The design of SUPERelastix– a unifying framework for a wide range of image registration methodologies

Floris F. Berendsen[1], Kasper Marstal[2], Stefan Klein[2], and Marius Staring[1]

[1]Division of Image Processing (LKEB), Department of Radiology, Leiden University Medical Center, P.O. Box 9600, 2300 RC Leiden, the Netherlands, {f.berendsen,m.staring}@lumc.nl
[2]Biomedical Imaging Group Rotterdam Erasmus MC, University Medical Center Rotterdam Departments of Medical Informatics and Radiology, P.O. Box 2040, 3000 CA Rotterdam, the Netherlands, {k.marstal,s.klein}@erasmusmc.nl

Abstract

A large diversity of image registration methodologies has emerged from the research community. The scattering of methods over toolboxes impedes rigorous comparison to select the appropriate method for a given application. Toolboxes typically tailor their implementations to a mathematical registration paradigm, which makes internal functionality nonexchangeable. Subsequently, this forms a barrier for adoption of registration technology in the clinic. We therefore propose a unifying, role-based software design that can integrate a broad range of functional registration components. These components can be configured into an algorithmic network via a single high-level user interface. A generic component handshake mechanism provides users feedback on incompatibilities. We demonstrate the viability of our design by incorporating two paradigms from different code bases. The implementation is done in C++ and is available as open source. The progress of embedding more paradigms can be followed via https://github.com/kaspermarstal/SuperElastix

1. Introduction

The objective of image registration is to find the spatial relationship between two or more images. Typically, intensity-based registration methods are formulated as an optimization problem. A transform describes the geometric mapping from one image to the other, a metric determines the dissimilarity between the images, and an optimizer searches for the transform giving the highest similarity between images.

In the last decades numerous image registration methods and tools have emerged from the research community. Implementation of these methods, however, are scattered over a plethora of toolboxes each with their own interface, limitations and modus operandi. It is therefore difficult for (clinical) researchers and company developers alike, to rigorously compare registration methods and select the appropriate one for a given application.

The scattering is amongst others driven by the difference in paradigms that tend to divide the field of image registration. Opposing approaches are, for example, small deformation versus large diffeomorphic deformation, and non-parametric versus parametric transforms. While most software designs structure the algorithmic concepts of a registration method in a modular fashion, i.e. having a notion of a metric, transform or optimizer, the definition of these modules may not be functionally compatible across paradigms. Current registration software is typically tailored to the paradigm it adheres, mixed with the role of mathematical definitions, data representation and computational tricks. This severely reduces software compatibility across toolboxes, and therefore the ability to share components or truly compare methods rather than implementation choices.

There are several interesting registration toolboxes that we mention here. Arguably the greatest collection of image registration tools can be found in the Insight ToolKit (ITK) [11]. This toolkit implements several registration paradigms, most notably the parametric intensity-based paradigm. A new framework (denoted by the extension v4) partly supersedes the original framework and adds diffeomorphic and symmetric registration capabilities. In addition there are the PDEDeformableRegistration and variational framework, both implementing variations on Demons registration. These paradigms, however, exhibit different object

oriented software designs, which has led to incompatibility of some of the registration components. It is therefore difficult to develop new algorithms that cross the borders of paradigms, as it requires a large effort to re-factor one design into another. In addition, the ITK provides C++ implementations rather than ready-to-use software.

Other works, using and extending ITK, are ANTs [24], [3] [23], `elastix` [14], DRAMMS [18] and plastimatch [20], [19]. Additionally, there are several non-ITK-based implementations including NiftyReg [15].

These tools provide a high-level interface (no need for programming) to configure a registration algorithm. Many options are usually available for each of the registration components. Rigorous evaluation to find the best application specific configuration can, however, only be performed within the specific paradigm or toolbox. This requires the researcher to become acquainted with the various tools, each with their own interface, parameters and peculiarities. Examples of such evaluation studies include [12], [16], [13], [17], [26].

Usability of these tools is greatly improved by environments like Nypipe [9] and Pydpiper [6], which wrap each algorithm as a module in a single python environment. This lowers the barrier for a user as it requires dealing with only one environment instead of multiple different environments (command line, bash scripts, Python, Matlab, etc). A remaining disadvantage however is that each paradigm is still treated as a monolithic block and provides little uniformity for the detailed settings of each algorithm. Analyzing the registration methods for a deeper understanding of the performance differences is limited by the ability to make their settings uniform, since the underlying implementations are still different even for similar components. In addition, these tools provide an interface for Python only, whereas different users require different interfaces.

