UNIVERSITÄT ZU LÜBECK

From the Institute of Mathematics and Image Computing
of the University of Lübeck
Director: Prof. Dr. Jan Modersitzki

Matrix-free approaches for deformable image registration with large-scale and real-time applications in medical imaging

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences

Submitted by

Lars König
from Preetz

Lübeck, 2018

First referee: Prof. Dr. Jan Lellmann

Second referee: Prof. Dr. Heinz Handels

Date of oral examination: November 8, 2018

Approved for printing. Lübeck, November 9, 2018

Abstract

In this thesis, we propose a novel computational approach to fast and memory-efficient deformable image registration. We demonstrate the relevance of the proposed method in three real-world medical applications with very different requirements, ranging from processing of large datasets to registration with real-time constraints.

The approach builds on a variational image registration model. In this model, finding a transformation which provides a reasonable image alignment is performed by minimizing an objective function. In the utilized discretize-then-optimize approach, the minimization is realized by using derivative-based optimization methods. Here, the objective function derivatives are typically the computationally most expensive operations, both in terms of runtime and memory requirements. Therefore, we analyze the matrix structure for all derivative components. Based on the analysis, we derive equivalent, fully matrix-free closed-form expressions for gradient computations as well as the Hessian-vector multiplication, enabling the use of matrix-free computations for both L-BFGS and Gauss-Newton optimization schemes.

The matrix-free computations completely eliminate the need for storing intermediate results and the cost of sparse matrix arithmetic. The expressions are fully parallelizable and the memory complexity for the derivative computations is reduced from linear to constant. We show that all important matrix-free derivative computations scale virtually linear, allowing to fully benefit from parallel execution. In comparison with matrix-based algorithms, the proposed approach is several orders of magnitude faster. The generic formulation of the matrix-free approach enables the implementation on different platforms. Besides multi-core CPUs, we present a GPU implementation which achieves a substantial, additional speedup. In order to justify the effort for deriving the matrix-free computations, we additionally implement the registration algorithm using an automatic differentiation framework. This method automatically computes optimized, analytically exact derivatives and allows for seamless execution on GPUs. In comparison with matrix-based methods, the automatic differentiation-based approach achieves comparable runtimes, making it a well-suited alternative for rapid prototyping and algorithm development.

In the second part of the thesis, we utilize the matrix-free registration in three clinical applications. First, in an application from oncology, we present an automatic pipeline for registration of follow-up thorax-abdomen CT scans. We evaluate the algorithm on a large number of datasets, achieving clinically feasible runtimes. Second, we consider registration of CT and cone-beam CT images in radiotherapy. To achieve physically plausible deformations, we introduce an additional local rigidity constraint. In comparison to a commercial registration method, we achieve comparable accuracy, while obtaining physically more plausible deformations within a clinically suitable runtime. Third, we use the registration in real-time liver ultrasound tracking in order to determine respiratory motion. For this, we integrate the matrix-free registration in a tracking scheme with a moving-window strategy. In a public benchmark, we achieve real-time performance with the lowest mean tracking error of all participants. In each of these applications, the matrix-free methods allow the use of registration in scenarios where it would not be possible otherwise, due to run-time or memory constraints. Thus, the matrix-free registration can contribute to further increasing the use of image registration in clinical practice.

Zusammenfassung

In dieser Arbeit stellen wir eine Methode für schnelle und speichereffiziente nicht-lineare Bildregistrierung vor. Wir zeigen die Relevanz der Methode in drei medizinischen Anwendungen mit sehr unterschiedlichen Anforderungen, von der Verarbeitung großer Datensätze bis hin zur Registrierung mit Echtzeitanforderungen.

Die Methode basiert auf einem variationellen Registrierungsmodell. In diesem Modell wird eine Transformation, die eine gute Bildüberlagerung erzeugt, durch Minimieren einer Zielfunktion bestimmt. Bei dem verwendeten Discretize-then-Optimize Ansatz wird dies durch Verwendung ableitungsbasierter Optimierungsverfahren realisiert. Hier sind die Ableitungen der Zielfunktion typischerweise die rechenintensivsten Operationen, sowohl hinsichtlich der Laufzeit als auch der Speicheranforderungen. Wir analysieren daher die Matrixstruktur für alle individuellen Komponenten der Ableitungen. Basierend darauf leiten wir äquivalente, vollständig matrixfreie Ausdrücke in geschlossener Form für Gradientenberechnungen und Hesse-Vektor-Multiplikationen her. Dies ermöglicht die Verwendung matrixfreier Methoden sowohl für L-BFGS- als auch für Gauß-Newton-Optimierung.

Die matrixfreien Berechnungen benötigen weder Speicher für Zwischenergebnisse noch Sparse-Matrix-Berechnungen. Die Ausdrücke sind vollständig parallelisierbar und die Speicherkomplexität der Ableitungsberechnungen verringert sich von linear auf konstant. Wir zeigen, dass alle wichtigen matrixfreien Ableitungsberechnungen nahezu linear skalieren und ideal von einer parallelen Ausführung profitieren. Im Vergleich zu matrixbasierten Algorithmen ist der Ansatz um mehrere Größenordnungen schneller. Die generische Formulierung des matrixfreien Ansatzes ermöglicht eine Implementierung auf verschiedenen Plattformen. Neben Multi-Core-CPUs präsentieren wir eine GPU-Implementierung, die eine wesentliche zusätzliche Beschleunigung erzielt. Um den Aufwand zur Herleitung der matrixfreien Berechnungen zu rechtfertigen, implementieren wir die Registrierung zusätzlich in einem Framework für automatisches Differenzieren. Diese Methode berechnet automatisch analytisch exakte Ableitungen und ermöglicht eine direkte Ausführung auf GPUs. Im Vergleich zu matrixbasierten Methoden erreicht dieser Ansatz ähnliche Laufzeiten und eignet sich damit für Rapid Prototyping und Algorithmenentwicklung.

Im zweiten Teil der Arbeit zeigen wir drei klinische Anwendungen der matrixfreien Registrierung. Zunächst präsentieren wir eine automatische Pipeline zur Registrierung von onkologischen Follow-Up Thorax-Abdomen-CTs. Wir evaluieren die Methode auf einer großen Anzahl von Datensätzen und erreichen klinisch geeignete Laufzeiten. Weiterhin betrachten wir CT- auf Cone-Beam-CT Registrierung in der Strahlentherapie. Für physikalisch plausible Deformationen führen wir eine Nebenbedingung für lokale Rigidität ein. Im Vergleich mit einem kommerziellen Algorithmus erreichen wir eine ähnliche Genauigkeit bei plausibleren Deformationen und klinisch geeigneter Laufzeit. Als dritte Anwendung präsentieren wir Echtzeit-Bewegungstracking auf Leber-Ultraschallbildern und integrieren die matrixfreie Registrierung in ein Tracking-Schema. In einem öffentlich verfügbaren Benchmark erzielen wir Echtzeit-Performance und den niedrigsten mittleren Trackingfehler aller Teilnehmer. In jeder dieser Anwendungen ermöglichen die matrixfreien Verfahren Bildregistrierung in Szenarien, in denen diese durch Laufzeit- oder Speicherbeschränkungen sonst unmöglich wäre und können damit dazu beitragen, die Verwendung von Registrierung in der klinischen Praxis weiter zu erhöhen.

Acknowledgments

This thesis emerged from my work at Fraunhofer MEVIS and many of the topics discussed here are closely related to research projects I have encountered during that time. Now that this journey comes to an end, I want to express my gratitude to all people that have supported me along the way.

Most of all, I thank Jan Lellmann for supervising my thesis. I am deeply grateful for his kindness and encouragement. His positive, constructive criticism and his unique talent for concise scientific writing truly motivated me at all times and taught me many things I wish I had learned sooner. Without him, this thesis would not have been possible.

I am also indebted to the late Bernd Fischer for raising my interest in image registration, his unique sense of humor and his passion for creating and leading a truly remarkable research group. He tragically passed away much too early, leaving behind a unique legacy. Without him, this thesis would certainly not exist.

Furthermore, I thank Jan Rühaak for his collaboration in many registration projects, his rational way of thinking, a lot of professional and personal advice, and for encouraging me to write and submit my very first paper, as well as many more thereafter. Special thanks go to Till Kipshagen for his tireless dedication to fine-grain performance optimizations, code refactoring and exceptional software development.

I thank Nils Papenberg for his continued support, which has lasted since the first time I came in contact with image registration and has finally led me to this thesis, as well as for the pleasant collaboration in many radiotherapy projects. I also thank Alexander Derksen for the joint work in radiotherapy, his support in deriving the matrix-free Hessian computations – which would not have been possible without his magic paper strip – and for enduring various journal paper reviews together.

Many thanks go to Johannes Lotz who shared an office with me for more than five years, his positive attitude and many shared laughs. A big thank you also goes to his successor Nadine Spahr who has repeatedly motivated me in the final months of this work, making this time much more bearable.

Furthermore, I thank everyone at Fraunhofer MEVIS and the Institute of Mathematics and Image computing for their unique team spirit, especially in the years of the Project Group Image Registration. These will always be an extraordinary experience to me.

Finally, I thank my family for always believing in me. I especially thank my wife Katrin, who had to sacrifice countless weekends and evenings for this thesis, for her love, patience and kindness, without which this work would never have been completed.

Contents

List of publications

The following list contains the author's publications related to this thesis. For clear identification, their citation keys are marked by an asterisk (*).

Journal publications

[BKR+14*] R. Berg, L. König, J. Rühaak, R. Lausen, and B. Fischer, "Highly efficient image registration for embedded systems using a distributed multicore DSP architecture", *Journal of Real-Time Image Processing*, vol. 14, no. 2, pp. 341–361, 2014. *Cit. on pp. 8, 20, 21, 26, 44, 85, 86.*

[DBK+15*] V. De Luca, T. Benz, S. Kondo, L. König, D. Lübke, S. Rothlübbers, O. Somphone, *et al.*, "The 2014 liver ultrasound tracking benchmark", *Physics in Medicine and Biology*, vol. 60, no. 14, pp. 5571–5599, 2015. *Cit. on pp. 6, 8, 16, 25, 155, 156, 161, 164.*

[KDPH16*] L. König, A. Derksen, N. Papenberg, and B. Haas, "Deformable image registration for adaptive radiotherapy with guaranteed local rigidity constraints", *Radiation Oncology*, vol. 11, no. 1, pp. 122–130, 2016. *Cit. on pp. 8, 25, 26, 141.*

[KRDL18*] L. König, J. Rühaak, A. Derksen, and J. Lellmann, "A matrix-free approach to parallel and memory-efficient deformable image registration", *SIAM Journal on Scientific Computing*, vol. 40, no. 3, pp. B858–B888, 2018. *Cit. on pp. 3, 8, 21, 23, 44, 80, 81, 106.*

[RKT+17*] J. Rühaak, L. König, F. Tramnitzke, H. Köstler, and J. Modersitzki, "A matrix-free approach to efficient affine-linear image registration on CPU and GPU", *Journal of Real-Time Image Processing*, vol. 13, no. 1, pp. 205–225, 2017. *Cit. on pp. 8, 15, 21, 44.*

Peer-reviewed conference proceedings

[KDHP15*] L. König, A. Derksen, M. Hallmann, and N. Papenberg, "Parallel and memory efficient multimodal image registration for radiotherapy using normalized gradient fields", in *IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, 2015, pp. 734–738. *Cit. on pp. 8, 21, 23, 44, 106.*

[KKR14*] L. König, T. Kipshagen, and J. Rühaak, “A non-linear image registration scheme for real-time liver ultrasound tracking using normalized gradient fields”, in *MICCAI Challenge on Liver Ultrasound Tracking (CLUST14)*, 2014, pp. 29–36. *Cit. on pp. 7, 8, 156, 159, 160, 162.*

[KR14*] L. König and J. Rühaak, “A fast and accurate parallel algorithm for non-linear image registration using normalized gradient fields”, in *IEEE 11th International Symposium on Biomedical Imaging (ISBI)*, 2014, pp. 580–583. *Cit. on pp. 8, 21, 23, 44, 106.*

[RKH+13*] J. Rühaak, L. König, M. Hallmann, N. Papenberg, S. Heldmann, H. Schumacher, and B. Fischer, “A fully parallel algorithm for multimodal image registration using normalized gradient fields”, in *IEEE 10th International Symposium on Biomedical Imaging (ISBI)*, 2013, pp. 572–575. *Cit. on pp. 8, 21, 25, 44.*

[TRK+14*] F. Tramnitzke, J. Rühaak, L. König, J. Modersitzki, and H. Köstler, “GPU based affine linear image registration using normalized gradient fields”, in *Seventh International Workshop on High Performance Computing for Biomedical Image Analysis (HPC-MICCAI)*, 2014, pp. 1–10. *Cit. on pp. 8, 21, 44, 83.*

Conference abstracts

[DKM16*] A. Derksen, L. König, and H. Meine, “An alternating image registration approach for large scale bladder deformations in radiation therapy”, in *18th International Conference on the use of Computers in Radiation Therapy (ICCR)*, 2016, pp. 1–2. *Cit. on p. 153.*

[DKMH16*] A. Derksen, L. König, H. Meine, and S. Heldmann, “A joint registration and segmentation approach for large bladder deformations in adaptive radiotherapy”, in *Medical Physics: Proceedings of the AAPM 58th annual meeting*, vol. 43, 2016, p. 3429. *Cit. on p. 153.*

[KDH+15*] L. König, A. Derksen, S. Heldmann, N. Papenberg, J. Modersitzki, and B. Haas, “Deformable image registration with guaranteed local rigidity”, in *Radiotherapy and Oncology: Proceedings of the 3rd ESTRO Forum*, vol. 115, 2015, pp. S197–S198. *Cit. on pp. 8, 141.*

List of symbols

d	ambient space dimension, $d \in \mathbb{N}$
x, y, z	coordinate axes
x	single point, $\mathrm{x} = (\mathrm{x}_1, \dots, \mathrm{x}_d)$, $\mathrm{x} \in \mathbb{R}^d$
x_i	single coordinate component of a point, $\mathrm{x}_i \in \mathbb{R}$
$\Omega_{\mathcal{R}}$	reference image domain, $\Omega_{\mathcal{R}} \subset \mathbb{R}^d$
$\mathcal{R}$	continuous reference (fixed) image function, $\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$
$\mathcal{T}$	continuous template (moving) image function, $\mathcal{T} : \mathbb{R}^d \to \mathbb{R}$
φ	transformation function, $\varphi : \Omega_{\mathcal{R}} \to \mathbb{R}^d$
u	displacement, $u(\mathrm{x}) := \varphi(\mathrm{x}) - \mathrm{x} \in \mathbb{R}^d$
u_i	i-th component of displacement, $u_i(\mathrm{x}) \in \mathbb{R}$
H	Hessian approximation, $H \in \mathbb{R}^{d\bar{m}^{\mathrm{y}} \times d\bar{m}^{\mathrm{y}}}$

Image grid

m_x, m_y, m_z	number of image grid cells for each axis
m	vector of number of grid cells, $m := (m_1, m_2, m_3) = (m_x, m_y, m_z)$
$\bar{m}$	total number of image grid cells
h_x, h_y, h_z	grid spacings of image grid for each axis
h	vector of grid spacings, $h := (h_1, h_2, h_3) = (h_x, h_y, h_z)$
$\bar{h}$	volume of a single image grid cell
$\hat{\imath}$	coordinate index in x-direction $\hat{\imath} = 1, \dots, m_x$
$\hat{\jmath}$	coordinate index in y-direction $\hat{\jmath} = 1, \dots, m_y$
$\hat{k}$	coordinate index in z-direction $\hat{k} = 1, \dots, m_z$
i	linear lexicographical index, $i := \hat{\imath} + \hat{\jmath} m_x + \hat{k} m_x m_y$ for $d = 3$; $i := \hat{\imath} + \hat{\jmath} m_x$ for $d = 2$
$\mathbf{x}$	vector of lexicographically ordered image grid points, $\mathbf{x} = (\mathrm{x}_1, \dots, \mathrm{x}_{d\bar{m}}) \in \mathbb{R}^{d\bar{m}}$
$\mathbf{x}_i$	i-th grid point, $\mathbf{x}_i := (\mathrm{x}_i, \mathrm{x}_{i+\bar{m}})$ for $d = 2$; $\mathbf{x}_i := (\mathrm{x}_i, \mathrm{x}_{i+\bar{m}}, \mathrm{x}_{i+2\bar{m}})$ for $d = 3$

Deformation grid

$m_x^{\mathrm{y}}, m_y^{\mathrm{y}}, m_z^{\mathrm{y}}$	number of deformation grid cells for each axis

$h_x^{\mathrm{y}}, h_y^{\mathrm{y}}, h_z^{\mathrm{y}}$	grid spacings of deformation grid for each axis
$\bar{h}^{\mathrm{y}}$	volume of a single deformation grid cell
$\mathbf{x}^{\mathrm{y}}$	vector of lexicographically ordered deformation grid points, $\mathbf{x}^{\mathrm{y}} = (\mathrm{x}_1^{\mathrm{y}}, \dots, \mathrm{x}_{d\bar{m}^{\mathrm{y}}}^{\mathrm{y}}) \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}$
$\mathbf{x}_i^{\mathrm{y}}$	i-th grid point, $\mathbf{x}_i^{\mathrm{y}} := \left(\mathrm{x}_i^{\mathrm{y}}, \mathrm{x}_{i+\bar{m}^{\mathrm{y}}}^{\mathrm{y}}\right)$ for $d = 2$; $\mathbf{x}_i^{\mathrm{y}} := \left(\mathrm{x}_i^{\mathrm{y}}, \mathrm{x}_{i+\bar{m}^{\mathrm{y}}}^{\mathrm{y}}, \mathrm{x}_{i+2\bar{m}^{\mathrm{y}}}^{\mathrm{y}}\right)$ for $d = 3$
$\mathbf{y}$	transformation evaluated at all points of the deformation grid, $\mathbf{y} = (\mathrm{y}_1, \dots, \mathrm{y}_{d\bar{m}^{\mathrm{y}}}) \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}$, such that $\varphi(\mathbf{x}_i^{\mathrm{y}}) = (\mathrm{y}_i, \mathrm{y}_{i+\bar{m}^{\mathrm{y}}})$ for $d = 2$; $\varphi(\mathbf{x}_i^{\mathrm{y}}) = (\mathrm{y}_i, \mathrm{y}_{i+\bar{m}^{\mathrm{y}}}, \mathrm{y}_{i+2\bar{m}^{\mathrm{y}}})$ for $d = 3$
$\mathbf{u}$	displacement evaluated on the deformation grid, $\mathbf{u} := \mathbf{y} - \mathbf{x}^{\mathrm{y}} \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}$

Grid conversion

P	grid conversion function from deformation grid to image grid, $P : \mathbb{R}^{d\bar{m}^{\mathrm{y}}} \to \mathbb{R}^{d\bar{m}}$
$\hat{\mathbf{y}}$	deformation, converted to image grid, $\hat{\mathbf{y}} := P(\mathbf{y}) \in \mathbb{R}^{d\bar{m}}$
$\hat{\mathbf{y}}_i$	i-th grid point of $\hat{\mathbf{y}}$, with $\hat{\mathbf{y}}_i := (P(\mathbf{y})_i, P(\mathbf{y})_{i+\bar{m}})$ for $d = 2$; $\hat{\mathbf{y}}_i := (P(\mathbf{y})_i, P(\mathbf{y})_{i+\bar{m}}, P(\mathbf{y})_{i+2\bar{m}})$ for $d = 3$

Image evaluation

$R(\mathbf{x})$	reference image evaluated on image grid, $R : \mathbb{R}^{d\bar{m}} \to \mathbb{R}^{\bar{m}}$ with $R(\mathbf{x}) := \mathcal{R}(\mathbf{x}_i)_{i=1,\dots,\bar{m}} \in \mathbb{R}^{\bar{m}}$
R_i	single value of reference image at point i, $R_i := \mathcal{R}(\mathbf{x}_i) \in \mathbb{R}$
$T(P(\mathbf{y}))$	deformed template on image grid, $T : \mathbb{R}^{d\bar{m}} \to \mathbb{R}^{\bar{m}}$, with $T(P(\mathbf{y})) = T(\hat{\mathbf{y}}) := \mathcal{T}(\hat{\mathbf{y}}_i)_{i=1,\dots,\bar{m}} \in \mathbb{R}^{\bar{m}}$
T_i	single value of reference image at point i, $T_i := \mathcal{T}(\hat{\mathbf{y}}_i) \in \mathbb{R}$

Finite difference operators

$\tilde{\nabla} I_i$	forward and backward finite difference gradient approximations of image I at point i, $\tilde{\nabla} I_i : \mathbb{R}^{\bar{m}} \to \mathbb{R}^{2d}$
$\tilde{\Delta}\mathbf{u}_i$	finite difference approximation of the Laplacian operator at point i, $\tilde{\Delta}\mathbf{u}_i : \mathbb{R}^{d\bar{m}^{\mathrm{y}}} \to \mathbb{R}$

1 Introduction and overview

Image registration describes the process of aligning two or more images [LNE11, §1.2] in order to obtain additional information from the result. More precisely: Given a reference and a template image, the goal of image registration is to find a transformation, such that the transformed template is similar to the reference image [FM08].

1.1. Motivation

Medical image registration has been extensively studied over the last decades. In virtually every medical field, image registration, image fusion, or more generally correspondence detection between two images has been used to improve diagnosis, treatment and follow-up [SDP13]. Applications range from motion detection and compensation to intra-operative fusion of different modalities and change detection in longitudinal studies [FDW+15; LPH+09; HD11].

However, accurate current registration models are often complex and computationally demanding, often exhibiting long processing times and requiring large amounts of memory. These requirements can hinder the transition of such advanced algorithms into clinical use, which could otherwise potentially support clinicians in performing diagnosis and treatment [RM03]. Additionally, decreased runtime has the potential to enable completely new use cases for image registration, such as real-time applications.

As computational capabilities of modern hardware are increasing, so are acquired image resolutions, diminishing the hope that hardware speed up will solve runtime problems over time. Furthermore, since the broad availability of multi-core CPUs, special measures such as multi-core parallelization and vectorized computations need to be used to utilize the full potential of modern hardware [Sut05]. As parallelization and vectorization pose different requirements to algorithms than serial computation models, sophisticated solutions are required to increase efficiency.

Over the last decades, various algorithms have been established for solving the registration problem in different ways: *"While a linear transformation can be simply defined as a superposition of rotations, translations, scaling and shear, and is always applied globally, there is no agreement in the image processing community about the best way to define and estimate an elastic transformation"* [CS05]. Especially in algorithms computing non-linear deformations, there does not exist a canonical approach, but different methods are being used throughout the community.

For many of these models performance was also a concern. Different approaches for high-performance computations have been proposed, ranging from shared-memory architectures with multi-core CPUs and distributed memory cluster architectures to specialized implementations on graphics cards [SSKH10a].

In this thesis, we will focus on the *variational image registration* model [Ami94; HCF04; DR04; Mod04; CDH+06]. It has been successfully used in various applications [RHKF13; BMR13; GRB+12; Mod09], and was thoroughly analyzed from a mathematical point of view [Mod04]. It will be introduced in detail in Chapter 2. Based on a continuous formulation of the registration problem, the model allows for theoretical analysis of mathematical properties, such as existence of a minimizer, e.g., [BMR13]. However, implementations of this model are commonly limited to demonstrating the feasibility and accuracy of the approach, rather than focusing on clinically feasible memory consumption or runtimes.

The core of the variational image registration scheme is the numerical minimization of an objective function, composed of an image-driven external energy and an internal regularization force. Using a so-called discretize-then-optimize approach, a derivative-based quasi-Newton optimization is employed to minimize this objective function after discretizing the continuous formulation. This ensures fast convergence to a local minimum of the objective function, but comes at the cost of gradient or even Hessian computations. Implemented in a naïve way, these computations are very memory-intensive and require long computation times, limiting image resolutions and clinical feasibility.

This constitutes the motivation of this thesis. Starting with an established registration framework, the structure of the formulation is analyzed from a mathematical point of view, in order to derive fast and efficient algorithms that are able to fully utilize the capabilities of modern hardware. This so-called *matrix-free* approach is described in Chapter 3. This greatly increases the feasibility of the variational image registration approach in real-world application scenarios with strict computational requirements, ranging from the processing of large-scale datasets to real-time constraints. Some of these applications will be introduced in the following.

1.2. Applications

In 2015, an estimated number of 17.5 million new cases of cancer have been diagnosed worldwide [FAB+17], with a predicted increase in the future [FAB+17; SW14]. After cardiovascular diseases, cancer is the second leading cause of death [WNA+16]. Therefore, investigating applications that support cancer treatment and diagnosis is a highly relevant topic.

Today, medical imaging plays an important role at every stage of cancer diagnosis and treatment. Imaging techniques such as computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET) or ultrasound (US) allow the localization and characterization of tumor tissue. Among others, images support the detection, monitoring and *follow-up diagnosis of cancer in radiology*, as well as *patient-specific treatment in radiotherapy* and *motion compensation in ultrasound imaging*. Each of these three areas of application has its own unique challenges and requirements for the image

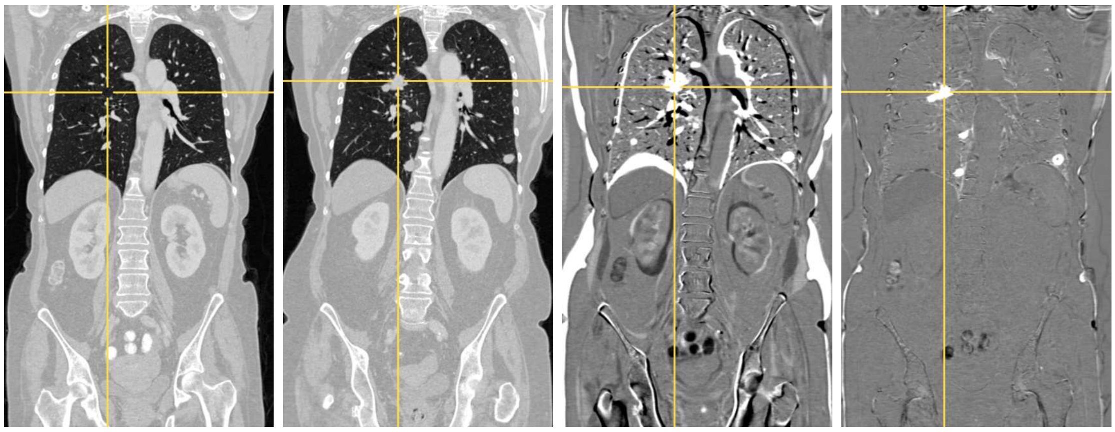

(a) Prior scan (b) Follow-up scan (c) Initial difference (d) After registration

Figure 1.1.: Registration of two three-dimensional follow-up thorax-abdomen CT scans from radiology, using the approach presented in Chapter 6, (a): coronal slice of prior scan, (b): coronal slice of follow-up scan, (c): subtraction image before registration, (d): subtraction image after registration. The most recent scan is deformed onto a prior scan using deformable image registration. The subtraction image after registration clearly highlights the areas of change (white spots in the lung) corresponding to tumor growth. Image courtesy of Radboud University Medical Center, Nijmegen, The Netherlands. Figure adapted from [KRDL18*].

registration component, ranging from processing of large datasets to real-time performance.

1.2.1. Follow-up imaging in radiology

If the treating physician suspects cancer, standard medical procedure is to arrange the acquisition of a CT scan [GSTA10]. This image is then transferred to the radiology department, where it is interpreted, or "read", by a radiologist. The radiologist locates, measures and classifies suspicious lesions in the image and summarizes the findings in a radiology report. In addition to identifying new lesions, changes in comparison to prior images of the same patient have to be assessed. In clinical practice, in this so-called follow-up diagnosis, the clinician typically navigates manually through the slices of the three-dimensional *current image*, and, once a lesion has been identified, repeats the process on the *prior image* in order to identify possible changes in size or appearance of the lesion.

Technical tasks such as navigating through the data can be a lengthy and tiring process and prevent the clinician from performing actual diagnoses. In this setting, image registration can be employed to automatically provide the corresponding location in one image to a specific location in the other image, also known as *cursor synchronization* or *synchronized navigation* [FGM+17; HBS06; Bal06; LPG+05]. Especially in cases where new lesions appeared, finding the exact location of a then non-existent lesion in a prior scan can be time consuming.

A typical area of application is in follow-up diagnosis of thorax-abdomen CT images. Due to the elastic behavior of organs and tissue, a non-linear deformation is required in order to map correspondences correctly. Furthermore, this deformation can be used to provide new types of visualization such as subtraction images, see Figure 1.1 for an example. On state-of-the-art scanners, the resolution of CT scans is approximately 0.7 mm in each direction, leading to large image sizes, sometimes exceeding $512 \times 512 \times 1000$ voxels. In a clinical setting, larger numbers of these images need to be processed within reasonable time, in order to allow timely reading by the radiologist. This constitutes a challenging registration problem involving large numbers of high-resolution datasets, and requires an efficient use of resources, especially available memory. An evaluation of the developed image registration algorithm in this clinical application on a large number of datasets is presented in Chapter 6.

1.2.2. Image-guided treatment in radiotherapy

Once a cancer diagnosis has been made there are several options for treatment and therapy. Generally, over the course of treatment, a combination of these is applied. First, tumor tissue can be removed by oncological surgery. Second, treatment can be performed within the field of medical oncology, using specific drugs to target cancer cells, e.g., by targeted therapy, immunotherapy or chemotherapy [BLYY12]. Third, the tumor can be treated with radiotherapy [BHG04]. Here, we consider image-guided radiotherapy treatment.

Radiotherapy aims to irreparably damage cancerous cells using ionizing radiation with high energy, while healthy tissue is spared as much as possible and allowed to regenerate [BHG04]. For this, the radiation needs to be targeted as accurately as possible, which can either be performed by injecting radioactive sources near the tumor (brachytherapy), or by using external beam radiotherapy. The latter is typically performed by a linear accelerator rotating around the patient, emitting a focused photon beam. In order to determine the optimal dose distribution, a so-called treatment plan is created before treatment. This treatment plan is based on a diagnostic CT image of the patient, where all important anatomy is visible (a so-called *planning CT*) and describes how the radiation is applied to the target area during treatment.

Generally, the treatment is divided into several *fractions*, or sessions, where in each fraction, a part of the planned total dose is applied. The treatment typically takes several weeks [BLYY12]. In order to verify the treatment plan, in image-guided radiotherapy, before application of a fraction a *cone-beam CT* (CBCT) image of the patient can be acquired, while the patient is already in treatment position. This image is then compared with the planning CT to verify patient position and to account for anatomical changes between fractions [JKDM07]. Here, image registration is used to map the planning CT image to the CBCT. The registration allows to propagate contours of important structures, such as the target volume or structures at risk, onto the CBCT image in order to

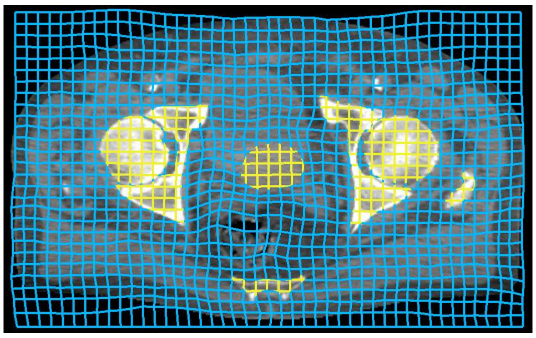


Figure 1.2.: Non-linear deformation grid (blue) with embedded locally rigid areas (yellow) for bones and prostate. A single axial slice of a male pelvis CT image is shown. Local rigidity constraints allow to take the physical properties of bones and rigid structures into account, which avoids unrealistic deformations. The approach is described in Chapter 7.

assess dose distribution. Furthermore, it allows to accumulate applied doses over time for adaptive radiotherapy [JKDM07; THT+16].

However, computing meaningful deformations in this setting can be a challenging task. For example, in the male pelvic area deformation properties of adjacent organs vary greatly between bladder (highly elastic), prostate (almost rigid) and bones (rigid). This requires the computation of high-resolution deformations to obtain physically plausible results. Nevertheless, these deformations might still be unrealistic, when not considering anatomical properties in the registration algorithm. Especially in the pelvic region, implausible elastic deformations of bones are observed in the registration result and large bladder deformations commonly cannot be handled well [TPB+11].

To address this issue, we propose deformable registration with a local rigidity constraint for radiotherapy in Chapter 7. Here, locally rigid areas (bones, prostate) are only allowed to rotate and translate, embedded in a non-linear deformation of the remaining tissue. An example of a non-linear deformation grid with local rigidity constraints is shown in Figure 1.2.

As the rigidity constraint acts on the deformation grid, in order to correctly map the boundaries of rigid areas of with the image anatomy, high-resolution deformations are required. Thus, an efficient algorithm that is capable of computing high deformation resolutions on a clinical workstation is required. Furthermore, in a clinical setting, the algorithm needs to finish rapidly in order not to interfere with clinical workflows, as the registration is performed while the patient is already in treatment position.

Fulfilling both of these requirements, the matrix-free algorithm, developed in Chapter 3 is well-suited for this application. We use the non-linear algorithm as a basis and include the local rigidity constraint. We evaluate the resulting method on ten clinical datasets and compare the results with a state-of-the-art algorithm used in a clinical workstation. Additionally, we compare our deformable registration with and without local rigidity constraints and show that physically more plausible results can be achieved by using local rigidity constraints.

Due to the fast runtimes and low memory requirements of the matrix-free algorithm this novel registration method can be utilized with clinically feasible performance, which could not be realized with matrix-based approaches before.

1.2.3. Real-time ultrasound tracking for motion compensation

In all areas of medical imaging, patient movement is an important factor and has to be carefully taken into account. While in image acquisition, patient motion impacts the image quality and subsequent diagnosis, interventional settings require even more urgent attention to patient motion: For example in radiotherapy, as discussed in the previous section, tumor tissue is intentionally damaged by radiation. A moving target area can therefore cause severe damage to healthy tissue, if patient motion is not accounted for [KMB+06]. Another example is high-intensity focused ultrasound (HIFU), where tissue in a small focal volume of an ultrasound beam is heated to cause cell death [KTC03]. In both cases, motion tracking can potentially make treatment safer.

Ultrasound (US) imaging features several benefits over other image acquisition techniques. It provides real-time image acquisition and has low requirements in component setup [FMB+15]. Furthermore, it uses non-harmful ultrasound waves. A sequence of ultrasound images can be used to determine patient motion, e.g., due to breathing. In these ultrasound sequences, tracking of prominent features over time, such as vessels, can be used to identify motion and to adjust the treatment location accordingly.

Automatically tracking the features is a challenging task. While special characteristics of the ultrasound modality, such as high noise level, make image processing difficult, additionally real-time performance is required in order to correct for patient motion during treatment. For real-time adaptive treatment, determining the motion from one to the following frame must be finished before the next frame arrives. With acquisition rates of 25 Hz or more [DBK+15*], highly efficient algorithms are needed in order to obtain the required processing speed.

To reach this goal, we embedded the developed image registration algorithm in a novel tracking method for liver vessels in ultrasound images, described in Chapter 8. An example of an ultrasound frame with tracked landmarks is shown in Figure 1.3. The tracking algorithm handles the specific challenges of long ultrasound sequences while still achieving real-time performance.

1.3. Contributions and outline

The main contribution of this work consists of a new matrix-free formulation of a variational image registration algorithm, which enables a fast, efficient and parallel computation. The approach is extended and applied to three very different clinical fields, proving the achieved benefits in real-world scenarios.

In detail, the contributions of this work can be divided in two parts:

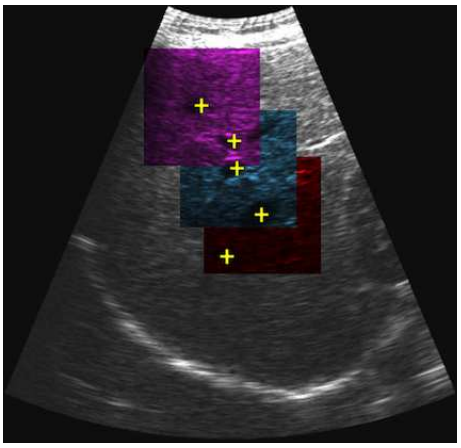


Figure 1.3.: Proposed ultrasound tracking scheme. Five landmarks that are to be tracked (yellow) are visualized on a single liver ultrasound frame shown in the background. The landmarks are placed in the centers of different liver vessels. Additionally shown are three colored windows (purple, blue, red) in which the tracking scheme computes a full, non-linear image registration in each step in real time at up to 131 frames per second (Chapter 8). Figure adapted from [KKR14*].

Derivation of the matrix-free method. In the first part, we develop a *matrix-free algorithm for deformable image registration.* Based on a variational image registration model, the derivative structure of important distance measures and a regularization term is analyzed. With this, we derive new, fully matrix-free computation rules for objective function

- gradient computations, and
- Hessian-vector multiplications.

These methods are systematically analyzed and different implementation alternatives are discussed, balancing runtime and memory usage. Furthermore, the developed methods are employed for deformable registration as well as for rigid and affine transformations. Besides multi-threaded CPU versions, additional implementations on graphics cards and a rigid registration on digital signal processors is presented, showing the flexibility of the approach.

As will be shown, the derivation of the matrix-free method can require a considerable amount of effort. In order to justify this effort, we additionally implement the registration algorithm in a computational framework using *automatic differentiation.* This method automatically computes analytically exact derivatives of a function without further user intervention and allows for automatic optimization of computations and seamless execution on GPUs.

We compare both algorithms with matrix-based approaches and perform evaluations in terms of parallel scalability, runtime and memory requirements, for components of different complexities, ranging from single functions up to a full registration.

Our method has been published in [KR14*; KDHP15*; KRDL18*] for deformable registration and in [RKH+13*; RKT+17*] for rigid and affine deformations and is presented in an extended and more comprehensive form in this thesis. Furthermore, we presented implementations on GPUs in [TRK+14*; RKT+17*; KRDL18*] and on DSPs in [BKR+14*].

Clinical applications. In the second part, the developed methods are used as a basis for different clinical applications. Each of the applications highlights a different aspect of the benefits of the matrix-free approach. Specifically,

- in *follow-up imaging in radiology*, cursor synchronization is utilized to easily obtain corresponding lesions of prior patient scans. The presented approach enables processing of large numbers of datasets, each with large image sizes, to aid the diagnosis by a radiologist,
- in *radiotherapy*, the application targets the physical plausibility of deformations in the male pelvic area, where the registration approach is extended with a local rigidity constraint. The approach is evaluated on three-dimensional clinical datasets and compared with state-of-the-art approaches. The proposed method enables the processing of high deformation resolutions with fast, clinically feasible runtimes and low memory requirements, published in [KDPH16*; KDH+15*], and
- in *liver ultrasound tracking*, the approach is used as a basis for a newly developed real-time tracking method. The fast and parallel execution of the matrix-free method allows to use a full image registration component as part of an application with real-time constraints. The tracking algorithm is based on our work published in [DBK+15*; KKR14*].

Reflecting this structure, the remainder of this thesis is structured as follows. In Chapter 2 a general introduction to image registration and related work is given, followed by a description of the variational model that serves as a basis for this thesis. Important aspects of the discretization and numerical optimization methods are described and necessary definitions and notation are introduced.

Using these definitions, in Chapter 3, the derivative structure of the objective function is analyzed in detail for important distance measures and a regularization term. Based on this analysis, matrix-free computation rules for objective function gradient computations and Hessian-vector multiplications are derived for deformable as well as rigid and affine deformation models. These matrix-free schemes are then analyzed in terms of computational operations as well as memory usage. Specialized implementations on CPU, GPU and DSP are discussed.

In Chapter 4, we present an alternative approach to the matrix-free computations using automatic differentiation. Here, function derivatives are computed fully automatic without further user intervention, which poses a valuable alternative in rapid prototyping and algorithm development. We discuss the fundamentals of algorithmic differentiation and present and evaluate an implementation of the registration objective function.

All derived schemes are analyzed and benchmarked in detail in Chapter 5. Evaluations are performed for scalability, runtime and memory requirements in comparison with approaches from Chapter 3 and Chapter 4 as well as alternative, matrix-based implementations. Further evaluations and comparisons are performed for individual components as well as for the complete objective function and registration algorithm.

In the last part of the thesis, clinical applications and extensions of the proposed method are presented. In Chapter 6, the registration algorithm is used to support follow-up diagnosis of cancer in radiology. Large size and numbers of datasets pose a special challenge here. In Chapter 7, the matrix-free approach is extended with a local rigidity constraint and evaluated on three-dimensional clinical datasets from radiotherapy in comparison with state-of-the-art approaches. In Chapter 8, the matrix-free registration algorithm serves as a basis for a real-time ultrasound tracking method, that is evaluated on a large number of benchmark datasets and compared with other approaches in a public challenge.

Finally, in Chapter 9, we summarize and discuss the findings of this work and future perspectives.

Part I.

Matrix-free approaches for deformable image registration

2 Image registration

This chapter gives a general introduction to the field of image registration, as well as a detailed description of the image registration approach that is utilized in this thesis. After defining the overall registration framework in Section 2.1, a general introduction is given in Section 2.2, where different deformation models, application scenarios and popular image registration methods are discussed. As the developed methods focus on fast and efficient image registration, related approaches with a similar focus are discussed in Section 2.2.4.

After this, important concepts and components of the variational image registration model, which provides the basis for the developed methods, are described in detail in Section 2.3. Following a *discretize-then-optimize* approach, the continuous formulation is discretized in Section 2.4. As the registration problem will be phrased as an optimization problem, implementation requires methods for numerical optimization which are described in Section 2.5. The methods and components in this chapter serve as a basis for further analysis and derivation of matrix-free methods in Chapter 3.

2.1. Registration framework

As outlined in Section 1.1, the goal of image registration is to establish spatial correspondence between images. While in some frameworks, this includes multiple images (time-series, video sequences), in this thesis, we will focus on the pairwise registration of exactly two images.

The first image is referred to as the *reference image* $\mathcal{R}$. This image remains unmodified during the registration process and serves as a basis for comparison with the second image. Therefore, this image is also sometimes called *fixed image*. The second image is denoted as *template image* $\mathcal{T}$. As the name “template” suggests, it is being adapted to match the reference image by applying a *transformation* φ. As the template image is being transformed, it is also sometimes called *moving image*.

The goal of image registration is now to find a reasonable transformation φ, such that the transformed template image becomes “similar” to the reference image [HM06a]. The notation in this section follows the variational model of [Mod04; Mod09].

2.1.1. Images and transformation model

In the variational model, the images are interpreted as scalar-valued, continuous functions

$$\mathcal{R} : \mathbb{R}^d \to \mathbb{R} \quad \text{and} \quad \mathcal{T} : \mathbb{R}^d \to \mathbb{R},$$

with support on reference and template image domains $\Omega_{\mathcal{R}} \subset \mathbb{R}^d$ and $\Omega_{\mathcal{T}} \subset \mathbb{R}^d$ [Mod04; Mod09]. Here, $d \in \mathbb{N}$ indicates the image dimension. The images are mapping a d-dimensional coordinate to a real-valued image intensity. In many applications, where photos or videos are used, it holds $d = 2$, while in medical applications we often have $d = 3$, such as in CT or MRI scans. Other cases also exist, as in the example of time-series of three-dimensional images, where $d = 4$ [CCM+10], or non-scalar images with different color channels. In this thesis, the two most common cases in medical imaging with $d \in \{2, 3\}$ and gray intensity values will be considered. For details on how we obtain continuous image representations from discrete data via image interpolation, see Section 2.4.3.

Given a transformation $\varphi : \Omega_{\mathcal{R}} \to \mathbb{R}^d$, mapping a coordinate location in the reference image domain to the template image domain, a *transformed template* is obtained by the composition $\mathcal{T}(\varphi) := \mathcal{T} \circ \varphi$, where

$$\mathcal{T}(\varphi) : \Omega_{\mathcal{R}} \to \mathbb{R}, \; x \mapsto \mathcal{T}(\varphi(x))$$

maps a point in the reference image domain to a template image gray value. Note that this transformation model uses an Eulerian representation of motion, in which the reference frame is fixed, also sometimes referred to as *backward mapping*. For every point in the reference image domain, a corresponding gray value of the deformed template image can be determined, which allows a direct comparison of $\mathcal{R}$ and $\mathcal{T}(\varphi)$.

2.2. Related work

The comparison of reference image and template image depending on a transformation φ is the common characteristic of most registration approaches. However, various image registration approaches have been proposed in the literature, which also rely on different definitions of images or the transformation model.

Surveys. Image registration is a key component in modern image processing solutions. A substantial number of different registration approaches and frameworks have been proposed in many application areas. Of these, medical imaging is an especially popular field, featuring special challenges such as large, three-dimensional images, non-linear deformations and clinical time constraints. Several recent surveys [SDP13; OT14] systematically categorize the medical image registration approaches, in addition to older surveys in [MV98; WFW+97; MF93; PV93]. Introductions to the field of image registration can also be found in the textbooks [Mod04; Mod09; LNE11; Gos05; HHH01]. A comparison of software packages for medical image registration is given in [KBD17].

Apart from medical applications, there are numerous other fields where image registration is also employed, ranging from visual surveillance [RJ02], automated surface inspection [LK03] and mobile robotics [HNCB10] to remote sensing [LCSC15]. Thus, there also exist broad overviews of image registration, including non-medical applications [ZF03; Bro92]. While each of these areas presents their own interesting problem settings and challenges, this work focuses on applications in the medical field.

In the following sections, the content of this thesis is put in context with important related work. Starting with a general classification of deformation models and applications of medical image registration in Section 2.2.1 and Section 2.2.2, the most popular image registration frameworks will be outlined in Section 2.2.3. This classification will then be used to give an overview of fast and efficient algorithms and implementations for each of these frameworks in Section 2.2.4.

2.2.1. Deformation models

Current image registration approaches can be largely divided into two groups, based on the transformation model [CHH04]. The first group is composed of *affine transformations*, mapping a single d-dimensional point $\mathrm{x} \in \mathbb{R}^d$ with

$$\varphi_w(\mathrm{x}) := A\mathrm{x} + b,$$

where $b \in \mathbb{R}^d$ is a translation vector and $A \in \mathbb{R}^{d \times d}$ is a linear transformation matrix. The transformation is parameterized by the vector $w \in \mathbb{R}^{d^2+d}$, consisting of the d^2 matrix elements from A and the d elements from the vector b. Every point $\mathrm{x} \in \Omega_{\mathcal{R}}$ is deformed with the same transformation, determined by the discrete number of parameters in w, which is why this model is sometimes also called a *parametric* registration.

Affine transformations only allow for rotation, translation, scaling, and shearing of images. While these transformations are rather limited, they have the advantage of preserving an important relation between points in the transformed image: straight lines are mapped onto straight lines. Therefore, they are preferable in some use cases, especially when an alignment of different fields of view is needed [HZ03; JBBS02; BT01; TRU98].

An important sub-class of affine transformations are *rigid transformations*. Further restricting the available degrees of freedom, they only allow for rotations and translations and are often used for image pre-alignment, but are also employed in other applications [FGM+17; RDH+15; RKT+17*; SSKH10b; SGR+08].

The second group is *non-linear transformations*, with the corresponding registration often called a *deformable registration*. In contrast to a rigid or affine transformation, every point is mapped with an individual deformation $\varphi(\mathrm{x}) : \mathbb{R}^d \to \mathbb{R}^d$. The non-linear transformation model allows for more degrees of freedom, driven by geometrically or physically motivated models [SDP13]. Deformable registration methods are capable of capturing complex, non-linear motion. Especially in the medical field, these deformations are ubiquitous in soft tissue and organs. As common in the literature, the terms "deformable registration" and "non-linear registration" will be used interchangeably in this thesis. Compared to the deformable model, the rigid and affine models restrict the set of possible deformations

φ. Therefore, affine models can also be interpreted as special cases of the deformable model.

2.2.2. Application scenarios

The interpretation of the sought-after deformation can vary largely, depending on the application context. A common example is the task of *image fusion* [SDP13], where images from the same object, acquired with different modalities are registered in order to combine them in an overlay image. For example, in the case of positron emission tomography (PET), the images show functional activity, such as increased metabolic activity in a tumor. However, PET images provide no anatomical context. Therefore, they are usually registered with a CT scan, in order to create an overlay image allowing to visualize activity together with patient anatomy [MHV+03]. Registration of functional images and anatomical scans is also often referred to as *co-registration*, especially in neuroimaging.

While in the above example, the goal is to obtain the deformed image, in other areas the transformation itself is directly utilized. In lung imaging, the deformation is used for *motion analysis*. The computed vector field gives information about respiratory motion and air flow in the lung. With this, information about regional lung function can be obtained [EWSH11; WES+10], which is utilized in radiotherapy or diagnosis of Chronic Obstructive Pulmonary Disease (COPD) [MPR+12].

In other areas, *point-to-point correspondence* information between two images is desired. In follow-up diagnosis for oncology, the position of the cursor is linked between two images from different days, which is also called *cursor synchronization* or *synchronized navigation* [FGM+17]. This helps to detect and assess size and appearance changes of tumor lesions.

In *motion tracking*, point locations are propagated from one frame to the next by applying the obtained transformations to a sequence of images, for example in ultrasound tracking [DBK+15*].

In radiotherapy, the obtained deformation is often used for *contour propagation* of organs and structures at risk. Previously delineated contours are transferred from an older planning image onto a newly acquired image of the same patient, in order to locate those structures in the new image and to adapt treatment accordingly [TPB+11].

Further application scenarios exist such as atlas registration [RBMM04] or image subtraction [TVF+02; EPW+07; LGB00], making image registration a highly versatile method with ubiquitous applications in image processing.

This also means that the derivation of a general image registration approach allows for applications in various problem settings without changing the core algorithm. Of the described applications, in this thesis, methods for contour propagation (Chapter 7), motion tracking (Chapter 8) and synchronized navigation (Chapter 6) will be presented.

2.2.3. Image registration methods

While there also exist differences in approaches for rigid and affine algorithms, largely different approaches have been established for deformable image registration [SDP13]. Here, some of the most popular approaches are *demons-type registration* [Thi98], which is derived from the physical paradox of Maxwell's demons; *spline-based free-form deformations* (FFD) [RSH+99; SRQ+01], which are constrained by geometrical properties; and *variational image registration* with physically motivated regularization [Ami94; Mod04]. Given the vast variety of available image registration methods, this list is neither exhaustive nor strict, but provides a rough categorization of the most popular methods. Therefore, in the following, each of the three approaches will shortly be summarized and similarities will be identified, before giving an overview fast and efficient implementations in Section 2.2.4.

Thirion's Demons. The *demons* approach was first presented by Thirion in [Thi95; Thi96; Thi98], and was physically motivated by the paradox of Maxwell's demons from thermodynamics. Subsequently, several approaches were presented in order to embed the algorithm within a mathematical framework [PCA99; CPA99; FM01; VPM+07]. The most popular formulation aims to satisfy the *optical flow* equation [HS81]

$$u(\mathrm{x})^\top \nabla \mathcal{R}(\mathrm{x}) = \mathcal{T}(\varphi(\mathrm{x})) - \mathcal{R}(\mathrm{x}),$$

adding an additional constraint for minimal length of the displacement vector u and numerical stabilization [Thi98]. In an iterative algorithm, the method computes an update of the deformation field with so-called "demon forces"

$$u(\mathrm{x}) = \frac{(\mathcal{T}(\varphi(\mathrm{x})) - \mathcal{R}(\mathrm{x}))\, \nabla \mathcal{R}(\mathrm{x})}{\|\nabla \mathcal{R}(\mathrm{x})\|^2 + (\mathcal{T}(\varphi(\mathrm{x})) - \mathcal{R}(\mathrm{x}))^2}$$

at all image points, followed by a Gaussian smoothing of the deformation and an iterative update of φ. This rather simple formulation allows for a fast computation even for high deformation resolutions, which has led to a large popularity of the algorithm, especially in clinical applications [WDO+05; NBD+09]. Several different variants of the algorithm were proposed [GPL+10], interchanging $\mathcal{R}$ and $\mathcal{T}$ in the update computation or symmetrically including information from both images. The standard demons algorithm was found to produce singularities in the obtained deformations [VRS10]. Therefore, symmetric [YLL+08] and diffeomorphic versions were presented [VPPA07; VPPA09; LAFP13], which use a composition of the obtained displacements in each iteration rather than an additive update as used in [GPL+10]. Further extensions of the algorithm have been presented since then, e.g., incorporating different distance measures such as mutual information [LRS+10] and local correlation coefficient [LAFP13], or surface registration [GMTT13]. In [PGSB16] stopping criteria were explored, finding a comparison strategy of previous objective function values to be favorable. As will be discussed below, some demons-based approaches can be related to special cases of the variational model, however corresponding explicit energy terms are not known for most variants [WES+10].

Free-form deformations. Deformable registration based on so-called *free-form deformations* was introduced in [SC97; RSH+99; SRQ+01]. In contrast to the demons algorithm, the deformation is explicitly modeled via a spline-based transformation model, typically using B-splines. Defining a grid of control points ϕ_{i_1,i_2,i_3} on $\Omega_\mathcal{R}$ with spacings h_1, h_2, h_3, for $d = 3$ the spline-based deformation is parameterized as

$$\varphi_\phi(\mathrm{x}) = \sum_{l=0}^{3} \sum_{m=0}^{3} \sum_{n=0}^{3} B_l(v_1) B_m(v_2) B_n(v_3) \phi_{i_1+l, i_2+m, i_3+n},$$

where $p_k := \lfloor \frac{\mathrm{x}_k}{h_k} \rfloor$, $i_k := p_k - 1$, $v_k := \frac{\mathrm{x}_k}{h_k} - p_k$, and B_k is the k-th cubic B-spline basis function [RSH+99].

The deformation is then optimized over the control points ϕ, either with implicit regularization by the spline parameterization itself or with an additional external regularization term [TAS+06]. The update is derived from an image-based *distance measure* such as the multi-modal normalized mutual information (NMI) [RSH+99; SRQ+01]. Several enhancements of the algorithm have been proposed, adding local rigidity [TSC+00; LMVS04] and volume constraints [RM01; RMBJ03; Sdi08] as well as extensions to the temporal domain [PMR05] and landmark constraints [STU05]. Additionally, modeling of inverse consistency [CJ01; FRD+09] and diffeomorphic deformations [RAH+06; MRD+11], as well as discontinuous deformations [SZP+12] were investigated. Research on different optimization methods was performed in [KSP07], where a Robbins-Monro stochastic gradient descent was found to be beneficial in most examples, at the cost of a large number of (fast) iterations. Generally, free-form deformations explicitly parameterize the deformation using a B-spline model, which allows for a reduced number of parameters in comparison to the image size. This discretization is comparable to the discretize-then-optimize approach as discussed in the next paragraph.

Variational methods. In contrast to the explicit deformation model of the free-form deformation approach, with a discrete amount of control points (also called *parametric* [Mod04]), are *variational image registration* methods [Ami94; HCF04; DR04; Mod04; CDH+06]. Here, as already shown in Section 2.1.1, a deformation function φ is searched for in a function space. Due to this continuous model, these approaches are also referred to as *non-parametric* [Mod04]. In the continuous model, the deformation is typically regularized by a physically inspired internal energy $\mathcal{S}$, also called *regularizer* [SDP13]. The driving external force is modeled by a *distance measure* $\mathcal{D}$ that depends on the images, similar to the free-form deformations. With these two terms, the registration is formulated as an optimization problem

$$\min_{\varphi:\Omega_\mathcal{R}\to\mathbb{R}^d} \mathcal{D}(\mathcal{R}, \mathcal{T}(\varphi)) + \alpha\mathcal{S}(\varphi),$$

where the parameter α balances between distance measure and regularizer, described in detail in Section 2.3. For solving the registration problem using the continuous model, two different approaches have been established [HM04].

In the first approach, the deformation is determined by deriving the Euler-Lagrange equation of the continuous model and then solving the resulting partial differential equations (PDEs). This requires computing optimality conditions from the Gâteaux derivative of distance measure and regularizer, and solving the PDEs by using numerical methods [Mod04]. This approach is often denoted as *optimize-then-discretize*. Since regularization is a core component of the variational model, much effort has been put into the derivation and analysis of different regularizers, such as diffusion [FM01], curvature [FM03b; FM04a; Hen06] and elastic regularization [Mod04]. Additionally, the multi-modal mutual information distance measure has been incorporated [DMM+03; Hel06] and the approach was extended to 4D motion models [EWSH11] and sliding motion [SWHE12]. An approach incorporating guaranteed point-to-point landmark correspondences was presented in [FM04b; FM03a]. In earlier works, related models for elastic [Bro81; BK89] and fluid [Chr94; BG96; CRM96; CJM97] registration have been presented. These have also been formulated in variational frameworks [GB98; FM03c; Mod04].

In the second approach, known as *discretize-then-optimize*, the continuous formulation is first discretized by using methods from numerical integration and – usually – finite differences [HM04; HM06a]. This results in a finite-dimensional optimization problem, which can be solved using well-known methods from numerical optimization, such as quasi-Newton methods [FM08; Mod09] or multi-grid solvers [HM06a; FHW08]. Based on this framework, different constraints were added to the registration framework, based on anatomical knowledge, such as local rigidity for bone movement [HHM09], volume constraints and displacement regularity [HM04; HM07a], mass preservation [GRB+12], mask-alignment [RHKF13] or hyper-elastic regularization [BMR13]. Point-based correspondences were included in [POL+09; PRW+14; RPH+17]. In addition to the mutual information distance measure [Hel06; Mod09], the multi-modal *normalized gradient fields* (NGF) distance measure was introduced in [HM07b]. This approach is pursued in this thesis and described in detail starting from Section 2.3. As will be shown, the model allows for great flexibility in the choice of distance measures, regularizers and transformation models together with a direct minimization of the energy function using numerical optimization.

Connections between approaches. The distinction between the above approaches is not strict, and several relations and connections between them exist. Deformable registration (or image matching) and motion estimation via *optical flow estimation* are closely related problems [SDP13; LC01; HSG10]. As noted before, there are different versions of the demons approach. The *"most suitable version for medical image analysis"* [SDP13] is based on the optical flow constraint. Furthermore, it has been shown in [ZKN10] that registration with the sum of squared differences (SSD) distance measure, diffusion regularization and Gauss-Newton optimization is equivalent to the Horn-Schunck optical flow solution [HS81; HJB+12]. Additionally, the demons approach can be seen as an approximation to fluid registration [BG96]. Also, the demons approach has been connected to variational models [FM01; FM04a; Mod04] and comparative evaluations have been made [WES+10], finding a masked demons registration superior to an SSD-based variational model for lung registration.

Further approaches. Several other approaches exist, that are not easily classified into the categories mentioned above [SDP13]. Related to the fluid approach [CRM96] is the so-called large deformation diffeomorphic metric mapping (LDDMM), which yields diffeomorphic transformations by solving a system of differential equations [BMTY05; HBO09; AF11]. A memory-efficient method of LDDMM has been presented in [PNH+16]. Other approaches use graphical models with efficient discrete optimization strategies [HJBS11; HSP+16; SOPD15; TC07] or evolutionary algorithms [SCD11; RJR00].

2.2.4. Fast and efficient image registration algorithms

Common to all deformable registration approaches is a comparably complex and expensive computation. This can result in limited practical adoption, especially in clinical settings [RM03; CHH04]. One obstacle for this can be runtime, such that the algorithm does not finish within clinically feasible time [SRG10; PDBS08]. Another issue can be limited hardware capability, such that the required accuracy cannot be achieved due to, e.g., memory limitations [PNH+16]. Therefore, much effort has gone into increasing computational efficiency for a variety of image registration solutions.

These approaches can be classified based on the deformation model, following Section 2.2.1: Rigid and affine movement or non-linear (deformable) transformations. Second, they can be differentiated based on the specific hardware platform that is targeted. There exist many approaches for CPU-based systems, mostly distributed cluster environments, but also shared-memory multi-core architectures [SSKH10a]. With the advent of general purpose computing on graphics processing units (GPGPU), a large variety of new implementations and algorithms was presented, focusing on the specific features of GPU many-core architectures [FVW+11]. Additionally, other embedded architectures were explored such as digital signal processors (DSPs) [WK98; BKR+14*] and field programmable gate arrays (FPGAs) [JLR02; CS05; CLQS12].

In the following, the most notable approaches for fast and efficient image registration algorithms will be summarized. An overview and classification of the cited literature is given in Table 2.1. A general overview of image registration with focus on high-performance computing architectures is [SSKH10a], while a more specific overview of GPU-based methods can be found in [FVW+11].

Affine and rigid registration. An early high-performance computing approach for medical image registration was presented in [WJK98], where rigid registration was implemented on a CPU cluster, connected via message passing interface (MPI) for automatic registration of brain scans, achieving a linear speedup. Other approaches for multi-modal affine registration followed, further using distributed cluster architectures with the mutual information (MI) distance measure and genetic optimization [BT01] and a correlation-based block-matching [OSP02] scheme. In [WP06], a derivative-free rigid approach with normalized mutual information (NMI) on a shared memory architecture was presented, allowing for faster and more accurate computations than derivative-free optimization with Powell's method.

A first GPU-based rigid registration using projections was presented in [KCW06], using GPU shaders for computations, in some cases achieving speedup factors up to four, compared with a CPU implementation. Following contributions focused on implementations using the NVIDIA Compute Unified Device Architecture (CUDA) [NVI17] with mutual information [LM08; SSKH10b; VW11] or mean squared error [BB09], reporting speedups between one and two orders of magnitude, also depending on whether they

Method	CPU (multi-core, multi-CPU)	CPU (distributed, cluster)	GPU	FPGA, DSP
Affine, rigid	[WP06] [RKH+13*] [RKT+17*]	[WJK98] [BT01] [OSP02]	[KCW06] [LM08] [BB09] [SSKH10b] [VW11] [LKHC15] [TRK+14*] [RKT+17*]	[WK98] [SHP+08] [CJS03] [BKR+14*]
Demons		[SPA03]	[SKSF07] [SXMO08] [MOXS08] [CH08] [GPL+10]	[CLQS12] [KCC+14]
Free-form deformations	[RM03] [SBL+13]	[IOH05] [ZUC09]	[PDBS08] [PDBS10] [SRG10] [MRT+10] [SKS13] [SBL+13] [IIH14] [EYSL15]	[JLR02]
Variational methods	[Chr98] [KR14*] [KDHP15*] [KRDL18*]	[MLSO01]	[KRDL18*]	
other		[MGB16]	[KDR+06] [LYC08] [RUCH09] [LFKC10]	[CS05] [DS07] [CHZ11]

Table 2.1.: Overview of literature cited in Section 2.2.4, focusing on fast and efficient medical image registration algorithms for different platforms (columns) and registration approaches (rows). Own publications incorporated in this thesis are marked with an (*).

compare to a single- or multi-threaded CPU implementation. The authors of [LKHC15] used a correlation ratio distance and compared different GPU architecture generations, achieving similar speedups. Other hardware has also been utilized for affine and rigid registration, such as digital signal processors (DSPs) [WK98] or field programmable gate arrays (FPGAs) [CJS03; SHP+08].

The aforementioned approaches achieve rewarding speed-ups and efficiently utilize modern high-performance computing topologies. However, none of the above approaches is based on a variational model. Therefore, in [RKH+13*; RKT+17*], we proposed a mathematical approach for fast and efficient affine image registration, based on a variational formulation, using the multi-modal normalized gradient fields distance measure. The approach does not rely on a specific hardware topology, but focuses on re-formulating the mathematical problem. However, it can be specialized to run on different platforms. We presented implementations for multi-threaded CPUs in [RKH+13*], GPUs in [TRK+14*; RKT+17*] and DSPs using the sum of squared differences distance measure in [BKR+14*], as well as in the theses [Ber12; Tra14], jointly supervised by the author of this thesis together with J. Rühaak. This approach will be discussed in detail in Chapter 3.

Deformable registration. Similar to rigid and affine approaches, high-performance strategies have also been employed for deformable registration. Initially, many approaches focused on targeting supercomputers. One of the earliest works implements elastic and fluid deformation models with the SSD distance measure on a massively parallel 128×128 processor computer as well as on a 16-processor workstation [Chr98], finding the multi-processor workstation to be at least four times faster than the massively parallel processor. A multi-threaded implementation of a registration based on free-form deformations with NMI was presented in [RM03]. The method was run on 64 processors of a 128-processor shared-memory system, achieving an average speedup factor of 31.

Instead of using a single machine, similar to many of the rigid and affine approaches, an elastic variational model with SSD was implemented on a distributed memory cluster with 48 PC workstations in [MLSO01], reporting speedups up to a factor of 40. A free-form deformation model with NMI was implemented on a 128 PC cluster [IOH05], reporting speedup factors of about 90. In [ZUC09], the authors implemented an FFD-based registration on a cluster of ten workstations, achieving a speedup of about factor five. Similar to this, also demons registration was implemented on a cluster of 15 PCs in [SPA03], reporting an 11-fold speedup. In [MGB16] a LDDMM registration algorithm was implemented on a supercomputer cluster, using up to 1024 nodes with a focus on processing large images up to 1024^3 voxels by using a distributed-memory approach. In this thesis, we are aiming for algorithms that are suitable for clinical practice and can be executed on a single workstation. Therefore, distributed-memory approaches will not be considered in the following.

Since deformable registration is computationally more demanding than rigid and affine models, their implementation on GPU has been very popular. While early approaches used shader languages for implementation [KDR+06; SKSF07; CH08; LYC08], subsequent approaches almost exclusively relied on the CUDA framework.

The demons approach was implemented on GPU in [CH08; SXMO08; MOXS08] and different variants were compared in [GPL+10], leading to speedup factors of approximately one order of magnitude, compared to CPU implementations.

Comparably, the FFD model has also been implemented on GPUs. Comparisons between a single GPU and a GPU cluster have been performed in [PDBS08; PDBS10], using the mean squared difference distance measure. Similar to the demons approach on GPU, speedups of about one order of magnitude were achieved. Further implementations using NMI [SRG10; MRT+10; SKS13; IIH14] and a logarithmic SSD variant [DDW+16] have been proposed, reporting speedups in the same range. In [IIH14], the authors additionally compare many previous GPU implementations to their approach, giving some insight into the difficulties of comparing published results, generated on different datasets and hardware. A diffeomorphic free-form registration was implemented in [EYSL15], achieving speedups of about 10 compared with a multi-threaded CPU. In [RUCH09] the authors utilized dual GPUs to register two-dimensional histology images and achieved a speedup factor of about seven. Registration with block matching was utilized in [LFKC10] and a 8-node CPU cluster was compared to a GPU implementation achieving a four times faster computation on the GPU. In contrast to the other approaches, [SBL+13] performed an implementation in OpenCL and relied on optimization of selected components both on GPU and on CPU, resulting in speedup factors from 2 to 60 for individual components.

Specialized implementations on FPGAs were presented in [JLR02; CS05; DS07; CHZ11; CLQS12; KCC+14].

Similar to the rigid and affine deformation model, out of all approaches discussed above, only [Chr98; MLSO01] use a variational approach. However, they only use the mono-modal SSD distance measure and focus on technical details of the distributed computing environment. Therefore, we extended the matrix-free, fast and efficient computation scheme from affine to deformable registration on multi-core systems [KR14*; KDHP15*; KRDL18*]. In addition to multi-core CPUs, a GPU implementation was presented in [KRDL18*] as well as in the Master's thesis [Mei16], jointly supervised by the author of this thesis together with J. Rühaak. A more comprehensive description of this approach can be found in Chapter 3.

2.3. Variational model

In this thesis, we focus on the variational registration model

$$\min_{\varphi:\Omega_{\mathcal{R}}\to\mathbb{R}^d} \mathcal{J}(\varphi) \tag{2.1}$$

with a functional $\mathcal{J}:\varphi\to\mathbb{R}$ defined as

$$\mathcal{J}(\varphi) := \mathcal{D}(\mathcal{R},\mathcal{T}(\varphi)) + \alpha\mathcal{S}(\varphi),$$

where φ denotes the transformation and $\mathcal{R}$, $\mathcal{T}$ denote the reference and template image as defined in Section 2.1. The functional $\mathcal{J}$ is composed of two main parts, a *distance measure* $\mathcal{D}$, and a *regularizer* $\mathcal{S}$. The distance measure $\mathcal{D}(\mathcal{R},\mathcal{T}(\varphi))$ compares the images depending on the transformation and drives the registration as a so-called external force. However, minimizing solely the distance measure is an underdetermined problem and *ill-posed* [FM08]. Therefore, the regularizer $\mathcal{S}(\varphi)$ is added as an “internal force”, only depending on the transformation itself. The regularizer ensures “reasonable” transformations, often motivated by physical models of motion and enforcing sufficiently smooth transformations. The weighting parameter $\alpha > 0$ can be used to balance between both criteria to achieve a trade-off between image similarity and transformation regularity.

This model gives a great amount of freedom regarding the choice of distance measures and regularizers and many different approaches have been proposed [SDP13], see also Section 2.2.

2.3.1. Distance measures

The distance measure represents a mathematical formulation of “image similarity”. In this work, since we are solving a minimization problem in (2.1), smaller values of the distance measure correspond to more similar images.

The existing distance measures can be roughly divided in two classes. In this thesis, two different measures will be described, each representing one of the classes.

The first class consists of so-called *mono-modal* distance measures. These distances rely on a direct comparison of image intensities, thus requiring equal acquisition equipment and conditions. The most popular mono-modal distance measure is the *sum of squared differences* (SSD), which will be described in the next section. Further mono-modal measures include SSD variants, such as the mean square difference [PDBS08], sum of absolute differences [HBHH01] or logarithmic squared differences [DDW+16].

The second class consists of *multi-modal* distance measures, which are better suited for comparing images that originate from different acquisition devices, such as CT and MRI, or that have different or non-standardized intensity distributions, such as in lung intensity changes in CT due to breathing, or cone-beam CT images. In this work, the *normalized gradient fields* (NGF) distance measure [HM05] is described in detail, which focuses on the alignment of image gradients. Further multi-modal distance measures exist, such as the popular (normalized) mutual information [VW97; CMD+95; MCV+97], which originates from information theory or correlation-based distances [RMAP98; SDP13]. See also Section 2.2 and [KBD17; SSKH10a] for an overview of commonly used distance measures.

Sum of squared differences

The SSD distance measure [Mod04] is defined as

$$\mathcal{D}^{\mathrm{SSD}}(\varphi) := \int_{\Omega_{\mathcal{R}}} \left(\mathcal{T}(\varphi(\mathrm{x})) - \mathcal{R}(\mathrm{x})\right)^2 \mathrm{dx}. \tag{2.2}$$

It is a simple and fast, easily differentiable, mono-modal distance measure, allowing for a direct, point-wise comparison of image gray values. However, especially in medical imaging, where commonly images from different modalities and devices need to be registered, multi-modal distances are often preferred. In this work, the SSD will be employed in an ultrasound tracking application, described in Chapter 8 in combination with the multi-modal NGF, which will be described in the following section.

