NTNU

Norwegian University of
Science and Technology

# Registration in hyperspectral and multispectral imaging

Stig Viste

Master of Science in Electronics
Submission date: January 2017
Supervisor: Lise Lyngsnes Randeberg, IES
Co-supervisor: Asgeir Bjørgan, IET

Norwegian University of Science and Technology
Department of Electronic Systems

# Abstract

Hyperspectral (HSI) and Multispectral (MSI) images consist of multiple images taken at different wavelengths (bands), often at different times. Analysis of these images require high accuracy of correlation between the different image bands. As the image bands are taken at different times, different positions, or both, this entails that the images must be processed before the images can be analysed. The process of alignment between two images is referred to as image registration.

A test-set consisting of 40 image bands was created, consisting of wavelengths in the range of (415-557nm). These images are 1601x1401 pixels, and have introduced simulating errors, to be corrected by the registration process. This enables validation of the registration results. Insight Toolkit (ITK), a `C++` library, was explored and implemented for this test-set. ITK was chosen, as this library contains most known registration methods, and boast high customizability. Six registration methods were implemented and tested against this image set.

One of these registration methods, Rigid transform, was found to achieve sub-pixel precision across multiple input parameters, with 0.22 standard deviation from the true pixel coordinates, and 0.018 standard deviation from the true angular value. This method has an optimal run time of six hours per image band using one central processing unit (CPU) core. The remaining five registration methods showed promise, but were not reliable against the test-set. These registration methods ranged between 33 and 76 standard deviation from the true pixel coordinates.

Rigid transform was then run on the unknown hyperspectral and multispectral image sets. Quantifiable accuracy results are not available for these image sets.

# Sammendrag

Hyperspektrale (HSI) og Multispektrale (MSI) bilder består av flere bilder tatt ved forskjellige bølgelengder (bånd), ofte ved forskjellige tidspunkt. Analyse av disse bildene krever høy nøyaktighet og korrelasjon mellom de forskjellige bildebåndene. Ettersom bildene er tatt ved forskjellige tidspunkt, forskjellige posisjoner, eller begge, innebærer dette at bildene må prosesseres før bildene kan analyseres. Justering mellom to bilder er en prosess omtalt som bilderegistrering.

Det ble laget et testsett med 40 bildebånd, med bølgelengder i området (415-557nm). Disse bildene er 1601x1401 piksler. Det ble lagt inn simulerte feil, som skal korrigeres for av registreringsprosessen. Dette muliggjør validering av resultatene fra registreringsprosessen. Insight Toolkit (ITK) er et `C++` bibliotek som ble utforsket og implementert for testsettet. ITK ble valgt, da biblioteket inneholder de mest kjente og utprøvde registreringsmetodene, og har gode tilpasningsmuligheter. Seks registreringsmetoder ble implementert og testet med bildesettet.

En av disse registreringsmetodene, Rigid transform, oppnådde presisjon under en piksel over flere parametre, med 0.22 standardavvik fra korrekte pikselkoordinater, og 0.018 standardavvik fra korrekt vinkelverdier. Denne metoden hadde en optimal kjøretid på seks timer per bildebånd med en prosessorkjerne. De gjenværende fem registreringsmetodene var lovende, men var ikke pålitelige mot testsettet. Disse registreringsmetodene hadde standardavvik for korrekte pikselkoordinater mellom 33 og 76.

Rigid transform ble deretter benyttet mot ukjente hyperspektrale og multispektrale bildesett. Kvantifiserbar nøyaktighet for resultatene er ikke tilgjengelig for disse bildesettene.

# Preface

This report is the result of the Master's thesis conducted during the fall of 2016 and january 2017, concluding a Master of Science degree in Electronics, Nanoelectronics and Photonics. The report is submitted to the Department of Electronics and Telecommunications (IET) at the Norwegian University of Science and Technology (NTNU). The work is an intersection between modelling, medicine and computer science.

The work in this thesis is a module in a larger project developed in previous master thesises[11, 90]. This project was proposed by professor Lise L. Randeberg from IET at NTNU. During the work with this thesis, I have learned a lot about the concepts, challenges and limitations image processing faces.

I would like to thank my supervisors professor Lise L. Randeberg and PhD candidate Asgeir Bjørgan, for their support, guidance and feedback throughout this project. Thanks to the board members at Omega Verksted for lending me hardware, and introducing the concept of timeshare to proof-reading. I also want to thank Eline for letting me vent about the seemingly impossible issues at hand and proof-reading. Thanks to my parents for showing me the door at the ripe age of nineteen.

Stig Viste
January 2017
NTNU Trondheim

# Contents

# List of Figures

# List of Tables

# Acronyms and Abbreviations

| | |
|---|---|
| **AOTF:** | Acousto-optical tunable filter |
| **ANTs:** | Advanced Normalization Tools |
| **API:** | Application Programming Interface |
| **BIL:** | Band Interleaved by Line |
| **BIP:** | Band Interleaved by Pixel |
| **BSQ:** | Band Sequential |
| **CP:** | Control Point |
| **CPU:** | Central Processing Unit |
| **DCMTK:** | DICOM Toolkit |
| **ENVI:** | Environment for Visualization |
| **Fixed:** | Image to be compared to moving image |
| **FPGA:** | Field Programmable Gate Array |
| **GIMIAS:** | Graphical Interface for Medical Analysis and Simulation |
| **GPU:** | Graphics Processing Unit |
| **HSI:** | Hyper Spectral Imaging |
| **IET:** | Department of Electronics and Telecommunications |
| **ITK:** | Insight Toolkit |
| **LBFGS:** | Limited memory Broyden, Fletcher, Goldfarb and Shannon minimization |
| **MATIO:** | MAT File I/O Library |
| **MITK:** | Medical Imaging Interaction Toolkit |
| **Moving:** | Image to be registered |
| **MMI:** | Mattes Mutual Information |
| **MI:** | Mutual Information |
| **MR:** | Magnetic Resonance |
| **MS:** | Mean Squares |
| **MSI:** | Multispectral Imaging |
| **NC:** | Normalized Correlation |
| **NMI:** | Normalized Mutual Information |
| **NTNU:** | Norwegian University of Science and Technology |
| **OpenCV:** | Open Computer Vision |
| **PET:** | Positron Emission Tomography |
| **SPECT:** | Single Photon Emission Computed Tomography |
| **VTK:** | Visualization Toolkit |
| **XML:** | Extensible Markup Language |

# 1 Introduction

Hyperspectral imaging (HSI) is a technique where each pixel represents the whole spectrum visible to the camera for all the images, instead of intensity values for red, green and blue. This technique combines high spectral and spatial information in one image, divided in image bands, where each image band consist of an image taken at a specific wavelength[16]. Multispectral imaging (MSI) is a similar technique, where the whole visible spectrum is replaced with specific wavelengths[9]. While hyperspectral imaging is widely used for remote sensing, it has also been adopted for diagnostic purposes in imaging of skin. Various techniques have been developed for extraction of the information[15, 82, 84, 91].