In this paper we propose a general design for unifying registration paradigms, a translation to a software design and a toolbox with an initial implementation. Specific aims for the toolbox are i) that the interface should be user-friendly; ii) that a single interface can be used for a broad range of registration paradigms; iii) that it should support algorithm modularity; which should iv) lead to the simplification of the tuning of registration configurations optimal for a specific task; and v) enable rigorous comparison of registration methods rather than implementation choices. In such an environment the user can evaluate mathematical (or implementation) choices of specific differences between paradigms, while eliminating the differences of other components. Furthermore, the toolbox should provide a (research) environment enabling exploration of cross-fertilization between paradigms.

At the core of our design there are two observations. First, even for conceptually very different frameworks, many components are functionally identical, albeit that those components are sometimes used differently. And secondly, almost all registration algorithms can be characterized by a network that organizes and connects the several components. While some paradigms strictly split the registration into separate modules consisting for example of a metric, a transform and an optimizer, other paradigms combine components into a larger block that performs the task of both the metric and a transform. Sometimes the components are even so entangled that the registration should be considered a single block. In other words, we need heterogeneous levels of functionality and granularity.

In order to implement a unifying registration toolbox we propose to reformulate registration algorithms into a software design similar to a collaboration-based or role-based software design [25], [21]. In contrast to a typical object-oriented design, a registration component is defined by what it can do (the role) instead of what it is (the class type), thereby allowing utilization of functionality across paradigms. To this end a generic handshake mechanism is implemented to verify whether connected components are compatible on a mathematical as well as on a software level, thus constructing a valid network topology. Via a single user-interface a high level configuration is supplied, from which at run-time the corresponding components are selected, connected into a network and executed.

In the remainder of the paper we perform an analysis of several existing registration paradigms (Section 2), based on which we will propose a design for a unifying registration toolbox (Section 3). Section 4 then gives initial results where we show that the toolbox is operational and allows for specifying, running and comparing quite different registration frameworks from a single toolbox. We end with a general discussion, outlook and conclusion in Section 5.

2. Analysis of registration methodologies

To inventorize the requirements of the design of a unifying toolbox we analyzed a selection of registration paradigms. The selection was made such that it covers some interesting variations among paradigms. Currently, the selection excludes approaches such as discrete optimization [8] [10], groupwise registration [29], geodesic shooting [2] and other Lagrangian or Hamiltonian [28] methods. However, we aim for a general design that can capture a wider span of paradigms, thus keeping possibilities open for integration of those paradigms and additionally future developments. The notation used in this paper is summarized in Table 1.

2.1. Existing methods

The majority of established registration algorithms exhibits distinct conceptual components each performing a sub-task in the registration procedure. Across toolboxes and

paradigms, however, these components can vary in functionality and granularity. Some toolboxes are subdivided into many small components, while others consist of a few larger components. These components can be part of multiple or only a single paradigm. Even if components implement the same concept they may not be interchangeable due to mathematical or software differences.

To expose the typical variation among toolboxes and paradigms, we analyzed 5 registration methods in detail, namely: B-spline registration [14], log-diffeomorphic Demons [1], time varying velocity field registration [23], symmetric log-diffeomorphic Demons [27] and greedy Symmetric Normalization (gSyN) [3]. We created detailed networks to facilitate analysis of these paradigms, see Figure 1.

The first example, as illustrated in Figure 1a, is a registration with a parametric transform, e.g. a B-spline transform which is parameterized by coefficients $\boldsymbol{\mu}$. The parameters $\boldsymbol{\mu}$ span a Euclidean space of possible solutions in which the optimizer searches for the optimal image alignment. The assumption of a Euclidean optimization space versus a manifold (subspace) is fundamental for many optimization strategies. For example, a large group of optimization strategies performs incremental updates of the form $\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k + \boldsymbol{\mu}^k_\delta$, amongst others (unconstrained) gradient descent routines, such as conjugate gradient, adaptive stochastic gradient descent, or quasi-Newton methods.

