rigid
- Domain of transformation
  - Local
  - Global
- Interaction
  - Interactive
  - Semi-automatic
  - Automatic
- Optimization procedure
- Modalities involved
  - Mono-modal
  - Multi-modal
- Subject
  - Intra subject: all images involved in the registration are from one patient
  - Inter subject: images are from different patients
  - Atlas: images from a single patient are registered to a database (obtained by the use of images from many subjects)
- Object: e.g. head, thorax etc.

In this thesis not every single criterion needs to be covered in detail. For the following section the most important one is the nature of transformation.

3.1.1. Transformation Models

For every image registration algorithm a mapping function is needed. The type of the underlying transformation model should correspond to the assumed geometric deformation of the moving image and to the required registration accuracy [11]. In Figure 3.1 several transformation models are shown according to the degrees of freedom (DOF).

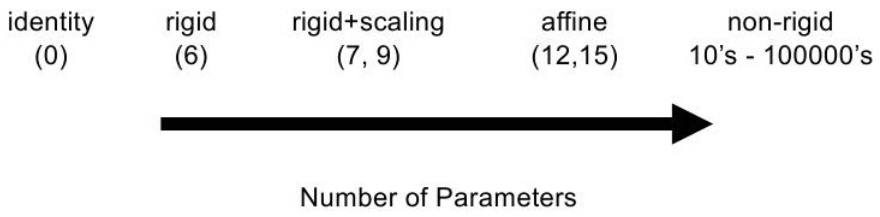

Figure 3.1.: *Different transformation models ordered by the number of parameters [14].*

A rigid image coordinate transformation only allows for translation and rotations. Thus, in 3D it has 6 DOF. If the transformation additionally allows for scaling and shearing it is called affine. A composition of several transformations is categorized as a single transformation of the most complex type in the composition. Both rigid and affine transformations can be described with a single matrix equation [10].

For a more complex deformation of an image a non-rigid transformation is needed. In this case the image can be viewed as a rubber sheet which is stretched by external forces. Stiffness or smoothness constraints act as internal forces to achieve alignment with minimal amount of stretching [11]. In this thesis non-rigid methods are referred to as *deformable image registration*.

3.2. Deformable Registration

A systematic overview of deformable image registration and recent advances in that field is given in [12]. The findings of this survey are presented in the current section. The authors claim three main components of a registration algorithm: a deformation model, an objective function and an optimization method.

3.2.1. Deformation Models

The DOF of a deformation model correspond to the number of parameters the registration algorithm has to extimate at the optimization process. Thus, the choice of a certain deformation model is a trade-off between computational efficiency on the one hand and richness of description of the registration problem on the other hand. Deformation models can be classified into three different categories of geometric transformations (compare Figure 3.2):

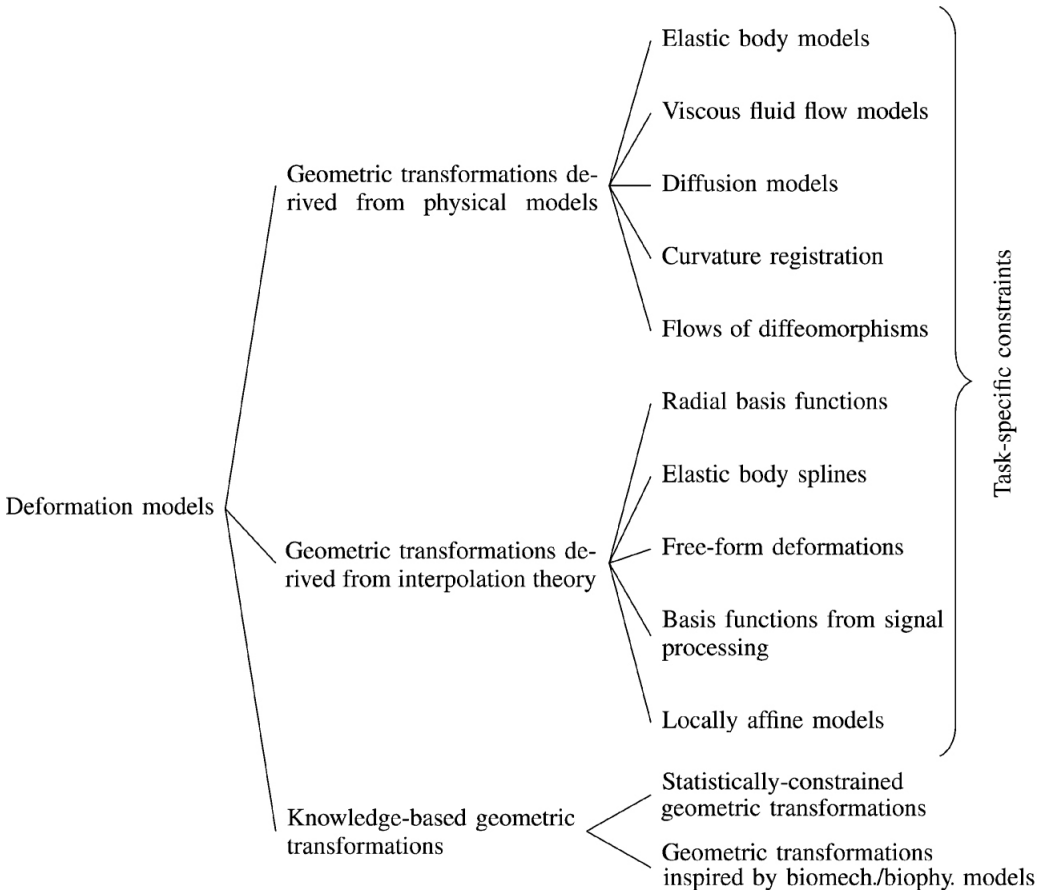

Figure 3.2.: *Classification of deformation models; task-specific constraints are topology preservation, volume preservation and rigidity constraints; detailed information about the single models and references to original papers can be found in [12].*

- Physical models: use a physical model for the deformation; can be computationally expensive.
- Interpolation theory: do not guarantee, that the deformation is according to natural laws; uses interpolation and approximation theory instead.
- Knowledge-based: includes prior information of the registration problem.

In addition there are models which satisfy a task-specific constraint, but in general they are used together with physical models or interpolation theory.

3.2.2. Matching Criteria

Three different groups of registration methods can be distinguished by how information is used to drive the matching process (compare Figure 3.3):

- Geometric methods aim to find correspondences between meaningful anatomical locations in the image. Such positions are called *landmarks*. Once such a

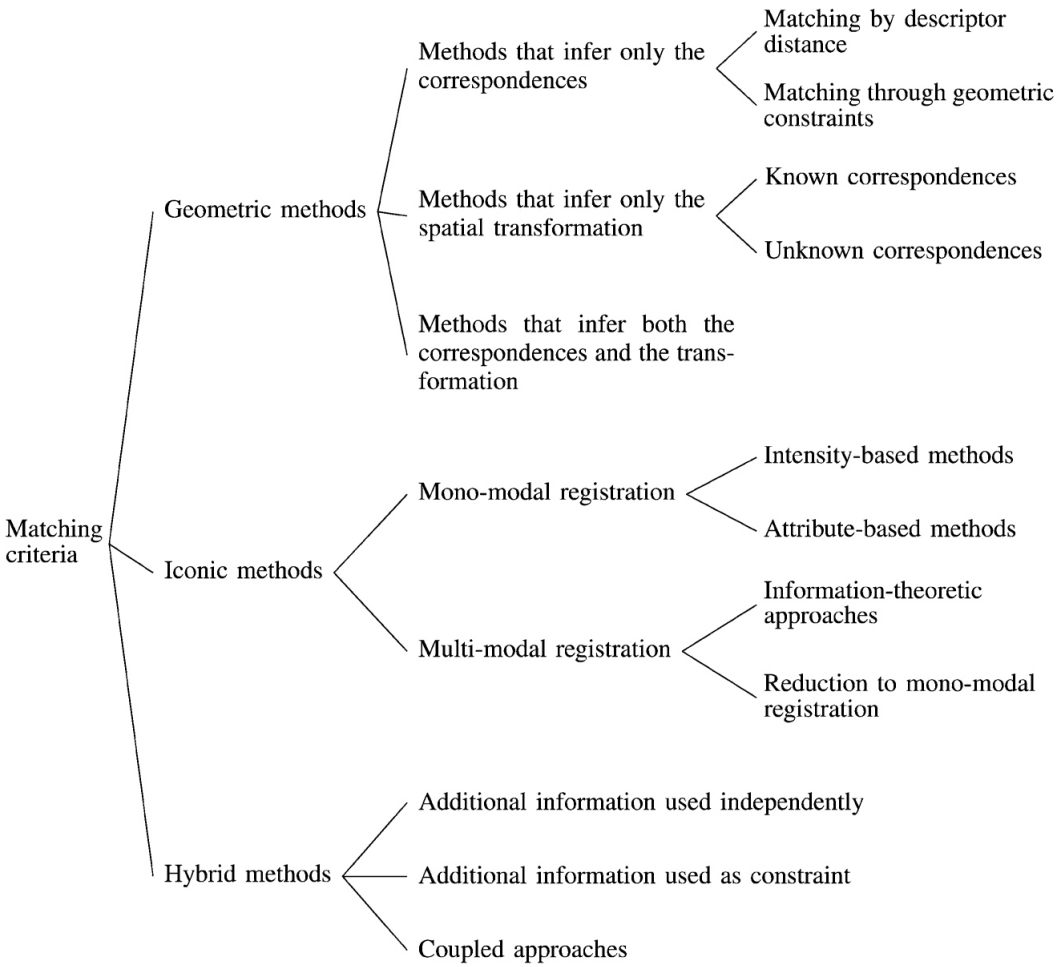

Figure 3.3.: *Classification of matching criteria; detailed information about the different criteria and references to original papers can be found in [12].*

landmark has been identified the registration problem is straightforward. Geometric registration can also handle large deformations. However, landmark extraction can be a difficult task and is an active topic of research.

- Iconic methods are also called *voxel-based* or *intensity-based* methods. They evaluate the alignment of two images by means of an intensity-based criterion. Such a criterion usually takes into account the whole image domain. Thus, iconic methods are computational more expensive. In addition, the obtained result is influenced by initialization. Nevertheless, this approach comes with the potential to better represent a dense deformation field.

- Hybrid methods are a combination of geometric and iconic methods. They try to use the advantages of each method.

Voxel-based Registration Methods

As for the featurelet algorithm voxel-based methods are used this particular topic is covered in more detail.

Voxel-based methods take into account the whole intensity information of an image. As there are different physical principles behind the numerous available image acquisition techniques, the criterion should be able to account for different intensity relations between two images. Thus, different matching criteria are necessary for images obtained with one modality and multiple modalities.

In the mono-modal case the sum of squared differences (SSD) and the sum of absolute differences (SAD) can be used if an identity relationship between image intensities is assumed. The latter one is less sensitive to outliers. In the case a linear relation between signal intensities is assumed correlation (C) and normalized cross correlation (CC) can be used. The latter one is useful if images were acquired with different intensity windowing.

In the multi-modal case two main approaches are used. The first one is the reduction of the multi-modal problem to a mono-modal problem. This can be done either by simulating one modality from another or by mapping both modalities to a common domain. The second approach is the use of information theoretic measures. Popular methods are mutual information (MI) and normalized mutual information (NMI). The latter one is indepentend from the amount of overlap of two images.

3.2.3. Optimization Methods

The optimization process tries to find the best alignment between two images according to a function which evaluates a certain matching criterion as well as a regularization term. Thus, the choice of an optimization methods also affects the quality of the result. Two categories based on the variables an optimization method tries to infer can be distinguished (compare Figure 3.4):

- Continuous: The variables take real values and the function of the matching criterion is differentiable. To estimate the optimal parameters continuous optimization methods use an update rule given by:

$$\vec{\Theta}_{n+1} = \vec{\Theta}_n + \alpha_n \vec{g}_n(\vec{\Theta}_n) \tag{3.1}$$

  with the vector of parameters of the transformation $\vec{\Theta}$, the number of iteration n, the step size α_n and the search direction $\vec{g}$. The search direction takes into account the matching criterion as well as the regularization term.

- Discrete: The variables take discrete values. Recently, discrete *Markov random field* (MRF) formulations have been investigated to be used for image

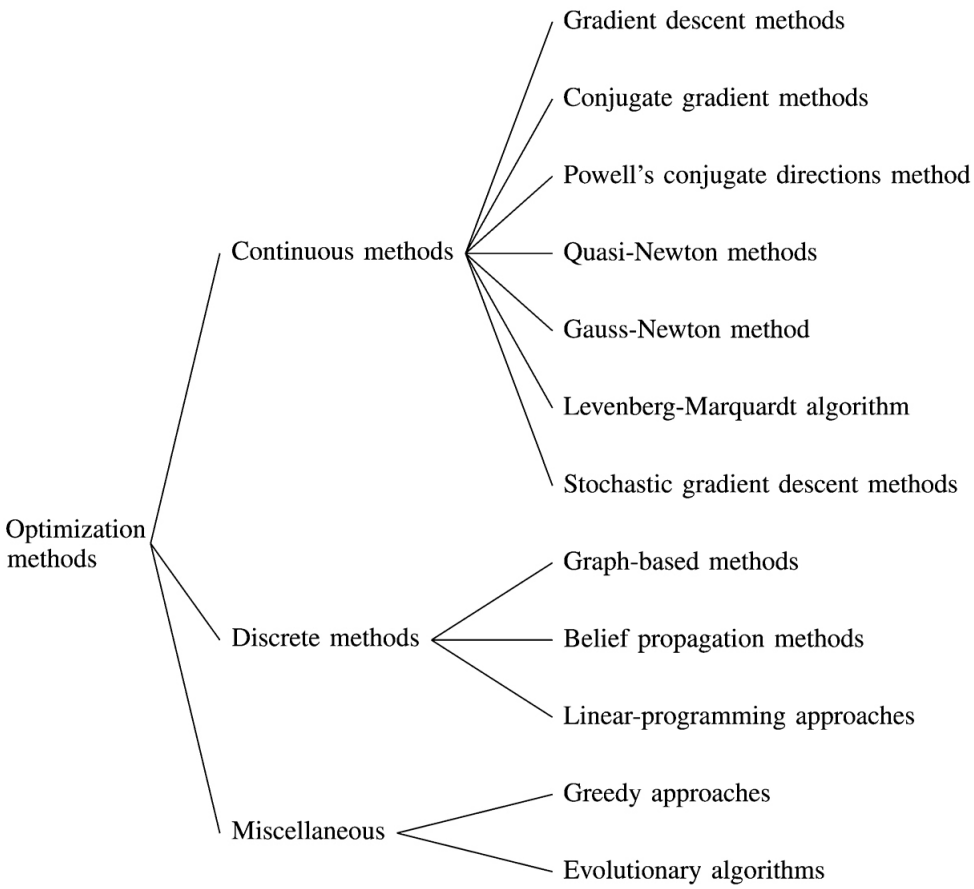

Figure 3.4.: *Classification of optimization methods; detailed information about the single methods and references to original papers can be found in [12].*

registration. In the domain of probability, a MRF is a undirected graphical model represented by:

$$G = \{V, E\} \tag{3.2}$$

with the undirected graph G consisting of a set of vertices V and a set of edges E. The nodes of the MRF encode the random variables. These random variables can take values of a discrete set of labels. The edges connect the nodes and represent the relationship between the variables. The MRF is solved by assigning the optimal label to each node.

Beside continuous and discrete optimization methods there are heuristic and meta-heuristic approaches such as evolutionary approaches representing a third category. Because of their global optimization nature [29], they are able to explore a large solution space, but do not provide any guarantee regarding the optimality of the solution.

3.2.4. Demons Registration

As an example of a deformable image registration algorithm the demons method of *Thirion* [30] is presented. The reason why such an algorithm was chosen to be presented in more detail is because in Chapter 6 the results of the *Featurelet-Registration* are compared to a demons registration method.

Thirion proposed a physical deformation model based on the thermodynamic concept of diffusion. In analogy with Maxwell's demons in one image the boundaries of the structures are considered as semi-permeable membranes which let the other image diffuse through these interfaces. For this step the second image is regarded as a deformable grid. This diffusion model is illustrated in Figure 3.5.