Using these methods, it is possible to attain both spatial and spectral information from an object. HSI are three-dimensional images, where the third dimension is spectral. These images contain a large quantity of information, and multicore CPU, GPU, or FPGA processing is often used to allow fast processing of the different images, decreasing the run time[11, 12, 14].

The human skin is the largest organ, covering the entire body. Skin is divided into layers; epidermis, dermis and hypodermis. Epidermis is the outer layer, and protects the skin against water loss and external organisms and stimuli. When exposed to UV-radiation, the epidermis will produce melanin for protection, and the melanin will affect the tone of the skin. Dermis mainly consists of connective tissue and blood vessels. This layer supplies nourishment to the skin. Hypodermis contains fat cells.[29] Light can penetrate through the skin layers and be reflected back. The reflectance spectrum of the skin layers depends on the wavelength of the exposed light[3, 55].

A hyperspectral image, of for example an arm, will therefore contain a large quantity of information about the different skin layers in the arm. This information may be extracted and used for various purposes; e.g. determining the age of a bruise[83], characterization of vascular structures[98], and characterizing wounds[22].

## 1.1 Motivation

Registration is the process of finding the spatial transform that maps points from one image to the corresponding points in another image. Medical image registration has many applications both in clinical and research settings. Larger changes in medical images taken at different times may be detected by visual comparison, for example PET scans taken months apart. Image registration enables the detection of subtle changes, by eliminating patient position, movement and motion artifacts. Optionally, one may remove everything that hasn't changed between images, using subtraction. Registration may also be applied to different imaging modalities, referred to as coregistration, such as combining the information from PET, SPECT and MR images[13, 109].

Unless the image bands are correctly aligned, any information extracted will be void. HSI is not instant, but takes some time per image band or image line, depending on the imaging technique used[11, 59, 90]. Motion artifacts typically occur when a scanning based technique is used[90]. Then, the camera scans one image line at all wavelengths at a time. Movement in the object between two image line scans will skewer the object in the image. With techniques using full spatial images taken at different wavelengths at different times, small movements and positioning of the object will impact the final image. Any such eventual movement, positioning and motion artifacts must be detected and eliminated before HSI analysis is used. Multispectral images are here taken at different angles, but at the same time. Thus, these images face a different, but similar, set of restrictions.

This means that we have data sets with alignment errors of different magnitudes, deformably erroneous images, and possibly a combination.

## 1.2 Previous work

As the time consumption of the camera scan rate and data processing approaches zero, movement and motion artifacts will approach extinction. This has been attempted with FPGAs[34, 102] and GPUs[11, 30, 31, 87, 92, 93, 95]. One of the data sets is taken using a HySpex VINIR-1600 camera[43]. This camera, with autofocus[90], uses 30 ms per line. With GPU acceleration, the data processing uses 3.5 ms per line[11], which makes the camera the bottleneck. While this reduces the amount of artifacts greatly, it is still possible for the object to move enough to induce artifacts between image lines. The severity of these artifacts depend on how and how much the object moves.

Image registration has been thoroughly tested with different implementations, such as CT scans[6, 106, 108] and PET scans[1, 74] using landmark recognition, rigid transformation in the spatial domain[8], and coregistration between different image modalities[2, 17, 18, 27, 33, 35, 37, 38, 40, 60, 61, 70, 74]. Several surveys[13, 58, 62–64, 86, 110] have been published, discussing different registration methodology and implementations. Although many of the publications discuss different topics, they all agree on two things;

- A perfect image registration method does not exist. It is up to the user to weigh if the registration method is adequate.
- No method is available for qualitative validation of a registration method. If such a technique existed, that method would be used in place of the registration method. Approaches for validation involve testing the registration method on simulated images, and relating these results to images that are similar to the simulated images.

## 1.3 Objectives

The objectives in this master thesis can be categorized as following;

- Implement reading, writing, and registration methods for the supplied data sets, based on literature on the subject.
- Test and discuss the implemented registration methods, based on the registration results.
- Validation of the registration methods

Additionally, methods for choosing parameters and registration method will be created. This will make the developed software more user-friendly.

## 1.4 Contributions

The intention of this work has been to create an adapted version of open source image registration software, targeted towards hyper spectral images from an Acousto-Optic Tunable Filter (AOTF) camera[73], a pushbroom camera[11, 90], a Zyla 5.5 sCMOS camera[59], and multispectral images from a five-camera system[39]. In order to achieve this, Insight Toolkit (ITK)[52] was used as base.

The following list summarize the contributions made through this thesis:

- Image reader modules have been created. These modules allow multiple inputs as arguments to the program.
- A framework for easy configuration has been developed. This framework allow for fast reconfiguration of the image registration process, including registration method, preprocessing filters and parameters for the optimizer, metric and transformation.
- Implementation of six different image registration methods have been created, where four of the image registration methods are rigid, and two of the image registration methods are deformable.
- Preprocessing filters have been implemented, allowing median and/or gradient filtering of the images to be registered.
- A test-set of images has been created, giving a method for validating the accuracy of the implemented image registration methods.

## 1.5 Method

The work performed in this thesis is based on multiple research methods. Before the problem could be solved, a study of the image formats, the ITK library, the MATIO[65] library, and image registration techniques had to be conducted. A plan for how to resolve each of the issues at hand was devised and discussed before being carried out, in order to ensure a good solution. The problems at hand requires in-depth knowledge of the ITK library, as well as an understanding of image registration concepts, techniques and issues.