Normalized gradient fields

In order to create a *"simple and robust alternative"* [HM05] to mutual information, the multi-modal NGF was proposed in [HM05; HM06b; HM07b]. A related approach was proposed in [DR04] using morphological matching including the alignment of normal fields. Defining the gradient of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ at point x as

$$\nabla f(\mathrm{x}) := \left(\frac{\partial f(\mathrm{x})}{\partial \mathrm{x}_1}, \ldots, \frac{\partial f(\mathrm{x})}{\partial \mathrm{x}_d}\right)^\top \in \mathbb{R}^d,$$

the NGF is formally given as

$$\mathcal{D}^{\mathrm{NGF}}(\varphi) := \int_{\Omega_{\mathcal{R}}} 1 - \left(\frac{\langle \nabla\mathcal{T}(\varphi(\mathrm{x})), \nabla\mathcal{R}(\mathrm{x})\rangle + \tau\varrho}{\|\nabla\mathcal{T}(\varphi(\mathrm{x}))\|_\tau \|\nabla\mathcal{R}(\varphi(\mathrm{x}))\|_\varrho}\right)^2 \mathrm{dx}, \tag{2.3}$$

where $\| \cdot \|_\varepsilon := \sqrt{\langle \cdot, \cdot \rangle + \varepsilon^2}$.[1] The parameters τ, ϱ are *edge filtering parameters*, that can be used to filter for image noise. The higher these values, the less influence do smaller gradients have in computing the distance. The NGF is based on the assumption, that even in images of different modalities, *intensity changes* still take place at *corresponding* locations. The formulation in (2.3) can also be interpreted as the approximation

$$\mathcal{D}^{\mathrm{NGF}}(\varphi) \approx \int\limits_{\Omega_{\mathcal{R}}} 1 - \cos^2 \theta(\mathrm{x}) \, \mathrm{dx} = \int\limits_{\Omega_{\mathcal{R}}} \sin^2 \theta(\mathrm{x}) \, \mathrm{dx},$$

where $\theta(\mathrm{x})$ denotes the angle between $\nabla \mathcal{T}(\varphi(\mathrm{x}))$ and $\nabla \mathcal{R}(\mathrm{x})$. Thus, the NGF measures the squared sine of the angle between normalized reference image gradients and transformed template image gradients at all points of the reference image domain. It assumes its minimum value when all gradients are aligned with either an angle of $0°$ (parallel) or $180°$ (antiparallel). Although analytically equivalent, in the discretization, discussed in Section 2.4.4, for numerical reasons using the cosine approximation in (2.3) is preferable to an approximation to the sine formulation [Pap08, §6.3.1]. The NGF has been successfully used in various medical applications such as lung registration [RHKF13; PNH+16], radiotherapy [KDPH16*], histology [LOM+16] and nuclear medicine [RKH+13*], as well as in non-medical applications such as remote sensing [LCSC15].

2.3.2. Regularizer

As described in Section 2.2.3 and Section 2.3, regularization is a central component of the variational model, typically favoring specific properties of the transformation, such as smoothness, or a deformation that conforms to a specific physical motion model. Different approaches have been presented, most are based on simplifications of physical models for elastic or viscous motion [FM01; FM04a; BMR13; Mod04].

The regularizer is acting as a so-called *internal force* on the solution of the optimization problem (2.1): Its value does not depend on the image or other input data, but solely on the *displacement*

$$u(\mathrm{x}) := \varphi(\mathrm{x}) - \mathrm{x}.$$

In this thesis, *curvature regularization* will be described, a regularizer based on second order derivatives of the displacement. Despite the existence of more sophisticated motion models such as [BMR13], this regularizer has proven to be successful in practical applications and public benchmarks [DBK+15*; RHKF13; RPH+17; KDPH16*]. Furthermore, as will be shown in Chapter 3, its specific structure allows for efficient computation.

[1] In this work, we focus on computational issues rather than functional-analytic questions; therefore, we generally assume that all functions satisfy the differentiability and integrability conditions dictated by the model.

Curvature regularizer

Curvature-based regularization was first presented in an image registration context in [FM03b]. Given a single component function of the displacement $u_j : \mathbb{R}^d \to \mathbb{R}$ with $u =: (u_1, \ldots, u_d)$, the curvature regularizer can be defined as

$$\mathcal{S}(\varphi) := \int\limits_{\Omega_\mathcal{R}} \sum_{j=1}^{d} (\Delta u_j(\mathrm{x}))^2 \ \mathrm{dx}, \tag{2.4}$$

where

$$\Delta u_j(x) := \sum_{i=1}^{d} \frac{\partial^2 u_j(x)}{\partial \mathrm{x}_i^2}$$

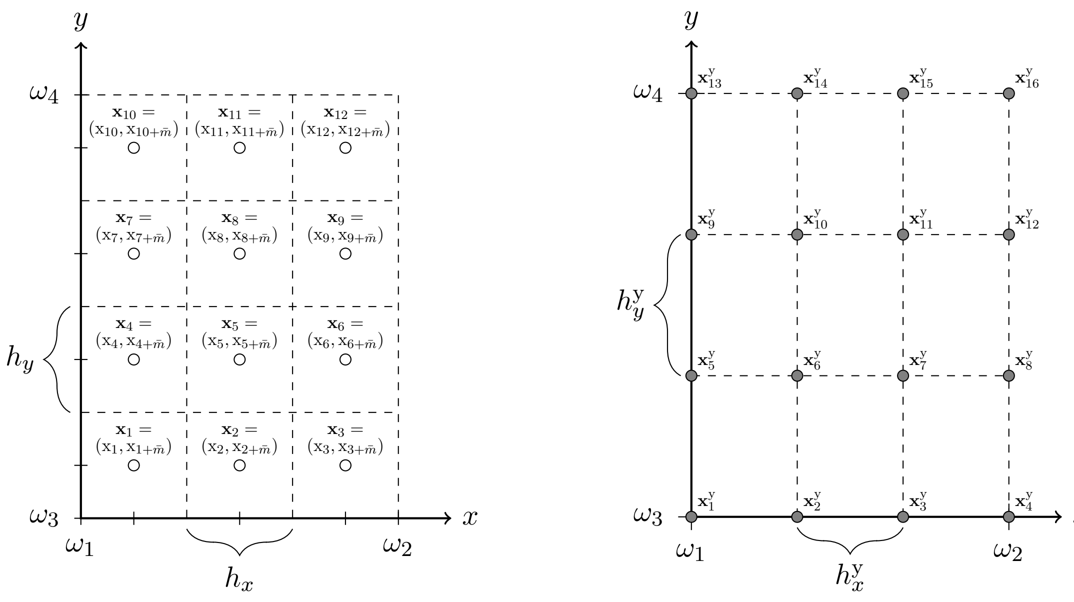
is the Laplacian of the displacement component function u_j at point x. Based on the L^2-norm of second-order derivatives of the displacement, the curvature regularizer particularly favors smooth deformations and is directly related to the thin-plate-spline bending energy [FM03b]. Furthermore, its kernel contains all affine transformations, which makes registration with curvature regularization less dependent on an exact pre-alignment [FM04a]. As the kernel also contains harmonic functions, additionally using mixed second order partial derivatives is proposed in [Hen06]. Since this requires numerically more involved boundary conditions [Hel06, §4.2.3], this approach is not considered in this work.

2.4. Discretization

As described in Section 2.2.3, two main approaches for minimizing the objective function (2.1) of the variational image registration model exist. In the first approach, denoted as *optimize-then-discretize* approach, the necessary conditions for a minimizer are determined by deriving the Euler-Lagrange equations of the continuous objective functional. These PDEs are then solved numerically, using, e.g., time-stepping methods or fixed point iteration schemes [FM04a].

The second approach is known as *discretize-then-optimize* and discretizes the continuous objective directly. The resulting finite-dimensional optimization problem is then solved using methods from numerical optimization. The benefits of both approaches are discussed in [Pap08, §2.2.2] and [Mod09, §2.2.1].

In this thesis, we aim for a fast and memory-efficient method, and therefore follow the *discretize-then-optimize* approach of [Mod09], which has proven to be suitable for different real-world registration solutions [RPH+17; KDPH16*; BKR+14*; LOM+16; LHL+16]. Therefore, in the following, discretized versions of distance measures and regularizer will be derived and described in detail. While technical at times, it will turn out that a detailed understanding of the discretization for the derivative components is crucial for obtaining efficient matrix-free computation schemes in Chapter 3. A list of symbols including important notation in can be found at the beginning of this thesis.


y
ω_4
$\mathbf{x}_{10} = (\mathrm{x}_{10}, \mathrm{x}_{10+\bar{m}})$
$\mathbf{x}_{11} = (\mathrm{x}_{11}, \mathrm{x}_{11+\bar{m}})$
$\mathbf{x}_{12} = (\mathrm{x}_{12}, \mathrm{x}_{12+\bar{m}})$
$\mathbf{x}_{7} = (\mathrm{x}_{7}, \mathrm{x}_{7+\bar{m}})$
$\mathbf{x}_{8} = (\mathrm{x}_{8}, \mathrm{x}_{8+\bar{m}})$
$\mathbf{x}_{9} = (\mathrm{x}_{9}, \mathrm{x}_{9+\bar{m}})$
h_y
$\mathbf{x}_{4} = (\mathrm{x}_{4}, \mathrm{x}_{4+\bar{m}})$
$\mathbf{x}_{5} = (\mathrm{x}_{5}, \mathrm{x}_{5+\bar{m}})$
$\mathbf{x}_{6} = (\mathrm{x}_{6}, \mathrm{x}_{6+\bar{m}})$
$\mathbf{x}_{1} = (\mathrm{x}_{1}, \mathrm{x}_{1+\bar{m}})$
$\mathbf{x}_{2} = (\mathrm{x}_{2}, \mathrm{x}_{2+\bar{m}})$
$\mathbf{x}_{3} = (\mathrm{x}_{3}, \mathrm{x}_{3+\bar{m}})$
ω_3
x
ω_1
ω_2
h_x
y
ω_4
$\mathbf{x}^{\mathrm{y}}_{13}$ $\mathbf{x}^{\mathrm{y}}_{14}$ $\mathbf{x}^{\mathrm{y}}_{15}$ $\mathbf{x}^{\mathrm{y}}_{16}$
$\mathbf{x}^{\mathrm{y}}_{9}$ $\mathbf{x}^{\mathrm{y}}_{10}$ $\mathbf{x}^{\mathrm{y}}_{11}$ $\mathbf{x}^{\mathrm{y}}_{12}$
h^{y}_y
$\mathbf{x}^{\mathrm{y}}_{5}$ $\mathbf{x}^{\mathrm{y}}_{6}$ $\mathbf{x}^{\mathrm{y}}_{7}$ $\mathbf{x}^{\mathrm{y}}_{8}$
$\mathbf{x}^{\mathrm{y}}_{1}$ $\mathbf{x}^{\mathrm{y}}_{2}$ $\mathbf{x}^{\mathrm{y}}_{3}$ $\mathbf{x}^{\mathrm{y}}_{4}$
ω_3
x
ω_1
ω_2
h^{y}_x

(a) Cell-centered image grid

(b) Nodal deformation grid

Figure 2.1.: Different grids, discretizing the same domain $\Omega_{\mathcal{R}}$ for $d = 2$ with visualization of the element ordering in $\mathbf{x}$ and $\mathbf{x}^{\mathrm{y}}$, (a): Image grid with cell-centered discretization and size $m = (3, 4)$. The grid points are located at the center of the cells (white circles), (b): Deformation grid with nodal discretization and size $m^{\mathrm{y}} = (4, 4)$. The grid points are located at the corners of the cells (gray circles).

2.4.1. Discretization grids

To approximate the continuous formulations of the objective function $\mathcal{J}$ and transformation φ using quadrature methods, they are discretized on regular grids. These grids consist of equidistantly distributed points in the cuboid (rectangular for $d = 2$) image domain

$$\Omega_{\mathcal{R}} := (\omega_1, \omega_2) \times \ldots \times (\omega_{2d-1}, \omega_{2d}) \subset \mathbb{R}^d. \tag{2.5}$$

Different discretization grids are used in the literature, differing mainly in the location of the grid points in the discretized domain. In this work, so-called *cell-centered* and *nodal* grids will be used, that will be described in the following sections. Depending on the discretization of, e.g., finite difference operators, other variants such as staggered grids can be useful [HM06a]. A further discussion on different discretization grids can be found in [Mod09, §8.3.1] and [Pap08, §4].

Image grid. The reference image $\mathcal{R}$ is discretized on a so-called *cell-centered* grid, which will be denoted as the *image grid* in the following. The number of grid points in each dimension is defined as $m := (m_x, m_y, m_z)$ in x-, y-, and z-direction with grid spacings $h := (h_x, h_y, h_z)$, where applicable. The total number of grid points and grid cells is then defined as the product $\bar{m} := m_1 \cdots m_d$ with cell volume $\bar{h} := h_1 \cdots h_d$. With the cartesian product $\hat{m} := \{1, \ldots, m_1\} \times \ldots \times \{1, \ldots, m_d\}$ and $\hat{i} \in \hat{m}$, the points are located at the *cell centers* with coordinates

$$\mathbf{x}_i := \left(\omega_1 + \hat{i}_1 h_1 - \frac{h_1}{2}, \ldots, \omega_{2d-1} + \hat{i}_d h_d - \frac{h_d}{2} \right) \tag{2.6}$$

using the linear lexicographical mapping $i := \hat{i}_1 + \hat{i}_2 m_x + \hat{i}_3 m_x m_y$ (or $i := \hat{i}_1 + \hat{i}_2 m_x$ for $d = 2$, respectively), which results in a vector

$$\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_{d\bar{m}}) \in \mathbb{R}^{d\bar{m}}.$$

Consequently, the grid vector $\mathbf{x}$ is composed of all the coordinate components of all grid points arranged *lexicographically*, such that first all x-, then all y- and finally all z-coordinates are stored. This will become important when deriving corresponding matrix structures in Chapter 3. A single point can be recovered from $\mathbf{x}$ as

$$\mathbf{x}_i = \left(\mathrm{x}_i, \ldots, \mathrm{x}_{i+(d-1)\bar{m}} \right) \in \mathbb{R}^d.$$

See Figure 2.1(a) for a visualization of the discretization grid and the grid point ordering in $\mathbf{x}$.

Using these definitions, the discretized reference image, evaluated on all image grid points, can now be written as

$$R(\mathbf{x}) := (\mathcal{R}(\mathbf{x}_i))_{i=1,\ldots,\bar{m}} \in \mathbb{R}^{\bar{m}}, \tag{2.7}$$

with $R : \mathbb{R}^{d\bar{m}} \to \mathbb{R}^{\bar{m}}$, resulting in a vector of image intensities.

Deformation grid. In a similar way, the transformation φ is discretized on a *deformation grid* $\mathbf{x}^{\mathrm{y}}$, resulting in a discretized transformation $\mathbf{y}$. This discretized transformation on the deformation grid will also be denoted as the *deformation* in the following.

Instead of locating the coordinates at the cell-centers, the transformation is discretized on a *nodal* grid, where the grid points are located at the *cell corners*. All quantities use the same notation as their counterparts for the image grid, but with an additional superscript y. Note that the deformation grid resolution can be chosen independently from the image grid.

The deformation grid consists of $m^{\mathrm{y}} := (m_x^{\mathrm{y}}, m_y^{\mathrm{y}}, m_z^{\mathrm{y}})$ grid points in each direction with grid spacings $h^{\mathrm{y}} := (h_x^{\mathrm{y}}, h_y^{\mathrm{y}}, h_z^{\mathrm{y}})$, resulting in $\bar{m}^{\mathrm{y}} := m_1^{\mathrm{y}} \cdots m_d^{\mathrm{y}}$ grid points in total with a cell volume of $\bar{h}^{\mathrm{y}} := h_1^{\mathrm{y}} \cdots h_d^{\mathrm{y}}$. Similar to above, but now having nodal coordinates, with $\hat{m}^{\mathrm{y}} := (0, \ldots, m_1^{\mathrm{y}} - 1) \times \ldots \times (0, \ldots, m_d^{\mathrm{y}} - 1)$ and $\hat{i}^{\mathrm{y}} \in \hat{m}^{\mathrm{y}}$, the grid points are located at

$$\mathbf{x}^{\mathrm{y}}_{i^{\mathrm{y}}} := \left(\omega_1 + \hat{i}_1^{\mathrm{y}} h_1^{\mathrm{y}}, \ldots, \omega_{2d-1} + \hat{i}_d^{\mathrm{y}} h_d^{\mathrm{y}} \right),$$

using the linear mapping $i^{\mathrm{y}} := \hat{i}_1^{\mathrm{y}} + \hat{i}_2^{\mathrm{y}} m_x^{\mathrm{y}} + \hat{i}_3^{\mathrm{y}} m_x^{\mathrm{y}} m_y^{\mathrm{y}}$ (or $i^{\mathrm{y}} := \hat{i}_1^{\mathrm{y}} + \hat{i}_2^{\mathrm{y}} m_x^{\mathrm{y}}$ for $d = 2$, respectively); see Figure 2.1(b) for a visualization and note the difference to (2.6). Again, this orders the coordinates lexicographically in a vector

$$\mathbf{x}^{\mathrm{y}} = (\mathrm{x}_1^{\mathrm{y}}, \ldots, \mathrm{x}^{\mathrm{y}}_{d\bar{m}^{\mathrm{y}}}) \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}},$$

where the individual grid points in the deformation grid are

$$\mathbf{x}_i^{\mathrm{y}} = \left(\mathrm{x}_i^{\mathrm{y}}, \ldots, \mathrm{x}_{i+(d-1)\bar{m}^{\mathrm{y}}}^{\mathrm{y}}\right) \in \mathbb{R}^d.$$

This can now be used to define the *deformation*

$$\mathbf{y} := (\mathrm{y}_1, \ldots, \mathrm{y}_{d\bar{m}^{\mathrm{y}}}) \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}},$$

which represents the transformation φ, evaluated at all points of $\mathbf{x}^{\mathrm{y}}$, ordered so that

$$\varphi(\mathbf{x}_i^{\mathrm{y}}) = (\mathrm{y}_i, \ldots, \mathrm{y}_{i+(d-1)\bar{m}^{\mathrm{y}}}).$$

Again, the transformed coordinates in $\mathbf{y}$ are ordered lexicographically, such that first all x-, then all y- and finally all z- coordinates of the transformed deformation grid points are stored in $\mathbf{y}$.

The nodal discretization of the deformation grid ensures that an image grid point is always surrounded by deformation grid points, even close to the boundary. This enables fast conversion schemes between deformation grid and image grid, that will be described in detail in Chapter 3.

The choice of two different grids for discretization of images and transformation allows to independently choose different discretization resolutions. In particular, it allows to choose a coarser discretization for the transformation, while still evaluating the images at full resolution. Since, as will be described in Section 2.5, the deformation resolution determines the size of the system of linear equations that needs to be solved during the registration, this can have a large influence on the registration runtime.

We always choose the deformation size such that $m_k^{\mathrm{y}} \leq m_k + 1, k = 1, \ldots, d$, i.e., the grid spacing of the deformation grid h^{y} is always equal to or coarser than the grid spacing of the image grid h.

Grid conversion. The two different discretization grids require an additional grid conversion step in order to evaluate the deformed template image $\mathcal{T}(\varphi)$ on the same grid as the discretized reference image $R(\mathbf{x})$.

In order to do so, we define a linear interpolation function

$$P : \mathbb{R}^{d\bar{m}^{\mathrm{y}}} \to \mathbb{R}^{d\bar{m}}, \tag{2.8}$$

which maps the deformation grid to the image grid by using bi- or trilinear interpolation for $d = 2$ or $d = 3$, respectively. With this, the discretized deformed template image, evaluated at all points of the deformation, can be defined as

$$T(P(\mathbf{y})) := (\mathcal{T}(\hat{\mathbf{y}}_i))_{i=1,\ldots,\bar{m}} \in \mathbb{R}^{\bar{m}}, \tag{2.9}$$

where $T : \mathbb{R}^{d\bar{m}} \to \mathbb{R}^{\bar{m}}$ and $\hat{\mathbf{y}}$ is the deformation on the image grid,

$$\hat{\mathbf{y}}_i := \left(P(\mathbf{y})_i, \ldots, P(\mathbf{y})_{i+(d-1)\bar{m}}\right) \in \mathbb{R}^{d\bar{m}}.$$

We now have $R(\mathbf{x}) \in \mathbb{R}^{\bar{m}}$ as well as $T(P(\mathbf{y})) \in \mathbb{R}^{\bar{m}}$, such that reference and deformed template image can be directly compared, depending on the deformation $\mathbf{y}$. Discretized versions of suitable distance measures will be described in Section 2.4.4.

2.4.2. Rigid and affine transformation models

As noted in Section 2.2.1, the affine transformation model can be interpreted as a special case of the deformable model. In the discretized setting, the deformation $\mathbf{y}$ is replaced by a parameter $w \in \mathbb{R}^{d^2+d}$, and the the grid conversion function $P(\mathbf{y})$ in (2.9) by a "grid-generating" function $\hat{\mathbf{y}} : \mathbb{R}^{d^2+d} \to \mathbb{R}^{d\bar{m}}$ with

$$\hat{\mathbf{y}}(w) := \left(\left[(A\mathbf{x}_1 + b)_1, \dots, (A\mathbf{x}_{\bar{m}} + b)_1\right], \dots, \left[(A\mathbf{x}_1 \; + b)_d, \dots, (A\mathbf{x}_{\bar{m}} + b)_d\right]\right)^\top, \quad (2.10)$$

where the new parameters $w = (a_1, \dots, a_{d^2}, b_1, \dots, b_d) \in \mathbb{R}^{d^2+d}$ consist of the entries from $A \in \mathbb{R}^{d\times d}$ and $b \in \mathbb{R}^d$. The function $\hat{\mathbf{y}}$ maps the affine transformation parameters to a deformed image grid. When further restricting the transformation to a rigid transformation, the matrix A is chosen as a d-dimensional rotation matrix, parameterized by a single rotation parameter for $d = 2$ and three rotation parameters for $d = 3$.

2.4.3. Image interpolation

In order to compute the deformed template image $T(P(\mathbf{y}))$, it must be possible to evaluate the template image on *any* point of the continuous domain $\Omega_{\mathcal{T}}$. However, in practice, images are commonly given as discrete image data, represented by equidistantly sampled pixel ($d = 2$) or voxel ($d = 3$) gray values $\mathbf{R}_i \in \Omega_{\mathcal{R}}$, $\mathbf{T}_j \in \Omega_T$, given on cell-centered data grids $\mathbf{x}^{\mathrm{R}} \in \mathbb{R}^{m^{\mathrm{R}}}, \mathbf{x}^{\mathrm{T}} \in \mathbb{R}^{m^{\mathrm{T}}}$, respectively. To transfer these data values to a continuous representation, we use d-dimensional image interpolation, with

$$\mathcal{R}(\mathbf{x}_i^{\mathrm{R}}) = \mathbf{R}_i,\ i = 1, \dots, \mathbb{R}^{m^{\mathrm{R}}}, \qquad \mathcal{T}(\mathbf{x}_j^{\mathrm{T}}) = \mathbf{T}_j,\ j = 1, \dots, \mathbb{R}^{m^{\mathrm{T}}} \quad (2.11)$$

at the data points.

Various interpolation models are used in the literature, ranging from nearest neighbor and linear interpolation to different spline-based models [LGS99]. In order to ensure continuity as well as efficient evaluation, in this work, we use standard *bi-linear* ($d = 2$) and *tri-linear* interpolation [Mod09, §3.3], which is a commonly applied interpolation model in medical image registration [SSKH10a]. Dirichlet boundary conditions are used, based on the assumption that medical images often contain a black (zero) background.

As linearly interpolated images are not always differentiable at the data grid points, from a mathematical point of view, more sophisticated schemes are desirable. However, these often also involve computationally more expensive models [LGS99]. Therefore, as a compromise to facilitate fast and efficient computation, we use one-sided limits to obtain derivative approximations at the grid points, whenever image derivatives are required.

Note that in this work, generally $\mathbf{x} = \mathbf{x}^{\mathrm{R}}$, such that the image grid equals the data grid of the reference image. Thus, no interpolation is required when evaluating $R(\mathbf{x})$ on the (un-deformed) image grid. However, the evaluation of $T(\hat{\mathbf{y}})$ heavily relies on image interpolation for evaluating the template image at the transformed coordinates, which typically do not coincide with the data points.

2.4.4. Distance measures

Using the notation above, the distance measures that were presented in Section 2.3.1 can now be discretized.

Sum of squared differences

Discretizing the SSD distance measure from (2.2) on the cell-centered image grid is straightforward using the midpoint quadrature rule with

$$\mathcal{D}^{\mathrm{SSD}}(\varphi) \approx D^{\mathrm{SSD}}(\mathbf{y}) := \bar{h} \sum_{i=1}^{\bar{m}} \left(T_i(P(\mathbf{y})) - R_i(\mathbf{x})\right)^2, \tag{2.12}$$

where T_i and R_i denote the i-th component function of the linearized image functions described in (2.9) and (2.7), respectively.

Normalized gradient fields

The discretization of the NGF in (2.3) is performed similarly to the SSD by using the midpoint quadrature rule with

$$\mathcal{D}^{\mathrm{NGF}}(\varphi) \approx D^{\mathrm{NGF}}(\mathbf{y}) := \bar{h} \sum_{i=1}^{\bar{m}} \left(1 - \left(\frac{\frac{1}{2}\langle \tilde{\nabla} T_i(P(\mathbf{y})), \tilde{\nabla} R_i(\mathbf{x})\rangle + \tau\varrho}{\|\tilde{\nabla} T_i(P(\mathbf{y}))\|_\tau \|\tilde{\nabla} R_i(\mathbf{x})\|_\varrho}\right)^2\right), \tag{2.13}$$

where $\tilde{\nabla} R$, $\tilde{\nabla} T$ are discretized image gradients, that will be defined in the following. The factor $\frac{1}{2}$ in the numerator also results from this discretization and will become clear at the end of this section.

Let $I \in \mathbb{R}^{\bar{m}}$ be a discretized image. We now define finite difference operators $\tilde{\nabla}_- I_i : \mathbb{R}^{\bar{m}} \to \mathbb{R}^d$ for *backward differences* and $\tilde{\nabla}_+ I_i : \mathbb{R}^{\bar{m}} \to \mathbb{R}^d$ for *forward differences*. These difference operators approximate the gradient of the discretized image at grid point i and can be written as

$$\tilde{\nabla}_- I_i := \left(\frac{I_i - I_{i_{-x}}}{h_x}, \frac{I_i - I_{i_{-y}}}{h_y}, \frac{I_i - I_{i_{-z}}}{h_z}\right) \tag{2.14}$$

for backward differences with $d = 3$, $\tilde{\nabla}_- I_i := \left(\frac{I_i - I_{i_{-x}}}{h_x}, \frac{I_i - I_{i_{-y}}}{h_y}\right)$ for $d = 2$, and

$$\tilde{\nabla}_+ I_i := \left(\frac{I_{i_{+x}} - I_i}{h_x}, \frac{I_{i_{+y}} - I_i}{h_y}, \frac{I_{i_{+z}} - I_i}{h_z}\right) \tag{2.15}$$

for forward differences with $d = 3$, $\tilde{\nabla}_+ I_i := \left(\frac{I_{i_{+x}} - I_i}{h_x}, \frac{I_{i_{+y}} - I_i}{h_y}\right)$ for $d = 2$. For the neighboring indices of I_i, where $i = \hat{\imath} + \hat{\jmath} m_x + \hat{k} m_x m_y$, with $\hat{\imath} = 1, \ldots, m_x$, $\hat{\jmath} = 1, \ldots, m_y$, $\hat{k} = 1, \ldots, m_z$, we introduce a special notation to handle the boundary conditions with

$$i_{-x} := \max(\hat{\imath} - 1, 1) + \hat{\jmath}\, m_x + \hat{k} m_x m_y, \quad i_{+x} := \min(\hat{\imath} + 1, m_x) + \hat{\jmath}\, m_x + \hat{k} m_x m_y, \tag{2.16}$$

$$i_{-y} := \hat{\imath} + \max(\hat{\jmath} - 1, 1) m_x + \hat{k} m_x m_y, \quad i_{+y} := \hat{\imath} + \min(\hat{\jmath} + 1, m_y)\, m_x + \hat{k} m_x m_y, \tag{2.17}$$

with $\hat{k} = 0$ for $d = 2$ and additionally

$$i_{-z} := \hat{\imath} + \hat{\jmath} m_x + \max(\hat{k} - 1, 1) m_x m_y, \quad i_{+z} := \hat{\imath} + \hat{\jmath} m_x + \min(\hat{k} + 1, m_z)\, m_x m_y \tag{2.18}$$

for $d = 3$. For indices inside the image domain, this maps to a regular linear index. At the boundary, however, this effectively duplicates the image values, implementing Neumann conditions for cell-centered grids [Hel06, §4.4.3]. Unlike Dirichlet conditions, this avoids introducing artificial edges at the image borders, which would otherwise influence image alignment.

Both operators from (2.14) and (2.15) are now concatenated into $\tilde{\nabla} : \mathbb{R}^{\bar{m}} \to \mathbb{R}^{2d}$ with

$$\tilde{\nabla} I_i := \left(\tilde{\nabla}_- I_i, \tilde{\nabla}_+ I_i\right), \tag{2.19}$$

such that we can define

$$\|\tilde{\nabla} I_i\|_\varepsilon := \sqrt{\frac{1}{2}\langle \tilde{\nabla} I_i, \tilde{\nabla} I_i \rangle + \varepsilon^2}. \tag{2.20}$$

This special finite difference scheme allows to symmetrically use forward and backward differences for approximating ∇R_i and ∇T_i in (2.13). This effectively considers I_i *and* its two neighbors, without resulting in central differences $\tilde{\nabla}_\pm I_i = \frac{1}{2}\left(\tilde{\nabla}_- I_i + \tilde{\nabla}_+ I_i\right)$, which would not consider I_i but *only* the neighboring points and can lead to checkerboarding issues. In (2.20), the forward and backward gradient approximations are averaged with

$$\begin{aligned} \frac{1}{2}\langle \tilde{\nabla} I_i, \tilde{\nabla} I_i \rangle &= \frac{1}{2}\left(\langle \tilde{\nabla}_- I_i, \tilde{\nabla}_- I_i \rangle + \langle \tilde{\nabla}_+ I_i, \tilde{\nabla}_+ I_i \rangle\right) \\ &= \frac{1}{2}\left(\left(\tilde{\nabla}_- I_i\right)_1^2 + \left(\tilde{\nabla}_+ I_i\right)_1^2\right) + \ldots + \frac{1}{2}\left(\left(\tilde{\nabla}_- I_i\right)_d^2 + \left(\tilde{\nabla}_+ I_i\right)_d^2\right), \end{aligned}$$

to obtain squared gradient values on the cell-centered image grid. This “square-then-average” scheme does not result in central differences, but retains the gradient information from the short differences. With this, it is capable to capture oscillating functions that would yield zero derivatives with regular central differences [Hel06].

2.4.5. Curvature regularizer

The curvature regularizer, defined in (2.4), only depends on the deformation and is therefore discretized on the deformation grid as

$$\mathcal{S}(\varphi) \approx S(\mathbf{y}) = \bar{h}^{\mathrm{y}} \sum_{i=1}^{\bar{m}^{\mathrm{y}}} \sum_{j=1}^{d} \left(\tilde{\Delta} \mathbf{u}_{i+(j-1)\bar{m}^{\mathrm{y}}}\right)^2, \tag{2.21}$$

with the displacement $\mathbf{u} := (\mathrm{u}_1, \ldots, \mathrm{u}_{d\bar{m}^{\mathrm{y}}}) := \mathbf{y} - \mathbf{x}^{\mathrm{y}} \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}$. Similar to the discretized gradient operator, $\tilde{\Delta}\mathbf{u}_i : \mathbb{R}^{d\bar{m}^{\mathrm{y}}} \to \mathbb{R}$ is a finite-differences approximation of the Laplacian with

$$\tilde{\Delta} \mathbf{u}_{i^{\mathrm{y}}+d\bar{m}^{\mathrm{y}}} := \sum_{k \in \hat{d}} \frac{\mathbf{u}_{i^{\mathrm{y}}_{-k}+d\bar{m}^{\mathrm{y}}} - 2\,\mathbf{u}_{i^{\mathrm{y}}+d\bar{m}^{\mathrm{y}}} + \mathbf{u}_{i^{\mathrm{y}}_{+k}+d\bar{m}^{\mathrm{y}}}}{(h^{\mathrm{y}}_k)^2}, \tag{2.22}$$

where $\hat{d} := \{x, y\}$ for $d = 2$, and $\hat{d} := \{x, y, z\}$ for $d = 3$. Again, Neumann boundary conditions are used [FM03b] for $\mathbf{u}$. Other authors propose the use of higher-order boundary conditions and considering mixed derivatives for “full curvature” regularization [Hen06]. However, since this includes much more involved computations [Hel06, §4.2.3], it is not considered here.

As the curvature regularizer is discretized on a nodal grid, a slightly different implementation of boundary conditions must be chosen, since some nodes are now placed at the boundary $\partial\Omega_{\mathcal{R}}$, see again Figure 2.1(b). With $i^{\mathrm{y}} = \hat{i}^{\mathrm{y}} + \hat{j}^{\mathrm{y}} m_x^{\mathrm{y}} + \hat{k}^{\mathrm{y}} m_x^{\mathrm{y}} m_y^{\mathrm{y}}$ and $\hat{i}^{\mathrm{y}} = 1, \dots, m_x^{\mathrm{y}}$, $\hat{j}^{\mathrm{y}} = 1, \dots, m_y^{\mathrm{y}}$, $\hat{k}^{\mathrm{y}} = 1, \dots, m_z^{\mathrm{y}}$, we define

$$
\begin{aligned}
i_{-x}^{\mathrm{y}} &:= \left|\hat{i}^{\mathrm{y}} - 1\right| + \hat{j}^{\mathrm{y}}\, m_x^{\mathrm{y}} + \hat{k} m_x^{\mathrm{y}} m_y^{\mathrm{y}}, \quad & i_{+x}^{\mathrm{y}} &:= m_x^{\mathrm{y}} - \left|\hat{i}^{\mathrm{y}} + 1 - m_x^{\mathrm{y}}\right| + \hat{j}^{\mathrm{y}}\, m_x^{\mathrm{y}} + \hat{k}^{\mathrm{y}} m_x^{\mathrm{y}} m_y^{\mathrm{y}}, \\
i_{-y}^{\mathrm{y}} &:= \hat{i}^{\mathrm{y}} + \left|\hat{j}^{\mathrm{y}} - 1\right| m_x^{\mathrm{y}} + \hat{k} m_x^{\mathrm{y}} m_y^{\mathrm{y}}, \quad & i_{+y}^{\mathrm{y}} &:= \hat{i}^{\mathrm{y}} + \left(m_y^{\mathrm{y}} - \left|\hat{j}^{\mathrm{y}} + 1 - m_y^{\mathrm{y}}\right|\right) m_x^{\mathrm{y}} + \hat{k}^{\mathrm{y}} m_x^{\mathrm{y}} m_y^{\mathrm{y}},
\end{aligned}
$$

with $\hat{k}^{\mathrm{y}} = 0$ for $d = 2$ and for $d = 3$ additionally

$$
i_{-z}^{\mathrm{y}} := \hat{i}^{\mathrm{y}} + \hat{j}^{\mathrm{y}} m_x^{\mathrm{y}} + \left|\hat{k}^{\mathrm{y}} - 1\right| m_x^{\mathrm{y}} m_y^{\mathrm{y}}, \quad i_{+z}^{\mathrm{y}} := \hat{i}^{\mathrm{y}} + \hat{j}^{\mathrm{y}} m_x^{\mathrm{y}} + \left(m_z^{\mathrm{y}} - \left|\hat{k}^{\mathrm{y}} + 1 - m_z^{\mathrm{y}}\right|\right) m_x^{\mathrm{y}} m_y^{\mathrm{y}},
$$

such that at the domain boundaries it holds $\mathbf{u}_{i_{-k} + d\bar{m}^{\mathrm{y}}} = \mathbf{u}_{i_{+k} + d\bar{m}^{\mathrm{y}}}$ [Hel06, §4.4.3]. Note that, since $\tilde{\Delta}$ is a linear operator, with an appropriate matrix $A \in \mathbb{R}^{d\bar{m}^{\mathrm{y}} \times d\bar{m}^{\mathrm{y}}}$ the curvature regularizer is often also written as a quadratic form

$$
S(\mathbf{y}) = \bar{h}^{\mathrm{y}}\ (A\mathbf{u})^{\top} A\mathbf{u} = \bar{h}^{\mathrm{y}}\, \mathbf{u}^{\top} A^{\top} A\mathbf{u}, \tag{2.23}
$$

which can be useful for the calculation of derivatives [Mod09, §8].

2.5. Numerical optimization

Using the discretized versions of distance measures and regularizer from the previous sections, a discrete objective function $J(\mathbf{y}) : \mathbb{R}^{d\bar{m}^{\mathrm{y}}} \to \mathbb{R}$ can be defined as

$$
J(\mathbf{y}) := D(\mathbf{y}) + \alpha S(\mathbf{y}), \tag{2.24}
$$

with the corresponding finite-dimensional optimization problem

$$
\min_{\mathbf{y} \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}} J(\mathbf{y}). \tag{2.25}
$$

This objective function can now be minimized by using methods from numerical optimization.

The derived functions are assumed to be continuous and sufficiently smooth, but are generally non-convex and non-linear. For this, in the literature a broad variety of *iterative optimization schemes* is available. To avoid local minima, the optimization is embedded in a multi-level strategy, that will be described in Section 2.5.6.

In this section, we will outline the main features of the optimization algorithms used in this thesis. For further in-depth discussion of mathematical details of the presented algorithms, we refer to [NW06; GMW81; DS96]. We do not discuss existence or uniqueness of solutions here, but refer to the references given in [Mod09, §10.2.1]. An overview of the registration algorithm can be found in Algorithm 2.1 at the end of this section. This algorithm description also contains references to the corresponding sections to aid navigation of the manuscript.

Generally, in iteration $k \in \mathbb{N}$, iterative optimization schemes update the current deformation $\mathbf{y}^k$ with a *search direction* $\mathbf{s}^k \in \mathbb{R}^{d\bar{m}^y}$, such that

$$\mathbf{y}^{k+1} = \mathbf{y}^k + \eta\, \mathbf{s}^k, \tag{2.26}$$

where $\eta > 0$ is a step-length parameter determined via a *line search* algorithm, see Section 2.5.4. In order to minimize the function value, $\mathbf{s}^k$ is chosen as a *descent direction*, such that [GMW81, §4.3]

$$\nabla J(\mathbf{y})^\top \mathbf{s}^k < 0. \tag{2.27}$$

Various algorithms have been used in medical image registration [OT14; SSKH10a] to determine a search direction $\mathbf{s}^k$ and comparative evaluations of optimizers have been performed for registration with mutual information [MVS99; KSP07], but no algorithm performed best for all described cases.

Some algorithms use derivative-free direct search methods, especially when the gradient computation is complicated, costly, or the objective function is not differentiable. The two most popular direct search algorithms are Powell's method of conjugate directions and the Nelder-Mead downhill simplex method [Fle87, §2.2], [Kel99, §8.1]. In a medical image registration context these have been utilized especially in conjunction with the originally discrete mutual information distance and a low number of degrees of freedom such as rigid and affine transformations. Powell's method has for example been utilized in [MCV+97; PMV00; CWN02; ARH05] and the Nelder-Mead method in [DSHK99; SZ02; CBMK04]. More recent derivative-free algorithms have been compared for rigid registration in [WP06]. While the computation of derivative free optimization schemes is comparably easy, they can exhibit very slow convergence in practice [KLT03].

An alternative are derivative-based methods. The simplest derivative-based method is *steepest descent*, where

$$\mathbf{s}^k = -\nabla J(\mathbf{y}^k).$$

Steepest descent has been used in several different image registration approaches and frameworks, see [SDP13] for further references. The steepest descent method exhibits a linear convergence rate [NW06, §3.3] and therefore still *"can have an unacceptably slow rate of convergence, even when the Hessian is reasonably well conditioned"* [NW06, §3.3]. In [KSP07], a stochastic gradient descent method was proposed, which results in a fast single update step but comes at the cost of a slower convergence and thus a very high number of iterations.

A faster convergence can be achieved with algorithms based on Newton's Method, which will be described in the following.

Other optimization algorithms that have been employed for image registration include a non-linear conjugate gradient method [KSP07] (which can be related to the L-BFGS method described in Section 2.5.2, cf. [NW06, §7.2]), discrete optimization strategies [HJBS11; HSP+16; SOPD15; TC07], and evolutionary algorithms [SCD11; RJR00], which are not considered here.

2.5.1. Newton's method

Assuming $J : \mathbb{R}^{d\bar{m}^y} \to \mathbb{R}$ is at least twice continuously differentiable, Newton's method is based on a quadratic Taylor series approximation

$$J(\mathbf{y}^k + \mathbf{s}^k) \approx J(\mathbf{y}^k) + \nabla J(\mathbf{y}^k)^\top \mathbf{s}^k + \frac{1}{2}\mathbf{s}^\top \nabla^2 J(\mathbf{y}^k)\mathbf{s}^k.$$

With the necessary condition for a minimizer, $\frac{\partial J(\mathbf{y}^k+\mathbf{s}^k)}{\partial \mathbf{s}^k} = 0$, we obtain the Newton equation

$$\nabla^2 J(\mathbf{y}^k)\mathbf{s}^k = -\nabla J(\mathbf{y}^k), \tag{2.28}$$

where solving for $\mathbf{s}^k$ gives the next iterative update. In a region close enough to a minimizer, Newton's method converges locally quadratic assuming sufficient smoothness [NW06, §3.3]. Note that in (2.28), $\nabla^2 J$ is generally required to be invertible and with (2.27) to be positive definite in order to obtain a descent direction.

The Newton update step is computationally expensive: First, the exact Hessian of the objective function needs to be computed. Second, for each iteration, the system of linear equations in (2.28) must be solved. Therefore, several algorithms have been developed that utilize approximations of the Hessian $\nabla^2 J$, instead of computing it exactly. While these algorithms do not converge as fast as Newton's method in theory, they typically still exhibit super-linear convergence rates [NW06, §3.3].

Newton-like methods

Algorithms, which approximate the Hessian in Newton's method with a matrix $H^k \approx \nabla^2 J(\mathbf{y}^k)$ are called Newton-like methods [DS96, p. xiii].

An important subclass of these algorithms are so-called quasi-Newton methods. These methods perform an iterative update of the inverse Hessian approximation $(H^k)^{-1}$ by using only gradient information. While several different methods exist, by far the most popular in the field of medical image registration is a limited-memory variant of the *Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.* This so-called *L-BFGS method* has for example been used in [MHV+03; LSM+10; CCBF09; VRK+10; Sdi08; RHKF13] and will be described in Section 2.5.2.

Another important subclass are so-called least-squares Newton methods. These methods exploit a least-squares structure of the problem formulation to perform an approximation of the Hessian by dropping specific higher order terms for their computation. In contrast to quasi-Newton methods, the Hessian approximation is freshly computed in each step instead of an iterative update. The resulting *Gauss-Newton* method has been applied in [HJB+12; SGR+08; HM06a; AF11] for medical image registration and will be described in Section 2.5.3. An extension of the Gauss-Newton method is the *Levenberg-Marquardt* method, that has been used in [TRU98; KNFM04; LYC08; Hen03]. This method extends the Gauss-Newton approach by a variable damping factor λ, such that far away from a minimizer the Levenberg-Marquardt algorithm approximates a slow, but globally convergent gradient descent method and behaves like the faster Gauss-Newton method when close to the solution.

In this work, we have chosen the L-BFGS method and the Gauss-Newton scheme for optimization, since these have been proven to be successful in various real-world applications. While the L-BFGS method only requires gradient information and iteratively updates the Hessian approximation, the Gauss-Newton scheme exploits the specific problem structure and performs a new approximation at every step, which may lead to convergence within fewer iterations, making both schemes an interesting choice.

2.5.2. Limited-memory BFGS

The limited-memory BFGS (L-BFGS) method represents a specialized limited-memory variant of the underlying BFGS algorithm.

In order to obtain a new search direction, the BFGS method performs an iterative update of the Hessian approximation H^k at each step k. For computing the update, the method requires the Hessian approximation H^{k+1} to fulfill the condition

$$\nabla J(\mathbf{y}^{k+1}) = \nabla J(\mathbf{y}^k) + H^{k+1}\eta\mathbf{s}^k.$$

Using $\eta\mathbf{s}^k = \mathbf{y}^{k+1} - \mathbf{y}^k$ from (2.26) and re-ordering gives the so-called *secant equation* [NW06, §6.1]

$$H^{k+1}(\mathbf{y}^{k+1} - \mathbf{y}^k) = \nabla J(\mathbf{y}^{k+1}) - \nabla J(\mathbf{y}^k),$$

which with $\hat{\mathbf{s}}^k := \eta\mathbf{s}^k = \mathbf{y}^{k+1} - \mathbf{y}^k$ and $\mathbf{g}^k := \nabla J(\mathbf{y}^{k+1}) - \nabla J(\mathbf{y}^k)$ reads

$$H^{k+1}\hat{\mathbf{s}}^k = \mathbf{g}^k.$$

By imposing further conditions, requiring the updated Hessian approximation to be symmetric, and H^{k+1} chosen so that it minimizes $\|H^{k+1} - H^k\|$ for the weighted Frobenius norm [NW06, §6.1], a rank-two update rule with

$$H^{k+1} = H^k - \frac{H^k\hat{\mathbf{s}}^k\hat{\mathbf{s}}^{k\top}H^k}{\hat{\mathbf{s}}^{k\top}H^k\hat{\mathbf{s}}^k} + \frac{\mathbf{g}^k\mathbf{g}^{k\top}}{\mathbf{g}^{k\top}\hat{\mathbf{s}}^k}$$

can be derived. For a details we refer to [NW06, §6.1] or [DS96, §9.2].

A benefit of the BFGS method is that instead of solving (2.28) with $H^{k+1} \approx \nabla^2 J(\mathbf{y}^k)$ in each step, the inverse $\tilde{H}^{k+1} := \left(H^{k+1}\right)^{-1}$ can be updated directly with

$$\tilde{H}^{k+1} = \left(I - \frac{\hat{\mathbf{s}}^k\mathbf{g}^{k\top}}{\mathbf{g}^{k\top}\hat{\mathbf{s}}^k}\right)\tilde{H}^k\left(I - \frac{\mathbf{g}^k\hat{\mathbf{s}}^{k\top}}{\mathbf{g}^{k\top}\hat{\mathbf{s}}^k}\right) + \frac{\hat{\mathbf{s}}^k\hat{\mathbf{s}}^{k\top}}{\mathbf{g}^{k\top}\hat{\mathbf{s}}^k}, \tag{2.29}$$

where I is the identity matrix. Since the inverse Hessian approximation will generally be dense [NW06, §7.2], storing $\tilde{H}^k \in \mathbb{R}^{d\bar{m}^y \times d\bar{m}^y}$ becomes infeasible. However, as can be seen from (2.29), by recursively substituting the computations for $\tilde{H}^k$, the current iterate can be computed by using only the vectors $\hat{\mathbf{s}}^i, \mathbf{g}^i \in \mathbb{R}^{d\bar{m}^y}$, $i = 0, \ldots, k$ with an initial value for $\tilde{H}_0$, which will be described at the end of this section.

In this scheme, the required memory for storing these vectors grows with the number of iterations, which can still become significantly large. Therefore, in contrast to the original BFGS algorithm, the L-BFGS method discards older $\hat{\mathbf{s}}^i, \mathbf{g}^i$ with $i < k - L$, where $L \in \mathbb{N}$ is a small number; in this thesis we use $L = 5$. For the iterative update, the two-loop update as presented in [Noc80] and [NW06, Algorithm 7.4] is used. This algorithm directly computes $\mathbf{s}^{k+1} = -\tilde{H}^k \nabla J(\mathbf{y}^k)$, implicitly solving the Newton equation (2.28) with the L-BFGS Hessian approximation to obtain a new search direction.

To complete the L-BFGS algorithm, an initial value $\tilde{H}^0$ is required. As proposed by [Hel06, §5], we utilize the Hessian of the regularizer, described in (3.49), with an additional diagonal term, such that

$$H^0 = \nabla^2 S + \gamma I, \tag{2.30}$$

with $\gamma > 0$ to ensure positive definiteness. Note, that since the regularizer is a quadratic form, (2.30) is constant. To compute the required multiplication with the inverse $\tilde{H}^0$ in the two-loop update, we solve a system of linear equations as proposed in [NW06, §7.2] using the conjugate gradient method [GV13, §11.3].

2.5.3. Gauss-Newton

Similar to the methods described before, the Gauss-Newton method also employs an approximation $H^k \approx \nabla^2 J(\mathbf{y}^k)$ in solving the Newton equation (2.28). However, in contrast to quasi-Newton methods, instead of iteratively updating the Hessian approximation, the Gauss-Newton scheme computes H^k anew in every iteration. The Gauss-Newton method can be utilized for least-squares type objective functions of the form

$$\hat{J}(\mathbf{y}) = r(\mathbf{y})^\top r(\mathbf{y}) = \|r(\mathbf{y})\|_2^2,$$

where $r(\mathbf{y}) : \mathbb{R}^{d\bar{m}^y} \to \mathbb{R}^{d\bar{m}^y}$ is a non-linear residual function [Fle87, §6.1]. Using this notation, the gradient of $\hat{J}$ can be written as

$$\nabla \hat{J}(\mathbf{y}) = 2r(\mathbf{y})^\top \mathrm{d}r(\mathbf{y}), \tag{2.31}$$

where $\mathrm{d}r \in \mathbb{R}^{d\bar{m}^y \times d\bar{m}^y}$ is the Jacobian matrix of r with

$$\mathrm{d}r := \left(\frac{\partial r_i}{\partial \mathbf{y}_j} \right)_{i=1,\ldots,d\bar{m}^y;\ i=1,\ldots,d\bar{m}^y}.$$

Using the product rule, the Hessian is

$$\nabla^2 \hat{J}(\mathbf{y}) = 2\mathrm{d}r(\mathbf{y})^\top \mathrm{d}r(\mathbf{y}) + 2 \sum_{i=1}^{d\bar{m}^y} r_i(\mathbf{y}) \nabla^2 r_i(\mathbf{y}),$$

where $\nabla^2 r_i(\mathbf{y}) \in \mathbb{R}^{d\bar{m}^{\mathrm{y}} \times d\bar{m}^{\mathrm{y}}}$, $i = 1, \dots, d\bar{m}^{\mathrm{y}}$ are the individual matrices of the third-order tensor $\nabla^2 r$. It can be shown that for problems where the residual $r(\mathbf{y})$ is small near the solution, the second term can be neglected, see [DS96, §10.2] or [NW06, §10.3], giving a quadratic Gauss-Newton approximation of the Hessian with

$$\nabla^2 \hat{J}(\mathbf{y}^k) \approx H^k := 2\mathrm{d}r(\mathbf{y}^k)^\top \mathrm{d}r(\mathbf{y}^k). \tag{2.32}$$

This approximation avoids the expensive computation of second-order derivatives and can be obtained by only using information already computed for the gradient (2.31). To obtain a new search direction, we use H^k as a Hessian approximation in (2.28) and solve the system of linear equations by using a standard conjugate gradient method [GV13, §11.3]. Note that with (2.28) and (2.32) it holds

$$\mathbf{s}^{k^\top} \nabla \hat{J}(\mathbf{y}^k) = -\mathbf{s}^{k^\top} H^k(\mathbf{y}^k)\mathbf{s}^k = -\mathbf{s}^{k^\top} 2\mathrm{d}r(\mathbf{y}^k)^\top \mathrm{d}r(\mathbf{y}^k)\mathbf{s}^k = -2\|\mathrm{d}r(\mathbf{y}^k)\mathbf{s}^k\|_2^2 \leq 0,$$

such that according to (2.27) every search direction computed with the Gauss-Newton method is either a descent direction, or – if the norm is zero – $\mathbf{y}^k$ is a stationary point [NW06, §10.3].

2.5.4. Line search

To determine the step-length parameter $\eta > 0$ given in the update step (2.26), for L-BFGS as well as Gauss-Newton, a line search algorithm is required. Given a descent direction $\mathbf{s}^k$, the objective function is linearized as

$$\phi(\eta) := J(\mathbf{y}^k + \eta\mathbf{s}^k).$$

The goal is now to efficiently find a value for η, which ensures sufficient minimization of the objective function. As an exact minimum is costly to determine, especially for non-linear functions, inexact line search algorithms rely on quickly finding a suitable η, which must fulfill certain conditions. A typical condition, which we also employ in this work as proposed in [Mod09, §6.3.3], is the so-called *Armijo condition* [NW06, §3.1]

$$J(\mathbf{y}^k + \eta\mathbf{s}^k) \leq J(\mathbf{y}^k) + \delta_1 \eta \nabla J(\mathbf{y}^k)^\top \mathbf{s}^k, \tag{2.33}$$

where δ_1 is a small constant. In this work, we set $\delta_1 = 10^{-4}$. The Armijo condition ensures a sufficient decrease in the objective function value. Starting with $\eta = 1$, if (2.33) is not fulfilled, the step length is updated with $\eta \leftarrow 0.5\eta$, until the condition is satisfied or a maximum of ten iterations is reached as a safeguard, in which case the optimization is stopped. Another condition that is often required is the so-called *curvature condition*

$$\nabla J(\mathbf{y}^k + \eta\mathbf{s}^k)^\top \mathbf{s}^k \geq \delta_2 \nabla J(\mathbf{y}^k)^\top \mathbf{s}^k,$$

with both conditions together also known as *Wolfe conditions*. However, since this condition requires the additional costly evaluation of objective function gradients, as proposed in [Hel06, §5.3.1], we only enforce (2.33) in the line search algorithm, also known as Armijo line search.

2.5.5. Stopping criteria

Finally, for every iterative optimization algorithm, stopping criteria have to be defined, indicating a sufficient minimization of the objective function. We employed the stopping criteria

$$\|\mathbf{y}^k - \mathbf{y}^{k-1}\|_\infty = \|\eta \mathbf{s}^{k-1}\|_\infty < \delta_3 \tag{2.34}$$

$$\frac{J\left(\mathbf{y}^{k-1}\right) - J(\mathbf{y}^k)}{J(\mathbf{y}^0) - J(\mathbf{y}^k)} < \delta_3 \tag{2.35}$$

$$\frac{\|\nabla J(\mathbf{y}^k)\|_\infty}{\|\nabla J(\mathbf{y}^0)\|_\infty} < \delta_3 \tag{2.36}$$

and as a safeguard

$$k \geq k_{\max}, \tag{2.37}$$

which are closely related to those proposed in [GMW81, §8.3.2]. The first two criteria (2.34) and (2.35) consider the progress of the iteration regarding change in the current iterate, utilizing the search step length and change in the function value, while (2.36) examines the gradient magnitude of the objective function. Here, $\mathbf{y}^0$ is an external starting guess, which can be the identity or the result of a pre-registration. The iteration is stopped if any of these conditions becomes true. The last criterion (2.37) acts as a safeguard which terminates the iteration if the other criteria fail within a reasonable amount of iterations, typically $k_{\max} = 100$.

2.5.6. Multi-level scheme

In image registration, the objective function is generally not convex. Therefore, the above derivative-based local optimization schemes are only capable to determine *local* minima of the objective function, which may not coincide with the desired transformation. Therefore, to avoid “getting stuck” in local minima, we use an additional coarse-to-fine multi-level scheme, consecutively solving the registration problem on different resolutions.

First, to enable the multi-level scheme, image representations of different resolutions need to be computed. For this, a lowpass *image pyramid* is created [Jäh05, §5.2.2]. By filtering the image with the normalized binomial kernel $\mathcal{G} := (0.25, 0.5, 0.25)$ consecutively in each dimension, a smoothed representation of both images is created [Jäh05, §11.4]. Then, the smoothed image is subsampled to a resolution of $m^{\mathrm{R}}_{\mathrm{coarse}} = \lfloor \frac{m^{\mathrm{R}}}{2} \rfloor$ or $m^{\mathrm{T}}_{\mathrm{coarse}} = \lfloor \frac{m^{\mathrm{T}}}{2} \rfloor$, for reference and template image, respectively, using linear interpolation if required. Additionally, the corresponding deformation grid resolution is reduced accordingly, maintaining the original ratio between image grid and deformation grid. This process is repeated until a sequence of progressively coarser image representations has been created.

Second, starting with the coarsest image grid and deformation grid resolutions using the identity transformation (or the result of an external pre-registration) as a first iterate, the registration problem is solved on consecutively finer deformation and image grids. At the end of each level, the result deformation is upsampled to the next level's deformation grid resolution and utilized as an initial guess.

In this process, the smoothing of the image data serves as a lowpass filter, eliminating detailed structures, which typically cause local minima.

In a practical setting, compared to block filters, used for example in [Mod09, §3.7], this method has the advantage that it is not restricted to image sizes that are an even number or a power of two.

This concludes the description of all important components of the registration framework, which serves as a basis for this thesis. The full registration algorithm is summarized as pseudocode in Algorithm 2.1.

```
1:  $\mathbf{T}, \mathbf{R} \leftarrow$ createPyramids$(T, R)$          ▷ Create image pyramids, Section 2.5.6
2:  load $\mathbf{y}^0_{\text{coarsest}}$          ▷ Initialize with identity or pre-registration
3:
4:  for $level$ in [coarsest, finest] do          ▷ Multi-level loop, Section 2.5.6
5:      load $\mathbf{T}_{level}, \mathbf{R}_{level}$
6:      $k \leftarrow 0$
7:                                                  ▷ Iterative optimization loop, Section 2.5
8:      while !$stop$ do                            ▷ Check stopping criteria, Section 2.5.5
9:          $\mathbf{s}^k \leftarrow$ searchDirection$\Big(J(\mathbf{y}^k_{level})\Big)$          ▷ Search direction, Section 2.5.2, 2.5.3
10:         $\eta^{opt} \leftarrow$ linesearch$\Big(J(\mathbf{y}^k_{level} + \eta\mathbf{s}^k)\Big)$          ▷ Line search, Section 2.5.4
11:         $\mathbf{y}^{k+1}_{level} \leftarrow \mathbf{y}^k_{level} + \eta^{opt}\mathbf{s}^k$
12:         $k \leftarrow k + 1$
13:     end while
14:
15:     if $level \neq$ finest then
16:         $\mathbf{y}^0_{level+1} \leftarrow$ upsample$\Big(\mathbf{y}^k_{level}\Big)$          ▷ Upsample to next level
17:     else
18:         result $\leftarrow \mathbf{y}^k_{level}$          ▷ Final result
19:     end if
20: end for
```

Algorithm 2.1: Pseudocode for the multi-level registration algorithm. The objective function, consisting of distance measure and regularizer, is minimized by using a numerical optimization scheme. This optimization is embedded in a coarse-to-fine multi-level scheme, where the resulting deformation is used as a starting guess for the next finer level.

2.6. Summary

In this chapter, we described the underlying image registration framework and its individual components and building blocks, which will serve as a basis for the methods presented in Chapter 3. For this, the concept and general field of image registration was introduced in Section 2.2 and important approaches and applications were highlighted. Since the overall goal of this work is to develop a fast and efficient algorithm, an overview of related works concerning performance of registration algorithms was given in Section 2.2.4. After this, the variational image registration framework used in this thesis was described in Section 2.3 with two important distance measures, the mono-modal sum of squared differences and the multi-modal normalized gradient fields. Furthermore, curvature regularization was introduced, which completes the objective function – the core component of the registration algorithm. Since we employ a discretize-then-optimize approach, in Section 2.4 a discrete formulation of the objective function was derived and in Section 2.5 numerical optimization schemes were discussed. Special emphasis was placed on derivative-based Newton-like methods, especially the limited-memory BFGS algorithm and the Gauss-Newton method. A description of the coarse-to-fine multi-level scheme in Section 2.5.6 completes the description of the registration algorithm.

3 Matrix-free methods for efficient derivative computations

Considering the Newton-like optimization methods described in Section 2.5, computing derivatives of the objective function is a critically important task for solving the registration problem. First, for a fast convergence of the optimization algorithms the derivatives need to be determined as accurately as possible. Second, as will be seen later on, derivative computations amount to a considerable portion of the overall runtime and memory consumption of the registration algorithm. Therefore, analytical derivatives of the discretized objective function components and their computation are analyzed and described in detail in this chapter. From this analysis, a novel matrix-free computation scheme is derived, aiming at fast and efficient derivative evaluations in order to improve the memory consumption and runtime of the registration algorithm.

Derivative computation approaches. While some approaches for computing derivatives simply approximate the gradient from objective function values via finite differences [RM03], for large problem sizes this approach can be time consuming and the inherent inaccuracy may cause a slower convergence of the optimization scheme. Another approach is to compute analytical derivatives of the objective function, which is utilized in the following sections.

In the most popular implementation of the presented variational approach, described in [Mod09], the individual derivatives are constructed based on individual Jacobian matrices, derived from a decomposition of the objective function into a composition of multiple functions. These Jacobian matrices are constructed as sparse matrices, which are then multiplied to obtain the result, also called a *matrix-based* approach. This approach is very close to the analytical derivation and allows for flexibility in interchanging components. Additionally, further insights can be obtained from examining the matrix structures, which also makes this a good choice for teaching purposes. However, this method requires considerable resources for constructing, handling, storing and computing sparse matrices and their products as intermediate results. Furthermore, the approach makes an efficient parallelization and full exploitation of the problem structure difficult.

Therefore, in the following sections, a so-called *matrix-free* formulation for the objective function derivatives is presented. As will be shown, the known matrix structures of the derivatives can be broken down into closed-form formulations which are mathematically equivalent with those obtained by the matrix-based approach. However, these closed forms no longer require special data structures, such as sparse matrices, nor explicit matrix multiplications. They allow for an efficient computation of the derivatives without computational overhead and largely reduce memory requirements. Additionally, parallelization is straightforward.

In the following, derivative formulations for gradient and Gauss-Newton Hessian computations of the presented distance measures are introduced in Section 3.1. From this, matrix-free formulations for gradient and Hessian computations are derived for the sum of squared differences in Section 3.2 and for the normalized gradient fields in Section 3.3. Matrix-free computations for the curvature regularizer are presented in Section 3.4. Furthermore, efficient matrix-free methods for the grid conversion from deformation grid to image grid and the corresponding transposed operator are derived in Section 3.5, together with an efficient parallelization strategy.

As previously discussed in Section 2.2.1 and Section 2.4.2, the rigid and affine transformation models can be interpreted as a special case of the deformable registration model. However, there are several implications of the now fixed number of unknowns, which also require changes in the matrix-free formulation. This is discussed in Section 3.6.

In Section 3.7, the properties of the obtained matrix-free computations are analyzed with regard to computational operations and memory requirements. Also, a generalization of the approach to other methods is discussed. Furthermore, in Section 3.8, details of specific implementations of the matrix-free approach are presented, targeting different platforms such as CPU, GPU and DSP.

Acknowledgments and related publications. We have previously published parts and preliminary versions of the methods presented in this chapter in several conference proceedings and journal articles. We published the matrix-free approaches for deformable registration in [KRDL18*; KR14*; KDHP15*] and matrix-free methods for the affine deformation model in [RKH+13*; RKT+17*].

We have published the implementations on specialized platforms discussed in Section 3.8 in [BKR+14*; TRK+14*]. Additionally, implementations on GPUs were supported by the work performed in connection with the Master's Theses [Mei16; Tra14] and the Bachelor's Thesis [Ber12] a for a DSP platform. All of these theses were jointly supervised by the author of this thesis and J. Rühaak. The matrix-free computations for *affine* registration, the *affine implementation* on GPU and DSP and the matrix-free computations for the *gradient*, which we published in [RKT+17*; KR14*; RKH+13*; BKR+14*; TRK+14*], were also incorporated in [Rüh17]. The author of this thesis and J. Rühaak contributed equally to the relevant publications.

3.1. Distance measure derivatives

In this section, we derive *analytical derivatives* for gradient and Gauss-Newton Hessian approximation of the SSD (Section 3.2) and NGF distance measure (Section 3.3).

In order to compute derivatives of the distance measure computations, we make use of the fact that the distance measures in (2.12) and (2.13) exhibit a similar structure and can be formulated as a composition of multiple functions:

$$D(\mathbf{y}) = \psi(r(T(P(\mathbf{y})))). \tag{3.1}$$

Here, the functions $T : \mathbb{R}^{d\bar{m}} \to \mathbb{R}^{\bar{m}}$ and $P : \mathbb{R}^{d\bar{m}^{\mathrm{y}}} \to \mathbb{R}^{d\bar{m}}$ are the known definitions from (2.9), representing the template *image interpolation* and the *grid conversion* function. Additionally, $r : \mathbb{R}^{\bar{m}} \to \mathbb{R}^{\bar{m}}$ is a residual function, evaluating the image similarity between reference and template image for each point on the reference image grid. Furthermore, $\psi : \mathbb{R}^{\bar{m}} \to \mathbb{R}$ is a reduction function, mapping the residual to a real-valued number. This function evaluation in four steps can be written as a mapping

$$\mathbb{R}^{d\bar{m}^{\mathbf{y}}} \xrightarrow{P} \mathbb{R}^{d\bar{m}} \xrightarrow{T} \mathbb{R}^{\bar{m}} \xrightarrow{r} \mathbb{R}^{\bar{m}} \xrightarrow{\psi} \mathbb{R}, \tag{3.2}$$

which transforms the deformation $\mathbf{y} \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}$ to a single scalar, indicating the image similarity.

In order to compute derivatives of this expression, i.e., gradient and Gauss-Newton Hessian approximation of (3.1), we make use of the chain rule. For the gradient, we obtain the column vector

$$\nabla D(\mathbf{y}) = \left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial T}\frac{\partial T}{\partial P}\frac{\partial P}{\partial \mathbf{y}}\right)^{\top} \in \mathbb{R}^{d\bar{m}^{\mathrm{y}} \times 1}, \tag{3.3}$$

which consists of four partial derivatives. As can be seen in (3.2), the individual Jacobian matrices have the dimensions $\frac{\partial \psi}{\partial r} \in \mathbb{R}^{1\times\bar{m}}$, $\frac{\partial r}{\partial T} \in \mathbb{R}^{\bar{m}\times\bar{m}}$, $\frac{\partial T}{\partial P} \in \mathbb{R}^{\bar{m}\times d\bar{m}}$ and $\frac{\partial P}{\partial \mathbf{y}} \in \mathbb{R}^{d\bar{m}\times d\bar{m}^{\mathrm{y}}}$.

Using these matrices, the Gauss-Newton approximation of the Hessian (2.32) can be written as

$$\nabla^2 D(\mathbf{y}) \approx H(\mathbf{y}) = \frac{\partial P}{\partial \mathbf{y}}^{\top}\frac{\partial T}{\partial P}^{\top}\frac{\partial r}{\partial T}^{\top}\frac{\partial^2 \psi}{\partial r^2}\frac{\partial r}{\partial T}\frac{\partial T}{\partial P}\frac{\partial P}{\partial \mathbf{y}}, \tag{3.4}$$

where $\frac{\partial^2 \psi}{\partial r^2}$ is the second-order derivative of the outer reduction function ψ which typically is a constant, as we will see later on.

Despite different computations of r and ψ, all of the presented distance measures share the same definition for T and P, and therefore also for the Jacobian matrices $\frac{\partial T}{\partial P}$ and $\frac{\partial P}{\partial \mathbf{y}}$. The grid conversion derivative $\frac{\partial P}{\partial \mathbf{y}}$ will be described dedicatedly in Section 3.5, while the derivative of the image interpolation function will be derived in the following.

3.1.1. Image interpolation derivatives

As described in Section 2.4.3, we use bi-linear ($d = 2$) or tri-linear interpolation ($d = 3$) to obtain image values at non-integer coordinates. In this section, the structure of the image interpolation derivatives $\frac{\partial T}{\partial P}$ will be described.

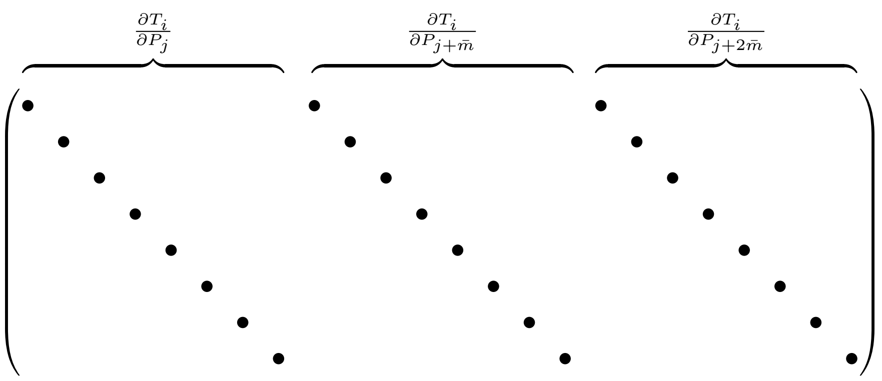

$\frac{\partial T_i}{\partial P_j}$
$\frac{\partial T_i}{\partial P_{j+\bar{m}}}$
$\frac{\partial T_i}{\partial P_{j+2\bar{m}}}$

Figure 3.1.: The Jacobian $\frac{\partial T}{\partial P}$ of the template image interpolation function exhibits a very sparse structure of d horizontally aligned diagonal matrices. The figure shows the matrix for $d = 3$. The three diagonals correspond to the derivatives in x-, y- and z-direction.

From the definition of the interpolation function $T : \mathbb{R}^{d\bar{m}} \to \mathbb{R}^{\bar{m}}$ in (2.9),

$$T_i(\hat{\mathbf{y}}) = \mathcal{T}(\hat{\mathbf{y}}_i) = \mathcal{T}\Big(\hat{\mathrm{y}}_i, \dots, \hat{\mathrm{y}}_{i+(d-1)\bar{m}}\Big)$$

defines the evaluation of the image at a single point. Since the evaluation only depends on the d coordinates of that single point, the derivative of T_i can be written as

$$\frac{\partial T_i}{\partial \hat{\mathrm{y}}_{j+k\bar{m}}} = \frac{\partial T_i}{\partial P_{j+k\bar{m}}} = \begin{cases} \frac{\partial \mathcal{T}}{\partial \hat{\mathrm{y}}_{j+k\bar{m}}} & \text{if } i = j + k\bar{m}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.5}$$

with $j = 1, \dots, \bar{m}$ and $k = 0, \dots, d-1$ for all directional derivatives. Consequently, the Jacobian matrix

$$\frac{\partial T}{\partial P} = \left(\frac{\partial T_i}{\partial P_j}\right)_{i=1,\dots,\bar{m}, j=1,\dots,d\bar{m}} \in \mathbb{R}^{\bar{m} \times d\bar{m}}$$

has the structure of d horizontally aligned diagonal matrices,

$$\frac{\partial T}{\partial P} = \left(\operatorname{diag}\left(\frac{\partial T_1}{\partial P_1}, \dots, \frac{\partial T_{\bar{m}}}{\partial P_{\bar{m}}}\right), \dots, \operatorname{diag}\left(\frac{\partial T_1}{\partial P_{(d-1)\bar{m}+1}}, \dots, \frac{\partial T_{\bar{m}}}{\partial P_{d\bar{m}}}\right)\right), \tag{3.6}$$

where $\operatorname{diag}(\mathbf{x}) : \mathbb{R}^{\bar{m}} \to \mathbb{R}^{\bar{m} \times \bar{m}}$ maps a vector to the corresponding square diagonal matrix with the vector on the diagonal, see Figure 3.1. Each diagonal represents the derivatives of the interpolation function with respect to one coordinate direction.