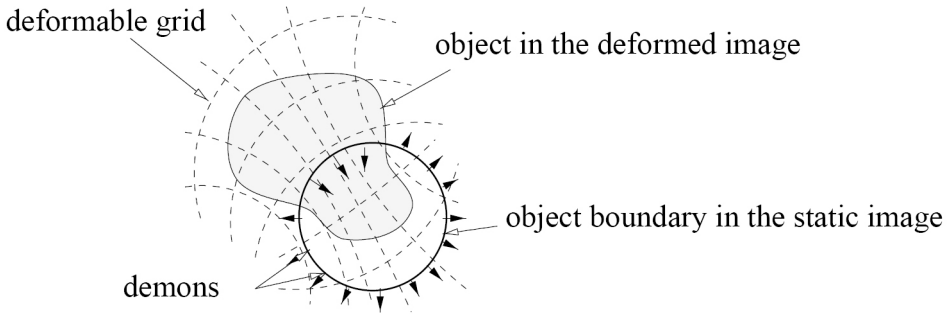

Figure 3.5.: *Thirions diffusion model for demons registration; the moving image, here considered as a deformable grid, is diffusing through the contours of the structures in the fixed image. This action is controlled by the so called „demons" sitting within these interfaces (picture from [30]).*

The diffusion model relies mainly on the concept of polarity, which is in this case „inside" or „outside". In addition, for a demons registration an attraction force depending on the distance can be introduced by an optical flow method.

Implementation of the Diffusion Model

Given a moving image M and a fixed image F it is assumed that:

- M is a deformable grid with particles as vertices; the particles can be classified as „inside" or „outside"
- and the contour of a structure S in F is a membrane.

Along this contour there are the demons. A demon is defined as an effector situated in a point P of the boundary of a structure S. If a point of M is labeled with „inside" or „outside" the demon either pushes it inside or outside of S. For the optical flow the intensity function f in F as well as the intensity m in M are introduced at a given point P. Based on the gradient $\vec{\nabla} f(P)$ a direction $\vec{r}$ from the inside to the

outside can be attached to each demon. The aim is to iteratively compute a final transformation T between the space of the moving image M and the fixed image F. After extracting a set of demons D_F from the fixed image F, the iteration consists of two steps:

- For each demon $d \in D_F$ the associated elementary demon force $\vec{a}_n(d)$ is computed. This force depends on the direction $\vec{r}_F$ of the demon at point P and on the polarity of M at point $T_n^{-1}(P)$.
- Using all the elementary demon forces T_{n+1} can be calculated out of T_n.

The use of *Demons*, as introduced by Thirion, was an efficient algorithm providing a dense correspondence, but lacked a sound theoretical justification in the beginning. Thus, numerous papers tried to give theoretical insight into the mechanisms of this successful algorithm [12].

3.3. Featurelet Registration

The concept of the algorithm, which is used for the 3D Slicer extension, is based on the approach of *Söhn et al.* [13]. The authors suggest to break down the global problem of deformable image registration to multiple independent rigid registrations.

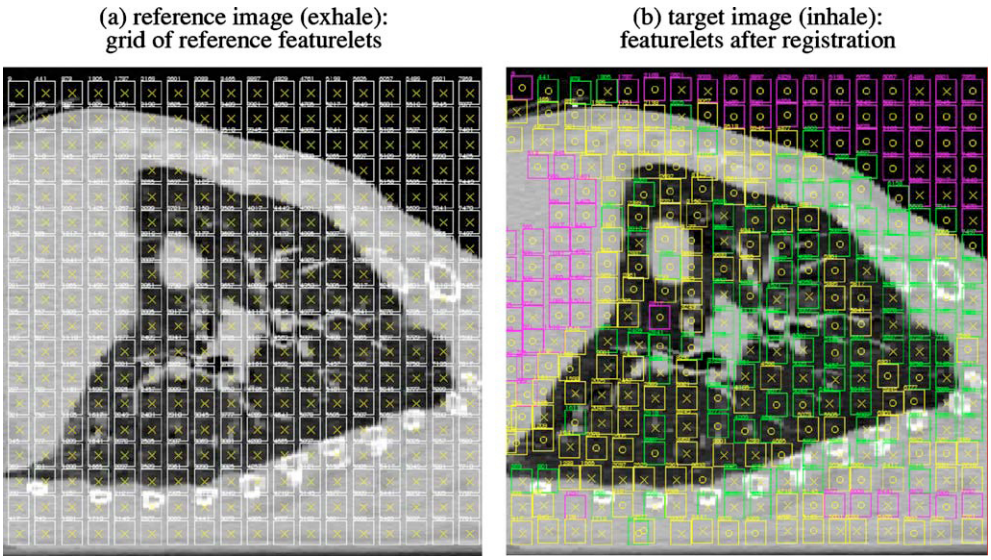

Figure 3.6.: *Sagital view of a lung; the images are subdivided into a grid of featurelets. The left side shows the grid before registration (a) and the right side the deformed grid after registration (b). Featurelets that are missing in (b) moved completely out of the presented slice (picture from [13]).*

As illustrated in Figure 3.6 the image is divided into regularly distributed subvolumes, the so called „featurelets". It is assumed that a featurelet is small and

deformed only by a small amount. Thus, the deformation can be approximated by a 3D rigid registration. In addition, it is assumed that the fixed and the moving image are already rigidly preregistered. Thus, the center of mass shift as well as the bulk rotation should be eliminated. The featurelet-based image registration algorithm can be represented by five steps:

Step 1: Initialization of the featurelet grid

The region of the moving image which is indented to be registered is covered by a 3D grid of equally sized rectangular featurelets. The size of a featurelet is a parameter of the algorithm. It has to be chosen according to the typical size of distinct anatomical features.

Step 2: Local rigid registration of featurelets

Each featurelet is registered to the fixed image. For algorithmic efficiency possible transformations are restricted to translations. For every featurelet in the moving image a corresponding rectangular local search region in the fixed image is defined. The size of the search region is another parameter of the algorithm. It has to be chosen according to the typical shift of a featurelet. Two different voxel-based similarity measure functions are used: the correlation coefficient (CC) for intra modal registration and the normalized mutual information (NMI) for inter modal registration.

Step 3: Assessment of local registration quality

The registration quality of an individual featurelet can be assessed by the configuration of the similarity measure around the optimum. Three cases are distinguished:

- Case I: local image information of a featurelet is high enough to find a unique, distinct optimum of the similarity measure; in Figure 3.6 (b) featurelets with green frame.
- Case II: local image information is not enough to find a unique registration; similarity measure function has a degenerate optimum; in Figure 3.6 (b) featurelets with yellow frame.
- Case III: low or no image contrast of a featurelet; similarity measure function has no clear optimum and low similarity values for all possible shifts; in Figure 3.6 (b) featurelets with magenta frame.

For case III the deformation of the featurelet can not be determined based on image contrast alone. Here, additional information from neighboring featurelets is used. This is introduced by a subsequent relaxation step.

Step 4: Relaxation

To assure that the deformation field is physically meaningful additional assumptions are made. This is implemented as minimization of a deformation energy given by a spring-mass model. To each featurelet a virtual mass m_i is

assigned. The dynamics are represented by the center position X_i which has to satisfy the following differential equation:

$$m_i \ddot{X}_i = -D\dot{X}_i + F^{[i]}_{springs} + F^{[i]}_{image} \tag{3.3}$$

where D is a damping factor, $F^{[i]}_{springs}$ are the „internal" and $F^{[i]}_{image}$ the „external" forces. The internal forces are the resulting forces introduced by virtual springs connected to the center of each 3D neighbor. The external forces are given by the gradient of the similarity measure. The minimum deformation energy is represented by the asymptotic equilibrium where speed and acceleration for all featurelets vanish $\rightarrow \ddot{X}_i = 0 = \dot{X}_i$. For the parameters $D = 1.4$ and $m = 1$ was chosen for case II and III. To effectively fix the well-registered featurelets a high mass of $m = 10^7$ was assigned for case I.

Step 5: Calculation of global deformation field
The shift vectors δX_i of all featurelets sampled on the grid of featurelet centers of the fixed image represent the total displacement field. The deformation field for any possible position is obtained by a B-spline interpolation of order 1 between the δX_i.

The main advantages of the featurelet-based approach is a fast calculation time of well below one minute for a modern multiprocessor PC and the fact that the method is model-independent. Thus, maximum portability for use with different organ sites as well as different image modalities is assured [13]. Validation studies of the featurelet registration algorithm of *Söhn et al.* are presented in Section 6.1.1. Beside the general hospital vienna, a comparable approach is used by *Robertson et al.* [31] for a block matching-based registration of locally advanced lung tumors.

3.4. Validation of Registration Accuracy

For a user, a very important characteristic of a registration algorithm is accuracy. It is a direct measure of the actual, „true" error at a specific image location [10]. Thus, for the validation of the registration accuracy a *true answer* i.e. a gold standard is needed. Unfortunately such a gold standard is generally not available for deformable image registration. In literature, many different methods for validation have been reported, but in most cases comparison of the accuracy claimed for one method with the accuracy claimed for another one is difficult because of methodological incompatibilities [32].

3.4.1. Validation Methods

Roger P. Woods discusses in [32] several strategies available for validation studies of registration accuracy:

Validation by Visual Inspection An easy to „implement" qualitative validation method is visual inspection of the registration result. This may seem like an unreliable approach, but in [33] and [34] the authors have shown that a misregistration of 2 mm can be reliably identified. In general, if a result looks misregistered, it probably is. Thus, visual inspection should be used at every image registration as a routine validation.

Validation of Point-Based Registration A basic method for rigid registration is the selection of homologous points in the fixed and moving image. This can be done by a person with anatomic expertise or with the use of internal (e.g. gold markers) or external markers. In three dimensions three points would be sufficient to compute the registration parameters. However, the selection of corresponding points is usually inaccurate. Thus, more than three points are chosen and the registration procedure tries to minimize the summed squared distance between homologous points. The minimization is split up in a translational component and a rotational component. The first one is solved by aligning the centroids of the points of both images. The second one is then derived by the so called *Procrustes algorithm*. With increasing distance from the centroid of the fiducials the errors grow larger. Thus, for the selection of fiducial points it is preferable to use points that are on the one hand widely dispersed and on the other hand as far from their centroid as possible. The use of external fiducials markers is especially advantageous when registration accuracy is extremely important e.g. during neurosurgical procedures. The external fiducials are usually located farther from their centroid as it would be the case for anatomically identified fudicials. In addition they are designed to be easily seen and detected with high accuracy and are often constructed as part of a stereotactic frame with known spatial relationships among the fiducials. Thus, external fiducial frames can be used to derive gold standard transformations for validation of other registration methods.

Cross-Validation It is also possible to use the results of an already validated registration method for cross-validation of a new method. Hereby, only highly accurate methods that can be considered as *gold standards* should be used as reference.

Simulations When there is no gold standard available a simulation can be used to estimate registration accuracy. In that case real data is deformed with an appropriate transformation model to create a virtual second image. Thus, the entire deformation field is known precisely and can be compared with the output deformation field of the registration of the initial image and the virtual second image. Although this method can be very helpful in optimizing the performance of a registration algorithm some important limitations have to be taken into account. A simulation is always based on certain models of reality. If these models (e.g. for spatial transformation, interpolation, or noise) are congruent to the models used for the registration method the estimated

registration accuracy will be very good. Nevertheless the algorithm can fail in the real world.

Phantoms and Cadavers Phantoms designed for medical imaging can reproduce many limiting factors regarding registration accuracy. Especially characteristic distortions produced by the imaging equipment or noise introduced either by the imaging equipment or by image reconstruction are included by the use of phantoms and cadavers for validation.

Internal Consistency Internal inconsistencies of a registration method itself can be used for placing a boundary on true registration accuracy. This is done by comparing two transformations: the result of the registration of an image A to another image B and the result of the registration of the image B to the image A. This procedure is not possible for every algorithm as some of them are designed to output the exact inverse transformation of one another in this context. For these registration methods a strategy involving three images is necessary. In that case the combined transformation of the registration of image A to B and image B to C is compared with the transformation of the registration of image A to C.

If there would be a reliable gold standard available, there is still the question how to report registration errors. For one user the average error at a certain location, or as an alternative the root mean square (RMS) error at this point might be of interest. For another one the most extreme error at a certain structure is useful, but the extremes of a distribution can be misleading as they are usually not reproducible, especially with a small number of observations. To be able to compare a distribution of errors at a point of interest, the Kolmogorov-Smirnov test[1] can be used to test whether two distributions are significantly different or not.

A problem that remains is that errors are very likely to be not homogeneously distributed over the image. Thus, to report the whole information a 3D plot of all errors showing the direction and magnitude would be necessary, which is unfortunately not very practical to handle. Thus, a common approach is to collapse all errors to the same point and show the distribution in a single plot [32].

3.4.2. Validation Metrics

In the case of nonrigid registration usually there is no gold standard available. A single metric is not sufficient to evaluate image registration results. However, the use of different metrics can provide insight of the nonrigid image registration performance [35]. *Christensen et al.* [35] propose for their *Non-Rigid Image Registration Evaluation Project (NIREP)*[2] four different metrics:

[1] The Kolmogorov-Smirnov test is a statistical test for equality of continuous, 1D probability distributions.

[2] http://www.nirep.org/

Relative overlap The alignment of corresponding objects or structures within two images can be used as indicator of how well the registration algorithm performed. With P and S as two corresponding segmentations the relative overlap metric is given by:

$$RO(P,S) = \frac{volume(P \cap S)}{volume(P \cup S)} \tag{3.4}$$

Generally, this metric is used to determine how well one image is registered onto another image. For comparing two registration algorithms the relative overlap on its own is not satisfactory as there might be biases and errors due to noise in the images, differences in anatomy and of course errors in the segmentations P and S. To reduce these errors, a large number (N=16) of datasets is proposed.

Squared intensity error Another method to measure the performance of a registration algorithm is to register a certain amount of images with a target image and take the average intensity of all registered images. An algorithm with a better performance should give registered images that look more like the target image. This gives a sharper intensity average image. One way to characterize the sharpness of the intensity average image is to calculate the variance of the registered intensity images. The voxel-wise intensity variance ($IV_j(x)$) or *squared intensity error* of a population of M images registered to an image j is given by:

$$IV_j(x) = \frac{1}{M-1}\sum_{i=1}^{M}\Big(T_i\big(h_{ij}(x)\big) - ave_j(x)\Big)^2 \quad ; ave_j(x) = \frac{1}{M}\sum_{i=1}^{M} T_i\big(h_{ij}(x)\big) \tag{3.5}$$

T_i is the i^{th} image of the population and $h_{ij}(x)$ is the transformation from image i to j with respect to an Eulerian coordinate system.

Inverse consistency error This metric evaluates the performance of a registration algorithm based on desired transformation properties. It compares forward and reverse transformation between two images. In the optimal case the forward transformation should be the same as the inverse of the reverse transformation. Thus, when there is no error the composition of the forward and the reverse transformations should produce the identity map. The inverse consistency error is therefore defined as the squared difference between this composition and the identity map. The voxel-wise cumulative inverse consistency error ($CICE_j(x)$) with respect to a template image j is given by:

$$CICE_j(x) = \frac{1}{M}\sum_{i=1}^{M}\big\|h_{ji}\big(h_{ij}(x)\big) - x\big\|^2 \tag{3.6}$$

where h_{ij} is again the transformation from image i to j and M the number of images in the evaluation population.

Transitivity error The transitivity metric evaluates the quality of a registration algorithm by checking how good the transitivity property of an image population is satisfied. This property is important for minimizing correspondence errors in the case of composed transformations. In the optimal case a transformation defining correspondence between three images should project a certain point from image A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A back to the initial position. The transitivity error for such a set of transformations is therefore defined as the squared error difference between the composition to the transformations and the identity map. The voxel-wise cumulative transitivity error ($CTE_k(x)$) with respect to a template image j is given by:

$$CTE_k(x) = \frac{1}{(M-1)(M-2)} \sum_{\substack{i=1 \\ i \neq k}}^{M} \sum_{\substack{j=1 \\ j \neq i \\ j \neq k}}^{M} \left\| h_{ki}\Big(h_{ij}\big(h_{jk}(x)\big)\Big) - x \right\|^2 \tag{3.7}$$

An additional metric used in Chapter 6 for validation of the featurelet algorithm is the *Hausdorff-distance* as described in [36]:

Hausdorff-distance The Hausdorff-distance $H(A, B)$ is a measure of mismatch between two structures A and B. Such a structure can be e.g. the delineation of an organ in a CT image. The error of a registration algorithm is quantified by measuring the distance of the point of structure A which is farthest from any point of structure B and vice versa. Given two finite point sets $A = a_1, ..., a_n$ and $B = b_1, ...b_m$ the Hausdorff-distance is defined as:

$$H(A, B) = \max\big(h(A, B), h(B, A)\big) \quad \text{with } h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\| \tag{3.8}$$

The function $h(A, B)$ is called the *direct Hausdorff-distance* from A to B.