A temporary program was developed in order to convert the input image files into standardized image files, and tested with the different image registration methods available in the ITK library. The methods with the best registration results were then implemented in a more adaptive and configurable program, combining the modules directly, reducing computational cost and memory usage. A subset of images was extracted from an image set, with introduced errors. This image subset was tested with the different registration methods using different parameters in order to find the accuracy of the registration method.

In general, this thesis is divided into six chapters, in addition to four appendices. In chapter 2, the theory and background required to understand the content of the thesis is described. Chapter 3 contains descriptions of the implementation and development of the program. All relevant results are presented in chapter 4. These Results are discussed against the validation criteria found in the literature[110] in chapter 5, and where applicable, a conclusion has been drawn in chapter 6. Chapter 6 also include a section of future works, describing aspects that are interesting to look into, as well as aspects that should be looked into. The appendices includes code-listings of the header files in the program, example source code, inheritance diagrams, and includes overviews of hardware and software used.

# 2 Theory and background

This chapter includes some theory and background information explaining different aspects and issues affiliated with image registration. Chapter 2.1 provides some theoretical background information for the images provided. Chapter 2.2 aim to provide in-depth information about the image registration process as well as common algorithms and methods. Chapter 2.3 gives a short overview over readily available registration software, while chapter 2.4 gives in-depth information about ITK and its implementations and adaptions of the methods presented in chapter 2.2.

## 2.1 Image formats

Image files are composed of digital data in a file format. Many of these file formats are standardized, such as `TIFF`, `PNG` and `GIF`. These standardized formats store the image metadata in a known manner, typically as part of the image file, in a header at the beginning of the file. This image file may then easily be interpreted and read.[88]

When storing multiple images in one digital file, the different images are referred to as bands. Each band has its own corresponding identifier. This identifier may be the wavelength $\lambda$ the image is taken with, or the camera it is taken from.

### 2.1.1 Band Interleaved by Pixel

Band Interleaved by Pixel (BIP) stores the first pixel for all bands in sequential order, followed by the next pixel for all the bands, interleaved up to the number of pixels. For accessing pixel number 45 in band 3, accessing a pixel then requires the knowledge of number of bands. If there are 90 bands, or images, accessing pixel number 45 in band 3 is then stored at position $44 * 90 + 45$ in a Band-Interleaved-by-Pixel image. BIP offers optimal performance for spectral access[10]. This is visualized in table 2.1.

| | Pixel 1,1 | | | ⋯ | Pixel 1,n | | |
|---|---|---|---|---|---|---|---|
| Row 1 | Band 1 | Band 2 | Band 3 | ⋯ | Band 1 | Band 2 | Band 3 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋱ | ⋮ | ⋮ | ⋮ |
| Row n | Band 1 | Band 2 | Band 3 | ⋯ | Band 1 | Band 2 | Band 3 |

Table 2.1: Band Interleaved by Line

### 2.1.2 Band Sequential

Band Sequential (BSQ) stores the spatial image of each band in a sequence. It follows that BSQ offers optimal performance for spatial access, as all pixels representing the same spatial space is stored sequentially[10]. This is visualized in table 2.2.

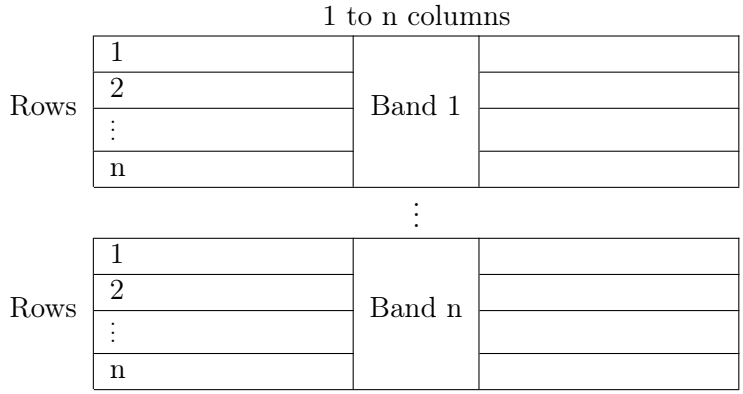



Table 2.2: Band Sequential

### 2.1.3 Band Interleaved by Line

Band Interleaved by Line, or BIL, stores the first line for all the bands, then the next line for all the bands, until the last line is stored. If one wants to access pixel number 45, line 1, in band 3, one must first know the size of the line. If the line is 1000 pixels wide, pixel number 45, line 1, band 3 is then stored at position $1000 * 2 + 45$ in a Band-Interleaved-by-Line image. BIL is a compromise between BIP and BSQ, and offers good performance for both spectral and spatial access[10]. This is visualized in table 2.3.

| | 1 to n columns | 1 to n columns | 1 to n columns |
|---|---|---|---|
| Row 1 | Band 1 | Band 2 | Band 3 |
| Row 2 | Band 1 | Band 2 | Band 3 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| Row n | Band 1 | Band 2 | Band 3 |

Table 2.3: Band Interleaved by Line

### 2.1.4 ENVI Standard

The Environment for Visualization (ENVI) standard file format[26] is an image format that supports BIL, BIP and BSQ directly. It stores the image data in a binary file, and information about the storage format, data type, image size and band information in a separate header file. The header file may be easily accessed in a text viewer, or by an application capable of reading clear text. The supplied HSI in this thesis are of this format, stored using BIL.

### 2.1.5 RAW

Images stored in a RAW format do not have any information about the image stored in the file. Thus, information about the data type and image dimensions must be known before the image may be extracted. The RAW format is, as the name depicts, an unaltered version of the image. This means no compression, which means no loss of accuracy and precision. The MSI images analyzed in this thesis is stored as BSQ in separate files.

## 2.2 Image registration

Image registration is a fundamental task in image processing used to match two or more images. These images may be taken at different times, from different sensors, from different viewpoints, or at different wavelengths. Most large systems which evaluate images require some level of registration of the images. Over the years, a broad range of techniques have been independently studied and developed for several applications[13, 110].

The extremely simplified registration problem in figure 2.1 illustrates a registration process where the same object has been pictured at different viewpoints. In this case, the object has moved closer to the camera.