Variations on Demons registration, such as diffeomorphic Demons (not illustrated) and log-diffeomorphic Demons [5] (Figure 1b) introduce, amongst others, different update rules. Instead of the incremental update $\phi^{k+1} = \phi^k + \phi^k_\delta$ in classical Demons, a functional composition $\phi^{k+1} = \phi^k \circ \phi^k_\delta$ is used in diffeomorphic Demons. Another difference is that where diffeomorphic Demons models the transformation by a vector field pointing from one image to the other, the log-diffeomorphic Demons models such transformations by a stationary velocity field. Instead of an additive or compositional update log-diffeomorphic Demons uses the function $v^{k+1}_{\text{stat}} = \text{BCH}\left(v^k_{\text{stat}}, \phi^k_\delta\right)$.

Table 1: Notation used in this paper.

symbol	description
F	Fixed image
M	Moving image
$T(\boldsymbol{x};\boldsymbol{\mu})$	Parameterized transformation
$\boldsymbol{\mu}$	Transformation parameters
$\delta\boldsymbol{\mu}$	parameter updates
ϕ	deformation field to moving
ϕ^{-1}	deformation field to fixed
δv	forces update field
$S_\phi()$, $S_v()$	diffusion-like regularization
$S_\delta()$	fluid-like regularization
v_t	time varying velocity field
v_{stat}	stationary velocity field
BCH	Baker-Campbell-Hausdorff update rule

While the velocity field of the log-diffeomorphic Demons transform is a stationary vector field, LDDMM-based approaches [4], see Figure 1c, use a time varying velocity field. The modeling of a (virtual) time results in an extra dimension on the velocity field, which can be expressed by parallel paths in the network layout.

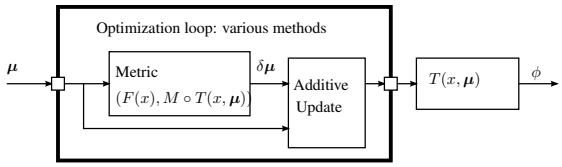
Various registration methods focus on symmetry of the mapping process, so that interchanging the role of the fixed and moving image does not produce different results. Examples of these methods are the symmetric version of log-Demons, see Figure 1d and greedy SyN (gSyN) [3], see Figure 1e. The symmetry imposed by registration methods is typically reflected in the network layout. Whereas gSyN consists of two fully (anti) symmetric paths, the network of symmetric log-Demons exposes only a partial symmetry.

2.2. Analysis

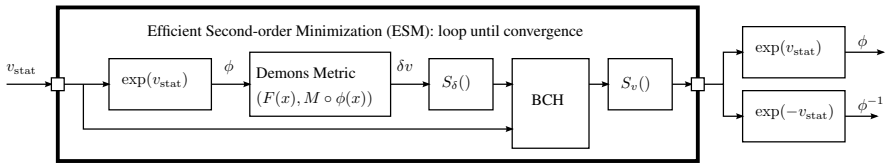
Generalizing from the discussion above, in order to come to a unifying design, we observe the following. Registration algorithms can generally be described as networks of functional components. The network layout of the various registration paradigms, however, are considerably different. Some networks may be summarized as a pipeline of components, whereas others may consist of parallel paths (e.g. symmetric Demons and gSyN) with more or less interconnectivity between the components. Among paradigms, components of a network can be present in other networks as well, although possibly at different locations.

Components also vary in how exchangeable they are between paradigms. Some components are completely paradigm specific, such as the inversion component from gSyN in Figure 1e. Other components can be freely exchanged between paradigms, for example image blurring components for multi-resolution approaches (not shown). In between there is a gray area of components that seem similar task-wise, but due to mathematical differences between paradigms the components are not straightforwardly interchangeable. Examples are in the group of metric components: Demons metrics generally include dedicated terms for the ESM optimization routine which are not present in B-spline registration, and B-spline methods integrally perform a mapping of the metric gradient to the B-spline transform parameters. instead of providing a vector field like in Demons and other diffeomorphic methods. This difference can have a software-related origin as well, for instance when the type of data communicated between components differs, or when toolboxes have a slightly different definition of the boundaries of a component.