In order to calculate the non-zero indices of $\frac{\partial T}{\partial P}$, we first define the interpolation function $\mathcal{T}(\hat{\mathbf{y}}_i)$ in more detail. Let $c : \mathbb{R}^d \to \mathbb{R}^d$,

$$c(\hat{\mathbf{y}}_i) := \begin{pmatrix} \frac{\hat{\mathrm{y}}_i - \omega_1}{h_1} - 0.5 \\ \vdots \\ \frac{\hat{\mathrm{y}}_{i+(d-1)\bar{m}} - \omega_{2d-1}}{h_d} - 0.5 \end{pmatrix} \tag{3.7}$$

be a function converting a coordinate $\hat{\mathbf{y}}_i$ from the image domain $\Omega_{\mathcal{R}}$, as defined in (2.5), to zero-based indices. Then for $\alpha, \beta, \gamma \in \{0, 1\}$ with

$$\xi(c, \alpha, \beta, \gamma) := (\lfloor c_1 + 1 \rfloor + \alpha) + (\lfloor c_2 + 1 \rfloor + \beta)\, m_x + (\lfloor c_3 + 1 \rfloor + \gamma)\, m_x m_y$$

(for $d = 2$, $\xi(c, \alpha, \beta) := (\lfloor c_1 + 1 \rfloor + \alpha) + (\lfloor c_2 + 1 \rfloor + \beta)\, m_x$), and additionally abbreviating $\xi_i(\alpha, \beta, \gamma) := \xi(c(\hat{\mathbf{y}}_i)), \alpha, \beta, \gamma)$, as defined in (2.11), $\mathbf{T}_{\xi_i(\alpha,\beta,\gamma)}$ determines the neighboring image data points of $\hat{\mathbf{y}}_i$. If an index is located outside of the domain, a zero value is assumed, which is a reasonable assumption for medical images that often exhibit a black background.

To obtain an interpolated image value, the neighboring data points are weighted according to their distance to the coordinate $\hat{\mathbf{y}}_i$. With the weights

$$w^s(\hat{\mathbf{y}}_i) := \begin{cases} 1 - (c(\hat{\mathbf{y}}_i) - \lfloor c(\hat{\mathbf{y}}_i) \rfloor), & \text{if } s = 0, \\ c(\hat{\mathbf{y}}_i) - \lfloor c(\hat{\mathbf{y}}_i) \rfloor & \text{otherwise,} \end{cases}$$

the interpolated value can be written as

$$T_i(\hat{\mathbf{y}}) = \mathcal{T}(\hat{\mathbf{y}}_i) = \sum_{\alpha=0}^{1} \sum_{\beta=0}^{1} \sum_{\gamma=0}^{1} w(\hat{\mathbf{y}}_i)_1^\alpha w(\hat{\mathbf{y}}_i)_2^\beta w(\hat{\mathbf{y}}_i)_3^\gamma \mathbf{T}_{\xi_i(\alpha,\beta,\gamma)}, \tag{3.8}$$

i.e., a weighted sum of all eight neighboring points of $\hat{\mathbf{y}}_i$. Using Horner's scheme [GV13, §9.2.4] and abbreviating $w := w(\hat{\mathbf{y}}_i)$, this can be implemented efficiently as

$$\begin{aligned} T_i(\hat{\mathbf{y}}) = {} & w_1^0 \left(w_2^0 \left(w_3^0 \mathbf{T}_{\xi_i(0,0,0)} + w_3^1 \mathbf{T}_{\xi_i(0,0,1)} \right) + w_2^1 \left(w_3^0 \mathbf{T}_{\xi_i(0,1,0)} + w_3^1 \mathbf{T}_{\xi_i(0,1,1)} \right) \right) \\ & + w_1^1 \left(w_2^0 \left(w_3^0 \mathbf{T}_{\xi_i(1,0,0)} + w_3^1 \mathbf{T}_{\xi_i(1,0,1)} \right) + w_2^1 \left(w_3^0 \mathbf{T}_{\xi_i(1,1,0)} + w_3^1 \mathbf{T}_{\xi_i(1,1,1)} \right) \right). \end{aligned}$$

For $d = 2$, the third sum over γ in (3.8) vanishes, resulting in a weighted sum of four points. From this, the non-zero derivatives in (3.5) can directly be computed as

$$\begin{aligned} \frac{\partial T_i}{\partial \hat{\mathrm{y}}_i} = {} & \frac{1}{h_1} \Big(- w_2^0 \left(w_3^0 \mathbf{T}_{\xi_i(0,0,0)} + w_3^1 \mathbf{T}_{\xi_i(0,0,1)} \right) - w_2^1 \left(w_3^0 \mathbf{T}_{\xi_i(0,1,0)} + w_3^1 \mathbf{T}_{\xi_i(0,1,1)} \right) \\ & + w_2^0 \left(w_3^0 \mathbf{T}_{\xi_i(1,0,0)} + w_3^1 \mathbf{T}_{\xi_i(1,0,1)} \right) + w_2^1 \left(w_3^0 \mathbf{T}_{\xi_i(1,1,0)} + w_3^1 \mathbf{T}_{\xi_i(1,1,1)} \right) \Big) \qquad (3.9) \\ \frac{\partial T_i}{\partial \hat{\mathrm{y}}_{i+\bar{m}}} = {} & \frac{1}{h_2} \Big(w_1^0 \left(- \left(w_3^0 \mathbf{T}_{\xi_i(0,0,0)} + w_3^1 \mathbf{T}_{\xi_i(0,0,1)} \right) + \left(w_3^0 \mathbf{T}_{\xi_i(0,1,0)} + w_3^1 \mathbf{T}_{\xi_i(0,1,1)} \right) \right) \\ & + w_1^1 \left(- \left(w_3^0 \mathbf{T}_{\xi_i(1,0,0)} + w_3^1 \mathbf{T}_{\xi_i(1,0,1)} \right) + \left(w_3^0 \mathbf{T}_{\xi_i(1,1,0)} + w_3^1 \mathbf{T}_{\xi_i(1,1,1)} \right) \right) \Big) \qquad (3.10) \\ \frac{\partial T_i}{\partial \hat{\mathrm{y}}_{i+2\bar{m}}} = {} & \frac{1}{h_3} \Big(w_1^0 \left(w_2^0 \left(-\mathbf{T}_{\xi_i(0,0,0)} + \mathbf{T}_{\xi_i(0,0,1)} \right) + w_2^1 \left(-\mathbf{T}_{\xi_i(0,1,0)} + \mathbf{T}_{\xi_i(0,1,1)} \right) \right) \\ & + w_1^1 \left(w_2^0 \left(-\mathbf{T}_{\xi_i(1,0,0)} + \mathbf{T}_{\xi_i(1,0,1)} \right) + w_2^1 \left(-\mathbf{T}_{\xi_i(1,1,0)} + \mathbf{T}_{\xi_i(1,1,1)} \right) \right) \Big), \qquad (3.11) \end{aligned}$$

for the derivatives in x-, y- and z-direction, respectively. The derivatives for $d = 2$ can be computed analogously.

Using (3.9) – (3.11), the Jacobian matrix of the image interpolation function, given in (3.6), can be computed explicitly, which will be a central component in the matrix-free formulations.

3.2. Derivative computations for SSD

Using the formulations of the distance measure gradient and Gauss-Newton Hessian approximation as a product of individual Jacobian matrices, given in (3.3) and (3.4), the derivative computations can be analyzed in more detail.

Recalling the definition of the discrete SSD in (2.12) with

$$D^{\mathrm{SSD}}(\mathbf{y}) = \bar{h} \sum_{i=1}^{\bar{m}} \left(T_i(P(\mathbf{y})) - R_i(\mathbf{x})\right)^2 ,$$

we can define the individual functions from (3.1) in order to derive their Jacobians. For the SSD, the function $\psi^{\mathrm{SSD}} : \mathbb{R}^{\bar{m}} \to \mathbb{R}$ is

$$\psi^{\mathrm{SSD}}(r) := \bar{h} \sum_{i=1}^{\bar{m}} r_i^2. \tag{3.12}$$

The residual function $r^{\mathrm{SSD}} : \mathbb{R}^{\bar{m}} \to \mathbb{R}^{\bar{m}}$ is

$$r^{\mathrm{SSD}}(T) := T - R = (T_1, \dots, T_{\bar{m}}) - (R_1, \dots, R_{\bar{m}}) , \tag{3.13}$$

i.e., a point-wise difference. As in (3.1), the SSD can be written as

$$D^{\mathrm{SSD}}(\mathbf{y}) = \psi^{\mathrm{SSD}}(r^{\mathrm{SSD}}(T(P(\mathbf{y})))).$$

3.2.1. Gradient computations

As can be seen from (3.13), derivative calculation for the SSD is comparably simple, as

$$\frac{\partial r^{\mathrm{SSD}}}{\partial T} = I \in \mathbb{R}^{\bar{m} \times \bar{m}} \tag{3.14}$$

is the identity matrix. Furthermore, we have

$$\frac{\partial \psi^{\mathrm{SSD}}}{\partial r^{\mathrm{SSD}}} = 2\bar{h} r^{\mathrm{SSD}} \in \mathbb{R}^{1 \times \bar{m}}.$$

Therefore, with (3.3), the full gradient of the SSD can be written as

$$\nabla D^{\mathrm{SSD}}(\mathbf{y}) = 2\bar{h} \left(r^{\mathrm{SSD}} \frac{\partial T}{\partial P} \frac{\partial P}{\partial \mathbf{y}} \right)^{\top} . \tag{3.15}$$

In a matrix-based approach, (3.15) is evaluated by creating $\frac{\partial T}{\partial P}$ and $\frac{\partial P}{\partial \mathbf{y}}$ as sparse matrices, using the computations from Section 2.4.3 and Section 3.5, and computing their matrix-matrix product as well as the matrix-vector product with r_{SSD}.

In matrix-free computations, single elements of the gradient $(\nabla D_{\mathrm{SSD}})_i \in \mathbb{R}$ are computed on-the-fly from the original input data, i.e., the images R, T and the transformation $\mathbf{y}$, without storing intermediate results, or creating data structures such as sparse matrices. On one hand, this lowers the memory usage of the algorithm, since intermediate results

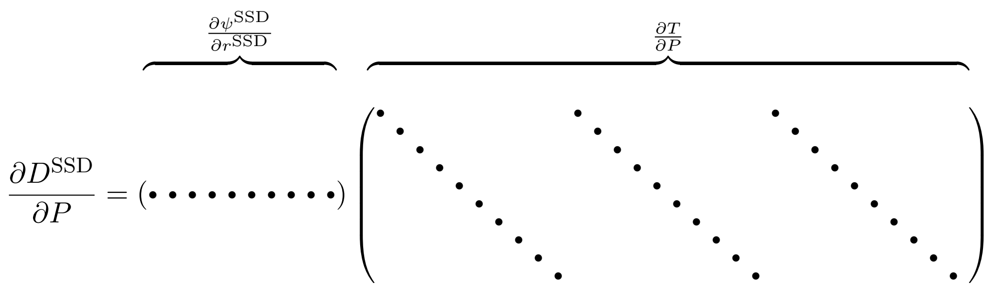

$\frac{\partial \psi^{\mathrm{SSD}}}{\partial r^{\mathrm{SSD}}}$
$\frac{\partial T}{\partial P}$
$\frac{\partial D^{\mathrm{SSD}}}{\partial P} =$

Figure 3.2.: Sparse matrix structure for computing the SSD gradient for $d = 3$.

are not stored. On the other hand, some calculations have to be performed multiple times. However, loading and storing values from and to memory also takes time for memory access and transfer. We will see later on that in many cases it is in fact faster to perform some recalculations, which is the central motivation of this thesis.

Another benefit is that single gradient elements can be computed independently, which enables direct parallelization. While for the SSD these computations are comparably simple, in the following sections the methods will also be applied to the more involved computations of the NGF.

The grid conversion and its Jacobian $\frac{\partial P}{\partial \mathbf{y}}$ will be described in detail in Section 3.5. Considering (3.14), the remaining components in (3.3) are

$$\frac{\partial D^{\mathrm{SSD}}}{\partial P} := \frac{\partial \psi^{\mathrm{SSD}}}{\partial r^{\mathrm{SSD}}} \frac{\partial T}{\partial P} \in \mathbb{R}^{d\bar{m}}.$$

A schematic view of the matrix structure of these computations is shown in Figure 3.2. The matrix-free computation of a SSD single gradient element can now be written as

$$\begin{aligned} \left(\frac{\partial D^{\mathrm{SSD}}}{\partial P}\right)_{i+l\bar{m}} &= 2\bar{h}\,\left(r^{\mathrm{SSD}}\right)_i \frac{\partial T_i}{\partial P_{i+l\bar{m}}} \\ &= 2\bar{h}\,(T_i - R_i)\,\frac{\partial T_i}{\partial P_{i+l\bar{m}}} \end{aligned} \tag{3.16}$$

with $i = 1, \ldots, \bar{m}$ and $l = 0, \ldots, d-1$, such that $\frac{\partial D^{\mathrm{SSD}}}{\partial P} \in \mathbb{R}^{d\bar{m}}$ as stated above. Substituting the image interpolation computations and their derivatives from Section 2.4.3, all elements can now be directly computed. Additionally, all elements can be computed in parallel, and no intermediate results need to be stored.

3.2.2. Hessian-vector multiplication

As described in Section 2.5.3 and (3.4), when using the Gauss-Newton optimizer, the computation of a Hessian-approximation is required. However, this requires first-order derivatives only, which have already been computed for the gradient. In a matrix-based framework, this makes the computation of the Gauss-Newton approximation of the Hessian comparably easy. However, storing the Hessian matrix approximation requires a significant amount of memory.

In order not to store the Hessian approximation in a matrix-free approach, in the following we derive a matrix-free matrix-vector multiplication, which is frequently required when solving (2.28), e.g., with a conjugate gradient iterative solver.

Using (3.4) and $\frac{\partial r^{\text{SSD}}}{\partial T} = I$, the Gauss-Newton Hessian approximation for the SSD can be written as

$$H^{\text{SSD}} = 2\bar{h}\frac{\partial P}{\partial \mathbf{y}}^\top \frac{\partial T}{\partial P}^\top \frac{\partial T}{\partial P}\frac{\partial P}{\partial \mathbf{y}}.$$

This computation not only requires the Jacobians of image interpolation and grid conversion, but also their transpose. Due to the diagonal structure, the transpose of $\frac{\partial T}{\partial P}$ is trivial (Figure 3.1). Again, for the transpose of the grid conversion Jacobian we refer to Section 3.5. Therefore, for the time being it is not considered here and we define

$$\hat{H}^{\text{SSD}} := \frac{\partial T}{\partial P}^\top \frac{\partial T}{\partial P}, \tag{3.17}$$

such that $H^{\text{SSD}} = 2\bar{h}\frac{\partial P}{\partial \mathbf{y}}^\top \hat{H}^{\text{SSD}}\frac{\partial P}{\partial \mathbf{y}}$. Using the known structure of $\frac{\partial T}{\partial P}$, we can now derive a matrix-free formulation for the matrix-vector multiplication

$$\hat{H}^{\text{SSD}}\,\hat{\mathbf{p}} = \frac{\partial T}{\partial P}^\top \frac{\partial T}{\partial P}\,\hat{\mathbf{p}} = \hat{\mathbf{q}}$$

with $\hat{\mathbf{p}}, \hat{\mathbf{q}} \in \mathbb{R}^{d\bar{m}}$ (Figure 3.3). A single element $\hat{\mathrm{q}}_i \in \mathbb{R}$ of the result vector $\hat{\mathbf{q}}$ can be computed as

$$\hat{\mathrm{q}}_{l\bar{m}+i} = \sum_{k=0}^{d-1} \frac{\partial T_i}{\partial P_{l\bar{m}+i}}\frac{\partial T_i}{\partial P_{k\bar{m}+i}}\;\hat{\mathrm{p}}_{k\bar{m}+i}, \tag{3.18}$$

with $i = 1, \dots, \bar{m}$ and $l = 0, \dots, d-1$. Similar to the gradient computation in (3.16), a single element of the result vector can be computed without the need for matrices or storage of intermediate results. Parallelization can be performed over all elements of the result vector. However, for each element the d image interpolation derivatives $\frac{\partial T}{\partial P_{l\bar{m}+i}}, l = 0, \dots, d$ need to be computed. Without storing of the image interpolation derivatives between computations of the elements $\hat{\mathrm{q}}_i$, this requires computing all image derivatives d times instead of just once, as in the matrix-based case. Therefore, compared to the gradient computation in Section 3.2.1, this involves an additional computational overhead due to multiple recalculations of image derivatives, see Section 3.7.

3.3. Derivative computations for NGF

From the definition of the discretized NGF in (2.13),

$$D^{\text{NGF}}(\mathbf{y}) = \bar{h}\sum_{i=1}^{\bar{m}}\left(1 - \left(\frac{\frac{1}{2}\langle\tilde{\nabla}T_i(P(\mathbf{y})), \tilde{\nabla}R_i(\mathbf{x})\rangle + \tau\varrho}{\|\tilde{\nabla}T_i(P(\mathbf{y}))\|_\tau \|\tilde{\nabla}R_i(\mathbf{x})\|_\varrho}\right)^2\right),$$

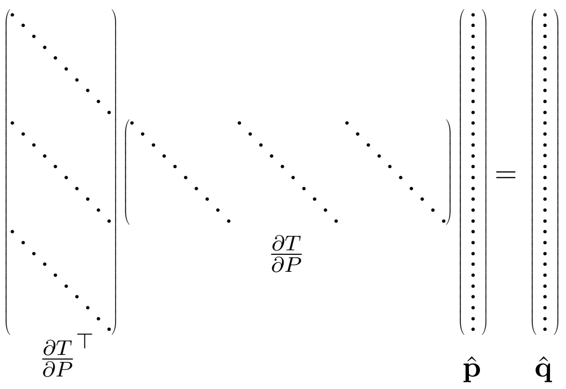

$\frac{\partial T}{\partial P}^\top$
$\frac{\partial T}{\partial P}$
$=$
$\hat{\mathbf{p}}$
$\hat{\mathbf{q}}$

Figure 3.3.: Sparse matrix structure of the SSD Hessian-vector multiplication $\hat{H}^{\text{SSD}}\,\hat{\mathbf{p}} = \hat{\mathbf{q}}$. The figure shows the matrix structures for $d = 3$.

it can be seen that, in comparison with the SSD (2.12), the residual function is more involved. Considering (2.13), we define $r^{\text{NGF}} : \mathbb{R}^{\bar{m}} \to \mathbb{R}^{\bar{m}}$ and

$$r_i^{\text{NGF}}(T) := \frac{\frac{1}{2}\langle \tilde{\nabla} T_i(P(\mathbf{y})), \tilde{\nabla} R_i(\mathbf{x})\rangle + \tau\varrho}{\|\tilde{\nabla} T_i(P(\mathbf{y}))\|_\tau \|\tilde{\nabla} R_i(\mathbf{x})\|_\varrho}, \tag{3.19}$$

such that

$$r^{\text{NGF}}(T) = \left(r_1^{\text{NGF}}(T), \dots, r_{\bar{m}}^{\text{NGF}}(T)\right).$$

In comparison to the SSD reduction function (3.12), the outer reduction function is slightly different with $\psi^{\text{NGF}} : \mathbb{R}^{\bar{m}} \to \mathbb{R}$ and

$$\psi^{\text{NGF}}(r) := \bar{h} \sum_{i=1}^{\bar{m}} \left(1 - r_i^2\right).$$

Using these definitions, analogously to (3.1), the NGF can be written as

$$D^{\text{NGF}}(\mathbf{y}) = \psi^{\text{NGF}}(r^{\text{NGF}}(T(P(\mathbf{y})))).$$

3.3.1. Gradient computations

Recalling the distance measure gradient decomposition (3.3), the NGF gradient is

$$\nabla D^{\text{NGF}}(\mathbf{y}) = \left(\frac{\partial \psi^{\text{NGF}}}{\partial r^{\text{NGF}}} \frac{\partial r^{\text{NGF}}}{\partial T} \frac{\partial T}{\partial P} \frac{\partial P}{\partial \mathbf{y}}\right)^\top \in \mathbb{R}^{d\bar{m}^{\mathbf{y}} \times 1}.$$

The computation of the derivative of the vector reduction $\psi^{\text{NGF}}(r)$ is straightforward:

$$\frac{\partial \psi^{\text{NGF}}}{\partial r^{\text{NGF}}} = -2\bar{h} r^{\text{NGF}} = -2\bar{h}\left(r_1^{\text{NGF}}, \dots, r_{\bar{m}}^{\text{NGF}}\right) \in \mathbb{R}^{1\times\bar{m}}. \tag{3.20}$$

Similar to Section 3.2.1, since the grid conversion and its Jacobian will be described in detail in Section 3.5, we will only consider the partial gradient

$$\begin{aligned}\frac{\partial D^{\mathrm{NGF}}}{\partial P} &:= \frac{\partial \psi^{\mathrm{NGF}}}{\partial r^{\mathrm{NGF}}}\frac{\partial r^{\mathrm{NGF}}}{\partial T}\frac{\partial T}{\partial P} \in \mathbb{R}^{1\times d\bar{m}} \\ &= -2\bar{h}r^{\mathrm{NGF}}\frac{\partial r^{\mathrm{NGF}}}{\partial T}\frac{\partial T}{\partial P} \in \mathbb{R}^{1\times d\bar{m}} \end{aligned} \tag{3.21}$$

in this section. Again, the goal is to derive fully matrix-free expressions for the gradient, that do not require the explicit formulation of the Jacobian matrices shown above.

While the derivative of r^{SSD} is the identity for the SSD in (3.14), computing the Jacobian matrix $\frac{\partial r^{\mathrm{NGF}}}{\partial T} \in \mathbb{R}^{\bar{m}\times\bar{m}}$ is more involved. We first focus on the computation of a single row of the Jacobian, denoted as $\frac{\partial r_i^{\mathrm{NGF}}}{\partial T} \in \mathbb{R}^{1\times\bar{m}}$ and further simplify the calculations by splitting r_i^{NGF} into sub-expressions.

From (3.19), the residual $r_i^{\mathrm{NGF}} : \mathbb{R}^{\bar{m}} \to \mathbb{R}$ can be interpreted as the quotient

$$r_i^{\mathrm{NGF}} = \frac{r_{1,i}^{\mathrm{NGF}}}{r_{2,i}^{\mathrm{NGF}}}, \tag{3.22}$$

with

$$r_{1,i}^{\mathrm{NGF}} := \frac{1}{2}\langle\tilde{\nabla}T_i, \tilde{\nabla}R_i\rangle + \tau\varrho \tag{3.23}$$

$$r_{2,i}^{\mathrm{NGF}} := \|\tilde{\nabla}T_i\|_\tau\|\tilde{\nabla}R_i\|_\varrho. \tag{3.24}$$

Numerator and denominator can now be differentiated separately, yielding

$$\frac{\partial r_{1,i}^{\mathrm{NGF}}}{\partial T} = \frac{1}{2}(\tilde{\nabla}R_i)^\top\frac{\partial\tilde{\nabla}_i}{\partial T} \in \mathbb{R}^{1\times\bar{m}} \tag{3.25}$$

$$\frac{\partial r_{2,i}^{\mathrm{NGF}}}{\partial T} = \|\tilde{\nabla}R_i\|_\varrho\frac{(\tilde{\nabla}T_i)^\top}{2\|\tilde{\nabla}T_i\|_\tau}\frac{\partial\tilde{\nabla}_i}{\partial T} \in \mathbb{R}^{1\times\bar{m}}. \tag{3.26}$$

Additionally, both terms contain the Jacobian $\frac{\partial\tilde{\nabla}_i}{\partial T}$ of the finite-differences gradient computation. As defined in (2.19), the function $\tilde{\nabla} : \mathbb{R}^{\bar{m}} \to \mathbb{R}^{2d}$ maps an image to both forward and backward difference gradients. Correspondingly, $\tilde{\nabla}_i : \mathbb{R}^{\bar{m}} \to \mathbb{R}$ is a single component function, computing the gradient for a single point of the image. As the gradient at that point only depends on the values at the point itself and its neighboring points, for $d = 3$, the Jacobian $\frac{\partial\tilde{\nabla}_i}{\partial T} \in \mathbb{R}^{2d\times\bar{m}}$ is of the form

$$\frac{\partial\tilde{\nabla}_i}{\partial T} = \begin{array}{c} \begin{array}{ccccccc} i_{-z} & i_{-y} & i_{-x} & i & i_{+x} & i_{+y} & i_{+z} \end{array} \\ \begin{pmatrix} & & \frac{-1}{h_x} & \frac{1}{h_x} & & & \\ & \frac{-1}{h_y} & & \frac{1}{h_y} & & & \\ \frac{-1}{h_z} & & & \frac{1}{h_z} & & & \\ & & & \frac{-1}{h_x} & \frac{1}{h_x} & & \\ & & & \frac{-1}{h_y} & & \frac{1}{h_y} & \\ & & & \frac{-1}{h_z} & & & \frac{1}{h_z} \end{pmatrix} \end{array}, \tag{3.27}$$

where only non-zero elements are shown and the non-zero column indices are displayed above the matrix. Note that for the column indices the notation for boundary values from (2.16) – (2.18) has been used. This way, if boundary conditions become active, column indices are automatically correct. For $d = 2$, the Jacobian is

$$\frac{\partial \tilde{\nabla}_i}{\partial T} = \begin{array}{c} \begin{array}{ccccc} i_{-y} & i_{-x} & i & i_{+x} & i_{+y} \end{array} \\ \begin{pmatrix} & \frac{-1}{h_x} & \frac{1}{h_x} & & \\ \frac{-1}{h_y} & & \frac{1}{h_y} & & \\ & & \frac{-1}{h_x} & \frac{1}{h_x} & \\ & & \frac{-1}{h_y} & & \frac{1}{h_y} \end{pmatrix} \end{array}. \tag{3.28}$$

In both cases, the locations of the non-zero matrix elements are fixed and the matrix exhibits a very sparse pattern. With the elements of $\frac{\partial \tilde{\nabla}_i}{\partial T}$ known, we can now continue the derivation of r_i^{NGF}.

The quotient rule applied to (3.22) leads to

$$\frac{\partial r_i^{\text{NGF}}}{\partial T} = \frac{1}{\left(r_{2,i}^{\text{NGF}}\right)^2} \left(\frac{\partial r_{1,i}^{\text{NGF}}}{\partial T} r_{2,i}^{\text{NGF}} - r_{1,i}^{\text{NGF}} \frac{\partial r_{2,i}^{\text{NGF}}}{\partial T} \right) \tag{3.29}$$

$$= \frac{\partial r_{1,i}^{\text{NGF}}}{\partial T} \frac{1}{r_{2,i}^{\text{NGF}}} - \frac{r_{1,i}^{\text{NGF}}}{\left(r_{2,i}^{\text{NGF}}\right)^2} \frac{\partial r_{2,i}^{\text{NGF}}}{\partial T}. \tag{3.30}$$

Substituting (3.23), (3.24) for $r_{1,i}^{\text{NGF}}$ and $r_{2,i}^{\text{NGF}}$, and (3.25), (3.26) for the corresponding derivatives, it holds

$$\begin{aligned} \frac{\partial r_i^{\text{NGF}}}{\partial T} &= \frac{(\tilde{\nabla} R_i)^\top}{2\|\tilde{\nabla} T_i\|_\tau \|\tilde{\nabla} R_i\|_\varrho} \frac{\partial \tilde{\nabla}_i}{\partial T} - \frac{\frac{1}{2}\langle \tilde{\nabla} T_i, \tilde{\nabla} R_i\rangle + \tau\varrho}{\|\tilde{\nabla} T_i\|_\tau^2 \|\tilde{\nabla} R_i\|_\varrho^2} \|\tilde{\nabla} R_i\|_\varrho \frac{(\tilde{\nabla} T_i)^\top}{2\|\tilde{\nabla} T_i\|_\tau} \frac{\partial \tilde{\nabla}_i}{\partial T} \\ &= \frac{1}{2} \left(\frac{(\tilde{\nabla} R_i)^\top}{\|\tilde{\nabla} T_i\|_\tau \|\tilde{\nabla} R_i\|_\varrho} \frac{\partial \tilde{\nabla}_i}{\partial T} - \frac{\frac{1}{2}\langle \tilde{\nabla} T_i, \tilde{\nabla} R_i\rangle + \tau\varrho}{\|\tilde{\nabla} T_i\|_\tau^3 \|\tilde{\nabla} R_i\|_\varrho} (\tilde{\nabla} T_i)^\top \frac{\partial \tilde{\nabla}_i}{\partial T} \right), \end{aligned} \tag{3.31}$$

with $\frac{\partial r_i^{\text{NGF}}}{\partial T} \in \mathbb{R}^{1\times\bar{m}}$. This expression already contains computations that are explicit enough to compute a row of the Jacobian from the previous derivations. However, it still contains vector and matrix operations with large, sparse matrices, such as $\frac{\partial \tilde{\nabla}_i}{\partial T}$, given in (3.27) and (3.28). The full Jacobian $\frac{\partial r^{\text{NGF}}}{\partial T}$ exhibits a matrix structure with $2d+1$ diagonals. Together with the derivatives of the image interpolation $\frac{\partial T}{\partial P}$ and $\frac{\partial \psi^{\text{NGF}}}{\partial r^{\text{NGF}}}$ in (3.20), the matrix structures for computing $\frac{\partial D^{\text{NGF}}}{\partial P}$ are visualized in Figure 3.4. In comparison with Figure 3.2, the structure of the computation differs by the $2d+1$ banded Jacobian matrix $\frac{\partial r^{\text{NGF}}}{\partial T}$.

In order to derive a matrix-free expression for a single element of $\frac{\partial D^{\text{NGF}}}{\partial P}$, from (3.31), we introduce the abbreviation

$$\rho_i(k) := \frac{-R_i + R_{i_k}}{\|\tilde{\nabla} T_i\|_\tau \|\tilde{\nabla} R_i\|_\varrho} - \frac{\left(\frac{1}{2}\langle \tilde{\nabla} T_i, \tilde{\nabla} R_i\rangle + \tau\varrho\right)(-T_i + T_{i_k})}{\|\tilde{\nabla} T_i\|_\tau^3 \|\tilde{\nabla} R_i\|_\varrho} \in \mathbb{R}. \tag{3.32}$$

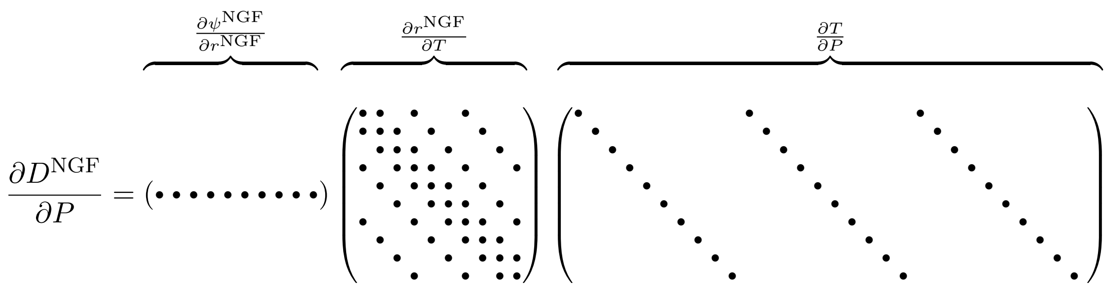


Figure 3.4.: Sparse matrix structure for the computation of the NGF gradient for $d = 3$.

Compared to (3.31), the Jacobian matrix $\frac{\partial \tilde{\nabla}_i}{\partial T}$ still needs to be considered

First, we define the set of indices k, to be used in (3.32), corresponding to the neighboring indices in each direction with

$$\mathcal{K} := \{-z, -y, -x, 0, x, y, z\}, \tag{3.33}$$

for $d = 3$ and $\mathcal{K} := \{-y, -x, 0, x, y\}$ for $d = 2$. Then, with the weights from (3.27), (3.28), abbreviated as

$$\hat{h}_k := \frac{1}{2h_{|k|}^2},$$

the Jacobian matrix $\frac{\partial \tilde{\nabla}_i}{\partial T}$ can be incorporated into (3.32). This results in

$$\hat{\rho}_i(k) := \begin{cases} \sum_{j \in \mathcal{K} \setminus \{0\}} -\hat{h}_j \rho_i(j) & \text{if } k = 0, \\ \hat{h}_k \rho_i(k) & \text{otherwise,} \end{cases} \tag{3.34}$$

with $\hat{\rho}_i(k) : \mathcal{K} \to \mathbb{R}$, representing the non-zero elements of the Jacobian matrix row $\frac{\partial r_i^{\text{NGF}}}{\partial T}$. The matrix row can finally be compactly written as

$$\frac{\partial r_i^{\text{NGF}}}{\partial T} = \begin{array}{c} \begin{array}{ccccccc} i_{-z} & i_{-y} & i_{-x} & i & i_{+x} & i_{+y} & i_{+z} \end{array} \\ \begin{pmatrix} \hat{\rho}_i(-z) & \hat{\rho}_i(-y) & \hat{\rho}_i(-x) & \hat{\rho}_i(0) & \hat{\rho}_i(x) & \hat{\rho}_i(y) & \hat{\rho}_i(z) \end{pmatrix} \end{array} \in \mathbb{R}^{1 \times \bar{m}}, \tag{3.35}$$

where only non-zero elements are shown and column indices are indicated above the vector. For $d = 2$, we have

$$\frac{\partial r_i^{\text{NGF}}}{\partial T} = \begin{array}{c} \begin{array}{ccccc} i_{-y} & i_{-x} & i & i_{+x} & i_{+y} \end{array} \\ \begin{pmatrix} \hat{\rho}_i(-y) & \hat{\rho}_i(-x) & \hat{\rho}_i(0) & \hat{\rho}_i(x) & \hat{\rho}_i(y) \end{pmatrix} \end{array} \in \mathbb{R}^{1 \times \bar{m}}. \tag{3.36}$$

As in Figure 3.4, the structure of $\frac{\partial r^{\text{NGF}}}{\partial T}$ as a $2d+1$ banded diagonal matrix is apparent.

Exploiting this pattern, a single element of the partial gradient $\frac{\partial D^{\text{NGF}}}{\partial P}$, given in (3.21), can now explicitly be computed as

$$\left(\frac{\partial D^{\text{NGF}}}{\partial P} \right)_{i+l\bar{m}} = -2\,\bar{h} \left(\sum_{k \in \mathcal{K}} r_{i_k}^{\text{NGF}} \hat{\rho}_{i_k}(-k) \right) \frac{\partial T_i}{\partial P_{i+l\bar{m}}}, \tag{3.37}$$

where $i = 1, \dots, \bar{m}$, $l = 0, \dots, d$. Using the explicit descriptions of r_i^{NGF} in (3.19), $\hat{\rho}_i$ in (3.34) and $\frac{\partial T_i}{\partial P_{i+l\bar{m}}}$ in (3.9) – (3.11), each element of the gradient can be computed directly from the input data, independently and in parallel. This allows for a straightforward element-wise parallel implementation.

3.3.2. Hessian-vector multiplication

We will consider the grid conversion Jacobian matrix $\frac{\partial P}{\partial \mathbf{y}}$ separately in Section 3.5 and only compute the partial Hessian approximation. Similar to Section 3.2.2,

$$\hat{H}^{\mathrm{NGF}} := \frac{\partial T}{\partial P}^{\top} \frac{\partial r^{\mathrm{NGF}}}{\partial T}^{\top} \frac{\partial^2 \psi^{\mathrm{NGF}}}{\partial r^{\mathrm{NGF}^2}} \frac{\partial r^{\mathrm{NGF}}}{\partial T} \frac{\partial T}{\partial P}, \tag{3.38}$$

such that

$$H^{\mathrm{NGF}} = \frac{\partial P}{\partial \mathbf{y}}^{\top} \hat{H}^{\mathrm{NGF}} \frac{\partial P}{\partial \mathbf{y}}.$$

Differentiating (3.20), we obtain

$$\frac{\partial^2 \psi^{\mathrm{NGF}}}{\partial r^{\mathrm{NGF}^2}} = -2\bar{h}.$$

In contrast to the Gauss-Newton method in (2.32), the derivative of the outer function ψ^{NGF} has a different sign. Therefore, in order to obtain descent directions in (2.28), the sign of the Hessian approximation has been inverted, as proposed in [Mod09, §7.4]. This leads to a first simplification of (3.38) with

$$\hat{H}^{\mathrm{NGF}} = 2\bar{h} \frac{\partial T}{\partial P}^{\top} \frac{\partial r^{\mathrm{NGF}}}{\partial T}^{\top} \frac{\partial r^{\mathrm{NGF}}}{\partial T} \frac{\partial T}{\partial P}.$$

As can be seen from (3.4), besides the image interpolation derivatives $\frac{\partial T}{\partial P}$ already investigated in Section 3.2.2, the Hessian approximation contains the matrix multiplication $\frac{\partial r^{\mathrm{NGF}}}{\partial T}^{\top} \frac{\partial r^{\mathrm{NGF}}}{\partial T}$. The overall goal is to derive a matrix-free expression for the matrix-vector multiplication

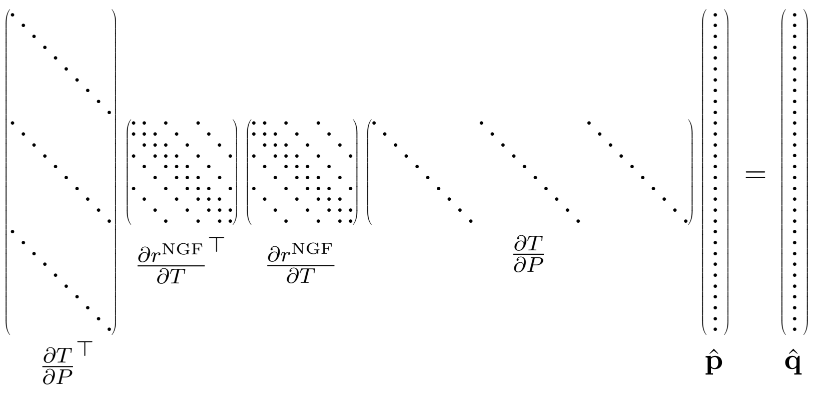
$$\hat{H}^{\mathrm{NGF}} \hat{\mathbf{p}} = \hat{\mathbf{q}},$$

with $\hat{\mathbf{p}}, \hat{\mathbf{q}} \in \mathbb{R}^{d\bar{m}}$, which can be schematically visualized as shown in Figure 3.5.

Abbreviating

$$dr := \frac{\partial r^{\mathrm{NGF}}}{\partial T},$$

the main challenge lies in the matrix-free computation of $dr^{\top} dr$. Performing a standard matrix multiplication with $dr \in \mathbb{R}^{\bar{m} \times \bar{m}}$, a single element $\left(dr^{\top} dr\right)_{i,j}$ of the result is the



$\frac{\partial T}{\partial P}^\top$
$\frac{\partial r^{\mathrm{NGF}}}{\partial T}^\top$
$\frac{\partial r^{\mathrm{NGF}}}{\partial T}$
$\frac{\partial T}{\partial P}$
$\hat{\mathbf{p}}$
$=$
$\hat{\mathbf{q}}$

Figure 3.5.: Sparse matrix structure of the NGF Hessian-vector multiplication $\hat{H}^{\mathrm{NGF}}\hat{\mathbf{p}} = \hat{\mathbf{q}}$ for $d = 3$. The computation involves highly sparse matrices with a fixed pattern of non-zeros. The main computational effort results from the product $\frac{\partial r^{\mathrm{NGF}}}{\partial T}^\top \frac{\partial r^{\mathrm{NGF}}}{\partial T}$, where $\frac{\partial r^{\mathrm{NGF}}}{\partial T}$ has $2d + 1$ non-zero diagonals.

scalar product of column i and j,

$$\begin{aligned}\left(dr^\top dr\right)_{i,j} &= \sum_{k=1}^{d\bar{m}} \left(dr^\top\right)_{i,k} dr_{k,j} \\ &= \sum_{k=1}^{d\bar{m}} dr_{k,i}\, dr_{k,j},\end{aligned} \tag{3.39}$$

where $dr_{i,j}$ denotes the element at the i-th row and the j-th column of the matrix dr. Abbreviating the i-th column of dr as

$$dr_i := \frac{\partial r_i^{\mathrm{NGF}}}{\partial T} = dr_{k,i}, \;\; k = 1, \ldots, d\bar{m}$$

with $dr_i \in \mathbb{R}^{\bar{m}\times 1}$, (3.39) becomes

$$\left(dr^\top dr\right)_{i,j} = \langle dr_i, dr_j\rangle. \tag{3.40}$$

However, from the previous section and especially (3.35), we already know that dr is very sparse, such that most elements of the scalar product (3.39) will be zero. The remainder of this section is dedicated to exactly identifying the non-zero elements for all elements in $dr^\top dr$ and to deriving a closed form matrix-free expression.

Similar to (3.35), a single *column* is of the form

$$dr_i = \begin{array}{c} \begin{array}{ccccccc} i_{-z} & i_{-y} & i_{-x} & i & i_{+x} & i_{+y} & i_{+z} \end{array} \\ \left(\begin{array}{ccccccc} \hat{\rho}_{i_{-z}}(z) & \hat{\rho}_{i_{-y}}(y) & \hat{\rho}_{i_{-x}}(x) & \hat{\rho}_i(0) & \hat{\rho}_{i_x}(-x) & \hat{\rho}_{i_y}(-y) & \hat{\rho}_{i_z}(-z) \end{array}\right)^\top \end{array} \in \mathbb{R}^{\bar{m}\times 1}, \tag{3.41}$$

for $d = 3$ and analogously to (3.36) for $d = 2$.

Diagonal elements. We first consider the diagonal elements of the matrix multiplication $dr^\top dr$, i.e., $\langle dr_i, dr_i\rangle,\ i = 1, \dots, \bar{m}$, before advancing to the general case $\langle dr_i, dr_j\rangle$. From (3.41), it can be seen that in dr_i, the non-zero elements are located at $(dr_i)_{i_k}$, $k \in \mathcal{K}$, with $\mathcal{K}$ as in (3.33). Therefore,

$$\langle dr_i, dr_i\rangle = \sum_{k\in\mathcal{K}} (dr_i)^2_{i_k} = \sum_{k\in\mathcal{K}} \hat{\rho}_{i_k}(-k)^2,$$

which is a sum of $2d + 1$ terms.

General case. For the general case $\langle dr_i, dr_j\rangle$ with arbitrary i, j, we define $\kappa := j - i$, such that $\langle dr_i, dr_j\rangle = \langle dr_i, dr_{i+\kappa}\rangle$. Using (3.41), it can be seen that in dr_i, the non-zero elements are located at indices $i + \mathcal{M}$, with

$$\mathcal{M} := \{-m_x m_y, -m_x, -1, 0, 1, m_x, m_x m_y\}$$

and $\mathcal{M} := \{-m_x, -1, 0, 1, m_x\}$ for $d = 2$, for indices with $0 < i + \mathcal{M} \leq \bar{m}$.

The elements of the scalar product $\langle dr_i, dr_{i+\kappa}\rangle$ can only be non-zero if the two vectors dr_i and $dr_{i+\kappa}$ have non-zero elements at one or more *identical* indices. Using the previous definitions, this is fulfilled if

$$(i + \mathcal{M}) \cap (i + \kappa + \mathcal{M}) \neq \emptyset. \tag{3.42}$$

From this we can define a new set

$$\mathcal{N} := \mathcal{M} - \mathcal{M} \tag{3.43}$$

of indices $\kappa \in \mathcal{N}$ fulfilling (3.42), so that the scalar product $\langle dr_i, dr_{i+\kappa}\rangle,\ \kappa \in \mathcal{N}$ is potentially non-zero. Considering (3.43), the number of elements $|\mathcal{N}|$ is bounded by the number of element pairs $\mathcal{M}$, giving $|\mathcal{N}| \leq |\mathcal{M}| \cdot |\mathcal{M}|$. Recalling that the scalar products originate from the non-zero elements in the matrix $dr^\top dr$, this bound directly translates to a maximum number of $|\mathcal{N}| \leq 49$ non-zero elements per row in $dr^\top dr$ for $d = 3$ and $|\mathcal{N}| \leq 25$ for $d = 2$.

However, we will see that the actual number of diagonal elements that need to be computed is substantially lower. To determine the exact number of elements $|N|$, we define the mapping $N : \mathcal{M} \times \mathcal{M} \to \mathbb{Z}$ with

$$N(i, j) := j - i.$$

Using this mapping, the set $\mathcal{N}$ can be written as

$$\mathcal{N} = N(\mathcal{M}, \mathcal{M}) = \left\{\kappa \in \mathbb{Z} \;\middle|\; \kappa = N(i, j),\ i, j \in \mathcal{M}\right\}.$$

From this representation, all elements of the set $\mathcal{N}$ can be determined explicitly, as shown in Table 3.1. We find that the number of non-zero elements in the set is $|\mathcal{N}| = 25$ for $d = 3$ and $|\mathcal{N}| = 13$ for $d = 2$, which also gives the maximum number of elements for each column in $dr^\top dr$.

With the pre-image $N^{-1}(\kappa)$, the exact indices of the non-zero elements contributing to the scalar products $\langle dr_i, dr_j\rangle$ can be derived. This can be used to construct an explicit

$\kappa \in \mathcal{N}$	$N^{-1}(\kappa)$		$\lvert N^{-1}(\kappa)\rvert$	
$-2m_1m_2$	$\{(m_1m_2, -m_1m_2)\}$		1	
$-m_1m_2 - m_1$	$\{(m_1, -m_1m_2),$	$(m_1m_2, -m_1)\}$	2	
$-m_1m_2 - 1$	$\{(1, -m_1m_2),$	$(m_1m_2, -1)\}$	2	
$-m_1m_2$	$\{(0, -m_1m_2),$	$(m_1m_2, 0)\}$	2	$d = 3$
$-m_1m_2 + 1$	$\{(-1, -m_1m_2),$	$(m_1m_2, 1)\}$	2	
$-m_1m_2 + m_1$	$\{(-m_1, -m_1m_2),$	$(m_1m_2, m_1)\}$	2	
$-2m_1$	$\{(m_1, -m_1)$		1	
$-m_1 - 1$	$\{(1, -m_1),$	$(m_1, -1)\}$	2	
$-m_1$	$\{(0, -m_1),$	$(m_1, 0)\}$	2	
$-m_1 + 1$	$\{(-1, -m_1),$	$(m_1, 1)\}$	2	
-2	$\{(1, -1)\}$		1	
-1	$\{(0, -1),$	$(1, 0)\}$	2	
0	$\{(\hat{\imath}, \hat{\imath}) \mid \hat{\imath} \in \mathcal{M}\}$		$2d + 1$	$d = \{2, 3\}$
1	$\{(0, 1),$	$(-1, 0)\}$	2	
2	$\{(-1, 1)\}$		1	
$m_1 - 1$	$\{(1, m_1),$	$(-m_1, -1)\}$	2	
m_1	$\{(0, m_1),$	$(-m_1, 0)\}$	2	
$m_1 + 1$	$\{(-1, m_1),$	$(-m_1, 1)\}$	2	
$2m_1$	$\{(-m_1, m_1)$		1	
$m_1m_2 - m_1$	$\{(m_1, m_1m_2),$	$(-m_1m_2, -m_1)\}$	2	
$m_1m_2 - 1$	$\{(1, m_1m_2),$	$(-m_1m_2, -1)\}$	2	
m_1m_2	$\{(0, m_1m_2),$	$(-m_1m_2, 0)\}$	2	$d = 3$
$m_1m_2 + 1$	$\{(-1, m_1m_2),$	$(-m_1m_2, 1)\}$	2	
$m_1m_2 + m_1$	$\{(-m_1, m_1m_2),$	$(-m_1m_2, m_1)\}$	2	
$2m_1m_2$	$\{(-m_1m_2, m_1m_2)\}$		1	

Table 3.1.: Offsets $\kappa := j - i$ for which $\langle dr_i, dr_j \rangle \neq 0$, corresponding to non-zero elements $(dr^\top dr)_{i,j}$ in the matrix-product $dr^\top dr$. The maximum number of elements in each row is $|\mathcal{N}| = 25$ for $d = 3$ and $|\mathcal{N}| = 13$ for $d = 2$. The pre-image sets $N^{-1}(\kappa)$ indicate the locations of each non-zero element in dr_i and dr_j. The third column shows the number of products involving non-zero coefficients in the evaluation of $\langle dr_i, dr_j \rangle$, and sums to $|\mathcal{M}| \cdot |\mathcal{M}|$.

formulation of a single element $\left(dr^\top dr\right)_{i,j}$, involving only non-zero elements. As shown in Table 3.1, for the main diagonal of $dr^\top dr$, $|N^{-1}(0)| = 2d + 1$ non-zero elements are involved in the scalar product, while for all other scalar products only one or two elements are non-zero. With $\kappa = j - i$ and $(\hat{\imath}, \hat{\jmath}) \in N^{-1}(\kappa)$, the elements of the matrix $dr^\top dr$ can be computed as

$$
\begin{aligned}
\left(dr^\top dr\right)_{i,j} &= \begin{cases} \displaystyle\sum_{(\hat{\imath},\hat{\jmath}) \in N^{-1}(j-i)} (dr_i)_{i+\hat{\imath}} \, (dr_j)_{j+\hat{\jmath}}\,, & \text{if } j - i \in \mathcal{N}, \\ 0, & \text{otherwise,} \end{cases} \\
&= \begin{cases} \displaystyle\sum_{(\hat{\imath},\hat{\jmath}) \in N^{-1}(j-i)} \hat{\rho}_{i+\hat{\imath}}(-\hat{\imath}) \; \hat{\rho}_{j+\hat{\jmath}}(-\hat{\jmath}), & \text{if } j - i \in \mathcal{N}, \\ 0, & \text{otherwise,} \end{cases}
\end{aligned} \tag{3.44}
$$

where, in a slight abuse of notation, $\hat{\rho}(\pm m_1 m_2), \hat{\rho}(\pm m_1), \hat{\rho}(\pm 1)$ should be interpreted as $\hat{\rho}(\pm z), \hat{\rho}(\pm y), \hat{\rho}(\pm x)$, respectively.

By substituting the explicit formulation of $\hat{\rho}_i(\kappa)$ from (3.34) into (3.44), a complete formulation of each element of $dr^\top dr$ is now available. Compared to a matrix-matrix multiplication, the computational costs are greatly reduced: Only non-zero elements are considered and no handling, locating and storing of sparse matrix elements is needed.

Coming back to Figure 3.5, in order to compute the final matrix-vector multiplication $\hat{H}^{\mathrm{NGF}} \hat{\mathbf{p}} = \hat{\mathbf{q}}$, we need to right-multiply with the image interpolation derivatives $\frac{\partial T}{\partial P}$ and left-multiply with its transpose $\frac{\partial T}{\partial P}^\top$. Again, the block diagonal structure of the image derivative matrices can be easily exploited, so that for the final matrix-vector product $\hat{H}^{\mathrm{NGF}} \hat{\mathbf{p}} = \hat{\mathbf{q}}$, we obtain

$$\hat{\mathrm{q}}_{l\bar{m}+i} = \sum_{\kappa \in \mathcal{N}} \sum_{s=0}^{d-1} \frac{\partial T_i}{\partial P_{l\bar{m}+i}} \left(dr^\top dr \right)_{i,i+\kappa} \frac{\partial T_\kappa}{\partial P_{s\bar{m}+i+\kappa}} \, \hat{\mathrm{p}}_{s\bar{m}+i+\kappa}, \tag{3.45}$$

for $l = 0, \ldots, d-1$ and $i = 1, \ldots, \bar{m}$, using the definition of $\left(dr^\top dr \right)_{i,i+\kappa}$ from (3.44) and the image derivatives given in (3.9) – (3.11). Again, each element of the result vector can be computed in parallel and no intermediate storage is required.

3.4. Derivative computations for curvature regularization

Compared to the distance measures, the derivative calculations of the curvature regularizer are much simpler. The curvature regularizer is defined in (2.21) as

$$S(\mathbf{y}) = \bar{h}^{\mathrm{y}} \sum_{i=1}^{\bar{m}^{\mathrm{y}}} \sum_{j=1}^{d} \left(\tilde{\Delta} \mathbf{u}_{i+(j-1)\bar{m}^{\mathrm{y}}} \right)^2 .$$

As described in (2.23), in a matrix-based formulation,

$$S(\mathbf{y}) = \bar{h}^{\mathrm{y}} (A\mathbf{u})^\top A\mathbf{u} = \bar{h}^{\mathrm{y}} \mathbf{u}^\top A^\top A\mathbf{u},$$

where $A \in \mathbb{R}^{d\bar{m}^{\mathrm{y}} \times d\bar{m}^{\mathrm{y}}}$ is the corresponding curvature matrix operator as defined in [Hel06, §4.4.4], resulting from the weights of the finite difference stencil in (2.22) and Neumann boundary conditions as given in Section 2.4.5. With this, the gradient is

$$\nabla S(\mathbf{y}) = 2\bar{h}^{\mathrm{y}} A^\top A\mathbf{u}, \tag{3.46}$$

which, using the discretized Laplace operator from (2.22) and $\tilde{\Delta}\mathbf{u} := \left(\tilde{\Delta}\mathbf{u}_1, \ldots, \tilde{\Delta}\mathbf{u}_{d\bar{m}^{\mathrm{y}}} \right)$, can also be written as

$$\nabla S(\mathbf{y}) = 2\, \bar{h}^{\mathrm{y}} \tilde{\Delta} \left(\tilde{\Delta}\mathbf{u} \right).$$

Furthermore, the Hessian matrix of the regularizer

$$\nabla^2 S(\mathbf{y}) = 2\bar{h}^{\mathrm{y}} A^\top A \tag{3.47}$$

is constant and can easily be computed exactly. Thus, no Gauss-Newton approximation of the Hessian is needed for the regularizer.

3.4.1. Gradient computations

A matrix-free computation of the curvature regularizer gradient can easily be performed by point wise applying the symmetric discretized Laplace operator $\tilde{\Delta}$ twice. This can be performed independently for each point with

$$(\nabla S(\mathbf{y}))_{i+l\bar{m}^{\mathrm{y}}} = 2\,\bar{h}^{\mathrm{y}}\tilde{\Delta}\left(\tilde{\Delta}\mathbf{u}\right)_{i+l\bar{m}^{\mathrm{y}}} \tag{3.48}$$

and $l = 0, \ldots, d-1$, $i = 1, \ldots, \bar{m}^{\mathrm{y}}$. As the curvature regularizer operates directly on the deformation grid, no further grid conversion steps are necessary and (3.48) constitutes the final gradient.

3.4.2. Hessian computations

The gradient computation (3.48) and a Hessian-vector multiplication $\nabla^2 S\mathbf{p} = \mathbf{q}$ are closely related (see (3.46) and (3.47)), so that

$$\begin{aligned}
\mathrm{q}_{i+l\bar{m}^{\mathrm{y}}} &= \left(\nabla^2 S\mathbf{p}\right)_{i+l\bar{m}^{\mathrm{y}}} \\
&= (\nabla S(\mathbf{p}+\mathbf{x}^{\mathrm{y}}))_{i+l\bar{m}^{\mathrm{y}}} \\
&= 2\bar{h}^{\mathrm{y}}\tilde{\Delta}\left(\tilde{\Delta}\mathbf{p}\right)_{i+l\bar{m}^{\mathrm{y}}},
\end{aligned} \tag{3.49}$$

for $l = 0, \ldots, d-1$ and $i = 1, \ldots, \bar{m}^{\mathrm{y}}$. While the (approximated) Hessian-matrix multiplication for the distance measures is much more complicated than the gradient, for the curvature regularizer it can be computed with the same effort.

In contrast to the distance measure Hessian approximations, however, in our registration scheme, the Hessian of the curvature regularizer is not only required for Gauss-Newton optimization, but also for the initial value H_0 for the Hessian approximation in the L-BFGS scheme (Section 2.5.2). The latter requires a multiplication with the inverse $(H_0)^{-1}$. Instead of directly inverting the matrix, a system of linear equations is solved with an iterative solver as described in Section 2.5.2. Here, the matrix-vector multiplication $\nabla^2 S\mathbf{p} = \mathbf{q}$ needs to be performed multiple times per iteration, such that a fast evaluation is essential.

3.5. Grid conversion

Two different discretization grids are used in this work: the *image grid* $\mathbf{x} \in \mathbb{R}^{\bar{m}}$, defined on the reference image domain $\Omega_{\mathcal{R}}$ with resolution h and size m, and the *deformation grid* $\mathbf{x}^{\mathrm{y}} \in \mathbb{R}^{\bar{m}^{\mathrm{y}}}$, which determines the resolution h^{y} and size m^{y} of the deformation $\mathbf{y}$ (see also Section 2.4.1). As the size of the system of linear equations (2.28) is determined solely by the deformation grid size, this allows to choose the deformation grid resolution coarser than the resolution of the image grid, decreasing computation time while still preserving the image resolution. However, this requires a *grid conversion* function in order to map the deformation between both grids.

3.5.1. Image grid to deformation grid

The grid conversion function (2.8)

$$P : \mathbb{R}^{d\bar{m}^{\mathrm{y}}} \to \mathbb{R}^{d\bar{m}},$$

maps from the deformation grid to the image grid. This function is frequently required in all discussed distance measures in order to evaluate the function value (see (3.1)). For a fast and efficient grid conversion, we use tri-linear interpolation for $d = 3$ and bi-linear interpolation for $d = 2$, such that the grid conversion can be written as a matrix-vector multiplication

$$\hat{\mathbf{y}} = P\mathbf{y} \tag{3.50}$$

with $P \in \mathbb{R}^{d\bar{m} \times d\bar{m}^{\mathrm{y}}}$, $\mathbf{y} \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}$ and $\hat{\mathbf{y}} \in \mathbb{R}^{d\bar{m}}$. Since the deformation consists of d components, each of these need to be interpolated. As each component is interpolated at the same location, P has a block-diagonal structure of d interpolation matrices $\tilde{P} \in \mathbb{R}^{\bar{m} \times \bar{m}^{\mathrm{y}}}$ with $P = \operatorname{diag}(\tilde{P}, \tilde{P}, \tilde{P})$ for $d = 3$ and $P = \operatorname{diag}(\tilde{P}, \tilde{P})$ for $d = 2$. An example of two grids and the corresponding matrix $\tilde{P}$ is shown in Figure 3.6.

For computing the grid conversion in a matrix-free way, the interpolation described in Section 3.1.1 can be used. However, since the deformation grid is always given with a *nodal* discretization (Section 2.4.1) each image grid point is always surrounded by deformation grid points. Therefore no checks for boundary conditions need to be performed, see also Figure 3.6(a). As can be seen in (3.8), each interpolated point $\hat{\mathrm{y}}_i$ is a weighted sum of its 2^d neighbors on the deformation grid, shown for $d = 2$ in Figure 3.7(a). Using the definitions from Section 3.1.1 with h^{y} instead of h in (3.7), it holds

$$\hat{\mathrm{y}}_{i+l\bar{m}} = \sum_{\alpha=0}^{1} \sum_{\beta=0}^{1} \sum_{\gamma=0}^{1} w(\hat{\mathbf{y}}_i)_1^{\alpha} w(\hat{\mathbf{y}}_i)_2^{\beta} w(\hat{\mathbf{y}}_i)_3^{\gamma} \; \mathrm{y}_{\xi_i(\alpha,\beta,\gamma)+l\bar{m}^{\mathrm{y}}}, \tag{3.51}$$

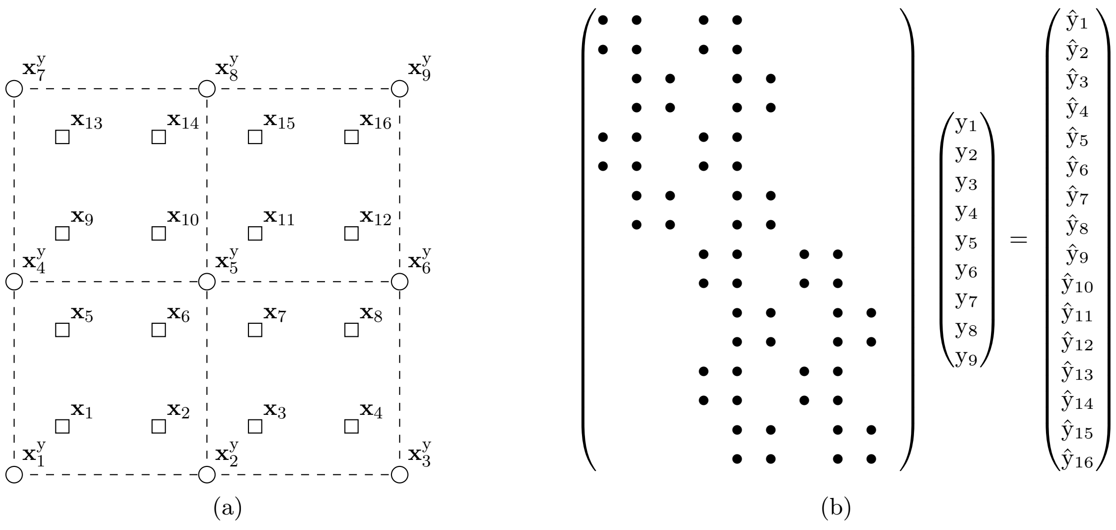
with $i = 1, \dots, \bar{m}$ and $l = 0, \dots, d-1$. For $d = 2$ the third weight is not needed, i.e., $w(\hat{\mathbf{y}}_i)_3^{\gamma} \equiv 1$. This is also visualized in Figure 3.7(a) for $d = 2$, where the weights $w^{\alpha,\beta} := w_1^{\alpha} w_2^{\beta}$ for one image grid point are shown as arrows.

Note that while in the example in Figure 3.6(a) each deformation grid cell contains the same number of image grid points, this is not valid in general: In Figure 3.7(a), the grid spacing is chosen such that the number of image grid points per deformation grid cell varies. This also results in a less regular sparsity pattern of the grid conversion matrix compared to Figure 3.6(b).

3.5.2. Transposed operator

From (3.3) and (3.4), in order to compute distance measure derivatives, the derivative $\frac{\partial P}{\partial \mathbf{y}}$ is required. Using (3.50), differentiating the grid conversion function is rather simple:

$$\frac{\partial P}{\partial \mathbf{y}} = P \in \mathbb{R}^{d\bar{m} \times d\bar{m}^{\mathrm{y}}}. \tag{3.52}$$



x^y_1 x^y_2 x^y_3 x^y_4 x^y_5 x^y_6 x^y_7 x^y_8 x^y_9
x_1 … x_{16}
y_1 … y_9
$\hat{y}_1$ … $\hat{y}_{16}$
(a)
(b)

Figure 3.6.: Grid conversion example for $d = 2$, (a): Two small grids; circles for deformation grid with $m^y = (3, 3)$, squares for image grid with $m = (4, 4)$, (b): Corresponding grid conversion matrix $\tilde{P}$, converting the x-component of the deformation $\mathbf{y}$ from deformation grid the image grid. While each row of $\tilde{P}$ contains exactly four elements, the number of elements in each column varies, requiring a different computation strategy for the right-multiply.

Note that in contrast to the image interpolation derivatives in Section 3.1.1, we compute the derivative $\frac{\partial P}{\partial \mathbf{y}}$ with respect to the *data points* $\mathbf{y}$. Stating (3.50) as

$$\hat{\mathbf{y}} = P(\mathbf{x}) \cdot \mathbf{y},$$

we see that the interpolation matrix $P(\mathbf{x})$ evaluates the data points $\mathbf{y}$ on the image grid $\mathbf{x}$. Analogous to this, the image interpolation can be written as

$$T(\hat{\mathbf{y}}) = Q(\hat{\mathbf{y}}) \cdot \mathbf{T},$$

where $Q(\hat{\mathbf{y}}) \in \mathbb{R}^{\bar{m} \times \bar{m}^{\mathrm{T}}}$ is an interpolation matrix, evaluating the template image data points $\mathbf{T} \in \mathbb{R}^{\bar{m}^{\mathrm{T}}}$ on the deformed image grid $\hat{\mathbf{y}}$. In contrast to the grid conversion, we compute the derivative $\frac{\partial T}{\partial \hat{\mathbf{y}}}$ with respect to the *evaluation points*, resulting in a very different derivative structure.

While the matrix P in (3.52) is the same as in (3.50), for computing (3.3) and (3.4), instead of a left-multiply now a right-multiply

$$\hat{\mathbf{y}}^\top P = (P^\top \hat{\mathbf{y}})^\top$$

is required, which can also be interpreted as a left-multiply with the transposed matrix $P^\top$.

As can be seen from Figure 3.6(b), instead of computing a weighted sum of a fixed number of points as in (3.51), multiplication with $P^\top$ corresponds to the computation

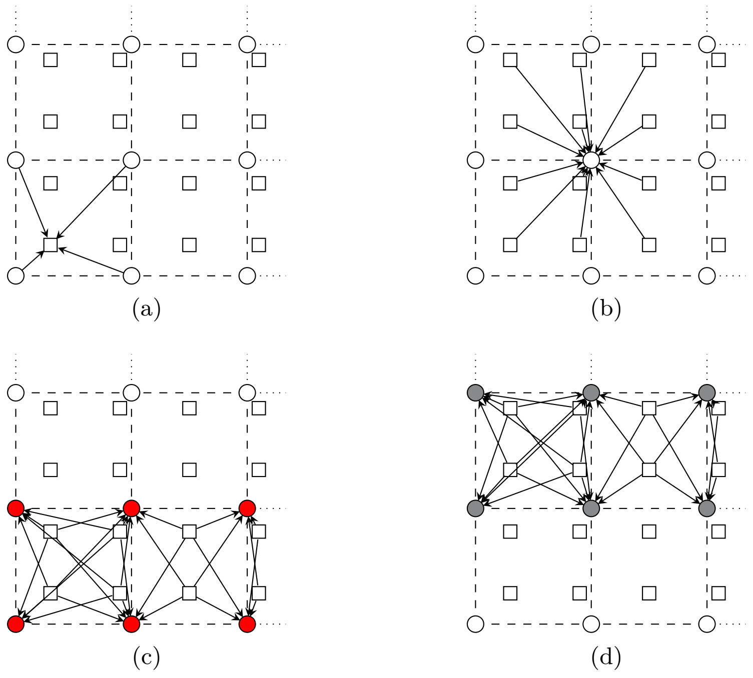


Figure 3.7.: Grid conversion operations for $d = 2$. *Circles* indicate a nodal deformation grid, *squares* indicate a cell-centered image grid, different interpolation weights are visualized by arrows, (a): deformation grid to image grid, exemplarily shown for one image grid point. Each value on the image grid is computed by a weighted sum of the values of its deformation grid neighbors, (b): transposed operation, exemplarily shown for one deformation grid point. Each value on the deformation grid is a weighted sum of all image grid values from neighboring deformation grid cells, (c), (d): transposed operation with red-black scheme. For each deformation grid cell, weighted values on the image grid are accumulated into all surrounding deformation grid points. Weights need to be computed only once per cell. First all odd rows are processed in parallel (red), as shown in (c), then all even rows (black) are processed in parallel as shown in (d). This avoids write conflicts in overlapping deformation grid points during parallel execution.

of a weighted sum of values from all image grid points in the adjacent deformation grid cells, as shown in Figure 3.7(b) for $d = 2$. Depending on the spacing of both grids, the number of elements of that sum can vary for different points and with $h^{\mathrm{y}} \geq h$ often has more than 2^d elements.

The variable number of elements implies that these points need to be identified during the computations before the sum can be computed, which makes the computation more involved. Apart from this, deriving a matrix-free computation that can be parallelized per deformation grid point, as shown in Figure 3.7(b) and similar to (3.51), is straightforward. However, in a matrix-free computation scheme, parts of weights (shown as arrows in Figure 3.7(b)) already used for the previous points need to be recalculated for every deformation grid point. Especially when $h^{\mathrm{y}} \gg h$ and $d = 3$, the number of recalculations is a significant obstacle, as will be discussed further in Section 3.7. Therefore, a red-black scheme was used instead.

Red-black computation scheme. Instead of computing the complete weighted sum for every deformation grid point as in Figure 3.7(b), those elements of the sum are computed which utilize the same weights w^s. These elements contribute to the weighted sums for different deformation grid points surrounding one deformation grid cell, which suggests an evaluation as shown in Figure 3.7(c). In this scheme, for each deformation grid cell, only the contributions of the image grid points in the current deformation grid cell are added to the weighted sums of the surrounding deformation grid points. This way, the weights w^s only need to be computed once.

However, this scheme is not straightforward to parallelize. Contributions of the neighboring deformation grid cells are added to the weighted sums of the same deformation grid points. When parallelizing over the deformation grid cells, this creates simultaneous write accesses, resulting in write conflicts. Therefore, the domain is split in an alternating fashion in the last dimension (y-dimension for $d = 2$ and z-dimension for $d = 3$), separating it into alternating red and black rows for $d = 2$ and xy-slices for $d = 3$. All rows/slices of the same color can now be computed in parallel, first computing all red rows/slices in parallel and then all black rows/slices, as shown in Figure 3.7(c) and Figure 3.7(d). The deformation grid points *within* a row/slice are processed sequentially.

3.5.3. In-place vs. separate grid conversion

In the previous sections, the grid conversion operations are treated as separate steps. As given in (3.1), the distance measure function evaluation can be written as

$$D(\mathbf{y}) = \psi(r(T(P(\mathbf{y})))).$$

Considering the grid conversion separately, this can be split into two steps

$$\begin{aligned} &1. \quad \hat{\mathbf{y}} \leftarrow P\mathbf{y} \\ &2. \quad D \leftarrow \psi(r(T(\hat{\mathbf{y}}))). \end{aligned}$$

First, the deformation is interpolated on the image grid. Second, using the deformation on the image grid, the final function value is computed.

For gradient and Gauss-Newton Hessian computations, additionally a third step is required. The distance measure gradient computation (3.3) can be decomposed into

$$\nabla D(\mathbf{y}) = \left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial T}\frac{\partial T}{\partial P}\frac{\partial P}{\partial \mathbf{y}}\right)^\top .$$

Separating the grid conversion steps, the gradient can be computed by

$$\begin{aligned} &1. \quad \hat{\mathbf{y}} \leftarrow P\mathbf{y} \\ &2. \quad \hat{\mathbf{g}} \leftarrow \frac{\partial \psi}{\partial r}\frac{\partial r}{\partial T}\frac{\partial T}{\partial P}(\hat{\mathbf{y}}) \\ &3. \quad \nabla D^\top \leftarrow P^\top \hat{\mathbf{g}}. \end{aligned}$$

First, again, the deformation is interpolated on the image grid. Then, a "gradient on the image grid" $\hat{\mathbf{g}} \in \mathbb{R}^{d\bar{m}}$ is computed. Finally, by multiplying with the transposed grid conversion matrix $\frac{\partial P}{\partial \mathbf{y}}^\top = P^\top \in \mathbb{R}^{d\bar{m}^{\mathrm{y}} \times \bar{m}}$, the gradient $\nabla D \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}$ is obtained.

The computation of the distance measure Hessian-vector multiplication is similar. Multiplication of the Gauss-Newton Hessian approximation with a vector $\mathbf{p} \in \mathbb{R}^{\bar{m}^{\mathrm{y}}}$ can be written as (3.4)

$$\mathbf{q} = H\mathbf{p} = \frac{\partial P}{\partial \mathbf{y}}^\top \hat{H} \frac{\partial P}{\partial \mathbf{y}} \mathbf{p} = \frac{\partial P}{\partial \mathbf{y}}^\top \frac{\partial T}{\partial P}^\top \frac{\partial r}{\partial T}^\top \frac{\partial^2 \psi}{\partial r^2} \frac{\partial r}{\partial T} \frac{\partial T}{\partial P} \frac{\partial P}{\partial \mathbf{y}} \mathbf{p}, \tag{3.53}$$

which can be split into the three steps

$$\begin{aligned} &1. \quad \hat{\mathbf{p}} \leftarrow \frac{\partial P}{\partial \mathbf{y}} \mathbf{p} = P\mathbf{p} \\ &2. \quad \hat{\mathbf{q}} \leftarrow \hat{H}\hat{\mathbf{p}} = \frac{\partial T}{\partial P}^\top \frac{\partial r}{\partial T}^\top \frac{\partial^2 \psi}{\partial r^2} \frac{\partial r}{\partial T} \frac{\partial T}{\partial P} \hat{\mathbf{p}} \\ &3. \quad \mathbf{q} \leftarrow P^\top \hat{\mathbf{q}}. \end{aligned}$$

The vector $\mathbf{p}$ is first converted to the image grid. The result $\hat{\mathbf{p}} \in \mathbb{R}^{d\bar{m}}$ is then multiplied by $\hat{H}$. Finally the result on the image grid $\hat{\mathbf{q}}$ is multiplied by the transposed grid conversion matrix to obtain the final result $\mathbf{q}$.