4. Software Development Tools

„The medical image computing community has historically been fragmented in its approach to software development and distribution. A result of this fragmentation has been a significant amount of re-implementation of common tools like file I/O, filters, core algorithms, user interfaces, etc. and at the same time a lack of reproducibility of results due to the complexity of algorithms and variations in the implementations of these core components." [37]
The National Alliance for Medical Image Computing (NA-MIC) was formed specifically to address these issues. It is funded by the National Centers for Biomedical Computing (NCBC) program which is a part of the NIH[1] Roadmap Initiative. NA-MIC started operating in late 2004. Since that time it brings together several ongoing threads of development dealing with various medical image computing software problems. In addition NA-MIC provides the community with the needed open platform. This platform is called *NA-MIC Kit* and includes three major types of software technology: programming toolkits, end-user application software, and system infrastructure. The programming toolkits rely both on the Insight Toolkit (ITK) for image registration and segmentation and the Visualization Toolkit (VTK) to support visualization and interactive rendering as well as manipulation. The end-user application software of NA-MIC is 3D Slicer. It provides a cross-platform GUI built on VTK and ITK [37].

For the implementation of the featurelets-algorithm (Chapter 5) into the 3D Slicer platform (Section 4.1) both ITK (Section 4.2) and VTK (Section 4.3) are used in this thesis project. To create a custom GUI for the Slicer module QtDesigner (Section 4.4.1) is used.

An overview of available open-source software and software tools for biological and medical imaging is given in [38].

4.1. 3D Slicer

Slicer[2], or 3D Slicer, is a free, open source software package for visualization and image analysis. It allows exploration of the imaging datasets in two, three and four dimensions. 3D Slicer is natively designed to be available on multiple platforms,

[1] The National Institutes of Health (NIH) is the primary agency of the United States government responsible for biomedical and health-related research.

[2] http://www.slicer.org/

including Windows, Linux and Mac Os X.

Slicer was initiated as a masters thesis project of David Gering between the Surgical Planning Laboratory (SPL) at the Brigham and Women's Hospital (BWH) and the MIT Artificial Intelligence Laboratory in 1998. Afterwards, Steve Pieper took over the role of the Chief Architect and started the work of transforming 3D Slicer into an industrial-strength package. Since 1999 Slicer has been under continuous development at the SPL under the leadership of Ron Kikinis. Today it is developed mostly by professional engineers and computer scientists, with the participation of Isomics Inc., Kitware Inc. and GE Global Research. At the same time numerous groups and individual users participate in the software development process. The 3D Slicer community is continuously improving the code by reporting software problems and contributing solutions, suggesting new features and developing new tools [39]. Altogether 3D Slicer consists nowadays of more than 1.8M lines of code, mostly C++ (68.5 %). Further languages are C (12.7 %), Python (5.7 %), XML (4.3 %), CMake (4.1 %) and Tcl (3.6 %) as well as 12 other languages[3].

This massive software development effort has been enabled by the participation of several large scale NIH funded efforts, including the NA-MIC[4], NAC[5], BIRN[6], CIMIT[7] and NCIGT[8] communities.

The Software is designed for research purposes only. It has not been reviewed or approved by the Food and Drug Administration (FDA) or by any other agency.
Regarding derivative works the licensing model of Slicer does not place any restrictions on the use of its source code. This has the aim to include both academic and industry partners to the user community. In addition, it simplifies the transition of a pure research tool into a commercial product.
Thus, a method can potentially be developed into an FDA-cleared clinical device after optimization and validation [39].

Slicer provides a powerful GUI to interact with the data. Amongst numerous possible applications there is manual segmentation and the creation of 3D surface models from conventional MRI images or deformable image registration. Slicer has also been used to incorporate models of the neurovascular bundle using image segmentation in MRI-guided prostate interventions. According to the homepage, Slicer is also frequently used in image-guided therapy research and is growing up alongside that field. The extension *SlicerRT* has the goal of making 3D Slicer a powerful radiotherapy research platform.

[3] Figures taken from https://www.openhub.net/p/slicer (June 2015)
[4] National Alliance for Medical Image Computing - http://www.na-mic.org/
[5] Neuroimage Analysis Center - http://nac.spl.harvard.edu/
[6] Biomedical Informatics Research Network - http://www.birncommunity.org/
[7] Center for Integration of Medicine and Innovative Technology - http://www.cimit.org/
[8] National Center for Image-Guided Therapy - http://www.ncigt.org/

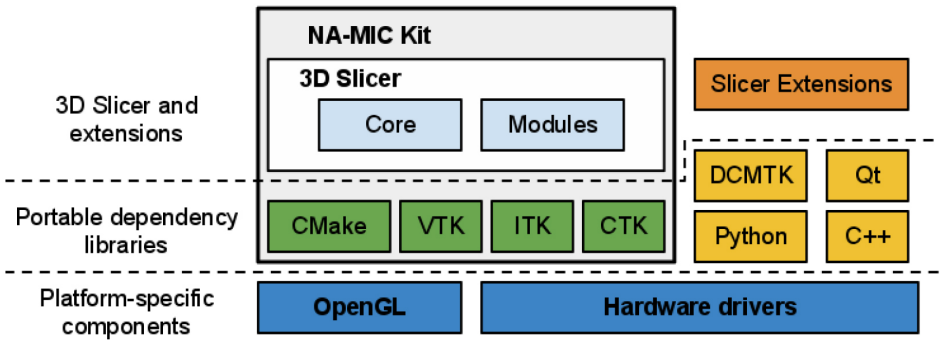

Figure 4.1.: *Slicer is a cross-platform package. However, certain requirements on the graphics system, such as OpenGL drivers, should be met to support accelerated rendering. The dependency libraries are portable across platforms and are distributed under compatible licenses. Slicer itself consists of the main application framework (*core*) and plugins (*modules*). Custom functionality is introduced by implementing external modules (*Slicer extensions*) [39].*

4.1.1. Slicer Architecture

The architecture of 3D Slicer follows a modular and layered approach, as shown in Figure 4.1. At the lower level of the architecture there are the fundamental libraries provided by the operating system. These are not packaged with Slicer. At the level above, there are the programming languages and libraries that provide higher level functionality and abstractions.
3D Slicer integrates a number of powerful open source projects into an end-user application. Some of the libraries used by the application are:

- Qt - a cross-platform GUI framework → Section 4.4
- DICOM Toolkit (DCMTK) - it implements parts of DICOM standard and is used to interact with DICOM data
- CMake - enables cross-platform build system configuration, packaging and testing of 3D Slicer and NA-MIC Kit libraries
- The Visualization Toolkit (VTK → Section 4.3) - it provides the key building blocks for 3D computer graphics and visualization
- The Insight Toolkit (ITK → Section 4.2) - it is a library developed specifically for the tasks related to medical image registration and segmentation, and for implementing new image analysis algorithms
- The Common Toolkit (CTK)[9] - it is a biomedical image computing library with the focus on application-level DICOM support, plugin framework and specialized GUI widgets.

[9] http://www.commontk.org/

All of these dependency libraries of 3D Slicer are cross-platform portable and are distributed under licenses fully compatible with Slicer, which do not restrict their use in either commercial or open source products [39].

3D Slicer itself consists of the lean application core, Slicer Modules, and Slicer Extensions. The core implements the Slicer user interface, provides support for data input/output (IO), visualization and developer interfaces that support extension of the application with new plugins.
The classes implementing the 3D Slicer core are organized into three main groups based on their functionality [39]:

The Model is responsible for data organization and serialization. It is supported by the XML-based Medical Reality Markup Language (MRML). MRML defines the hierarchies of the data elements, and the application programming interfaces (APIs) for accessing and serializing the individual nodes. The MRML library is a key component of the Slicer architecture, but does not have any dependencies on the other core components. It is used by the Slicer core and plugins to instantiate the MRML nodes and organize them into a coherent internal data structure called the *MRML Scene*. The MRML scene maintains the links between the individual data items, their visualization and any other persistent state of the application and modules.

The View maintains the state of the visual elements of the application. Its functionality is provided by the GUI and displayable manager classes of the Slicer core. These classes take care of consistency between the internal MRML state (Model), and the visual appearance of the Qt GUI data viewers of the application.

The Controller or Logic component encapsulates the processing functionality of the application core. The logic does not depend on the existence of GUI, but is fully aware of the MRML data structures. Communication between the view and controller components is realized indirectly through changes of the MRML data structures. The logic classes use the MRML nodes for storing the computation results. The view classes can register to receive event updates from the MRML scene and the individual nodes, which in turn initiate updates of the visualization elements.

4.1.2. Slicer Modules

To implement new functionality into the software, modules are used. Slicer modules are the plugins that depend on the Slicer core. A module can be independent, or it can rely on other modules.
Slicer supports three different types of modules. While the developer has to choose

between one of them to implement its module, the end user won't notice the difference as they all share the same look and feel. The choice for a given type of module is usually based on the type of inputs/parameters of that module.

Command Line Interface (CLI) are standalone executables with a limited input-output arguments complexity (simple argument types, no user interactions). They are typically implemented using ITK (→ Section 4.2)

Loadable Modules are C++ plugins that are built against Slicer. They define custom GUIs for their specific behavior and have full control over the application; they are optimized for heavy computations.

Scripted Modules are written in Python. They have full access to the API; recommended for fast prototyping and custom workflow development.

Unlike Slicer modules that are packaged as part of the Slicer distribution, Slicer extensions are external plugins. They can be installed „on demand" by the user. The extensions mechanism enables sharing of tools based on Slicer that can not be included into the package due to incompatible licenses or other constraints [39].

4.2. ITK

The Insight Segmentation and Registration Toolkit (ITK)[10] is an open-source, cross-platform system that provides developers with an extensive suite of software tools for image analysis. ITK provides leading-edge algorithms for registering and segmenting multidimensional data.
The toolkit does not address visualization or graphical user interface. These are left to other toolkits e.g. VTK, ParaView or Slicer.

ITK is implemented in C++. It is using the CMake build environment to manage the configuration process. In addition, an automated wrapping process generates interfaces between C++ and interpreted programming languages such as Java and Python. This enables developers to create software using a variety of programming languages. ITK's C++ implementation style is referred to as generic programming (i.e., using templated code). Such C++ templating means that the code is highly efficient, and that many software problems are discovered at compile-time, rather than at run-time during program execution. It also enables ITK to work on two, three, four or more dimensions.

The toolkit is organized around a data-flow architecture. That is, data is represented using data objects which are in turn processed by process objects (filters). Data objects and process objects are connected together into pipelines. Pipelines

[10] http://www.itk.org/

are capable of processing the data in pieces according to a user-specified memory limit set on the pipeline. Object factories are used to instantiate objects. Factories allow run-time extension of the system. A command/observer design pattern is used for event processing.

The memory model depends on *smart pointers* that maintain a reference count to objects. Smart pointers can be allocated on the stack, and when scope is exited, the smart pointers disappear and decrement their reference count to the object that they refer to [40].

4.3. VTK

The Visualization Toolkit (VTK)[11] is an open-source, freely available software system for 3D computer graphics, image processing, and visualization. It consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python. VTK was created by the team of Kitware[12] which also continues to extend the toolkit.

VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods, as well as advanced modeling techniques such as implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. VTK has an extensive information visualization framework and a suite of 3D interaction widgets. The toolkit supports parallel processing and integrates with various GUI toolkits such as Qt and Tk. VTK is cross-platform and runs on Linux, Windows, Mac, and Unix platforms [41, 42].

4.4. Qt

Qt[13] („cute") is a cross-platform application and UI framework used for application development. It supports deployment to over a dozen leading platforms.
The Qt framework comprises of modular cross-platform C++ class Qt libraries and an Integrated Development Environment (IDE) with *Qt Creator* as well as User Interface (UI) designer tools.

4.4.1. Qt Designer

Qt Designer[14] is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. All properties set in Qt Designer can be changed dy-

[11] http://www.vtk.org/
[12] http://www.kitware.com/
[13] http://www.qt.io/
[14] https://doc.qt.io/qt-5/qtdesigner-manual.html

namically within the code. Widgets and forms created with Qt Designer integrate seamlessly with programmed code, using Qt's signals and slots mechanism.

Signals and Slots

Signals and slots[15] are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks.
In GUI programming, when one widget changes, often another widget is wanted to be notified. For example, if a user clicks a `Close button` the window's `close()` function should be called.
Other toolkits achieve this kind of communication using callbacks. A callback is a pointer to a function. While successful frameworks using this method do exist, callbacks can be unintuitive and may suffer from problems in ensuring the type-correctness of callback arguments. More generally, objects of any kind should be able to communicate with one another.
In Qt a signal is emitted when a particular event occurs. A slot is a function that is called in response to a particular signal. Qt's widgets have many predefined signals as well as slots. Widgets can always be subclassed to add customized signals and slots to them. The signals and slots mechanism is type safe: The signature of a signal must match the signature of the receiving slot. In fact a slot may have a shorter signature than the signal it receives because it can ignore extra arguments. Signals and slots are loosely coupled: A class which emits a signal neither knows nor cares which slots receive the signal.

4.4.2. Qt Creator

Qt Creator[16] provides a cross-platform, complete integrated development environment (IDE) for application developers to create applications for multiple desktop and mobile device platforms. It is designed to make development with the Qt application framework faster and easier.

[15] https://doc.qt.io/qt-5/signalsandslots.html
[16] http://www.qt.io/ide/

5. Software Development - *Featurelet-Registration*

The main aim of this work was to develop a graphical user interface (GUI) for the already existing in-house-developed DIR method of *Fabri* [14] to facilitate user interaction and to allow for further investigation of cases with unsatisfying registration results. The software should enable direct visualization of the output and quick adjustment of registration parameters. Thus, we decided to create an extension for the open source software platform 3D Slicer. Our extension consists of a loadable module called *Featurelet-Registration*.

5.1. Basic Design of the Featurelet Algorithm

The registration logic of our 3D Slicer extension is an implementation of the piecewise non rigid registration algorithm of *Fabri* [14]. It is based on the approach of *Söhn et al.* [13] as presented in Section 3.3. The basic design of the algorithm is illustrated in Figure 5.1. It is written in C++ and follows the subsequent steps:

- Open fixed and moving image: reading of volumetric DICOM images from a file is implemented using the ITK class `itk::ImageFileReader`.
- Create subvolumes: subvolumes are featurelets of a certain size A in the moving image and search regions of a certain size B in the fixed image; they are created as temporal images.
- Rigid registration of every featurelet of the moving image to the corresponding search region of the fixed image. For the whole registration process functions and objects from the ITK library are used:
  - Transform: `itk::TranslationTransform`;
  - Metric: `itk::NormalizedCorrelationImageToImageMetric`;
  - Optimizer: `itk::RegularStepGradientDescentOptimizer`;
  - Interpolator: `itk::LinearInterpolateImageFunction`.
- Save the transformation in a vector volume; for storing the output of the registration process the class `FeatureletRegistrationResult` was created; the main parameters are:
  - offset given by the transform vector,

  - final value of the metric,
  - size and index of search region,
  - size and index of featurelet and
  - number of featurelets.

- Create three 1D volumes for every direction of the vector volume: this is done by the member function `GenerateDeformationField()` of the class `DeformationFieldGenerator` which processes the deformation field; all the following steps are handled by member functions of this class.

- Interpolate the center of every featurelet to an actual voxel of the image: done by the function `RescaleDeformationField()`; this function uses the ITK class `itk::ResampleImageFilter`.

- Create and save a 3D vector image: the interpolated transformation vectors of x, y and z direction are brought together to obtain the total deformation field by the function `TotalDeformationFieldCreator()`; it is written to a file by the ITK class `itk::ImageFileWriter`.