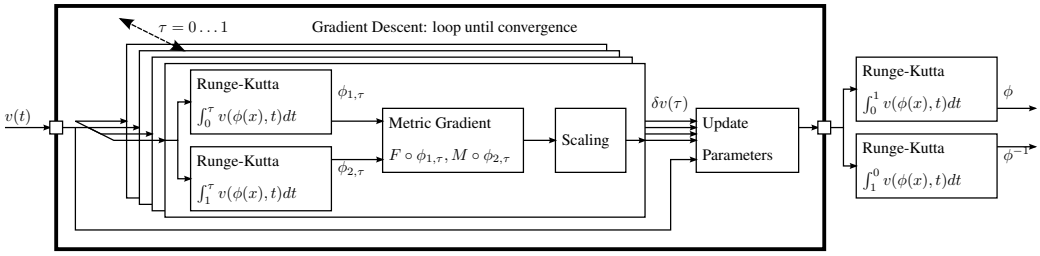
The degree of granularity of paradigms can be different



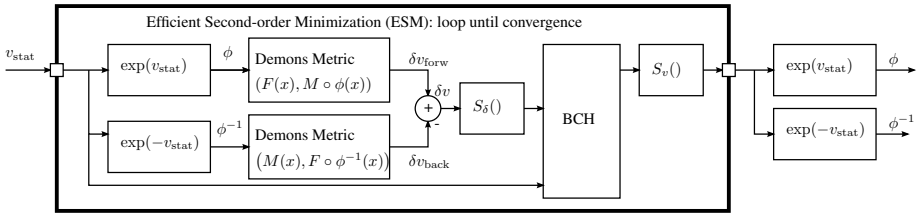
(a) B-spline registration



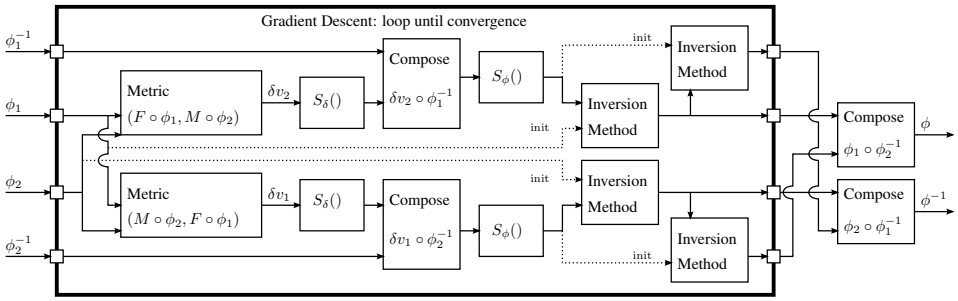
(b) Log-diffeomorphic Demons



(c) ANTs time varying velocity field registration. ANTs' interpretation of LDDMM.



(d) Symmetric log-diffeomorphic Demons



(e) Greedy Symmetric Normalization (gSyN)

Figure 1: Networks of five exemplary registration frameworks.

as well. For example in parametric (e.g. B-spline) registration the optimization strategy is typically regarded as a plug-in component, which can be chosen independently from the transformation model, whereas in gSyN the optimization strategy is inherent to the registration paradigm. To regard components as potentially being composed of multiple tasks has a practical advantage as well, as it would enable integration of complete toolboxes coarse-grained or even monolithically.

To conclude, a design that can capture a broad variety of paradigms 1) needs to be very flexible with respect to the algorithmic network layout and 2) needs to be able to capture a wide variation of components with various levels of granularity. We therefore propose to follow a collaboration-

based or role-based software design pattern [21], in which the definition of a component is defined by the role(s) it can take, rather than by its inheritance, as is used in all current major registration toolboxes.

3. Method

The design we propose consists of three key elements: a) an algorithmic network topology that is completely user-configurable, b) a generic component design that is able to capture a broad diversity of functionalities, and c) a user configuration that dynamically (without programming) sets up the network, performs validation via a handshake mechanism, and establishes a connection between components. In the following sections, we describe the network, how we define components and how they connect, and how the algorithm is executed.

3.1. User configurable component networks

From the analysis of registration algorithms in Section 2 we concluded that although components may be equal among paradigms, the algorithmic networks can be very different. Instead of adhering to one algorithmic network (e.g. `elastix`) or programmatically hard-code a variety of paradigms (e.g. ANTs, plastimatch), we make the network topology part of the user configuration as well. That is, on a high-level the user describes the network layout in terms of nodes and connections. Due to the large variety of possible networks we use a light-weighted notion of a network, which is not intrinsically tied to specific functionality of the components at its nodes. Instead, we choose to define all components to be handled generically and to dispatch the conceptual validation of the user configuration to a handshake mechanism, which is explained next. On a software level the network is modeled as a (boost) graph, where the nodes denote components and the vertices denote connections. Examples of such networks are given in Figure 4.