Treating the grid conversion as separate steps, however, requires additional memory for temporarily storing $\hat{\mathbf{y}}, \hat{\mathbf{p}} \in \mathbb{R}^{d\bar{m}}$ and $\hat{\mathbf{g}}, \hat{\mathbf{y}} \in \mathbb{R}^{d\bar{m}^{\mathrm{y}}}$. Therefore, it is tempting to substitute the matrix-free grid conversion into the distance measure computations, so that no temporary storage is required.

This in-place grid conversion further increases the amount of recalculations since converted grid points on the image grid are required multiple times for the computation of different result elements. This will be analyzed further in Section 3.7. In some cases it leads to slower overall runtimes. Since, depending on the application scenario, either runtime or memory requirements may be important, both variants will be evaluated in Chapter 5.

3.6. Rigid and affine deformation model

In comparison to the deformable deformation model, for an affine registration model (Section 2.4.2) the grid conversion function $P(\mathbf{y})$ is replaced by a "grid-generating" function $\hat{\mathbf{y}} : \mathbb{R}^{d^2+d} \to \mathbb{R}^{d\bar{m}}$, defined in (2.10), with

$$\hat{\mathbf{y}}(w) = \left(\left[(A\mathbf{x}_1 + b)_1, \dots, (A\mathbf{x}_{\bar{m}} + b)_1 \right], \dots, \left[(A\mathbf{x}_1 + b)_d, \dots, (A\mathbf{x}_{\bar{m}} + b)_d \right] \right)^\top, \tag{3.54}$$

which transforms the new parameters w to a deformed grid. Here, the parameters $w = (a_1, \dots, a_{d^2}, b_1, \dots, b_d) \in \mathbb{R}^{d^2+d}$ consist of the entries from the linear transformation matrix $A \in \mathbb{R}^{d \times d}$ and the translation vector $b \in \mathbb{R}^d$, which results in 12 parameters for $d = 3$, and 6 parameters for $d = 2$. In affine registration, a regularization term is usually not necessary [Mod09, §6] such that the optimization problem (2.25) becomes

$$\min_{w \in \mathbb{R}^{d^2+d}} D(\hat{\mathbf{y}}(w)). \tag{3.55}$$

For a rigid deformation, the matrix A is additionally replaced by a function

$$A(\theta) : \mathbb{R}^{\vartheta} \to \mathbb{R}^{d^2+d}, \tag{3.56}$$

with $\vartheta := 3$ for $d = 3$ and $\vartheta := 1$ for $d = 2$, which maps three rotation angles $\theta \in \mathbb{R}^3$ for $d = 3$ or a single rotation angle $\theta \in \mathbb{R}$ for $d = 2$ to a d-dimensional rotation matrix. With the additional translation $b \in \mathbb{R}^d$ this results in a total of 6 unknowns for $d = 3$ and 3 unknowns for $d = 2$ for the rigid case.

Since the main parts of the distance measure computations remain unchanged, with the definition of $\hat{\mathbf{y}}$ the function value of the distance measures can directly be computed. However, the number of unknowns is now fixed and largely reduced. This requires different approaches for matrix-free methods for the derivatives, which will be described in the following section.

3.6.1. Derivative computations

Replacing the grid conversion function in (3.3) with $\hat{\mathbf{y}}(w)$, we obtain

$$\nabla D(w) = \left(\frac{\partial \psi}{\partial r} \frac{\partial r}{\partial T} \frac{\partial T}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial w} \right)^{\top} \in \mathbb{R}^{(d^2+d) \times 1} \tag{3.57}$$

for the affine distance measure gradient. Apart from the term $\frac{\partial \hat{\mathbf{y}}}{\partial w}$, the computation remains unchanged compared to the deformable case. Especially when treating the grid conversion as a separate step, the matrix-free computations (3.16) for SSD and (3.21) for NGF and can be utilized. Analogously, following (3.4), the Hessian approximation for the affine deformation model can be written as

$$\nabla^2 D(w) \approx H(w) = \frac{\partial \hat{\mathbf{y}}}{\partial w}^{\top} \frac{\partial T}{\partial \hat{\mathbf{y}}}^{\top} \frac{\partial r}{\partial T}^{\top} \frac{\partial^2 \psi}{\partial r^2} \frac{\partial r}{\partial T} \frac{\partial T}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial w} \in \mathbb{R}^{(d^2+d) \times (d^2+d)}. \tag{3.58}$$

Differentiating (3.54), we obtain the Jacobian matrix $\frac{\partial \hat{\mathbf{y}}}{\partial w}$ which has a block diagonal structure. With

$$W := \begin{pmatrix} \mathrm{x}_1 & \dots & \mathrm{x}_{1+(d-1)\bar{m}} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{x}_{\bar{m}} & \dots & \mathrm{x}_{1+d\bar{m}} & 1 \end{pmatrix} \in \mathbb{R}^{\bar{m} \times (d+1)}, \tag{3.59}$$

the Jacobian matrix of $\hat{\mathbf{y}}$ can be written as

$$\frac{\partial \hat{\mathbf{y}}}{\partial w} = \begin{pmatrix} W & & \\ & \ddots & \\ & & W \end{pmatrix} \in \mathbb{R}^{d\bar{m} \times d(d+1)}, \tag{3.60}$$

putting W on the diagonal d times and zeros otherwise. As can be seen, the structure of $\frac{\partial \hat{\mathbf{y}}}{\partial w}$ consists of constant values from the image grid $\mathbf{x}$ (Section 2.4.1), as well as ones. With the definition of this Jacobian, we can now revisit the derivative computations for the different distance measures.

Sum of squared differences. To derive a matrix-free computation for the SSD gradient, the additional step of multiplying with the Jacobian $\frac{\partial \hat{\mathbf{y}}}{\partial w}$, replacing the grid conversion function, will now be integrated into the matrix-free computations. For the SSD gradient (3.16) this results in

$$\left(\nabla D^{\mathrm{SSD}}\right)_{k+ld} = \sum_{i=1}^{\bar{m}} \left(T_i - R_i\right) \frac{\partial T_i}{\partial \hat{\mathrm{y}}_{i+l\bar{m}}} \mathrm{x}_{i+(k-1)\bar{m}}, \tag{3.61}$$

with $k = 1, \dots, d+1$, $l = 0, \dots, d-1$ and, corresponding to the column of ones in (3.59), $\mathrm{x}_{i+d\bar{m}} := 1$, such that $\nabla D^{\mathrm{SSD}} \in \mathbb{R}^{d^2+d}$.

Note the additional sum compared to (3.16), incorporating all residual elements $r_i^{\mathrm{SSD}} = (T_i - R_i)$ for each gradient entry. In a per-element parallel computation as in the deformable deformation model, the full residual has to be computed multiple times. Additionally, parallelization is problematic, since only $d^2 + d$ gradient elements exist, limiting the maximum number of parallel threads. Therefore, we pursue a different computation strategy: Instead of considering all summands for a single gradient element, a vector of single summands

$$\left(\nabla D^{\mathrm{SSD}}\right)^i := \left(\left(\nabla D^{\mathrm{SSD}}\right)^i_1, \dots, \left(\nabla D^{\mathrm{SSD}}\right)^i_{d^2+d}\right) \in \mathbb{R}^{d^2+d}$$

with

$$\left(\nabla D^{\mathrm{SSD}}\right)^i_{k+ld} := \left(T_i - R_i\right) \frac{\partial T_i}{\partial \hat{\mathrm{y}}_{i+l\bar{m}}} \mathrm{x}_{i+(k-1)\bar{m}} \tag{3.62}$$

is computed for $k = 1, \dots, d+1$, $l = 0, \dots, d-1$, i.e., computing only one summand but for all gradient elements at once. This can then be parallelized for $i = 1, \dots, \bar{m}$. Finally, a reduction

$$\nabla D^{\mathrm{SSD}} = 2\bar{h} \sum_{i=1}^{\bar{m}} \left(\nabla D^{\mathrm{SSD}}\right)^i \tag{3.63}$$

is performed. In order to perform this efficiently, the reduction needs to be performed without allocation of auxiliary memory for each summand, i.e., using specialized reduction variables in OpenMP, see Section 3.8.1.

For computing the Gauss-Newton Hessian approximation, this approach can be used as well. Since $H^{\mathrm{SSD}} \in \mathbb{R}^{(d^2+d)\times(d^2+d)}$ has only few elements in comparison to the deformable case, the Hessian approximation can be explicitly and efficiently stored instead of computing a matrix-free Hessian-vector multiplication. The number of stored elements can be further reduced by exploiting the symmetry of the Hessian.

Utilizing (3.17) and $\frac{\partial r^{\mathrm{SSD}}}{\partial T} = I$, computations for the Hessian approximation (3.58) can be performed in a similar way as the gradient (3.63). The rows of the matrix

$$dw^{\mathrm{SSD}} := \frac{\partial T}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial w} \in \mathbb{R}^{\bar{m}\times(d^2+d)}$$

are

$$\left(dw^{\mathrm{SSD}}\right)^i := \left(\left(dw^{\mathrm{SSD}}\right)^i_1, \dots, \left(dw^{\mathrm{SSD}}\right)^i_{d^2+d}\right) \in \mathbb{R}^{1\times(d^2+d)}, \tag{3.64}$$

where

$$\left(dw^{\mathrm{SSD}}\right)^i_{k+ld} := \frac{\partial T_i}{\partial \hat{\mathrm{y}}_{i+l\bar{m}}} \mathrm{x}_{i+(k-1)\bar{m}} \in \mathbb{R}. \tag{3.65}$$

In order to obtain $\left(dw^{\mathrm{SSD}}\right)^\top dw^{\mathrm{SSD}}$ as in (3.58), a reduction can be performed over the *outer product* of the rows (3.64), so that

$$H^{\mathrm{SSD}}(w) = 2\bar{h} \sum_{i=1}^{\bar{m}} \left(\left(dw^{\mathrm{SSD}}\right)^i \right)^\top \left(dw^{\mathrm{SSD}}\right)^i \in \mathbb{R}^{(d^2+d)\times(d^2+d)}. \tag{3.66}$$

Equations (3.65) and (3.62) are very similar, except for the residual term $(T_i - R_i)$. Therefore, the gradient and the Hessian approximation can be computed together efficiently.

Normalized gradient fields. As in (3.61), the affine transformation function can be integrated into the matrix-free NGF gradient computations (3.37), resulting in

$$\left(\nabla D^{\mathrm{NGF}}\right)_{k+l\bar{d}} = -2\bar{h} \sum_{i=1}^{\bar{m}} \left(\sum_{j\in\mathcal{K}} r^{\mathrm{NGF}}_{i_j} \hat{\rho}_{i_j}(-j) \right) \frac{\partial T_i}{\partial \hat{\mathrm{y}}_{i+l\bar{m}}} \mathrm{x}_{i+(k-1)\bar{m}}, \tag{3.67}$$

with $k = 1, \dots, d+1$, $l = 0, \dots, d-1$ and $\mathrm{x}_{i+d\bar{m}} := 1$, corresponding to the column of ones in (3.59). Similar to the SSD computations in (3.63) and (3.62), this formulation requires to compute the same values multiple times to obtain the full gradient and per-element parallelization can only be computed with $d^2 + d$ parallel tasks. Therefore, the computation is again reformulated and split into single summands $\left(\nabla D^{\mathrm{NGF}}\right)^i \in \mathbb{R}^{d^2+d}$ for parallelization, which results in

$$\left(\nabla D^{\mathrm{NGF}}\right)^i_{k+l\bar{d}} := r^{\mathrm{NGF}}_i \sum_{j\in\mathcal{K}} \hat{\rho}_{i_j}(-j) \frac{\partial T_i}{\partial \hat{\mathrm{y}}_{i_j+l\bar{m}}} \mathrm{x}_{i_j+(k-1)\bar{m}} \tag{3.68}$$

for $k = 1, \dots, d+1$, $l = 0, \dots, d-1$ and the vector of single summands

$$\left(\nabla D^{\mathrm{NGF}}\right)^i := \left(\left(\nabla D^{\mathrm{NGF}}\right)^i_1, \dots, \left(\nabla D^{\mathrm{NGF}}\right)^i_{d^2+d} \right) \in \mathbb{R}^{d^2+d}.$$

The full affine gradient $\nabla D^{\mathrm{NGF}} \in \mathbb{R}^{d^2+d}$ can then be written as

$$\nabla D^{\mathrm{NGF}} = -2\bar{h} \sum_{i=1}^{\bar{m}} \left(\nabla D^{\mathrm{NGF}}\right)^i, \tag{3.69}$$

which can now easily be parallelized with up to $\bar{m}$ parallel tasks.

Similar to the SSD, the small number of affine parameters can be exploited to simplify computation of the Hessian approximation. Instead of deriving a matrix-free computation of the matrix product $\frac{\partial r^{\mathrm{NGF}}}{\partial T}^\top \frac{\partial r^{\mathrm{NGF}}}{\partial T}$, as in (3.44), again a row of the matrix $dw^{\mathrm{NGF}} := \frac{\partial r^{\mathrm{NGF}}}{\partial T} \frac{\partial T}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial w} \in \mathbb{R}^{\bar{m}\times(d^2+d)}$ is computed via

$$\left(dw^{\mathrm{NGF}}\right)^i := \left(\left(dw^{\mathrm{NGF}}\right)^i_1, \dots, \left(dw^{\mathrm{NGF}}\right)^i_{d^2+d} \right) \in \mathbb{R}^{1\times(d^2+d)} \tag{3.70}$$

with

$$\left(dw^{\mathrm{NGF}}\right)^{i}_{k+l\bar{d}} = \sum_{j\in\mathcal{K}} \hat{\rho}_{i_j}(-j)\frac{\partial T_i}{\partial \hat{\mathrm{y}}_{i_j+l\bar{m}}}\mathrm{x}_{i_j+(k-1)\bar{m}}. \tag{3.71}$$

The Hessian approximation can then be computed as the reduction of the outer product of the rows (3.70), which only have d^2+d elements each:

$$H^{\mathrm{NGF}}(w) = 2\bar{h}\sum_{i=1}^{\bar{m}} \left(\left(dw^{\mathrm{NGF}}\right)^{i}\right)^{\top} \left(dw^{\mathrm{NGF}}\right)^{i} \in \mathbb{R}^{(d^2+d)\times(d^2+d)}. \tag{3.72}$$

Again, similar to the SSD affine computations, because of the similarity between (3.67) and (3.71), gradient and Hessian approximation can be efficiently computed together.

Rigid deformation model. For a rigid transformation model as in (3.56), an additional function is concatenated to the computations, resulting in another Jacobian matrix in the derivative calculations. However, since $A_\vartheta : \mathbb{R}^\vartheta \to \mathbb{R}^{d^2+d}$ is a function with a fixed and comparably small number of parameters, its derivatives can be computed efficiently in a separate step and no specialized matrix-free computations are required.

3.7. Algorithm analysis

In the previous sections of this chapter, we presented matrix-free formulations for the objective function derivatives. In this section, we first analyze these methods from a theoretical point of view, before performing practical evaluations in Chapter 5. Emphasis is placed on two main characteristics:

- runtime, and
- memory consumption.

In comparison with a matrix-based approach, we discuss differences between both concepts in terms of in-place recalculations versus precomputations that are stored to memory. Based on this, we analyze possible trade-offs between memory usage and runtime.

The matrix-based approach heavily relies on the multiplication of sparse matrices. The actual computational cost for sparse matrix multiplications, however, is highly dependent on the chosen sparse matrix format and the specific implementation and therefore hard to quantify from theory. However, besides matrix multiplications, meaningful comparisons between matrix-based and matrix-free methods can be made by analyzing the cost of computing the entries of the matrices involved and storing them to memory.

For the matrix-based approach, the relation between computing and storing matrix elements is simple: Every non-zero matrix element is computed exactly once, the elements are composed to a sparse matrix, and this matrix is written to memory for later use, i.e., all matrix elements are *precomputed.* This gives a one-to-one ratio of elements computed and elements stored to memory.

For the matrix-free approach, this is different. In a fully matrix-free computation, no intermediate results are written to memory and all computations are performed on-the-fly only using initial data, such as the images and the current deformation. Some intermediate results might need to be computed multiple times, since they occur in the computation of multiple result elements. These operations are denoted as *recalculations*. Since storing a value to memory typically consumes much more time than performing a floating point operation, an overall decrease of runtime is still possible.

In the matrix-free approach, the ratio between element computations and elements stored to memory can additionally be influenced by selectively precomputing crucial elements, such as performing the grid conversion in separate steps (Section 3.5.3), or precomputing the deformed template image $T(\hat{\mathbf{y}})$. In some scenarios this can yield an additional speedup of the computations at the cost of a higher memory usage, which will also be discussed in detail below.

3.7.1. Sum of squared differences

For the SSD distance measure, matrix-free computations for the gradient and Hessian-vector multiplication were given in Section 3.2, which will now be analyzed.

Gradient. With the general definition of the distance measure gradient in (3.3), as

$$\nabla D(\mathbf{y}) = \left(\frac{\partial \psi}{\partial r} \frac{\partial r}{\partial T} \frac{\partial T}{\partial P} \frac{\partial P}{\partial \mathbf{y}} \right)^\top \in \mathbb{R}^{d\bar{m}^{\mathrm{y}} \times 1},$$

an upper bound for the number of matrix coefficient computations and memory store operations for the gradient can be determined. The vector $\frac{\partial \psi}{\partial r} \in \mathbb{R}^{\bar{m}}$ requires the computation of $\bar{m}$ residual elements. For the SSD, the matrix $\frac{\partial r^{\mathrm{SSD}}}{\partial T}$ is the identity and thus can be neglected. The matrix $\frac{\partial T}{\partial P} \in \mathbb{R}^{\bar{m} \times d\bar{m}}$ is composed of d blocks of diagonal matrices (Figure 3.1), and contains $d\bar{m}$ non-zero elements. Finally, the grid conversion matrix $\frac{\partial P}{\partial \mathbf{y}} = P \in \mathbb{R}^{d\bar{m} \times d\bar{m}^{\mathrm{y}}}$ contains at most 2^d elements in each row. Since the matrix P consists of d identical blocks $\hat{P} \in \mathbb{R}^{\bar{m} \times \bar{m}^{\mathrm{y}}}$ (Section 3.5.1) we only count one block with at most $2^d \bar{m}$ coefficients here. Additionally, the matrix elements of P only depend on the size of the image and deformation grids, but not on the image data or deformation. Therefore they only need to be computed and stored once for each resolution.

In total this gives $(1 + d + 2^d)\bar{m}$ non-zero matrix coefficient calculations, which amounts to $12\bar{m}$ for $d = 3$ and $7\bar{m}$ for $d = 2$. In a matrix-based computation, each of these coefficients is computed and then stored to memory for later use, resulting in memory requirements of the same amount.

A matrix-free computation for the SSD gradient was given in (3.16). It can be seen that each computation of a single gradient element only requires one element of each matrix. Since these are not required for the computation of other gradient elements, for the computation of the full gradient also $(1 + d + 2^d)\bar{m}$ matrix element computations are required, as summarized in Table 3.2. While the number of coefficient computations is identical to the matrix-based case, no intermediate results are stored and a large benefit in runtime for the matrix-free computations can be expected.

Hessian approximation. In addition to the matrix-based gradient computation, for a matrix-based SSD Hessian-vector multiplication no additional coefficient computations are needed. As can be seen in (3.17), since the Gauss-Newton Hessian multiplication is composed from first-order derivatives, it consists of the same matrices that have already been computed for the gradient computation. Assuming that the Hessian approximation is computed anew from these matrices every time, no additional memory is required. If the Hessian approximation is explicitly stored, additional memory for $d^2\bar{m}$ coefficients is required, due to the structure of the image derivative matrices in (3.17), shown in Figure 3.1.

The matrix-free Hessian-vector multiplication is given in (3.18). As already discussed at the end of Section 3.2.2, a naive computation involves a d-fold overhead for the computation of the image derivatives. However, the d directional derivatives of T_i are usually computed together. It is therefore preferable to also compute the d result elements $\hat{\mathrm{q}}_{l\bar{m}+i}$, $l = 0, \ldots, d-1$ together, which eliminates the overhead and requires $d\bar{m}$ coefficients for the image derivatives and $2^d\bar{m}$ coefficients for the grid conversion, resulting in a total of $(d + 2^d)\bar{m}$ matrix element computations, see also Table 3.2. Again, no intermediate storage of matrix coefficients is required.

Summary. The matrix-free and matrix-based SSD gradient computation and Gauss-Newton Hessian-vector multiplication require the same amount of coefficient calculations. Therefore, the matrix-free version is potentially much faster while requiring less memory. However, these benefits might be slightly reduced when computing gradient and Hessian at the same time, since the coefficients stored for the matrix-based gradient can be reused for the matrix-based Hessian, while using the matrix-free approach these have to be recalculated.

Since the amount of coefficient computations is already minimal for gradient and Gauss-Newton Hessian-vector multiplication, for the SSD no computational overhead can be eliminated by precomputing certain often used values such as the deformation on the image grid $\hat{\mathbf{y}}$ or the transformed template $T(\hat{\mathbf{y}})$. For the NGF, discussed in the following section, this is different.

Summarizing the memory usage as shown in Table 3.2, for the matrix-free approach the memory requirements are in the order of $\mathcal{O}(\bar{m})$. The matrix-free approach requires only constant auxiliary space and thus reduces the memory requirements to $\mathcal{O}(1)$.

3.7.2. Normalized gradient fields

Matrix-free computations for the NGF gradient were given in Section 3.3. The main difference in comparison with the SSD is the more complicated structure of the Jacobian matrix $\frac{\partial r^{\mathrm{NGF}}}{\partial T}$, see Figure 3.4 and Figure 3.5. In the following, we will analyze the impact of this on the amount of computations and memory store operations.

Method	matrix-based	matrix-free gradient		matrix-free Hessian	
	(calc./stores)	calc.	stores	calc.	stores
$\frac{\partial\psi}{\partial r}$	$\bar{m}$	$\bar{m}$	0	0	0
$\frac{\partial T}{\partial P}$	$d\bar{m}$	$d\bar{m}$	0	$d\bar{m}$	0
$\frac{\partial P}{\partial \mathbf{y}}$	$2^d\bar{m}$	$2^d\bar{m}$	0	$2^d\bar{m}$	0
total $d = 2$	$7\bar{m}$	$7\bar{m}$	0	$6\bar{m}$	0
total $d = 3$	$12\bar{m}$	$12\bar{m}$	0	$11\bar{m}$	0

Table 3.2.: SSD derivative matrix element calculations and memory requirements. For the matrix-based approach, all computed values are always stored to memory, while for the matrix-free approach everything is computed on-the-fly without the need for intermediate storage. For the SSD matrix-free gradient and Gauss-Newton Hessian-vector multiplication no overhead in comparison to the matrix-based approach in terms of recalculations is required. A significant speedup can thus be expected while lowering the memory requirements for intermediate results from $\mathcal{O}(\bar{m})$ to $\mathcal{O}(1)$.

Gradient. As shown in (3.3) and already discussed in Section 3.7.1, four Jacobian matrices are required for a matrix-based distance measure gradient computation. Since everything except the matrix $\frac{\partial r^{\text{NGF}}}{\partial T}$ remains unchanged, from the previous section we know that for the other four Jacobian matrices $(1 + d + 2^d)\bar{m}$ matrix coefficient computations are needed and that the matrix-based approach requires the same amount of memory store operations. The matrix $\frac{\partial r^{\text{NGF}}}{\partial T} \in \mathbb{R}^{\bar{m}\times\bar{m}}$ itself exhibits a structure with $2d + 1$ diagonals, as shown in Section 3.3.

In the matrix-based case this results in an upper bound of $(2d+1)\bar{m}$ coefficient calculations and memory stores, which gives a total of $(2+3d+2^d)\bar{m}$ coefficient calculations. The same amount of memory is required for intermediate results, with an additional $\bar{m}$ coefficients for storing the deformed template image $T(\hat{\mathbf{y}})$, as discussed later on. In total, this amounts to $20\bar{m}$ for $d = 3$ and $13\bar{m}$ coefficients for $d = 2$, see also Table 3.3.

For the matrix-free case, the gradient computations are given in (3.37). For every point i, the values of $r^{\text{NGF}}_{i_k}$ and $\hat{\rho}_{i_k}$ are required at the indices $k \in \mathcal{K}$, i.e., $2d + 1$ locations. If, as suggested before for the SSD Gauss-Newton Hessian computations, the directional derivatives for $l = 0, \ldots, d-1$ are computed together, this also results in $(2d+1)\bar{m}$ matrix element calculations and thus no recalculations regarding the matrix $\frac{\partial r^{\text{NGF}}}{\partial T}$. However, the values $r^{\text{NGF}}_{i_k}$ in (3.37) also need to be computed $(2d + 1)\bar{m}$ times instead of $\bar{m}$ times, leading to recalculations.

Further recalculations become visible when examining the computation of the individual matrix elements $\hat{\rho}_{i_k}$ more closely. As defined in (3.34) and (3.32), the computation of $\hat{\rho}_{i_k}$ for $k \in \mathcal{K}$ requires the computation of finite difference gradient approximations at $2d + 1$ points. Each of these finite difference gradient approximations requires an evaluation of the image at all neighboring points, see (2.19), such that, with duplicates removed, $2d^2+2d+1$ neighborhood evaluations are required for each final gradient element in (3.37), resulting in at most $(2d^2+2d+1)\bar{m}$ neighborhood evaluations overall. Given that there are only $\bar{m}$ points in the image grid, this amounts to an additional recalculation of $(2d^2+2d)\bar{m}$ image values. While for the reference image simply existing values need to be loaded, for the deformed template image $T(\hat{\mathbf{y}})$ this involves expensive image interpolation. Therefore, as a trade-off between recalculations and memory-usage, the deformed template image $T(\hat{\mathbf{y}})$ can be precomputed. In this case the recalculations will vanish, but instead $\bar{m}$ image values need to be stored to memory.

A similar trade-off can be performed for the deformation on the image grid $\hat{\mathbf{y}}$. For evaluating the deformed template image as discussed above, $2d^2 + 2d + 1$ evaluations of the deformation $\hat{\mathbf{y}}$ are required. Given the matrix-based case, which needed $2^d\bar{m}$ coefficient evaluations for the grid conversion matrix, i.e., 2^d coefficient computations for the interpolation of a single point, this results in a total of $2^d(2d^2+2d+1)\bar{m}$ coefficient computations. In contrast, if the grid is precomputed, an additional amount of memory for the $d\bar{m}$ values of $\hat{\mathbf{y}}$ is required, but the computational overhead is largely reduced. However, instead of $2^d\bar{m}$ coefficient computations for the grid conversion, $2 \cdot 2^d\bar{m}$ computations are still needed, since the forward and the transposed grid conversion operator must be computed in two separate steps as described in Section 3.5.3. Furthermore, the resulting gradient on the image grid $\frac{\partial D}{\partial P}$ needs to be stored before the second grid conversion step. In total, computing the grid conversion in separate steps and precomputing the deformed template image $T(\hat{\mathbf{y}})$ requires to store $(2d + 1)\bar{m}$ coefficients to memory, but reduces the number of calculations by a factor of 7.11 for $d = 3$ and a factor of 3.67 for $d = 2$, as can be seen in the rightmost column of Table 3.3.

While there additionally exist recalculations for r_i^{NGF} in (3.37) as discussed above, it is not beneficial to perform precomputations here, since most parts for computing r_i^{NGF} are also required for $\hat{\rho}_i$ in $\frac{\partial r^{\mathrm{NGF}}}{\partial T}$, as can be seen when comparing (3.32) and (3.19). Thus, r_i^{NGF} can be computed as a by-product of the calculations of $\frac{\partial r^{\mathrm{NGF}}}{\partial T}$ at little additional cost such that a precomputation of r^{NGF} would not result in any benefit.

The discussed alternatives, their computational effort and memory stores in comparison with the matrix-based approach are summarized in Table 3.3.

Hessian approximation. For the matrix-based NGF Gauss-Newton Hessian approximation, the same observations are valid as in the SSD case. For the NGF, the matrix-based Gauss-Newton approximation of the Hessian (3.4) is

$$H^{\mathrm{NGF}} = 2\bar{h}\frac{\partial P}{\partial \mathbf{y}}^\top \frac{\partial T}{\partial P}^\top \frac{\partial r^{\mathrm{NGF}}}{\partial T}^\top \frac{\partial r^{\mathrm{NGF}}}{\partial T} \frac{\partial T}{\partial P} \frac{\partial P}{\partial \mathbf{y}}.$$

Since H^{NGF} is created from first-order derivatives, a matrix-based implementation requires no additional matrix coefficients to be computed or stored, compared to evaluation of the gradient. If the Hessian approximation as in (3.38) is explicitly stored, since the matrix $\frac{\partial r^{\mathrm{NGF}}}{\partial T}^\top \frac{\partial r^{\mathrm{NGF}}}{\partial T}$ contains at most $|\mathcal{N}| = 2d^2 + 2d + 1$ non-zero elements in each column (cf. Section 3.3.2), additional memory for $(2d^2 + 2d + 1)d^2\bar{m}$ coefficients is required, see also Figure 3.5.

The matrix-free Hessian-vector multiplication for the NGF is given in (3.45). As shown in Figure 3.5, the main computational effort results from the matrix-matrix multiplication

Method	matrix-based	matrix-free gradient		matrix-free gradient with precomputations	
	(calc./stores)	calc.	stores	calc.	stores
$\frac{\partial\psi}{\partial r^{\mathrm{NGF}}}$	$\bar{m}$	$(2d+1)\bar{m}$	0	$(2d+1)\bar{m}$	0
$\frac{\partial r^{\mathrm{NGF}}}{\partial T}$	$(2d+1)\bar{m}$	$(2d+1)\bar{m}$	0	$(2d+1)\bar{m}$	0
$\frac{\partial T}{\partial P}$	$d\bar{m}$	$d\bar{m}$	0	$d\bar{m}$	0
$\frac{\partial P}{\partial \mathbf{y}}, \hat{\mathbf{y}}$	$2^d\bar{m}$	$2^d(2d^2+2d+1)\bar{m}$	0	$2\cdot 2^d\bar{m}$	$2d\bar{m}$
$T(\hat{\mathbf{y}})$	$\bar{m}$	$(2d^2+2d+1)\bar{m}$	0	$\bar{m}$	$\bar{m}$
total $d=2$	$13\bar{m}$	$77\bar{m}$	0	$21\bar{m}$	$5\bar{m}$
total $d=3$	$20\bar{m}$	$242\bar{m}$	0	$34\bar{m}$	$7\bar{m}$

Table 3.3.: NGF derivative matrix element calculations and memory requirements for the matrix-based approach and matrix-free gradient. For the matrix-based approach, all computed values are always stored to memory, while for the matrix-free case, two approaches are shown: Third column: A fully matrix-free version, where everything is computed on the fly and the memory requirements are reduced from $\mathcal{O}(\bar{m})$ to $\mathcal{O}(1)$. Fourth column: Specific parts are selectively precomputed. By precomputing, the number of recalculations can be reduced at the cost of moderate extra memory requirements for intermediate results.

$\frac{\partial r^{\mathrm{NGF}}}{\partial T}^{\top}\frac{\partial r^{\mathrm{NGF}}}{\partial T}$. As discussed in Section 3.3.2, each column of the resulting matrix has at most $|\mathcal{N}| = 2d^2+2d+1$ non-zero entries. To compute all non-zero elements in one column, a maximum of $|\mathcal{M}|^2 = (2d+1)^2$ different elements from $\frac{\partial r^{\mathrm{NGF}}}{\partial T}$, i.e., coefficients $\hat{\rho}_i$ in (3.44) is required (Table 3.1). If again the result elements for $l = 0,\dots,d-1$ in (3.45) are computed simultaneously, this results in $(2d+1)^2\bar{m}$ matrix element computations. Additionally, each non-zero element is multiplied by image derivatives $\frac{\partial T_i}{\partial P_j}$, which are therefore required for $2d^2+2d+1$ points per column, each point with d directional derivatives, resulting in $(2d^2+2d+1)d\bar{m}$ coefficient computations for $\frac{\partial T}{\partial P}$ (Table 3.4).

Observations similar to the gradient can be made for recalculations of the deformed template $T(\hat{\mathbf{y}})$ and the deformation on the image grid $\hat{\mathbf{y}}$. Identical to the gradient, the deformed template image $T(\hat{\mathbf{y}})$ needs to be evaluated at $(2d^2+2d+1)$ points for each result element, which results in $(2d^2+2d+1)\bar{m}$ image interpolation calculations. This overhead can be eliminated by precomputing $T(\hat{\mathbf{y}})$, which requires to store $\bar{m}$ values to memory. However, as described above, the image derivatives are also needed at these locations. Since the interpolated image value and the corresponding derivatives can efficiently be computed together, the benefit from precomputing $T(\hat{\mathbf{y}})$ for the Hessian is therefore lower than for the gradient since the derivatives need to be computed anyway, see also Section 5.2.

Similar to the gradient, the deformation on the image grid $\hat{\mathbf{y}}$ is also required at $(2d^2+2d+1)$ points for each result element, thus again resulting in a total of $2^d(2d^2+2d+1)\bar{m}$ interpolation coefficient computations. When precomputing $\hat{\mathbf{y}}$, besides $2\cdot 2^d\bar{m}$ coefficient

Method	matrix-free Hessian		matrix-free Hessian with precomputations	
	calc.	stores	calc.	stores
$\frac{\partial r^{\text{NGF}}}{\partial T}$	$(2d+1)^2\bar{m}$	0	$(2d+1)^2\bar{m}$	0
$\frac{\partial T}{\partial P}$	$(2d^2+2d+1)d\bar{m}$	0	$(2d^2+2d+1)d\bar{m}$	0
$\frac{\partial P}{\partial \mathbf{y}}, \hat{\mathbf{y}}, \hat{\mathbf{p}}, \hat{\mathbf{q}}$	$2^d(2d^2+2d+1)\bar{m}$	0	$2 \cdot 2^d\bar{m}$	$3d\bar{m}$
$T(\hat{\mathbf{y}})$	$(2d^2+2d+1)\bar{m}$	0	$\bar{m}$	$\bar{m}$
total $d=2$	$116\bar{m}$	0	$60\bar{m}$	$7\bar{m}$
total $d=3$	$349\bar{m}$	0	$141\bar{m}$	$10\bar{m}$

Table 3.4.: NGF derivative matrix element calculations and memory requirements for the matrix-free Gauss-Newton Hessian-vector multiplication. Similar to the gradient computation in Table 3.3, two versions are shown. In the second column no intermediate storage is needed, which comes at the cost of several recalculations. In the third column, some values are selectively precomputed, reducing the number of precomputations at the cost of additional storage requirements.

computations for the separate grid conversion steps, $d\bar{m}$ values need to be stored to memory. As shown in Figure 3.5 for the Hessian-vector multiplication, the vector $\hat{\mathbf{p}}$ and the result $\hat{H}^{\text{NGF}}\hat{\mathbf{p}} = \hat{\mathbf{q}}$ need to be stored as well, such that further $2d\bar{m}$ values need to be stored to memory (Table 3.4). While requiring total additional memory for $(3d+1)\bar{m}$ values, the precomputations result in a reduction of computations by a factor of 2.48 for $d=3$ and a factor of 1.93 for $d=2$.

Summary. In comparison to the SSD, the matrix-free computation of the NGF gradient and Hessian-vector multiplication requires an additional overhead of up to 30 times more matrix-element calculations. While the matrix-based approach requires more memory than the SSD case for additionally storing $\frac{\partial r^{\text{NGF}}}{\partial T}$, the fully matrix-free approach still does not require memory for intermediate results and reduces the memory requirements from $\mathcal{O}(\bar{m})$ to $\mathcal{O}(1)$.

If desired, the number of recalculations can be reduced by precomputing the deformed template image and the deformation on the image grid (Table 3.3 and Table 3.4). At the cost of a moderate memory use, this can potentially improve the overall runtime and is evaluated in Section 5.2.

3.7.3. Regularizer

As described in Section 3.4, the matrix-based curvature regularizer can be computed by using a single matrix operator A, containing weights from the finite difference stencil derived from (2.22). Since it operates directly on the deformation grid $\mathbf{y}$, no grid conversion is involved.

The matrix $A \in \mathbb{R}^{d\bar{m}^{\mathrm{y}} \times d\bar{m}^{\mathrm{y}}}$ exhibits a block-diagonal structure of d identical blocks, similar to the matrix P as discussed before. Following (2.22), it can be seen that for the computation of the Laplacian at each point, the point itself and all neighboring values are required, which results in $2d+1$ elements per row in A. Assuming that the d identical blocks are only stored once, this results in $(2d+1)\bar{m}^{\mathrm{y}}$ matrix element calculations and memory stores.

For the matrix-based curvature gradient as given in (3.46) as

$$\nabla S(\mathbf{y}) = 2\bar{h}^{\mathrm{y}} A^\top A \mathbf{u},$$

the computation of $A^\top A$ is required. As shown in (3.48), this is equivalent to applying the discretized Laplacian operator (2.22) twice, which can also directly be used for computing a Hessian-vector multiplication as shown in (3.49).

By recursively substituting the discretized Laplacian operator in (2.22), it can be seen that for the computation all second-order neighbors of each point are needed. Similar to the NGF finite difference computations in the previous section, this results in an upper bound of $2d^2 + 2d + 1$ diagonals in $A^\top A$ and thus $(2d^2 + 2d + 1)\bar{m}^{\mathrm{y}}$ matrix element calculations and stores for the matrix-based approach.

Since the matrix elements result from the finite difference stencil (2.22), they are highly redundant and can very easily be computed on-the-fly in a matrix-free computation.

When computing a fully matrix-free version of the curvature regularizer as in (3.48), as discussed above, each evaluation of the outer Laplacian requires $2d+1$ evaluations of the inner Laplacian function, which results in a total of $(2d+1)\bar{m}^{\mathrm{y}}$ instead of $\bar{m}^{\mathrm{y}}$ inner Laplacian evaluations. For the matrix-based case, this is equivalent to first computing the matrix $A^\top A$ and then applying it to $\mathbf{u}$. Alternatively, A and $A^\top$ can also be applied subsequently. However, in the matrix-free case this requires an intermediate storage of size $d\bar{m}^{\mathrm{y}}$ for the precomputation of $A\mathbf{u}$, see Table 3.5, reducing the number of element calculations by a factor of 1.79 for $d = 3$ and 1.30 for $d = 2$.

3.7.4. Rigid and affine deformation model

In the affine deformation model, while the components of the distance measure derivative computations are largely identical, the additional function $\hat{\mathbf{y}} : \mathbb{R}^{d^2+d} \to \mathbb{R}^{d\bar{m}}$ requires a different computation scheme, as discussed in Section 3.6. In this scheme, an increment for all $d^2 + d$ gradient elements is computed in each parallel task and the final gradient is obtained by a sum of all these increment vectors. This is in contrast to the deformable case, where a complete, single gradient element is computed in each parallel task.

Sum of squared differences. A matrix-based computation for the affine distance measure gradient is given in (3.57) as

$$\nabla D(w) = \left(\frac{\partial \psi}{\partial r} \frac{\partial r}{\partial T} \frac{\partial T}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial w} \right)^\top \in \mathbb{R}^{(d^2+d)\times 1}.$$

Method	matrix-based	matrix-free		matrix-free with precomputations	
	(calc./stores)	calc.	stores	calc.	stores
$A^\top A\mathbf{u}$	$(2d^2+2d+1)\bar{m}^{\mathrm{y}}*$	$(2d^2+2d+1)\bar{m}^{\mathrm{y}}$	0	$2(2d+1)\bar{m}^{\mathrm{y}}$	$d\bar{m}^{\mathrm{y}}$
total $d=2$	$13\bar{m}^{\mathrm{y}}*$	$13\bar{m}^{\mathrm{y}}$	0	$10\bar{m}^{\mathrm{y}}$	$2\bar{m}^{\mathrm{y}}$
total $d=3$	$25\bar{m}^{\mathrm{y}}*$	$25\bar{m}^{\mathrm{y}}$	0	$14\bar{m}^{\mathrm{y}}$	$3\bar{m}^{\mathrm{y}}$

Table 3.5.: Curvature regularizer derivative matrix element calculations and memory requirements for the matrix-based and matrix-free approach. The highly redundant coefficients can be efficiently computed on-the-fly for the matrix-free approach. By storing the result of $A\mathbf{u}$, the number of calculations can be reduced, at the cost of additional intermediate storage.
*When computing $A^\top(A\mathbf{y})$ instead of $(A^\top A)\mathbf{y}$, only $(2d+1)\bar{m}^{\mathrm{y}}$ calculations and stores are needed.

As before, $\frac{\partial\psi}{\partial r}$ requires the computation of $\bar{m}$ elements, $\frac{\partial r^{\mathrm{SSD}}}{\partial T}$ is the identity and $\frac{\partial T}{\partial \hat{\mathbf{y}}}$ requires $d\bar{m}$ matrix-element computations. Additionally, as shown in (3.60), the matrix $\frac{\partial \hat{\mathbf{y}}}{\partial w} \in \mathbb{R}^{d\bar{m}\times d(d+1)}$ has block-diagonal structure with the dense matrix $W \in \mathbb{R}^{\bar{m}\times(d+1)}$ d-times on the diagonal (3.59). As the last column of W only consists of ones it can be neglected, resulting in additional $d\bar{m}$ element calculations.

As can be seen from (3.62) and (3.63), for the computation of a single increment vector of the matrix-free SSD distance measure $\left(\nabla D^{\mathrm{SSD}}\right)^i$ the following factors are needed: the residual $(T_i - R_i)$, all directional image derivatives $\left(\frac{\partial T_i}{\partial \hat{y}_i}, \dots, \frac{\partial T_i}{\partial \hat{y}_{i+(d-1)\bar{m}}}\right)$ and the image grid coordinates at point $\mathbf{x}_i$. Since for $\left(\nabla D^{\mathrm{SSD}}\right)^i$ they are all only needed at the current index i, no recalculations are involved for computing the final gradient (Table 3.6).

Since the affine Hessian can directly be computed from already-calculated items for the gradient, no additional calculations are needed for both approaches.

Normalized gradient fields. For the NGF, similar to the deformable case, there are recalculations involved when computing the matrix-free affine derivatives. Compared to the deformable case, however, the recalculations are now required for different components, as will be described in the following.

First, we will again analyze the matrix-based case. In comparison to the SSD, three of the four matrices in (3.57) remain unchanged. Furthermore, identical to the deformable case $\frac{\partial r^{\mathrm{NGF}}}{\partial T} \in \mathbb{R}^{\bar{m}\times\bar{m}}$ is no longer the identity, but exhibits a sparse structure with $2d+1$ diagonals as described in Section 3.3, see also Table 3.7.

The matrix-free computation for the affine NGF is described in (3.68). As can be seen, the residual r_i^{NGF} originating from $\frac{\partial\psi}{\partial r^{\mathrm{NGF}}}$ is only required once for every increment vector $\left(\nabla D^{\mathrm{NGF}}\right)^i$, resulting in $\bar{m}$ element calculations. This is in contrast to the deformable matrix-free computations, where $(2d+1)\bar{m}$ element calculations were required for $\frac{\partial\psi}{\partial r^{\mathrm{NGF}}}$,

Method	matrix-based	matrix-free	
	(calc./stores)	calc.	stores
$\frac{\partial \psi}{\partial r}$	$\bar{m}$	$\bar{m}$	0
$\frac{\partial T}{\partial \hat{\mathbf{y}}}$	$d\bar{m}$	$d\bar{m}$	0
$\frac{\partial \hat{\mathbf{y}}}{\partial w}$	$d\bar{m}$	$d\bar{m}$	0
total $d=2$	$5\bar{m}$	$5\bar{m}$	0
total $d=3$	$7\bar{m}$	$7\bar{m}$	0

Table 3.6.: SSD affine derivative matrix element calculations and memory requirements. While for the matrix-based approach, all computed values are always stored to memory, for the matrix-free approach everything is computed on-the-fly without the need for intermediate storage. No recalculations are needed for the matrix-free approach, so that a significant speedup can be expected. Since the Hessian can efficiently be computed together with the gradient, no additional calculations are needed for both approaches.

see Table 3.3. Instead, the image derivatives, which only required dm matrix element calculations in the deformable case, now involve recalculations. Considering (3.68), for every increment vector $\left(\nabla D^{\mathrm{NGF}}\right)^i$, the image derivatives and also elements from $\frac{\partial \hat{\mathbf{y}}}{\partial w}$ are now needed for $2d+1$ points $j \in \mathcal{K}$, which results in $(2d+1)\bar{m}$ element calculations each and thus recalculations, see Table 3.7. Matrix elements from $\frac{\partial r^{\mathrm{NGF}}}{\partial T}$ are required at $2d+1$ points, which corresponds to the number of diagonals and thus no recalculations are needed in this case.

3.7.5. Summary and conclusion

In Section 3.7.1 it was found that for the SSD distance measure the same amount of matrix element computations is needed for the matrix-based and matrix-free approach. However, since the matrix-free approach does not require to store intermediate results, a significant speed up can be expected and the memory requirements are largely reduced.

For the NGF distance measure in Section 3.7.2 and curvature regularizer in Section 3.7.3 there is a certain computational overhead involved. While the matrix-free computations can still be performed without any intermediate storage, additional recalculations need to be performed. Therefore, we presented alternatives with selective precomputations, which reduce the number of recalculations at the cost of a moderate memory usage and will be evaluated in the next section.

For all of the analyzed matrix-free derivative computations, memory usage was reduced from $\mathcal{O}(\bar{m})$ to $\mathcal{O}(1)$ for the derivatives using fully matrix-free versions. Besides largely reduced memory consumption, as will be shown in the next chapter, this also results in an additional reduction of runtime, since memory stores are often significantly slower than floating point operations.

Method	matrix-based	matrix-free	
	(calc./stores)	calc.	stores
$\frac{\partial\psi}{\partial r^{\text{NGF}}}$	$\bar{m}$	$\bar{m}$	0
$\frac{\partial r^{\text{NGF}}}{\partial T}$	$(2d+1)\bar{m}$	$(2d+1)\bar{m}$	0
$\frac{\partial T}{\partial \hat{\mathbf{y}}}$	$d\bar{m}$	$(2d+1)d\bar{m}$	0
$\frac{\partial \hat{\mathbf{y}}}{\partial w}$	$d\bar{m}$	$(2d+1)d\bar{m}$	0
total $d=2$	$10\bar{m}$	$26\bar{m}$	0
total $d=3$	$14\bar{m}$	$50\bar{m}$	0

Table 3.7.: NGF affine derivative matrix element calculations and memory requirements. In comparison to the deformable deformation model in Table 3.3, recalculations of the image derivatives $\frac{\partial T}{\partial \hat{\mathbf{y}}}$ are required instead of $\frac{\partial\psi}{\partial r^{\text{NGF}}}$.

The general concepts for deriving matrix-free computations can be applied to use cases other than SSD, NGF and curvature regularization. Considering the general distance measure formulation $D = \psi(r(T(P(\mathbf{y}))))$, the image interpolation T and grid conversion operator P are independent of the actual distance measure, so that matrix-free formulations of their derivatives can be reused for other image distances.

Of the remaining parts, the main component is the Jacobian $\frac{\partial r}{\partial T}$. While $\frac{\partial r^{\text{SSD}}}{\partial T} = I$ for SSD, for NGF the sparse structure of $\frac{\partial r^{\text{NGF}}}{\partial T}$ favors the derivation of closed-form expressions for reduced runtimes and memory usage. Therefore, in order to derive efficient matrix-free computations, it is beneficial if

- derivative components exhibit a sparse structure with a fixed pattern, and
- result elements have no interdependencies,

enabling fully parallel, on-the-fly computations.

3.8. Implementation details

Achieving optimal performance requires fine-tuning the implementation to the characteristics of the platform and programming language. Additionally, different forms of parallelism can be utilized, ranging from multi-core computations and vectorized instructions on CPU to many-core parallelism using *general purpose computation on graphics processing units* (GPGPU).

Implementation details for matrix-free algorithms on standard CPUs will be described in Section 3.8.1, while specialized platforms such as *graphics processing units* (GPUs) and *digital signal processors* (DSPs) will be considered in Section 3.8.2 and Section 3.8.3.

```
fval ← $0$
for $\hat{k}$ in $[0, m_z - 1]$ do
    for $\hat{j}$ in $[0, m_y - 1]$ do
        for $\hat{i}$ in $[0, m_x - 1]$ do
            $i \leftarrow \hat{i} + m_x \hat{j} + m_x m_y \hat{k}$                          ▷ Compute linear index
            $i_{\pm x}, i_{\pm y}, i_{\pm z} \leftarrow$ as in (2.16) – (2.18)            ▷ Compute neighbor indices
            [dTx, dTy, dTz] ← imageDerivative($T_i$)          ▷ Compute image derivative

            r ← $[r^{\text{NGF}}_{i_{-z}}, r^{\text{NGF}}_{i_{-y}}, r^{\text{NGF}}_{i_{-x}}, r^{\text{NGF}}_{i}, r^{\text{NGF}}_{i_{+x}}, r^{\text{NGF}}_{i_{+y}}, r^{\text{NGF}}_{i_{+z}}]$   ▷ $r^{\text{NGF}}_i$ defined in (3.22)
            fval ← fval + $r^{\text{NGF}}_i \cdot r^{\text{NGF}}_i$                ▷ Accumulate function value
            dr ← $[\hat{\rho}_{i_{-z}}(z), \hat{\rho}_{i_{-y}}(y), \hat{\rho}_{i_{-x}}(x), \hat{\rho}_i(0), \hat{\rho}_{i_{+x}}(-x), \hat{\rho}_{i_{+y}}(-y), \hat{\rho}_{i_{+z}}(-z)]$
                                                              ▷ $\hat{\rho}_i$ defined in (3.34)
            drSum ← $-2\bar{h}$ (r[0]dr[0] + r[1]dr[1] + r[2]dr[2] + r[3]dr[3]
                          + r[4]dr[4] + r[5]dr[5] + r[6]dr[6])
                                                              ▷ Compute sum of (3.37)
            grad[$i$       ] ← drSum · dTx
            grad[$i + \bar{m}$ ] ← drSum · dTy
            grad[$i + 2\bar{m}$] ← drSum · dTz
        end for
    end for
end for
fval = $\bar{h}$ ($\bar{m}$ − fval)                         ▷ Compute final function value, see (2.13)
```

Algorithm 3.1: Pseudocode for the matrix-free NGF gradient (`grad`) and function value (`fval`) computation as in (3.37) for $d = 3$ for deformable registration. The algorithm can be fully parallelized over all loop iterations with a reduction for the function value `fval`. Figure modified from [KRDL18*].

3.8.1. Matrix-free computations on the CPU

Despite the recent popularity of GPU-accelerated computations, the main target platform for medical image registration algorithms is still the standard PC workstation. In order to utilize the full computational power, thread-level parallelization is indispensable on current multi-core CPUs. Furthermore, every computational core is able to utilize data level parallelism via *single instruction, multiple data* (SIMD) computations. These vector instructions perform the same operation on several values concurrently and can further improve algorithm runtime.

Implementation. In this work, we implemented all matrix-free CPU code in C++. Two main components of the algorithm that have been discussed in the previous sections are the distance measure gradient and Hessian-vector multiplication. Especially for the NGF, these components involve numerous different computations. In order simplify the translation from mathematical formulation to implementation, we provide pseudocode for the deformable NGF gradient and Hessian-vector multiplication for $d = 3$ in Algorithm 3.1 and Algorithm 3.2, including references to the corresponding definitions in the previous sections. Related computations for $d = 2$ as well as for the SSD as described in

```
N ← $[-2m_1m_2, -m_1m_2 - m_1, -m_1m_2 - 1, -m_1m_2, -m_1m_2 + 1, -m_1m_2 + m_1, -2m_1, -m_1 - 1, -m_1,$
      $-m_1 + 1, -2, -1, 0, 1, 2, m_1 - 1, m_1, m_1 + 1, 2m_1, m_1m_2 - m_1, m_1m_2 - 1, m_1m_2, m_1m_2 + 1$
      $m_1m_2 + m_1, 2m_1m_2]$                          ▷ Initialize 25 indices in N, see Table 3.1
q ← 0
for $\hat{k}$ in $[0, m_z - 1]$ do
    for $\hat{j}$ in $[0, m_y - 1]$ do
        for $\hat{i}$ in $[0, m_x - 1]$ do
            $i \leftarrow \hat{i} + m_x\hat{j} + m_xm_y\hat{k}$                  ▷ Compute linear index
            $i_{\pm x}, i_{\pm y}, i_{\pm z} \leftarrow$ as in (2.16) – (2.18)          ▷ Compute neighbor indices
            [dTx, dTy, dTz] ← imageDerivative($T_{i+\mathtt{N}}$)
                                    ▷ Compute image derivatives at 25 points
            for k in $[0, 24]$ do                 ▷ Compute 25 values of $dr^\top dr$, see (3.45)
                drdr ← 0

                for $(\tilde{i}, \tilde{j})$ in $N^{-1}$(N[k]) do
                                 ▷ Compute 49 values of $\hat{\rho}_i$, see (3.44) and Table 3.1
                    drdr ← drdr + $\hat{\rho}_{i+\tilde{i}}(-\tilde{i}) \cdot \hat{\rho}_{i+\mathtt{N[k]}+\tilde{j}}(-\tilde{j})$
                end for

                q[$i$      ] ← q[$i$      ] + dTx[12] · drdr · dTx[k] · $\hat{\mathrm{p}}_{i+\mathtt{N[k]}}$
                q[$i + \bar{m}$] ← q[$i + \bar{m}$] + dTy[12] · drdr · dTy[k] · $\hat{\mathrm{p}}_{i+\mathtt{N[k]}+\bar{m}}$
                q[$i + 2\bar{m}$] ← q[$i + 2\bar{m}$] + dTz[12] · drdr · dTz[k] · $\hat{\mathrm{p}}_{i+\mathtt{N[k]}+2\bar{m}}$
                 ▷ Element [12] corresponds to the derivative at index $i$ + N[12] = $i$
            end for
        end for
    end for
end for
```

Algorithm 3.2: Pseudocode for the matrix-free NGF Hessian-vector multiplication $\hat{H}\hat{\mathbf{p}} = \hat{\mathbf{q}}$ for $d = 3$ for deformable registration as in (3.45). The algorithm can be fully parallelized over all loop iterations, computing three elements of the result simultaneously in each thread. For ease of presentation the inner computations involving k and $(\tilde{i}, \tilde{j})$ have been formulated as loops. These are fully unrolled in the actual implementation. Figure modified from [KRDL18*].

Section 3.2 can be derived by simplifying these algorithms. For simplicity, implementations without grid conversion are shown, which then needs to be performed in separate steps, as discussed in Section 3.5.3. In Algorithm 3.2, for ease of presentation the inner computations involving k and $(\tilde{j}, \tilde{j})$ are shown as a loop, but are explicitly implemented (unrolled) in the actual implementation. Additionally, as shown in Algorithm 3.2, to reduce the number of recalculations, three elements of the result vector are computed in each inner loop iteration of the Hessian-vector multiplication.

Parallelization. In Algorithm 3.1 as well as Algorithm 3.2, every loop iteration can be computed in an individual thread, resulting in a maximum of $\bar{m}$ concurrent threads. For parallelization, we utilized the OpenMP framework [DM98], which enables parallelization by adding compiler directives to the code. In order to parallelize individual loop iterations the `#pragma omp for` statement was used.

As utilization of individual threads comes with a certain overhead, a suitable balance between the amount of work that is performed by each thread and the total number of threads has to be found. For the CPU code, the parallelization was performed for the *outer loop*, iterating over $\hat{k}$, resulting in m_z threads for $d = 3$ ($\hat{j}$ with m_y threads for $d = 2$). Since m_z (or m_y for $d = 2$) is in most cases still larger than the typical number of cores on the CPU, this ensures sufficient computational load for each core and reduces thread management overhead. For parallelization on GPU a more fine-grained parallelization strategy is used (Section 3.8.2).

Additionally, in Algorithm 3.1 the function value is accumulated over all loop iterations. To achieve this without write conflicts from different threads, we used the OpenMP `#pragma omp for reduction(+: fval)` statement. However, in this case the small number of m_z (m_y for $d = 2$) individual results of all threads could also easily be stored separately and a reduction could be performed manually afterwards.

For the affine deformation model as described in Section 3.6, additional reduction variables are needed. As can be seen in (3.63) for SSD and (3.69) for NGF, all $d^2 + d$ elements of the gradient are updated in each inner loop iteration, thus $d^2 + d$ reduction variables are required. Additionally, the affine Hessian computations in (3.66) and (3.72) update a matrix with $(d^2 + d)^2$ values in each iteration. Exploiting the symmetry of the Hessian, additional $\frac{(d^2+d)(d^2+d+1)}{2}$ reduction variables are required here, which we also implemented using the OpenMP `reduction` statement.

Vectorization. In addition to multiple computational cores, most modern CPUs also feature additional vectorized instructions. These *single instruction, multiple data* (SIMD) instructions can perform the same operation simultaneously on different data. On the x86 architecture, the recent version of SIMD instructions is called *Advanced Vector Extensions* (AVX) [Lom11] and provides registers with a size of 256 bits for computations. Using typical `double` values with a size of 64 bits (8 bytes), each operation can be performed on four values simultaneously. To utilize AVX in our C++ code, we used the Intel AVX Intrinsics [Lom11], which enable a manual vectorization of the code using specialized function calls for SIMD operations.

Several ways of vectorizing an algorithm with AVX exist. For example, each function can be optimized individually by grouping identical arithmetic operations together and performing them with SIMD operations. In contrast to this, due to the favorable structure of the matrix-free computations, we were able to replace all operations in the inner loop of Algorithm 3.1 and Algorithm 3.2 by SIMD operations. This means that four indices $\hat{\imath}$ are processed simultaneously, such that the inner loop can be incremented by four elements at a time. For cases where m_x is not divisible by four, special boundary handling is required: The remaining 1-3 iterations are simply performed by using the non-vectorized code.

3.8.2. Graphics processing units

The matrix-free algorithm exhibits favorable characteristics for a GPU implementation: parallelizability and low memory requirements. In comparison to the CPU, current GPUs exhibit a many-core architecture with a much higher amount (i.e., thousands) of parallel processing units. In order to exploit this architecture, excellent parallelizability is mandatory. Additionally, common GPUs are implemented as separate expansion cards so that all data has to be copied to the GPU device before computation. As GPU memory is limited and cannot be easily upgraded, memory requirements can become a serious concern.

We implemented the matrix-free registration for the rigid and affine deformation model as well as for deformable registration. The GPU-based matrix-free registration was presented in the Master's Thesis [Tra14] and conference paper [TRK+14*] for rigid and affine registration. In the Master's Thesis [Mei16], a GPU-based deformable registration was developed, which is evaluated in Chapter 5.

For utilizing the GPU acceleration capabilities, we relied on the popular *NVIDIA Compute Unified Device Architecture* (CUDA) C/C++ framework [NVI17]. In the following, we will discuss how specialized platform features such as different memory areas have been utilized and give additional implementation details.

Parallelization. The CUDA framework separates two scopes of code execution: *device* and *host*. *Host code* executes on the CPU. It performs GPU initialization, data transfer and manages the execution of GPU code. Code that runs on the GPU is called *device code*. The device code executes in specialized parallel programs, so-called *kernels*, which are launched from host code.

Each kernel is executed in parallel on a number of *threads*, which can be specified in its function call. When executing on the GPU, multiple threads are additionally grouped into *thread blocks*, which will become important when dealing with the memory layout in the next section.

The number of parallel processors, so-called *CUDA cores*, is significantly larger than the usual number of cores on a standard CPU. For example, the GeForce GTX980 graphics card that we used for the evaluation in Chapter 5 features 16 so-called *streaming multiprocessors* (SMP) with 128 CUDA cores each, resulting in a total of 2048 CUDA cores. Therefore, we used a different parallelization strategy. As described in Section 3.8.1, parallelization on the CPU was restricted to the outer loop, resulting in m_z threads for $d = 3$ (m_y for $d = 2$), in order to provide high computational load for each thread and to reduce initialization overhead. On the GPU, however, there are typically more than m_z (m_y for $d = 2$) CUDA cores available. In order to fully utilize the GPU, we therefore performed a full parallelization for all loops, resulting in $\bar{m}$ individual threads that are distributed onto the CUDA cores. With this, we ensure a sufficient utilization of the GPU.

Memory layout. Before computations in device code can be performed, all required memory contents must be transferred from the host main memory to the device global memory. On a standard PC workstation, these transfers are performed over the PCIe 3.0

bus with a theoretical maximum bandwidth of $\approx 16\,\mathrm{GB\,s^{-1}}$, resulting in a measured performance of $\approx 12\,\mathrm{GB\,s^{-1}}$. Compared to a theoretical maximum bandwidth of $224\,\mathrm{GB\,s^{-1}}$ for device-to-device transfers on the GeForce GTX980, transfers from and to the host are significantly slower and can thus become a bottleneck.

Therefore, all steps of the registration algorithm were implemented in GPU code, which minimizes the required host-to-device transfers. The reference image R and template image T are transferred to the GPU only once at the beginning of the algorithm. The multi-level image pyramid is created directly on the GPU and remains in GPU global memory. All further computations are performed in GPU memory, so that only the final registration result needs to be copied back to the host memory.

Besides the GPU global device memory, further specialized memory areas exist, which were exploited in our implementation, see Figure 3.8. These memory areas provide faster access times than global memory, but have limited scope or size. The first is *constant memory*. Constant memory is limited to a size of 64 kB and read-only, but can be cached and read from all threads. Thus, it is ideal for small values that are constant but accessed frequently. Therefore, we stored the values $m, m^{\mathrm{y}}, h, h^{\mathrm{y}}, \bar{m}, \bar{m}^{\mathrm{y}}\bar{h}, \bar{h}^{\mathrm{y}}$ in constant memory.

Another specialized memory area is the *shared memory*. The shared memory has a limited size of 96 kB per multiprocessor, but is located on-chip and thus very fast. However, it can only be accessed from within the same thread block. Therefore, for deformable registration, we performed the reductions for the function value, also discussed in Section 3.8.1, in shared memory, using an efficient reduction scheme described in [Har07]. Additionally, the affine registration relies heavily on shared-memory use, since also gradient and Hessian-vector computations contain reductions (Section 3.6).

Image interpolation. Another specialized memory area is the *texture memory*. Texture memory provides caching and optimized memory patterns for localized access in 2D or 3D and is thus faster than global memory. Additionally, texture memory supports hardware-based linear interpolation, including automatic boundary value handling. Thus, texture memory appears ideal for the the reference image R and especially the template image T (Section 2.4.3).

However, as described in [Mei16], the interpolation weights for hardware-based linear interpolation are stored in 9-bit fixed point format [NVI17, §G.2]. This leads to comparatively large errors in the order of 10^{-3} in comparison to interpolation on the CPU, which results in erroneous image gradient computations and caused the optimization to fail for deformable registration with NGF. Therefore, we relied on manual interpolation on the GPU with full precision.

Grid conversion. The transposed grid conversion operator $P^{\top}\hat{\mathbf{y}}$ was implemented on the CPU using a red-black scheme in order to avoid write conflicts and to enable efficient parallelization (Section 3.5.2, Figure 3.7). However, in this scheme parallelization can only be performed for the outer loop, which does not fully utilize all CUDA cores, as discussed earlier. Therefore, for the GPU implementation of the transposed grid conversion operator, we used the specialized *atomic operations* [NVI17, §B.12], specifically the `atomicAdd` function. Atomic operations enable write access to global and shared

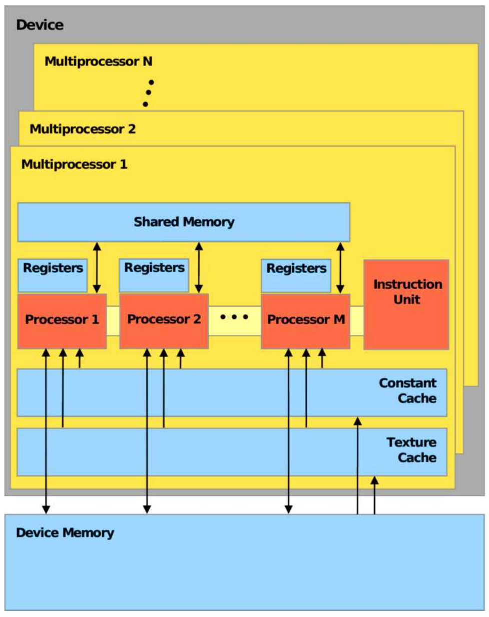


Figure 3.8.: CUDA hardware model, including different memory areas. While the global device memory is large but comparatively slow, constant memory and shared memory are faster but with small size and are read-only or only available within the same thread block, respectively. Image modified from [NVI09, §4.1].

memory without write conflicts and allow for a fast computation of the transposed grid conversion operator without the red-black scheme. Note that while atomic operations also exist on the CPU with OpenMP, we found that these operations are significantly slower than the red-black scheme in this case.

3.8.3. Digital signal processors

A different interesting platform for accelerating image registration are so-called *digital signal processors*. In comparison to CPUs and GPUs, these are specialized processors, which are optimized for performing algorithmic calculations while exhibiting a low power consumption and are therefore often utilized in embedded applications. The recent availability of *multi-core* DSP processors and the special constraints of the embedded platform make DSPs an attractive target for a matrix-free registration algorithm.

We presented and implemented a prototype of DSP-based matrix-free registration in [BKR+14*]. In addition to multi-core DSPs, this prototype includes a distributed computing concept with four DSP processors connected over a local area network.

This unique setup requires additional extensions to the algorithm in order to efficiently utilize the distributed DSP setup, which will be presented exemplarily in the following. More details on the setup and implementation can be found in [BKR+14*] as well as in the Bachelor's Thesis [Ber12].

Registration framework. For the implementation of the DSP-based matrix-free registration, a rigid deformation model in 2D was considered. In practice, rigid deformations are often required when pre-aligning images before a deformable registration or, e.g., for comparison of similar objects for quality control. We utilized the SSD distance measure, which is suitable for mono-modal registration. In combination with 2D images, this registration type is often required for registration of camera images in industrial manufacturing lines or surveillance scenarios.

As described in Section 3.6, in order to limit the affine deformation model to rigid motion (rotation and translation only), the parameters w in the affine distance measure $D(\hat{\mathbf{y}}(w))$ from (3.55) are replaced by a function

$$A(\theta) : \mathbb{R}^{\vartheta} \to \mathbb{R}^{d^2+d}.$$

For $d = 2$, we set $\vartheta = 3$ with $\theta := (\alpha, t_x, t_y) \in \mathbb{R}^3$, where α indicates rotation angle and t_x, t_y are translation parameters. The rigid deformation model results in the optimization problem

$$\min_{\theta \in \mathbb{R}^3} D(\hat{\mathbf{y}}(A(\theta))),$$

with

$$A(\theta) := (\cos(\alpha), \sin(\alpha), -\sin(\alpha), \cos(\alpha), t_x, t_y)\,,$$

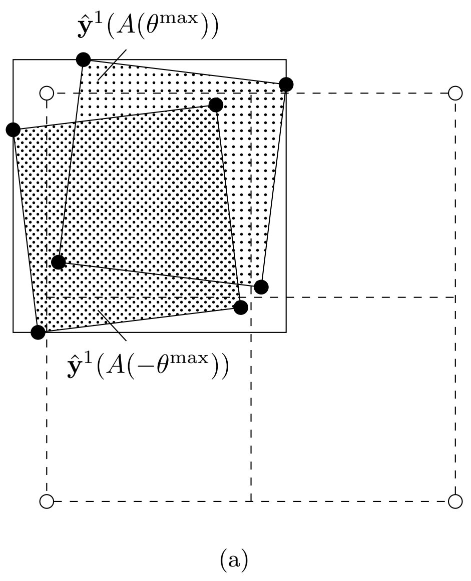
representing a rotation matrix with additional translation components.

From this, matrix-free derivatives can be easily computed analogously to Section 3.6.1, see [BKR+14*] for details. An interactive live-demo of this matrix-free rigid registration algorithm in 2D, running in a web-browser on the local CPU can be found at `http://imaging.live/rigid-image-registration`.

Distribution of image data. As a major difference to the previously presented approaches, the distributed setup requires an additional strategy for data handling. In [BKR+14*], we utilized a setup of four independent processors $\mathrm{DSP}^1, \dots, \mathrm{DSP}^4$ connected over a local area network. Since each of the processors has its own private RAM, initially the image data needs to be transferred. For parallelization, the discretized image grid $\mathbf{x}$ on the reference image domain $\Omega_{\mathcal{R}}$ is split into four equal squares, each containing the image grid points $\mathbf{x}^1, \dots, \mathbf{x}^4$, such that DSP^k computes the function value and derivative calculations for the image grid points $\mathbf{x}^k$, $k = 1, \dots, 4$. Besides the benefit of parallel computation, this also reduces the amount of image data that needs to be transferred to the DSPs.

However, while for the reference image R the image data that needs to be transfered to DSP^k is equal to the pixels located at $\mathbf{x}^k$, due to the transformation, additional image data is required for computing the deformed template image $T^k(\theta) := T(\hat{\mathbf{y}}^k(A(\theta)))$, where $\hat{\mathbf{y}}^k$ is the transformed image grid $\mathbf{x}^k$, depending on the transformation parameters θ, analogously to (3.54).

Without any further assumptions, the full template image would need to be transferred to each DSP. Therefore, in order to limit the amount of transferred data, worst-case transformation parameters $\theta^{\max} = (\alpha^{\max}, t_x^{\max}, t_y^{\max})$ are introduced. From these



$\hat{\mathbf{y}}^1(A(\theta^{\max}))$
$\hat{\mathbf{y}}^1(A(-\theta^{\max}))$
(a)

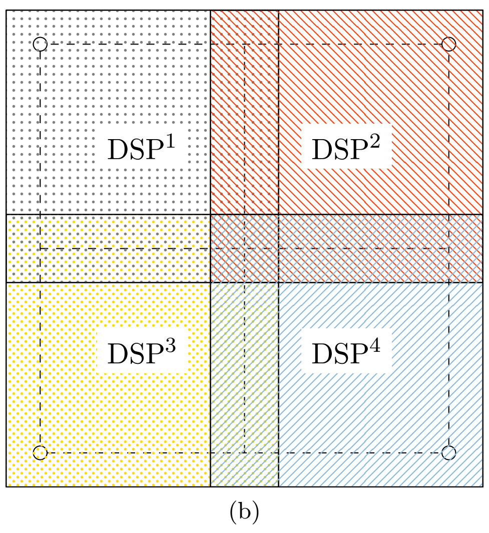


Figure 3.9.: Distribution of image data to four DSPs, (a): The maximum image area of T (white circles) to be transferred to each DSP (solid lines) is computed by dividing T into four parts (dashed lines) and applying worst case transformation parameters $\pm\theta^{\max}$ to each of them; shown exemplarily for DSP^1, (b): Final image data of T that is transferred to each DSP. Areas outside of T are not transferred, but inserted by padding the out-of-bounds value during transfer. Both strategies minimize the amount of image data that is transferred and thus decrease total runtime.

worst-case parameters, all permutations within $-\theta^{\max}$ and $\theta^{\max}$ are sampled in $\Theta := [-\alpha^{\max}, \alpha^{\max}] \times [-t_x^{\max}, t_x^{\max}] \times [-t_y^{\max}, t_y^{\max}]$. The maximum area of T, that needs to be transferred for DSP^k is then determined by the rectangle spanned by the two points

$$\hat{\mathbf{y}}^k_{\text{min}} := \min_{\theta\in\Theta} \hat{\mathbf{y}}^k(A(\theta)) \in \mathbb{R}^2$$
$$\hat{\mathbf{y}}^k_{\text{max}} := \max_{\theta\in\Theta} \hat{\mathbf{y}}^k(A(\theta)) \in \mathbb{R}^2,$$

i.e., the component-wise minimum and maximum transformed coordinates that can be reached by any combination of the worst-case transformation parameters $\theta^{\max}$, as shown in Figure 3.9.