- Create and save a warped image: the deformation field is applied to the moving image by the function `WarpImagebyDeformationField()` to obtain a deformed image for visual inspection; it is written to a file by using again the ITK class `itk::ImageFileWriter`.

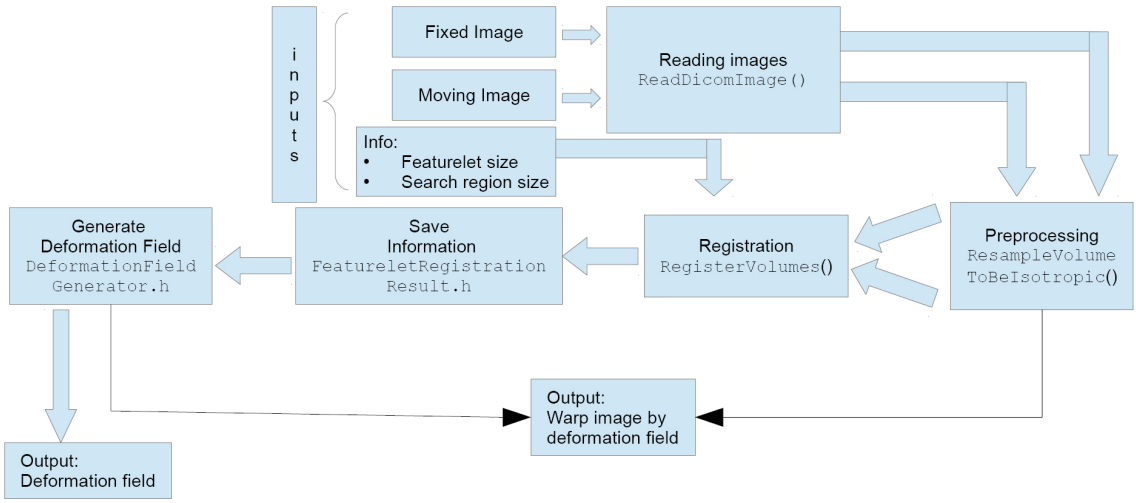

Figure 5.1.: *Flowchart of featurelet algorithm (picture from [14]).*

5.2. Featurelet-Registration

The software platform 3D Slicer (see Section 4.1) offers the possibility to introduce new functionality with its extension mechanism. As an initial step the Slicer application itself has to be build in `Release` mode. Then, an extension can be developed.

5.2.1. Slicer Extension

An extension is a „delivery package" bundling together one or more Slicer modules. Such a module typically consists of several files of various types, those can be CMake files, source files, and resource files. Usually, the names of the files and the names of text strings within a file are related and need to match. To simplify the process of creating and contributing an extension 3D Slicer offers a tool called „Extension Wizard". It provides a mechanism to create extensions and modules from templates. The following line is used to run the wizard in a terminal program:

Code 5.1 # Invoke the Wizard

```
/path/to/slicer/bin/slicerExtensionWizard
```

The launcher script ensures that Python- and library-paths are set correctly to find packages and libraries provided by Slicer, and it invokes the correct Python binary if provided by Slicer. To create the folder structure for the extension called „Slicer-Featurelets" comprising one C++ module called „Featurelet-Registration" the following commands are needed:

Code 5.2 # Create an extension with a module written in C++

```
slicerExtensionWizard --create Slicer-Featurelets ~/code/
cd ~/code/Slicer-Featurelets
slicerExtensionWizard --addModule loadable:Featurelet-Registration
```

This process automatically creates files for the extension Slicer-Featurelets and its module Featurelet-Registration with appropriate names, and makes some crucial content substitutions within the templates in order to produce code that can be built immediately. Our registration module was chosen to be a loadable module, as they are optimized for heavy computations, define custom GUIs for their specific behavior and have full control over the application.

5.2.2. Development Environment

For the development of the 3D Slicer extension the following environment was used:

- Hardware:

  - Processor: 2x Intel Xeon CPU E5345 2.33 GHz
  - RAM: 10GB DDR2
  - Graphics card: NVIDIA GeForce 8800 GTS

- Software:
  - Operating System: Ubuntu 14.04 LTS
  - Development platform: Qt Creator 3.0.1
  - Compiler: GCC 4.8.2
  - Tool kits: Qt 4.8.6, Insight Toolkit 4.6.0
  - 3D Slicer Version: 4.4
  - Build system: CMake 2.8

5.2.3. Structure of the Source Code

Within the folder of the extension Slicer-Featurelets is the build-directory of the whole extension, a folder for the module Featurelets-Registration and the CMakeList for the extension. Within the module-directory there are three important folders:

Logic Contains the source code of the registration logic; is described in more detail in Section 5.2.4.

MRML Contains the `.cxx` and `.h`-file of the `vtkMRMLRegistrationNode` which defines a MRML-Node for the Featurelets-Registration module.

Resources Contains the resources for the custom GUI of the module and consists of the file `qSlicerRegistrationModule.qrc` which stores information about the icon of the module, and two folders:

- Icons: contains a `.png`-file which is used as icon of the module;
- UI: contains `.ui`-files which define custom widgets for the GUI; these files are written in `xml` and can be edited with the Qt Designer (see Section 4.4.1).

In addition, the module-directory contains the following files:

- `.cxx` and `.h`-file `qSlicerRegistrationModule` which defines the module and its dependencies as well as the category of the module, and is responsible for loading the logic; helptext, acknowledgment and contributors can be edited in the `.cxx`-file.
- `.cxx` and `.h`-file `qSlicerRegistrationModuleWidget` for the custom widget of the GUI; here signals and slots can be defined.
- CMakeList for the module.

The main difference between the initial code of Fabri presented in Section 5.1 and the code used in the 3D Slicer extension is the handling of input/output and of the images itself. The initial code reads DICOM-images from a file and uses a class called `regImage` for image handling. The results of the registration process are written to a file. For the extension, reading from and writing to a file is done by 3D Slicer. Datasets are stored in so called „MRML-nodes". For the Featurelet-Registration module data is stored in the node defined by `vtkMRMLRegistrationNode`. Beside the information of the volumes selected as fixed and moving image, the node contains the values of all GUI widgets. This can be the integer value of a slider for the size of a featurelet as well as the boolean value of a radio button. Communication of changes in the GUI and the `vtkMRMLRegistrationNode` is defined in the file `qSlicerRegistrationModuleWidget`.

5.2.4. Registration Logic

Beside the CMakeList the folder for the source code of the registration logic contains the following files:

- `vtkSlicerRegistrationLogic`: is the main class containing a loop that runs through the whole grid of featurelets as well as the connection of logic and GUI.
- `FeatureletRegistrationResult`: a class for storing the information of the registration of each featurelet.
- `DeformationFieldGenerator`: a class for generating the final deformation field out of all the featurelet registration results.
- `CommandIterationUpdate`: a class for monitoring the evolution of the optimization process of the registration.

vtkSlicerRegistrationLogic

The Featurelet-Registration module uses pixel type short and an image dimension of three. Both are defined in the header file of the class `vtkSlicerRegistrationLogic`. Beside the connection to the GUI the class comprises the following member functions:

- `RunClicked`: is the „main" function of the registration logic; it is called when the button „Run Registration" of the GUI is clicked. All needed parameters from the GUI are loaded from the `vtkMRMLRegistrationNode`. After error handling fixed and moving image are converted to ITK images by calling the member function `ConvertVolumeNodeToItkImage`. Then the result storage for the registration process is initialized and the following loop over every featurelet grid position is used:

Code 5.3 Loop over every featurelet position

```
for(unsigned int i=0;i<imageSizeFixed[0];i=i+FeatureletSize[0]){
  iFeatureletGridPos[0]++;
  iFeatureletGridPos[1]=0;
  for(unsigned int j=0;j<imageSizeFixed[1];j=j+FeatureletSize[1]){
    iFeatureletGridPos[1]++;
    iFeatureletGridPos[2]=0;
    for(unsigned int k=0;k<imageSizeFixed[2];k=k+FeatureletSize[2]){
      this->UpdateFromMRMLScene();
      iFeatureletGridPos[2]++;
      iCurrentFeaturelet++;
      ImageIndex[0] = i;
      ImageIndex[1] = j;
      ImageIndex[2] = k;
      TempSize = FeatureletSize;     //for featurelets on the edges
      if (ImageIndex[0] + FeatureletSize[0] > imageSizeFixed[0])
        TempSize[0] = imageSizeFixed[0] - ImageIndex[0];
      if (ImageIndex[1] + FeatureletSize[1] > imageSizeFixed[1])
        TempSize[1] = imageSizeFixed[1] - ImageIndex[1];
      if (ImageIndex[2] + FeatureletSize[2] > imageSizeFixed[2])
        TempSize[2] = imageSizeFixed[2] - ImageIndex[2];

      SubsampleVolume(fixedImageItk, TempSize, SearchRegionSize,
                      ImageIndex, fixed=true);
      SubsampleVolume(movingImageItk,TempSize, SearchRegionSize,
                      ImageIndex, fixed=false);
      statusMoving=CheckFeaturelet(fixed=false,UseFiducialPoints,
                                   fiducialPoints);
      statusFixed =CheckFeaturelet(fixed=true, UseFiducialPoints,
                                   fiducialPoints);

      if(statusFixed==Featurelet::OK &&
         statusMoving==Featurelet::OK){
        if((!correl)&&(linear)) {
          regResult = RegisterVolumesI(debugMode, rigid);
          regis++;
        }
        if((correl)&&(!linear)) {
          regResult = RegisterVolumes1(fixedImageItk,
                                movingImageItk, debugMode, rigid);
          regis++;
        }
        if((!correl)&&(!linear)) {
          regResult = RegisterVolumesII(debugMode, rigid);
```

```
          regis++;
        }
        if((correl)&&(linear)) {
          regResult = RegisterVolumes2(fixedImageItk,
                                movingImageItk, debugMode, rigid);
          regis++;
        }
      }
      else if( statusFixed==Featurelet::fixedFeaturelet ||
               statusMoving==Featurelet::fixedFeaturelet ) {
        fixedreg++;
        regResult = FeatureletRegistrationResultType::New();
      }
      else {
        noreg++;
        regResult = FeatureletRegistrationResultType::New();
      }

      regResult->SetStatusFixed(statusFixed);
      regResult->SetStatusMoving(statusMoving);
      regResult->SetFeatureletIndex(ImageIndex[0],ImageIndex[1],
                                    ImageIndex[2]);
      regResult->SetFeatureletSize(TempSize[0],TempSize[1],
                                   TempSize[2]);
      regResult->SetFeatureletGridPosition(iFeatureletGridPos[0],
                   iFeatureletGridPos[1],iFeatureletGridPos[2]);
      regResult->SetImageSizeM(iSize[0], iSize[1], iSize[2]);
      regResult->SetFeatureletNumber(iCurrentFeaturelet);
      regResults[iCurrentFeaturelet] = regResult;
      regPointer = regResult.GetPointer();
      resultStorage.AddFeaturelet(regPointer);
    }
  }
  progress = (i*100)/imageSizeFixed[0];
  pnode->Setprogress(progress);
  this->UpdateFromMRMLScene();
}
```

Subsequently the total deformation field is created by calling the member functions of the class `DeformationFieldGenerator` and the results are passed on to corresponding MRML-nodes.

- `ShowVolume`: to show the selected volume if the corresponding button is pressed.
- `ConvertVolumeNodeToItkImage`: takes the volume of the selected node and converts it to an ITK image for further processing; this function is taken from the Slicer-RT-commons.
- `ResampleVolumesToBeIsotropic`
- `SubsampleVolume`: takes the input image and divides it into small subvolumes; the subvolumes are featurelets for the moving image and the corresponding search regions on the fixed image.
- `CheckFeaturelet`: checks a featurelet and returns its `status`
- `RegisterVolumes*`: performs the actual registration; the user can select between normalized cross-correlation (NCC) and mutual information (MI) as metric and between linear and nearest neighbor (NN) interpolation. Thus, there are four different functions for every possible case:
  - `RegisterVolumes1`: for NCC and linear interpolation,
  - `RegisterVolumes2`: for NCC and NN interpolation,
  - `RegisterVolumesI`: for MI and NN interpolation and
  - `RegisterVolumesII`: for MI and linear interpolation.

  For both metric and interpolator classes which are already defined in ITK are used:
  - `NormalizedCorrelationImageToImageMetric` for NCC,
  - `MutualInformationImageToImageMetric` for MI,
  - `LinearInterpolateImageFunction` for linear interpolation and
  - `NearestNeighborInterpolateImageFunction` for NN interpolation.

FeatureletRegistrationResult

The class `FeatureletRegistrationResult` is taken almost unchanged from the code of Fabri. It stores the following five important parameters after every step in the featurelet grid:

- The number of the current featurelet,
- size and index of both featurelet and its corresponding search region,
- status of both featurelet and its corresponding search region,
- the final value of the metric and

- the transformation vector of the featurelet.

In addition the `namespace Featurelet` is defined in this class. It is used to store the `status` of a featurelet or a search region. In comparison to the initial code of Fabri the added `status fixedFeaturelet` is available. It is used in case a fixed point is located within a featurelet or a search region.

Code 5.4 Definition of the `namespace Featurelet`

```
namespace Featurelet {
  enum Status {
    OK = 0,
    SizeError = 1,
    sameColor = 2,
    internalError = 3,
    fixedFeaturelet = 4
  };
}
```

DeformationFieldGenerator

The class `DeformationFieldGenerator` comprises five important member functions:

- `GenerateDeformationField()`: as in the code of Fabri, this function creates three 1D volumes for every direction of the vector volume out of the registration result ; the source code can be found in the Appendix A.1.
- `RescaleDeformationField()`: this function interpolates the center of each featurelet after its registration to an actual voxel position of the fixed image; the source code can be found in the Appendix A.2.
- `TotalDeformationFieldCreator()`: creates a 3D vector volume with the same size as the fixed image; the deformation at the edge of the deformation field is set to zero; within the 3D volume the interpolated transformation vectors of x, y and z direction are brought together to obtain the total deformation field; the source code can be found in the Appendix A.3.
- `VTKDeformationFieldCreator()`: this function is needed to transform the deformation field from the coordinate system used by ITK to the one used by VTK for visualization in 3D Slicer; the source code can be found in the Appendix A.4.
- `WarpImagebyDeformationField()`: to create a warped image for visual inspection of the registration result the ITK class `itk::WarpImageFilter` is used ; the source code can be found in the Appendix A.5.

CommandIterationUpdate

Uses the class `itk::Command` as observer method. In `debugMode` the observer outputs the number of the current iteration of the optimizer as well as the current value of the metric and its current position. Thus, the user gets information about the optimization process of each featurelet. This can be useful in the case of a misregistration.

5.2.5. GUI of Featurelet-Registration Module

The GUI of the 3D Slicer module Featurelet-Registration is defined in the folder `Resources`. The module-widget consists of four sections, as shown in Figure 5.2:

Help & Acknowledgment The Help-section provides the user with an exemplary workflow for image registration using the module Featurelet-Registration and information about other Slicer modules for pre- and post-processing of images and the registration output. The Acknowledgment-section provides information about the basic idea used for the module and its contributors.

Input In the Input-section the user can select the fixed and moving image intended to use for registration. Both images can be visualized immediately using the particular „Show"-button. In addition, the user can define fixed points in the image. A featurelet containing such a point is non moving.

Parameters In the Parameter-section both feturelet and search region size can be defined. Optionally a different size in the z-direction can be set for the case of large inter-slice spacings of an image. Regarding the registration process itself the user can select the type of interpolator (linear or NN) and the metric used for measuring the voxel-based similarity (MI or NCC). As advanced parameters optimizer settings can be changed and the registration process can be executed in debug mode. In this debug mode additional information about the registration process and its progress is shown in the terminal.

Output In the Output-section the user has to define the volume used for the deformed image as well as the one for the deformation field. By hitting the „Run Registration"-button the registration process starts and the progress is shown in the progress-bar below the button.

To facilitate usability several `tooltips` provide additional information for the user about the item being hovered over.

5.2.6. Workflow using *Featurelet-Registration*

After loading the required images in 3D Slicer there are several built-in modules available for pre- and post-processing as well as for evaluation of registration results.