3.2. Component handshakes

Currently the majority of registration toolboxes adhere a classical object-oriented design that decomposes the registration problem into classes like metrics, transforms, optimizers, etc. Extended types of behavior (mutual information, affine transform, etc) are implemented via subtyping. However, as we concluded in Section 2, this decomposition is generally different among paradigms and toolboxes, and hampers unification. To address the so-called "tyranny of the dominant decomposition" [22], we introduce a role-based software paradigm, similar to the Data Context Interaction (DCI) pattern, for specifying collaboration between components. Building our toolbox around this design pattern, we are able to reuse code bases without suffering from the effects of invasive modification and re-architecture. This design allows to cherry-pick behavioral aspects that components have in common, without a strict classification what that component is.

Fundamental to our design is a generic handshake mechanism that validates whether components can be connected or not. This mechanism performs the necessary checks on what a component can do, which is required to establish a connection. The advantage of explicitly handling this generically and on a higher level, is that components themselves do not need to perform these checks on neighboring components, which would require a component to embed specific knowledge about other components. The proposed design counters the entanglement between the sets of components of the same code base and opens up the potential for cross-paradigm connections.

To manage all possible types of collaboration, we maintain an extensible collection of so-called interfaces between components. Any component in our toolbox must be defined in terms of one or more interfaces, which are either *accepting* or *providing*. Figure 2 illustrates a component based on various interfaces. The user configures a generic connection between the components, while a handshake mechanism determines the types of interfaces and their compatibility. If a connection is possible, the component with the accepting interface gains control over the communication and is responsible for setting up its internals for the execution of the registration algorithm. At the start of the execution of an algorithm all components check if sufficient accepting interfaces have been connected.

From an implementation point of view, all interfaces are defined as abstract base classes, in the DCI pattern, also known as methodless Roles. Via helper classes that are variadicly templated, a component class inherits from any number of interface classes either being accepting or providing. The developer of a component is responsible for implementing all chosen interfaces, i.e. methodful Roles in DCI. In the handshake mechanism compatibility is verified based on the C++ types of the interfaces. While the number of types of interfaces may increase with future developments, the handshake mechanism itself is very general and not that sensitive to changes, due to its loose coupling to component functionality. As an example, a handshake mechanism may be in place where image samplers *provide* a list of samples, while metrics or more aggregated components *accept* a list of samples. When a new sampling component becomes available it does not need to inherit from a base sampling class, which often will require code refactoring, but only needs to define a providing interface.

Within this design the notion of hybrid components (components that fulfill multiple tasks) is captured naturally, by simply proving different accepting or providing interfaces. In this way monolithic blocks that implement a full registration pipeline can also be easily integrated in

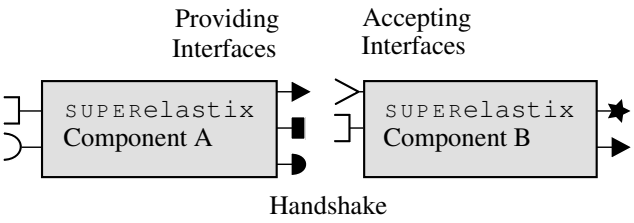


Figure 2: Component handshakes are performed at run-time based on interface types. The toolbox maintains an extensible list of possible interfaces, e.g. ▮◗★▶⧓◆, by which components can connect and collaborate.

the toolbox.

3.3. Algorithm embedding

Whereas previously we described the core functionality of the toolbox, i.e. the setup of components, the network layout and the handshake between components, this subsection describes how this network is embedded in the toolbox, how the algorithm is executed and how data is passed to and from the toolbox.

For a good deployment of our toolbox it will be provided as a library. This can then be naturally integrated in many different software interfaces, such as a command-line application, GUI applications or scripting languages. Figure 3 schematically illustrates how the SUPERelastix library is embedded, which incorporates an Overlord and the registration network. The Configuration Object acts as an intermediate representation of the user-specified configuration of the registration algorithm. This contains the network description, the component names and their settings. In a command-line tool this configuration is read from disk, whereas in library usage it serves as a lightweight container that can be manipulated, passed and stored without requiring disk access.

When a Configuration Object is passed to SUPERelastix the Overlord performs three steps: a) it parses the configuration and instantiates the network with the required components, providing the user with feedback on any incompatibilities between components, as detected by the handshake mechanism, b) it connects the network's Sink and Source components to external data (pipelines), and c) it executes the registration algorithm by connecting itself, via a handshake, to the component that includes the execution trigger.

The Sink and Source component are illustrated by dashed lines in Figure 3 and are special types of components in the sense that the Overlord performs handshakes to these components to let the data pass. By configuring Sink and Source components the user controls whether components should create and pass data such as images, deformation fields or point sets.