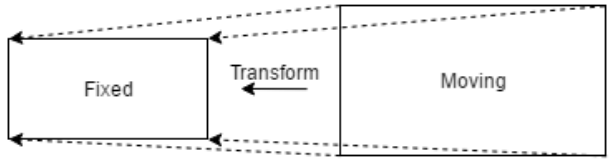



Figure 2.1: Simple registration problem

### 2.2.1 Concepts

In the existing literature, different criteria has been used as basis for aligning two images. In general, these criteria can be categorized as landmark-, segmentation-, and intensity-based[28, 62, 103].

Landmark-based registration uses salient features, selected by the user. These features may be chosen as points, lines or more complex structures such as corners or crossings. This technique is fast to compute, as the number of features are sparse compared to the full content of the image. However, this method requires user interaction for location of the features. This lacks consistency and reproducibility, in addition to adding more complexity to the usage.

Segmentation-based methods attempt to align the binary structure, i.e. curves, surfaces or volumes, rigidly or deformably through obtained information from segmentation. The segmented structure of one image may be aligned to a segmented structure in the second image or to the unsegmented second image. This criteria typically requires that the boundaries of the binary structure matches the edges in the second image. As segmentation-based methods reduce the overall image information, segmentation yields faster computation than registration using the full content of the image. One drawback of segmentation-based methods is that the performance relies on the accuracy of the pre-processing step.

Intensity-based registrations operate directly on the image intensity. They are more flexible as they use all the available information without reduction of data from user input or automated segmentation algorithms. This is computationally expensive, and hence not suited for time-constrained applications. It is therefore common to use a multi-resolution approach in order to speed up the computational time and improve the capture range of the algorithm[109].

Popular approaches include correlation ratios and the information theoretic measure of mutual information [21, 105].

### 2.2.2 Process

Regardless of the methods used, the majority of registration methods consists of the following steps[64, 86, 89, 110];

- Feature detection
- Feature matching
- Tranform model estimation
- Image resampling and transformation

For the first step, salient and distinctive objects are detected. These objects may be regions, edges, contours, intersections, corners, etc, as described in chapter 2.2.1, and may be detected manually or automatically. For further processing, these features need to be represented by their point representatives. These representatives are called control points (CP) in the literature.

The second step is matching these CPs between the moving and fixed images, and establish this correspondence for further use. Various feature descriptors and similarity measures along with spatial relationships in the features are used.

Next, a transformation model is estimated using the established feature correspondence. When the transformation model is estimated, the moving image is finally transformed using the transformation model. Any image values in non-integer coordinates need to be computed by an appropriate interpolation technique before the image may be resampled.

The basic components of a registration framework are depicted in figure 2.2, and consists of two input images, a transform, a metric, an interpolator, and an optimizer.

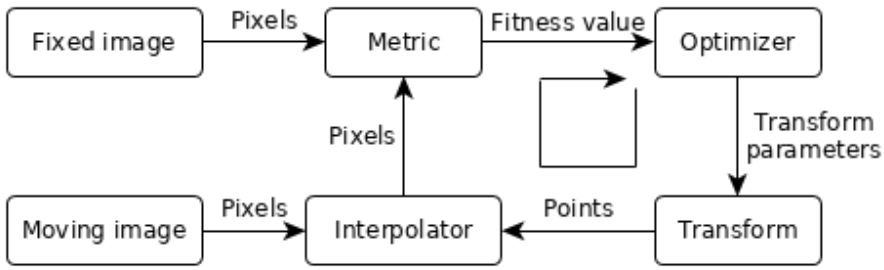



Figure 2.2: Registration framework

### 2.2.3 Framework

The optimizer will iteratively search for the optimal solution by evaluation of the metric at different positions of the transform parameter search space. The basic input to an optimizer is a metric object or a cost function. The metric is initialized, and the optimization algorithm started. Once the optimization stop condition has been reached, the transform and final parameters are ready for extraction[54, 77].

The objective of an initializer is to simplify the computation of the center of rotation and the translation required to initialize certain transforms. The initializer accepts two images and a transform as inputs, and returns an initial start position for the metric.

Interpolation is a technique for determining the value between two prescribed values, and it is believed that the Babylonians used linear interpolation for determining the movement of astronomical objects in the centuries before the common era[75]. Several interpolation methods are simple, yet effective. Nearest neighbor is simple rounding; If $x = 0$ contains the value 7, and $x = 1$ contains the value 230, $x = 0.49$ will contain value 7. Just as nearest neighbour assumes discrete values, linear interpolation never assumes discrete values, and is the straight line between the prescribed values. For the coordinates $(x_0, y_0)$ and $(x_1, y_1)$, equation 2.1 describes this line.

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0} \tag{2.1}$$

For the previously used $x$ values, $x = 0.49$ contain value $7 + (230 - 7) * 0.51 = 120.73$[72].

### 2.2.4 Metrics

The metric component is a critical element in the registration framework, and the selection of which metric to use is highly dependent on the registration problem to be solved. The matching metric controls most parts of the registration process as it handles fixed, moving and virtual images as well as fixed and moving transforms and interpolators. Typically, the metric samples points within a defined region of the virtual lattice. For each point, the corresponding fixed and moving image positions are computed using the initial transform and the moving transform with specified parameters.

Mean squares is a simple adaption of Mean Squared Error commonly used in probability. Equation 2.2 computes the mean squared pixel-wise difference in intensity between image A and image B over a user-defined region. This metric is simple to compute and has a large capture radius.

$$MS(A,B) = \frac{1}{N}\sum_{i=1}^{N}(A_i - B_i)^2 \tag{2.2}$$

This metric relies on the assumption that intensity representing the same homologous point must be the same in image A and B, and is thus restricted to images of the same modality. Any linear changes in the intensity result in a poor match value. Poor matches result in large values, whereas the optimal value of the metric is zero.