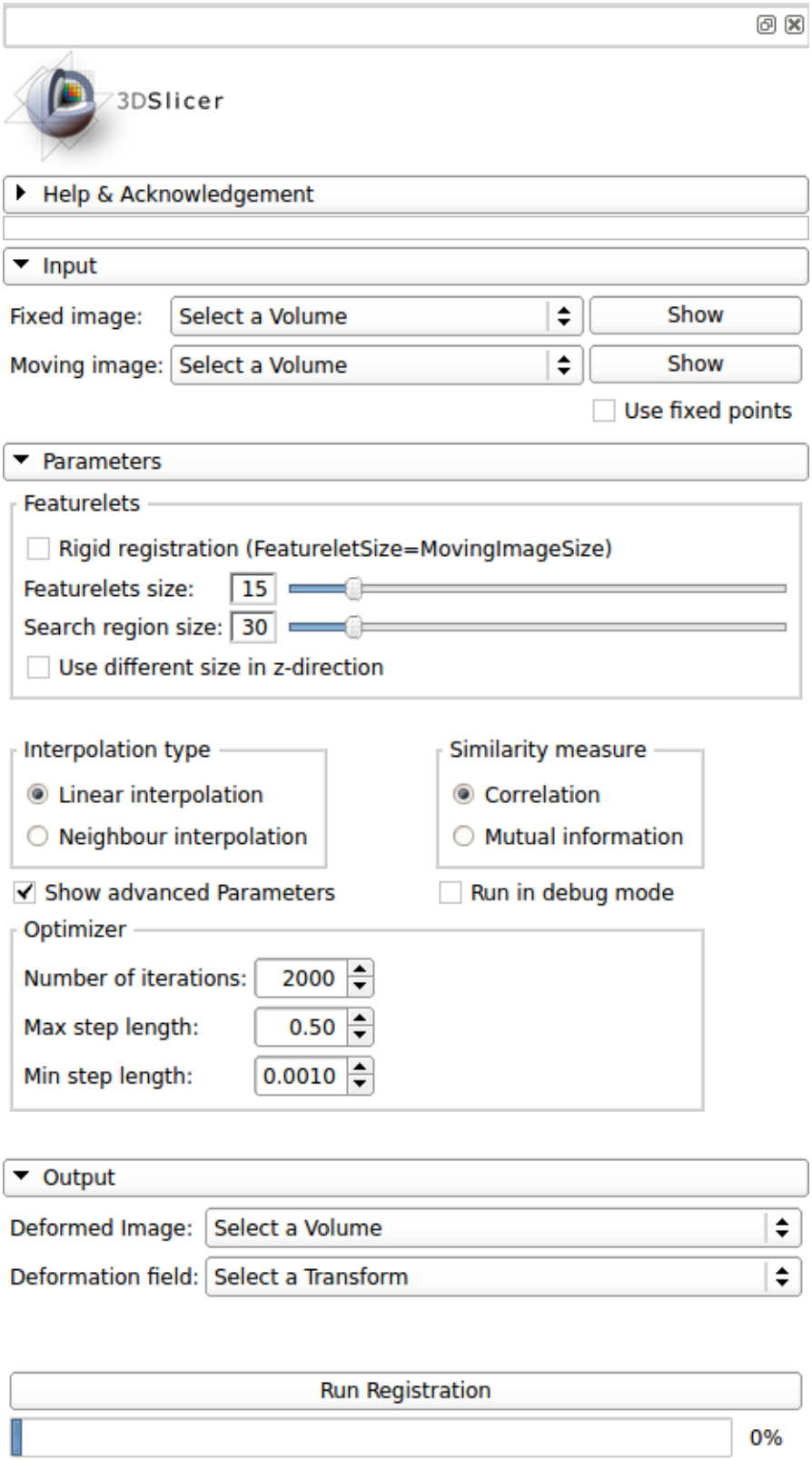

Figure 5.2.: *Custom GUI of the 3D Slicer module Featurelet-Registration.*

A typical workflow using the Featurelet-Registration module consists of the following steps:

- In the module `Volumes` in section `Volume Information` the user can get basic information about the active dataset.
- If the pixel type is different to `short` the module `Cast Scalar Volume` can be used to change the pixel type to `short`.
- In the module `Threshold Scalar Volume` the user can set image values to a certain value if they are below, above, or between a user-specified threshold value.
- For initial alignment of the fixed and the moving image the module `Expert Automated Registration` can be used. It provides rigid, affine and BSpline registration methods.
- To extract a subvolume of an image the module `Crop Volume` can be used. In this module the user can set a region of interest (ROI) and apply it on both the fixed and the moving image or just on one of them.
- After these pre-processing steps the registration can be performed using the module Featurelet-Registration.
- For visualization of the output deformation field (an exampel is given in Figure 6.8 in the following Chapter) 3D Slicer offers the module `Transforms`. It can also be used to apply the deformation field on other datasets such as contours or labelmaps.
- For further steps the extension `SlicerRT` can be used. It aims to make 3D Slicer a powerful radiotherapy (RT) research platform. It offers several tools for the field of RT such as contour and dose comparison and computation of dose volume histograms.

6. Validation of the Featurelet Algorithm

6.1. Prior Validation Studies

The *Featurelet-Registration* module of Slicer is an implementation of an existing algorithm which was already evaluated in previous studies.

6.1.1. Validation Studies of *Söhn et al.*

For validation studies of the basic concept of the featurelet algorithm described in Section 3.3 from *Söhn et al.* [13] lung datasets were used. Additionally, visual inspection was performed on head-and-neck datasets.

The registration quality was assessed:

- qualitatively by visually checking magenta-green overlay of the registered data sets (results are presented in Figure 6.1) and
- quantitatively by
  - a virtual thorax phantom and
  - Landmark-based evaluation on respiratory-correlated CT (RCCT) patient test cases.

Virtual Thorax Phantom

For the first quantitative test, a simulation of the exhale state of a lung RCCT patient was used. To create the virtual thorax phantom the deformation field of a free-form deformable registration algorithm as described in *Zhang et al.* [43] was used. Notice that for this step registration accuracy is not important as long as the output deformation field roughly mimics the physiological motion of the original lung dataset.

Results: The result of a voxelwise evaluation is illustrated in Figure 6.2. Mean and standard deviation of the displacements of the virtual thorax phantom were 2.9 ± 2.8 mm. After deformable registration of the phantom i.e. of the inhale deformed to exhale CT as moving image and the original inhale CT as fixed image, the 3D residuals were reduced to 1.1 ± 1.2 mm.

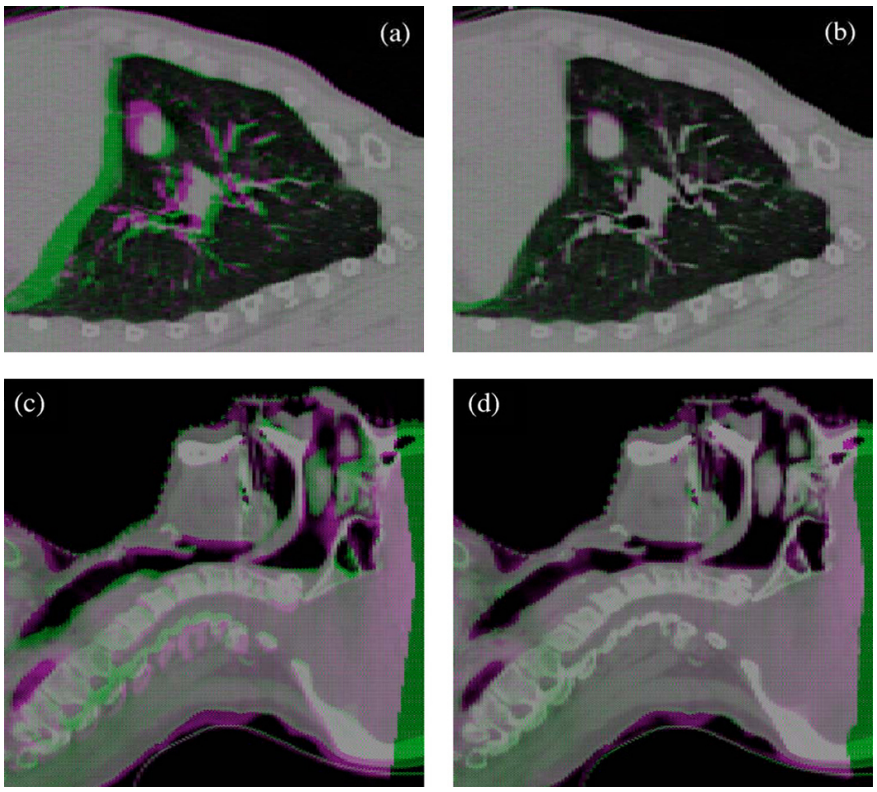

Figure 6.1.: *Visual inspection: the upper part shows an overlay of an exhale and an inhale CT. (a) shows the overlay before registration. In (b) the inhale CT is deformed to the exhale CT. The overall registration quality appears satisfactory, except for a region around the posterior part of the diaphragm. There, some misregistrations occur because of rotations on a scale smaller than the used featurelet size.*
The lower part shows an overlay of a CT and a CBCT of head-and-neck. (c) is before and (d) after the registration. In the area of the upper spine and the nasal cavities a remarkable reduction of the residuals could be achieved through featurelet registration [13].

Landmark-based Evaluation

For the second quantitative test, bronchial bifurcations were deployed as anatomical landmarks. The lung RCCT datasets of four patients were marked by a physician in both inhale and exhale state. The landmarks were distributed regularly over the lung volume. 11 to 15 homologous points were chosen per patient by the physician. The corresponding displacements were calculated. Then these displacements were compared with the outcome of the deformable registration.

Results: The result of an averaging over all 55 landmarks of all patients yielded considerably smaller 3D residuals than the breathing motion itself. Mean and standard deviation of the 3D displacements of the breathing motion were 7.8 ± 5.1 mm with a maximum of 21.3 mm. After deformable registration the 3D residuals were reduced to 1.6 ± 1.0 mm with a maximum of 4.6 mm.

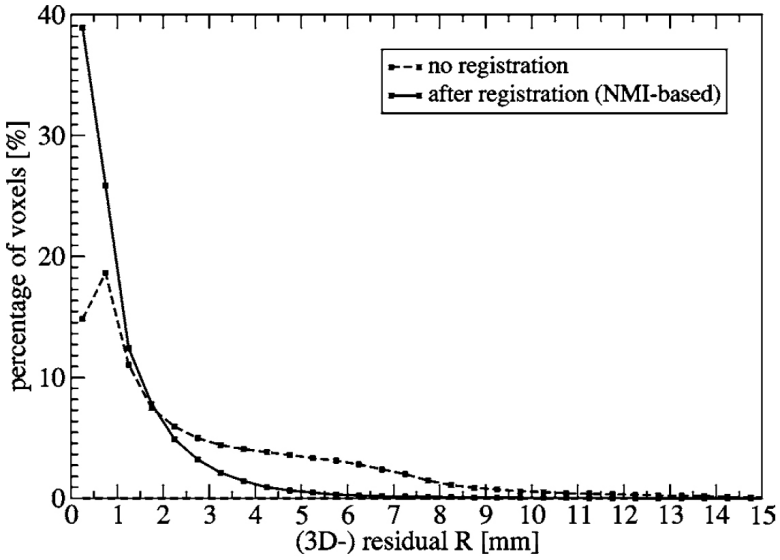

Figure 6.2.: *Virtual thorax phantom: comparison of the 3D displacement of the simulated deformation field (dotted line) with the 3D residuals after deformable registration with the featurelet algorithm (continuous line) [13].*

6.1.2. Validation Studies of *Fabri et al.*

Validation studies of the algorithm that is the basis for the Slicer implementation were performed by [14, 15, 16]. Two phantoms and 29 clinical datasets were used.

Phantom I

A deformable box-shaped phantom consisting of four rigid and two flexible faces was built. The box was filled with nine liters of gelatin containing 24 randomly distributed spherical Teflon markers with 10 mm diameter. Three series of CBCT images of the phantom were taken. The first one without deformation and the other two with different weights added to the flexible faces to create two different deformations. The total displacement of the 24 Teflon markers was calculated using 3D Slicer. For the image registration three different algorithms (rigid registration, B-spline registration implemented in 3D Slicer, and the featurelet algorithm) were used and the results compared.

Results: The results of both deformations are illustrated in Figure 6.3. Mean and standard deviation of the 3D displacements and maximum displacement of the first deformation and the corresponding 3D residuals of all three algorithms were:

- Deformation: 4.57 ± 1.25 mm (max 7.83 mm)
- Rigid registration: 2.78 ± 1.58 mm (max 7.15 mm)

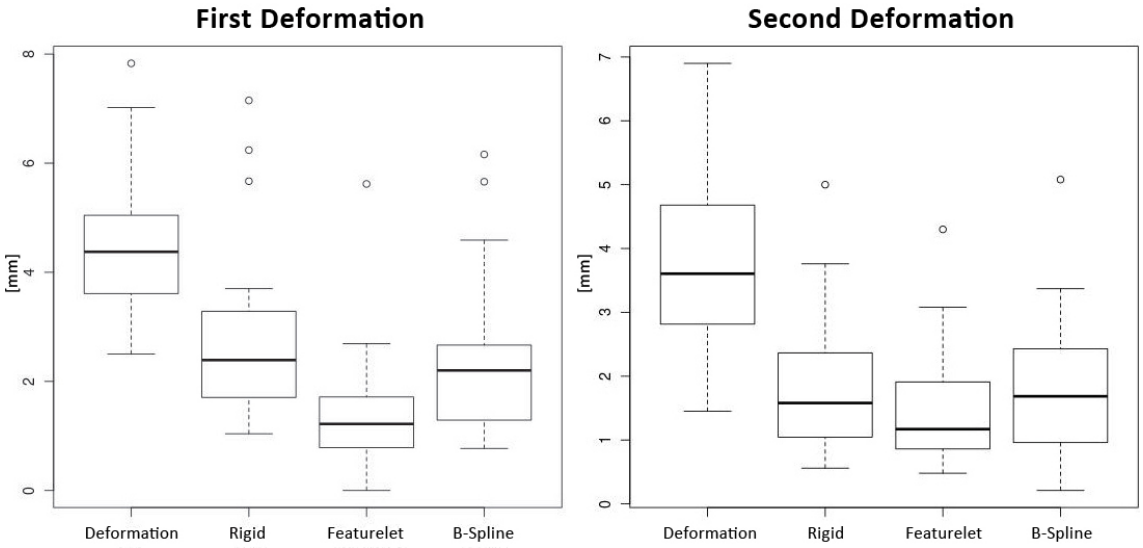

Figure 6.3.: *Phantom I: comparison of 3D displacements of the deformation and 3D residuals of the three different registration algorithms applied. The left side shows the results of the first deformation and the right side of the second deformation (picture from [14]).*

- B-spline registration: 2.43 ± 1.46 mm (max 6.16 mm)
- Featurelet registration: 1.40 ± 1.15 mm (max 5.62 mm)

Mean and standard deviation of the 3D displacements and maximum displacement of the second deformation and the corresponding 3D residuals of all three registration algorithms were:

- Deformation: 3.84 ± 1.41 mm (max 6.90 mm)
- Rigid registration: 1.86 ± 1.12 mm (max 5.00 mm)
- B-spline registration: 1.84 ± 1.13 mm (max 5.08 mm)
- Featurelet registration: 1.43 ± 0.91 mm (max 4.30 mm)

Phantom II

A deformable pelvis-shaped phantom with substructures representing bladder, prostate, colon and intestines as well as spine and pelvis was built within a cubic plastic box and used for simulation of intestine and bladder motion. The deformation was achieved by variation of the volumina of two balloons inside the box (compare Figure 6.4). More details about *Phantom II* can be found in [15]. Three series of CT images of the phantom were taken. For every image the filling state of the balloons as well as the position of the substructures was changed to get an internal deformation. Then, another three CBCT images were taken, where the relative