In many applications, the worst case parameters $\theta^{\max}$ can be determined from the field of view or physical setup, such that they cannot be exceeded.

Data padding for boundary conditions. As discussed, e.g., in Section 2.4.3, the required image interpolation for the deformed template image $T(\hat{\mathbf{y}})$ uses Dirichlet boundary conditions for values outside of the image domain. In an implementation, these boundary conditions require conditional statements distinguishing whether valid image data or an out-of-bounds value has to be used. However, these conditional statements introduce branch commands into the program.

In a so-called pipelined processor, such as the DSPs that we used, multiple instructions are executed in parallel in different stages of a pipeline, increasing computational performance. Conditional branches in the instruction sequence however pose so-called pipeline hazards, which require the instruction pipeline to be rebuilt, and result in performance losses.

A solution to prevent pipeline hazards in the image interpolation is data padding. The image data is artificially enlarged at the boundaries with areas that are filled with the out-of-bounds value. This way, the computation can be performed without conditional statements and thus without pipeline hazards.

In our DSP setup, the fact that the data needs to be initially copied to each DSP can be exploited for this. From the previous section, the amount of padding that is required for each DSP is already known from the worst-case transformation parameters $\theta^{\max}$, as shown in Figure 3.9(b). When copying the image data to each DSP, the data can now be placed at the correct, padded position in DSP memory, which is pre-initialized with the out-of-bounds value (typically zero). This way, the amount of transferred memory remains equal to only the valid image data values, while still achieving padded image data on the DSPs, that can be computed without pipeline hazards.

3.9. Summary

In this chapter, matrix-free methods for derivative computations of distance measures and curvature regularization were derived and analyzed. In Section 3.1, we derived analytical derivatives for gradient and Gauss-Newton Hessian approximations as a product of Jacobian matrices. These formulations served as a starting point for derivation of matrix-free computations in the following sections.

In Section 3.2, matrix-free computations were derived for the SSD distance measure. Both, gradient computations as well as a matrix-free Hessian-vector multiplication were considered. For both cases, we derived fully closed-form matrix-free expressions that do not require any intermediate storage and are parallelizable element-wise. In the following Section 3.3, we used these findings to derive matrix-free computations for the more involved NGF. Similar to the SSD before, we developed matrix-free expressions for the gradient as well as a Hessian-vector multiplication. Additionally, in Section 3.4 we determined a matrix-free formulation for the curvature regularizer.

Finally, in Section 3.5 we considered grid-conversion operators for mapping between image grid and deformation grid. We derived efficient, matrix-free and parallelizable computation schemes, including a red-black method for computation of the transposed grid conversion operator. Additionally, methods for directly integrating the grid conversion into the distance measure computations were discussed. With this, matrix-free versions of all important components of the objective function derivatives for deformable registration were derived, allowing for a memory efficient and parallel computation.

Furthermore, in Section 3.6 we analyzed the affine and rigid deformation model. It was shown that due to the different structure, different methods for deriving a matrix-free formulation have to be utilized and corresponding expressions were subsequently derived for the objective function.

All previously derived matrix-free computations were analyzed from a theoretical point of view in Section 3.7. Special emphasis was placed on memory usage and algorithm runtime and it was found that options exist to influence the trade-off between both. Additionally, comparisons were performed with a matrix-based approach and benefits of the matrix-free computations in terms of memory usage and possible recalculations were discussed.

Finally, in Section 3.8 we discussed details on specific implementations of the matrix-free approach. Starting with a standard CPU implementation, methods of parallelization were discussed and pseudocode for gradient and Hessian-vector multiplications was given. Since the derived methods are not limited to a specific architecture, they can additionally easily be used to exploit the benefits of specialized hardware platforms such as GPUs and DSPs. These were discussed in the following, giving insights on exploitation of specific platform characteristics in order to achieve optimal performance.

This concludes the derivation and theoretical analysis of the proposed matrix-free registration approach. In Chapter 5, its practical performance is evaluated in different implementations and applications.

4 Automatic differentiation

In addition to manually determining the derivatives of the objective function as described in Chapter 3, automatic approaches to determine analytical derivatives exist [NW06, §8]. An approach that has been routinely used in areas such as optimal control [GW08, §1], but recently received more attention due to the popularity of “deep learning” techniques, is *automatic differentiation*. Also called algorithmic differentiation, these methods are capable of automatically computing function derivatives only based on the source code of the function. In contrast to symbolic differentiation, instead of closed-form expressions of the derivative only numerical values are computed. However, these numerical values represent the exact derivatives and are not prone to approximation errors such as finite difference approximations [BPRS18].

In automatic differentiation, derivatives are computed automatically by tracing individual computational operations during the function evaluation and repeated application of the chain rule. Apart from implementation of the function itself, this approach does not require any further mathematical derivations from the user. Therefore, automatic differentiation presents an alternative to “manual” methods presented in Chapter 3, eliminating the burden of manual derivative computations.

In the following, after a general introduction, we will first present the two basic operation modes of automatic differentiation in Section 4.2 and Section 4.3. After this, we will introduce the Theano framework [AAA+16], which implements a special variant of algorithmic differentiation, allowing for easy implementation and fast derivative evaluation. In Section 4.5, we will then discuss details and pitfalls we encountered while implementing the registration objective function in Theano. Finally, in Section 4.6, we evaluate different aspects of the implemented Theano functions such as compilation and evaluation time on CPU as well as GPU. A comparison with the hand-tuned, manual implementation of the registration algorithm is postponed to Chapter 5.

4.1. Introduction

For derivative computations, automatic differentiation relies heavily on the chain-rule of calculus. The original function $f(x)$ is interpreted as a composition of elementary operations $f_i(x)$ with known derivatives, such that

$$f(x) = f_1 \circ f_2 \circ \ldots \circ f_n.$$

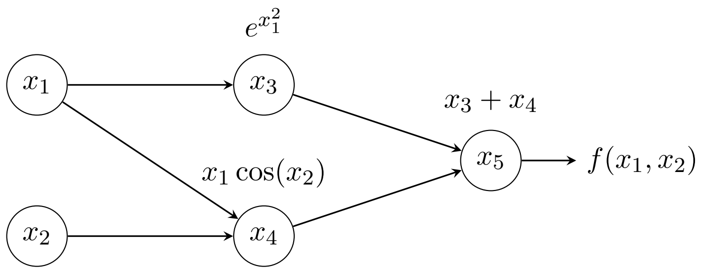


Figure 4.1.: Computational graph for $f(x_1, x_2) = x_1 \cos(x_2) + e^{x_1^2}$, each node corresponds to an elementary computational operation. The function value is computed by traversing the graph from left to right.

The elementary operations can, for example, be basic operations such as addition, subtraction, multiplication or division of variables, trigonometric functions such as $\sin(x)$ and $\cos(x)$, or the exponential e^x. Each operation is then differentiated and the derivatives are evaluated recursively according to the chain rule.

As an example, consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ with

$$f(x_1, x_2) = x_1 \cos(x_2) + e^{x_1^2}. \tag{4.1}$$

The function $f(x_1, x_2)$ can be decomposed into elementary operations, complementing the input variables x_1, x_2 by intermediate variables

$$\begin{aligned} x_3 &:= e^{x_1^2} \\ x_4 &:= x_1 \cos(x_2) \\ x_5 &:= x_3 + x_4. \end{aligned} \tag{4.2}$$

Evaluating these expressions consecutively from top to bottom with initial values x_1, x_2 can be interpreted as a computational graph of concatenated operations as shown in Figure 4.1, obtaining x_5 as the final function value output.

Utilizing the chain rule, this structure can now be used to automatically compute numerical values for derivatives with automatic differentiation. Here, two major computation modes exist, forward and reverse mode computation, which will be discussed in the following. Further comprehensive information on automatic differentiation can be found in the textbook [GW08].

4.2. Forward mode

The *forward mode*, also called tangent-linear mode, enables the computation of a directional derivative $\nabla f^\top p$ in the seed direction $p \in \mathbb{R}^k$, with $k = 2$ in the example (4.1). The computational graph is traversed from inputs to outputs and partial derivatives of the intermediate functions are concatenated following the chain rule [NW06, §8.2].

Using the previous example, we calculate the directional derivatives of the intermediate variables (4.2) with respect to the input variables as

$$\dot{x}_i := \nabla x_i^\top p = \left(\frac{\partial x_i}{x_1}, \frac{\partial x_i}{x_2}\right)(p_1, p_2)^\top = \frac{\partial x_i}{x_1} p_1 + \frac{\partial x_i}{x_2} p_2,$$

and obtain

$$\begin{aligned}
\dot{x}_3 &= 2x_1 e^{x_1^2} p_1 \\
\dot{x}_4 &= \cos(x_2) p_1 - x_1 \sin(x_2) p_2 \\
\dot{x}_5 &= \dot{x}_3 + \dot{x}_4.
\end{aligned}$$

With initial values for p and x_1, x_2, a value for the directional derivative $\dot{x}_5 = \nabla f^\top p$ can be determined by following this expression from top to bottom, corresponding to a forward direction through the computational graph.

A great benefit of the forward mode is that the computation of the derivative can be performed simultaneously with the function evaluation. Additionally, for each step only the values of the parent nodes are required. Thus it is not required to store further derivative information or to have all derivative information available at the same time.

However, for each directional derivative, one forward pass or “sweep” through the computational graph has to be performed. Using the i-th unit vector e_i and $p = e_i$ yields the i-th column of the Jacobian matrix. Thus, for a function $f : \mathbb{R}^N \to \mathbb{R}^M$, N passes are required for the computation of the full Jacobian. For functions such as the deformable objective function (2.24) with $J(\mathbf{y}) : \mathbb{R}^{d\bar{m}^y} \to \mathbb{R}$, $d\bar{m}^y$ passes are required to compute the gradient, which is highly inefficient [BPRS18].

Nevertheless, the forward mode can still be utilized in computing higher-order derivatives. The directional derivative $\nabla f^\top p$, which can be computed in a single pass, corresponds to a right-hand Jacobian-vector multiplication. This can be utilized when computing the Hessian-vector multiplication $H\mathbf{p}$ for the Gauss-Newton method, as shown in Figure 3.5. In this case, instead of applying the forward mode to the function itself, it is applied to the Jacobian of the objective function, effectively computing $H\mathbf{p}$. The Jacobian matrix itself is previously computed by using the reverse mode, described in the next section. Further details on the automatic Hessian computation are also given in Section 4.5.5.

4.3. Reverse mode

In comparison to the forward mode, the automatic differentiation *reverse mode* traverses the computational graph in inverse direction, from the outputs to the inputs. Thus, the derivative computation cannot be performed simultaneously with the function value computation and an additional reverse pass through the graph is required. Again, the reverse mode utilizes the chain rule, but now initially computing partial derivatives at the outputs and then recursively updating the derivative values in the direction of the input variables. For this, the reverse mode adds so-called adjoint variables $\bar{x}_i$ to each node x_i in the computational graph, defined as

$$\bar{x}_i := \frac{\partial f}{\partial x_i} = \sum_{x_j \in \text{children}(x_i)} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i}$$

[NW06, §8.2], which again results from the chain rule. In our example from (4.1), starting with $\bar{x}_5 = \frac{\partial f}{\partial x_5} = 1$ since x_5 is the output variable, we can now compute

$$\begin{aligned}
\bar{x}_5 &= \frac{\partial f}{\partial x_5} = 1 \\
\bar{x}_4 &= \frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_5}\frac{\partial x_5}{\partial x_4} = 1 \cdot 1 \\
\bar{x}_3 &= \frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_5}\frac{\partial x_5}{\partial x_3} = 1 \cdot 1 \\
\bar{x}_2 &= \frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial x_4}\frac{\partial x_4}{\partial x_2} = 1 \cdot (-x_1 \sin(x_2)) \\
\bar{x}_1 &= \frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_4}\frac{\partial x_4}{\partial x_1} + \frac{\partial f}{\partial x_3}\frac{\partial x_3}{\partial x_1} = 1 \cdot \cos(x_2) + 1 \cdot 2x_1 e^{x_1^2},
\end{aligned}$$

by processing the computations from top to bottom, moving backwards through the computational graph. The derivatives $\frac{\partial x_j}{\partial x_i}$ are obtained from (4.2) and their values are stored in the forward pass for each node. As can be seen, we finally obtain $\nabla f = (\bar{x}_1, \bar{x}_2)^\top$ and thus a value for all elements of the gradient in a single reverse pass. Note, that in an actual computation, numerical values are substituted for x_1, x_2 and the derivatives that are propagated through the graph, such that a full symbolic expression for the derivative never exists.

Generally, for each output variable, in the reverse mode a row of the Jacobian matrix can be computed in each pass, such that for computing the full Jacobian of a function $f : \mathbb{R}^N \to \mathbb{R}^M$, M passes are needed. This is particularly advantageous for the deformable registration objective function (2.24) with $J : \mathbb{R}^{d\bar{m}^y} \to \mathbb{R}$, where the gradient can be computed in a single pass. However, a drawback of the reverse mode is that an initial forward pass is required, computing the function value and storing all intermediate derivative values. Depending on the function structure and the number of elementary operations, this can require a considerable amount of memory.

4.4. Theano framework

There exist many different frameworks for automatic differentiation in several programming languages [NW06, §8.2], [GW08, §6], [BHWB18]. Due to the recent popularity of deep learning algorithms, the concept of automatic differentiation has also received much attention in the field of machine learning. Several different frameworks with focus on deep learning have been developed, consisting of an automatic differentiation framework at the core, which is required for backpropagation in neural networks [BPRS18].

Besides others, such as TensorFlow [ABC+16], PyTorch [PCL+17], Caffe2 [JSD+14] and CNTK [SA16], a widely used framework in this context is *Theano* [AAA+16; BLP+12;

BBB+10]. Theano is based on the Python programming language and uses a syntax similar to the popular NumPy library [WCV11] to define tensor variables for automatic differentiation. This makes Theano a suitable choice for an implementation of the registration algorithm as given in Chapter 2, automatically determining the required derivatives. Additionally, Theano computations can automatically be performed on the GPU without manual intervention.

An intriguing question is whether these automatic tools can offer similar performance as the manual matrix-free approach discussed in the previous chapters. In order to investigate this question, we implemented the deformable image registration in Theano. This allows for a direct comparison in terms of runtime as well as memory usage on CPU and GPU (Section 5.3 and Section 5.4). In the following, we will discuss the details of the automatic differentiation framework in Theano, as well as details of our implementation.

Special properties of Theano. In comparison to the concept of automatic differentiation as discussed in the previous sections, Theano contains additional features that optimize the computational graph before computation. Therefore, its computations can be interpreted as a hybrid of symbolic differentiation and automatic differentiation [BPRS18].

For derivative computation, Theano builds a computational graph that is evaluated in reverse mode. Additionally, so-called R-operators are available, which perform forward mode differentiation [AAA+16]. These operations are based on the automatic differentiation principles as discussed before.

However, Theano additionally utilizes a graph optimization algorithm to simplify computations and improve numerical stability. This optimization allows to replace parts of the graph by optimized operations, to simplify computations, precompute constants and to remove redundant operations. Additionally Theano performs optimizations for numerical stability by replacing certain expressions with stable implementations [AAA+16].

This has different effects: First, an additional compilation step is required before function value or derivatives can be computed. In this step, the graph is built and optimizer and stabilization are applied. In contrast, in a "naïve" automatic differentiation framework, the graph is built on-the-fly while evaluating the function value. Second, the graph optimization poses some limitations on control flow and loops in the function computation. In a pure automatic differentiation framework, control flow statements and loops are transparent to the algorithm. In Theano, this is realized by using certain `scan` and `IfElse` operations, which require special consideration.

Additionally, Theano not only optimizes the graph, but replaces suitable operations by compiled C/C++ or GPU code. This enables to execute Theano functions on a GPU device without any code modifications. Furthermore, the optimized graph structure reduces memory requirements for the reverse mode, making Theano even more interesting for comparison with the matrix-free methods.

Additionally, as no manual effort is required for derivative computations, this approach is also well-suited for rapid prototyping. New distance measures or regularizers can be implemented and evaluated quickly without the lengthy process of manually determining, implementing and testing derivatives. The automatic differentiation framework also allows to compute exact higher-order derivatives, such as the full Hessian matrix, that are tedious to compute manually.

4.5. Implementation details

We implemented the NGF and SSD distance measures as well as the curvature regularizer, including grid conversion and linear interpolation in Theano. In the following, we will discuss important obstacles we encountered during the implementation and details that deserve special attention. Additionally, useful hints for implementing a registration objective function in Theano and integration with an existing framework will be given. For a full tutorial and examples on the usage of Theano, we refer to the project web page [The18].

4.5.1. Tensor types and function compilation

In order to create functions that can be differentiated automatically, in Theano so-called *tensor datatypes* are utilized. These tensors serve as symbolic variables and do not necessarily have associated data or fixed sizes. However, their basic data type, such as integer or floating-point, as well as their structure, such as vector or matrix, have to be defined.

The tensor variables are then used to program the function value computation in usual Python code, as shown in Listing 1. The resulting symbolic expression can then be compiled into a callable function by calling `theano.function` and specifying the input parameters. Additionally, the expression can be automatically differentiated by calling `theano.tensor.grad`. This results in another symbolic expression for the gradient, which again can be compiled into a callable function.

As can be seen in Listing 1, for computations that are not operations between two tensor variables, such as product or sum, specialized Theano operations have to be used, since the exact size of the tensor and its value is not specified when creating the graph.

4.5.2. Vectorization

As discussed earlier, Theano supports loops in the code by using the `scan` function. Since the discussed distance measures and regularizers contain an element-wise sum, this could be implemented in Theano with the `scan` function, similar to the gradient in Algorithm 3.1, looping over all image points. However, we found that the `scan` function is associated with a runtime penalty since its the execution is not parallelized. Therefore, we moved to a *vectorized implementation*. As shown exemplarily in Listing 1, in a vectorized implementation, the loop is implemented as element-wise operations on the elements of a vector. These computations can then benefit from optimized, parallel implementations during the graph optimization step.

```
1  import theano, numpy as np
2
3  #define images as tensors with float vector type
4  R, T, h = theano.tensor.vectors('R','T','h')
5
6  #create symbolic expression for function value
7  r = R-T
8  fval = theano.tensor.prod(h)*theano.tensor.sum(r**2)
9
10 #compile function
11 f = theano.function([R,T,h],fval)
12
13 #automatically compute gradient wrt. T
14 grad = theano.tensor.grad(fval,T)
15
16 #compile gradient function
17 df = theano.function([R,T,h],grad)
18
19 #evaluate for d=2 with random data of size 32x32
20 prod_m = np.prod([32,32])
21 R_rand = np.random.rand(prod_m)
22 T_rand = np.random.rand(prod_m)
23 h_rand = np.random.rand(2)
24
25 f(R_rand, T_rand, h_rand)
26 df(R_rand, T_rand, h_rand)
```

Listing 1: Exemplary Python script using Theano to create and evaluate a SSD-like function and the corresponding gradient. In a full implementation of the registration, additionally image interpolation and grid conversion are required, depending on the deformation $\mathbf{y}$. The gradient is computed fully automatic without additional user input.

4.5.3. Graph optimization and compilation

There are two different aspects when evaluating the runtime of Theano-based computations: function compilation time and runtime of the function object. The compilation time includes graph optimization and only needs to be performed once to create a function object. Afterwards, the function can be called freely, without requiring another compilation step. In Listing 1, the compilation steps are performed in lines 11 and 17. The resulting functions are then evaluated in lines 25 and 26, respectively. As the compilation step is generally much more time consuming than the function evaluation itself, it is beneficial to minimize the number of function compilations and to reuse compiled functions as often as possible, as discussed in the next paragraph. Note, however, that Theano keeps a local compilation cache, which can also reduce the function compilation time, if similar functions are compiled multiple times on the same machine.

4.5.4. Reuse of compiled functions

The code in Listing 1 passes the spacing `h` as well as the images `R,T` to Theano as independent variables of `f`, rather than defining them as NumPy-arrays before the compilation step and compiling them into the graph. The advantage of this approach is that the function `f` is now independent of the image data, size, and resolution and can be reused in a multi-level scheme and with arbitrary images without recompiling. In our experiments we found no significant runtime difference for the function execution between both approaches.

For GPU computations, additionally the `theano.shared` function can be beneficial, defining variables that are compiled into the graph and automatically copied to the GPU. Different values for the variables can then be substituted by using the `set_value` command.

While a single compilation step is faster than compiling a new function for every registration level, it can still take up a considerable amount of time. This time can be further reduced by using the Python `pickle` library. This allows to store and reuse functions even for multiple registration calls with different data, which further reduces the runtime. Note, however, that for each platform such as CPU or GPU a separate function has to be stored, since different specialized operations are used in the computational graph.

4.5.5. Hessian computation

The creation and compilation of a function and its gradient has been shown exemplarily in Listing 1. As previously discussed in Section 4.2, for computing a Hessian-vector multiplication $H\mathbf{p}$, the forward mode of automatic differentiation is beneficial. In Theano, this is realized by the `theano.tensor.Rop` operator, which implements the multiplication of the Jacobian matrix with a vector from the right-hand side. Applied to the gradient from Listing 1, this can be implemented as

```
p = theano.tensor.vector()
Hp = theano.tensor.Rop(grad,T,p)
d2f_times_p = theano.function([R,T,h,p],Hp),
```

which can then be evaluated using some vector $\mathbf{p}$ with

```
p_rand = np.random.rand(prod_m)
d2f_times_p(R_rand, T_rand, h_rand, p_rand).
```

Note that this will compute the *exact Hessian* as opposed to the Gauss-Newton approximation discussed previously in Section 2.5.3. If the Gauss-Newton Hessian approximation is desired, e.g., since its quadratic form guarantees a descent direction in Newton's method (Section 2.5.3), it can also be computed automatically with

```
drr_times_p = theano.tensor.dot(theano.tensor.Rop(r, y, p),r)
Hp_GN = theano.tensor.grad(drr_times_p, T,
                                    consider_constant=[drr_times_p])
d2f_times_p_GN = theano.function([R,T,h,p],Hp_GN).
```

This computes $(\mathrm{d}r\,\mathbf{p})^\top r = (r^\top \mathrm{d}r\,\mathbf{p})^\top$ as given in (2.31). Then, this expression is derived again, but considering $\mathrm{d}r$ a constant, such that we derive an expression for $(\mathrm{d}r^\top \mathrm{d}r)\,\mathbf{p}$ as in (2.32), i.e., the matrix-vector multiplication with the Gauss-Newton Hessian approximation.

4.5.6. Further remarks

For GPU computation, the `THEANO_FLAGS` environment variable needs to be set, containing the `device=cuda0` parameter. When this option is set, all suitable computations are automatically performed on the first GPU device supporting NVIDIA CUDA.

As the double precision floating point performance of current GPUs is much lower than their single precision performance, it is recommended to additionally set `floatX=float32`. The `floatX` type is used as a placeholder in all Theano tensor variable floating point computations and is set to single precision this way. However, this will generally lead to reduced computational precision, which may influence convergence in the optimization scheme.

Since the main benefit of Theano lies in its automatic derivative computations, we found that not much can be gained by also implementing the optimization algorithm and multi-level scheme in Theano. Instead, we utilized the Python `ctypes` library to integrate the Theano objective function and its derivatives into our C++ implementation. The `ctypes` library allows to create C/C++ compatible data types from Python objects. We used this to make the compiled Theano functions available from within C/C++ for function value and derivative computations. This allows to seamlessly switch between the matrix-free implementations from Chapter 3 and the Theano implementation for comparison and is utilized in Section 5.4.

4.6. Runtime comparison

We implemented the registration objective function as discretized in Section 2.4 with the SSD and NGF distance measures as well as the curvature regularizer, including grid conversion and linear image interpolation. For the evaluation we used Python 3.6 with Theano 0.9 on a 12-core Intel Xeon E5-2630v2 workstation with a GeForce GTX 980 graphics card running Ubuntu Linux 16.04, as also used in Chapter 5.

All evaluations were averaged over 30 tests and achieved numerically identical results in comparison with the matrix-free implementation. The measured runtimes for function

	Method	Compile	Unpickle	Runtime
CPU	D^{SSD}	2.0	0.5	1.4
	∇D^{SSD}	5.1	0.8	3.1
	$\nabla^2 D^{\mathrm{SSD}}\mathbf{p}$	100.9	2.0	6.0
	$H^{\mathrm{SSD}}\mathbf{p}$	41.8	1.5	6.7
GPU	D^{SSD}	13.7	7.3	0.8
	∇D^{SSD}	21.8	16.2	2.1
	$\nabla^2 D^{\mathrm{SSD}}\mathbf{p}$	48.8	32.2	2.8
	$H^{\mathrm{SSD}}\mathbf{p}$	39.4	25.9	2.8

Table 4.1.: Evaluation of compile times and runtimes (in seconds) for the SSD distance measure and automatically computed derivatives in Theano for $d = 3$ with $m = (128, 128, 128)$ and $m^{\mathrm{y}} = (33, 33, 33)$; $\nabla^2 D^{\mathrm{SSD}}$ denotes the exact Hessian, while H^{SSD} is the Gauss-Newton approximation. While compiling each function is relatively slow, loading compiled functions from disk ("unpickle") can be considerably faster. By executing the functions on the GPU, Theano achieves speedups of up to 2.4.

compilation and evaluation for NGF, SSD and the curvature regularizer are given in Table 4.2, Table 4.1 and Table 4.3. Measurements were performed on CPU and GPU for the function value as well as for the derivatives. The evaluations on GPU were performed with single precision performance and the `gpuarray` back-end.

4.6.1. Function compilation

We evaluated different scenarios for the function compilation. First, we performed a regular function compilation as shown in Listing 1. Note, that since Theano keeps a compile cache of previously utilized C++ operators, the initial function compilation can take slightly longer than the reported values. Second, as discussed in the previous section, we used the Python `pickle` library to store compiled functions to disk. As recommended by Theano, we utilized the faster `cPickle` equivalent. Then, we measured the time that is required to unpickle (i.e., load) a stored function again. This scenario is especially interesting when integrating a stable algorithm into end-user software in order to minimize compilation times. While some parts of the computational graph might still be re-optimized upon unpickling, generally this decreases runtime. Note, that since the functions were defined solely by using tensor variables as discussed in Section 4.5, the compile times are independent of the image and deformation size.

Comparing the compile times for the function value and the derivatives on the CPU, it can be observed that for the distance measures in Table 4.1 and Table 4.2, the gradient function compilation is only slightly slower than the function value compilation. The Hessian computations, however, are more expensive, while generally the compilation of the exact Hessian takes more time than the compilation of the Gauss-Newton approximation. Especially for the SSD, the creation of the exact Hessian and the Gauss-Newton approximation are slower by factors of 51.3 and 21.2, respectively. In comparison, the

	Method	Compile	Unpickle	Runtime
CPU	D^{NGF}	3.5	0.6	1.5
	∇D^{NGF}	7.4	1.2	3.6
	$\nabla^2 D^{\mathrm{NGF}}\mathbf{p}$	38.7	3.1	7.9
	$H^{\mathrm{NGF}}\mathbf{p}$	19.1	2.3	7.9
GPU	D^{NGF}	40.8	38.1	0.8
	∇D^{NGF}	69.2	59.1	2.2
	$\nabla^2 D^{\mathrm{NGF}}\mathbf{p}$	133.3	95.1	2.8
	$H^{\mathrm{NGF}}\mathbf{p}$	92.4	73.0	2.8

Table 4.2.: Evaluation of compile times and runtimes (in seconds) for the NGF distance measure in Theano, see Table 4.1 for details. On the GPU, runtime speedups of up to 2.82 are achieved.

	Method	Compile	Unpickle	Runtime
CPU	S	0.99	0.25	0.0025
	∇S	5.78	0.88	0.0072
	$\nabla^2 S\mathbf{p}$	5.99	0.88	0.0071
GPU	S	7.70	0.42	0.0139
	∇S	30.40	1.27	0.0046
	$\nabla^2 S\mathbf{p}$	7.88	1.27	0.0041

Table 4.3.: Evaluation of compile times and runtimes (in seconds) for the curvature regularizer and automatically computed derivatives, implemented in Theano for $d = 3$ with $m^{\mathrm{y}} = (33, 33, 33)$, see Table 4.1 for further descriptions. On the GPU, except for the function value, runtime speedups of up to 1.71 are achieved.

same computations for the NGF are only slower by a factor of 11.0 and 5.44. This is likely caused by different graph optimizations and use of different compiled functions, also depending on the complexity of the functions. As can be seen, more complex derivatives require longer compilation times. Since the Gauss-Newton approximation of the Hessian discards certain second order terms, this also reduces the compilation time.

For the curvature regularizer in Table 4.3, this is different. While the gradient function compilation is slower by a factor of 5.81, compiling the Hessian function takes almost the same time as the gradient. This is expected, since the gradient computation and Hessian-vector multiplication are closely related as discussed in Section 3.4.2 and thus contain the same computations.

Function compilation on the GPU takes longer than on the CPU, except for the SSD Hessians. Thus, while the obtained GPU-based functions potentially have faster runtimes, the higher compilation times can diminish this benefit, depending on the number of subsequent function evaluations.

Pickling and unpickling the compiled function can in some cases largely reduce the runtime. However, there is only a smaller reduction in runtime for the distance measure function value and derivatives on the GPU. This indicates that a larger number of re-optimizations need to be peformed on this platform after loading the function from disk. Generally, each function only occupies a couple of megabytes when stored on disk, such that pickling the compiled functions is a valuable alternative, especially, when the function implementation does not change often.

4.6.2. Function evaluation

In comparison to the compile times, the runtimes for function evaluation depend on the image and deformation dimensions (rightmost column in Table 4.1, Table 4.2 and Table 4.3). In our experiments, we used image sizes of $m = (128, 128, 128)$ with a deformation size of $m^{\mathrm{y}} = (33, 33, 33)$, which are common medium-size resolutions in image registration problems.

For all functions on CPU and GPU the function value computations are faster than the gradient computations by factors ranging from 2.28 to 3.04. For the Hessian, these factors range from 3.42 to 5.20, except for the curvature regularizer, where as discussed before, Hessian-vector multiplication and gradient are very similar. Additionally, in contrast to the compilation times, exact Hessian and Gauss-Newton approximation exhibit almost identical runtimes.

On the CPU, unpickling is very fast and does not pose a major runtime penalty. As the unpickle times on GPU are much longer for the distance measures, the use of GPU functions is only beneficial when a large number of function evaluations is expected.

The speedup factors gained by the computation on GPU range from 1.57 for the curvature gradient to 2.81 for the NGF Gauss-Newton Hessian, with an exception of the curvature function value, which is slower on the GPU. It can be observed that more complex functions obtain a higher speedup on the GPU, possibly utilizing the GPU more efficiently. Our analysis showed that the slower curvature function value most likely results from a specific tensor concatenation operator on the GPU, utilized for the boundary conditions. Re-formulating the curvature implementation to avoid that specific operator might improve the runtime, but may, however, on the contrary slow down the CPU implementation.

Further runtime comparisons between the Theano implementations and matrix-free as well as matrix-based methods are performed in Chapter 5.

4.7. Summary

In this section, we presented an implementation of the registration objective function and derivatives, based on an automatic differentiation framework. We implemented the SSD and NGF distance measures as well as the curvature regularizer, including image interpolation and grid conversion in the Python-based framework Theano, enabling automatic computation of gradient and Hessian functions. Furthermore, Theano allows to automatically perform all computations on the GPU without further user effort.

The Theano framework requires an initial compilation step for each function, optimizing the computational graph, which can take up a considerable amount of the overall runtime. In almost all cases, loading compiled functions decreases the runtime of the compilation step. This makes the automatically determined derivatives also feasible for application in end-user software.

An execution of the implemented functions on the GPU achieved speedup factors of up to 2.8. While this speedup can be obtained without manual intervention, the distance measure computations on GPU require longer times for loading the compiled functions from disk than the CPU versions, possibly caused by re-optimizations performed by Theano during load. If only few iterations are performed in the registration, this can diminish the runtime benefit of the GPU execution.

In summary, automatic differentiation of the objective function constitutes a very flexible strategy, which allows for rapid evaluation of new models and ideas. In Section 5.3 and Section 5.4, we will investigate how this flexible automatic approach compares to the hand-optimized approach proposed in Chapter 3.

5 Experimental results

In this chapter, we verify the analyses of the matrix-free computations by evaluating and comparing different aspects of the derived algorithm. Previously, we showed that matrix-free methods are element-wise parallelizable. Now, we will investigate whether this also results in corresponding speedups in practice. Furthermore, the matrix-free methods promise a largely reduced memory consumption, as no intermediate results are stored. In the following, we will measure the memory requirements to verify these claims. Additionally, we compare the matrix-free computations (MFC) to matrix-based implementations as well as the Theano-based algorithm (THEANO) on CPU and GPU.

We proceed from analyzing the smallest components of the objective function, such as distance measure and regularizer derivatives, to an evaluation of the complete objective function derivatives. Finally, we characterize different implementations and algorithms for the full multi-level registration in terms of runtime and memory usage.

For this, initially, in Section 5.1, we analyze the parallel scalability of the matrix-free derivative computations. Additionally, different variants of the matrix-free computations, as discussed in Section 3.7, are compared in Section 5.2 to determine the effects of the proposed trade-offs between recalculations and memory use.

Afterwards in Section 5.3, the runtime of the complete objective function derivatives is compared for several algorithm implementations. Besides the Theano-based implementation using automatic differentiation, the C++ implementation of the matrix-free algorithm is compared to a matrix-based C++ implementation (MBC) and a publicly available MATLAB implementation using the FAIR toolbox [Mod09] (FAIR).

Additionally, we present a real-world registration dataset and perform a full multi-level registration with all algorithms in Section 5.4. Besides runtime, we also compare peak memory usage for all algorithms as well as for MFC implementation alternatives. Finally, in Section 5.5, we present a GPU-based implementation of the matrix-free computations using NVIDIA CUDA and compare the runtime and memory requirements to the CPU-based implementation.

Benchmark environment. Unless otherwise stated, all experiments in this chapter were performed on a 12-core dual processor workstation with two Intel Xeon E5-2630v2 CPUs running at 2.6 GHz, with 32 GB RAM and a NVIDIA GeForce GTX 980 graphics card using Ubuntu Linux 16.04. To minimize the influence of unrelated processes, the workstation was run without a graphical user interface, all unneeded processes were terminated and all tests were run with highest process priority. Furthermore, we disabled memory swapping to obtain meaningful results for runtime and memory consumption.

The C++ implementations were compiled with gcc 5.4.0 using OpenMP and AVX support with highest optimization (`-O3`), while the FAIR evaluations were performed with MATLAB R2017b. For the Theano evaluations, we used Theano 0.9.0 with Python 3.6.0.

Acknowledgments and related publications. We presented preliminary results of the matrix-free computations performed in this chapter in [KRDL18*; KDHP15*; KR14*]. We especially thank Daniel Budelmann and Martin Meike for their contribution to the CUDA implementation of the matrix-free registration algorithm. A preliminary version of the CUDA-based registration was presented in the thesis [Mei16].

5.1. Scalability on the CPU

In this section, we evaluate the scalability of the matrix-free computations, i.e., we analyze the runtime of a fixed problem size in relation to the number of utilized computational cores. For this, we measured the runtime of all objective function components that were discussed in Chapter 3. Besides the distance measures and regularizer with their function value, gradient and Hessian-vector multiplication computations, as given in Section 3.2, Section 3.3 and Section 3.4, respectively, we also investigated the grid conversion and its corresponding transposed operator from Section 3.5. All methods were implemented in C++, fully parallelized with OpenMP and further accelerated using vectorized computations with AVX as discussed in Section 3.8.1.

We evaluated two problem sizes, corresponding to typical image registration scenarios: Small problems with an image size of $m = (64, 64, 64)$ and a deformation size of $m^{\mathrm{y}} = (17, 17, 17)$, which typically occur in coarser levels of a multi-level computation, and large problems with an image size of $m = (512, 512, 512)$, which are, e.g., found in typical clinical CT images, and a deformation size of $m^{\mathrm{y}} = (129, 129, 129)$.

For evaluation we used randomly generated image data. Starting with a serial execution on a single core, for each test an increasing number of computational cores were activated and runtimes were measured. The obtained runtimes were averaged over 30 evaluations with identical data and initialization in order to reduce measurement noise.

The results for serial and parallel computation as well as the resulting speedup factors are shown in Table 5.1. Additionally, detailed speedup factors are visualized in Figure 5.1 for the small images and in Figure 5.2 for the large images.

5.1.1. Small images

For the evaluation with small images in Figure 5.1, we obtained speedup factors ranging from 8.93 to 10.5 for the distance measures and their derivatives as well as the grid conversion from image grid to deformation grid. For up to eight cores, the computation

Method	Small images			Large images		
	Serial (ms)	Parallel (ms)	Speedup	Serial (s)	Parallel (s)	Speedup
D^{NGF}	41.41	3.93	10.5	27.27	2.02	13.5
$\frac{\partial D^{\text{NGF}}}{\partial P}$	45.82	5.13	8.9	32.03	2.69	11.9
$\hat{H}^{\text{NGF}}\hat{\mathbf{p}}$	250.47	23.02	10.9	146.80	12.14	12.1
D^{SSD}	6.07	0.59	10.3	3.80	0.32	11.8
$\frac{\partial D^{\text{SSD}}}{\partial P}$	6.65	0.66	10.1	5.05	0.46	11.0
$\hat{H}^{\text{SSD}}\hat{\mathbf{p}}$	7.16	0.69	10.4	5.65	0.54	10.5
$P\mathbf{y}$	12.01	1.15	10.4	6.71	0.56	11.9
$P^\top\hat{\mathbf{y}}$	8.80	2.24	3.9	4.67	0.45	10.4
S	0.10	0.03	3.9	0.05	0.01	3.5
∇S	0.18	0.04	4.6	0.10	0.02	4.0
$\nabla^2 S\mathbf{p}$	0.18	0.04	4.5	0.10	0.03	3.9

Table 5.1.: Scaling behavior of the matrix-free computations on a 12-core workstation compared to a single-core execution. Two different image resolutions are evaluated for $d = 3$: Small images with $m = (64, 64, 64)$ and $m^{\mathrm{y}} = (17, 17, 17)$, and large images with $m = (512, 512, 512)$ and $m^{\mathrm{y}} = (129, 129, 129)$. Distance measure and grid conversion computations scale approximately linear, for the small images only limited by the number of available parallel tasks. Due to memory-bound computations, speedup factors of the curvature regularizer are lower.

scales approximately linear. After this, similar performance is observed for eight, nine and ten cores and, after another increase, for eleven and twelve cores. This behavior is not directly related to algorithm scalability and can partly be explained by the ratio of parallel tasks to computational cores. As shown in Section 3.8.1, on the CPU we only parallelize the outer loop of the computation, which in this case iterates over the z-dimension. This is recommended in OpenMP and typically represents the best trade-off between number of parallel tasks and administration overhead. Thus, 64 parallel tasks are available here. When distributing these tasks to the computational cores, they can rarely be divided without a remainder. Since in our computations, all tasks need approximately the same times to execute, these remaining tasks are executed while the other computational cores are idle, consequently increasing runtime. Theoretically the resulting speedup can be modeled as

$$s^{n_{\text{tasks}}}(n_{\text{cores}}) := \frac{n_{\text{tasks}}}{\left\lceil \frac{n_{\text{tasks}}}{n_{\text{cores}}} \right\rceil}, \tag{5.1}$$

where n_{tasks} and n_{cores} denote the number of available parallel tasks and computational cores, respectively. In this case, we have especially $s^{64}(8) = s^{64}(9) = 8$ and $s^{64}(11) = s^{64}(12) \approx 10.7$, which explains the areas of unchanged speedup in Figure 5.1 as well as the limited maximum speedup. However, while $s^{64}(10) \approx 9.1$, we only achieve an approximately 8-fold speedup for ten cores. As the measurements closely follow (5.1)

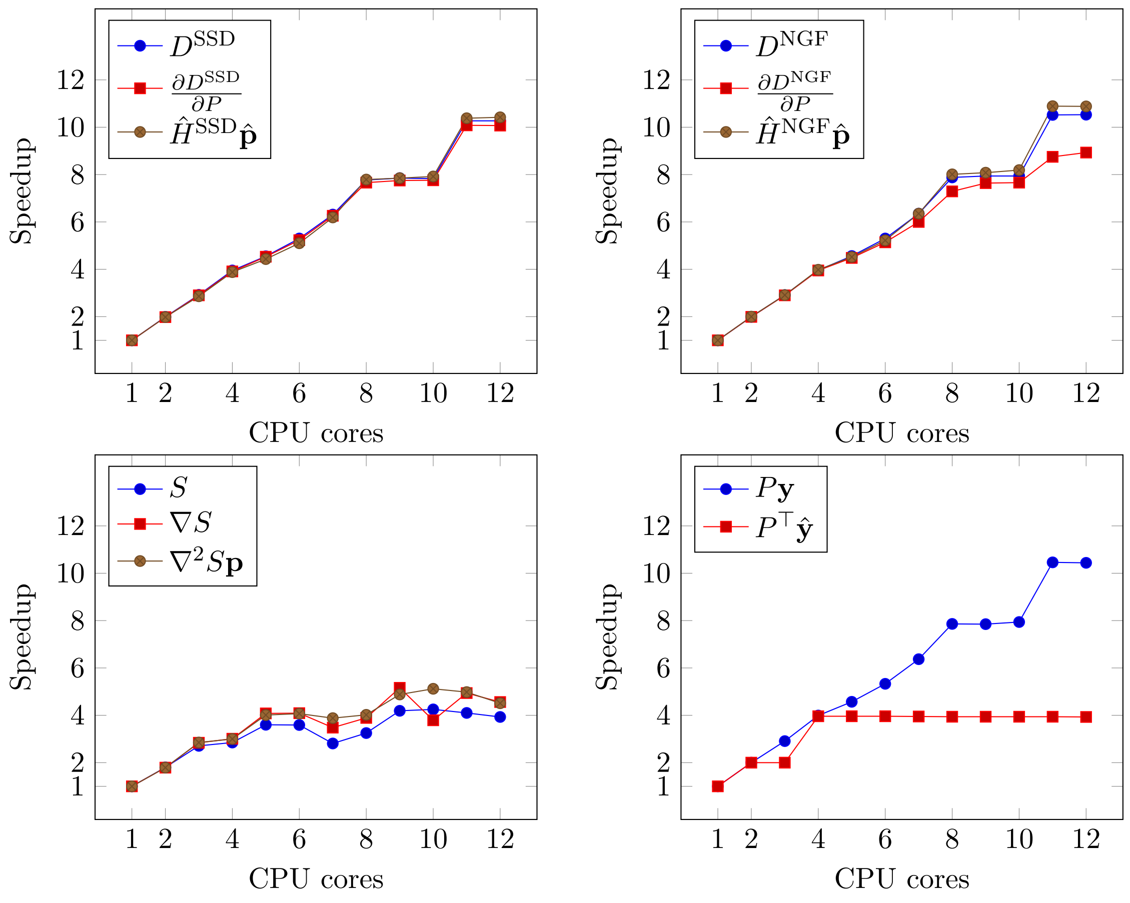


Figure 5.1.: Speedup for parallel computations of matrix-free functions using a different number of CPU cores for small images with $m = (64, 64, 64)$ and $m^y = (17, 17, 17)$. While the distance measure computations exhibit the desired linear scalability, for larger numbers of CPU cores the speedup is limited by the ratio of parallel tasks to cores. The transposed grid conversion operator $P^\top\hat{\mathbf{y}}$ is limited by the number of available tasks due to the used red-black scheme, while the curvature regularizer is limited by memory bandwidth.

for all other numbers of parallel cores, the speedup for this particular case must be additionally limited by other aspects, such as additional synchronization overhead.

Additionally, the transposed grid conversion operator $P^\top\hat{\mathbf{y}}$ only exhibits a speedup factor of 3.93. As can be seen in Figure 5.1, after an initial increase, the speedup remains constant after four cores. This can be explained with the limited number of computational tasks in connection with the used red-black scheme, as described in Section 3.5.2. Since $P^\top\hat{\mathbf{y}}$ operates on the deformation grid, only 17 parallel tasks are available in this example. Due to the red-black scheme, only half of them can be executed simultaneously. Considering (5.1), this results in a maximum speedup of 4.5 for up to eight cores. For higher numbers of computational cores no further speedup is observed since the parallelization overhead is likely too large with only one thread per core. This is confirmed by the measurements using the large images with a maximum speedup of 10.4.

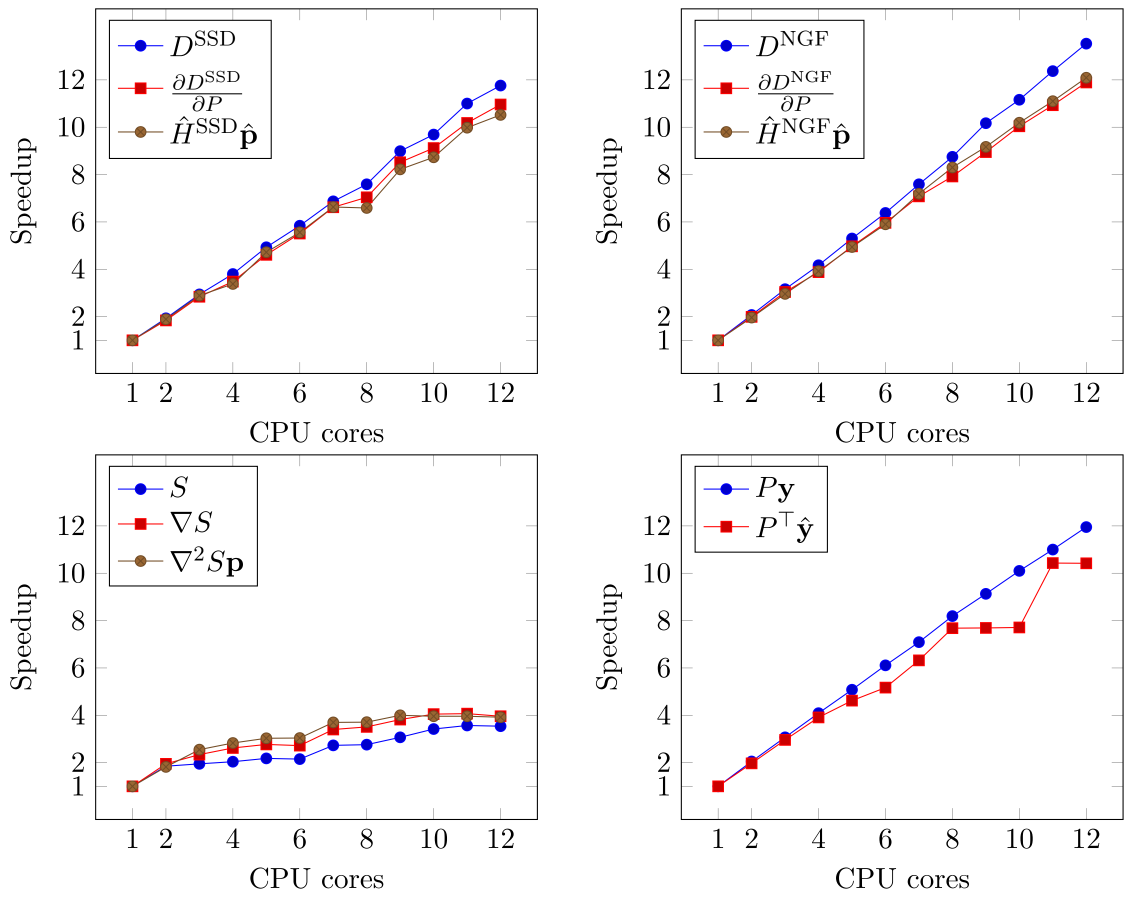


Figure 5.2.: Speedup for parallel computations of matrix-free functions using a varying number of CPU cores, in comparison to single-core execution. Distance measure, regularizer and grid conversion functions are evaluated for large images with $m = (512, 512, 512)$ and $m^{\mathrm{y}} = (129, 129, 129)$. Distance measure computations and grid conversion operators exhibit the desired approximately linear scalability. The curvature regularizer computations are again limited by memory transfer bandwidth.

In comparison to the distance measures and grid conversion operators, the scaling behavior of the curvature regularizer is limited. We measured maximum speedup factors ranging from 3.57 to 5.12 for all image sizes. While an execution on three cores still yields a speedup of about three, for an execution on more cores only a small additional increase can be observed. Here, the computation is most likely bound by memory transfer speed: Compared to the distance measures, only a small number of computational operations is performed on every deformation element. However, still the complete deformation has to be accessed. This results in a low arithmetic intensity (floating point operations per byte of data loaded or stored, FLOPS/byte) and thus a limited speedup when adding more computational cores, since the overall runtime is limited by the maximum memory transfer bandwidth. However, as the absolute runtimes of the curvature regularizer are several magnitudes lower than those of grid conversion and distance measures and their derivatives, this has only limited effect on the overall runtimes.

5.1.2. Large images

As can be seen from Figure 5.2, for the evaluation with large images all computations for SSD and NGF achieve desirable results with approximately linear scaling. In comparison with a serial computation on one core, we obtain speedup factors from 10.5 to 13.5 for a parallel computation on twelve cores. Similarly, the grid conversion operators achieve speedup factors of 11.9 and 10.4. The transposed operator $P^\top \hat{\mathbf{y}}$ exhibits an almost linear scaling for one to eight cores, while the performance does not increase from eight to nine and ten, and from eleven to twelve cores. This behavior is again related to the number of available tasks in relation to the number of computational cores as discussed in Section 5.1.1.

Summary. The relevant matrix-free computations for distance measures and grid conversion exhibit approximately linear scaling behavior on larger images. On smaller images, the scalability can be limited by different factors, such as the ratio between parallel tasks and CPU cores or a limited total number of available parallel tasks. Here, more fine-grained parallelism would not be beneficial, as the smaller image sizes in a multi-level scheme make up only a small fraction of the overall computation time. Similarly, while the scaling behavior of the curvature regularizer is limited by memory transfer bandwidth, its execution is several orders of magnitude faster compared to the other functions and should thus not impede the overall execution time of the final registration algorithm.

If a sufficient number of parallel tasks is available, we have shown that a substantial speedup can be expected when utilizing additional computational cores, which is an important requirement for a fast algorithm given the steadily increasing core counts in modern multi-core CPUs and ensures further speedups on future platforms.

5.2. Selective precomputation

In Section 3.7, we discussed different implementation alternatives for the matrix-free methods. A main principle of the matrix-free algorithm is to perform recalculations instead of storing precomputed intermediate values to memory. However, some intermediate results require especially high numbers of recalculations. Thus, it can be beneficial to selectively precompute some intermediate values.

In this section, we will evaluate for $d = 3$ if these precomputations, which come at the cost of moderate and closely controlled additional memory requirements, result in a reduced runtime. In Table 5.2, we show the runtimes of three different matrix-free implementations for distance measures: (1) precomputing both deformed template image $T(\hat{\mathbf{y}})$ and computing the grid conversions $P\mathbf{y} = \hat{\mathbf{y}}$ and $P^\top \hat{\mathbf{g}}$ in separate steps as described in Section 3.5.3, (2) precomputing only $T(\hat{\mathbf{y}})$, and (3) calculating everything on the fly. Furthermore, we evaluate two alternatives for the curvature regularizer: Precomputing the first application of the Laplace operator or calculating everything on the fly. In the following, we discuss the results for each individual method in detail.

Precomp.	Function value			Gradient			Hessian		
$T(\hat{\mathbf{y}})$	●	●	○	●	●	○	●	●	○
$\hat{\mathbf{y}}$	●	○	○	●	○	○	●	○	○
SSD (s)	0.9	0.7	0.6	1.7	1.2	1.2	2.0	1.8	1.9
NGF (s)	2.7	3.8	3.7	4.0	3.9	18.3	13.6	29.2	29.1

Table 5.2.: Impact of selective precomputation on distance measure runtime; ●: values precomputed and stored, ○: values recalculated on the fly. Parallel computations, evaluated for images with size $m = (512, 512, 512)$ and deformation size $m^{\mathbf{y}} = (129, 129, 129)$. For the gradient, the precomputation of $\hat{\mathbf{y}}$ includes the gradient on the image grid $\frac{\partial D}{\partial P}$, for the Gauss-Newton Hessian approximation it includes $\hat{\mathbf{p}}, \hat{\mathbf{q}}$, as described in Section 3.5.3. For SSD, there is no recalculation overhead, thus variants which compute everything on the fly are fastest. For NGF, gradient computations only precomputing the deformed template image $T(\hat{\mathbf{y}})$ are fastest, while for function value and Hessian, precomputing both deformed template and deformed grids yields the lowest runtimes.

Sum of squared differences. In theory, matrix-free SSD computations can be performed without recalculations as shown in Table 3.2. This is confirmed by the experimental results in Table 5.2. No runtime benefit can be observed by precomputing the deformed template image $T(\hat{\mathbf{y}})$ or the deformed grids, since the number of element computations is already minimal. For the function value and gradient computations, implementations computing everything on the fly without precomputations achieve the lowest runtimes. For the Hessian-vector multiplications, similar runtimes are observed with and without precomputations. We therefore conclude that for SSD computations the implementations without precomputations can always be preferred, as they exhibit both the fastest runtimes and the lowest memory usage.

Normalized gradient fields. For the NGF distance measure the results in Table 5.2 are qualitatively different. When computing the function value, the fastest runtime is achieved when precomputing both the deformed template image $T(\hat{\mathbf{y}})$ and the deformed grids $P\mathbf{y}$ and $P^{\top}\hat{\mathbf{y}}$, with an decrease in runtime by a factor of 1.4. For the gradient, no relevant speedup is achieved by precomputing the deformed grids. However, also computing the deformed template image on the fly drastically slows down the computations by a factor of 4.6. The Hessian-vector multiplication again is similar to the function value computations. The fastest runtime is achieved by precomputing both the deformed template image and the deformed grids, while the two other alternatives achieve similar runtimes which are slower by a factor of 2.1.

Thus, when aiming for the fastest runtimes, precomputing at least the deformed template image is recommended for NGF. When utilizing the L-BFGS optimization scheme where the Gauss-Newton Hessian-vector multiplication is not required, computing the grid conversion steps on the fly results only in a moderately slower computation for the function value and the fastest possible gradient computations. However, when the Hessian approximation is required for Gauss-Newton optimization, the deformed grids as well as the deformed template image should be precomputed when fastest runtimes are desired. This, however, requires memory for storing $10\bar{m}$ elements of $\hat{\mathbf{y}}$, $\hat{\mathbf{p}}$, $\hat{\mathbf{q}}$ and $T(\hat{\mathbf{y}})$, as shown in Table 3.4, amounting to 10 GB for an image size of 512^3 voxels.

Curvature regularizer. We proposed implementation alternatives for the curvature regularizer in Table 3.5, suggesting to precompute the result of the matrix multiplication $A\mathbf{u}$ for the computation of $A^\top A\mathbf{u}$. For both alternatives, precomputing $A\mathbf{u}$ and on-the-fly computation of $A^\top A\mathbf{u}$, we obtained runtimes of 0.025 s with precomputations and 0.027 s when computing everything on the fly, computed for $d = 3$ with a deformation size of $m^{\mathrm{y}} = (129, 129, 129)$. Thus, given difference in the range of milliseconds, no relevant difference in runtimes for computations with or without precomputations can be observed for the curvature regularizer.

While for the SSD this was expected since the number of computations is not reduced by the precomputations, this is not the case for the curvature regularizer, as can be seen in Table 3.5. The similar runtimes for both alternatives can be explained by the fact that the curvature computation speed is mainly limited by memory bandwidth as discussed in Section 5.1. Thus, further reducing the number of computations does not result in an additional runtime benefit. We therefore conclude that for the curvature regularizer the on-the-fly variant can always be preferred.

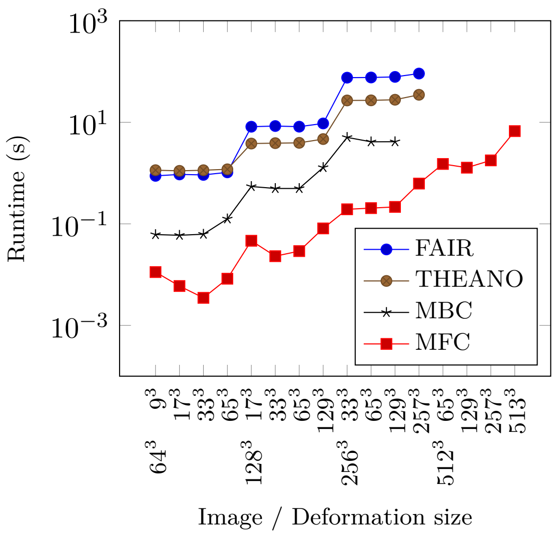
5.3. Objective function derivative runtime

We now evaluate the matrix-free approach in the full registration scheme and compare it to existing algorithms.

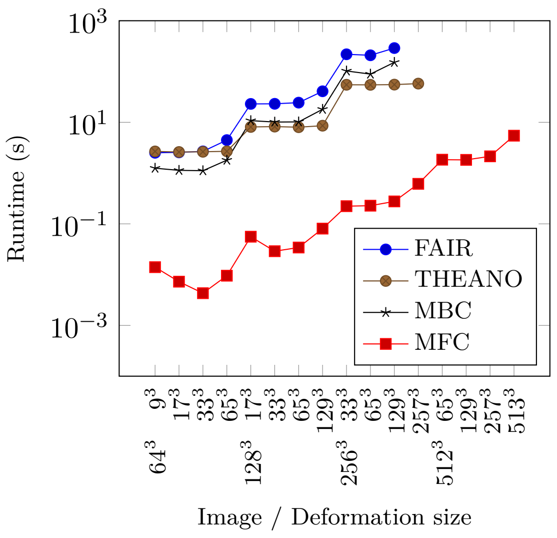
Algorithms. We compare our C++-based matrix-free computations (MFC) to three different implementations. Two implementations are matrix-based: the publicly available FAIR-toolbox [Mod09], implemented in MATLAB, and a matrix-based C++ implementation (MBC). Additionally, we make comparisons with the Python-based Theano implementation (THEANO) using automatically computed derivatives (Chapter 4).

While FAIR aims at a research and teaching audience, the C++-based implementation of MBC targets performance, while still being matrix-based. Comparing MFC with these implementations gives an impression of how much speedup can be gained by deriving a matrix-free implementation from already optimized research code. In contrast to this, the implementation of THEANO requires even less manual intervention than the matrix-based variants and therefore represents an alternative to the matrix-based implementations also in the research context.

Both C++ implementations, MBC and MFC, were optimized manually for fair comparison. In MBC, critical components such as distance measure computations, linear interpolation and sparse matrix multiplications were parallelized using OpenMP. In MFC, the objective function derivative computations were fully parallelized with OpenMP as described in Section 3.8.1. Furthermore, all matrix-free computations were accelerated by using vectorized computations with the Advanced Vector instructions (AVX) as also described in Section 3.8.1.



(a) Gradient runtime



(b) Hessian-vector multiplication runtime

Figure 5.3.: Runtime evaluation of SSD objective function derivatives for parallel execution on CPU for different image and deformation resolutions, as shown in Table 5.3 and Table 5.4, for four different algorithms (logarithmic scale), (a): gradient runtime, (b): Hessian-vector multiplication runtime. FAIR: MATLAB matrix-based, THEANO: Python/Theano with automatic differentiation, MBC: C++ matrix-based, MFC: C++ matrix-free computations. For THEANO, the runtimes include the unpickle times from Table 4.1. MFC is up to four magnitudes faster than the other algorithms and enables the computation of higher resolutions; THEANO yields competitive performance with FAIR (gradient) and even the manually optimized MBC (Hessian).

For evaluation of the MFC runtimes, we used the implementations that achieved the fastest runtimes in Section 5.2. Additional comparisons for different variants of MFC can be found in Section 5.4.

In FAIR, the fastest available components were chosen, using MATLAB MEX versions where available, also including OpenMP parallelized variants of linear interpolation and curvature regularizer. Additionally, we enabled FAIR's matrix-free implementation of the curvature regularizer.

Experimental setup. We first compare the runtimes of a single evaluation of the objective function derivatives before advancing to the full multi-level registration in Section 5.4.

For both SSD and NGF distance measures, parallel runtimes for gradient computations as well as a Hessian-vector multiplication with the Gauss-Newton Hessian approximation were measured with all four algorithms.

In order to determine the effect of different image and deformation resolutions on the runtime, we evaluated image sizes from 64^3 to 512^3 voxels with varying deformation resolutions. An image size of 512^3 voxels commonly occurs, e.g., in CT images as utilized in Section 5.4, while the coarser resolutions are then computed in the corresponding multi-level computations. Images of these smaller sizes are also common when processing other imaging modalities such as MRI, specific organs or regions of interest.

5.3.1. Sum of squared differences

Results of the runtime measurements using the registration objective function with SSD are visualized in Figure 5.3. The runtimes and speedup relative to the matrix-free algorithm are shown in Table 5.3 for gradient computations and in Table 5.4 for the Hessian matrix-vector multiplication.

As can be seen in Figure 5.3, MFC achieves the fastest runtimes. For the gradient, MBC is faster than THEANO and FAIR; for the Hessian-vector multiplication, THEANO is faster than MBC for image sizes larger than 64^3 voxels.

In the rightmost three columns of Table 5.3, speedup factors for gradient computations in comparison with MFC are shown. In comparison to FAIR, MFC achieves a speedup of about two orders of magnitude, ranging from 120 to 370. Compared to MBC, the speedup is about one order of magnitude, ranging from 10 to 49.

The THEANO computations are slower than MBC but faster than FAIR. In comparison to THEANO, the matrix-free computations still achieve a speedup of up to two orders of magnitude, ranging from 56 to 330. Note that the THEANO runtimes include the time for loading (unpickling) the functions, as shown in Table 4.1.

Even larger speedups are achieved by MFC for the SSD Hessian-vector multiplication shown in Table 5.4. Here, the speedups in comparison to FAIR range from a factor of 350 to 1000. In comparison to MBC, the speedup factors are now also in the range of two orders of magnitude, ranging from 160 to 550. Here, the THEANO computations are not only faster than FAIR, but also faster than MBC for image sizes larger than 64^3 voxels. Additionally, the difference in runtime between MFC and THEANO increased, with MFC now being faster by a factor of 94 to 610.

For both gradient and Hessian-vector computations, MFC is the only algorithm that is able to compute results for an image size of 512^3 voxels. All other algorithms exceed the available 32 GB of memory. Additionally, for the Hessian-vector multiplication, FAIR and MBC run out of memory for an image resolution of 256^3 voxels with a deformation resolution of 257^3, while THEANO successfully computes a result. This indicates that THEANO exhibits a lower memory usage than FAIR and MBC, which will be investigated further in Section 5.4.2.

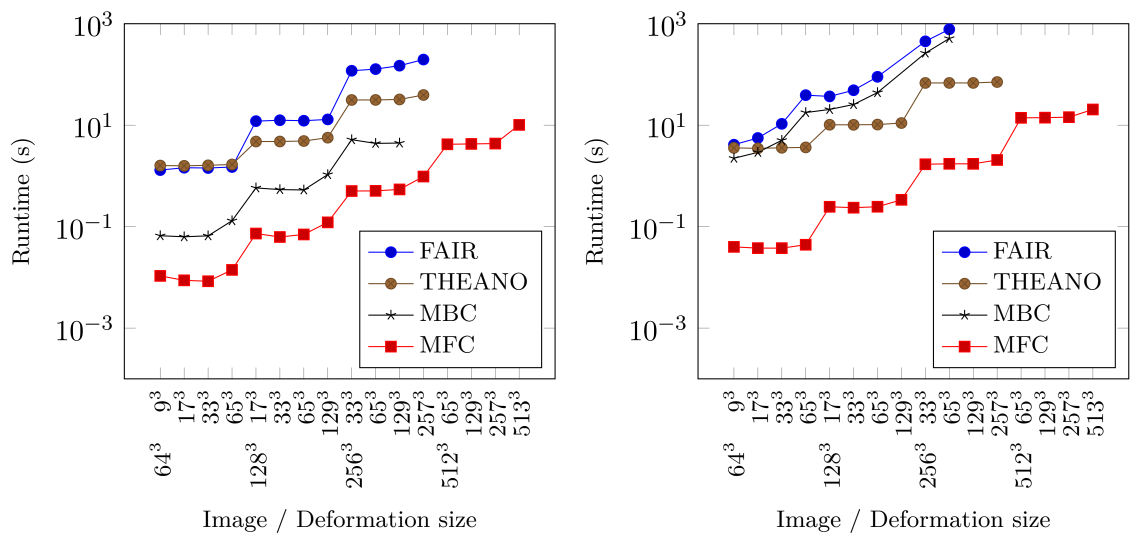
Another observation that can be made from Table 5.3 and Table 5.4 is the ratio of gradient to Hessian computations for each algorithm. While the MFC exhibits similar runtimes for both, the THEANO Hessian computations are slower by a factor of approximately two in comparison to the corresponding gradient compuations. For MBC and FAIR this factor is even larger, ranging from 2.7 to 4.5 for FAIR and 14 to 37 for MBC. This shows unused potential for optimizations, which is realized only by the manually derived MFC.

Size		Runtime (s)				Speedup		
m_i	m_i^{y}	FAIR	MBC	THEANO	MFC	MFC vs. FAIR	MFC vs. MBC	MFC vs. THEANO
512	513	*	*	*	6.707	–	–	–
	257	*	*	*	1.778	–	–	–
	129	*	*	*	1.277	–	–	–
256	257	91.2	*	34.8	0.621	150	–	56
	129	78.4	4.14	27.8	0.216	360	19	130
	65	76.3	4.13	27.1	0.204	370	20	130
128	129	9.5	1.29	4.7	0.081	120	16	57
	65	8.2	0.50	3.9	0.029	280	17	140
	33	8.4	0.50	3.9	0.023	370	22	170
64	65	1.0	0.13	1.2	0.008	120	15	140
	33	0.9	0.06	1.1	0.003	260	18	330
	17	0.9	0.06	1.1	0.006	160	10	190

Table 5.3.: Runtime evaluation of the SSD objective function gradient for different image and deformation resolutions for four algorithms. For abbreviations and details see Figure 5.3.

Size		Runtime (s)				Speedup		
m_i	m_i^{y}	FAIR	MBC	THEANO	MFC	MFC vs. FAIR	MFC vs. MBC	MFC vs. THEANO
512	513	*	*	*	5.448	–	–	–
	257	*	*	*	2.136	–	–	–
	129	*	*	*	1.818	–	–	–
256	257	*	*	57.8	0.614	–	–	94
	129	289.4	151.6	55.2	0.277	1000	550	200
	65	208.8	88.9	54.6	0.228	920	390	240
128	129	40.8	18.0	8.6	0.081	510	220	110
	65	24.4	10.2	8.0	0.034	710	300	230
	33	23.1	10.1	8.2	0.029	800	350	280
64	65	4.5	1.8	2.7	0.010	470	190	280
	33	2.7	1.1	2.6	0.004	620	260	610
	17	2.6	1.1	2.6	0.007	350	160	360

Table 5.4.: Runtime evaluation of the SSD objective function Hessian-vector multiplication for different image and deformation resolutions for four algorithms. For abbreviations and details see Figure 5.3.



(a) Gradient runtime

(b) Hessian-vector multiplication runtime

Figure 5.4.: Runtime evaluation of NGF objective function derivatives for different image and deformation resolutions, as shown in Table 5.5 and Table 5.6, for four different algorithms (logarithmic scale), (a): gradient runtime, (b): Hessian-vector multiplication runtime. FAIR: MATLAB matrix-based, THEANO: Python/Theano with automatic differentiation, MBC: C++ matrix-based, MFC: C++ matrix-free computations. MFC is up to three magnitudes faster than the other algorithms and enables the computation of higher resolutions. For the Hessian-vector multiplication, the automatically computed THEANO computations are even faster than the manually optimized MBC.

5.3.2. Normalized gradient fields

As can be seen in Figure 5.4, for the NGF distance measure runtime distributions for the four algorithms are similar to the previously discussed SSD results. MFC is the fastest algorithm for both gradient and Hessian-vector computations. While MBC is faster than THEANO for gradient computations, THEANO is faster for Hessian computations with image sizes larger than 64^3 voxels.

Detailed results and speedup factors are given in Table 5.5 for gradient computations and in Table 5.6 for the Hessian-vector multiplication. Similar to the SSD computations, computing the objective function gradient with MFC is about two orders of magnitude faster than with FAIR and about one order of magnitude faster than using MBC. In comparison to THEANO, MFC is faster by factors ranging from 40 to 190.

For the NGF Hessian-vector computations, MFC is again faster by about two orders of magnitude compared to MBC and FAIR, with speedup factors ranging from 77 to 890. In comparison to the SSD Hessian runtimes, THEANO computations are closer to computations with MFC. Here, speedup factors in comparison with MFC range from 33 to 95.

Again, the gradient for an image resolution of 512^3 can only be computed with MFC. Additionally, FAIR and MBC run out of memory for an image size of 256^3 voxels with deformation sizes of 257^3 and 129^3 as well as an image size of 128^3 voxels with a deformation size of 129^3. In comparison, THEANO is able to compute Hessian results for all resolutions, except with an image size of 512^3 voxels.

In contrast to the SSD derivative computations with MFC, which achieved similar runtimes for gradient and Hessian, the NGF Hessian computations with MFC are slower by a factor of 2 to 4.5 compared to the corresponding gradient computations. For FAIR and MBC these factors range from 3.7 to 26 and 46 to 140, respectively, which are also larger than for the SSD computations. This can be partly explained with the more involved computations of the NGF Hessian. However, similar to the SSD computations, THEANO NGF Hessian computations are slower only by a factor of approximately two. This shows that an automatically derived and optimized derivative calculation can potentially obtain computations with less overhead than manual optimization, especially for complicated computations such as the NGF Hessian. The absolute runtimes, however, still remain high in comparison to MFC, such that manual optimization of the derivatives still yields a large benefit in this case.

5.3.3. Theano on the GPU

The objective function evaluations in Section 4.6 indicate that a substantial speedup can be expected from performing the THEANO objective function computations on the GPU. However, as also stated in Section 4.6, creating the Theano functions or loading these from disk takes much longer than on the CPU. Therefore, unreasonably large runtimes are required for a single objective function derivative evaluation when including the unpickle time, as done for Theano on CPU in the previous sections. When running a full registration, however, this overhead is only required once when each function is loaded at the beginning of the registration. After this, each function can be evaluated arbitrarily often throughout the registration. Depending on the number of function evaluations the faster runtime might compensate for the initial overhead and result in a faster overall runtime. We will therefore evaluate the Theano GPU computations for a full multi-level registration in Section 5.4.