Mutual information (MI) is defined in terms of entropy[80, 94]. $H(A)$ and $H(B)$ are the entropies of random variables $A$ and $B$ such that

$$H(X) = -\int p_X(x)\log(p_X(x))dx \tag{2.3}$$

Then the joint entropy of $A$ and $B$ becomes

$$H(A,B) = \int p_{AB}(a,b)\log(p_{AB}(a,b))dadb \tag{2.4}$$

If $A$ and $B$ are independent, then

$$H(A,B) = H(A) + H(B)$$

If $A$ and $B$ are dependent, then

$$H(A,B) < H(A) + H(B)$$

The difference, $I(A,B)$, is the Mutual Information;

$$I(A,B) = H(A) + H(B) - H(A,B) \tag{2.5}$$

MI measures how much information one random variable in image $A$ tells about another random variable in image $B$. An advantage of MI is that the form of the dependency does not have to be specified. This allows modelling of complex mapping between two images, and is thus well suited for multimodal registration.

Equation 2.6 computes pixel-wise cross-correlation and normalizes it by the square root of the autocorrelation of the images.

$$NC(A,B) = -1 \cdot \frac{\sum_{i=1}^{N}(A_i \cdot B_i)}{\sqrt{\sum_{i=1}^{N} A_i^2 \cdot \sum_{i=1}^{N} B_i^2}} \tag{2.6}$$

The $-1$ factor is used to optimize the metric when its minimum is reached. Optimal value is hence $-1$, and misalignment result in small measure values. The use of this metric is limited to images with the same image modality. The metric has a relatively small capture radius, and is insensitive to multiplicative factors between image A and B.

Changes in overlap of very low-intensity regions of an image can disproportionately contribute to the mutual information. This may be solved using an alternative normalization, as shown in equation 2.7. This measure involves normalizing mutual information with respect to the joint entropy of the overlap volume[41, 97].

$$NMI(A,B) = 1 + \frac{I(A,B)}{H(A,B)} = \frac{H(A) + H(B)}{H(A,B)} \tag{2.7}$$

When images A and B have virtually identical fields of view, mutual information and normalized mutual information have been shown to perform equivalently[42].

### 2.2.5 Transforms

Transforms encapsulate the mapping of points and vectors from an input space to an output space. The mapping may support simple translation, rotation and scaling. Transforms are general and can be used for applications other than registration.

The concept of demons was introduced by Maxwell in the 19th century in order to illustrate a paradox of thermodynamics. Assume a gas composed of a mixture of two types of particles $a$ and $b$, separated by a semi-permeable membrane containing a set of "demons". These demons distinguish between the particles, and only allow $a$ particles to diffuse to the first side, and $b$ particles to diffuse to the second side. In the end, the first side only contains $a$ particles, and the second side only contains $b$ particles. This corresponds to a decrease of entropy, which contradicts the second principle of thermodynamics. As the demons generate a higher amount of entropy to recognize the particles, the total entropy increases, and the paradox is solved.

Demons is a diffusion model transformation, using the concept of attraction. A point $P$ in the model $M$ is attracted by all the points $P'$ in $S$ which are similar. If $K(P, P')$ is a similarity criterion, and $D(P, P')$ a function of the distance, the induced force $\vec{f}$ on $P$ by the attraction of all the points of $S$ can be written as

$$\vec{f}(P) = \sum_{P' \in S} \frac{K(P, P')}{D(P, P')} P\vec{P}' \tag{2.8}$$

$M$ is thus deformed according to these forces.

A special method is optical flow, which may be used to find small deformations in temporal sequences of images. At a given point $P$, $s$ is the intensity function in $S$ and $m$ the intensity in $M$. The basic hypothesis of optical flow is to consider that the intensity of a moving object is constant with time, which for small misplacement give the optical flow equation;

$$\vec{v} \cdot \vec{\nabla} s = m - s \tag{2.9}$$

In order to sufficiently define the velocity $\vec{v}$, it is possible to consider that the end point of $\vec{v}$ is the closest point of the hypersurface $m$, with respect to spatial $(x, y, z)$ translations, and in turn obtaining local values of $\vec{v}$;

$$\vec{v} = \frac{(m - s)\vec{\nabla} s}{(\vec{\nabla} s)^2} \tag{2.10}$$

This equation is unstable for small values of $\vec{\nabla} s$, giving infinite values for $\vec{v}$. Ideally, small $\vec{\nabla} s$ should give solutions close to zero. A solution to this problem may be found by multiplying equation 2.10 with $(\vec{\nabla} s)^2/((\vec{\nabla} s)^2 + (m - s)^2)$, which gives equation 2.11.

$$\vec{v} = \frac{(m - s)\vec{\nabla} s}{(\vec{\nabla} s)^2 + (m - s)^2} \tag{2.11}$$

With this expression, the optical flow can be calculated in two steps: Instantaneous optical flow, and then regularize the deformation field[23, 100, 101].

Euler Transform is a rigid transformation in two-dimensional space, and is composed of a plane rotation and a two-dimensional translation. The rotation is applied first, followed by the translation, as expressed in equation 2.12.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix} \tag{2.12}$$

Here, $\theta$ is the rotation angle and $(T_x, T_y)$ are the components of the translation.

The Identity transform, shown in equation 2.13, is a `NULL` operation, as $M * I = M$.

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.13}$$

A requirement imposed on the transform classes by the registration framework is the computation of their Jacobians, as the metrics in general require the knowledge of the Jacobian in order to compute Metric derivatives[54]. The Jacobian is a matrix (2.14) whose elements are the partial derivatives of the output point with respect to the array of parameters that defines the transform[51].

$$J = \begin{bmatrix} \frac{\delta x_1}{\delta p_1} & \frac{\delta x_1}{\delta p_2} & \cdots & \frac{\delta x_1}{\delta p_m} \\ \frac{\delta x_2}{\delta p_1} & \frac{\delta x_2}{\delta p_2} & \cdots & \frac{\delta x_2}{\delta p_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta x_n}{\delta p_1} & \frac{\delta x_n}{\delta p_2} & \cdots & \frac{\delta x_n}{\delta p_m} \end{bmatrix} \tag{2.14}$$

Here, $\{p_i\}$ are the transform parameters and $\{x_i\}$ are the coordinates of the output point. The Jacobian can be noted as $J(X)$, where $X = \{x_i\}$.

The use of transform Jacobians enable efficient computation of metric derivatives. If Jacobians are not available, metric derivatives have to be computed using finite differences at a price of $2M$ evaluations of the metric value, where $M$ is the number of transform parameters.

Sometimes the term is used to refer to the determinant of a matrix representing the derivatives of output point coordinates with respect to input point coordinates.