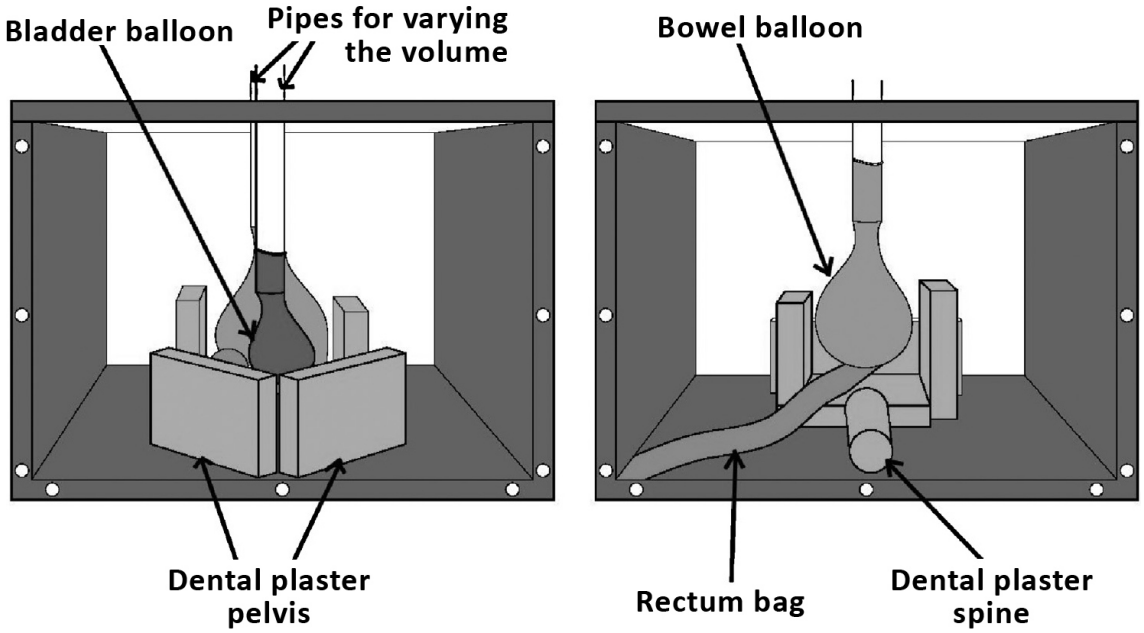

Figure 6.4.: *Phantom II was a pelvis-shaped deformable phantom. To mimic bony structures dental plaster was used. To simulate varying filling states of bladder and intestines the volume of the balloons was changed by injecting a water-iodine soap solution. Additionally, a prostate-shaped polystyrene object was glued to the „bladder" and a plastic bag filled with glass beads was used to mimic the colon [15].*

position of the substructure of the phantom was changed. The first CT was defined to be „planning CT". In all six datasets the bladder balloon, the prostate-shaped polystyrene object and the rectum bag were delineated manually. For the evaluation of the deformation first a rigid registration was applied for initial alignment and then three different deformable registration algorithms (demon registration of the iPlan software[1], featurelet registration with normalized correlation (NC) metric, and featurelet registration with mutual information (MI) metric) were used and the results compared.
For analyzing the deformed contours two different metrics were used:

- Dice similarity coefficient (DSC): is also called *volume overlap index* and defined as

$$\mathrm{DSC} = \frac{V_{deformed} \cap V_{reference}}{(V_{deformed} \cup V_{reference})/2} \times 100 \tag{6.1}$$

- Hausdorff-distance: is defined as the maximum distance in pixels between two contours A and B

$$H(A,B) = \max\big(h(A,B), h(B,A)\big) \quad \text{with } h(A,B) = \max_{a\in A} \min_{b\in B} \|a-b\| \tag{6.2}$$

[1]RT planning software of *Brainlab* - http://www.brainlab.com/

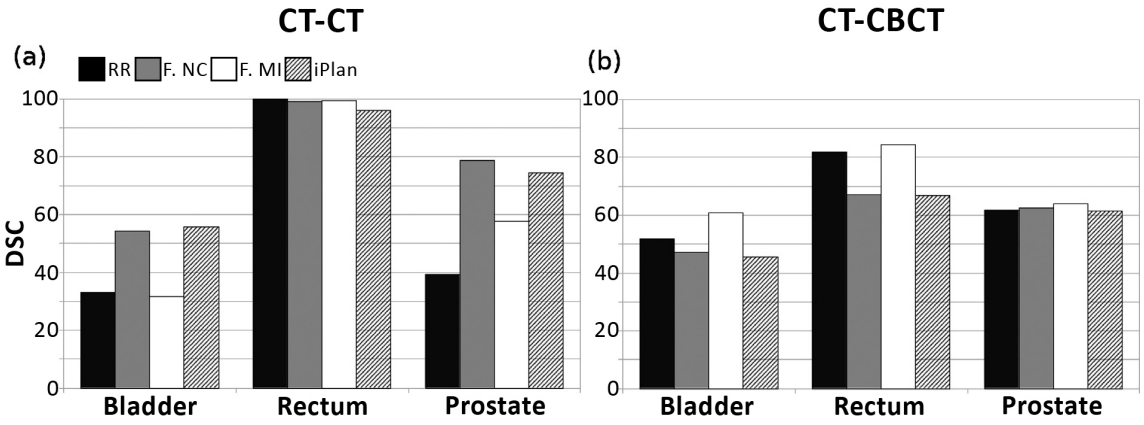

Figure 6.5.: *DSC for phantom II: results of the average DSC values of four different registration algorithms for intra- (CT-CT; two deformations are included) and inter modal (CT-CBCT; three deformations are included) registration of the deformable phantom. The algorithms used are rigid registration (RR), featurelet registration with normalized correlation (F. NC), featurelet registration with mutual information metric (F. MI) and the demons algorithm of iPlan (iPlan).*
In the case of intra modal registration the reference DSC for the rectum is 100 because this structure is not moving [15].

Results: The results of the average DSC values for the deformable phantom can be found in Figure 6.5. In the case of intra modal registration, both the featurelet NC method and the iPlan method were able to substantially improve the DSC for bladder and prostate. The featurelet MI method was just able to improve the DSC for prostate. In the case of inter modal registration, for all structures only the featurelet MI method was able to improve the DSC.

The results of the average maximum Hausdorff-distances for the deformable phantom can be found in Figure 6.6. In the case of intra modal registration, the featurelet NC method was able to substantially improve the Hausdorff-distance for bladder and prostate. The featurelet MI method and the iPlan method were just able to slightly improve the Hausdorff-distance for prostate. In the case of inter modal registration, no deformable algorithm was able to significantly improve the result in comparison to rigid registration, but at least both featurelet methods performed better than the iPlan method.

Clinical Datasets

In addition to phantom studies some clinical datasets were examined. Detailed results can be found in [15] and [16].

For validation studies in [15] nine patients treated for prostate cancer and another ten patients undergoing radiation therapy for non-small cell lung cancer or

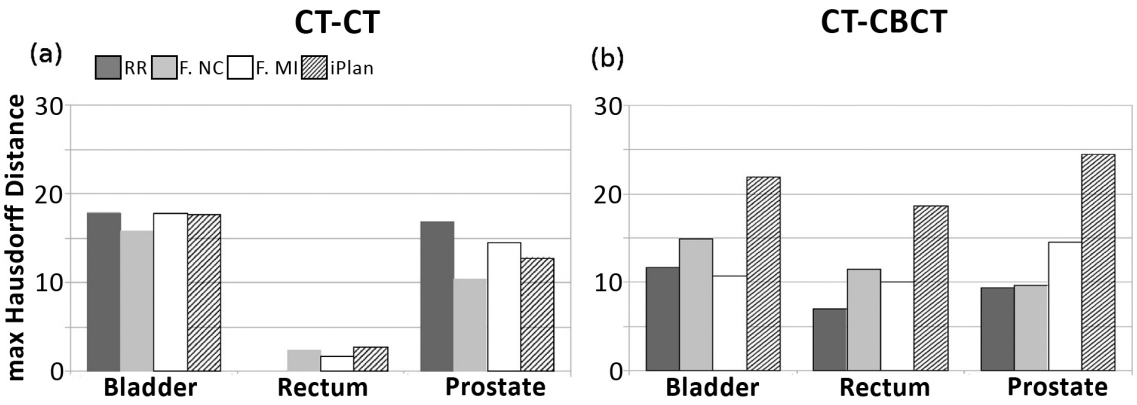

Figure 6.6.: *Hausdorff-distance for phantom II: results of the average maximum Hausdorff-distance for four different registration algorithms for intra- (CT-CT; two deformations are included) and inter modal (CT-CBCT; three deformations are included) registration of the deformable phantom. The algorithms are rigid registration (RR), featurelet registration with normalized correlation (F. NC), featurelet registration with mutual information metric (F. MI) and the demons algorithm of iPlan (iPlan).*
In the case of intra modal registration the reference Hausdorff-distance for the rectum is 0 pixel because this structure was not moving [15].

lung metastasis were selected. For both prostate and lung cases a planning CT and repeated CBCT images were acquired. For the prostate cancer patients bladder, prostate and rectum were delineated on all images by one radiation oncologist. For the validation study all datasets were taken into account. For the lung cases each planning CT and one randomly selected CBCT were considered for the validation study and the gross target volume (GTV) was delineated. For image registration the same procedure as for *phantom II* (compare Section 6.1.2) was used (a rigid registration and three different deformable registration algorithms) and the deformed contours were also compared with DSC and Hausdorff-distance. To check for statistical significance of the results the *Wilcoxon signed rank test* was used.

For validation studies in [16] datasets from ten cervical cancer patients were used. For each patient a planning CT and repeated CT as well as CBCT images were acquired. For the evaluation each planning CT, one repeated CT and one CBCT was considered. The OARs were delineated by an expert physician. For the study only bladder and rectum contours were taken into account. For image registration the same procedure as for *phantom II* (compare Section 6.1.2) was used (a rigid registration and three different deformable registration algorithms) and the deformed contours were compared by calculating their DSC. To check for statistical significance of the difference between two methods a *two-tailed Student's t-test* at a 95 % confidence level was used.

Results: The results of the first study by *Fabri et al.* [15] did not show a clear superiority over rigid registration for any deformable registration method for the

prostate cases. Only for the bladder contour the iPlan method was able to significantly improve the result of the rigid registration.
For the lung cases the results can be found in Figure 6.7. Every deformable method was able to achieve a significant improvement of the DSC value. For the Hausdorff-distance only the featurelet method with normalized correlation was significantly better than the initial rigid registration.

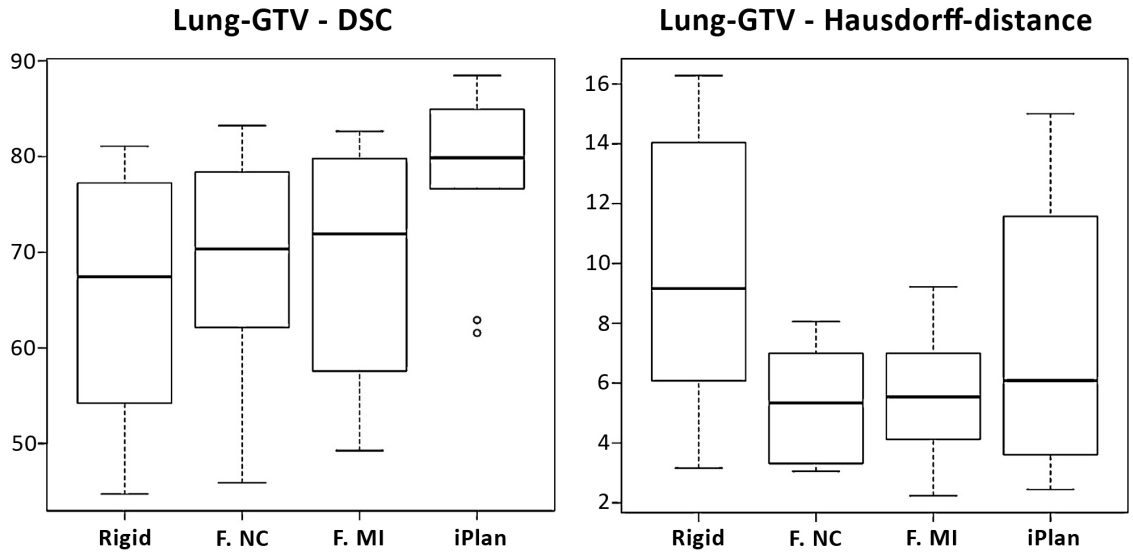

Figure 6.7.: *Clinical studies - lung cases: boxplots of the dice similarity coefficient (left side) and the Hausdorff-distance (right side) for the ten lung patients. The algorithms used for registration are rigid registration (RR), featurelet registration with normalized correlation (F. NC), featurelet registration with mutual information metric (F. MI) and the demons algorithm of iPlan (iPlan).*
Regarding the DSC all methods were able to significantly improve the result of the RR, with the best performance from the iPlan algorithm. Regarding the Hausdorff-distance again all methods showed an improvement from RR, but only the F. NC method was statistically significantly better than the RR [15].

The results of the second study by *Zambrano et al.* [16] did not show a significant improvement after deformable image registration for both featurelet methods as well as the iPlan method. Reasons for failure could be on the one hand the high-intensity variations introduced by the presence of air and on the other hand large deformations in the bladder observed for some patients.

6.2. Validation of the Slicer Implementation

As the basic structure of the registration-logic is the same for the 3D Slicer module *Featurelet-Registration* and the algorithm developed by *Fabri* [14], also some of the validation methods as presented in Section 6.1.2 are used. Thus, the performance can be compared directly, because there are no methodological incompatibilities.

6.2.1. Visual Inspection

The first stage of the validation of the 3D Slicer implementation was a visual inspection of the registration results. The main focus was on the orientation of the deformation field. It had to be verified that all coordinate transformations of the algorithm are performed in the right way. The reason behind this is the usage of different coordinate systems of VTK and ITK. A transformation is necessary, as visualization is done by VTK and computation by ITK.

For visual inspection two different datasets have been used. The first one is a series of two MR-scans of a brain tumor patient. The images are provided by the 3D Slicer „Sample Data" module. This dataset was used to check if the algorithm produces meaningful deformations and if the size of the output deformation field corresponds to the image size. The second dataset was a series of 11 thorax CT-scans showing the lung from 0 % to 100 % inhalation in 10 % steps. This dataset was chosen as an initial validation of the registration accuracy, because the extend of the motion of the lung is clearly visible. The result and the corresponding deformation field of the registration of the 0 % to the 100 % inhalation image is shown in Figure 6.8.

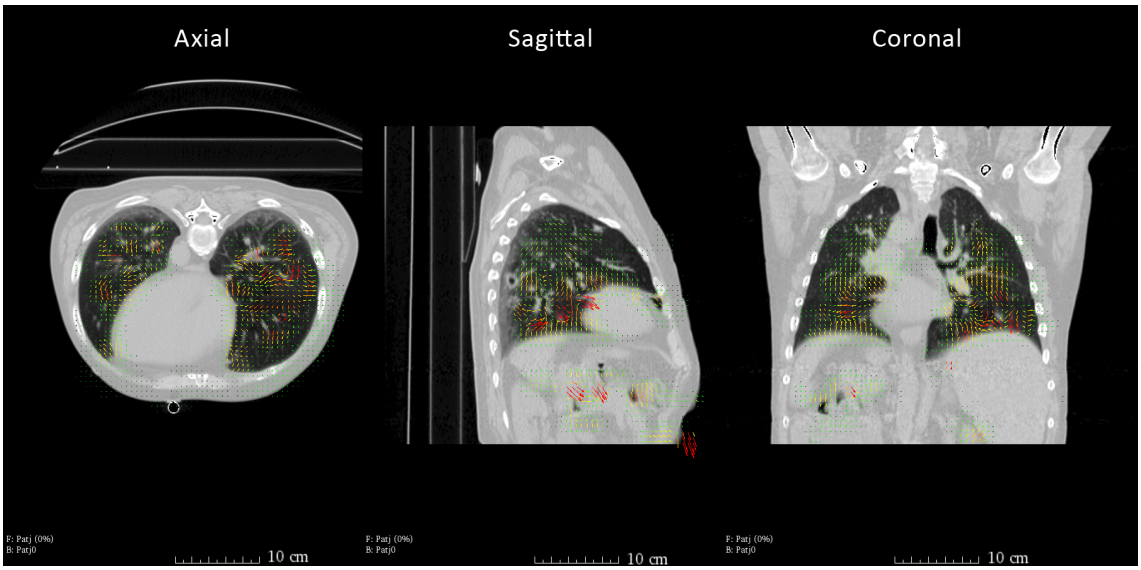

Figure 6.8.: *The axial, sagital and coronal view show the warped image of the registration of a maximum exhale to a maximum inhale state CT of a thorax; the corresponding deformation field is visualized directly in 3D Slicer using the built-in „Transforms" module.*

6.2.2. Phantom I

For quantification of registration accuracy the box-shaped deformable phantom described in Section 6.1.2 was used. The results of the Slicer implementation are compared to the results of the initial featurelet-algorithm of *Fabri*, a B-Spline registration of Slicer and a rigid registration. Together with the actual displacements of two deformations the figures are summarized in Table 6.1.