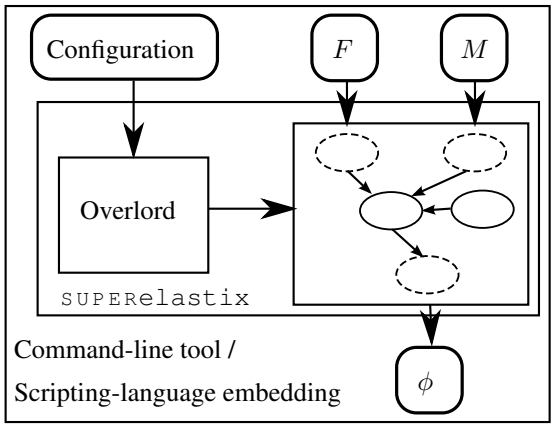


Figure 3: The grand design of the SUPERelastix toolbox. The toolbox can be used as a command-line application or be embedded in other applications and scripting languages as a library. In this illustration F, M and ϕ are examples of typical sources and sinks, but the framework supports any type of source and sink.

4. Results

In this section we demonstrate the initial results of the embedding of two non-rigid registration paradigms each from a different code base, namely the stationary velocity field transform from the ITKv4 registration framework and the B-spline based registration of the elastix toolbox.

To demonstrate that the toolbox provides a single interface to multiple paradigms we perform a small comparison study on synthetic images. The study consists of four experiments, that is, two variations of each paradigm: 1) ITKv4 with a stationary velocity field transform and 1a) a mean squared difference metric (MSD) or 1b) an ANTs neighborhood correlation metric (ANTsC), 2) elastix with a B-spline transform and 2a) a MSD metric, or 2b) a normalized correlation metric (NC).

Currently, the ITKv4 framework is embedded by three components: a main component containing the registration framework plus the transformation model, the MSD metric and the ANTsC metric. With these components two similar algorithm networks can be constructed using each of the two metric components. Currently, elastix is embedded as a monolithic component in which the choice of metric is considered a setting. This illustrates that the framework can deal with different levels of granularity. The network layouts of the experiments, which are automatically generated by SUPERelastix using Graphviz [7], are shown in Figure 4.

For the experiments, synthetic images were chosen such that evident differences between the experiments are expected to show up. As shown in the left column of Figure 6 both the fixed and the moving image have triangular

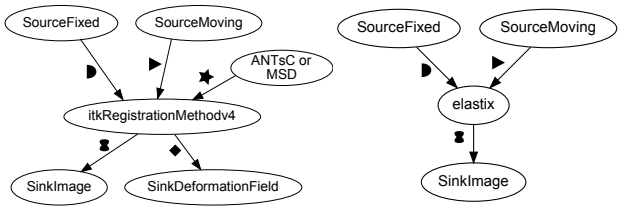


Figure 4: Network configurations for ITKv4 (left) and `elastix` (right). Each symbol (◗▶★▮◆) denotes an interface. The handshake determines that a fixed image interface (◗) is provided by the SourceFixed component and accepted by the itkRegistrationMethodv4 component. Currently, the ITKv4 network can be realized with a ANTsC component or an MSD component that both provide a metric interface (★) that is accepted by the itkRegistrationMethodv4 component. An incorrect configuration, e.g. an ANTsC component connected with a SinkImage component, is detected by the handshake and reported to the user.

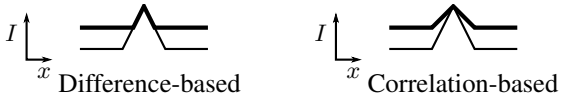


Figure 5: Expected alignment per metric type, with an intensity profile of the fixed image (thin line) and the transformed moving image (thick line).

shaped intensity profiles with the same maximum intensity and a constant but different background. To test the differences due to the non-rigid transformation model of the two paradigms, the fixed image was constructed to have iso-intensity lines that are elliptical whereas they are diamond-shaped in the moving image. Simultaneously, the influence of the image similarity metric was tested by setting the background intensity value to zero for the fixed image compared to a value of one fourth of the maximum for the moving image. For the correlation-based metrics an optimal alignment will be where the bases of the intensity shapes align, irrespective of the absolute value of the background. In contrast, for the squared difference-based metrics, the alignment will be optimal if the triangle profile of the moving image aligns with the top part of the profile of the fixed image, since this minimizes the total absolute difference. This is schematically illustrated in Figure 5. All registration experiments use a multi-resolution approach with 3 levels, using the default settings of the toolboxes.