Equation 2.15 illustrates the effect of a scaling transform on a 3D point.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} S_1 & 0 & 0 \\ 0 & S_1 & 0 \\ 0 & 0 & S_1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.15}$$

## 2.3 Software

Image registration has seen a surge of development during the last twenty years. Since then, multiple image registration software have been developed, many of which are still maintained. The most readily available software are presented here, giving some insight into the different focuses the developers keep in mind; ease of use versus complexity, optionality and availability.

### 2.3.1 DRAMMS

DRAMMS is an open source application. When compiled, DRAMMS will take two images as inputs, and output the deformation as well as the registered image[79]. DRAMMS supports NIFTI and ANALYSE 7.5 image formats, with data types `unsigned char`, `uint8`, `int8`, `short`, `int16`, `uint16`, `int32`, `float` or `float32`.

The input images are at first subject to attribute extraction and selection. Then they are weighted with mutual-saliency. Finally, a numerical optimization is applied, and the registered image is output[25].

### 2.3.2 ITK

ITK is a large open-source object-oriented software system for image processing, segmentation and registration[53]. Unlike most available software, ITK does not come with any graphical user interface or command line tools, but is first and foremost an extensive library for image registration. In place of command line tools or graphical user interface, well documented examples are available at a source code level[49].
A large number of modules are available, categorized as follows;

| Name | Content |
|---|---|
| Core | Central definitions and classes. |
| Third party | Various third-party libraries, such as I/O for spesific files. |
| Filtering | Image filters, laplacian, gaussian, etc. |
| IO | Reading, writing, transforming and geometry. |
| Bridge | Classes that connect with other libraries. |
| Numerics | Collection of numerical modules, optimization, statistics, etc. |
| Registration | Classes for registration of images or other data structures. |
| Segmentation | Classes for segmentation of images or other data structures. |
| Video | Classes for I/O and processing of static and real-time data. |

Table 2.4: Insight Toolkit modules

#### 2.3.2.1 ANTs

ANTs is a wrapping for ITK, which aims to simplify and unify the available tools that exists in ITK. For the end-user, this is done through the command line or scripting. ANTs uses R[81] and ANTsR[4] in order to calculate the registrations[7].

#### 2.3.2.2 Elastix

National Library of Medicine Insight Segmentation and Registration Toolkit (ITK)[52] is an open-source software system. Elastix is a command-line application largely based on ITK. The most popular transforms, metrics and optimizers from ITK are included, and Elastix aims for simplicity and ease of use. It is also possible to use Elastix as third party library[56, 57].

#### 2.3.2.3 GIMIAS

Graphical Interface for Medical Analysis and Simulation (GIMIAS) is a framework designed for fast prototyping in clinical evaluations. It provides a graphical interface, and a simple API, allowing the creation of plug-ins for different applications. GIMIAS is built on ITK, VTK, DCMTK and MITK, and supports image formats normally used in medical image analysis; Analyze, CGNS, DICOM, GDF and XML[48].

### 2.3.3 NiftyReg

NiftyReg is an open source application developed at University College London for use with Nifti or Analyze images. It supports both CPU and GPU implementations, and is able to perform rigid, affine and non-linear registration[76].

### 2.3.4 OpenCV

Open Computer Vision (OpenCV) is a large open source project in image processing. Although OpenCV initially was not created for image registration, there is a module available for image registration[45] which implements direct alignment. That is, it uses pixel values directly in order to register two or more images.

## 2.4 Insight Toolkit

Of the image registration software readily available, ITK provides the most methods, in addition to being highly customizable.

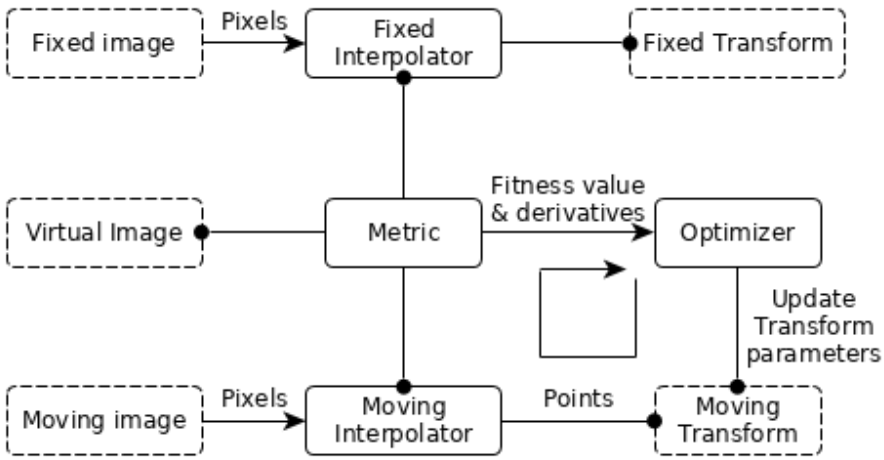



Figure 2.3: Basic components of the ITKv4 registration framework

Figure 2.3 illustrates various Insight Toolkit version 4 (ITKv4) registration components. Boxes with solid borders show the process objects. Boxes with dashed borders show the data objects[54].

The Metric class controls most parts of the registration process, as it handles the images, interpolators, and transforms in order to evaluate the intensity values in the images at each physical point of the virtual space. The metric cost function then evaluates the fitness value and derivatives, which are passed to the optimizer, which in turn updates the parameters of the moving transform based on the outputs of the cost function. This process will be repeated until the convergence criteria are met.

### 2.4.1 Definitions

Although ITK may be used for general image processing, the primary purpose of the toolkit is processing of medical imaging. Information associated with the physical spacing between pixels and the position of the image in space with respect to a common coordinate system is mandatory, and particularily so for different image modalities.

A pixel is considered as a rectangular region surrounding the pixel center, where the pixel center has a value assumed to exist as a dirac delta function. This region can be viewed as the Voronoi region of the image grid. Pixel spacing is measured between the pixel centers, and can be different along each dimension. The rectangle with corners in pixel centers is the Delaunay region, which is used for linear interpolation of image values. The image origin is associated with the coordinates of the first pixel in the image. This is illustrated in figure 2.4. The pixels have associated intensities, which for `float` numbers are between (0.0,1.0). An intensity of 0.5 will visualized be gray, 0.0 is black and 1.0 white.