	Displacement	Rigid	B-Spline	Featurelet	Slicer-Featurelet
1st	4.6±1.3 mm	2.8±1.6 mm	2.4±1.5 mm	1.4±1.2 mm	0.9±0.6 mm
2nd	3.8±1.4 mm	1.9±1.1 mm	1.8±1.1 mm	1.4±0.9 mm	1.3±0.9 mm

Table 6.1.: *Results Phantom I: comparison of mean and standard deviations of displacements and registration residuals of the 24 Teflon markers based on two different deformations of the box-shaped phantom. „Featurelet" is the in-house-developed algorithm of* Fabri *and „Slicer-Featurelet" is the Slicer implementation.*

The improved registration accuracy of the 3D Slicer module *Feautrelet-Registration* over the initial algorithm is due to the possibility of user interaction as well as the availability of visualization. This makes it easier to leverage the powerful pre-processing tools directly available in Slicer. Pre-processing of the box-shaped phantom is illustrated in Figure 6.9.

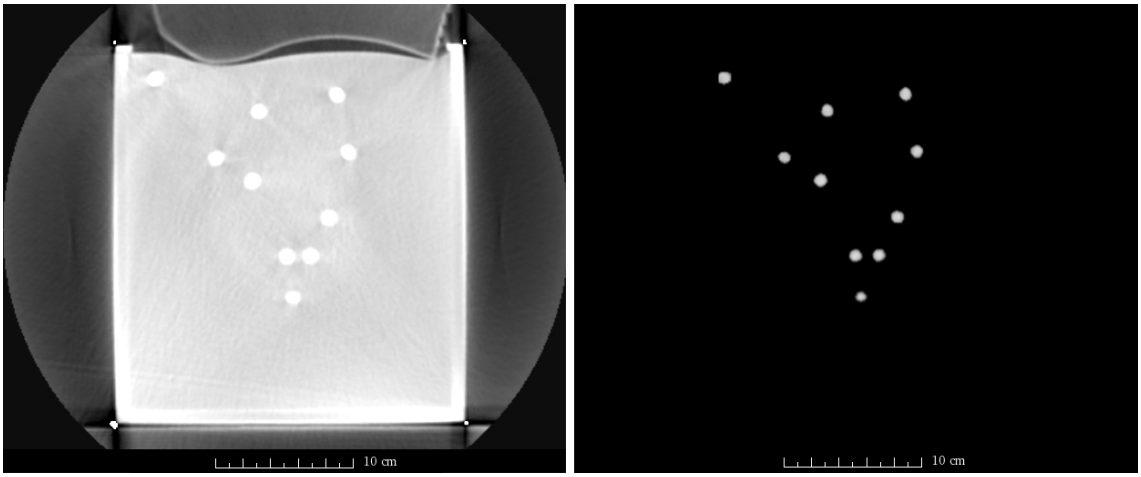

Figure 6.9.: *Pre-processing of Phantom I: to facilitate the registration process several 3D Slicer modules allow for pre-processing of the dataset. The left side shows the box-shaped phantom before and the right side after the selection of a region of interest as well as setting all voxels below a certain threshold value to zero.*

6.2.3. Phantom II

For a more sophisticated quantification of registration accuracy the pelvis-shaped deformable phantom described in Section 6.1.2 was used. The evaluation was based on the same three series of CT images and the same manually delineated datasets as used by Fabri [14]. Also the first CT was defined to be the „planning CT". The other two CT scans of the phantom with altered filling state of the balloons and changed position of the substructures were registered on the first CT scan. The deformation field obtained with the Featurelet-Registration module was applied on the manually delineated structures to obtain the deformed contours. For calculating the Dice similarity metrics and the maximum Hausdorff-distance the SlicerRT module „Contour Comparison" was used. There, the Hausdorff-distance is given in *mm*. Thus, the numerical values are not directly comparable to the values from Section 6.1.2 obtained by Fabri, as they are given in *pixel*.

The results of the evaluation are shown in Figure 6.10. DSC values and maximum Hausdorff-distances are plotted against an ascending degree of user interaction. Registration without user interaction (→ *No Pre-Process*) already leads to an improvement of the relative overlap of the CTV. By reducing the images to a subvolume containing only the necessary information (→ *Crop Volume*) registration accuracy is comparable to the one of the initial algorithm. With a higher degree of user interaction (→ *Structure only*) an improvement of the results presented by *Fabri et al.* [15] is achievable.

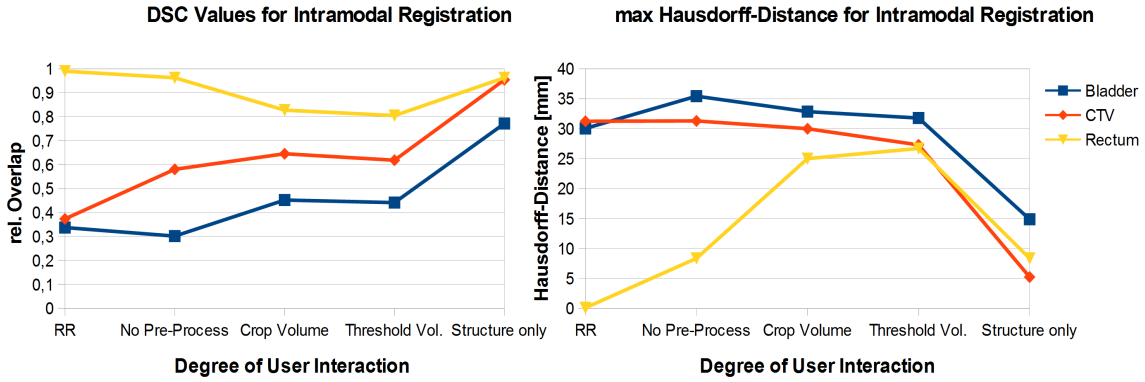

Figure 6.10.: *Results Phantom II: the graphics show the average DSC values and the average maximum Hausdorff-distance of two deformations. Values of the rigid registration (RR) correspond to the values for RR in the results of Fabri [15]. The first set of registrations was performed without pre-processing of the images and with default settings of the registration-parameters. For the second set only a region of interest (ROI) of moving and fixed image was considered by using the Crop Volume module of Slicer. Additional thresholding was used for the third set of registrations. The best result was achieved with the highest degree of user interaction. For* Structure only *a ROI was set around the particular organ and individual threshold values were used.*

7. Discussion and Conclusion

We have created the *Featurelet-Registration* module, which adds usability to the in-house-developed algorithm and makes it an accessible research tool for model-independent image registration. It allows for direct visualization of the output and quick adjustment of registration parameters. Thus, our algorithm can be used also as tool for further investigation of cases with less satisfying registration results. In addition, as the whole code of both 3D Slicer and our extension is open source, it is easily accessible for everyone and can be used as basis for further development.

For the implementation the software platform 3D Slicer was chosen, as it provides a powerful GUI to explore and interact with imaging datasets. Steps such as thresholding an image, cropping a volume or making an initial rigid registration of two images as well as visualization of a deformation field can all be done by the use of powerful built-in modules of 3D Slicer. Thus, pre-processing as well as image registration and post-processing is combined within a single software application. In addition, 3D Slicer is designed to be available on multiple platforms, including Windows, Linux and Mac Os X.

One of the drawbacks of the algorithm, which still hinders full usability is computation time. A typical intra-modal registration of a CT with default settings takes about 5-20 min. Thus, we plan to implement Graphics Processing Unit (GPU) optimizations in the future to speed up the overall registration time. This parallelization of the optimization process is possible, as each featurelet is registered independently to its corresponding search region.

8. Acknowledgments

I would like to thank all the people, who supported me with this master thesis. Especially I would like to thank in person:

- ***Prof. Dietmar Georg*** for introducing me to the interesting topic of radiation oncology and enabling to carry out my master thesis at AKH, Vienna. In addition, I want to express my gratitude for the guidance in writting and the rapid responses.
- ***Hugo Furtado*** for introducing me to the world of software development and his guidance, support and time he spent on discussing problems with me focusing on comprehension.
- ***Yvette Seppenwoolde*** for her assistance and feedback on my thesis as well as for fruitful discussions in our workpackage 2 meetings.
- The ***students meeting*** was also nice to be kept informed about other projects of the department and to maintain a good working atmosphere.

Last but by no means least I would like to thank my girlfriend ***Anna*** as well as my ***Family*** who have been supporting me throughout my studies without any reservations.

A. Appendix

A.1. Code for `DeformationFieldGenerator`

Code A.1 Member function `GenerateDeformationField()` of the class `DeformationFieldGenerator`

```
void GenerateDeformationField()
{
  FeatureletRegistrationResultListType::iterator it;
  DeformationFieldType::IndexType start;
  start[0] = start[1] = start[2] = 0.0;

  DeformationFieldType::SizeType size;
  size[0] = m_NumberOfFeaturelets[0]+1;
  size[1] = m_NumberOfFeaturelets[1]+1;
  size[2] = m_NumberOfFeaturelets[2]+1;

  DeformationFieldType::RegionType region0;
  region0.SetSize(size);
  region0.SetIndex(start);

  double spacing1[Dimension];
  spacing1[0] = m_SizeOfFeaturelets[0]*m_ImageSpacingFix[0];
  spacing1[1] = m_SizeOfFeaturelets[1]*m_ImageSpacingFix[1];
  spacing1[2] = m_SizeOfFeaturelets[2]*m_ImageSpacingFix[2];

  m_DeformationFieldRawX = DeformationFieldType::New();
  m_DeformationFieldRawX->SetRegions(region0);
  m_DeformationFieldRawX->Allocate();
  m_DeformationFieldRawX->SetSpacing(spacing1);
  m_DeformationFieldRawX->SetOrigin(m_ImageOriginFix);

  PixelType zeroVector = 0;
  m_DeformationFieldRawX->FillBuffer(zeroVector);

  m_DeformationFieldRawY = DeformationFieldType::New();
```

```
m_DeformationFieldRawY->SetRegions(region0);
m_DeformationFieldRawY->Allocate();
m_DeformationFieldRawY->SetSpacing(spacing1);
m_DeformationFieldRawY->SetOrigin(m_ImageOriginFix);
m_DeformationFieldRawY->FillBuffer(zeroVector);

m_DeformationFieldRawZ = DeformationFieldType::New();
m_DeformationFieldRawZ->SetRegions(region0);
m_DeformationFieldRawZ->SetSpacing(spacing1);
m_DeformationFieldRawZ->SetOrigin(m_ImageOriginFix);
m_DeformationFieldRawZ->Allocate();
m_DeformationFieldRawZ->FillBuffer(zeroVector);

m_MetricValue = DeformationFieldType::New();
m_MetricValue->SetRegions(region0);
m_MetricValue->SetSpacing(spacing1);
m_MetricValue->Allocate();
m_MetricValue->FillBuffer(zeroVector);

PixelType transformVectorX, transformVectorY, transformVectorZ;

double bestValue;
metricthreshold = 1000;
int underthreshold = 0;
int abovethreshold = 0;

for (it = m_RegistrationResultList.begin(); //
     it != m_RegistrationResultList.end(); it++) {
  DeformationFieldType::IndexType pixelIndex ;
  TransformType::OutputVectorType offset;
  offset[0]= (*it)->GetTransform()->GetOffset()[0];
  offset[1]= (*it)->GetTransform()->GetOffset()[1];
  offset[2]= (*it)->GetTransform()->GetOffset()[2];
  pixelIndex[0]= (*it)->GetFeatureletGridPosition()[0];
  pixelIndex[1]= (*it)->GetFeatureletGridPosition()[1];
  pixelIndex[2]= (*it)->GetFeatureletGridPosition()[2];

  transformVectorX = offset[0];
  transformVectorY = offset[1];
  transformVectorZ = offset[2];

  bestValue = (*it)->GetOptimizer()->GetValue();
  if (bestValue < metricthreshold) {
    abovethreshold++;
```

```
      m_DeformationFieldRawX->SetPixel(pixelIndex,transformVectorX);
      m_DeformationFieldRawY->SetPixel(pixelIndex,transformVectorY);
      m_DeformationFieldRawZ->SetPixel(pixelIndex,transformVectorZ);
    }
    else {
      underthreshold++;
      m_DeformationFieldRawX->SetPixel(pixelIndex, 0);
      m_DeformationFieldRawY->SetPixel(pixelIndex, 0);
      m_DeformationFieldRawZ->SetPixel(pixelIndex, 0);
    }
  }
}
```

Code A.2 Member function `RescaleDeformationField()` of the class `DeformationFieldGenerator`

```
void RescaleDeformationField()
{
  typedef itk::ResampleImageFilter<DeformationFieldType,
                                   DeformationFieldType> FilterType;
  FilterType::Pointer filterX = FilterType::New();
  FilterType::Pointer filterY = FilterType::New();
  FilterType::Pointer filterZ = FilterType::New();
  TransformType::Pointer transform = TransformType::New();

  typedef itk::LinearInterpolateImageFunction< DeformationFieldType,
                                               double > InterpolatorType;
  InterpolatorType::Pointer interpolator = InterpolatorType::New();

  DeformationFieldType::IndexType startII;
  startII[0] = startII[1] = startII[2] = 0.0;

  DeformationFieldType::SizeType sizeII;
  sizeII[0] = m_ImageSizeFix[0];
  sizeII[1] = m_ImageSizeFix[1];
  sizeII[2] = m_ImageSizeFix[2];

  DeformationFieldType::SizeType sizeFeaturelet;
  sizeFeaturelet[0] = m_NumberOfFeaturelets[0]+1;
  sizeFeaturelet[1] = m_NumberOfFeaturelets[1]+1;
  sizeFeaturelet[2] = m_NumberOfFeaturelets[2]+1;
```

```
m_DeformationFieldRawXI = DeformationFieldType::New();
m_DeformationFieldRawYI = DeformationFieldType::New();
m_DeformationFieldRawZI = DeformationFieldType::New();

DeformationFieldType::RegionType regionII;
regionII.SetSize(sizeII);
regionII.SetIndex(startII);

m_DeformationFieldRawXI->SetRegions(regionII);
m_DeformationFieldRawYI->SetRegions(regionII);
m_DeformationFieldRawZI->SetRegions(regionII);

m_DeformationFieldRawXI->Allocate();
m_DeformationFieldRawYI->Allocate();
m_DeformationFieldRawZI->Allocate();

PixelType zeroVector = 0;
m_DeformationFieldRawXI->FillBuffer(zeroVector);
m_DeformationFieldRawYI->FillBuffer(zeroVector);
m_DeformationFieldRawZI->FillBuffer(zeroVector);

filterX->SetTransform(transform);
filterX->SetInterpolator(interpolator);
filterY->SetTransform(transform);
filterY->SetInterpolator(interpolator);
filterZ->SetTransform(transform);
filterZ->SetInterpolator(interpolator);

double origin[Dimension];
origin[0] = m_ImageOriginFix[0];
origin[1] = m_ImageOriginFix[1];
origin[2] = m_ImageOriginFix[2];

double spacingFeaturelet[Dimension];
spacingFeaturelet[0] = m_SizeOfFeaturelets[0]*m_ImageSpacingFix[0];
spacingFeaturelet[1] = m_SizeOfFeaturelets[1]*m_ImageSpacingFix[1];
spacingFeaturelet[2] = m_SizeOfFeaturelets[2]*m_ImageSpacingFix[2];

DeformationFieldType::SpacingType spacing;
spacing[0] = m_ImageSpacingFix[0];
spacing[1] = m_ImageSpacingFix[1];
spacing[2] = m_ImageSpacingFix[2];
```

```
  double o2[3];
  for(unsigned int i=0; i<3; i++) {
    double center = origin[i] + (sizeFeaturelet[i] *
      spacingFeaturelet[i]) / 2.0;
    o2[i] = center - sizeII[i] * spacing[i] / 2.0;
  }

  filterX->SetOutputOrigin(o2);
  filterX->SetSize( sizeII );
  filterX->SetOutputSpacing(spacing);
  filterY->SetOutputOrigin(o2);
  filterY->SetSize( sizeII );
  filterY->SetOutputSpacing(spacing);
  filterZ->SetOutputOrigin(o2);
  filterZ->SetSize( sizeII );
  filterZ->SetOutputSpacing(spacing);

  DeformationFieldType::DirectionType direction;
  direction.SetIdentity();

  filterX->SetOutputDirection( direction );
  filterY->SetOutputDirection( direction );
  filterZ->SetOutputDirection( direction );

  filterX->SetInput(m_DeformationFieldRawX);
  filterY->SetInput(m_DeformationFieldRawY);
  filterZ->SetInput(m_DeformationFieldRawZ);

  filterX->Update();
  filterX->UpdateLargestPossibleRegion();
  filterY->Update();
  filterY->UpdateLargestPossibleRegion();
  filterZ->Update();
  filterZ->UpdateLargestPossibleRegion();

  m_DeformationFieldRawXI = filterX->GetOutput();
  m_DeformationFieldRawYI = filterY->GetOutput();
  m_DeformationFieldRawZI = filterZ->GetOutput();

  m_DeformationFieldRawXI->Update();
  m_DeformationFieldRawYI->Update();
  m_DeformationFieldRawZI->Update();
}
```