The results of the four experiments are shown in Figure 6, where both the deformed moving image and the resulting deformation field are given. As expected from our experimental setup the results of each experiment are slightly different. Both experiments with correlation-based metrics show a larger cone shape compared to both squared

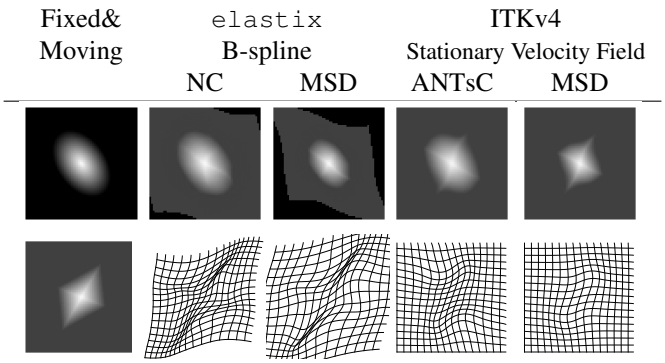


Figure 6: The fixed and moving image and the results of four experiments. The resampled moving image and a transformed grid are shown for each experiment.

difference-based metrics. Between paradigms the differences in deformation fields can be appreciated. For instance, it is clear that the ITKv4 stationary velocity field and the `elastix` B-spline transformation employ different boundary conditions.

By these experiments we showed that registration algorithms from two paradigms can be executed and compared using the same interface for passing the images and algorithm configuration.

5. Discussion and Conclusions

In this paper we propose a design for a unifying toolbox based on the analysis of multiple registration paradigms. Our design uses a role- and collaboration-based software design pattern as a framework to unify the very diverse algorithms found in literature. This framework captures components with various degrees of granularity and functionality, as they are found in most toolboxes, and lets the user configure the network layout of the registration components. To validate the user configuration we set up a handshake mechanism that checks whether connected components are compatible, i.e. have the correct interfaces for collaboration. After a successful configuration the registration algorithm is executed using data from user-defined sources and sinks.

Whereas this paper primarily presents the generic design that we propose, together with an initial implementation and results, the functionality of the toolbox can be greatly extended. An immediate point of action is to extend the functionality of the toolbox by including components from more registration paradigms, and to include more fine-grained components, such as optimizers, transforms and multi-resolution handling. This modularization will obviously lead to more complex network graphs than currently shown in the paper. Furthermore, we aim to include paradigms from non-ITK-based code bases, while keeping in mind component compatibility for cross fertil-

ization of paradigms.

The toolbox is available as open source at https://github.com/kaspermarstal/SuperElastix where the latest version can be found and where all developments can be followed.

The design we presented is both capable of generalizing across paradigms, fully supporting modularity, as well as provides an on-ramp for method integration. This makes it a suitable foundation for a unifying registration toolbox, which, as we hope, should drastically improve the accessibility of a wide range of modern registration capabilities to a diverse audience.

Acknowledgment

The authors acknowledge support from the Dutch Technology Foundation STW (Stichting Technische Wetenschappen), grant number 13351, and the Netherlands Organisation for Scientific Research NWO (Nederlandse Organisatie voor Wetenschappelijk Onderzoek), grant number 184033111.

References

[1] J. Ashburner. A fast diffeomorphic image registration algorithm. *Neuroimage*, 38(1):95–113, 2007. 3

[2] J. Ashburner and K. J. Friston. Diffeomorphic registration using geodesic shooting and Gauss-Newton optimisation. *Neuroimage*, 55(3):954–967, 2011. 2

[3] B. B. Avants, C. L. Epstein, M. Grossman, and J. C. Gee. Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain. *Med Image Anal*, 12(1):26–41, 2008. 2, 3

[4] 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*, 61(2):139–157, 2005. 3

[5] F. Dru and T. Vercauteren. An ITK implementation of the symmetric log-domain diffeomorphic demons algorithm. *Insight Journal*, 2009. 3

[6] M. Friedel, M. C. van Eede, J. Pipitone, M. M. Chakravarty, and J. P. Lerch. Pydpiper: a flexible toolkit for constructing novel registration pipelines. *Front Neuroinform*, 8:67, 2014. 2