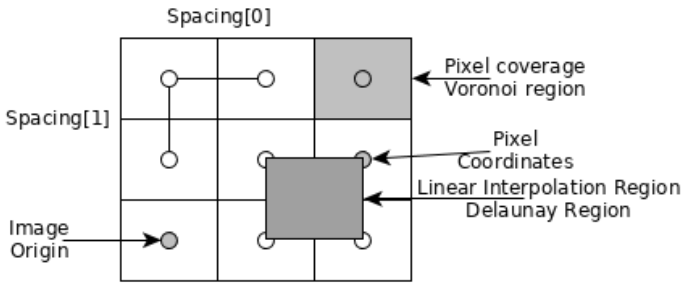



Figure 2.4: Geometrical definitions

Note that this is a non-grid position since the values are non-integers. This means that the gray value to be assigned to the output image pixel $I = (1, 2)$ must be computed by interpolation of the input image values.

For centered image operations, the image origin, spacing and size is used to calculate the center of the image, as shown in 2.16

$$\text{Center[dim]} = \text{Origin[dim]} + \text{Spacing[dim]} * \text{Size[dim]}/2.0 \tag{2.16}$$

Where, dim is the dimensions $X$, $Y$ and $Z$ for three dimensional, respectively. Simple translations in ITK includes offset in (x,y) coordinates, rotation and scaling. Offsets in (x,y) coordinates simply moves the image origin, or center, by a number of pixels. Rotation is applied around the image origin. In centered image operations, the image center is placed at the image origin before rotation is applied. First, offsets for (x,y) equal to $-\text{Center}[x, y]$ is applied, then the image rotation is applied, then offsets for (x,y) equal to $\text{Center}[x, y]$ is applied. This minimizes possible precision errors discussed in chapter 2.5, as well as simplifying the operations, as all operations apply to the image center. Scaling is applied to the spacing, and effectively increases both Voronoi- and Delaunay regions.

### 2.4.2 Filters

An image filter is image processing that changes the content of an image. ITK features a large range of filters, including smoothing filters, edge detection, casting, fast fourier transform (FFT), rescaling and geometrical transformations. The filters implemented in chapter 3 is presented here. The ITK book "Design and Functionality" has the following to say about `ChangeInformationFilter`; "This one is the scariest and most dangerous filter in the entire toolkit. You should not use this filter unless you are entirely certain that you know what you are doing. In fact if you decide to use this filter, you should write your code, then go for a long walk, get more coffee and ask yourself if you really needed to use this filter. If the answer is yes, then you should discuss this issue with someone you trust and get his/her opinion in writing. In general, if you need to use this filter, it means that you have a poor image provider that is putting your career at risk along with the life of any potential patient whose images you may end up processing"[54]. It follows some of the filters should not be used unless deemed absolutely necessary.

`itk::SubtractImageFilter` is a pixel-wise subtraction of two images. $\text{output}[x, y, z] = \text{image}_1[x, y, z] - \text{image}_2[x, y, z]$, where the output is the difference between $\text{image}_1$ and $\text{image}_2$. This filter is excellent for visualizing the effect of an image registration, or visualizing changes in a patient in a period of time, for example visualizing how much an abdominal growth has grown in two months.

The magnitude of the image gradient is extensively used in image analysis, mainly to help determine the contours and the separation of homogeneous regions. The `itk::GradientMagnitudeImageFilter` computes the magnitude of the image gradient at each pixel location using a simple finite differences approach.

`itk::GradientMagnitudeRecursiveGaussianImageFilter` computes the magnitude of the image gradient at each pixel location. The user selects a value for $\sigma$, which chooses the size of convolution with a Gaussian kernel. Then, the derivative of the Gaussian kernel is calculated. This filter will work with any image dimensions, as the components may be separated against the different dimensions.

`itk::RescaleIntensityImageFilter` linearly scales the pixel values in such a way that the minimum and maximum values of the input are mapped to minimum and maximum values provided by the user. This is a typical process for forcing the dynamic range of the image to fit within a particular scale and is common for image display. The linear transformation applied by this filter can be expressed as;

$$\text{outPixel} = (\text{inPixel} - \text{inMin}) * \frac{\text{outMax} - \text{outMin}}{\text{inMax} - \text{inMin}} + \text{outMin} \tag{2.17}$$

Where outMin and outMax are set by the user, and inMin and inMax are found in the input image.

`itk::MeanImageFilter` computes the value of each output pixel by finding the mean pixel intensity of the surrounding pixels. This algorithm is sensitive to outliers in the surrounding pixels. `itk::MeanImageFilter` is a precursor to `itk::MedianImageFilter`, and is commonly used as a simple approach for noise reduction.

`itk::MedianImageFilter` is commonly used as a more robust approach for noise reduction. This filter is particularly efficient against salt-and-pepper noise. In other words, it is robust in the presence of outliers in the surrounding outliers. `itk::MedianImageFilter` computes the value of each output pixel as the statistical median of the surrounding pixels. The size of the neighborhood is defined along every dimension by passing an object that defines the size with the corresponding values. The value on each dimension is used as the semi-size of a rectangular box. For example, in 2D a radius of 2 will result in a 5x5 neighborhood.

### 2.4.3 Interpolators

When image transforms yields translations where the pixel coordinates are not integer, but is placed somewhere in the spacing between pixel coordinates, interpolation is required in order to fit the pixels to the grid, as shown in figure 2.4.

`itk::NearestNeighborInterpolateImageFunction` simply uses the intensity of the nearest grid position. That is, it assumes that the image intensity is constant throughout the Voronoi regions. This interpolation scheme is cheap as it does not require any floating point computations.

`itk::LinearInterpolateImageFunction` assumes that intensity varies linearly between pixel coordinates. Unlike nearest neighbor interpolation, the interpolated intensity is spatially continuous. However, the intensity gradient will be discontinuous at grid positions.

ITK features a direct implementation of the linear interpolation in chapter 2.2.3. By default, `itk::LinearInterpolateImageFunction` assumes that all pixel values are inside the image bounds. The first pixel, is set to coordinates $(0, 0)$. The function then iterates through the consecutive pixel coordinates. This iteration process is shown in equation 2.18 and 2.19.