Code A.3 Member function `TotalDeformationFieldCreator()` of the class `DeformationFieldGenerator`

```
void TotalDeformationFieldCreator()
{
  DeformationFieldTypeVector::IndexType startII;
  startII[0] = startII[1] = startII[2] = 0.0;

  DeformationFieldTypeVector::SizeType sizeII;
  sizeII[0] = m_ImageSizeFix[0];
  sizeII[1] = m_ImageSizeFix[1];
  sizeII[2] = m_ImageSizeFix[2];
  DeformationFieldTypeVector::RegionType regionII;
  regionII.SetSize(sizeII);
  regionII.SetIndex(startII);

  m_DeformationFieldRawII = DeformationFieldTypeVector::New();
  m_DeformationFieldRawII->SetRegions(regionII);

  DeformationFieldTypeVector::SpacingType spacingII;
  spacingII[0] = m_ImageSpacingFix[0];
  spacingII[1] = m_ImageSpacingFix[1];
  spacingII[2] = m_ImageSpacingFix[2];
  m_DeformationFieldRawII->SetSpacing(spacingII);

  m_DeformationFieldRawII->SetOrigin(m_ImageOriginFix);

  try {
    m_DeformationFieldRawII->Allocate();
  }
  catch(itk::ExceptionObject &e) {
    std::cerr << "Failed to allocate memory: "
              << e.GetDescription() << std::endl;
    return;
  }

  PixelTypeVector zeroVector;
  zeroVector.Fill(0);
  m_DeformationFieldRawII->FillBuffer(zeroVector);
  PixelTypeVector TotalDeformationVector;
  DeformationFieldTypeVector::IndexType pixelIndexII;

  for (int i=1; i<m_ImageSizeFix[0]-1; i++) {
    for (int j=1; j<m_ImageSizeFix[1]-1; j++) {
      for (int k=1; k<m_ImageSizeFix[2]-1; k++) {
```

```
        pixelIndexII[0] = i;
        pixelIndexII[1] = j;
        pixelIndexII[2] = k;

        TotalDeformationVector[0] = m_DeformationFieldRawXI
                                          ->GetPixel(pixelIndexII);
        TotalDeformationVector[1] = m_DeformationFieldRawYI
                                          ->GetPixel(pixelIndexII);
        TotalDeformationVector[2] = m_DeformationFieldRawZI
                                          ->GetPixel(pixelIndexII);

        m_DeformationFieldRawII->SetPixel(pixelIndexII,
                                          TotalDeformationVector);
      }
    }
  }
}
```

Code A.4 Member function `VTKDeformationFieldCreator()` of the class `DeformationFieldGenerator`

```
vtkSmartPointer<vtkOrientedGridTransform> VTKDeformationFieldCreator()
{
  grid_Ras = vtkSmartPointer<vtkOrientedGridTransform>::New();
  deformationField = vtkSmartPointer<vtkImageData>::New();

  deformationField->SetOrigin
                          (m_DeformationFieldRawII->GetOrigin()[0],
                           m_DeformationFieldRawII->GetOrigin()[1],
                           m_DeformationFieldRawII->GetOrigin()[2]);
  deformationField->SetSpacing
                         (m_DeformationFieldRawII->GetSpacing()[0],
                          m_DeformationFieldRawII->GetSpacing()[1],
                          m_DeformationFieldRawII->GetSpacing()[2]);
  const vtkSmartPointer<vtkMatrix4x4> gridDirectionMatrix_LPS =
                               vtkSmartPointer<vtkMatrix4x4>::New();
  for (unsigned int row=0; row<3; row++) {
    for (unsigned int column=0; column<3; column++) {
      gridDirectionMatrix_LPS->SetElement(row, column,
                                  m_ImageDirectionFix(row,column));
```

```
    }
  }
  grid_Ras->SetGridDirectionMatrix
                              (gridDirectionMatrix_LPS.GetPointer());

  DeformationFieldTypeVector::RegionType region =
                        m_DeformationFieldRawII->GetBufferedRegion();
  DeformationFieldTypeVector::SizeType imageSize = region.GetSize();

  int extent[6]={0,(int) imageSize[0]-1, 0, (int) imageSize[1]-1, 0,
                 (int) imageSize[2]-1};
  deformationField->SetExtent(extent);
#if (VTK_MAJOR_VERSION <= 5)
  deformationField->SetScalarTypeToDouble();
  deformationField->SetNumberOfScalarComponents(3);
  deformationField->AllocateScalars();
#else
  deformationField->AllocateScalars(VTK_DOUBLE, 3);
#endif

  double* displacementVectors_Ras = reinterpret_cast<double*>
                              (deformationField->GetScalarPointer());

  itk::ImageRegionConstIterator<DeformationFieldTypeVector>
                          inputIt(m_DeformationFieldRawII,
                      m_DeformationFieldRawII->GetRequestedRegion());
  inputIt.GoToBegin();
  while( !inputIt.IsAtEnd() ) {
    DeformationFieldTypeVector::PixelType displacementVectorLps =
                                                        inputIt.Get();
    *(displacementVectors_Ras++) = displacementVectorLps[0];
    *(displacementVectors_Ras++) = displacementVectorLps[1];
    *(displacementVectors_Ras++) = displacementVectorLps[2];
    ++inputIt;
  }

#if (VTK_MAJOR_VERSION <= 5)
  grid_Ras->SetDisplacementGrid( deformationField.GetPointer() );
#else
  grid_Ras->SetDisplacementGridData( deformationField.GetPointer() );
#endif
  return grid_Ras;
}
```

Code A.5 Member function `WarpImagebyDeformationField()` of the class `DeformationFieldGenerator`

```
int  WarpImagebyDeformationField(ImageType* InputImage,
                                 ImageType* FixedImage,
                                 ImageType* OutputImage)
{
  typedef itk::WarpImageFilter<ImageType, ImageType,
   DeformationFieldTypeVector> WarperType;
  typedef itk::LinearInterpolateImageFunction<ImageType,
   double> InterpolatorType;
  WarperType::Pointer warper = WarperType::New();
  InterpolatorType::Pointer interpolator = InterpolatorType::New();

  DeformationFieldTypeVector::SizeType sizeII;
  sizeII[0] = m_ImageSizeFix[0];
  sizeII[1] = m_ImageSizeFix[1];
  sizeII[2] = m_ImageSizeFix[2];
  warper->SetOutputSize(sizeII);

  warper->SetInput(InputImage);
  warper->SetInterpolator(interpolator);
  warper->SetOutputSpacing(FixedImage->GetSpacing());
  warper->SetOutputOrigin(FixedImage->GetOrigin());
  warper->SetOutputDirection(FixedImage->GetDirection());
  warper->SetDisplacementField(m_DeformationFieldRawII);
  warper->UpdateLargestPossibleRegion();
  warper->Update();
  OutputImage->Graft(warper->GetOutput());
  try {
    OutputImage->Update();
  }
  catch(itk::ExceptionObject &e) {
    std::cerr << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

Bibliography

[1] Stewart, J. *et al.* Automated weekly replanning for intensity-modulated radiotherapy of cervix cancer. *International Journal of Radiation Oncology Biology Physics* **78**, 350–358 (2010).

[2] Bondar, L. *et al.* Toward an individualized target motion management for IMRT of cervical cancer based on model-predicted cervix-uterus shape and position. *Radiotherapy and Oncology* **99**, 240–245 (2011).

[3] Bondar, M. L. *et al.* Individualized nonadaptive and online-adaptive intensity-modulated radiotherapy treatment strategies for cervical cancer patients based on pretreatment acquired variable bladder filling computed tomography scans. *International Journal of Radiation Oncology Biology Physics* **83**, 1617–1623 (2012).

[4] Bondar, M. L., Hoogeman, M., Schillemans, W. & Heijmen, B. Intra-patient semi-automated segmentation of the cervix-uterus in CT-images for adaptive radiotherapy of cervical cancer. *Physics in medicine and biology* **58**, 5317–32 (2013).

[5] Ahmad, R. *et al.* A margin-of-the-day online adaptive intensity-modulated radiotherapy strategy for cervical cancer provides superior treatment accuracy compared to clinically recommended margins: a dosimetric evaluation. *Acta oncologica (Stockholm, Sweden)* **52**, 1430–6 (2013).

[6] Castelli, J. *et al.* Impact of head and neck cancer adaptive radiotherapy to spare the parotid glands and decrease the risk of xerostomia. *Radiat Oncol.* **10**, Epub ahead of print (2015).

[7] Keall, P. J., Chen, G. T. Y., Joshi, S., Mackie, T. R. & Stevens, C. W. Time - The fourth dimension in radiotherapy (ASTRO Panel Discussion). *International Journal of Radiation Oncology Biology Physics* **57**, 8–9 (2003).

[8] Yan, D., Vicini, F., Wong, J. & Martinez, A. Adaptive radiation therapy. *Physics in Medicine and Biology* **42**, 123 (1997).

[9] Georg, D. Basic Seminar of Radiation Oncology Physics. *Chapter 4* (2015).

[10] Maintz, J. B. A. & Viergever, M. a. A survey of medical image registration. *Medical Image Analysis* **2**, 1–36 (1998).

[11] Zitová, B. & Flusser, J. Image registration methods: A survey. *Image and Vision Computing* **21**, 977–1000 (2003).

[12] Sotiras, A., Davatzikos, C. & Paragios, N. Deformable medical image registration: A survey. *IEEE Transactions on Medical Imaging* **32**, 1153–1190 (2013).

[13] Söhn, M. *et al.* Model-independent , multimodality deformable image registration by local matching of anatomical features and minimization of elastic energy. *Medical Physics* **35**, 866–878 (2008).

[14] Fabri, D. *Non-rigid Registration for Radiotherapy.* Ph.D. thesis, Medical University of Vienna (2014).

[15] Fabri, D. *et al.* A quantitative comparison of the performance of three deformable registration algorithms in radiotherapy. *Zeitschrift für Medizinische Physik* **23**, 279–290 (2013).

[16] Zambrano, V. *et al.* Performance validation of deformable image registration in the pelvic region. *Journal of Radiation Research* **54** (2013).

[17] Siegel, R. *et al.* Cancer Treatment and Survivorship Statistics , 2012. *CA: a cancer journal for clinicians* **62**, 220–241 (2012).

[18] Mayles, P., Nahum, A., Rosenwald, J. C. & Papanikolaou, N. Handbook of Radiotherapy Physics: Theory and Practice (2008).

[19] Podgorsak, E. *Radiation Oncology Physics: A Handbook for Teachers and Students* (International Atomic Energy Agency, 2005).

[20] ICRU Report 50. Prescribing, Recording and Reporting Photon Beam Therapy (1993).

[21] ICRU Report 62. Prescribing, Recording and Reporting Photon Beam Therapy. *(Supplement to ICRU Report 50)* (1999).

[22] ICRU Report 83. Prescribing, Recording, and Reporting Photon-Beam Intensity-Modulated Radiation Therapy (IMRT). *Journal of the ICRU* **10**, 1–106 (2010).

[23] Van Herk, M. Errors and Margins in Radiotherapy. *Seminars in Radiation Oncology* **14**, 52–64 (2004).

[24] Chan, P. *et al.* Inter- and Intrafractional Tumor and Organ Movement in Patients With Cervical Cancer Undergoing Radiotherapy: A Cinematic-MRI Point-of-Interest Study. *International Journal of Radiation Oncology Biology Physics* **70**, 1507–1515 (2008).

[25] Eiland, R. B., Maare, C., Sjöström, D., Samsø e, E. & Behrens, C. F. Dosimetric and geometric evaluation of the use of deformable image registration in adaptive intensity-modulated radiotherapy for head-and-neck cancer. *Journal of radiation research* 1–7 (2014).

[26] Yan, D. Adaptive Radiotherapy: Merging Principle into Clinical Practice. *Seminars in Radiation Oncology* **20**, 79–83 (2010).

[27] Oh, S., Jaffray, D. & Cho, Y.-B. A novel method to quantify and compare anatomical shape: application in cervix cancer radiotherapy. *Physics in medicine and biology* **59**, 2687–704 (2014).

[28] Birkfellner, W., Figl, M. & Hummel, J. *Medical Image Processing - A Basic Course* (CRC Press, 2011).

[29] Santamaría, J., Cordón, O. & Damas, S. A comparative study of state-of-the-art evolutionary image registration methods for 3D modeling. *Computer Vision and Image Understanding* **115**, 1340–1354 (2011).

[30] Thirion, J.-P. Image matching as a diffusion process: an analogy with Maxwell's demons. *Medical Image Analysis* **2**, 243–260 (1998).

[31] Robertson, S. P., Weiss, E. & Hugo, G. D. A block matching-based registration algorithm for localization of locally advanced lung tumors. *Medical physics* **41**, 11 (2014).

[32] Bankman, I. N. *Handbook of Medical Imaginng - Processing and Analysis* (Academic Press, London, 2000).

[33] Wong, J. C. H., Studholme, C., Hawkes, D. J. & Maisey, M. N. Evaluation of the limits of visual detection of image misregistration in a brain fluorine-18 fluorodeoxyglucose PET-MRI study. *European Journal of Nuclear Medicine* **24**, 642–650 (1997).

[34] Fitzpatrick, J. M. *et al.* Visual assessment of the accuracy of retrospective registration of MR and CT images of the brain. *IEEE Trans Med Imaging* **17**, 571–585 (1998).

[35] Christensen, G. E. *et al.* Introduction to the Non-Rigid Image Registration Evaluation Project. *Lecture Notes in Computer Science* **4057**, 128–135 (2006).

[36] Huttenlocher, D. P., Klanderman, G. A. & Rucklidge, W. J. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15**, 850–863 (1993).

[37] Pieper, S., Lorensen, B., Schroeder, W. & Kikinis, R. The NA-MIC Kit: ITK, VTK, Pipelines, Grids and 3D Slicer as an Open Platform for the Medical Image Computing Community. *Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium* 698–701 (2006).

[38] Eliceiri, K. W. *et al.* Biological Imaging Software Tools. *Nature Methods* **9**, 697–710 (2013).

[39] Fedorov, A. *et al.* 3D Slicer as an Image Computing Platform for the Quantitative Imaging Network. *Magn Reson Imaging* **30**, 1323–1341 (2012).

[40] Johnson, H. J., McCormick, M. M. & Ibànez, L. *The ITK Software Guide Book 1: Introduction and Development Guidelines Fourth Edition Updated for ITK version 4.7* (Insight Software Consortium, 2015).

[41] Schroeder, W., Martin, K. & Lorensen, W. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. *Proceedings of Seventh Annual IEEE Visualization '96* **1**, 93–100 (1996).

[42] Schroeder, W. J. & Martin, K. M. The visualization toolkit. In *Visualization Handbook*, 593–614 (2005).

[43] Zhang, T., Chi, Y., Meldolesi, E. & Yan, D. Automatic Delineation of On-Line Head-And-Neck Computed Tomography Images: Toward On-Line Adaptive Radiotherapy. *International Journal of Radiation Oncology Biology Physics* **68**, 522–530 (2007).