[7] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11):1203–1233, 2000. 6

[8] B. Glocker, A. Sotiras, N. Komodakis, and N. Paragios. Deformable medical image registration: setting the state of the art with discrete methods. *Annu Rev Biomed Eng*, 13:219–244, 2011. 2

[9] K. Gorgolewski, C. D. Burns, C. Madison, D. Clark, Y. O. Halchenko, M. L. Waskom, and S. S. Ghosh. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Front Neuroinform*, 5, 2011. 2

[10] M. Heinrich, M. Jenkinson, S. M. Brady, and J. Schnabel. Globally optimal deformable registration on a minimum spanning tree using dense displacement sampling. *Image Comput Comput Assist Interv*, 15, Pt 3:115–22, 2012. 2

[11] Johnson, McCormick, and Ibanez. *The ITK Software Guide: Design and Functionality*. Kitware Inc., fourth edition, 2015. 1

[12] T. Kanai, N. Kadoya, K. Ito, Y. Onozato, S. Y. Cho, K. Kishi, S. Dobashi, R. Umezawa, H. Matsushita, K. Takeda, and K. Jingu. Evaluation of accuracy of B-spline transformation-based deformable image registration with different parameter settings for thoracic images. *J. Radiat. Res.*, 55(6):1163–1170, 2014. 2

[13] A. Klein, J. Andersson, B. A. Ardekani, et al. Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration. *Neuroimage*, 46(3):786–802, 2009. 2

[14] S. Klein, M. Staring, K. Murphy, M. A. Viergever, and J. P. Pluim. elastix: a toolbox for intensity-based medical image registration. *IEEE Trans Med Imaging*, 29(1):196–205, 2010. 2, 3

[15] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, and S. Ourselin. Fast free-form deformation using graphics processing units. *Comput Methods Programs Biomed*, 98(3):278–284, 2010. 2

[16] K. Murphy, B. van Ginneken, J. M. Reinhardt, et al. Evaluation of registration methods on thoracic CT: the EMPIRE10 challenge. *IEEE Trans Med Imaging*, 30(11):1901–1920, 2011. 2

[17] Y. Ou, H. Akbari, M. Bilello, X. Da, and C. Davatzikos. Comparative evaluation of registration algorithms in different brain databases with varying difficulty: results and insights. *IEEE Trans Med Imaging*, 33(10):2039–2065, 2014. 2

[18] Y. Ou, A. Sotiras, N. Paragios, and C. Davatzikos. DRAMMS: Deformable registration via attribute matching and mutual-saliency weighting. *Med Image Anal*, 15(4):622–639, 2011. 2

[19] J. A. Shackleford, N. Kandasamy, and G. C. Sharp. On developing B-spline registration algorithms for multi-core processors. *Phys Med Biol*, 55(21):6329–6351, 2010. 2

[20] 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. *Phys Med Biol*, 52(19):5771–5783, 2007. 2

[21] Y. Smaragdakis and D. Batory. Mixin layers: An object-oriented implementation technique for refinements and collaboration-based designs. *ACM Trans. Softw. Eng. Methodol.*, 11(2):215–255, 2002. 2, 5

[22] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 107–119, 1999. 5

[23] N. J. Tustison and B. B. Avants. Explicit B-spline regularization in diffeomorphic image registration. *Front Neuroinform*, 7:39, 2013. 2, 3

[24] N. J. Tustison and S. G. Avants, B. B. Advanced normalization tools: V1.0. *Insight Journal*, 2009. 2

[25] M. VanHilst and D. Notkin. Using role components to implement collaboration-based designs. *ACM SIGPLAN Notices*, 31(10):359–369, 1996. 2

[26] R. Varadhan, G. Karangelis, K. Krishnan, and S. Hui. A framework for deformable image registration validation in radiotherapy clinical applications. *J Appl Clin Med Phys*, 14(1):4066, 2013. 2

[27] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache. Symmetric log-domain diffeomorphic registration: a demons-based approach. *Med Image Comput Comput Assist Interv*, 11(Pt 1):754–761, 2008. 3

[28] F.-X. Vialard, L. Risser, D. Rueckert, and C. J. Cotter. Diffeomorphic 3D image registration via geodesic shooting using an efficient adjoint calculation. *International Journal of Computer Vision*, 97(2):229–241, 2011. 2

[29] C. Wachinger and N. Navab. Simultaneous registration of multiple images: similarity metrics and efficient optimization. *IEEE Trans Pattern Anal Mach Intell*, 35(5):1221–1233, 2013. 2