5.3.4. Summary

In this section, we evaluated the runtime for derivative computation of the full registration objective function for the distance measures SSD and NGF. In comparison to the matrix-based algorithms FAIR and MBC, the matrix-free computations are faster by several orders of magnitude in every case. Additionally, MFC is the only algorithm that is able to compute results for an image size of 512^3 voxels. Thus, besides a very fast execution, MFC additionally benefits from its efficient memory use.

While the THEANO computations show considerably slower runtimes than MFC, runtimes are similar to FAIR and MBC, especially for Hessian computations, where additionally higher resolutions can be computed.

Size		Runtime (s)				Speedup		
m_i	m_i^{y}	FAIR	MBC	THEANO	MFC	MFC vs. FAIR	MFC vs. MBC	MFC vs. THEANO
512	513	*	*	*	10.12	–	–	–
	257	*	*	*	4.34	–	–	–
	129	*	*	*	4.27	–	–	–
256	257	196.6	*	39.4	0.97	200	–	40
	129	148.8	4.44	32.1	0.54	270	8.2	59
	65	127.4	4.37	31.4	0.51	250	8.6	62
128	129	13.0	1.06	5.7	0.12	110	8.8	47
	65	12.2	0.53	4.9	0.07	170	7.5	69
	33	12.5	0.54	4.8	0.06	200	8.6	76
64	65	1.5	0.13	1.7	0.01	110	9.3	120
	33	1.4	0.07	1.6	0.01	170	7.9	190
	17	1.5	0.06	1.6	0.01	170	7.2	180

Table 5.5.: Runtime evaluation of the NGF objective function gradient for different image and deformation resolutions for four algorithms. For abbreviations and details see Figure 5.4.

Size		Runtime (s)				Speedup		
m_i	m_i^{y}	FAIR	MBC	THEANO	MFC	MFC vs. FAIR	MFC vs. MBC	MFC vs. THEANO
512	513	*	*	*	20.39	–	–	–
	257	*	*	*	14.38	–	–	–
	129	*	*	*	14.07	–	–	–
256	257	*	*	70.9	2.06	–	–	34
	129	*	*	68.0	1.73	–	–	39
	65	775.9	511.8	68.0	1.73	450	300	39
128	129	*	*	11.0	0.34	–	–	33
	65	89.8	43.9	10.3	0.25	360	180	42
	33	48.8	25.6	10.2	0.24	210	110	43
64	65	39.1	17.7	3.6	0.04	890	400	83
	33	10.6	5.0	3.6	0.04	280	130	95
	17	5.6	2.9	3.5	0.04	150	77	93

Table 5.6.: Runtime evaluation of the NGF objective function Hessian-vector multiplication for different image and deformation resolutions for four algorithms. For abbreviations and details see Figure 5.4.

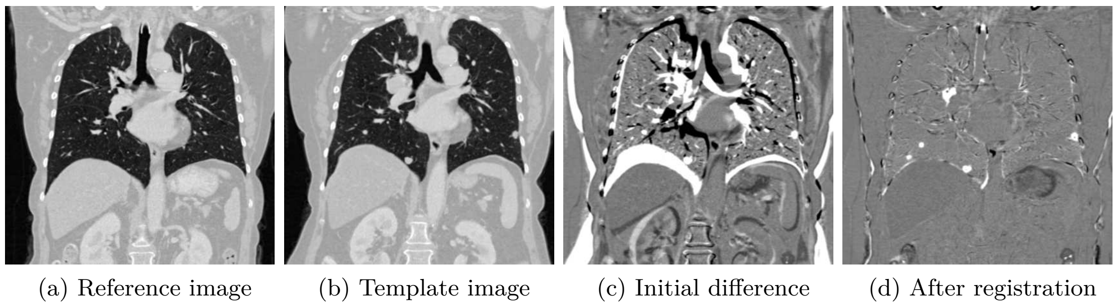


Figure 5.5.: Dataset for the full multi-level registration: Two three-dimensional thorax-abdomen CT scans of the same patient, acquired nine months apart, cropped to a size of $m = (512, 512, 512)$, (a): coronal slice of reference image, (b): coronal slice of template image, (c): difference image before registration, (d): difference image after registration for an exemplary registration with NGF. The difference image after registration highlights the areas of change (white spots in the lung) corresponding to tumor growth. Datasets courtesy of Radboud University Medical Center, Nijmegen, The Netherlands.

Thus, for computing objective function derivatives, both algorithms, MFC and THEANO, present valuable alternatives to matrix-based approaches. While MFC requires some effort in deriving and implementing the computations, the effort pays off and substantially faster runtimes are achieved. Additionally, higher resolutions can be processed. THEANO achieved runtimes similar to existing research codes, with minimal development effort and high flexibility and is thus especially suitable for model development and rapid prototyping.

While computing objective function derivatives represents a major component of the final registration algorithm, the total registration runtime and peak memory usage are ultimately the crucial factor when using a registration algorithm in practice. Therefore, in the next section, we will investigate the full multi-level registration for different algorithms with respect these aspects.

5.4. Multi-level registration

After characterizing the algorithms' performance on individual components of the objective function in the previous chapters, in this section, we evaluate the algorithms in a complete real-world problem from clinical practice: registering two CT images for follow-up diagnosis in oncology, shown in Figure 5.5. Both images are from the same patient and have been acquired nine months apart. A difference image can reveal regions of change for further investigation in order to assess tumor development. As shown in Figure 5.5(c), due to different respiratory states and nonlinear organ motion, simple subtraction is not sufficient, but with a prior registration allows to identify suspicious regions, see Figure 5.5(d).

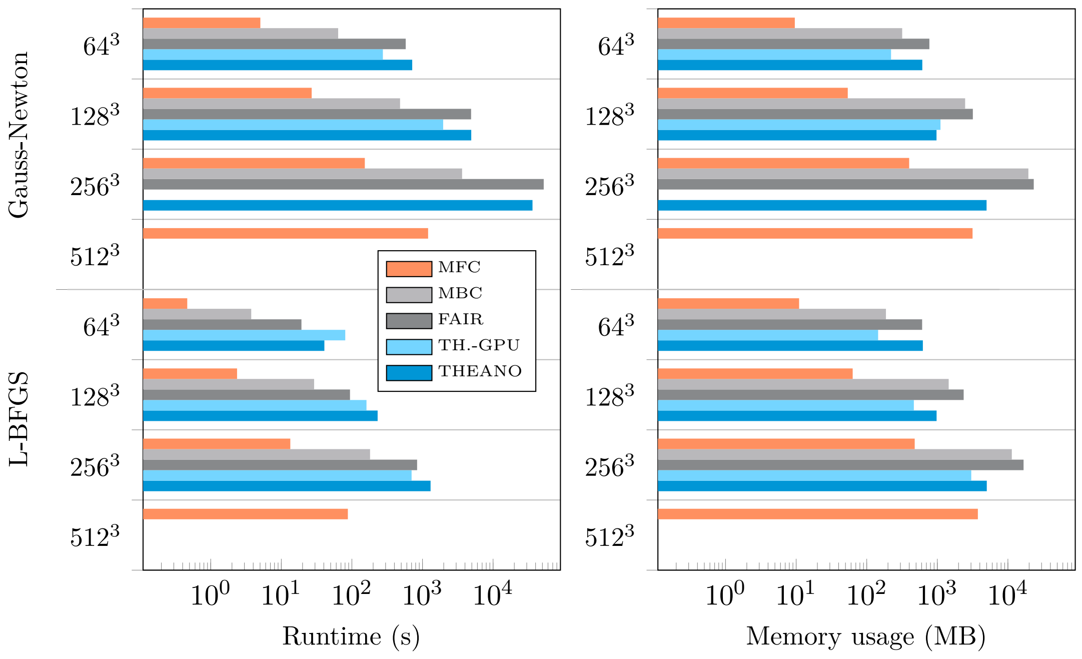


Figure 5.6.: Runtime (left) and memory usage (right) for a full multi-level registration with SSD for different algorithms and image sizes (logarithmic scale). For each evaluation, a coarse-to-fine multi-level registration with three levels was performed; the deformation grid size was chosen as one quarter of the image size on each level. The proposed matrix-free method (MFC) achieves both the fastest runtimes and the lowest memory consumption. Theano-based implementations with automatic differentiation are a flexible alternative to matrix-based methods MBC and FAIR with similar performance, especially for Gauss-Newton optimization. For detailed results see Table 5.7 and Table 5.8.

For the experiments in this section, the original images have been cropped to a size of 512^3 voxels. We then performed full multi-level registrations as described in Section 2.5.6 for various resolution settings at the finest level. Starting with the original image size of 512^3 voxels, we also investigated downsampled image sizes of 256^3, 128^3 and 64^3 in order to analyze performance on smaller images. All registrations were computed using a multi-level scheme with three levels and the deformation resolution was chosen as one quarter of the image resolution on each level. Specifically, a registration with the finest image size of 512^3 voxels starts with an image size of 128^3, then 256^3 and finally 512^3 voxels, with deformation grid sizes of 33^3, 65^3 and 129^3 grid points.

We recorded the runtime and memory requirements of the full multi-level registration for the matrix-based FAIR and MBC algorithms, for the Theano implementation with automatic differentiation from Chapter 4 (THEANO) on CPU as well as on GPU, and for the matrix-free computations (MFC) proposed in Chapter 3. For the THEANO implementations, the Python-based objective function derivative calculations were integrated into the C++-based registration framework as described in Section 4.5.6. We also included the MFC implementation alternatives, discussed in Section 5.2.

Algorithm	Runtime L-BFGS (s)				Runtime Gauss-Newton (s)			
	512^3	256^3	128^3	64^3	512^3	256^3	128^3	64^3
FAIR	*	839	94.0	19.3	*	52 178	4882	578.7
MBC	*	181	29.1	3.7	*	3641	481	63.9
THEANO	*	1302	231.5	40.7	*	36 200	4908	715.4
THEANO-GPU†	*	701	160.9	80.5	*	*	1964	274.7
MFC	88	13	2.4	0.5	1201	152	27	5.0

Table 5.7.: Total multi-level runtime in seconds using SSD for different image sizes. *: computations exceeded 32 GB of memory (4 GB for GPU); †: using 32-bit floating point precision. See Figure 5.6 for more details.

Algorithm	Memory L-BFGS (MB)				Memory Gauss-Newton (MB)			
	512^3	256^3	128^3	64^3	512^3	256^3	128^3	64^3
FAIR	*	16 669	2362	609	*	23 237	3187	772
MBC	*	11 360	1449	187	*	19 500	2478	318
THEANO	*	5022	977	624	*	4987	974	613
THEANO-GPU†	*	3025	465	145	*	*	1109	221
MFC	3754	478	63	11	3157	399	54	10

Table 5.8.: Total peak memory usage in megabytes (10^6 bytes) for a full multi-level registration with SSD for different image sizes. *: computations exceeded 32 GB (CPU) / 4 GB (GPU) of memory; †: using 32-bit floating point precision. See Figure 5.6 for more details.

All experiments were performed on the same 12-core workstation as in the previous section. To minimize measurement influence during the runtime measurements, all other non-essential processes, including graphical user interface, were terminated and the evaluation was run with highest process priority. For the parameters, we used $\alpha = 10$ as regularizer weight and $\tau, \varrho = 5$ for the NGF edge-filtering parameters. Due to small numerical differences, the number of optimization iterations performed before the stopping criteria are fulfilled can slightly vary between algorithms. As this impedes a fair runtime comparison, we chose a fixed number of 20 iterations on each level for benchmarking.

5.4.1. Runtime

Sum of squared differences. The results for the multi-level registration with SSD are visualized in the left part of Figure 5.6, detailed results are furthermore given in Table 5.7.

Our proposed MFC approach achieves the fastest runtimes for all resolutions and for both L-BFGS and Gauss-Newton optimization schemes. In comparison with FAIR, the speedup of MFC ranges from 38.6 to 64.5 for L-BFGS and from 115.7 to 343.3 for Gauss-Newton: the more complicated derivation of a matrix-free Hessian-vector multiplication for Gauss-Newton and MFC is rewarded by higher speedups than for L-BFGS and MFC.

Compared to MBC, speedup factors range from 7.4 to 13.9 for L-BFGS and from 12.8 to 24.0 for Gauss-Newton, i.e., approximately one order of magnitude for both schemes. Again, the speedups for the Gauss-Newton-based computations are higher than for L-BFGS.

The THEANO implementation on the CPU performed worse than FAIR for optimization with L-BFGS. For Gauss-Newton, the execution times are similar, with a slightly slower computation time for an image resolution of 64^4 voxels and a slightly faster computation for 256^3 voxels. Executing THEANO on the GPU yields speedup factors from 1.4 to 2.5 compared CPU computations, except for L-BFGS at the smallest image size of 64^3 voxels, where the GPU computation is in fact slower than the CPU version. This can be attributed to the longer unpickle times. Generally, for the GPU computations, larger image sizes correlate with larger speedups, which is related to a better utilization of the many-core architecture of the GPU by a larger number of image voxels. For larger images, THEANO on GPU achieves computation times that are in a comparable range with FAIR for L-BFGS optimization. For Gauss-Newton optimization, the computations on GPU are faster than FAIR by a factor of up to 2.5.

As will be further discussed in Section 5.4.2, in all cases MFC is able to compute higher resolutions than all other evaluated algorithms within the 32 GB memory constraint of the benchmark workstation. Especially THEANO-GPU is limited by the 4 GB of graphics memory and can only be used for image resolutions up to 128^3 voxels with Gauss-Newton.

Normalized gradient fields. The results of the runtime evaluation for the multi-level registration with NGF are visualized in the left part of Figure 5.7. Detailed results can be found in Table 5.9. As in Section 5.3, we used the fastest version of MFC from Section 5.2 for the evaluation, precomputing both $\hat{\mathbf{y}}$ and the deformed template image $T(\hat{\mathbf{y}})$, which we thus denote as MFC(T,$\hat{\mathbf{y}}$). Other MFC implementation alternatives, computing different values on the fly, will be compared at the end of this section.

Similar speedups to the SSD case can be observed when comparing the runtimes of MFC(T, $\hat{\mathbf{y}}$) and FAIR for L-BFGS: The MFC(T, $\hat{\mathbf{y}}$) implementation is faster by a factor of 26.2 to 41.1. For Gauss-Newton, the factor ranges from 37.0 to 70.7: the speedup is slightly lower than in the SSD case, consistent with the observations in the previous sections.

MFC(T, $\hat{\mathbf{y}}$) is faster than MBC by a factor ranging from 5.2 to 8.6 for L-BFGS optimization and from 11.1 to 14.6 for Gauss-Newton. These numbers are lower than for the evaluation of only the objective function, especially for the Gauss-Newton Hessian-vector multiplication. This can be attributed to the fact that, in addition to the Hessian matrix, the gradient is also required for solving the Newton equation (Section 2.5.1). Since the speedup for the gradient computations is lower, as shown in Table 5.5 and Table 5.6, this also affects the overall speedup for the Gauss-Newton multi-level registration.

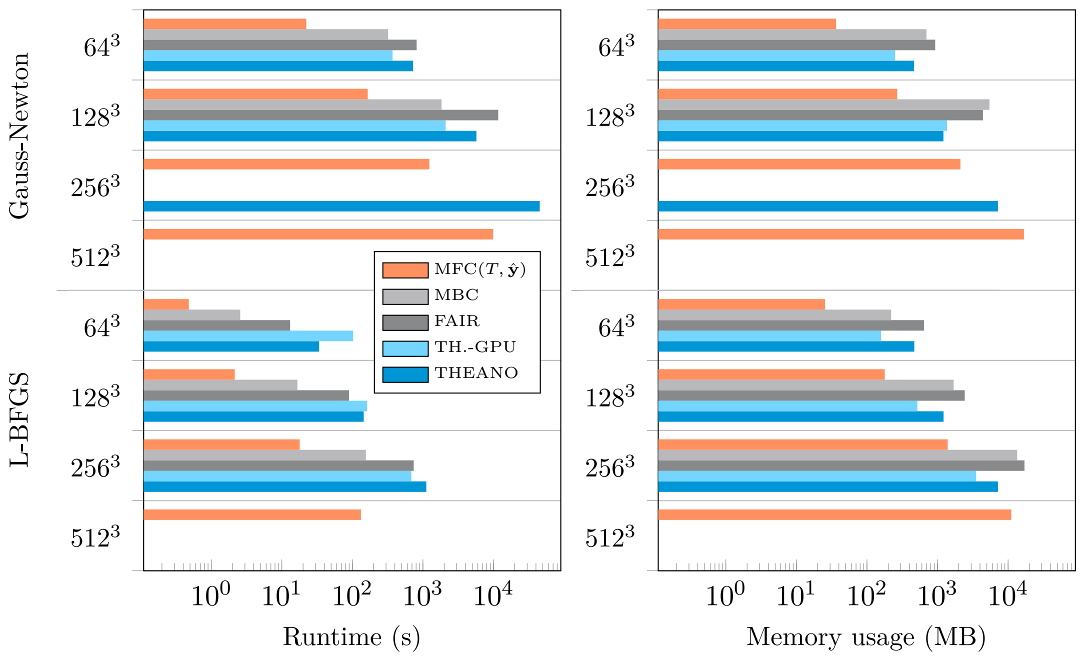


Figure 5.7.: Runtime (left) and memory usage (right) for a full multi-level registration with NGF for different algorithms and image sizes (logarithmic scale). For each evaluation, a coarse-to-fine multi-level registration with three levels was performed; the deformation grid size was chosen as one quarter of the image size on each level. The matrix-free computations (MFC) exhibit both the fastest runtimes and the lowest memory consumption. Theano-based computations with automatic differentiation are a viable alternative to matrix-based methods MBC and FAIR, especially for Gauss-Newton optimization. For detailed results see Table 5.9 and Table 5.10.

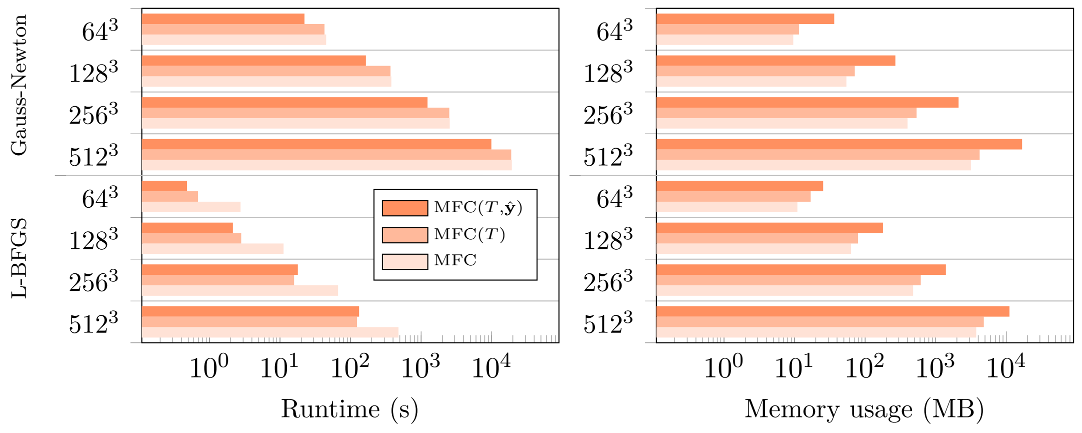


Figure 5.8.: Runtime (left) and memory usage (right) for a full multi-level registration with NGF for different alternative implementations of MFC and varying image sizes (logarithmic scale). While MFC(T,$\hat{\mathbf{y}}$), MFC(T) with selected precomputations achieve faster runtimes in most cases, the memory usage can be considerably higher. For detailed results see Table 5.9 and Table 5.10.

The THEANO computations on CPU are slower than FAIR for L-BFGS on all resolutions. For Gauss-Newton, however, all computations are faster than FAIR, with a speedup factor of 2.03 for an image resolution of 128^3 voxels. Additionally, THEANO on CPU is able to compute the registration with Gauss-Newton at an image resolution of 256^3 voxels, while both FAIR and MBC run out of memory for this size. However, at this resolution, THEANO requires an unreasonably long runtime of more than 12 hours.

In comparison to the CPU version, THEANO on GPU is slower for L-BFGS and image sizes of 64^3 and 128^3 voxels. This mainly results from the large overhead of loading (unpickling) the Theano functions (Table 4.2). For the smaller image sizes, the GPU speedup is not large enough to outweigh the initial overhead. For larger images, resulting in longer iterations times in relation to the loading overhead, computation on GPU is faster by a factor of 1.6 (L-BFGS) and 2.0 to 2.7 (Gauss-Newton) at 256^3 voxels. For Gauss-Newton, the GPU-based computations are then only slightly slower than MBC and from 2.2 to 5.6 times faster than FAIR.

In comparison to the SSD-based multi-level registrations, for the Gauss-Newton optimization the restrictions imposed by memory size are even larger. While all MFC implementations handle all evaluated sizes, image sizes of 256^3 and 512^3 exceed the capabilities of FAIR and MBC.

Effect of selective precomputation. Figure 5.8 and Table 5.9 visualize the multi-level runtimes for the MFC implementation alternatives from Section 5.2. The alternatives differ in the amount of selectively precomputed structures. While MFC(T, $\hat{\mathbf{y}}$), which was used previously in this section, precomputes the deformation on the image grid $\hat{\mathbf{y}}$ as well as the deformed template image $T(\hat{\mathbf{y}})$, MFC(T) precomputes only $T(\hat{\mathbf{y}})$ but computes $\hat{\mathbf{y}}$ on the fly. Finally, MFC computes both $T(\hat{\mathbf{y}})$ and $\hat{\mathbf{y}}$ on the fly.

For L-BFGS optimization, there is no substantial difference in runtime between MFC(T, $\hat{\mathbf{y}}$) and MFC(T), which is consistent with the observations in Section 5.2. The MFC computations, however, which compute everything on the fly, are slower than MFC(T) by a factor of approximately 4 for all image sizes and L-BFGS optimization.

For Gauss-Newton optimization MFC(T, $\hat{\mathbf{y}}$) yields the fastest computations, with no substantial difference between MFC(T) and MFC, consistent with the observations in Section 5.2.

In summary, selective precomputations can further reduce the runtimes of the matrix-free registration. Depending on the optimization scheme, different amounts of precomputations achieve the lowest runtime. While for L-BFGS further precomputations than $T(\hat{\mathbf{y}})$ do not result in further benefits, the runtime for Gauss-Newton can be reduced even more. Naturally, this results in higher memory requirements, which will be discussed in the next section.

5.4.2. Memory requirements

In contrast to algorithm runtime, which can be inconvenient or impractical when too large, peak memory requirements impose a hard limit on the problem size.

Algorithm	Runtime L-BFGS (s)				Runtime Gauss-Newton (s)			
	512^3	256^3	128^3	64^3	512^3	256^3	128^3	64^3
FAIR	*	740	89.6	13.1	*	*	11 667	814
MBC	*	155	16.6	2.6	*	*	1835	321
THEANO	*	1117	144.9	33.8	*	45 276	5734	725
THEANO-GPU†	*	685	161.2	102.6	*	*	2100	371
MFC(T,$\hat{\mathbf{y}}$)	132	18	2.2	0.5	9908	1233	165	22
MFC(T)	124	16	2.8	0.7	18 937	2509	368	43
MFC	478	67	11.2	2.8	19 327	2547	378	45

Table 5.9.: Total multi-level runtime in seconds using NGF for different image sizes. *: computations exceeded 32 GB (CPU) / 4 GB (GPU) of memory; †: using 32-bit floating point precision. See Figure 5.7 and Figure 5.8 for more details.

Algorithm	Memory L-BFGS (MB)				Memory Gauss-Newton (MB)			
	512^3	256^3	128^3	64^3	512^3	256^3	128^3	64^3
FAIR	*	17 086	2432	643	*	*	4402	927
MBC	*	13 459	1694	220	*	*	5446	697
THEANO	*	7185	1219	469	*	7183	1213	466
THEANO-GPU†	*	3526	518	158	*	*	1363	251
MFC(T,$\hat{\mathbf{y}}$)	11 103	1395	179	25	16 783	2110	268	36
MFC(T)	4812	615	79	17	4200	537	71	12
MFC	3755	478	63	11	3158	399	54	10

Table 5.10.: Total peak memory usage in megabytes (10^6 bytes) for a full multi-level registration with NGF for different image sizes. *: computations exceeded 32 GB (CPU) / 4 GB (GPU) of memory; †: using 32-bit floating point precision. See Figure 5.7 and Figure 5.8 for more details.

Therefore, we evaluated the peak memory usage of all algorithms for the full multi-level registration, including the images and their multi-level representations, the objective function derivatives, memory required for the optimization algorithm and CG solver, and further auxiliary variables.

Sum of squared differences. For SSD, the memory requirements are visualized in the right part of Figure 5.6. Additionally, detailed results are given in Table 5.8. The two matrix-based algorithms FAIR and MBC have the highest memory usage. For L-BFGS optimization, MBC and FAIR require 16.6 GB and 11.3 GB of memory for a full multi-level registration with an image size of 256^3 voxels, respectively. Even more memory is required for Gauss-Newton optimization: FAIR requires 23.2 GB of memory, MBC needs 19.5 GB. Larger image sizes exceed 32 GB and therefore cannot be computed on the benchmark workstation.

In contrast to this, the memory usage of MFC is substantially lower. Even for an image resolution of 512^3 voxels, only 3.7 GB are required for L-BFGS optimization and 3.1 GB for Gauss-Newton optimization: MFC improves the overall memory consumption of the full algorithm by two orders of magnitude in comparison to the matrix-based algorithms.

Since the L-BFGS scheme requires additional buffers for storing gradient information from previous iterations, memory consumption for L-BFGS is even higher than for Gauss-Newton, which is the opposite for the matrix-based algorithms.

The memory consumption for THEANO on CPU is lower than FAIR and MBC for large image sizes by factors of 3.3 and 2.3 for L-BFGS, and 4.7 and 3.9 for Gauss-Newton and an image size of 256^3 voxels, respectively. However, the THEANO computations exhibit a comparatively large baseline memory-usage: For images of size 64^3, the required memory is about 600 MB, which is comparable to FAIR, while MBC only requires 187 MB for L-BFGS and 318 MB for Gauss-Newton optimization in this case, which makes THEANO more suitable in cases where larger images are processed.

The GPU version of THEANO requires less memory for L-BFGS optimization than the CPU implementation. This is at least partially caused by the use of single precision floating-point values (Section 4.5.6). However, for an image size of 128^3 voxels and Gauss-Newton optimization, the required memory of the GPU version is even larger than for the CPU version, which is probably caused by different graph optimizations performed for Theano GPU computations. Therefore, computations with an image size of 256^3 voxels already exceed the available 4 GB GPU memory on the benchmark workstation.

Normalized gradient fields. The measured memory requirements for the full multi-level registration with the NGF distance measure are visualized in the right part of Figure 5.7. Furthermore, detailed results are given in Table 5.10.

The obtained results are generally similar to the SSD results reported above, although the memory requirements of the matrix-based algorithms and THEANO are slightly higher. While a full multi-level registration with L-BFGS optimization requires 17.1 GB and 13.4 GB of memory for FAIR and MBC with an image size of 256^3 voxels, a registration with MFC only requires 3.8 GB on the next larger image size with 512^3 voxels. Similarly to the SSD computations, MFC reduces memory consumption by two orders magnitude.

Again, THEANO-GPU has a lower memory consumption than the matrix-based FAIR and MBC. THEANO has a higher baseline memory consumption, such that for L-BFGS at 64^3 voxels more memory than MBC is required. For larger image sizes, the relation reverses: for L-BFGS optimization and 256^3 voxels, THEANO has a memory consumption that is lower by a factor of 2.4 when compared to FAIR and 1.9 when compared to MBC. For Gauss-Newton, the memory consumption is even lower by a factor of 3.6 when compared to FAIR and 4.5 when compared to MBC. Due to this, THEANO is able to compute a full multi-level registration with Gauss-Newton optimization and image sizes of 256^3 voxels, while the matrix-based algorithms exceed the available 32 GB of memory.

Effect of selective precomputations. As the precomputation-based variants selectively store often-used elements, they require substantially more memory. As can be seen in Table 5.10, in particular the amount of memory required for storing $\hat{\mathbf{y}}$ and the resulting gradient on the image grid weighs heavily for larger image sizes. For 512^3 voxels and L-BFGS, MFC requires 3.8 GB of memory, while MFC(T,$\hat{\mathbf{y}}$) needs 11.1 GB. For Gauss-Newton, the difference is even larger and MFC(T,$\hat{\mathbf{y}}$) needs 16.8 GB, while MFC only requires 3.2 GB of memory.

While these differences in memory consumption can make a large difference on the ability to run an algorithm with a certain image size in practice, they have to be considered in relation to the corresponding runtimes. As shown in Figure 5.8, the selectively precomputed values yield a faster runtime in most cases. While for L-BFGS optimization, MFC(T) yields the best compromise between runtime and memory requirements, for Gauss-Newton optimization, MFC(T,$\hat{\mathbf{y}}$) obtains substantially lower runtimes, such that, if possible, the additional memory requirements should be accepted.

Additionally, it can be observed that the memory requirements for plain MFC, i.e., computing everything on the fly, are identical for SSD and NGF. This lines up with theory: Since no intermediate results are stored for gradient and Hessian computation, memory requirements are solely for the remaining parts of the algorithm, which are identical between the SSD and NGF versions.

5.4.3. Summary

Our analysis has shown that the matrix-free computations allow for a substantially faster runtime for all evaluated image sizes and both L-BFGS and Gauss-Newton optimization when compared to matrix-based algorithms. With speedup factors of up to three orders of magnitude for SSD and two orders of magnitude for NGF, depending on image size and optimization method, MFC computations can potentially achieve clinically feasible runtimes for image sizes that were impractical to register in daily routine before.

Additionally, MFC features highly reduced memory requirements. While the largest image size of 512^3 voxels exceeds the limits of all other algorithms tested, MFC potentially enables computations with even larger images.

The THEANO algorithm cannot compete with MFC. However, since very little development effort is required due to the automatic differentiation, it represents a valuable alternative to matrix-based algorithms, especially FAIR. This makes the Theano-based algorithm especially suitable for model development and rapid prototyping scenarios.

THEANO transfers transparently to the GPU with an additional speedup over the CPU version on larger images. However, this limits the computations by the available GPU memory, and results in a large runtime overhead for loading the compiled functions.

	Algorithm	Runtime (s)				Memory usage (MB)			
		512^3	256^3	128^3	64^3	512^3	256^3	128^3	64^3
SSD	MFC	88	13.4	2.4	0.5	3754	478	63	11
SSD	MFC-GPU	*	1.5	0.5	0.2	*	566	159	104
NGF	MFC(T,$\hat{\mathbf{y}}$)	132	17.9	2.2	0.5	11 103	1395	179	25
NGF	MFC-GPU	*	1.5	0.3	0.2	*	566	155	107

Table 5.11.: Multi-level runtime (s) and memory usage (MB) for matrix-free registration on the GPU for different image sizes. *: computations exceeded 4 GB of GPU memory. See Figure 5.9 for more details.

5.5. GPU-based matrix-free registration

In addition to the matrix-free algorithm on the CPU, we implemented the matrix-free gradient computations on the GPU using NVIDIA CUDA. While the Theano-based computations can be automatically performed on the GPU by simply activating a runtime flag, performing the matrix-free computations on GPU requires a completely new implementation (Section 3.8.2). Therefore, we evaluate the matrix-free algorithm on GPU (MFC-GPU) separately in this section and compare the runtime and memory usage to its equivalent on the CPU (MFC) in order to assess the potential benefits of a specialized implementation.

Again, we utilized the same workstation, equipped with an NVIDIA GeForce GTX 980 GPU with a base clock speed of 1.1 GHz and 4 GB of memory.

The workstation achieves a theoretical peak performance of approximately 250 GFLOPS for CPU computations. Compared to this, the double-precision peak performance of the GPU is much lower with only 144 GFLOPS. However, the GPU has a 32 times higher theoretical single-precision peak performance [Arr18] with up to 4.6 TFLOPS [NVI18], which constitutes a potential performance advantage over the CPU. Therefore, GPU-accelerated computations are performed using single-precision computations, as in the THEANO-GPU implementation (Section 4.6).

We evaluated MFC-GPU with the same images and parameters as used in Section 5.4. Only the matrix-free gradient computations were implemented on GPU, as the implementation was used primarily in an L-BFGS-based application. Therefore, we restricted the evaluation to L-BFGS optimization. Additionally, in the GPU implementations, $T(\hat{\mathbf{y}})$ and $\hat{\mathbf{y}}$ were precomputed. For a fair comparison, we compare the results with MFC(T,$\hat{\mathbf{y}}$) from the previous section, which precomputes the same values.

5.5.1. Runtime

The runtimes are given in Table 5.11 and Figure 5.9 for SSD and NGF. The GPU computations achieve an additional speedup ranging from 2.5 to 8.9 for SSD and from 2.5 to 11.9 for NGF. Similar to the results of THEANO-GPU discussed in Section 5.4, larger image

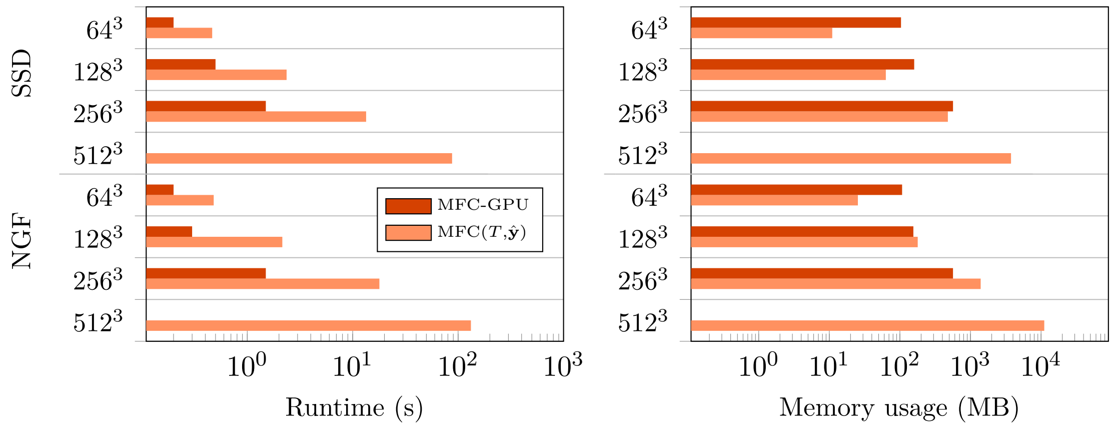


Figure 5.9.: Runtime (left) and memory usage (right) for a full multi-level registration, using matrix-free computations on the GPU for different image sizes. The GPU-based implementation achieves an additional speedup of up to 11.9, but is limited by the size of the GPU memory (4 GB).

sizes are beneficial. The lowest speedup factors are achieved for images with a size of 64^3 voxels, while the highest speedup is obtained for image sizes with 256^3 voxels, as this was the largest size that could be evaluated with MFC-GPU. The additional speedup now allows to perform a full multi-level registration in 1.5 s for SSD and NGF. In conclusion, while the implementation of MFC-GPU requires additional effort for creating the CUDA implementation, it is rewarded with a considerable additional speedup.

5.5.2. Memory requirements

The GPU implementation uses precomputations for $T(\hat{\mathbf{y}})$ and $\hat{\mathbf{y}}$, similar to MFC(T,$\hat{\mathbf{y}}$) for NGF as well as for SSD. However, since single-precision computations are used, less memory is required for deformations and deformed template image. As shown in Table 5.11, a considerable amount of memory is still required for MFC-GPU. Together with the limited GPU memory of 4 GB, this leads to limitations on the maximum image sizes, such that the largest images with 512^3 voxels cannot be computed. We presume that an implementation without precomputations would be beneficial, at the the cost of additional implementation effort.

5.6. Discussion and summary

Our findings in the previous sections of this chapter show that the derived matrix-free computations for *distance measure derivatives* and *grid conversion* exhibit the desired approximately linear scaling behavior (Section 5.1). However, a sufficient number of parallel tasks in relation to the number of computational cores is required in order not to limit the computational performance. In comparison, we found that the scalability of the *curvature regularizer* is limited, due to a small ratio of computations to memory operations, which causes the computations to be limited by memory bandwidth. This, however, is not a major concern for the final algorithm, since the evaluation of the curvature regularizer is several orders of magnitude faster than the distance measure computations.

For the full objective function, the matrix-free computations were up to three orders of magnitude faster than their matrix-based equivalents (Section 5.3). Additionally, higher resolutions can be computed, where the matrix-based algorithms run out of memory.

On the full multi-level real-world registration dataset (Section 5.4) the matrix-free computations are several orders of magnitude faster than the matrix-based algorithms. Additionally, the matrix-free computations require substantially less memory: a multi-level registration with an image size of 512^3 voxels can only be performed by MFC, all other algorithms exceed the available memory on our workstation. Together, fast runtimes which scale linearly with additional computational cores, as well as very low memory requirements make the matrix-free computations favorable in every aspect over the matrix-based implementations.

While substantially slower than the matrix-free methods, the Theano-based implementations achieve runtimes on par with the matrix-based methods. Therefore, considering their excellent flexibility, the Theano framework is especially suitable for development of new models and rapid prototyping. Switching to GPU-based computations can lead to further speedup, although mostly for larger images due to large overheads when loading the compiled functions on GPU. Furthermore, these methods are limited by the smaller GPU memory.

We found that even matrix-free computations with higher memory requirements due to precomputed values still exhibit a substantially lower memory usage than all other evaluated algorithms. On the other extreme, the matrix-free derivative computations without any precomputations exhibit very low, constant memory requirements which makes the matrix-free methods capable of potentially computing registrations with very large image sizes. In practice, the requirements of the specific application scenario are essential deciding between faster runtime and lower memory consumption.

In Section 5.5, we evaluated a GPU-based implementation of the matrix-free algorithm. In comparison with the CPU implementation, a considerable additional speedup could be achieved, at the cost of considerable implementation effort and stricter size limits due to the smaller GPU memory. In particular, for time-critical applications, the GPU implementation is a valuable addition to further reduce runtime. Here, the low runtimes potentially enable new application scenarios, e.g., in intra-operative settings.

Together, matrix-free and Theano-based methods offer suitable alternatives for all stages of algorithm development, ranging from development and exploration of the mathematical model to derivation of a fast and efficient algorithm for practical use. Here, especially the matrix-free computations enable new potential applications due to their speed and low memory requirements. Selected applications, where these properties are crucial, will be presented in the following chapters.

Part II.

Large-scale and real-time applications in medical imaging

6 Follow-up thorax-abdomen registration in radiology

In radiology, so-called *follow-up diagnoses* routinely involve, comparing current CT images with CT images from an earlier time point. In particular in the thorax-abdomen region, non-linear deformations due to different breathing states, intestinal motion, tumor growth, and cancer therapy can make a manual comparison of images difficult.

The diagnosing radiologist is obliged to inspect the entire scan, compare it with prior scans and document all findings and observed changes. However, manually navigating through both datasets side-by-side in order to identify corresponding locations can be a lengthy and tiring process: the clinician often spends a substantial amount of time with the manual process of navigating the three-dimensional datasets.

In this setting, deformable image registration between *current* and *prior image* can support the clinician in quickly finding the corresponding location. Using the computed deformation, positions in both images can be linked using *cursor synchronization* or *synchronized navigation* [FGM+17; HBS06; Bal06; LPG+05]. Additionally, subtraction images can reveal locations of subtle anatomical changes that need closer investigation [EPW+07; TVF+02].

In a clinical setting, the computation time for registering current and prior CT scan is important in order not to impede the clinical workflow. As thorax-abdomen CT scans can easily exceed image sizes of $512 \times 512 \times 1000$ voxels, this poses a challenge in terms of both memory usage and runtime. This makes the application scenario attractive for our proposed matrix-free registration, potentially allowing to process these large datasets in clinically feasible runtime.

In Section 6.1, we present an automatic processing pipeline for the registration of thorax-abdomen CT images and a method for obtaining synchronized locations on different images using the computed deformation. The registration is evaluated on a large clinical dataset in Section 6.2, and the experimental results are discussed in Section 6.3.

6.1. Thorax-abdomen registration

In order to support the radiologist in examining follow-up scans, we implemented a fully automatic registration pipeline for registering intra-patient thorax-abdomen CT images.

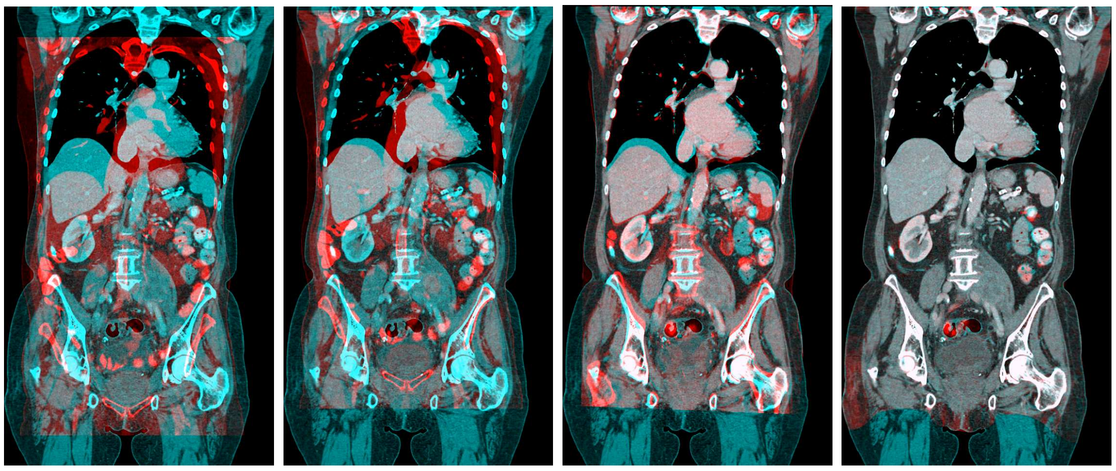

(a) Center align (b) Grid search (c) Pre-registration (d) Deformable reg.

Figure 6.1.: Color overlay for coronal slices of prior (blue) and follow-up image (red) at different stages of the processing pipeline and the final aligned result after registration. At each stage, the registration is further refined, leading to a pre-registration (c), which is a suitable starting point for the deformable registration. Datasets courtesy of Radboud University Medical Center, Nijmegen, The Netherlands.

The main goal was to robustly and automatically register large numbers of original clinical CT scans without the need for user intervention.

As images from the same patient taken at a different time points may have been acquired with different devices and with different diagnostic purpose, varying scanners, image sizes and spacings as well as different fields of view have to be considered. The different steps of the proposed framework are shown in Figure 6.1. Here, an image from a Toshiba Aquilion ONE scanner, with a size of $m = (512, 512, 766)$ voxels and spacings of $h = (0.69\,\text{mm}, 0.69\,\text{mm}, 0.8\,\text{mm})$ was registered to an image acquired with a Siemens Sensation 64 device, with a size of $m = (512, 512, 1008)$ voxels and spacings of $h = (0.86\,\text{mm}, 0.86\,\text{mm}, 0.7\,\text{mm})$. Each step of the pre-processing pipeline successively refines the registration result, resulting in a pre-alignment that is suitable as a starting point for the deformable registration.

6.1.1. Processing pipeline

Depending on the device used, gray values can be stored differently, either directly as Hounsfield units (HU) or as unsigned integer values with a corresponding offset value, typically -1024. Therefore, we initially normalize the image gray values such that a gray value of zero corresponds to -1024 HU, which typically characterizes the air around the body and is considered the image background. Lower HU values are clamped to zero, such that all gray values are nonnegative. Furthermore, we normalize the orientation of the patient, which is required for some images which were acquired in a prone or lateral recumbent (lying on the side) position.

Coordinate center alignment. As images from different scanners often have very different world coordinate locations, first an initial overlap of the images has to be established. For this, we align the centers of both images with the coordinate origin. While this already provides a good pre-alignment in some cases where scanner and field of view of both images are identical, in other cases this only provides a starting point for the following alignment steps.

Grid search. In the second step we perform a coarse translational grid-search, using all possible translations with a pre-defined step width and maximum extent. The final alignment is determined as the one that minimizes the SSD distance over all sampled alignment positions.

As image pairs with differing fields of view typically contain parts which are contained in only one of the images, we only evaluated the distance measure at deformation grid points that lie within the intersection of the transformed template and reference image: using $\mathcal{V} := \{i \in [1, \ldots, \bar{m}] \mid \varphi(\mathbf{x}_i) \in \Omega_\mathcal{T}\}$, we compute

$$D_\mathcal{V}^{\text{SSD}}(\mathbf{y}) := \frac{\bar{h}}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} (T_i(P(\mathbf{y})) - R_i(\mathbf{x}))^2 \,,$$

which includes an additional normalization by $|\mathcal{V}|$.

Pre-registration. After the coarse grid search, the alignment is refined by a *rigid* (rotation and translation only) multi-level pre-registration using the SSD distance measure, using the coarsely aligned images as a starting point. The restriction to rigid transformations is implemented as described in Section 2.4.2. As shown in Figure 6.1(d), the resulting alignment is a suitable starting point for the deformable registration.

The purpose of this multi-step approach is to successively move from robust methods with fewer degrees of freedom to more precise, but less robust methods that require better starting points due to their higher degrees of freedom.

Deformable registration. The final step in the proposed registration framework is a matrix-free deformable registration as discussed in Chapter 3. We employ the NGF distance measure, as presented in Section 2.3.1, which focuses on aligning image gradients. In our experiments NGF has provided more robust alignments than SSD for the deformable registration. The registration is embedded in a multi-level scheme with L-BFGS optimization (Section 2.5). Due to the large sizes of the CT datasets, the registration runtime and memory consumption greatly benefits from the matrix-free algorithm, allowing to register both images within a couple of seconds (see Section 6.2).

6.1.2. Propagation of lesion locations

To provide synchronized navigation between the prior and current image, the computed deformation needs to be evaluated in both directions: Selecting a point in the prior image and obtaining the corresponding location in the current image and vice versa, as shown in Figure 6.2.

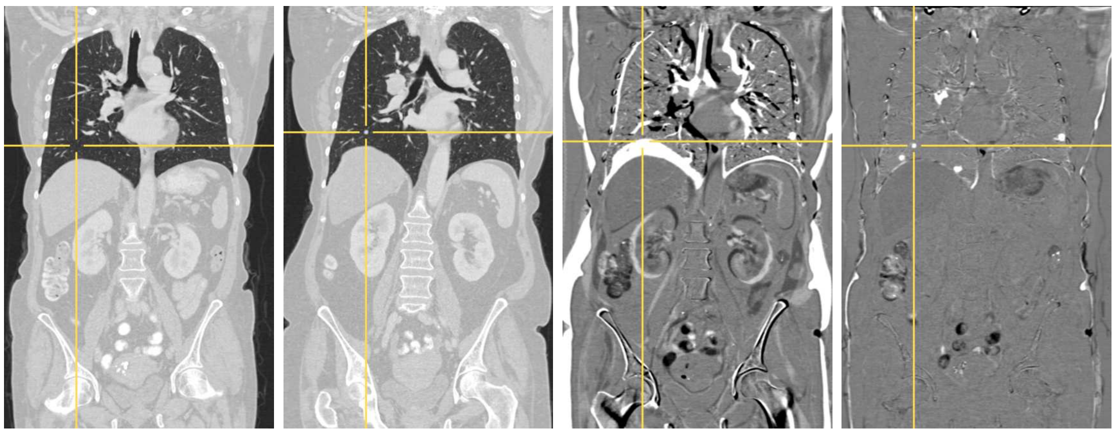


Figure 6.2.: Cursor synchronization for lesion investigation, cursor (yellow) in (a) prior image, (b) follow-up image, as well as in difference images (c) before registration and (d) after registration. The cursor synchronization is performed in real time using the deformation previously computed with the proposed processing pipeline. The new lesion is clearly visible in the difference image after deformable registration. Cursor synchronization allows to easily locate the corresponding location in the prior scan. Datasets courtesy of Radboud University Medical Center, Nijmegen, The Netherlands.

One of the directions is trivial: using the transformation φ, any point $\mathrm{x} \in \Omega_{\mathcal{R}}$ on the reference image domain can be directly mapped to the template image with $\varphi(\mathrm{x}) = \mathrm{y}$. However, computing the opposite direction $\varphi^{-1}(\mathrm{y}) = \mathrm{x}$ requires the inverse of the transformation. While there exist algorithms for inverting the full deformation field [CCH07; CLC+08], these require additional computation time and memory for storing the inverse transformation. Therefore, we compute a point-wise inverse at the current location on the fly. For a point $\mathrm{y} \in \Omega_{\mathcal{T}}$ in the template image domain, with $f : \mathbb{R}^3 \to \mathbb{R}$, we compute an approximation x with

$$\min_{\mathrm{x} \in \Omega_{\mathcal{R}}} f(\mathrm{x}) := \|\varphi(\mathrm{x}) - \mathrm{y}\|^2$$

on the reference image domain $\Omega_{\mathcal{R}}$ as defined in Section 2.4.1, obtaining an approximation for $\varphi^{-1}(\mathrm{y})$. As box constraints are applied on the variable, we solve the constrained optimization problem using the gradient projection method [NW06, §16.7] with the center of $\Omega_{\mathcal{R}}$ as a starting guess and terminating if $f < 10^{-6}$ or 100 iterations are reached. In this application, the deformations have generally been found to be sufficiently smooth for the optimization to converge. As derivatives of f can be computed rapidly, the approximation to the inverse transformation can be computed on the fly without perceptible delay.

6.2. Evaluation

We evaluated the proposed registration framework on a large dataset of oncological thorax-abdomen CT scans, collected at Radboud University Medical Center, Nijmegen,

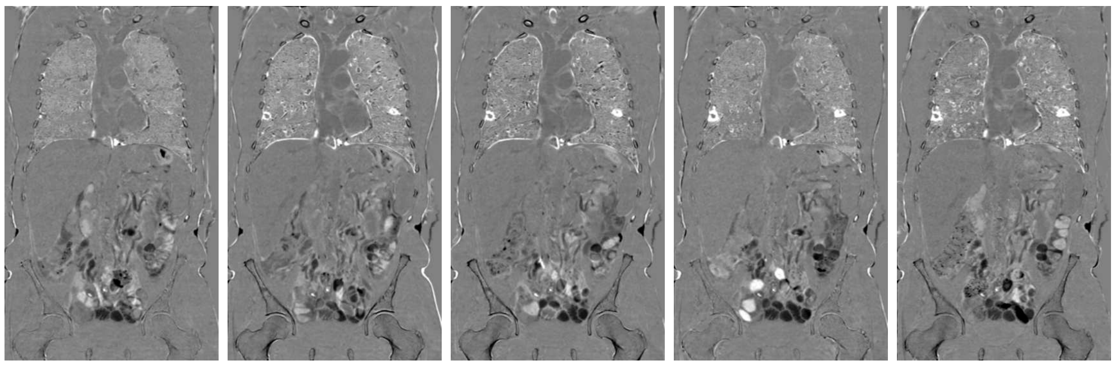

(a) Follow-up 1 (b) Follow-up 2 (c) Follow-up 3 (d) Follow-up 4 (e) Follow-up 5

Figure 6.3.: Difference images of deformed follow-up images and prior scan for multiple time points, spaced approximately two months apart. While, due to missing correspondence, different bowel contents appear as non-gray areas in the lower part of the images, lesion progression is clearly visible in the lung.

The Netherlands. In total, datasets from 489 different patients were available, each with one to seven follow-up scans, resulting in a total of 986 individual registrations. The images originated from four different scanners, with sizes of $m_x = 512$ and $m_y = 512$ voxels for all images and an average of $m_z = 648$ voxels, with average spacings of $h = (0.76\,\text{mm}, 0.76\,\text{mm}, 1.14\,\text{mm})$.

For the deformable registration we manually chose the NGF edge parameters as $\varrho, \tau = 5$ and a regularizer weight of $\alpha = 10$, achieving visually acceptable results. We used three levels in the multi-level scheme, with a quarter of the original resolution in each direction as the finest level. The deformation resolution was chosen as half the image resolution in each direction on every level.

All 986 registrations were processed successfully using the proposed scheme, without clearly visible failures or misalignments. Since no manual annotations for evaluation of the registration error were available on the dataset, and to the best of our knowledge no publicly available dataset for thorax-abdomen registration with manual annotations exists, we evaluated the results by visual inspection.

In all cases, the deformable registration substantially improved the pre-registration results, as shown for an exemplary dataset in Figure 6.2. Difference images for an exemplary case with four follow-up scans, acquired at two month intervals are shown in Figure 6.3. Here, lesion changes over time can be observed. Due to intestinal motion and different bowel contents, the accuracy of the registration in the bowel region, visible in the lower part of the image, is limited and missing correspondences appear as non-gray areas in the difference images.

The average runtime of the complete processing pipeline was 38.1 s per dataset on a workstation with Intel Core i7-6700K CPU with four cores and 4.00 GHz, with a maximum value of 104.5 s and a minimum value of 18.4 s. Out of this, the matrix-free deformable registration took 8.3 s on average, with a minimum of 2.0 s and a maximum of 69.3 s, while the remaining runtime is spent for data loading, grid search, pre-registration and computation of the deformed image for visualization of the difference image.

6.3. Summary and conclusion

We presented a method for automatic pre-processing and registration of thorax-abdomen CT datasets. In this important scenario, our matrix-free approach brings down the – usually dominant – time for computing the deformable registration to less than one quarter of the total runtime. The computed deformation allows for real time cursor synchronization between the images and can potentially aid radiologists in performing follow-up diagnoses.

Evaluated on a large number of datasets, our approach shows visually convincing results. Future evaluations with ground truth annotations on thorax-abdomen datasets could potentially be used to further refine the registration results and to allow for a quantitative evaluation. At the time of writing, however, such ground truth data was not available for the thorax-abdomen region.

Improvements to the registration accuracy could be made by including further anatomical information into the registration, as proposed in Chapter 7. For instance in the lung region, by explicitly aligning the lung contours the registration accuracy could potentially be improved, using a similar approach as [RHKF13]. In the intestinal region, masking the image distance in areas without correspondence could also help to improve alignment for surrounding organs. Segmentations of the bowel region could additionally be used in the result visualization to hide differences in this area, allowing the radiologist to focus on indicated lesions in difference images. These approaches, however, require segmentations of the respective anatomical areas, which could either be created manually by radiologists or an automatic segmentation method.

With an average total runtime of 38.1 s, the proposed method is well suited for clinical application. As part of a larger oncology workstation, the algorithm is currently being evaluated in clinical practice, potentially leading to a faster and easier radiology workflow.

7 Deformable registration with local rigidity in radiotherapy

In this chapter, we present an extension of the matrix-free registration algorithm with a local rigidity constraint for a clinical application in the field of radiotherapy. As discussed in Section 1.2.2, patient imaging is an important component of every image-guided radiation treatment process, which relies on an efficient image registration in order to transfer and compare patient information between images acquired at different points in time.

In a typical clinical workflow, a diagnostic CT image of the patient is acquired before radiation therapy is started. A then clinician delineates the tumor volume as well as important organs and risk structures, as shown in Figure 7.1(a). With this information, a radiation treatment plan is created. Therefore, the diagnostic CT is also called *planning CT*. The treatment plan includes exact information on how the radiation is applied to the patient by the treatment device, so that as much radiation as possible is delivered to the *clinical target volume* (CTV), while keeping irradiation of surrounding tissue, organs and other high-risk structures as low as possible.

For this, the treatment is typically split into sessions over several weeks, so-called *fractions*, where in each fraction a part of the planned total dose is applied. In image-guided radiotherapy, a *cone-beam CT* (CBCT) image of the patient is acquired before each fraction, as shown in Figure 7.1(b), while the patient is already in treatment position. Using deformable image registration, this CBCT image is then compared to the planning CT in order to verify the correct patient position and to account for anatomical changes that occurred since the creation of the treatment plan [JKDM07].

Additionally, with the computed deformation, delineations of CTV and risk structures can be propagated from the planning CT to the CBCT image in order to assess the planned radiation dose distribution with respect to the current patient anatomy. Further applications include the retrospective accumulation of the total dose, based on CBCT anatomy for all fractions, in order to assess potential differences between treatment plan and actually delivered radiation [TAP+14], as well as customization of the treatment in adaptive radiotherapy [Kes06; THT+16].

Local rigidity. Especially in the pelvic region, the deformable registration of CT and CBCT is difficult: in the male pelvis structures with very different deformation properties are very close to each other [KLW+08; MSD03] (Figure 7.2). While the bladder deforms in a highly elastic way, bones exhibit rigid, and prostate exhibits nearly-rigid

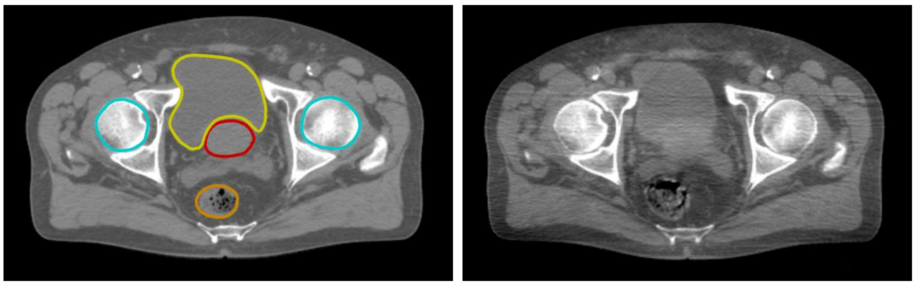


Figure 7.1.: Male pelvis images from radiotherapy, (a): axial slice of planning CT with delineations of bladder (yellow), prostate (red), femoral heads (blue) and rectum (orange), (b): axial slice of CBCT. The delineations on the planning CT are typically created manually during the creation of the treatment plan and are to be transferred to the CBCT by using deformable registration.

deformation behavior [HMC+07], creating a challenging registration problem. As the prostate is the CTV in the case of prostate cancer treatment in the male pelvic region, a substantial amount of radiation needs to be delivered to this area. However, structures in very close proximity such as the bladder wall or the femoral heads are very sensitive to radiation. Thus, high gradients in radiation dose distribution can be expected between these structures such that slight deviations in the propagated structures can have a large influence on dose evaluations. Standard registration algorithms often achieve unsatisfying results with physically implausible deformations, such as elastic deformation of bones [TPB+11].

Therefore, in this chapter, we introduce an extension of the matrix-free registration scheme to this application scenario. In Section 7.1 and Section 7.2, we add an additional local rigidity constraint to the registration model, which restricts the deformation of certain structures to rotation and translation, while still allowing for non-linear deformations of surrounding tissue.

In Section 7.3, we will evaluate the algorithm on a dataset of ten cases from clinical practice. Besides evaluating the algorithm against manually created gold standard contours, we will also compare the algorithm to the registration available in the commercial software Varian SmartAdapt [WDO+05; RPH+15]. Additionally, we compare the registration with local rigidity to the unconstrained matrix-free registration as presented in Chapter 3 and evaluate different choices of rigid structures. Finally, we discuss the results in Section 7.4.

This application highly benefits from the fast runtimes and low memory consumption of the matrix-free algorithm. Since the registration is embedded in a clinical environment, the runtimes must be feasible for daily practice. Especially when applied in adaptive radiotherapy, the duration between image acquisition and treatment must be as short as possible, since the patient is already in treatment position. Furthermore, a high deformation resolution is required to allow the deformation grid to closely match the shape of the rigid structures in the images. This makes our matrix-free registration algorithm an ex-

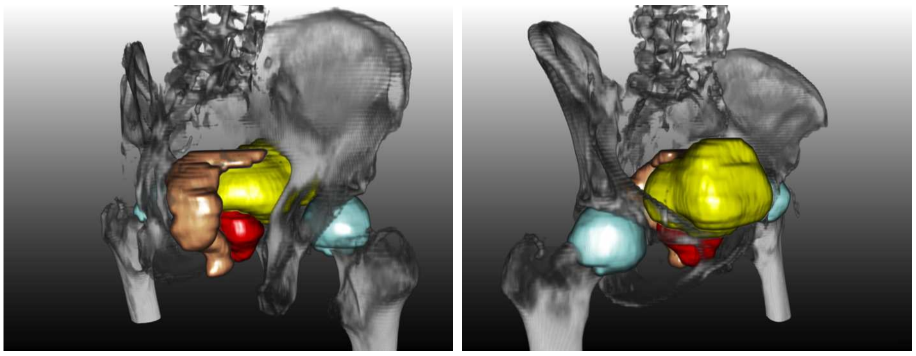

(a) Organ locations in pelvis (back) (b) Organ locations in pelvis (front)

Figure 7.2.: Three-dimensional rendering of planning CT with organ locations of bladder (yellow), prostate CTV (red), femoral heads (blue) and rectum (orange) in the pelvic area, (a): back view, (b): front view. The proximity of organs with very different deformation properties requires special consideration during registration.

cellent candidate, as it allows for registration of large images and deformation resolutions which could otherwise not be processed with matrix-based methods.

Acknowledgments and related publications. The evaluation results presented in this chapter are published in [KDPH16*; KDH+15*]. This chapter contains an extended version with an advanced mathematical description of the method. We especially thank Alexander Derksen, Nils Papenberg and Benjamin Haas for their collaboration on this project and Luís Vasco Luoro, Nuno Pimentel and Joep Stroom from Champalimaud Centre for the Unknown, Lisbon, Portugal for providing the evaluation datasets and delineations.

7.1. Registration framework

In this section, we present the mathematical framework for integrating the local rigidity constraint into the deformable image registration algorithm.

7.1.1. Related work

Different approaches for including further anatomical knowledge in registration algorithms have been previously pursued in the field of radiotherapy. Some approaches incorporate improved deformation models, using bio-mechanical models based on finite-element approaches [BHE+08; ZKL+12; WS15] to achieve more realistic deformation behavior. Others rely on contour-guided registration [GDW+13] or include shape based regularization [WS15]. All of these works show in their respective applications that adding additional information to the deformation model generally improves registration accuracy.

Local rigidity has been previously integrated into deformation models in the field of medical image registration in various ways, based on a B-spline deformation model [SKP07; RWU+14] and in a variational framework [HHM09; Mod08]. Additionally, B-spline-based approaches have been proposed with focus on radiation therapy [KKL+13; GCP+09], showing that applications in this field can benefit from locally rigid deformations.

In this chapter, we present an extension of a theoretical framework by [HHM09] to three-dimensional real-world data in a radiotherapy application. The method models local rigidity as a hard constraint, i.e., it is enforced. We embed this local rigidity constraint in our fast and efficient matrix-free registration algorithm described in Chapter 3 in order to process clinically relevant image sizes and deformation resolutions and to achieve clinically acceptable runtimes.

7.1.2. Local rigidity

As basis of the registration algorithm, we rely on the variational framework presented in Chapter 2. Since CBCT images do not exhibit a normalized range of intensity values such as CT images, the radiotherapy registration problem is treated as a multi-modal setting. We therefore rely on the normalized gradient fields (NGF) distance measure $\mathcal{D}^{\mathrm{NGF}}$ from Section 2.3.1.

As given in Section 2.3, we then obtain the optimization problem

$$\min_{\varphi:\Omega_{\mathcal{R}}\to\mathbb{R}^d} \mathcal{J}(\varphi) = \mathcal{D}^{\mathrm{NGF}}(\mathcal{R}, \mathcal{T}(\varphi)) + \alpha\mathcal{S}(\varphi). \tag{7.1}$$

However, this model does not include any local knowledge about the anatomy. Therefore, non-linear deformations can be applied to bones and other rigid structures, which leads to physically implausible results.

We therefore add an additional rigidity constraint to the registration framework as follows: Given k segmentations of rigid structures in the reference image $\mathcal{R}$, we define the domains $\Sigma_k, k = 1, \ldots, M$ of the segmentations as disjoint sub-domains of the reference image domain, i.e., $\Sigma_k \subset \Omega_{\mathcal{R}}$. Each of these areas can each exhibit a different rigid motion.

To integrate local rigidity into the optimization problem (7.1), we introduce auxiliary optimization variables in the form of three-dimensional rotation matrices $Q(\theta_k) : \mathbb{R}^3 \to \mathbb{R}^{3\times3}$, depending on the three rotation angles $\theta_k := (\theta_k^x, \theta_k^y, \theta_k^z)^\top \in \mathbb{R}^3$, and translation vectors $b_k := (b_k^x, b_k^y, b_k^z)^\top \in \mathbb{R}^3$.

We now substitute a rigid transformation in the deformation φ for all points that are inside any of the sub-domains $\Sigma_k, k = 1, \ldots, M$, such that

$$\varphi^\phi := \begin{pmatrix} \varphi_0 \\ \varphi_1 \\ \vdots \\ \varphi_M \end{pmatrix} = \begin{pmatrix} \varphi_0 \\ Q(\theta_1)x + b_1 \\ \vdots \\ Q(\theta_M)x + b_M \end{pmatrix},$$

where

$$\varphi_0(x) := \varphi(x) \quad \text{if } x \notin \Sigma_k \ \forall\ k \in \{1, \ldots, M\}$$

and

$$\varphi_k(x) := Q(\theta_k)x + b_k \quad \text{if } x \in \Sigma_k,\ k \in \{1, \ldots, M\}.$$

Substituting this in the optimization problem (7.1), we obtain an unconstrained optimization problem

$$\min_{\varphi_0, \theta_{1,\ldots,k}, b_{1,\ldots,k}} \mathcal{J}(\varphi^\phi) = \mathcal{D}^{\mathrm{NGF}}(\mathcal{R}, \mathcal{T}(\varphi^\phi)) + \alpha \mathcal{S}(\varphi^\phi),$$

where the deformation of each rigid region is controlled by the transformation parameters θ_k and b_k.

Note that $\Sigma_k \subset \Omega_{\mathcal{R}}$, i.e., the rigid regions are defined on the reference image domain. We therefore choose the planning CT image as reference image and the CBCT image as template image, since manual segmentations of all important structures are readily available on the CT from the planning process in the radiotherapy setting. As the deformation maps a coordinate location in the reference image domain to the template image domain, when treated as sets of points, the segmentations can directly be propagated from the CT to the CBCT image.

Derivatives and discretization. The extended approach can be discretized analogously to Section 2.4. The discretized optimization problem is then optimized using a L-BFGS optimization scheme as described in Section 2.5.2 and embedded in a coarse-to-fine multi-level scheme, as described in Section 2.5.6.

On each level, the points $\mathbf{y}^{\mathrm{c}}$ need to be determined that lie within any of the Σ_k on the deformation grid and are thus constrained by the local rigidity. Assuming that we have n rigid deformation grid points within any of the Σ_k, and $d = 3$, it holds $\mathbf{y}^{\mathrm{c}} \in \mathbb{R}^{3n}$. In return, there are $\mathbf{y}^{\mathrm{u}} \in \mathbb{R}^{3(\bar{m}^{\mathrm{y}} - n)}$ unconstrained points. Since the reference image domain remains unchanged during the registration, these points only need to be determined once at the beginning of each level in the multi-level scheme.

To obtain the new parameter set for optimization, all constrained points $\mathbf{y}^{\mathrm{c}}$ in the deformation $\mathbf{y} \in \mathbb{R}^{3\bar{m}^{\mathrm{y}}}$ must now be replaced by six rigid transformation parameters θ_k, b_k for each rigid region. This results in a vector

$$\mathbf{y}^\phi := \begin{pmatrix} \mathbf{y}^{\mathrm{u}} \\ \theta_1 \\ b_1 \\ \vdots \\ \theta_M \\ b_M \end{pmatrix} \in \mathbb{R}^{\bar{m}^\phi}, \tag{7.2}$$

with

$$\bar{m}^\phi := 3(\bar{m}^{\mathrm{y}} - n) + 6M,$$

consisting of deformations for all unconstrained grid points and six rigid transformation parameters each for all M rigid regions.

A function that maps from $\mathbf{y}^{\phi}$ to a full deformation $\mathbf{y}$ can now be defined as $\phi : \mathbb{R}^{\bar{m}^{\phi}} \to \mathbb{R}^{3\bar{m}^{\mathrm{y}}}$ with

$$\phi(\mathbf{y}^{\phi}) := \mathbf{y}.$$

This function computes transformed deformation grid points $Q(\theta_k)\mathbf{x}_i^{\mathrm{y}} + b_k$ at points that lie within the rigid areas of the deformation grid and uses the existing deformation values $\mathbf{y}^{\mathrm{u}}$ for unconstrained points, such that a full deformation $\mathbf{y} \in \mathbb{R}^{3\bar{m}^{\mathrm{y}}}$ is obtained.

We can now write the discretized, unconstrained objective function as

$$J(\mathbf{y}^{\phi}) = D^{\mathrm{NGF}}(\phi(\mathbf{y}^{\phi})) + \alpha S(\phi(\mathbf{y}^{\phi}))$$

in analogy to (2.25), such that the optimization problem becomes

$$\min_{\mathbf{y}^{\phi} \in \mathbb{R}^{\bar{m}^{\phi}}} J(\mathbf{y}^{\phi}).$$

For minimization of this objective function using the previously presented methods again its derivatives are required, as described in Chapter 3. In comparison to (3.1), the local rigidity constraint concatenates another function to the distance measure and regularizer evaluation, such that the distance measure evaluation can now be written as

$$D(\mathbf{y}^{\phi}) = \psi(r(T(P(\phi(\mathbf{y}^{\phi}))))).$$

Furthermore, following Section 3.1, this additional function appends another Jacobian matrix $\frac{\partial \phi}{\partial \mathbf{y}} \in \mathbb{R}^{3\bar{m}^{\mathrm{y}} \times \bar{m}^{\phi}}$ to the derivative computation (3.3), i.e,

$$\nabla D(\mathbf{y}^{\phi}) = \left(\frac{\partial \psi}{\partial r} \frac{\partial r}{\partial T} \frac{\partial T}{\partial P} \frac{\partial P}{\partial \phi} \frac{\partial \phi}{\partial \mathbf{y}^{\phi}} \right)^{\top} \in \mathbb{R}^{\bar{m}^{\phi} \times 1},$$

and similarly for the curvature regularizer. Since the new Jacobian matrix is multiplied from the right side, the left part of the gradient can be reused (see Chapter 3).

Multi-level. When utilizing the local rigidity scheme in a multi-level registration as described in Section 2.5.6, further consideration has to be given to the prolongation of the final result of the optimization $\mathbf{y}^{\phi}$ on a coarse level to the next finer level. Since $\mathbf{y}^{\phi}$ does no longer represent a full deformation, as shown in (7.2), it cannot simply be interpolated to the next finer level. Here, using $\phi(\mathbf{y}^{\phi})$, a full deformation has to be created first from the final optimization result. Then, this full deformation can be prolongated to the next finer level as usual. On the fine level, the sets of deformation grid points that lie within the rigid areas have to be determined anew, and a new vector $\mathbf{y}^{\phi}$ needs to be created from the prolongated deformation. The rigid parameters θ_k, b_k, on the other hand, can simply be adopted unchanged from the coarse level, as they do not depend on the deformation resolution.

Since the local rigidity constraint acts on the deformation grid, it is important that a sufficiently high number of deformation grid points is chosen, in order to closely match the boundaries of the rigid structures on the image. Therefore, in the following we will use the same deformation and image resolution on each level, with $m_i^{\mathrm{y}} = m_i + 1$.

As shown in Chapter 5, such high resolutions cannot be computed with matrix-based algorithms within reasonable runtime, or not all due to memory constraints, making the matrix-free algorithm a suitable choice.

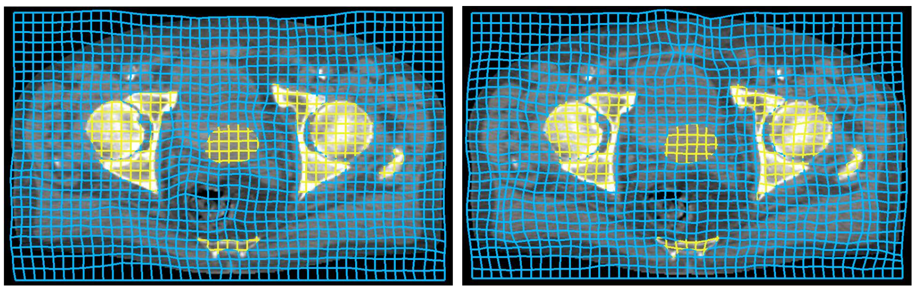
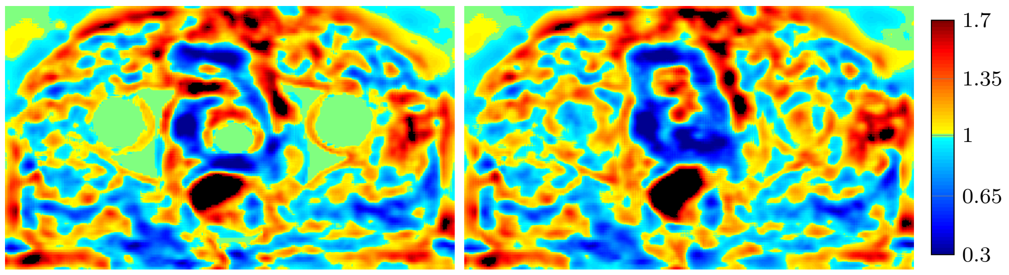

(a) with local rigidity (b) unconstrained

Figure 7.3.: Single axial slice of the deformation for registrations with and without local rigidity constraints, using the images shown in Figure 7.1. Contours of bones and prostate are shown in yellow, (a): deformation with local rigidity for bones and prostate, (b): deformation for an unconstrained registration. While bones and prostate are kept rigid when using local rigidty, non-linear deformations can be observed in the unconstrained registration.



(a) with local rigidity (b) unconstrained

Figure 7.4.: Single axial slice of the Jacobian determinant of the deformation in Figure 7.3. Color indicates volume change, green areas correspond to $\det(\nabla \mathbf{y}_i) = 1$. While the locally rigid areas can be easily identified in (a), undesirable volume-changing deformations occur in (b), especially in the prostate area.

7.2. Example dataset

In order to analyze the effect of the rigidity constraint on the resulting deformation, we evaluated the registration with local rigidity on the exemplary dataset shown in Figure 7.1 and visualized the results. We registered the CBCT image to the CT with and without local rigidity constraint. We modeled the prostate, the femoral bones and the pelvic bones as four individual rigid structures.

The resulting deformation is visualized in Figure 7.3. In Figure 7.3(a), using the local rigidity constraint, the rigid areas (yellow) exhibit a regular grid structure. In contrast, the unconstrained deformable registration in Figure 7.3(b) results in unrealistic non-linear deformations of bones and prostate.

The differences become even more apparent when visualizing the Jacobian determinant of the deformation $\det(\nabla \mathbf{y}_i)$ [KD04] (Figure 7.4). For volume preserving deformations, which includes rigid deformations, it holds $\det(\nabla \mathbf{y}_i) = 1$. Values smaller than one correspond to a compression of the deformation field, while larger values correspond to an expansion [RMBJ03]. In Figure 7.4(a), the rigid regions can clearly be seen as uniform areas under the local rigidity constraint. In contrast, when performing an unconstrained registration, non-linear deformations can be observed in the area of bones and prostate, as shown in Figure 7.4(b). Especially for the bones, these deformations are physically highly implausible, which can reduce the physician's confidence in the automatic registration process.

7.3. Evaluation on clinical datasets

For a quantitative evaluation of the deformable registration with local rigidity constraints, we examined the approach on clinical datasets with gold-standard annotations.

7.3.1. Method

Datasets. We used a dataset of ten prostate cancer cases from clinical routine with images of the male pelvic region. Every case consists of a planning CT image, acquired using a diagnostic CT scanner, and a CBCT image, which was acquired using the On-Board Imager of a Varian TrueBeam radiotherapy system, similar to those shown in Figure 7.1. The CT images have an average size of $512 \times 512 \times 238$ voxels, while the CBCT images are slightly smaller in z-direction with an average size of $512 \times 512 \times 81$ voxels. In six cases (cases $0, 2-6$), a rectal balloon was utilized to restrict prostate movement [TMU+01]. Four cases were treated without a rectal balloon.

During treatment plan creation, the CT images were manually contoured. Each CT image is annotated with contours of femoral heads, pelvic bones, prostate (CTV) and bladder. For evaluation, each CBCT image was manually supplemented with contours of femoral heads, pelvic bones, prostate and bladder by a radiation oncologist, which serve as a gold standard during evaluation.

Evaluation. Due to the general availability of contours on the CT images in this clinical setting, we used the CT image as reference image, providing the structures for the local rigidity constraint, and the CBCT image as template image, as also described in Section 7.1.2. After the registration, the contours for all structures were then propagated from the CT to the CBCT image using the obtained deformation.

We compared the proposed approach to unconstrained matrix-free methods as well as a commercial software package. Initially, for all methods, CT and CBCT images were registered using a rigid pre-alignment (RIG). Based on this rigid pre-alignment, the images were registered using different deformable registration algorithms. First, we evaluated the deformable image registration algorithm from the Varian SmartAdapt software (VSA). This method utilizes a demons-based algorithm (see Section 2.2.3) [WDO+05; RPH+15]. Second, we registered the images using the unconstrained matrix-free registration algorithm in Chapter 3 (MFC0). Third, we employed the constrained registration algorithm as described in this chapter with two configurations: Using only bones as rigid structures (MFC1) and bones as well as prostate as rigid structures (MFC2).

For each registration, we compared the propagated contours on the CBCT images with the gold standard delineations using the *Dice similarity coefficient* (DSC) [Dic45]. Given two finite sets A and B, the DSC is defined as

$$\text{DSC}(A, B) := \frac{2|A \cap B|}{|A| + |B|},$$

where $|\cdot|$ is the set cardinality. The DSC measures the spatial overlap between the propagated structures and the gold standard delineations and ranges from 0 to 1, with 1 corresponding to perfect agreement. While the DSC is widely used for reporting registration results, its validity is limited when used as the only evaluation metric [Roh12]. Therefore, we additionally computed the *Hausdorff distance* (HD) between propagated and gold standard contours, which is defined as [HRK93]

$$\text{HD}(A, B) := \max(h(A, B), h(B, A)),$$

where

$$h(A, B) := \max_{a \in A} \min_{b \in B} \|a - b\|.$$

For all points, the distance from a point in one set to the closest point in the other set is determined. The HD then gives the maximum of these distances, and becomes 0 for perfect agreement.

In addition to the accuracy of the propagated contours, physical plausibility of the deformation is important. Therefore, we used the Jacobian determinant of the deformation $\det(\nabla \mathbf{y}_i)$, which was already introduced in Section 7.2, to evaluate the deformations. For bones and prostate, which were used as rigid areas, we evaluated the average value of the Jacobian determinant for all algorithms.

Besides non-linear deformation of bones, we used the Jacobian determinant to detect so-called grid foldings in the deformation grid. These correspond to a mapping that is not topology preserving and are indicated by a non-positive determinant $\det(\nabla \mathbf{y}_i) \leq 0$ [KD04]. We evaluated the Jacobian determinant using a discretization of 64 tetrahedrons per grid cell as described in [KD04], which is essential, since only a proper discretization ensures that all grid foldings can be detected [HM04]. In most cases, a non-positive Jacobian determinant corresponds to physically implausible deformations, especially in tissue or bone areas. Exceptions concern in particular fluids or gases, such as fluids within the bladder or air within the rectum or surrounding the body.

To allow a fair runtime comparison to the commercial VSA registration, we performed all evaluations on a Varian Medical Systems workstation with Microsoft Windows 7, with two six-core Intel Xeon E5-2620 processors running at 2.0 GHz and 32 GB of RAM.

Parameterization. In order to determine a suitable parameterization for this application, we performed a parameter search for the regularizer weight α, as well as for the NGF edge filtering parameters τ and ϱ. We examined the registration results for all combinations of $\alpha = [1, 5, 10, 20, 50]$, $\tau = [1, 5, 10]$, $\varrho = [1, 5, 10]$. Based on this, we then chose $\alpha = 10$, $\tau, \varrho = 5$ for all evaluations as the parameters that obtained the best mean DSC value. However, we found that slightly varying the parameters did not yield a large difference in the results, such that a coarse adjustment of the parameters seems sufficient in practice.

For the multi-level computations, we chose one quarter of the image size in each dimension as finest level and additionally computed two coarser levels. The same deformation and image resolution was chosen, such that $m_i^{\mathrm{y}} = m_i + 1$ on each level in order to achieve consistency of rigid areas in the deformation field and the corresponding structures in the image (see Section 7.1.2). This resulted in an average deformation size of $105 \times 65 \times 41$ grid points on the finest level.

7.3.2. Results

We performed registrations for all ten cases and the five algorithms RIG, VSA, MFC0, MFC1 and MFC2.

Segmentation overlap. The DSC values of the propagated contours are shown in Table 7.1, while the HD values are given in Table 7.2. Averaged over all cases and contours, the rigid pre-alignment RIG achieved an average DSC of 0.79 ± 0.11 with a HD of $12.65\,\mathrm{mm} \pm 6.83\,\mathrm{mm}$.

In comparison, the unconstrained registration with VSA resulted in a DSC of 0.86 ± 0.07 and a HD of $10.22\,\mathrm{mm} \pm 6.62\,\mathrm{mm}$, while MFC0 obtained a DSC of 0.87 ± 0.06 with a HD of $8.74\,\mathrm{mm} \pm 5.95\,\mathrm{mm}$, averaged over all contours and cases. Note that *larger* DSC values and *smaller* HD values are better.

The approaches with local rigidity constraints achieved comparable results. Here, MFC1 obtained an average DSC of 0.87 ± 0.07 and a HD of $8.91\,\mathrm{mm} \pm 5.89\,\mathrm{mm}$, while MFC2 resulted in a DSC of 0.87 ± 0.06 with a HD of $8.73\,\mathrm{mm} \pm 6.02\,\mathrm{mm}$.

For comparison of the results, we performed a number of statistical tests: We compared the DSC results of the rigid pre-alignment RIG with VSA and MFC0–MFC2, and the DSCs for VSA with MFC0–MFC2. For the analysis a two-sided paired-sample t-test was chosen, corresponding to the assumption that the differences in the DSC values of different algorithms are normally distributed. Additionally, to test whether the rectal balloons, present in six of ten cases, had an influence on the results, a two-sided two-sample t-test was performed, comparing the DSC values for cases with rectal balloon and those without for VSA and MFC0–MFC2. Bonferroni correction was applied for the definition of significance ($p < 0.0045$).

Compared with the rigid pre-alignment RIG, all other algorithms showed a significant increase in DSC overlap with $p < 10^{-4}$ with 95% CI: 0.040–0.092 for VSA, $p < 10^{-5}$

ID	Prostate					Bladder					Right femoral head				
	RIG	VSA	MFC0	MFC1	MFC2	RIG	VSA	MFC0	MFC1	MFC2	RIG	VSA	MFC0	MFC1	MFC2
0	0.79	0.81	**0.85**	0.84	0.82	0.84	**0.91**	0.90	**0.91**	**0.91**	0.94	0.94	**0.96**	**0.96**	**0.96**
1	0.80	**0.89**	0.85	0.85	0.85	0.83	**0.91**	0.86	0.86	0.86	0.89	0.94	0.94	**0.95**	**0.95**
2	0.53	0.71	**0.75**	0.71	**0.75**	0.66	**0.87**	0.86	0.85	0.85	0.89	**0.91**	**0.91**	**0.91**	**0.91**
3	0.74	0.70	**0.80**	**0.80**	0.79	0.76	0.85	**0.89**	**0.89**	**0.89**	0.89	0.92	**0.93**	**0.93**	**0.93**
4	0.70	0.73	**0.81**	**0.81**	**0.81**	0.74	**0.82**	0.80	0.80	0.80	0.92	**0.94**	**0.94**	**0.94**	**0.94**
5	0.82	0.82	0.85	0.85	**0.87**	0.82	0.87	**0.92**	**0.92**	**0.92**	0.89	0.93	**0.94**	**0.94**	**0.94**
6	**0.84**	0.81	**0.84**	**0.84**	**0.84**	0.83	**0.88**	0.84	0.84	0.84	**0.90**	0.89	**0.90**	**0.90**	**0.90**
7	0.71	0.82	**0.85**	**0.85**	**0.85**	0.49	0.75	**0.78**	**0.78**	**0.78**	0.91	0.95	0.97	**0.98**	**0.98**
8	0.70	**0.86**	0.77	0.77	0.82	0.63	0.81	**0.83**	**0.83**	0.80	0.93	**0.95**	**0.95**	**0.95**	**0.95**
9	0.76	**0.81**	0.79	0.80	0.79	0.80	**0.89**	**0.89**	**0.89**	**0.89**	0.90	0.94	**0.95**	**0.95**	**0.95**
Avg.	0.74	0.80	**0.82**	0.81	**0.82**	0.74	**0.86**	**0.86**	**0.86**	0.85	0.91	0.93	**0.94**	**0.94**	**0.94**
Std.	0.09	0.06	**0.04**	0.05	**0.04**	0.11	0.05	**0.04**	0.05	0.05	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**

Table 7.1.: Comparison of final registration accuracy between different forms of matrix-free deformable registration, rigid registration, and the commercial VSA algorithm. Dice similarity coefficient values for propagated structures of ten prostate CT/CBCT cases, evaluated on CBCT gold standard contours of prostate, bladder and right femoral head. Higher values indicate better agreement with the gold standard annotations. RIG: rigid pre-alignment, VSA: Varian SmartAdapt deformable registration, MFC0: matrix-free unconstrained registration, MFC1: registration with rigid bones, MFC2: registration with rigid bones and prostate, Avg., Std.: average, standard deviation over all cases. The deformable registration approaches achieve better DSC values than the rigid alignment; MFC0-MFC2 achieve DSC values in the same range as VSA.

ID	Prostate					Bladder					Right femoral head				
	RIG	VSA	MFC0	MFC1	MFC2	RIG	VSA	MFC0	MFC1	MFC2	RIG	VSA	MFC0	MFC1	MFC2
0	7.2	**5.5**	5.8	6.0	**5.5**	11.1	8.5	**6.0**	**6.0**	8.2	5.7	**4.0**	**4.0**	**4.0**	**4.0**
1	7.9	**6.0**	6.5	6.5	6.5	11.2	**8.0**	8.8	8.3	8.5	5.5	**4.2**	4.5	10.4	4.5
2	12.8	**10.9**	14.0	14.0	14.0	18.2	13.1	10.0	11.8	**9.8**	8.3	**6.0**	7.1	7.5	7.5
3	7.2	10.0	7.5	**7.0**	7.5	13.1	14.8	11.9	**9.4**	**9.4**	4.5	5.3	**4.0**	**4.0**	4.1
4	11.1	10.6	**8.4**	**8.4**	**8.4**	26.4	**23.5**	24.5	24.5	24.8	5.0	**4.4**	4.5	**4.4**	**4.4**
5	**6.7**	18.9	8.1	8.1	8.8	9.4	19.3	**6.0**	**6.0**	8.1	6.4	**4.2**	4.4	4.4	4.4
6	**6.6**	9.3	8.8	8.6	7.0	11.9	**11.3**	12.1	12.1	13.4	**5.0**	6.4	5.5	6.0	5.5
7	11.3	13.5	9.3	9.7	**9.2**	34.4	32.5	31.3	**31.2**	31.6	5.0	3.8	3.6	3.3	**2.9**
8	10.3	6.5	7.5	7.5	**6.2**	14.4	15.9	**10.0**	**10.0**	10.2	5.0	5.5	**4.4**	4.5	**4.4**
9	12.2	10.8	11.1	11.1	**10.8**	9.8	9.9	**8.7**	**8.7**	**8.7**	4.7	4.1	**4.0**	**4.0**	**4.0**
Avg.	9.3	10.2	8.7	8.7	**8.4**	16.0	15.7	12.9	**12.8**	13.3	5.5	4.8	**4.6**	5.2	**4.6**
Std.	**2.4**	4.0	**2.4**	**2.4**	2.5	8.2	**7.7**	8.3	8.3	8.2	1.1	**0.9**	1.0	2.2	1.2

Table 7.2.: HD values in mm for propagated structures of ten prostate CT/CBCT cases, evaluated on CBCT gold standard contours of prostate, bladder and right femoral head. Lower values indicate better agreement with the gold standard annotations. See Table 7.1 for further description.

with 95% CI: 0.050–0.102 for MFC0, $p < 10^{-5}$ with 95% CI: 0.050–0.100 for MFC1 and $p < 10^{-5}$ with 95% CI: 0.051–0.102 for MFC2.

Comparing the results of MFC0–MFC2 with VSA, a statistical significant difference in DSC overlap could not be shown with $p = 0.18$ with 95% CI: -0.005–0.024 for MFC0, $p = 0.21$ with 95% CI: -0.005–0.023 for MFC1 and $p = 0.10$ with 95% CI: -0.002–0.023 for MFC2.

Furthermore, comparing DSC values of cases with rectal balloon to cases without rectal balloon did not show a statistical significant difference with $p = 0.36$ with 95% CI: -0.082–0.031 for VSA, $p = 0.98$ with 95% CI: -0.052–0.051 for MFC0, $p = 0.98$ with 95% CI: -0.052–0.051 for MFC1 and $p = 0.95$ with 95% CI: -0.050–0.047 for MFC2.

Displacement regularity. As discussed in the previous section, certain aspects of the computed deformation need to be considered. First, the deformation should not fold onto itself. Second, physically unlikely – but not entirely impossible – deformations, such as non-linear deformations of bones, must be avoided.

In Table 7.3, we list the percentage of grid foldings within the deformation fields of all algorithms for prostate, bladder and the complete body outline. Additionally, in Table 7.4, we evaluated the average deviation of the Jacobian determinant from a value of one within the prostate and right femoral head contours, exemplarily chosen as two structures which are predisposed for local rigidity constraints.

As can be seen in Table 7.3, VSA generates grid foldings within the body contour in all ten cases, while MFC0 and MFC1 exhibit a small number of grid foldings in three cases, MFC2 in only two cases. Within the prostate, VSA shows grid foldings in three cases, while MFC0–MFC2 do not show any grid foldings in this area. Similarly for bladder, VSA computes deformations with grid foldings in five cases, while deformations of MFC0 and MFC1 only contain grid foldings in a single case and MFC2 computes no deformations with grid foldings within the bladder.

The values in Table 7.4 show the expected behavior. For the unconstrained algorithms VSA and MFC0, values smaller and larger than one are reported for prostate and right femoral head, corresponding to a contraction or expansion of the deformation field, which relates to physically implausible deformations. As expected, MFC1 and MFC2 with local rigidity exhibit $\det(\nabla \mathbf{y}_i) = 1$ for all points within the bone structure. Additionally, MFC2 also shows rigid deformations within the prostate as desired.

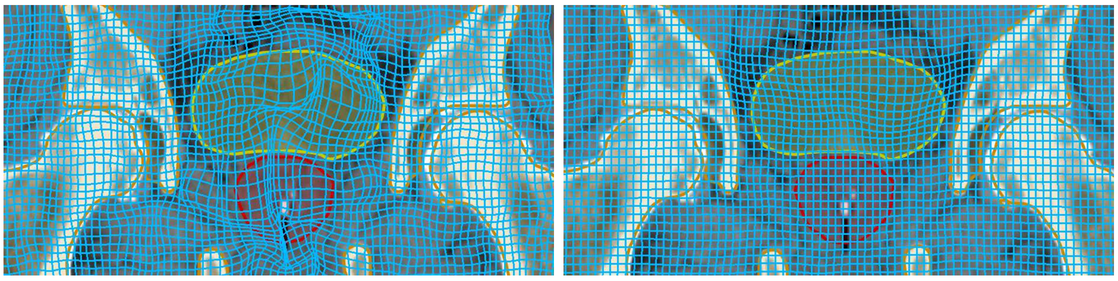
Case 4 shows especially large numbers of grid foldings for VSA in prostate and bladder. Details of the deformation field are visualized in Figure 7.5. Here, the grid foldings within prostate and the bladder wall can be clearly seen in Figure 7.5(a). Additional, non-linear deformations inside the bones can also be observed. In comparison, we present the deformation field of MFC2 for this case in Figure 7.5(b). The deformation is smooth within bladder and surrounding tissue while bones and prostate are deformed rigidly.

ID	Prostate				Bladder				Body			
	VSA	MFC0	MFC1	MFC2	VSA	MFC0	MFC1	MFC2	VSA	MFC0	MFC1	MFC2
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.19**	**0.01**	**0.02**	**0.04**
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.01**	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	**1.61**	0.00	0.00	0.00	**0.13**	0.00	0.00	0.00
3	**5.46**	0.00	0.00	0.00	**0.04**	0.00	0.00	0.00	**0.18**	0.00	0.00	0.00
4	**11.97**	0.00	0.00	0.00	**4.21**	0.00	0.00	0.00	**0.45**	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	**11.43**	0.00	0.00	0.00	**0.15**	0.00	0.00	0.00
6	**0.68**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.10**	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	**0.06**	0.00	0.00	0.00	**0.05**	**0.04**	**0.03**	**0.04**
8	0.00	0.00	0.00	0.00	0.00	**0.82**	**0.82**	0.00	**0.07**	**0.00***	**0.00***	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.03**	0.00	0.00	0.00
Avg.	**1.81**	0.00	0.00	0.00	**1.74**	**0.08**	**0.08**	0.00	**0.14**	**0.01**	**0.01**	**0.01**
Std.	**3.75**	0.00	0.00	0.00	**3.48**	**0.25**	**0.25**	0.00	**0.12**	**0.01**	**0.01**	**0.02**

Table 7.3.: Percentage of deformation grid foldings per organ and case for registration results of ten prostate CT/CBCT cases. *: contains a small amount of grid foldings which rounds to zero. See Table 7.1 for further abbreviations. While VSA generates grid foldings for all cases within the body contour and large amounts of grid foldings within the prostate and bladder for some cases, MFC0–MFC2 only exhibit a low percentage of grid foldings for a smaller number of cases. This indicates that the proposed algorithm is an effective method for greatly reducing physically implausible folding deformations.

ID	Prostate				Right femoral head			
	VSA	MFC0	MFC1	MFC2	VSA	MFC0	MFC1	MFC2
0	0.22	0.08	0.09	0.00	0.13	0.05	0.00	0.00
1	0.17	0.15	0.13	0.00	0.16	0.05	0.00	0.00
2	0.21	0.26	0.29	0.00	0.13	0.06	0.00	0.00
3	0.20	0.07	0.07	0.00	0.13	0.09	0.00	0.00
4	0.26	0.09	0.09	0.00	0.15	0.05	0.00	0.00
5	0.24	0.11	0.14	0.00	0.13	0.05	0.00	0.00
6	0.24	0.12	0.14	0.00	0.13	0.03	0.00	0.00
7	0.21	0.08	0.09	0.00	0.14	0.06	0.00	0.00
8	0.30	0.33	0.32	0.00	0.18	0.06	0.00	0.00
9	0.19	0.11	0.09	0.00	0.15	0.07	0.00	0.00
Avg.	0.23	0.14	0.15	0.00	0.14	0.06	0.00	0.00
Std.	0.04	0.09	0.09	0.00	0.02	0.01	0.00	0.00

Table 7.4.: Average deviation of the Jacobian determinant of the deformation from a value of one, i.e., $\frac{1}{\bar{m}^{\mathrm{y}}}\sum_{i=1}^{\bar{m}^{\mathrm{y}}} |\det(\nabla \mathbf{y}_i) - 1|$. Evaluated for the prostate and right femoral head for each deformation in the registration of ten prostate CT/CBCT cases. See Table 7.1 for abbreviations. Due to the implemented hard constraints, MFC1 and MFC2 keep the designated areas perfectly rigid, while the unconstrained algorithms VSA and MFC0 exhibit less realistic nonlinear deformations in bones and prostate.



(a) VMS (b) MFC2

Figure 7.5.: Single coronal slice of the deformation for case 4, (a): computed with the unconstrained commercial software package VMS, (b): computed with the proposed approach MFC2 with rigidity for bones and prostate. The image is overlaid by deformation grid and delineations of bladder (yellow), prostate (red) and bones (orange). The VMS deformation shows two properties that are problematic in clinical practice: Non-linear deformation of bones and grid foldings within critical areas such as bladder wall and prostate. In contrast, the MFC2 deformation shows the desired rigid behavior in bones and prostate and an overall smooth deformation.

Runtime. The measured runtimes for all evaluated cases and algorithms are given in Table 7.5. On average, both unconstrained algorithms achieve comparable runtimes with 10.7 s for VSA and 10.5 s for MFC0. Incorporating the local rigidity constraints increases the average runtime to 26.1 s for MFC1. Additionally adding the prostate as a rigid structure further increases the runtime to 31.1 s on average for MFC2.

7.3.3. Discussion

All evaluated results show a significant improvement of DSC values in comparison with a rigid pre-registration, confirming the need for deformable registration in this application scenario. A statistically significant difference in DSC values for VSA and MFC0–MFC2 could not be shown and the results in terms of DSC and HD are in a comparable range. However, the resulting deformations of MFC0–MFC2 are smoother and exhibit far less unphysical deformation grid foldings, especially in critical areas such as the bladder wall or prostate.

In these areas deformation grid foldings are particularly dangerous: While the prostate is the clinical target volume in this case, and as such receives a very high radiation dose, its surroundings and the bladder are spared from radiation. This leads to high dose gradients in the prostate region. Grid foldings in this area can thus potentially cause critical errors in propagated structures or evaluated doses. While maintaining the same accuracy, the results of MFC0–MFC2 do not contain grid foldings in these critical areas in most cases, which makes them preferable over VSA.

While there exist approaches to completely avoid grid foldings in the utilized registration framework [RPH+17], such foldings might be tolerable or even realistic in areas with fluid

ID	VSA	MFC0	MFC1	MFC2
0	10.1	10.9	25.5	30.8
1	13.8	12.8	33.3	40.1
2	11.4	10.1	26.6	30.8
3	11.0	11.4	28.5	32.8
4	9.4	9.6	23.1	28.7
5	10.4	10.4	24.5	34.7
6	8.9	8.0	19.9	19.4
7	11.8	10.5	25.5	33.6
8	9.3	10.5	26.0	27.3
9	10.6	11.1	28.6	33.2
Avg.	10.7	10.5	26.1	31.1
Std.	1.4	1.3	3.6	5.4

Table 7.5.: Runtime in seconds for the registration of ten prostate CT/CBCT cases for the four evaluated deformable registration algorithms. See Table 7.1 for abbreviations. The unconstrained algorithms VMS and MFC0 achieve comparable runtimes, while utilizing the local rigidity constraints increases the registration runtime (MFC1, MFC2).

motion, such as the bladder contents or areas with air or missing correspondence such as bowel contents.

As in practice, non-linear deformations of bones are unlikely, the confidence in deformations computed by VSA or MFC for appropriately solving the given registration problem is limited. In comparison, MFC1 and MFC2 compute physically plausible deformations of bones due to the introduced local rigidity constraints, while still maintaining DSC and HD values in the same range as VSA and MFC0, such that MFC1 and MFC2 are preferable in this setting.

Comparing MFC1 and MFC2, there is no substantial difference in keeping the prostate rigidly constrained. Both methods achieve comparable results in terms of DSC and HD values as well as displacement regularity.

The overall achieved DSC values are comparable to DSC values documented in the literature. In [KKL+13], average DSC values of 0.7 for prostate and bladder were reported. In [TPB+11], VSA was also evaluated for registration in the pelvic region and obtained average DSC values of 0.80 for prostate and 0.73 for bladder.

In some cases, large bladder volume changes between CT and CBCT image show a limitation of the presented approach. Excessive volume differences cannot fully be matched by the registration algorithm. Here, specialized algorithms focused on matching the bladder as presented in [DKMH16*; DKM16*] can potentially improve the registration result. When utilized without additional constraints, they may, however, lead to further deformation grid foldings within the bladder.

While the unconstrained algorithms VSA and MFC0 obtain comparable runtimes, adding the local rigidity constraints results in longer computation times. This is mostly due to the additionally added computations required for the function $\phi(\mathbf{y}^{\phi})$. Furthermore, we observed a slower convergence of the optimization scheme, potentially relating to the additionally added constraints, where large areas with a great influence on distance measure and regularizer are controlled by only six rigid parameters each. However, within a clinical setting, the average runtimes of MFC1 and MFC2 of 26.1 s and 31.1 s are potentially still feasible in most cases, especially when compared with other approaches using local rigidity such as [KKL+13], where runtimes between 30 and 80 min are reported for registrations in the pelvic area.

7.4. Summary and conclusion

We presented an extension of the matrix-free deformable registration framework described in Chapter 3 with an additional local rigidity constraint for an application in radiotherapy. The approach keeps selected areas rigid using a hard constraint while allowing to re-use the unconstrained optimization framework presented earlier. This model guarantees the rigidity of selected structures, while still allowing tissue and organs to deform in a non-linear way.

Evaluated on ten male pelvis cases with gold standard delineations, the proposed approach achieved comparable accuracy in comparison with the unconstrained version and a commercially used algorithm. Furthermore, the locally rigid algorithm showed far less grid foldings, especially in critical areas such as the prostate or the bladder, and successfully prevented physically implausible non-linear deformation of bones.

In cases with large deformations in the bladder region, the algorithm could potentially be extended with additional constraints to enforce matching the bladder contours. Additional integration of further anatomical information, such as masking of areas with non-corresponding structures in the bowel region, could potentially further increase the physical plausibility of the results.

In summary, with clinically feasible runtimes, the proposed algorithm presents a more plausible alternative to unconstrained algorithms. The obtained results show great potential for improving the physical plausibility of clinically-used registration results, which may ultimately increase clinicians' confidence in deformable registration algorithms.

8 Real-time registration for liver ultrasound tracking

Patient motion is a critical factor in many areas of medical treatment. In addition to voluntary movement of patients, which in some cases can be controlled or restricted, also involuntary motion such as heartbeat or respiration has to be taken into account. The reliable determination and tracking of these movements may allow for advanced therapy and treatment. In radiotherapy, especially respiratory motion can limit the treatment of thoracic, abdominal and pelvic tumors [KMB+06], such that real-time tracking is beneficial [SSKO07].

In these cases, ultrasound imaging can provide the necessary data to determine organ motion. Ultrasound provides real-time image acquisition at low cost and with relatively easy setup. Furthermore, ultrasound uses non-ionizing radiation, which greatly reduces imaging-induced side effects. However, ultrasound images are typically subject to high noise levels, making the determination of motion information a challenging task.

Furthermore, the visibility of tumors in the ultrasound image is limited, depending on the size and type of tumor. Therefore, easily visible structures such as liver vessels are used as tracking landmarks [MHSK13; TBSS12]. Tracking these features over time makes it possible to determine liver movement caused by respiratory motion. Especially in long time series with several minutes in length, this can be difficult due to propagated tracking errors and increased probability for tracking failures [DTST13].

Many different approaches for ultrasound tracking exist, ranging from optical flow methods, speckle tracking and block matching approaches to various image registration approaches; see [DeLuc13, §2] for an overview. Due to the typically non-linear organ motion, deformable registration is of special interest. However, its widespread use is impeded by high computational cost in connection with the high frame rates in ultrasound imaging.

In this chapter, we derive a real-time point tracking method for liver ultrasound images, based on the matrix-free deformable image algorithm presented in Chapter 3. In Section 8.1, we first present the image registration algorithm that is used as a basis for the proposed method. In Section 8.2, we integrate it into a novel tracking scheme, which allows processing of long time series. Finally, in Section 8.3, we evaluate the tracking algorithm on 28 long liver ultrasound sequences, as shown in Figure 8.1, against manual gold-standard annotations and other algorithms. The datasets originate from the MICCAI challenge on liver ultrasound tracking (CLUST14) [DBK+15*], where our approach achieved the best mean tracking error in two-dimensional tracking.

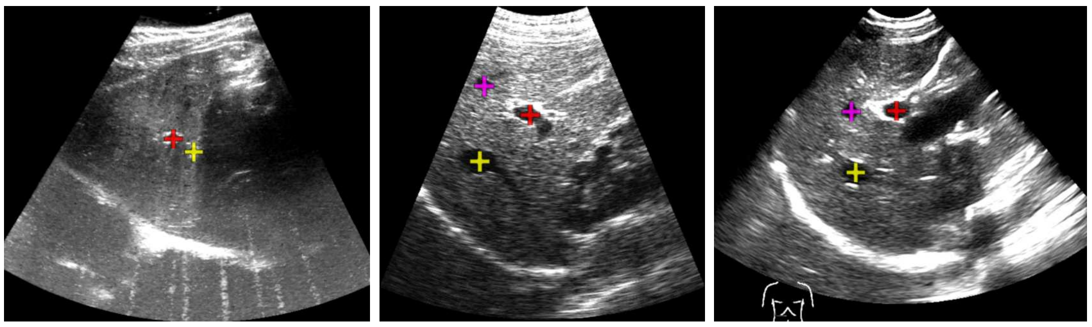

(a) ETH 05 (b) MED 04 (c) MED 13

Figure 8.1.: Initial frame of two-dimensional liver ultrasound datasets with annotations for three different scanners from the CLUST14 datasets [DBK+15*]. The images show different numbers of annotations, different noise levels and fields of view. Figure adapted from [DBK+15*].

This application uses a full deformable registration as its basis. The matrix-free approach is employed to achieve real-time performance that cannot be achieved with matrix-based approaches, allowing for a new area of application for deformable image registration.

Acknowledgments and related publications. The evaluation results and a short description of the method presented in this chapter have been previously published in [DBK+15*]. A preliminary version of the tracking algorithm was presented in [KKR14*]. This chapter contains an extended version with a detailed description of the method. We especially thank Till Kipshagen and Jan Rühaak for their joint work on the proposed tracking scheme as well as Valeria De Luca and Christine Tanner for organizing the challenge on liver ultrasound tracking at the MICCAI 2014 conference.

8.1. Image registration algorithm

As basis of the tracking algorithm, we utilize a deformable image registration framework, as discussed in Chapter 2. In contrast to the previous applications, we consider tracking of features on *two-dimensional* images, i.e., $d = 2$. The images $I_k \in \mathbb{R}^{m_x \times m_y}$ are part of a two-dimensional video sequence with T individual frames $I_1, \dots, I_T$. In the first registration step, we solve the two-frame registration problem

$$\min_{\varphi:\Omega_{\mathcal{I}}\to\mathbb{R}^2} \mathcal{J}(\varphi), \quad \mathcal{J}(\varphi) = \mathcal{D}(\mathcal{I}_1, \mathcal{I}_2(\varphi)) + \alpha\mathcal{S}(\varphi),$$

where $\mathcal{I}_1$, $\mathcal{I}_2$ are the continuous image functions of the first two frames. The obtained deformation then allows us to propagate an annotation $a_1 \in \mathbb{R}^2$ from I_1 to I_2 by evaluating $\varphi(a_k)$.

Since intensities in ultrasound images can vary throughout the sequence, we consider this a multi-modal registration problem and chose $\mathcal{D}^{\text{NGF}}$ as the distance measure. However, since we are tracking vessel structures that are generally visible as dark structures with a bright border (Figure 8.1), we additionally incorporate $\mathcal{D}^{\mathrm{SSD}}$, so that

$$\mathcal{D} := \mathcal{D}^{\mathrm{SSD}} + \beta \mathcal{D}^{\mathrm{NGF}},$$

and

$$\mathcal{J}(\varphi) = \mathcal{D}^{\mathrm{SSD}}(\mathcal{I}_1, \mathcal{I}_2(\varphi)) + \beta \mathcal{D}^{\mathrm{NGF}}(\mathcal{I}_1, \mathcal{I}_2(\varphi)) + \alpha \mathcal{S}(\varphi).$$

In addition to the regularizer weight α, this introduces another parameter β. In order to equally weight both distance measures, we automatically determine

$$\beta := \frac{\mathcal{D}^{\mathrm{SSD}}(\varphi_0)}{\mathcal{D}^{\mathrm{NGF}}(\varphi_0)}$$

at the beginning of the registration using the initial transformation φ_0, which is typically the identity transformation.

As can be seen from Section 3.3.1, specifically (3.32), computing the NGF gradient already yields all required components for the computation of the SSD gradient (3.16). Therefore, the addition of SSD comes with virtually no additional computational effort.

For minimizing the objective function, we employ L-BFGS optimization, as discussed in Section 2.5.2, embedded in a coarse-to-fine multi-level scheme, see Section 2.5.6. As stated earlier, we utilize the fast and efficient matrix-free algorithm from Chapter 3, using the CPU implementation as described in Section 3.8.1, which is fully parallelized and uses AVX instructions for additional speedup. Due to the relatively small image and deformation sizes in this application, which will be discussed later on, the GPU-based implementation achieved only incomplete utilization of the high number of parallel computational units on the GPU, resulting in a slower runtime than the CPU implementation. Therefore, we restrict ourselves to the CPU implementation in this chapter.

8.2. Tracking algorithm

In the evaluated ultrasound datasets, the movement of liver vessels is characterized by different types of motion occurring throughout the sequence, as can also be seen in Figure 8.2. Large movements of the vessel across the image, induced by the respiratory motion, are combined with deformations due to non-linear organ motion. Additionally, voluntary movement of the patient and scanner occasionally causes further displacement. Therefore, we embed the method described in Section 8.1 into a larger tracking framework.

We use a moving window approach with an additional fallback strategy to capture large, translational motion in addition to non-linear deformations, which will be described in the following.

Given a sequence of images $I_1, \ldots, I_T$ and annotation coordinates $a_1 \in \mathbb{R}^2$ on the first frame I_1, the goal is to track movement of the annotation a_1 over time for all frames in the sequence. For this, we define

$$W_n(I_k) := W(I_k, c_n) : \mathbb{R}^{m_x \times m_y} \to \mathbb{R}^{w_x \times w_y}, \quad n, k = 1, \ldots, T$$

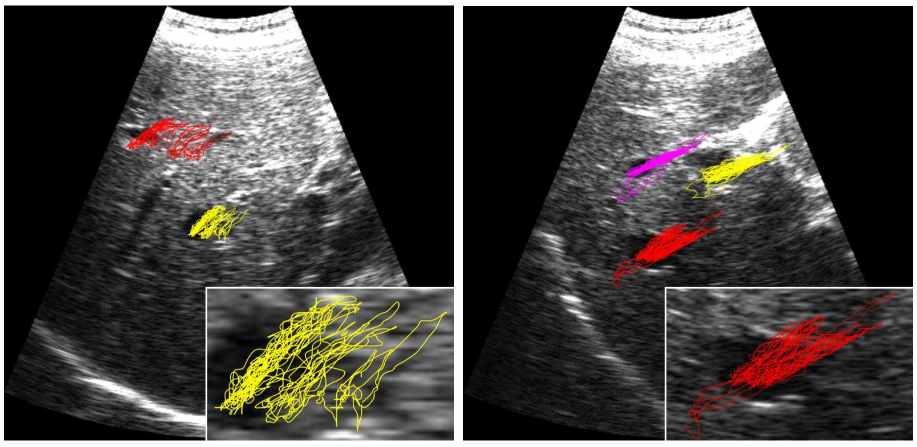


Figure 8.2.: Trajectories of liver vessel motion for manual annotations of the complete sequence, projected onto the first frame for two datasets with challenging motion patterns. An enlarged version of a single trajectory is shown in the lower right corner of each image. While the movement is largely periodic due to breathing, it also contains more erratic components, caused by additional scanner or irregular patient movement.

as a function which crops I_k to a smaller region of interest, with center point $c_n \in \mathbb{R}^2$ and extents $w_x < m_x$, $w_y < m_y$. While the center point c_n varies for each image in the sequence, the extents w_x, w_y are fixed throughout the tracking process.

Starting with $c_1 := a_1$, we then register $W_1(I_1)$ and $W_1(I_2)$, which results in the transformation φ_1, so that a_1 can be propagated from I_1 to I_2 via $a_2 := \varphi_1(a_1)$, as shown in Figure 8.3. The propagated annotation a_2 now serves as the center point $c_2 := a_2$ for a new window $W_2(I_3)$ on I_3. Now, we register $W_1(I_1)$ and $W_2(I_3)$, resulting in φ_2 and again compute the propagated annotation with $a_3 := \varphi_2(a_1)$. This process can then be repeated until the end of the sequence, see Algorithm 8.1.

As discussed, the registration of the current window is always performed relative to $W_1(I_1)$ on the first frame. This is essential for an accurate tracking of long sequences, as it prevents the accumulation of small registration errors.

The annotation can move considerably throughout the tracking process. Since we always register to the first frame, this eventually leads to large deformations. However, such large deformations are difficult to achieve with a deformable registration scheme, which could lead to failed registrations, and, due to the real-time requirements, we cannot utilize a rigid or affine pre-registration step in this application. Using the moving window strategy, large movements are compensated by the window location. While the window can move considerably throughout the tracking process, the contents of the window remain similar and only smaller movements have to be identified by the registration in each step, as shown in Figure 8.4.

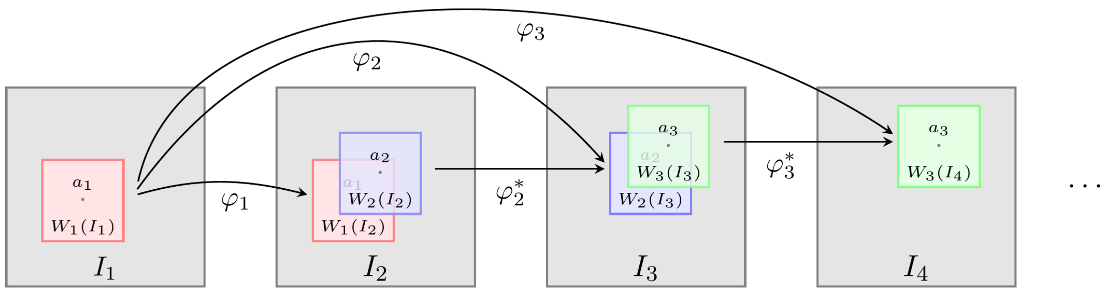


Figure 8.3.: Proposed tracking scheme. The registration between window $W_{k-1}(I_k)$ on the current frame and window $W_1(I_1)$ yields the transformation φ_k. If this registration fails, we employ a fallback strategy and compute φ_k^* by registering $W_{k-1}(I_k)$ to the window of the previous frame $W_{k-1}(I_{k-1})$. The annotation a_{k-1} is then propagated using the computed deformation. The resulting propagated annotation a_k serves as the center of a new window $W_k(I_k)$. Figure adapted from [KKR14*].

8.2.1. Fallback strategy

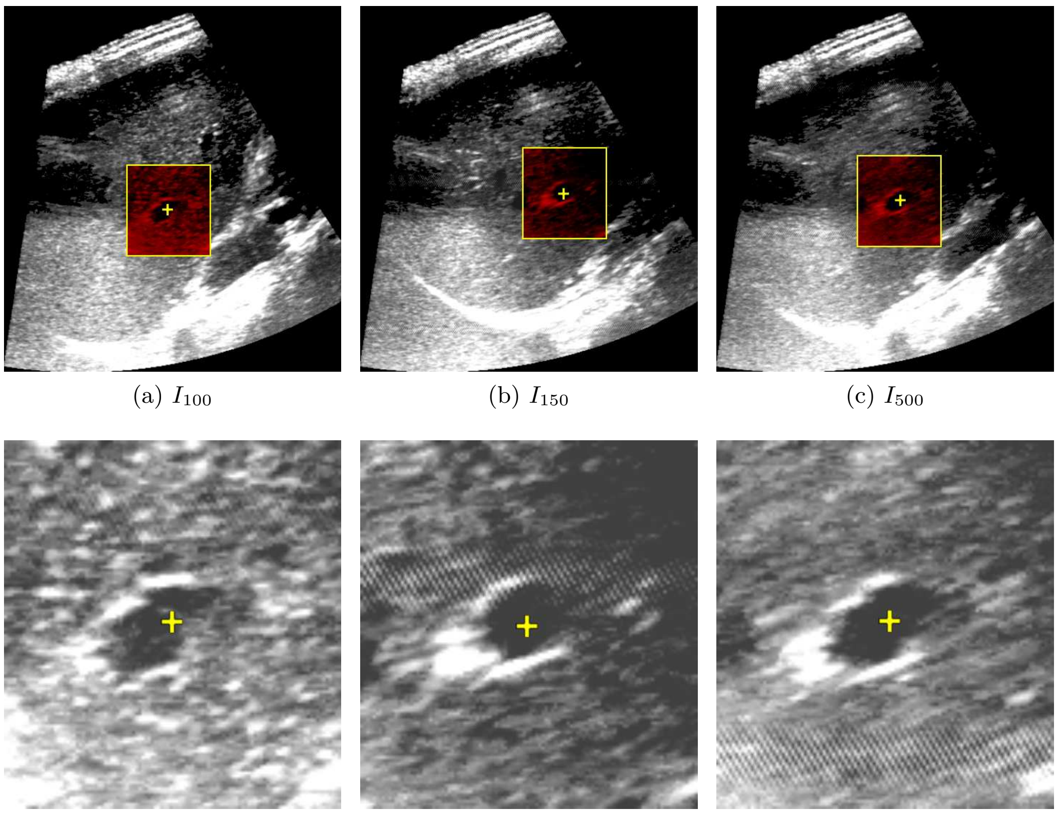
In some cases, the appearance of the vessel changes substantially compared to the first frame. Reasons for this can be out-of-plane movements or non-linear organ deformations. This can cause a mis-registration and ultimately a failure in the tracking process. For cases where the vessel only changes its appearance temporarily, we therefore added an additional fallback strategy.

For each step, we compare the image intensity at the location of annotation a_1, denoted as $I_1(a_1)$, to the image intensity $I_k(a_k)$. If the intensities differ by at least a certain threshold, specifically,

$$|I_1(a_1) - I_k(a_k)| > I_1(a_1)\theta + \epsilon, \tag{8.1}$$

we switch the registration strategy: instead of registering the current window to the window of the first frame, we register $W_{k-1}(I_k)$ to the window of the previous frame $W_{k-1}(I_{k-1})$, obtaining φ_k^*, as shown in Figure 8.3 and Algorithm 8.1. The parameters θ and ϵ need to be chosen manually, depending on the ultrasound device, see Section 8.3.2. The parameter ϵ acts as a safeguard in cases where the image intensity of $I_1(a_1)$ is very small.

This strategy is based on the assumption that if the current window largely differs from the window on the first frame, the previous frame will still be sufficiently similar to allow for a successful registration. Unfortunately, this registration scheme is susceptible to error accumulation since we compute $\varphi_k^*(a_k)$ instead of $\varphi_k(a_1)$. Therefore, this fallback strategy is only performed temporarily until (8.1) is not fulfilled anymore. Then, the original scheme takes over again, which eliminates the accumulated error by registering to $W_1(I_1)$.



(a) I_{100} (b) I_{150} (c) I_{500}

(d) $W_{99}(I_{100})$ (e) $W_{149}(I_{150})$ (f) $W_{499}(I_{500})$

Figure 8.4.: Selected frames of dataset ETH 07 with moving window (red) and tracked annotation (yellow). While the window around the annotation is subject to large global movements (top row), the contents of each window remain similar (bottom row), which serves as a good starting point for the deformable registration algorithm. Figure adapted from [KKR14*].

8.2.2. Multiple annotations and annotation coupling

In order to track more than one annotation, multiple tracking algorithms are executed simultaneously.

As each step of the tracking algorithm computes a dense deformation within the current window, further annotations in the vicinity can be propagated cheaply, “coupling” them to the main annotation (Figure 8.5). This can be used to further reduce the overall runtime of the tracking scheme.

However, the window movement is still determined by the movement of the main annotation. Thus, this strategy can only be used for close-by annotations for which a similar movement is expected. In the following, all annotations were tracked separately, as this was already sufficient to achieve real-time performance.

```
 1: load $I_1, a_1$                                          ▷ Load first frame and initial annotation
 2: for $k$ in $[2, T]$ do                                  ▷ Loop over all frames of the sequence
 3:     load $I_k$                                                              ▷ Load current frame
 4:     $\varphi_{k-1} \leftarrow \text{registration}(W_1(I_1), W_{k-1}(I_k))$          ▷ Register to first window
 5:     $a_k \leftarrow \varphi_{k-1}(a_1)$                                     ▷ Compute new annotation
 6:
 7:     if $|I_1(a_1) - I_k(a_k)| > I_1(a_1)\theta + \epsilon$ then                ▷ Check fallback criterion
 8:         $\varphi^*_{k-1} \leftarrow \text{registration}\ (W_{k-1}(I_{k-1}), W_{k-1}(I_k))$   ▷ Register to prev. window
 9:         $a_k \leftarrow \varphi^*_{k-1}(a_{k-1})$                            ▷ Replace new annotation
10:     end if
11: end for
```

Algorithm 8.1: Pseudocode of the tracking algorithm for a single annotation. The current window is registered to the window of the first frame, unless the fallback criterion becomes active. Then, the registration scheme is changed and registrations are performed to the previous frame. This prevents tracking failures in cases where the tracked feature changes its appearance from the first frame for a limited period of time.

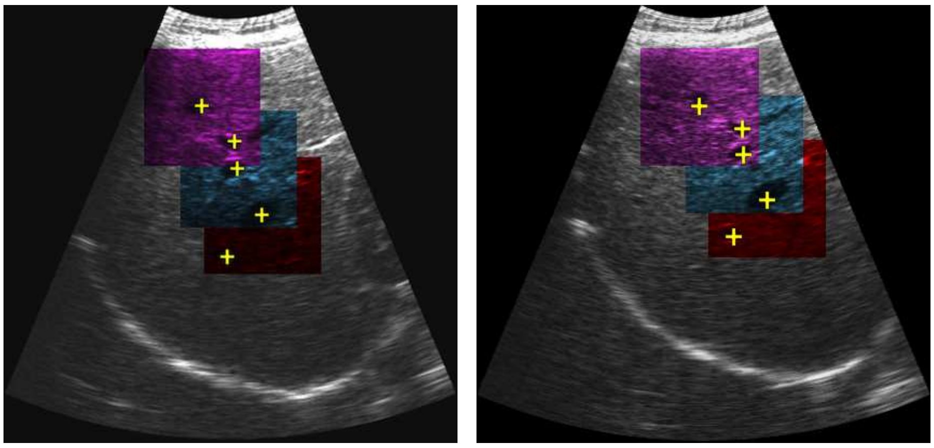
8.3. Evaluation

We submitted our tracking algorithm to the MICCAI challenge of liver ultrasound tracking (CLUST14) [DBK+15*]. The datasets, evaluation method and results are described in the following.

8.3.1. Datasets and evaluation method

For two-dimensional tracking, the CLUST14 organizers provided 30 long liver ultrasound sequences from volunteers under free breathing. The sequences were acquired by the Computer Vision Laboratory, ETH Zürich (ETH) and mediri GmbH, Heidelberg (MED) [DBK+15*] using three different scanners, with datasets ETH 01 – ETH 12, MED 01 – MED 12 (MED_1) and MED 13 – MED 16 (MED_2) being recorded on the same device. As shown in Table 8.3, the sequence length ranges from 2424 to 14516 frames, at 11 to 25 frames per second. For the ETH datasets, sizes and resolution slightly vary with an average size of $m = (453, 530)$ and average resolutions of $h = (0.41\,\text{mm}, 0.41\,\text{mm})$. For the MED_1 and MED_2 datasets, size and resolution are identical for all datasets of each device with a size of $m = (512, 512)$ and resolution of $h = (0.41\,\text{mm}, 0.41\,\text{mm})$ for MED_1 and a size of $m = (524, 591)$ and resolution of $h = (0.35\,\text{mm}, 0.35\,\text{mm})$ for MED_2.

Each sequence was provided with up to five annotations on the first frame, placed at locations of liver vessels. The first frame of an exemplary dataset for each of the three scanners is shown in Figure 8.1. As discussed in Section 8.2, motion patterns include mainly respiratory motion, but also non-linear organ motion and movement of the patient and scanner (Figure 8.2).



(a) I_1 (b) I_{1750}

Figure 8.5.: Possible coupling of five annotations in dataset MED 09, (a): first frame, (b): intermediate result. The windows are shown as colored rectangles. The second annotation is coupled with the third, and the fifth annotation is coupled with the fourth (numbering from top to bottom), so that only three windows are required for tracking five annotations. Coupling uses the deformation from tracking of the main annotation to track further annotations within the same window. This further reduces the runtime. Figure adapted from [KKR14*].

Prior to the challenge, the datasets ETH 05 and MED 04 were released as training data with annotations for all frames of the sequence, leaving 28 datasets for evaluation.

For benchmarking, the organizers of the challenge manually annotated 10% of the frames of all 28 evaluation datasets. Then, the tracking error for each frame with manual annotation was measured using the Euclidean distance between the tracked coordinates and the manual annotation.

8.3.2. Parameterization

In a real-time clinical scenario, free "tuning" parameters such as the regularizer weight α, the NGF edge filtering parameters τ, ϱ, and the fallback strategy thresholds θ, ϵ need to be fixed at the beginning of the tracking process and cannot be chosen in retrospective. We therefore used a fixed, manually chosen parameterization for each of the three different devices for all evaluation datasets. For the ETH datasets, we used $\alpha = 10$, the NGF edge filtering parameters $\tau = \varrho = 20$, and $\theta = 0.5, \epsilon = 5$ for the criterion of the fallback strategy. For MED$_1$ we used $\alpha = 5$, $\tau = \varrho = 5$, $\theta = 0.75, \epsilon = 5$, and for MED$_2$ we used $\alpha = 100$, $\tau = \varrho = 2$, $\theta = 0.5, \epsilon = 5$.

For all registrations between windows, we used a multi-level scheme with two levels, using the original image resolution as finest level with a deformation grid size of $m^{\mathrm{y}} = (17, 17)$. The window size w_x, w_y was chosen as 50 mm in each direction for all datasets. The smaller deformation size also acts as additional regularization.

Dataset	MTE_1	MTE_2	MTE_3	MTE_4	MTE_5
ETH 01	0.98 ± 1.12				
02	0.54 ± 0.26				
03	0.94 ± 0.84	0.43 ± 0.28	0.31 ± 0.19		
04	0.59 ± 0.85				
06	0.33 ± 0.24	0.55 ± 0.36			
07	0.83 ± 0.52				
08	0.63 ± 0.29	0.71 ± 0.48			
09	0.39 ± 0.23	0.79 ± 0.52			
10	0.60 ± 0.68	0.46 ± 0.44	0.71 ± 1.05	0.58 ± 0.97	
11	0.97 ± 0.75	1.50 ± 1.34			
12	2.21 ± 2.75	2.61 ± 2.26			
MED 01	1.36 ± 0.68	0.91 ± 0.36	0.90 ± 0.48		
02	1.09 ± 0.61	1.20 ± 0.71	0.95 ± 0.44		
03	1.16 ± 0.62	1.95 ± 1.12	1.05 ± 0.60	0.94 ± 0.49	
05	1.65 ± 0.78	1.91 ± 0.77	2.75 ± 1.24		
06	1.58 ± 0.85	1.17 ± 0.52	1.30 ± 0.57		
07	3.84 ± 2.47	1.48 ± 0.77	2.10 ± 0.99		
08	1.80 ± 0.95	8.75 ± 8.86			
09	1.68 ± 0.98	1.31 ± 0.68	1.14 ± 0.75	2.40 ± 0.85	1.38 ± 0.60
10	2.14 ± 1.12	1.20 ± 0.61	2.30 ± 0.93	2.67 ± 2.45	
11	1.88 ± 0.94	3.15 ± 2.19	1.20 ± 0.63		
12	3.33 ± 6.36	3.81 ± 4.80	5.17 ± 4.23		
MED 13	1.02 ± 0.74	1.48 ± 0.72	1.45 ± 0.80		
14	1.59 ± 0.78	1.72 ± 0.91	3.17 ± 1.91		
15	1.46 ± 1.53				
16	3.47 ± 1.60	2.02 ± 1.02			

Table 8.1.: Mean tracking error (MTE_n) in mm with standard deviation for annotation n on the the CLUST14 two-dimensional tracking evaluation datasets, using the proposed tracking algorithm. Datasets contain between one and five annotations.

8.3.3. Results

We tracked 66 annotations on 28 datasets in total. The results were submitted to the CLUST14 organizers and evaluated on the manually annotated frames, which were not publicly released.

Tracking error. The mean tracking error (MTE) achieved with the proposed tracking algorithm over all frames is shown in Table 8.1 for each individual annotation. Generally, we achieved a lower tracking error of 0.83 mm ± 1.15 mm for the ETH datasets, averaged over all frames, compared with the MED datasets, most likely related to different noise levels and motion patterns. For the MED_1 datasets, a MTE of 2.07 mm ± 2.75 mm was obtained, for MED_2 we achieved a MTE of 1.93 mm ± 1.42 mm. For the second annotation of dataset MED 08, an unusually high error can be observed. Here, the tracking failed around frame 1500 due to motion close to the image border, causing the tracked an-

Method	MTE	MTE (ETH)		MTE (MED_1)		MTE (MED_2)	
		min.	max.	min.	max.	min.	max.
König et al. (proposed)	**1.44** ± 2.04	**0.31**	2.61	0.90	8.75	1.02	3.47
Rothlübbers et al. [RSJG14]	1.53 ± 2.45	0.32	3.02	0.94	5.12	1.20	12.71
Benz et al. [BKN14]	1.64 ± 1.84	0.43	7.48	**0.56**	**4.24**	0.95	**2.83**
Kondo [Kon14]	1.83 ± 3.16	0.37	**1.73**	0.93	13.22	1.62	3.63
Somphone et al. [SAMD14]	2.00 ± 2.87	0.51	3.47	0.79	12.72	**0.88**	3.54
Lübke and Grozea [LG14]	2.09 ± 2.87	0.52	10.05	0.59	11.27	**0.88**	3.36
Identity transformation	6.64 ± 4.81	2.90	13.56	3.78	12.48	4.33	12.31

Table 8.2.: Mean tracking error (MTE) in mm for all datasets and minimum/maximum MTE values for individual devices for all algorithms competing in the two-dimensional tracking benchmark of the CLUST14 challenge. The proposed approach achieves the best (lowest) overall MTE. Table modified from [DBK+15*].

notation to drift away from the original structure. The algorithm then continued tracking a similar nearby structure.

Over all frames, our method achieved a mean tracking error of 1.44 mm ± 2.04 mm. As shown in Table 8.2, in comparison with the other five participants of the two-dimensional tracking in the CLUST14 challenge, we achieved the lowest mean tracking error. In comparison with the identity transformation, all algorithms achieved a significant reduction in mean tracking error.

Runtime. For the challenge submission reported in [DBK+15*], we used a preliminary version of the matrix-free registration algorithm without AVX acceleration. Additionally, the tracking algorithm from Section 8.2 was implemented in Python and not optimized for performance. Therefore, for this thesis, we re-evaluated our computations with an implementation in C++, using the matrix-free registration from Chapter 3. All evaluations were performed on a workstation with Intel Core i7-6700K CPU with four cores and 4.00 GHz.

The achieved tracking speed in frames per second (FPS) is shown in Table 8.3. For all datasets we achieved real-time performance above acquisition rate, ranging from 25.1 to 130.9 FPS. Since multiple annotations are tracked with separate tracking algorithms, more annotations lead to a slower tracking speed.

8.4. Discussion and summary

We presented an algorithm for annotation tracking in long ultrasound sequences. Using the fast and efficient matrix-free registration algorithm from Chapter 3 as a basis, the tracking method achieved the best mean tracking error for two-dimensional tracking in the MICCAI CLUST14 challenge at real-time performance.

Dataset	Frames	Acq. FPS	No. of ann.	Tracking FPS
ETH 01	14 516	25	1	130.9
02	5244	16	1	103.3
03	5578	17	3	33.5
04	2620	15	1	86.2
06	5586	17	2	50.3
07	4588	14	1	68.1
08	5574	17	2	55.6
09	5247	16	2	57.8
10	4587	15	4	31.0
11	4615	15	2	47.0
12	4284	14	2	42.6
MED 01	2470	20	3	45.1
02	2478	20	3	41.8
03	2456	20	4	30.7
05	2458	20	3	41.6
06	2443	20	3	38.7
07	2450	20	3	38.1
08	2442	20	2	54.6
09	2436	20	5	25.1
10	2427	20	4	28.9
11	2424	20	3	31.4
12	2450	20	3	36.4
MED 13	3304	11	3	31.9
14	3304	11	3	37.0
15	3304	11	1	97.0
16	3304	11	2	46.7

Table 8.3.: Image acquisition rate in frames per seconds (FPS) and achieved tracking speed of the proposed tracking algorithm. For all datasets, we achieve real-time performance. Multiple annotations are tracked independently and thus lead to lower frame rates.

Here, the matrix-free image registration algorithm is key to successfully using a full deformable image registration in an application setting with real-time constraints.

The algorithm uses a moving window approach and therefore can compensate for large translational movements in addition to non-linear motion. A fallback strategy enables continued tracking in cases where the vessel temporarily changes its appearance. However, no assumptions about the observed motion, such as periodicity, are made and no further feature recognition is required, making the proposed method a versatile tool also for other potential applications in motion tracking.

A current limitation of the algorithm is the limited – but necessary – parameter selection. While a single set of parameters for each device proved to be sufficient on the challenge dataset, parameters are currently chosen manually and also depend on observed motion artifacts, such as missing frames or motion of the scanner. Automatic parameter selection could improve the usability of the method in a clinical environment. Additionally, besides

NGF and SSD, further distances should be evaluated: based on our tracking approach, the authors of [HPC+15] proposed a demons-based registration with a scale invariant feature transform (SIFT) distance measure and achieved excellent results.

9 Conclusion and outlook

In this thesis, we presented a matrix-free algorithm for deformable image registration. Based on the variational image registration model presented in Chapter 2, we derived novel, fully matrix-free methods for the registration objective function derivative computations in Chapter 3. Initially, in Section 3.2, we considered the SSD distance measure. After analyzing the derivative structure, we derived fully matrix-free computations for the gradient as well as for the matrix-vector multiplication with the Gauss-Newton Hessian. After this, in Section 3.3 we considered the multi-modal NGF distance measure. Using the previously derived computations, we extended the matrix-free methods to the more complicated derivative structure of the NGF, again deriving matrix-free expressions for gradient computations and the Hessian-vector multiplication. Additionally, we determined a matrix-free formulation of the curvature regularizer in Section 3.4. Completing the objective function derivatives, we furthermore considered the necessary grid conversion operators in Section 3.5, mapping between image grid and deformation grid and derived efficient matrix-free computation schemes, including a red-black method for computing the transposed grid conversion operator.

With this, we have derived fully matrix-free, closed-form expressions for all important components of the objective function derivatives. These expressions have multiple benefits:

- they are fully parallelizable,
- do not require sparse matrix arithmetic, and
- do not need intermediate storage.

This reduces the memory requirements of the derivative computations to a constant amount and enables the full utilization of multi-core architectures. Instead of storing precomputed intermediate results, the matrix-free computations perform all required computations directly on the fly, based on the original image data. However, considering the number of computational operations, this is associated with a certain overhead of recalculations, when intermediate results are required that have been computed before, but were not stored. Therefore, in Section 3.7, we presented different implementation alternatives, optionally precomputing selected structures with high numbers of recalculations. We showed that these alternatives, with a trade-off of moderate memory requirements, further reduce the number of computational operations required.

The proposed matrix-free computations can also be extended to other deformation models; most notably the rigid and affine deformation model in Section 3.6, which requires a modified computational scheme for efficient parallelization. Furthermore, the matrix-free computations are not limited to a specific implementation or architecture: In Section 3.8, we presented specific implementations for multi-core CPUs, GPUs and a distributed DSP platform.

As the derivation of the matrix-free methods can require a considerable amount of effort, for comparison, we additionally implemented the registration algorithm using automatic differentiation in Chapter 4. Based on the Theano framework, this method automatically determines analytically exact derivatives. Furthermore, the computations can be optionally executed on the GPU without additional effort. However, for the optimization of the computational graph, the utilized framework requires a certain compilation time. As this time is higher on the GPU, but only required once on initialization, the benefit of the GPU computations depends on the number of expected function evaluations.

The proposed methods were evaluated comprehensively in Chapter 5, ranging from individual components to a full multi-level registration. In Section 5.1, we showed that, given a sufficient number of computational cores, all relevant matrix-free objective function components exhibit virtually linear scalability, allowing them to fully benefit from parallel execution. Furthermore, we compared the matrix-free computations and the Theano-based algorithm with matrix-based implementations for the evaluation of the objective function derivatives in Section 5.3 and for a full multi-level registration in Section 5.4:

- The matrix-free computations were several orders of magnitude faster than their matrix-based equivalents for different image and deformation sizes and were able to handle higher resolutions without exceeding the available memory.
- Our matrix-free algorithm was the only algorithm able to process registrations with image sizes of 512^3 voxels, while all other algorithms ran out of memory, making the matrix-free algorithm favorable over matrix-based computations in every important aspect.

Additionally, we evaluated different implementation alternatives for the matrix-free computations in Section 5.2 and Section 5.4, each realizing a different trade-off between recalculations and selectively precomputed values. Even the version with the highest number of precomputations achieved a lower memory consumption than the matrix-based approaches, while still being faster. At the cost of an increased runtime, the memory requirements can be further reduced to a constant amount for the objective function derivatives, potentially enabling the computation of very large image sizes. In Section 5.5, we evaluated a GPU implementation of the matrix-free registration algorithm, achieving a considerable additional speedup of up to one order of magnitude.

The automatic Theano-based algorithm did not reach the performance we achieved through manual analysis. However, it performs similar to the matrix-based approaches. Given that it does not require any manual effort for computing derivatives and can seamlessly utilize GPU acceleration, it constitutes an interesting alternative for rapid prototyping and algorithm development.

Applications. In the second part of this thesis, we presented three real-world applications of the matrix-free registration algorithm with very different requirements, ranging from processing of large datasets and high deformation resolutions to real-time performance.

In Chapter 6, we presented an **automatic registration pipeline for thorax-abdomen CT images** in follow-up diagnosis for oncology. Here, synchronized navigation between prior and follow-up scans can support the radiologist by reducing the time required for manually navigating the data. As images can easily exceed sizes of $512 \times 512 \times 1000$ voxels and need to be processed in a clinically feasible runtime, this is a challenging application scenario. We evaluated the algorithm on a large dataset of 489 patients with 986 registrations in total. Despite different scanners, image sizes and fields of view, the automatic registration pipeline successfully registered all datasets, with an average runtime of 38.1 s for the complete analysis, including an average of only 8.3 s for the deformable registration part. Therefore, the proposed method is well-suited for clinical application. It is currently being evaluated in practice as part of an oncology workstation.

Moving on from unconstrained deformable image registration, we incorporated further anatomical knowledge into the registration model for a **radiotherapy application** in Chapter 7: during the registration of male pelvic CT images with intra-treatment cone-beam CT images, very different deformation properties of adjacent organs have to be considered. Using unconstrained deformable registration can lead to physically implausible non-linear deformations of bones. Therefore, we added an **additional local rigidity constraint** to the registration model, allowing to keep bones and other structures such as the prostate rigid, while the remaining organs and tissue are allowed to deform non-linearly. This application is well suited for the matrix-free algorithm, as it requires high deformation resolutions in order to accurately preserve the desired rigidity boundaries. We evaluated the registration with local rigidity constraints on a dataset of ten male pelvis cases. In comparison with the unconstrained matrix-free algorithm and an unconstrained state-of-the-art registration method used in a commercial clinical workstation, we achieved comparable accuracy. Additionally, our method largely reduced the grid foldings and successfully preserved the rigid deformation properties of bones. Utilizing the matrix-free computations, we achieved **clinically feasible runtimes on real-world datasets**.

The third application, presented in Chapter 8, is characterized by strict real-time requirements. Here, using liver ultrasound sequences, vessel annotations are tracked with the goal of determining respiratory motion and ultimately correct the target location, e.g., in radiotherapy. For this, we integrated the matrix-free registration algorithm into a larger **tracking scheme using a pairwise moving window approach** with deformable registration. We evaluated the proposed algorithm on 28 ultrasound sequences with a total of 66 annotations as part of the MICCAI CLUST14 two-dimensional tracking challenge. Out of the six participants, our algorithm achieved the lowest mean tracking error at a real-time tracking speed of up to 130.9 frames per second. Here, the proposed matrix-free algorithm is key to successfully using a **full non-linear registration in an application with real-time constraints**.

Outlook. A currently unsolved problem is the automatic determination of a suitable parameterization of the registration algorithm. At the moment, several parameters such as the regularizer weight α, the NGF edge filtering parameters ϱ and τ, the deformation resolution and multi-level settings have to be determined manually for every application scenario, usually by performing an exhaustive search over a larger set of parameters and datasets. Here, methods which enable an automatic determination of these parameters, comparable to [HDH10], could potentially further increase the possible areas of clinical application for the matrix-free registration algorithm.

While the automatic registration pipeline in Chapter 6 successfully registered a large number of datasets, robustness in daily practice can often still be improved. Given a reasonable starting guess, the deformable registration generally achieves satisfying results. However, coarsely aligning the datasets automatically is often a challenge. Here, novel methods for image pre-alignment, potentially based on deep learning algorithms [LMT+17], can further increase the robustness of image registration solutions on the path to a truly generic registration framework.

Just as the matrix-free algorithm has enabled new use cases for registration in the three presented applications, it can potentially enable registration in further clinical areas where image registration was impractical before. Using the matrix-free computations as a basis, it would be interesting to incorporate further anatomical knowledge, which was not feasible before in a clinical setting due to runtime or memory constraints. We hope that this will further increase the use of reliable automatic image registration in clinical practice, with the ultimate goal of improving diagnosis and treatment.

Bibliography

[AAA+16] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, *et al.*, "Theano: A Python framework for fast computation of mathematical expressions", *ArXiv preprint*, no. arXiv:1605.02688v1, 2016. *Cit. on pp. 91, 94, 95.*

[ABC+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, *et al.*, "TensorFlow: A system for large-scale machine learning", in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–284. *Cit. on p. 94.*

[AF11] J. Ashburner and K. J. Friston, "Diffeomorphic registration using geodesic shooting and Gauss-Newton optimisation", *NeuroImage*, vol. 55, no. 3, pp. 954–967, 2011. *Cit. on pp. 19, 36.*

[Ami94] Y. Amit, "A nonlinear variational problem for image matching", *SIAM Journal on Scientific Computing*, vol. 15, no. 1, pp. 207–224, 1994. *Cit. on pp. 2, 17, 18.*

[ARH05] M. Auer, P. Regitnig, and G. A. Holzapfel, "An automatic nonrigid registration for stained histological sections", *IEEE Transactions on Image Processing*, vol. 14, no. 4, pp. 475–486, 2005. *Cit. on p. 34.*

[Arr18] ArrayFire, *Explaining FP64 performance on GPUs*, 2018. [Online]. Available: `https://arrayfire.com/explaining-fp64-performance-on-gpus` (visited on 05/28/2018). *Cit. on p. 128.*

[Bal06] R. Bale, "Multimodality registration in daily clinical practice", in *Mathematical Models for Registration and Applications to Medical Imaging*, O. Scherzer, Ed., Springer Berlin Heidelberg, 2006, pp. 165–183. *Cit. on pp. 3, 133.*

[BB09] P. Bui and J. Brockman, "Performance analysis of accelerated image registration using GPGPU", in *2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-2)*, 2009, pp. 38–45. *Cit. on pp. 20, 21.*

[BBB+10] J. Bergstra, O. Breuleux, F. F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, *et al.*, "Theano: A CPU and GPU math compiler in Python", in *Python for Scientific Computing Conference (SciPy)*, 2010, pp. 1–7. *Cit. on p. 95.*

[Ber12] R. Berg, "Leistungssteigerung in der medizinischen Bildregistrierung durch Mehrkern-Signalprozessoren", Bachelor's thesis, Wilhelm Büchner Hochschule Darmstadt, 2012. *Cit. on pp. 21, 44, 85.*

[BG96] M. Bro-Nielsen and C. Gramkow, "Fast fluid registration of medical images", in *4th International Conference on Visualization in Biomedical Computing*, 1996, pp. 265–276. *Cit. on p. 19.*

[BHE+08] K. K. Brock, M. Hawkins, C. Eccles, J. L. Moseley, D. J. Moseley, D. A. Jaffray, and L. A. Dawson, "Improving image-guided target localization through deformable registration", *Acta Oncologica*, vol. 47, no. 7, pp. 1279–1285, 2008. *Cit. on p. 141.*

[BHG04] J. Bernier, E. J. Hall, and A. Giaccia, "Radiation oncology: A century of achievements", *Nature Reviews. Cancer*, vol. 4, no. 9, pp. 737–747, 2004. *Cit. on p. 4.*

[BHWB18] M. Bücker, P. Hovland, C. Wente, and H. Bach, *www.Autodiff.org*, 2018. [Online]. Available: `http://www.autodiff.org` (visited on 05/18/2018). *Cit. on p. 94.*

[BK89] R. Bajcsy and S. Kovačič, "Multiresolution elastic matching", *Computer Vision, Graphics, and Image Processing*, vol. 46, no. 1, pp. 1–21, 1989. *Cit. on p. 19.*

[BKN14] T. Benz, M. Kowarschik, and N. Navab, "Kernel-based tracking in ultrasound sequences of liver", in *MICCAI Challenge on Liver Ultrasound Tracking (CLUST14)*, 2014, pp. 21–28. *Cit. on p. 164.*

[BKR+14*] R. Berg, L. König, J. Rühaak, R. Lausen, and B. Fischer, "Highly efficient image registration for embedded systems using a distributed multicore DSP architecture", *Journal of Real-Time Image Processing*, vol. 14, no. 2, pp. 341–361, 2014. *Cit. on pp. 8, 20, 21, 26, 44, 85, 86.*

[BLP+12] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, *et al.*, "Theano: New features and speed improvements", in *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*, 2012, pp. 1–10. *Cit. on p. 94.*

[BLYY12] R. Baskar, K. A. Lee, R. Yeo, and K. W. Yeoh, "Cancer and radiation therapy: Current advances and future directions", *International Journal of Medical Sciences*, vol. 9, no. 3, pp. 193–199, 2012. *Cit. on p. 4.*

[BMR13] M. Burger, J. Modersitzki, and L. Ruthotto, "A hyperelastic regularization energy for image registration", *SIAM Journal on Scientific Computing*, vol. 35, no. 1, pp. B132–B148, 2013. *Cit. on pp. 2, 19, 25.*

[BMTY05] M. F. Beg, M. I. Miller, A. Trouvé, and L. Younes, "Computing large deformation metric mappings via geodesic flows of diffeomorphisms", *International Journal of Computer Vision*, vol. 61, no. 2, pp. 139–157, 2005. *Cit. on p. 19.*

[BPRS18] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey", *Journal of Machine Learning Research*, vol. 18, no. 153, pp. 1–43, 2018. *Cit. on pp. 91, 93–95.*

[Bro81] C. Broit, "Optimal registration of deformed images", PhD thesis, University of Pennsylvania, 1981. *Cit. on p. 19.*

[Bro92] L. G. Brown, “A survey of image registration techniques”, *ACM Computing Surveys*, vol. 24, no. 4, pp. 325–376, 1992. *Cit. on p. 15.*

[BT01] T. Butz and J.-P. Thiran, “Affine registration with feature space mutual information”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2001, pp. 549–556. *Cit. on pp. 15, 20, 21.*

[CBMK04] M. M. Coselmon, J. M. Balter, D. L. McShan, and M. L. Kessler, “Mutual information based CT registration of the lung at exhale and inhale breathing states using thin-plate splines”, *Medical Physics*, vol. 31, no. 11, pp. 2942–2948, 2004. *Cit. on p. 34.*

[CCBF09] M. D. Craene, O. Camara, B. H. Bijnens, and A. F. Frangi, “Large diffeomorphic FFD registration for motion and strain quantification from 3D-US sequences”, in *International Conference on Functional Imaging and Modeling of the Heart (FIMH)*, 2009, pp. 437–446. *Cit. on p. 35.*

[CCH07] W. R. Crum, O. Camara, and D. J. Hawkes, “Methods for inverting dense displacement fields: Evaluation in brain image registration”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2007, pp. 900–907. *Cit. on p. 136.*

[CCM+10] E. Castillo, R. Castillo, J. Martinez, M. Shenoy, and T. Guerrero, “Four-dimensional deformable image registration using trajectory modeling”, *Physics in Medicine and Biology*, vol. 55, no. 1, pp. 305–327, 2010. *Cit. on p. 14.*

[CDH+06] U. Clarenz, M. Droske, S. Henn, M. Rumpf, and K. Witsch, “Computational methods for nonlinear image registration”, in *Mathematical Models for Registration and Applications to Medical Imaging*, vol. 10, Springer Berlin Heidelberg, 2006, pp. 81–101. *Cit. on pp. 2, 18.*

[CH08] N. Courty and P. Hellier, “Accelerating 3D non-rigid registration using graphics hardware”, *International Journal of Image and Graphics*, vol. 1, no. 8, pp. 1–18, 2008. *Cit. on pp. 21, 22.*

[CHH04] W. R. Crum, T. Hartkens, and D. L. G. Hill, “Non-rigid image registration: Theory and practice”, *The British Journal of Radiology*, vol. 77, pp. S140–S153, 2004. *Cit. on pp. 15, 20.*

[Chr94] G. E. Christensen, “Deformable shape models for anatomy”, PhD thesis, Washington University, 1994. *Cit. on p. 19.*

[Chr98] ——, “MIMD vs. SIMD parallel processing: A case study in 3D medical image registration”, *Parallel Computing*, vol. 24, pp. 1369–1383, 1998. *Cit. on pp. 21–23.*

[CHZ11] J. Cong, M. Huang, and Y. Zou, “Accelerating fluid registration algorithm on multi-FPGA platforms”, *21st International Conference on Field Programmable Logic and Applications (FPL)*, pp. 50–57, 2011. *Cit. on pp. 21, 23.*

[CJ01] G. E. Christensen and H. J. Johnson, “Consistent image registration”, *IEEE Transactions on Medical Imaging*, vol. 20, no. 7, pp. 568–582, 2001. *Cit. on p. 18.*

[CJM97] G. E. Christensen, S. C. Joshi, and M. I. Miller, “Volumetric transformation of brain anatomy”, *IEEE Transactions on Medical Imaging*, vol. 16, no. 6, pp. 864–877, 1997. *Cit. on p. 19.*

[CJS03] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, “FAIR: A hardware architecture for real-time 3-D image registration”, *IEEE Transactions on Information Technology in Biomedicine*, vol. 7, no. 4, pp. 426–434, 2003. *Cit. on p. 21.*

[CLC+08] M. Chen, W. Lu, Q. Chen, K. J. Ruchala, and G. H. Olivera, “A simple fixed-point approach to invert a deformation field”, *Medical Physics*, vol. 35, no. 1, pp. 81–88, 2008. *Cit. on p. 136.*

[CLQS12] W. M. Chiew, F. Lin, K. Qian, and H. S. Seah, “Demons kernel computation with single-pass stream processing on FPGA”, in *IEEE 14th International Conference on High Performance Computing and Communications (HPCC)*, 2012, pp. 1321–1328. *Cit. on pp. 20, 21, 23.*

[CMD+95] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal, “Automated multi-modality image registration based on information theory”, in *14th International Conference on Information Processing in Medical Imaging*, 1995, pp. 263–274. *Cit. on p. 24.*

[CPA99] P. Cachier, X. Pennec, and N. Ayache, “Fast non rigid matching by gradient descent: Study and improvements of the ’demons’ algorithm”, *Technical Report*, no. RR-3706, INRIA, pp. 1–25, 1999. *Cit. on p. 17.*

[CRM96] G. E. Christensen, R. D. Rabbitt, and M. I. Miller, “Deformable templates using large deformation kinematics”, *IEEE Transactions on Image Processing*, vol. 5, no. 10, pp. 1435–1447, 1996. *Cit. on p. 19.*

[CS05] C. R. Castro-Pareja and R. Shekhar, “Hardware acceleration of mutual information based 3D image registration”, *Journal of Imaging Science and Technology*, vol. 49, no. 2, pp. 105–113, 2005. *Cit. on pp. 1, 20, 21, 23.*

[CWN02] A. Chung, W. Wells, and A. Norbash, “Multi-modal image registration by minimising kullback-leibler distance”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2002, pp. 525–532. *Cit. on p. 34.*

[DBK+15*] V. De Luca, T. Benz, S. Kondo, L. König, D. Lübke, S. Rothlübbers, O. Somphone, *et al.*, “The 2014 liver ultrasound tracking benchmark”, *Physics in Medicine and Biology*, vol. 60, no. 14, pp. 5571–5599, 2015. *Cit. on pp. 6, 8, 16, 25, 155, 156, 161, 164.*

[DDW+16] X. Du, J. Dang, Y. Wang, S. Wang, and T. Lei, “A parallel nonrigid registration algorithm based on B-Spline for medical images”, *Computational and Mathematical Methods in Medicine*, vol. 2016, pp. 1–14, 2016. *Cit. on pp. 22, 24.*

[DeLuc13] V. De Luca, “Liver motion tracking in ultrasound sequences for tumor therapy”, PhD thesis, ETH Zürich, 2013. *Cit. on p. 155.*

[Dic45] L. R. Dice, “Measures of the amount of ecologic association between species”, *Ecology*, vol. 26, no. 3, pp. 297–302, 1945. *Cit. on p. 147.*

[DKM16*] A. Derksen, L. König, and H. Meine, "An alternating image registration approach for large scale bladder deformations in radiation therapy", in *18th International Conference on the use of Computers in Radiation Therapy (ICCR)*, 2016, pp. 1–2. *Cit. on p. 153.*

[DKMH16*] A. Derksen, L. König, H. Meine, and S. Heldmann, "A joint registration and segmentation approach for large bladder deformations in adaptive radiotherapy", in *Medical Physics: Proceedings of the AAPM 58th annual meeting*, vol. 43, 2016, p. 3429. *Cit. on p. 153.*

[DM98] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming", *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998. *Cit. on p. 82.*

[DMM+03] E. D'Agostino, J. Modersitzki, F. Maes, D. Vandermeulen, B. Fischer, and P. Suetens, "Free-form registration using mutual information and curvature regularization", in *International Workshop on Biomedical Image Registration (WBIR)*, 2003, pp. 11–20. *Cit. on p. 19.*

[DR04] M. Droske and M. Rumpf, "A variational approach to non-rigid morphological registration", *SIAM Journal on Applied Mathematics*, vol. 64, no. 2, pp. 668–687, 2004. *Cit. on pp. 2, 18, 24.*

[DS07] O. Dandekar and R. Shekhar, "FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions", *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 2, pp. 116–127, 2007. *Cit. on pp. 21, 23.*

[DS96] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Society for Industrial and Applied Mathematics (SIAM), 1996. *Cit. on pp. 34–36, 38.*

[DSHK99] D. Dey, P. J. Slomka, L. J. Hahn, and R. Kloiber, "Automatic three-dimensional multimodality registration using radionuclide transmission CT attenuation maps: A phantom study", *Journal of Nuclear Medicine*, vol. 40, no. 3, pp. 448–455, 1999. *Cit. on p. 34.*

[DTST13] V. De Luca, M. Tschannen, G. Székely, and C. Tanner, "A learning-based approach for fast and robust vessel tracking in long ultrasound sequences", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2013, pp. 518–525. *Cit. on p. 155.*

[EPW+07] B. J. Erickson, J. Patriarche, C. Wood, N. Campeau, E. Paul, V. Savcenko, N. Arslanlar, *et al.*, "Image registration improves confidence and accuracy of image interpretation", *Cancer Informatics*, vol. 4, pp. 19–24, 2007. *Cit. on pp. 16, 133.*

[EWSH11] J. Ehrhardt, R. Werner, A. Schmidt-Richberg, and H. Handels, "Statistical modeling of 4D respiratory lung motion using diffeomorphic image registration", *IEEE Transactions on Medical Imaging*, vol. 30, no. 2, pp. 251–265, 2011. *Cit. on pp. 16, 19.*

[EYSL15] N. D. Ellingwood, Y. Yin, M. Smith, and C. L. Lin, “Efficient methods for implementation of multi-level nonrigid mass-preserving image registration on GPUs and multi-threaded CPUs”, *Computer Methods and Programs in Biomedicine*, vol. 127, pp. 290–300, 2015. *Cit. on pp. 21, 22.*

[FAB+17] C. Fitzmaurice, C. Allen, R. M. Barber, L. Barregard, Z. A. Bhutta, H. Brenner, D. J. Dicker, *et al.*, “Global, regional, and national cancer incidence, mortality, years of life lost, years lived with disability, and disability-adjusted life-years for 32 cancer groups, 1990 to 2015”, *JAMA Oncology*, vol. 3, no. 4, pp. 524–548, 2017. *Cit. on p. 2.*

[FDW+15] M. Fatyga, N. Dogan, E. Weiss, W. C. Sleeman, B. Zhang, W. J. Lehman, J. F. Williamson, *et al.*, “A voxel-by-voxel comparison of deformable vector fields obtained by three deformable image registration algorithms applied to 4DCT lung studies”, *Frontiers in Oncology*, vol. 5, no. 17, pp. 1–9, 2015. *Cit. on p. 1.*

[FGM+17] D. Forsberg, A. Gupta, C. Mills, B. MacAdam, B. Rosipko, B. A. Bangert, M. D. Coffey, *et al.*, “Synchronized navigation of current and prior studies using image registration improves radiologist’s efficiency”, *International Journal of Computer Assisted Radiology and Surgery*, vol. 12, no. 3, pp. 431–438, 2017. *Cit. on pp. 3, 15, 16, 133.*

[FHW08] C. Frohn-Schauf, S. Henn, and K. Witsch, “Multigrid based total variation image registration”, *Computing and Visualization in Science*, vol. 11, no. 2, pp. 101–113, 2008. *Cit. on p. 19.*

[Fle87] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. John Wiley & Sons, 1987. *Cit. on pp. 34, 37.*

[FM01] B. Fischer and J. Modersitzki, “Fast diffusion registration”, in *Contemporary Mathematics*, vol. 313, 2001, pp. 117–127. *Cit. on pp. 17, 19, 25.*

[FM03a] ——, “Combination of automatic non-rigid and landmark based registration: The best of both worlds”, in *SPIE Medical Imaging 2003: Image Processing*, vol. 5032, 2003, pp. 1037–1048. *Cit. on p. 19.*

[FM03b] ——, “Curvature based image registration”, *Journal Mathematical Imaging and Vision*, vol. 18, no. 1, pp. 81–85, 2003. *Cit. on pp. 19, 26, 33.*

[FM03c] ——, “FLIRT: A flexible image registration toolbox”, in *International Workshop on Biomedical Image Registration (WBIR)*, 2003, pp. 261–270. *Cit. on p. 19.*

[FM04a] ——, “A unified approach to fast image registration and a new curvature based registration technique”, *Linear Algebra and Its Applications*, vol. 380, no. 1-3, pp. 107–124, 2004. *Cit. on pp. 19, 25, 26.*

[FM04b] ——, “Intensity-based image registration with a guaranteed one-to-one point match”, *Methods of Information in Medicine*, vol. 43, no. 4, pp. 327–330, 2004. *Cit. on p. 19.*

[FM08] ——, “Ill-posed medicine—an introduction to image registration”, *Inverse Problems*, vol. 24, no. 034008, pp. 1–16, 2008. *Cit. on pp. 1, 19, 23.*

[FMB+15] D. Fontanarosa, S. van der Meer, J. Bamber, E. Harris, T. O'Shea, and F. Verhaegen, "Review of ultrasound image guidance in external beam radiotherapy: I. Treatment planning and inter-fraction motion management", *Physics in Medicine and Biology*, vol. 60, no. 3, pp. R77–R114, 2015. *Cit. on p. 6.*

[FRD+09] W. Feng, S. J. Reeves, T. S. Denney, S. Lloyd, L. Dell'Italia, and H. Gupta, "A new consistent image registration formulation with a B-Spline deformation model", *IEEE 6th International Symposium on Biomedical Imaging (ISBI)*, pp. 979–982, 2009. *Cit. on p. 18.*

[FVW+11] O. Fluck, C. Vetter, W. Wein, A. Kamen, B. Preim, R. Westermann, and D.-. Magdeburg, "A survey of medical image registration on graphics hardware", *Computer Methods and Programs in Biomedicine*, vol. 104, no. 3, pp. e45–e57, 2011. *Cit. on p. 20.*

[GB98] J. C. Gee and R. K. Bajcsy, "Elastic matching: Continuum mechanical and probabilistic analysis", in *Brain Warping*, 1998, pp. 183–197. *Cit. on p. 19.*

[GCP+09] W. H. Greene, S. Chelikani, K. Purushothaman, J. P. Knisely, Z. Chen, X. Papademetris, L. H. Staib, *et al.*, "Constrained non-rigid registration for use in image-guided adaptive radiotherapy", *Medical Image Analysis*, vol. 13, no. 5, pp. 809–817, 2009. *Cit. on p. 142.*

[GDW+13] X. Gu, B. Dong, J. Wang, J. Yordy, L. Mell, X. Jia, and S. B. Jiang, "A contour-guided deformable image registration algorithm for adaptive radiotherapy", *Physics in Medicine and Biology*, vol. 58, no. 6, pp. 1889–1901, 2013. *Cit. on p. 141.*

[GMTT13] B. A. Gutman, S. K. Madsen, A. W. Toga, and P. M. Thompson, "A family of fast spherical registration algorithms for cortical shapes", in *International Workshop on Multimodal Brain Image Analysis*, 2013, pp. 246–257. *Cit. on p. 17.*

[GMW81] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization.* Academic Press, 1981. *Cit. on pp. 34, 39.*

[Gos05] A. Goshtasby, *2-D and 3-D Image Registration.* John Wiley & Sons, 2005. *Cit. on p. 14.*

[GPL+10] X. Gu, H. Pan, Y. Liang, R. Castillo, D. Yang, D. Choi, E. Castillo, *et al.*, "Implementation and evaluation of various demons deformable image registration algorithms on a GPU", *Physics in Medicine and Biology*, vol. 55, pp. 207–219, 2010. *Cit. on pp. 17, 21, 22.*

[GRB+12] F. Gigengack, L. Ruthotto, M. Burger, C. H. Wolters, and X. Jiang, "Motion correction in dual gated cardiac PET using mass-preserving image registration", *IEEE Transactions on Medical Imaging*, vol. 31, no. 3, pp. 698–712, 2012. *Cit. on pp. 2, 19.*

[GSTA10] G. Goeckenjan, H. Sitter, M. Thomas, and E. Al., "Prevention, diagnosis, therapy, and follow-up of lung cancer: Interdisciplinary guideline of the German Respiratory Society and the German Cancer Society", *Pneumologie*, vol. 65, no. 1, pp. 39–59, 2010. *Cit. on p. 3.*

[GV13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Johns Hopkins University Press, 2013. *Cit. on pp. 37, 38, 47.*

[GW08] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed. Society for Industrial and Applied Mathematics (SIAM), 2008. *Cit. on pp. 91, 92, 94.*

[Har07] M. Harris, "Optimizing parallel reduction in CUDA", *NVIDIA Developer Technology*, 2007. *Cit. on p. 84.*

[HBHH01] D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration", *Physics in Medicine and Biology*, vol. 46, no. 3, pp. 1–45, 2001. *Cit. on p. 24.*

[HBO09] M. Hernandez, M. N. Bossa, and S. Olmos, "Registration of anatomical images using paths of diffeomorphisms parameterized with stationary vector field flows", *International Journal of Computer Vision*, vol. 85, no. 3, pp. 291–306, 2009. *Cit. on p. 19.*

[HBS06] B. F. Hutton, M. Braun, and P. Slomka, "Image registration techniques in nuclear medicine imaging", in *Quantitative Analysis in Nuclear Medicine Imaging*, H. Zaidi, Ed. Springer US, 2006, pp. 272–307. *Cit. on pp. 3, 133.*

[HCF04] G. Hermosillo, C. Chefd'hotel, and O. Faugeras, "Variational methods for multimodal image matching", *International Journal of Computer Vision*, vol. 50, no. 3, pp. 329–343, 2004. *Cit. on pp. 2, 18.*

[HD11] D. Holland and A. M. Dale, "Nonlinear registration of longitudinal images and measurement of change in regions of interest", *Medical Image Analysis*, vol. 15, no. 4, pp. 489–497, 2011. *Cit. on p. 1.*

[HDH10] D. A. Hahn, V. Daum, and J. Hornegger, "Automatic parameter selection for multimodal image registration", *IEEE Transactions on Medical Imaging*, vol. 29, no. 5, pp. 1140–1155, 2010. *Cit. on p. 170.*

[Hel06] S. Heldmann, *Non-Linear Registration Based on Mutual Information: Theory, Numerics, and Application*. Logos-Verlag, 2006. *Cit. on pp. 19, 26, 32, 33, 37, 38, 59.*

[Hen03] S. Henn, "A levenberg–marquardt scheme for nonlinear image registration", *BIT Numerical Mathematics*, vol. 43, pp. 743–759, 2003. *Cit. on p. 36.*

[Hen06] ——, "A full curvature based algorithm for image registration", *Journal of Mathematical Imaging and Vision*, vol. 24, no. 2, pp. 195–208, 2006. *Cit. on pp. 19, 26, 33.*

[HHH01] J. V. Hajnal, D. L. G. Hill, and D. J. Hawkes, *Medical Image Registration*. CRC Press, 2001. *Cit. on p. 14.*

[HHM09] E. Haber, S. Heldmann, and J. Modersitzki, "A computational framework for image-based constrained registration", *Linear Algebra and Its Applications*, vol. 431, no. 3–4, pp. 459–470, 2009. *Cit. on pp. 19, 142.*

[HJB+12] M. P. Heinrich, M. Jenkinson, M. Bhushan, T. Matin, F. V. Gleeson, J. M. Brady, and J. A. Schnabel, "MIND: Modality independent neighbourhood descriptor for multi-modal deformable registration", *Medical Image Analysis*, vol. 16, no. 7, pp. 1423–1435, 2012. *Cit. on pp. 19, 36.*

[HJBS11] M. P. Heinrich, M. Jenkinson, J. M. Brady, and J. A. Schnabel, "Non-rigid image registration through efficient discrete optimization", in *Medical Image Analysis and Understanding*, 2011, pp. 187–192. *Cit. on pp. 20, 35.*

[HM04] E. Haber and J. Modersitzki, "Numerical methods for volume preserving image registration", *Inverse Problems*, vol. 20, no. 5, pp. 1621–1638, 2004. *Cit. on pp. 18, 19, 147.*

[HM05] ——, "Beyond mutual information: A simple and robust alternative", in *Bildverarbeitung für die Medizin*, 2005, pp. 350–354. *Cit. on p. 24.*

[HM06a] ——, "A multilevel method for image registration", *SIAM Journal on Scientific Computing*, vol. 27, no. 5, pp. 1–17, 2006. *Cit. on pp. 13, 19, 27, 36.*

[HM06b] ——, "Intensity gradient based registration and fusion of multi-modal images", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2006, pp. 726–733. *Cit. on p. 24.*

[HM07a] ——, "Image registration with guaranteed displacement regularity", *International Journal of Computer Vision*, vol. 71, no. 3, pp. 361–372, 2007. *Cit. on p. 19.*

[HM07b] ——, "Intensity gradient based registration and fusion of multi-modal images", *Methods of Information in Medicine*, vol. 46, no. 3, pp. 292–299, 2007. *Cit. on pp. 19, 24.*

[HMC+07] J. M. Hensel, C. Ménard, P. W. M. Chung, M. F. Milosevic, A. Kirilova, J. L. Moseley, M. A. Haider, *et al.*, "Development of multiorgan finite element-based prostate deformation model enabling registration of endorectal coil magnetic resonance imaging for radiotherapy planning", *International Journal of Radiation Oncology Biology Physics*, vol. 68, no. 5, pp. 1522–1528, 2007. *Cit. on p. 140.*

[HNCB10] M. Hossny, S. Nahavandi, D. Creighton, and A. Bhatti, "Towards autonomous image fusion", in *IEEE 11th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2010, pp. 1748–1754. *Cit. on p. 15.*

[HPC+15] A. Hallack, B. W. Papiez, A. Cifor, M. J. Gooding, and J. A. Schnabel, "Robust liver ultrasound tracking using dense distinctive image features", in *MICCAI Challenge on Liver Ultrasound Tracking (CLUST15)*, 2015, pp. 28–35. *Cit. on p. 166.*

[HRK93] D. Huttenlocher, W. Rucklidge, and G. Klanderman, "Comparing images using the Hausdorff distance under translation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 850–863, 1993. *Cit. on p. 147.*

[HS81] B. K. B. Horn and B. G. Schunck, "Determining optical flow", *Artificial Intelligence*, vol. 17, no. 1–3, pp. 185–203, 1981. *Cit. on pp. 17, 19.*

[HSG10] M. P. Heinrich, J. A. Schnabel, and F. Gleeson, "Non-rigid multimodal medical image registration using optical flow and gradient orientation", in *Medical Image Analysis and Understanding*, 2010, pp. 141–145. *Cit. on p. 19.*

[HSP+16] M. P. Heinrich, I. J. A. Simpson, B. W. Papiez, J. M. Brady, and J. A. Schnabel, “Deformable image registration by combining uncertainty estimates from supervoxel belief propagation”, *Medical Image Analysis*, vol. 27, pp. 57–71, 2016. *Cit. on pp. 20, 35.*

[HZ03] R. Hartley and A. Zisserman, *Multiple View Geometry.* Cambridge University Press, 2003. *Cit. on p. 15.*

[IIH14] K. Ikeda, F. Ino, and K. Hagihara, “Efficient acceleration of mutual information computation for nonrigid registration using CUDA”, *IEEE Journal of Biomedical and Health Informatics*, vol. 18, no. 3, pp. 956–968, 2014. *Cit. on pp. 21, 22.*

[IOH05] F. Ino, K. Ooyama, and K. Hagihara, “A data distributed parallel algorithm for nonrigid image registration”, *Parallel Computing*, vol. 31, no. 1, pp. 19–43, 2005. *Cit. on pp. 21, 22.*

[Jäh05] B. Jähne, *Digital Image Processing*, 6th ed. Springer, 2005. *Cit. on p. 39.*

[JBBS02] M. Jenkinson, P. Bannister, J. M. Brady, and S. Smith, “Improved optimization for the robust and accurate linear registration and motion correction of brain images”, *NeuroImage*, vol. 17, no. 2, pp. 825–841, 2002. *Cit. on p. 15.*

[JKDM07] D. Jaffray, P. Kupelian, T. Djemil, and R. M. Macklis, “Review of image-guided radiation therapy”, *Xpert Review of Anticancer Therapy*, vol. 7, no. 1, pp. 89–103, 2007. *Cit. on pp. 4, 5, 139.*

[JLR02] J. Jiang, W. Luk, and D. Rueckert, “FPGA-based computation of free-form deformations”, in *IEEE International Conference on FieId-Programmable Technology (FPT)*, 2002, pp. 407–410. *Cit. on pp. 20, 21, 23.*

[JSD+14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, *et al.*, “Caffe: Convolutional architecture for fast feature embedding”, in *22nd ACM International Conference on Multimedia (MM)*, 2014, pp. 675–678. *Cit. on p. 94.*

[KBD17] A. P. Keszei, B. Berkels, and T. M. Deserno, “Survey of non-rigid registration tools in medicine”, *Journal of Digital Imaging*, vol. 30, no. 1, pp. 102–116, 2017. *Cit. on pp. 14, 24.*

[KCC+14] K.-W. Kwok, G. C. Chow, T. C. Chau, Y. Chen, S. H. Zhang, W. Luk, E. J. Schmidt, *et al.*, “FPGA-based acceleration of MRI registration: an enabling technique for improving MRI-guided cardiac therapy”, *Journal of Cardiovascular Magnetic Resonance*, vol. 16, no. 1, pp. W11–W13, 2014. *Cit. on pp. 21, 23.*

[KCW06] A. Khamene, R. Chisu, and W. Wein, “A novel projection based approach for medical image registration”, in *International Workshop on Biomedical Image Registration (WBIR)*, 2006, pp. 247–256. *Cit. on pp. 20, 21.*

[KD04] B. Karaçali and C. Davatzikos, “Estimating topology preserving and smooth displacement fields”, *IEEE Transactions on Medical Imaging*, vol. 23, no. 7, pp. 868–880, 2004. *Cit. on pp. 145, 147.*

[KDH+15*] L. König, A. Derksen, S. Heldmann, N. Papenberg, J. Modersitzki, and B. Haas, "Deformable image registration with guaranteed local rigidity", in *Radiotherapy and Oncology: Proceedings of the 3rd ESTRO Forum*, vol. 115, 2015, pp. S197–S198. *Cit. on pp. 8, 141.*

[KDHP15*] L. König, A. Derksen, M. Hallmann, and N. Papenberg, "Parallel and memory efficient multimodal image registration for radiotherapy using normalized gradient fields", in *IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, 2015, pp. 734–738. *Cit. on pp. 8, 21, 23, 44, 106.*

[KDPH16*] L. König, A. Derksen, N. Papenberg, and B. Haas, "Deformable image registration for adaptive radiotherapy with guaranteed local rigidity constraints", *Radiation Oncology*, vol. 11, no. 1, pp. 122–130, 2016. *Cit. on pp. 8, 25, 26, 141.*

[KDR+06] A. Köhn, J. Drexl, F. Ritter, M. König, and H. O. Peitgen, "GPU accelerated image registration in two and three dimensions", in *Bildverarbeitung für die Medizin*, 2006, pp. 261–265. *Cit. on pp. 21, 22.*

[Kel99] C. T. Kelley, *Iterative Methods for Optimization.* Society for Industrial and Applied Mathematics (SIAM), 1999. *Cit. on p. 34.*

[Kes06] M. L. Kessler, "Image registration and data fusion in radiation therapy", *The British Journal of Radiology*, vol. 79, pp. S99–S108, 2006. *Cit. on p. 139.*

[KKL+13] J. Kim, S. Kumar, C. Liu, H. Zhong, D. Pradhan, M. Shah, R. Cattaneo, *et al.*, "A novel approach for establishing benchmark CBCT/CT deformable image registrations in prostate cancer radiotherapy", *Physics in Medicine and Biology*, vol. 58, no. 22, pp. 8077–8097, 2013. *Cit. on pp. 142, 153, 154.*

[KKR14*] L. König, T. Kipshagen, and J. Rühaak, "A non-linear image registration scheme for real-time liver ultrasound tracking using normalized gradient fields", in *MICCAI Challenge on Liver Ultrasound Tracking (CLUST14)*, 2014, pp. 29–36. *Cit. on pp. 7, 8, 156, 159, 160, 162.*

[KLT03] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by direct search: New perspectives on some classical and modern methods", *SIAM Review*, vol. 45, no. 3, pp. 385–482, 2003. *Cit. on p. 34.*

[KLW+08] P. A. Kupelian, K. M. Langen, T. R. Willoughby, O. A. Zeidan, and S. L. Meeks, "Image-guided radiotherapy for localized prostate cancer: Treating a moving target", *Seminars in Radiation Oncology*, vol. 18, no. 1, pp. 58–66, 2008. *Cit. on p. 139.*

[KMB+06] P. J. Keall, G. S. Mageras, J. M. Balter, R. S. Emery, K. M. Forster, S. B. Jiang, J. M. Kapatoes, *et al.*, "The management of respiratory motion in radiation oncology report of AAPM Task Group 76", *Medical Physics*, vol. 33, no. 10, pp. 3874–3900, 2006. *Cit. on pp. 6, 155.*

[KNFM04] S. Kabus, T. Netsch, B. Fischer, and J. Modersitzki, "B-Spline registration of 3D images with Levenberg-Marquardt optimization", in *SPIE Medical Imaging 2004: Image Processing*, vol. 5370, 2004, pp. 304–313. *Cit. on p. 36.*

[Kon14] S. Kondo, "Liver ultrasound tracking using long-term and short-term template matching", in *MICCAI Challenge on Liver Ultrasound Tracking (CLUST14)*, 2014, pp. 13–20. *Cit. on p. 164.*

[KR14*] L. König and J. Rühaak, "A fast and accurate parallel algorithm for non-linear image registration using normalized gradient fields", in *IEEE 11th International Symposium on Biomedical Imaging (ISBI)*, 2014, pp. 580–583. *Cit. on pp. 8, 21, 23, 44, 106.*

[KRDL18*] L. König, J. Rühaak, A. Derksen, and J. Lellmann, "A matrix-free approach to parallel and memory-efficient deformable image registration", *SIAM Journal on Scientific Computing*, vol. 40, no. 3, pp. B858–B888, 2018. *Cit. on pp. 3, 8, 21, 23, 44, 80, 81, 106.*

[KSP07] S. Klein, M. Staring, and J. P. W. Pluim, "Evaluation of optimization methods for nonrigid medical image registration using mutual information and B-Splines", *IEEE Transactions on Image Processing*, vol. 16, no. 12, pp. 2879–2890, 2007. *Cit. on pp. 18, 34, 35.*

[KTC03] J. E. Kennedy, G. R. Ter Haar, and D. Cranston, "High intensity focused ultrasound: Surgery of the future?", *The British Journal of Radiology*, vol. 76, no. 909, pp. 590–599, 2003. *Cit. on p. 6.*

[LAFP13] M. Lorenzi, N. Ayache, G. B. Frisoni, and X. Pennec, "LCC-Demons: A robust and accurate symmetric diffeomorphic registration algorithm", *NeuroImage*, vol. 81, pp. 470–483, 2013. *Cit. on p. 17.*

[LC01] M. Lefebure and L. D. Cohen, "Image registration, optical flow and local rigidity", *Journal of Mathematical Imaging and Vision*, vol. 14, pp. 131–147, 2001. *Cit. on p. 19.*

[LCSC15] J. Lee, X. Cai, C.-B. Schönlieb, and D. A. Coomes, "Nonparametric image registration of airborne LiDAR, hyperspectral and photographic imagery of wooded landscapes", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 11, pp. 6073–6084, 2015. *Cit. on pp. 15, 25.*

[LFKC10] Y. Liu, A. Fedorov, R. Kikinis, and N. Chrisochoides, "Non-rigid registration for brain MRI: Faster and cheaper", *International Journal of Functional Informatics and Personalised Medicine*, vol. 3, no. 1, pp. 48–57, 2010. *Cit. on pp. 21, 22.*

[LG14] D. Lübke and C. Grozea, "High performance online motion tracking in abdominal ultrasound imaging", in *MICCAI Challenge on Liver Ultrasound Tracking (CLUST14)*, 2014, pp. 37–44. *Cit. on p. 164.*

[LGB00] T. M. Lehmann, H. G. Gröndahl, and D. K. Benn, "Computer-based registration for digital subtraction in dental radiology", *Dentomaxillofacial Radiology*, vol. 29, no. 6, pp. 323–346, 2000. *Cit. on p. 16.*

[LGS99] T. M. Lehmann, C. Gönner, and K. Spitzer, "Survey: Interpolation methods in medical image processing", *IEEE Transactions on Medical Imaging*, vol. 18, no. 11, pp. 1049–1075, 1999. *Cit. on p. 30.*

[LHL+16] J. M. Lotz, F. Hoffmann, J. Lotz, S. Heldmann, D. Trede, J. Oetjen, M. Becker, *et al.*, “Integration of 3D multimodal imaging data of a head and neck cancer and advanced feature recognition”, *Biochimica et Biophysica Acta (BBA) – Proteins and Proteomics*, vol. 1865, no. 7, pp. 946–956, 2016. *Cit. on p. 26.*

[LK03] F. P. Leon and S. Kammel, “Image fusion techniques for robust inspection of specular surfaces”, in *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2003*, vol. 5099, 2003, pp. 77–86. *Cit. on p. 15.*

[LKHC15] A. Li, A. Kumar, Y. Ha, and H. Corporaal, “Correlation ratio based volume image registration on GPUs”, *Microprocessors and Microsystems*, vol. 39, no. 8, pp. 998–1011, 2015. *Cit. on p. 21.*

[LM08] Y. Lin and G. Medioni, “Mutual information computation and maximization using GPU”, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2008, pp. 1–6. *Cit. on pp. 20, 21.*

[LMT+17] R. Liao, S. Miao, P. de Tournemire, S. Grbic, A. Kamen, T. Mansi, and D. Comaniciu, “An artificial agent for robust image registration”, in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017, pp. 4168–4175. *Cit. on p. 170.*

[LMVS04] D. Loeckx, F. Maes, D. Vandermeulen, and P. Suetens, “Nonrigid image registration using free-form deformations with a local rigidity constraint”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2004, pp. 639–646. *Cit. on p. 18.*

[LNE11] J. Le Moigne, N. S. Netanyahu, and R. D. Eastman, *Image Registration for Remote Sensing*, J. Le Moigne, N. S. Netanyahu, and R. D. Eastman, Eds. Cambridge University Press, 2011. *Cit. on pp. 1, 14.*

[LOM+16] J. Lotz, J. Olesch, B. Muller, T. Polzin, P. Galuschka, J. M. Lotz, S. Heldmann, *et al.*, “Patch-based nonlinear image registration for gigapixel whole slide images”, *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 9, pp. 1812–1819, 2016. *Cit. on pp. 25, 26.*

[Lom11] C. Lomont, “Introduction to Intel advanced vector extensions”, *Intel Whitepaper*, 2011. *Cit. on p. 82.*

[LPG+05] C. Lau, S. D. Pathak, L. Gong, P. Kinahan, P. Cheng, and L. Ng, “Advanced PET/CT fusion workstation for oncology imaging”, in *SPIE Medical Imaging 2005: Visualization, Image-Guided Procedures, and Display*, vol. 5744, 2005, pp. 670–676. *Cit. on pp. 3, 133.*

[LPH+09] T. Lange, N. Papenberg, S. Heldmann, J. Modersitzki, B. Fischer, H. Lamecker, and P. M. Schlag, “3D ultrasound-CT registration of the liver using combined landmark-intensity information”, *International Journal of Computer Assisted Radiology and Surgery*, vol. 4, no. 1, pp. 79–88, 2009. *Cit. on p. 1.*

[LRS+10] H. Lu, M. Reyes, A. Serifovi, Y. Sakurai, S. Weber, H. Yamagata, and P. C. Cattin, "Multi-modal diffeomorphic demons registration based on point-wise mutual information", in *IEEE 7th International Symposium on Biomedical Imaging (ISBI)*, 2010, pp. 372–375. *Cit. on p. 17.*

[LSM+10] D. Loeckx, P. Slagmolen, F. Maes, D. Vandermeulen, and P. Suetens, "Nonrigid image registration using conditional mutual information", *IEEE Transactions on Medical Imaging*, vol. 29, no. 1, pp. 19–29, 2010. *Cit. on p. 35.*

[LYC08] B. Li, A. A. Young, and B. R. Cowan, "GPU accelerated non-rigid registration for the evaluation of cardiac function", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2008, pp. 880–887. *Cit. on pp. 21, 22, 36.*

[MCV+97] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens, "Multimodality image registration by maximization of mutual information", *IEEE Transactions on Medical Imaging*, vol. 16, no. 2, pp. 187–198, 1997. *Cit. on pp. 24, 34.*

[Mei16] M. Meike, "GPU-basierte nichtlineare Bildregistrierung", Master's thesis, University of Lübeck, 2016. *Cit. on pp. 23, 44, 83, 84, 106.*

[MF93] C. R. Maurer and J. M. Fitzpatrick, "A review of medical image registration", in *Interactive Image-Guided Neurosurgery*, American Association of Neurological Surgeons, 1993, pp. 17–44. *Cit. on p. 14.*

[MGB16] A. Mang, A. Gholami, and G. Biros, "Distributed-memory large deformation diffeomorphic 3D image registration", in *ACM/IEEE Conference on Supercomputing*, 2016, pp. 1–12. *Cit. on pp. 21, 22.*

[MHSK13] J. R. McClelland, D. J. Hawkes, T. Schaeffter, and A. P. King, "Respiratory motion models: A review", *Medical Image Analysis*, vol. 17, no. 1, pp. 19–42, 2013. *Cit. on p. 155.*

[MHV+03] D. Mattes, D. R. Haynor, H. Vesselle, T. K. Lewellen, and W. Eubank, "PET-CT image registration in the chest using free-form deformations", *IEEE Transactions on Medical Imaging*, vol. 22, no. 1, pp. 120–128, 2003. *Cit. on pp. 16, 35.*

[MLSO01] J. Modersitzki, G. Lustig, O. Schmitt, and W. Obelöer, "Elastic registration of brain images on large PC-Clusters", *Future Generation Computer Systems*, vol. 18, no. 1, pp. 115–125, 2001. *Cit. on pp. 21–23.*

[Mod04] J. Modersitzki, *Numerical Methods for Image Registration.* Oxford University Press, 2004. *Cit. on pp. 2, 13, 14, 17–19, 24, 25.*

[Mod08] ——, "FLIRT with rigidity – image registration with a local non-rigidity penalty", *International Journal of Computer Vision*, vol. 76, no. 2, pp. 153–163, 2008. *Cit. on p. 142.*

[Mod09] ——, *FAIR: Flexible Algorithms for Image Registration.* Society for Industrial and Applied Mathematics (SIAM), 2009. *Cit. on pp. 2, 13, 14, 19, 26, 27, 30, 33, 34, 38, 40, 43, 55, 66, 105, 112.*

[MOXS08] P. Muyan-Özcelik, J. D. Owens, J. Xia, and S. S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of demons", in *IEEE International Conference on Computational Sciences and Its Applications (ICCSA)*, 2008, pp. 223–233. *Cit. on pp. 21, 22.*

[MPR+12] K. Murphy, J. P. W. Pluim, E. M. van Rikxoort, P. A. de Jong, B. de Hoop, H. A. Gietema, O. Mets, *et al.*, "Toward automatic regional analysis of pulmonary function using inspiration and expiration thoracic CT", *Medical Physics*, vol. 39, no. 3, pp. 1650–1662, 2012. *Cit. on p. 16.*

[MRD+11] M. Modat, G. R. Ridgway, P. Daga, M. J. Cardoso, D. J. Hawkes, J. Ashburner, and S. Ourselin, "Log-Euclidean free-form deformation", in *SPIE Medical Imaging 2011: Image Processing*, vol. 7962, 2011, pp. 79621Q1–79621Q6. *Cit. on p. 18.*

[MRT+10] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, *et al.*, "Fast free-form deformation using graphics processing units", *Computer Methods and Programs in Biomedicine*, vol. 98, no. 3, pp. 278–284, 2010. *Cit. on pp. 21, 22.*

[MSD03] L. P. Muren, R. Smaaland, and O. Dahl, "Organ motion, set-up variation and treatment margins in radical radiotherapy of urinary bladder cancer", *Radiotherapy and Oncology*, vol. 69, no. 3, pp. 291–304, 2003. *Cit. on p. 139.*

[MV98] J. B. Maintz and M. A. Viergever, "A survey of medical image registration", *Medical Image Analysis*, vol. 2, no. 1, pp. 1–36, 1998. *Cit. on p. 14.*

[MVS99] F. Maes, D. Vandermeulen, and P. Suetens, "Comparative evaluation of multiresolution optimization strategies for multimodality image registration by maximization of mutual information", *Medical Image Analysis*, vol. 3, no. 4, pp. 373–386, 1999. *Cit. on p. 34.*

[NBD+09] S. Nithiananthan, K. K. Brock, M. J. Daly, H. Chan, J. C. Irish, and J. H. Siewerdsen, "Demons deformable registration for CBCT-guided procedures in the head and neck: Convergence and accuracy", *Medical Physics*, vol. 36, no. 10, pp. 4755–4764, 2009. *Cit. on p. 17.*

[Noc80] J. Nocedal, "Updating quasi-Newton matrices with limited storage", *Mathematics of Computation*, vol. 35, no. 151, pp. 773–782, 1980. *Cit. on p. 37.*

[NVI09] NVIDIA Corporation, *NVIDIA CUDA Programming Guide*, Version 2.3.1. 2009. *Cit. on p. 85.*

[NVI17] ——, *CUDA C Programming Guide*, PG-02829-001_v8.0. 2017. *Cit. on pp. 20, 83, 84.*

[NVI18] ——, *GeForce GTX980 Whitepaper*, 2018. [Online]. Available: `https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF` (visited on 05/28/2018). *Cit. on p. 128.*

[NW06] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006. *Cit. on pp. 34–38, 91, 92, 94, 136.*

[OSP02] S. Ourselin, R. Stefanescu, and X. Pennec, “Robust registration of multi-modal images: Towards real-time clinical applications”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2002, pp. 140–147. *Cit. on pp. 20, 21.*

[OT14] F. P. Oliveira and J. M. R. Tavares, “Medical image registration: A review”, *Computer Methods in Biomechanics and Biomedical Engineering*, vol. 17, no. 2, pp. 73–93, 2014. *Cit. on pp. 14, 34.*

[Pap08] N. Papenberg, “Ein genereller Registrierungsansatz mit Anwendung in der navigierten Leberchirurgie”, PhD thesis, University of Lübeck, 2008. *Cit. on pp. 25–27.*

[PCA99] X. Pennec, P. Cachier, and N. Ayache, “Understanding the ”Demon’s Algorithm”: 3D non-rigid registration by gradient descent”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 1999, pp. 597–605. *Cit. on p. 17.*

[PCL+17] A. Paszke, G. Chanan, Z. Lin, S. Gross, E. Yang, L. Antiga, and Z. Devito, “Automatic differentiation in PyTorch”, in *31st Conference on Neural Information Processing System (NIPS)*, 2017, pp. 1–4. *Cit. on p. 94.*

[PDBS08] W. Plishker, O. Dandekar, S. S. Bhattacharyya, and R. Shekhar, “Towards systematic exploration of tradeoffs for medical image registration on heterogeneous platforms”, in *IEEE Biomedical Circuits and Systems Conference (BIOCAS)*, 2008, pp. 53–56. *Cit. on pp. 20–22, 24.*

[PDBS10] ——, “Utilizing hierarchical multiprocessing for medical image registration”, *IEEE Signal Processing Magazine*, vol. 27, no. 2, pp. 61–68, 2010. *Cit. on pp. 21, 22.*

[PGSB16] M. Peroni, P. Golland, G. C. Sharp, and G. Baroni, “Stopping criteria for log-domain diffeomorphic demons registration: An experimental survey for radiotherapy application”, *Technology in cancer research & treatment*, vol. 15, no. 1, pp. 77–90, 2016. *Cit. on p. 17.*

[PMR05] D. Perperidis, R. H. Mohiaddin, and D. Rueckert, “Spatio-temporal free-form registration of cardiac MR image sequences”, *Medical Image Analysis*, vol. 9, no. 5, pp. 441–456, 2005. *Cit. on p. 18.*

[PMV00] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever, “Image registration by maximization of combined mutual information and gradiant information”, *IEEE Transactions on Medical Imaging*, vol. 19, no. 8, pp. 809–814, 2000. *Cit. on p. 34.*

[PNH+16] T. Polzin, M. Niethammer, M. P. Heinrich, H. Handels, and J. Modersitzki, “Memory efficient LDDMM for lung CT”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2016, pp. 28–36. *Cit. on pp. 19, 20, 25.*

[POL+09] N. Papenberg, J. Olesch, T. Lange, P. M. Schlag, and B. Fischer, “Landmark constrained non-parametric image registration with isotropic tolerances”, in *Bildverarbeitung für die Medizin*, 2009, pp. 122–126. *Cit. on p. 19.*

[PRW+14] T. Polzin, J. Rühaak, R. Werner, H. Handels, and J. Modersitzki, "Lung registration using automatically detected landmarks", *Methods of Information in Medicine*, vol. 53, no. 4, pp. 250–256, 2014. *Cit. on p. 19.*

[PV93] E. J. D. Pol and M. H. Viergever, "Medical image matching – A review with classification", *IEEE Engineering in Medicine and Biology Magazine*, vol. 12, no. 1, pp. 26–39, 1993. *Cit. on p. 14.*

[RAH+06] D. Rueckert, P. Aljabar, R. A. Heckemann, J. V. Hajnal, and A. Hammers, "Diffeomorphic registration using B-Splines", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2006, pp. 702–709. *Cit. on p. 18.*

[RBMM04] T. Rohlfing, R. Brandt, R. Menzel, and C. R. Maurer, "Evaluation of atlas selection strategies for atlas-based image segmentation with application to confocal microscopy images of bee brains", *NeuroImage*, vol. 21, no. 4, pp. 1428–1442, 2004. *Cit. on p. 16.*

[RDH+15] J. Ruehaak, A. Derksen, S. Heldmann, M. Hallmann, and H. Meine, "Accurate CT-MR image registration for Deep Brain Stimulation: A multi-observer evaluation study", in *SPIE Medical Imaging 2015: Image Processing*, 2015, pp. 941337–941337–7. *Cit. on p. 15.*

[RHKF13] J. Rühaak, S. Heldmann, T. Kipshagen, and B. Fischer, "Highly accurate fast lung CT registration", in *SPIE Medical Imaging 2013: Image Processing*, vol. 8669, 2013, pp. 86690Y1–86690Y–9. *Cit. on pp. 2, 19, 25, 35, 138.*

[RJ02] P. Remagnino and G. A. Jones, "Automated registration of surveillance data for multi-camera fusion", in *5th International Conference on Information Fusion (FUSION)*, vol. 2, 2002, pp. 1190–1197. *Cit. on p. 15.*

[RJR00] J.-M. Rouet, J.-J. Jacq, and C. Roux, "Genetic algorithms for a robust 3-D MR-CT registration", *IEEE Transactions on Information Technology in Biomedicine*, vol. 4, no. 2, pp. 126–136, 2000. *Cit. on pp. 20, 35.*

[RKH+13*] J. Rühaak, L. König, M. Hallmann, N. Papenberg, S. Heldmann, H. Schumacher, and B. Fischer, "A fully parallel algorithm for multimodal image registration using normalized gradient fields", in *IEEE 10th International Symposium on Biomedical Imaging (ISBI)*, 2013, pp. 572–575. *Cit. on pp. 8, 21, 25, 44.*

[RKT+17*] J. Rühaak, L. König, F. Tramnitzke, H. Köstler, and J. Modersitzki, "A matrix-free approach to efficient affine-linear image registration on CPU and GPU", *Journal of Real-Time Image Processing*, vol. 13, no. 1, pp. 205–225, 2017. *Cit. on pp. 8, 15, 21, 44.*

[RM01] T. Rohlfing and C. R. Maurer, "Intensity-based non-rigid registration using adaptive multilevel free-form deformation with an incompressibility constraint", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2001, pp. 111–119. *Cit. on p. 18.*

[RM03] T. Rohlfing and C. R. Maurer, "Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees", *IEEE Transactions on Information Technology in Biomedicine*, vol. 7, no. 1, pp. 16–25, 2003. *Cit. on pp. 1, 20–22, 43.*

[RMAP98] A. Roche, G. Malandain, N. Ayache, and X. Pennec, “Multimodal image registration by maximization of the correlation ratio”, *Technical Report*, no. RR-3378, INRIA, pp. 1–42, 1998. *Cit. on p. 24.*

[RMBJ03] T. Rohlfing, C. R. Maurer, D. A. Bluemke, and M. A. Jacobs, “Volume-preserving nonrigid registration of MR breast images using free-form deformation with an incompressibility constraint”, *IEEE Transactions on Medical Imaging*, vol. 22, no. 6, pp. 730–741, 2003. *Cit. on pp. 18, 146.*

[Roh12] T. Rohlfing, “Image similarity and tissue overlaps as surrogates for image registration accuracy: Widely used but unreliable”, *IEEE Transactions on Medical Imaging*, vol. 31, no. 2, pp. 153–163, 2012. *Cit. on p. 147.*

[RPH+15] I. S. Ramadaan, K. Peick, D. A. Hamilton, J. Evans, D. Iupati, A. Nicholson, L. Greig, *et al.*, “Validation of Varian’s SmartAdapt® deformable image registration algorithm for clinical application”, *Radiation Oncology*, vol. 10, no. 1, pp. 73–82, 2015. *Cit. on pp. 140, 146.*

[RPH+17] J. Rühaak, T. Polzin, S. Heldmann, I. J. A. Simpson, H. Handels, J. Modersitzki, and M. P. Heinrich, “Estimation of large motion in lung CT by integrating regularized keypoint correspondences into dense deformable registration”, *IEEE Transactions on Medical Imaging*, vol. 36, no. 8, pp. 1746–1757, 2017. *Cit. on pp. 19, 25, 26, 152.*

[RSH+99] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes, “Nonrigid registration using free-form deformations: Application to breast MR images”, *IEEE Transactions on Medical Imaging*, vol. 18, no. 8, pp. 712–721, 1999. *Cit. on pp. 17, 18.*

[RSJG14] S. Rothlübbers, J. Schwaab, J. Jenne, and M. Günther, “Bayesian real-time liver feature ultrasound tracking”, in *MICCAI Challenge on Liver Ultrasound Tracking (CLUST14)*, 2014, pp. 45–52. *Cit. on p. 164.*

[RUCH09] A. Ruiz, M. Ujaldon, L. Cooper, and K. Huang, “Non-rigid registration for large sets of microscopic images on graphics processors”, *Journal of Signal Processing Systems*, vol. 55, no. 1–3, pp. 229–250, 2009. *Cit. on pp. 21, 22.*

[Rüh17] J. Rühaak, “Matrix-free techniques for efficient image registration and their application to pulmonary image analysis”, PhD thesis, Jacobs University Bremen, 2017. *Cit. on p. 44.*

[RWU+14] S. Reaungamornrat, A. S. Wang, A. Uneri, Y. Otake, A. J. Khanna, and J. H. Siewerdsen, “Deformable image registration with local rigidity constraints for cone-beam CT-guided spine surgery”, *Physics in Medicine and Biology*, vol. 59, no. 14, pp. 3761–3787, 2014. *Cit. on p. 142.*

[SA16] F. Seide and A. Agarwal, “CNTK: Microsoft’s open-source deep-learning toolkit”, in *22nd ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016, p. 2135. *Cit. on p. 94.*

[SAMD14] O. Somphone, S. Allaire, B. Mory, and C. Dufour, “Live feature tracking in ultrasound liver sequences with sparse demons”, in *MICCAI Challenge on Liver Ultrasound Tracking (CLUST14)*, 2014, pp. 53–60. *Cit. on p. 164.*

[SBL+13] D. P. Shamonin, E. E. Bron, B. P. F. Lelieveldt, M. Smits, S. Klein, and M. Staring, “Fast parallel image registration on CPU and GPU for diagnostic classification of Alzheimer’s disease”, *Frontiers in Neuroinformatics*, vol. 7, no. 50, pp. 1–15, 2013. *Cit. on pp. 21, 22.*

[SC97] R. Szeliski and J. Coughlan, “Spline-based image registration”, *International Journal of Computer Vision*, vol. 22, no. 3, pp. 199–218, 1997. *Cit. on p. 17.*

[SCD11] J. Santamaría, O. Cordón, and S. Damas, “A comparative study of state-of-the-art evolutionary image registration methods for 3D modeling”, *Computer Vision and Image Understanding*, vol. 115, no. 9, pp. 1340–1354, 2011. *Cit. on pp. 20, 35.*

[Sdi08] M. Sdika, “A fast non rigid image registration with constraints on the jacobian using large scale constrained optimization”, *IEEE Transactions on Medical Imaging*, vol. 27, no. 2, pp. 271–281, 2008. *Cit. on pp. 18, 35.*

[SDP13] A. Sotiras, C. Davatzikos, and N. Paragios, “Deformable medical image registration: A survey”, *IEEE Transactions on Medical Imaging*, vol. 32, no. 7, pp. 1153–1190, 2013. *Cit. on pp. 1, 14–19, 23, 24, 34.*

[SGR+08] D. Salas-Gonzalez, J. M. Gorriz, J. Ramirez, A. Lassl, C. Puntonet, and In, “Improved Gauss-Newton optimisation methods in affine registration of SPECT brain images”, *Electronics Letters*, vol. 44, no. 22, pp. 1291–1292, 2008. *Cit. on pp. 15, 36.*

[SHP+08] M. Sen, Y. Hemaraj, W. Plishker, R. Shekhar, and S. S. Bhattacharyya, “Model-based mapping of reconfigurable image registration on FPGA platforms”, *Journal of Real-Time Image Processing*, vol. 3, no. 3, pp. 149–162, 2008. *Cit. on p. 21.*

[SKP07] M. Staring, S. Klein, and J. P. Pluim, “A rigidity penalty term for nonrigid registration”, *Medical Physics*, vol. 34, no. 11, pp. 4098–4108, 2007. *Cit. on p. 142.*

[SKS13] J. Shackleford, N. Kandasamy, and G. Sharp, *High Performance Deformable Image Registration Algorithms For Manycore Processors*. Newnes, 2013. *Cit. on pp. 21, 22.*

[SKSF07] G. C. Sharp, N. Kandasamy, H. Singh, and M. Folkert, “GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration”, *Physics in Medicine and Biology*, vol. 52, no. 19, pp. 5771–5783, 2007. *Cit. on pp. 21, 22.*

[SOPD15] A. Sotiras, Y. Ou, N. Paragios, and C. Davatzikos, “Graph-based deformable image registration”, in *Handbook of Biomedical Imaging*, Springer, 2015, pp. 331–359. *Cit. on pp. 20, 35.*

[SPA03] R. Stefanescu, X. Pennec, and N. Ayache, “Parallel non-rigid registration on a cluster of workstations”, in *HealthGrid’03*, 2003, pp. 1–8. *Cit. on pp. 21, 22.*

[SRG10] V. Saxena, J. Rohrer, and L. Gong, “A parallel GPU algorithm for mutual information based 3D nonrigid image registration”, in *European Conference on Parallel Processing*, Springer, 2010, pp. 223–234. *Cit. on pp. 20–22.*

[SRQ+01] J. A. Schnabel, D. Rueckert, M. Quist, J. M. Blackall, A. D. Castellano-Smith, T. Hartkens, G. P. Penney, *et al.*, “A generic framework for nonrigid registration based on non uniform multi-level free-form deformations”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2001, pp. 573–581. *Cit. on pp. 17, 18.*

[SSKH10a] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley, “A survey of medical image registration on multicore and the GPU”, *IEEE Signal Processing Magazine*, vol. 27, no. 2, pp. 50–60, 2010. *Cit. on pp. 2, 20, 24, 30, 34.*

[SSKH10b] ——, “Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images”, *Computer Methods and Programs in Biomedicine*, vol. 99, no. 2, pp. 133–146, 2010. *Cit. on pp. 15, 20, 21.*

[SSKO07] H. Shirato, S. Shimizu, K. Kitamura, and R. Onimaru, “Organ motion in image-guided radiotherapy: Lessons from real-time tumor-tracking radiotherapy”, *International Journal of Clinical Oncology*, vol. 12, no. 1, pp. 8–16, 2007. *Cit. on p. 155.*

[STU05] C. Ó. S. Sorzano, P. Thévenaz, and M. Unser, “Elastic registration of biological images using vector-spline regularization”, *IEEE Transactions on Biomedical Engineering*, vol. 52, no. 4, pp. 652–663, 2005. *Cit. on p. 18.*

[Sut05] H. Sutter, “The free lunch is over: A fundamental turn toward concurrency in software”, *Dr. Dobb’s Journal*, vol. 30, no. 3, pp. 202–210, 2005. *Cit. on p. 1.*

[SW14] B. W. Stewart and C. P. Wild, Eds., *World Cancer Report 2014*. International Agency for Research on Cancer, 2014. *Cit. on p. 2.*

[SWHE12] A. Schmidt-Richberg, R. Werner, H. Handels, and J. Ehrhardt, “Estimation of slipping organ motion by registration with direction-dependent regularization”, *Medical Image Analysis*, vol. 16, no. 1, pp. 150–159, 2012. *Cit. on p. 19.*

[SXMO08] S. S. Samant, J. Xia, P. Muyan-Özçelik, and J. D. Owens, “High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy”, *Medical Physics*, vol. 35, no. 8, pp. 3546–3553, 2008. *Cit. on pp. 21, 22.*

[SZ02] R. Shekhar and V. Zagrodsky, “Mutual information-based rigid and non-rigid registration of ultrasound volumes”, *IEEE Transactions on Medical Imaging*, vol. 21, no. 1, pp. 9–22, 2002. *Cit. on p. 34.*

[SZP+12] W. Shi, X. Zhuang, L. Pizarro, W. Bai, H. Wang, K.-P. Tung, P. Edwards, *et al.*, “Registration using sparse free-form deformations”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2012, pp. 659–666. *Cit. on p. 18.*

[TAP+14] M. Thor, E. S. Andersen, J. B. Petersen, T. S. Sorensen, K. O. Noe, K. Tanderup, L. Bentzen, *et al.*, “Evaluation of an application for intensity-based deformable image registration and dose accumulation in radiotherapy”, *Acta Oncologica*, vol. 53, no. 10, pp. 1329–1336, 2014. *Cit. on p. 139.*

[TAS+06] N. J. Tustison, B. B. Avants, T. A. Sundaram, J. T. Duda, and J. C. Gee, “A generalization of free-form deformation image registration within the ITK finite element framework”, in *International Workshop on Biomedical Image Registration*, 2006, pp. 238–246. *Cit. on p. 18.*

[TBSS12] C. Tanner, D. Boye, G. Samei, and G. Szekely, “Review on 4D models for organ motion compensation”, *Critical Reviews™ in Biomedical Engineering*, vol. 40, no. 2, pp. 135–154, 2012. *Cit. on p. 155.*

[TC07] T. W. H. Tang and A. C. S. Chung, “Non-rigid image registration using graph-cuts.”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2007, pp. 916–924. *Cit. on pp. 20, 35.*

[The18] Theano Development Team, *Theano Documentation*, 2018. [Online]. Available: `http://deeplearning.net/software/theano/` (visited on 05/18/2018). *Cit. on p. 96.*

[Thi95] J.-P. Thirion, “Fast non-rigid matching of 3D medical images”, *Technical Report*, no. RR-2547, INRIA, pp. 1–37, 1995. *Cit. on p. 17.*

[Thi96] ——, “Non-rigid matching using demons”, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1996, pp. 245–251. *Cit. on p. 17.*

[Thi98] ——, “Image matching as a diffusion process: An analogy with Maxwell’s demons”, *Medical Image Analysis*, vol. 2, no. 3, pp. 243–260, 1998. *Cit. on p. 17.*

[THT+16] S. Thörnqvist, L. B. Hysing, L. Tuomikoski, A. Vestergaard, K. Tanderup, L. P. Muren, and B. J. M. Heijmen, “Adaptive radiotherapy strategies for pelvic tumors – a systematic review of clinical implementations”, *Acta Oncologica*, vol. 55, no. 8, pp. 943–958, 2016. *Cit. on pp. 5, 139.*

[TMU+01] B. S. Teh, W. Y. Mai, B. M. Uhl, M. E. Augspurger, W. H. Grant, H. H. Lu, S. Y. Woo, *et al.*, “Intensity-modulated radiation therapy (IMRT) for prostate cancer with the use of a rectal balloon for prostate immobilization: Acute toxicity and dose-volume analysis”, *International Journal of Radiation Oncology Biology Physics*, vol. 49, no. 3, pp. 705–712, 2001. *Cit. on p. 146.*

[TPB+11] M. Thor, J. B. B. Petersen, L. Bentzen, M. Høyer, and L. P. Muren, “Deformable image registration for contour propagation from CT to cone-beam CT scans in radiotherapy of prostate cancer”, *Acta Oncologica*, vol. 50, no. 6, pp. 918–925, 2011. *Cit. on pp. 5, 16, 140, 153.*

[Tra14] F. Tramnitzke, “GPU based affine linear image registration”, Master’s thesis, University of Lübeck, 2014. *Cit. on pp. 21, 44, 83.*

[TRK+14*] F. Tramnitzke, J. Rühaak, L. König, J. Modersitzki, and H. Köstler, "GPU based affine linear image registration using normalized gradient fields", in *Seventh International Workshop on High Performance Computing for Biomedical Image Analysis (HPC-MICCAI)*, 2014, pp. 1–10. *Cit. on pp. 8, 21, 44, 83.*

[TRU98] P. Thévenaz, U. E. Ruttimann, and M. Unser, "A pyramid approach to subpixel registration based on intensity", *IEEE Transactions on Image Processing*, vol. 7, no. 1, pp. 27–41, 1998. *Cit. on pp. 15, 36.*

[TSC+00] C. Tanner, J. A. Schnabel, D. Chung, M. J. Clarkson, D. Rueckert, D. L. G. Hill, and D. J. Hawkes, "Volume and shape preservation of enhancing lesions when applying non-rigid registration to a time series of contrast enhancing MR breast images", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2000, pp. 327–337. *Cit. on p. 18.*

[TVF+02] I. L. Tan, R. A. Van Schijndel, F. Fazekas, M. Filippi, P. Freitag, D. H. Miller, T. A. Yousry, *et al.*, "Image registration and subtraction to detect active T2 lesions in MS: An interobserver study", *Journal of Neurology*, vol. 249, no. 6, pp. 767–773, 2002. *Cit. on pp. 16, 133.*

[VPM+07] T. Vercauteren, X. Pennec, E. Malis, A. Perchant, and N. Ayache, "Insight into efficient image registration techniques and the demons algorithm", in *20th International Conference On Information Processing in Medical Imaging (IPMI)*, 2007, pp. 495–506. *Cit. on p. 17.*

[VPPA07] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, "Non-parametric diffeomorphic image registration with the demons algorithm", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2007, pp. 319–326. *Cit. on p. 17.*

[VPPA09] ——, "Diffeomorphic demons: Efficient non-parametric image registration", *NeuroImage*, vol. 45, no. 1, pp. S61–S72, 2009. *Cit. on p. 17.*

[VRK+10] J. Vandemeulebroucke, S. Rit, J. Kybic, P. Clarysse, and D. Sarrut, "Spatiotemporal motion estimation for respiratory-correlated imaging of the lungs", *Medical Physics*, vol. 38, no. 1, pp. 166–178, 2010. *Cit. on p. 35.*

[VRS10] J. Vandemeulebroucke, S. Rit, and D. Sarrut, "Deformable image registration with automated motion-mask extraction", in *Grand Challenges in Medical Image Analysis (MICCAI workshop)*, 2010, pp. 119–125. *Cit. on p. 17.*

[VW11] C. Vetter and R. Westermann, "Optimized GPU histograms for multimodal registration", *IEEE 8th International Symposium on Biomedical Imaging (ISBI)*, no. 1, pp. 1227–1230, 2011. *Cit. on pp. 20, 21.*

[VW97] P. Viola and W. Wells, "Alignment by maximization of mutual information", *International Journal of Computer Vision*, vol. 24, no. 2, pp. 137–154, 1997. *Cit. on p. 24.*

[WCV11] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A struture for efficient numerical computation", *Computing in Science & Engeneering*, vol. 13, pp. 22–30, 2011. *Cit. on p. 95.*

[WDO+05] H. Wang, L. Dong, J. O'Daniel, R. Mohan, A. S. Garden, K. K. Ang, D. a. Kuban, *et al.*, "Validation of an accelerated 'demons' algorithm for deformable image registration in radiation therapy", *Physics in Medicine and Biology*, vol. 50, no. 12, pp. 2887–2905, 2005. *Cit. on pp. 17, 140, 146.*

[WES+10] R. Werner, J. Ehrhardt, A. Schmidt-Richberg, A. Hei, and H. Handels, "Estimation of motion fields by non-linear registration for local lung motion analysis in 4D CT image data", *International Journal of Computer Assisted Radiology and Surgery*, vol. 5, no. 6, pp. 595–605, 2010. *Cit. on pp. 16, 17, 19.*

[WFW+97] J. West, J. M. Fitzpatrick, M. Y. Wang, B. M. Dawant, C. R. J. Maurer, R. M. Kessler, J. Macjunas, Robert, *et al.*, "Comparison and evaluation of retrospective intermodality image registration techniques", *Journal of Computer Assisted Tomography*, vol. 21, pp. 554–568, 1997. *Cit. on p. 14.*

[WJK98] S. K. Warfield, F. A. Jolesz, and R. Kikinis, "A high performance computing approach to the registration of medical imaging data", *Parallel Computing*, vol. 24, no. 9-10, pp. 1345–1368, 1998. *Cit. on pp. 20, 21.*

[WK98] H. Wu and Y. Kim, "Fast wavelet-based multiresolution image registration on a multiprocessing digital signal processor", *International Journal of Imaging Systems and Technology*, vol. 9, no. 1, pp. 29–37, 1998. *Cit. on pp. 20, 21.*

[WNA+16] H. Wang, M. Naghavi, C. Allen, R. M. Barber, Z. A. Bhutta, A. Carter, D. C. Casey, *et al.*, "Global, regional, and national life expectancy, all-cause mortality, and cause-specific mortality for 249 causes of death, 1980–2015: a systematic analysis for the Global Burden of Disease Study 2015", *The Lancet*, vol. 388, no. 10053, pp. 1459–1544, 2016. *Cit. on p. 2.*

[WP06] M. P. Wachowiak and T. M. Peters, "High-performance medical image registration using new optimization techniques", *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, no. 2, pp. 344–353, 2006. *Cit. on pp. 20, 21, 34.*

[WS15] O. Weistrand and S. Svensson, "The ANACONDA algorithm for deformable image registration in radiotherapy", *Medical Physics*, vol. 42, no. 1, pp. 40–53, 2015. *Cit. on p. 141.*

[YLL+08] D. Yang, H. Li, D. A. Low, J. O. Deasy, and I. El Naqa, "A fast inverse consistent deformable image registration method based on symmetric optical flow computation", *Physics in Medicine and Biology*, vol. 53, no. 21, pp. 6143–6165, 2008. *Cit. on p. 17.*

[ZF03] B. Zitová and J. Flusser, "Image registration methods: A survey", *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, 2003. *Cit. on p. 15.*

[ZKL+12] H. Zhong, J. Kim, H. Li, T. Nurushev, B. Movsas, and I. J. Chetty, "A finite element method to correct deformable image registration errors in low-contrast regions", *Physics in Medicine and Biology*, vol. 57, no. 11, pp. 3499–3515, 2012. *Cit. on p. 141.*

[ZKN10] D. Zikic, A. Kamen, and N. Navab, "Revisiting Horn and Schunck: Interpretation as Gauss-Newton optimisation", in *British Machine Vision Conference (BMVC)*, 2010, pp. 113.1–113.12. *Cit. on p. 19.*

[ZUC09] X. Zheng, J. K. Udupa, and X. Chen, "Cluster of workstation based nonrigid image registration using free-form deformation", in *SPIE Medical Imaging 2009: Visualization, Image-Guided Procedures, and Modeling*, vol. 7261, 2009, pp. 72611N–72611N–9. *Cit. on pp. 21, 22.*