For a simple interpolation where one of the coordinates are integer, equation 2.18

$$\text{pixel} = \text{pixel}_n + (\text{pixel}_{n+1} - \text{pixel}_n) * \text{distance} \tag{2.18}$$

Where distance is the offset from the grid. When both coordinates are integer, equation 2.19 applies.

$$\text{value} = \text{value}_{0x0} + (\text{value}_{0x1} - \text{value}_{0x0}) * \text{distance} \tag{2.19}$$

Where $\text{value}_{0x0}$ and $\text{value}_{0x1}$ are found from equation 2.18 using coordinates $x$ and $y$, respectively.

`itk::BSplineInterpolateImageFunction` represents the image intensity using B-spline basis functions. When an input image is first connected to the interpolator, B-spline coefficients are computed using recursive filtering. Intensity at positions not intersecting with pixel coordinates is computed by multiplying the B-spline coefficients with shifted B-spline kernels within a small support region of the position.

### 2.4.4 Metrics

The metrics in chapter 2.2.4 are available as objects in `itk::ImageToImageMetricv4` in ITK, and measure how well the transformed moving image matches the fixed image by comparing the intensities. The objects implemented in chapter 3 are `itk::MeanSquaresImageToImageMetricv4` and `itk::MattesMutualInformationImageToImageMetricv4`, and function as shown in chapters 2.2.4 and 2.2.4, respectively.

### 2.4.5 Optimizers

Optimization algorithms in ITK are encapsulated as objects in `itk::ObjectToObjectOptimizer`. The basic input in registration is the metric classes, which provide a search space for the metric to be evaluated in. Parameters may at any time be retrieved using the `GetCurrentPosition()` function provided. `itk::RegularStepGradientDescentOptimizerv4` advances parameters in the direction of the gradient. A bipartition scheme computes the step size. The optimizer is also used for Versor[36] transforms parameters. The translational part of the transform parameters are updated in vector space. `itk::LBFGSOptimizerv4` is an adaptor to an optimizer in `vnl`, which is an implementation of "Limited memory Broyden, Fletcher, Goldfarb and Shannon minimization" (LBFGS)[69], shown in 2.20

$$K_i = K_{i-1} + \frac{\gamma\gamma^T}{\gamma^T\delta} - \frac{(K_{i-1}\delta)(K_{i-1}\delta)^T}{\delta^T K_{i-1}\delta} \tag{2.20}$$

Where $K_i$, or the next iteration step, is dependent on the previous iteration step.

### 2.4.6 Transforms

In ITK, `itk::Transform` objects encapsulate the mapping between points and vectors from an input space to an output space. Back transforms are provided if the transform is invertible.

`itk::AffineTransform` creates a transform specified by a $N \times N$ matrix and a $N \times 1$ vector where $N$ is the space dimension. The set of transform coefficients can be represented in a vector space of dimension $(N + 1) \times N$, which allows optimizers appropriate use on this search space. Equation 2.21 illustrates the effect of `itk::AffineTransform` in 2D space.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix} \cdot \begin{bmatrix} x - C_x \\ y - C_y \end{bmatrix} + \begin{bmatrix} T_x + C_x \\ T_y + C_y \end{bmatrix} \tag{2.21}$$

The coefficients $M$ are typically in a range of $\{-1, 1\}$, but are not restricted to this interval. The translation coefficients are related to the image size and pixel spacing, and can therefore be in the order of $[-100 : 100]$, or in extreme cases larger.

When using `itk::AffineTransform`, optimizer will start by focusing on removing large translations errors. This makes it computationally expensive for large deviations. Thus, it would be more optimal to use `itk::AffineTransform` for registration tasks that have had coarse registration, using other transformation methods first. This will decrease the size of the translation coefficients, and remove a relatively high computational cost.

`itk::BSplineDeformableTransform` is specifically designed for solving deformable registration problems. The transform generates a deformation field where a deformation vector is assigned to every point in space. The deformation vectors are computed by BSpline interpolation from the deformation values of points located in the BSpline grid. The BSpline deformable transform is not flexible, and is not able to account for large rotations, shearing or scaling differences. In order to compensate for this, an arbitrary transform known as "Bulk" transform is applied to points before they are mapped with the displacement field.

This transform does not provide functionality for anything other than mapping points, as the variations of a vector under a deformable transform depend on the location of the vector in space. A large number of parameters are involved with this transform, and is therefore well suited for use with `itk::LBFGSOptimizer` or `itk::LBFGSBOptimizer`[54, 67, 68, 85].

ITK implements a version of Thirion's "demons" algorithm, described in chapter 2.2.5. In this implementation, each image is viewed as a set of iso-intensity contours. The orientation and magnitude of the displacement is derived from the instantaneous optical flow equation 2.9:

$$\vec{D}(\vec{X}) \cdot \vec{\nabla} f(\vec{X}) = -(m(\vec{X}) - f(\vec{X})) \tag{2.22}$$

Where $f(\vec{X})$ is the fixed image, $m(\vec{X})$ is the moving image, and $\vec{D}(\vec{X})$ is the displacement or optical flow between the images.

For registration, the projection of the vector on the direction of the intensity gradient is used with the normalized equation 2.11 such that:

$$\vec{D}(\vec{X}) = -\frac{(m(\vec{X}) - f(\vec{X}))\vec{\nabla} f(\vec{X})}{\|\vec{\nabla} f\|^2 + (m(\vec{X}) - f(\vec{X}))^2/K} \tag{2.23}$$

Here, $K$ is a normalization factor that accounts for imbalance between intensities and gradients. $K$ is computed as the mean squared value of the pixel spacings. Starting with an initial deformation field $\vec{D}^0(\vec{X})$, the equation is updated such that the field at the N-th iteration is given by:

$$\vec{D}^N(\vec{X}) = \vec{D}^{N-1}(\vec{X}) - \frac{(m(\vec{X} + \vec{D}^{N-1}(\vec{X})) - f(\vec{X}))\vec{\nabla} f(\vec{X})}{\|\vec{\nabla} f\|^2 + (m(\vec{X} + \vec{D}^{N-1}(\vec{X})) - f(\vec{X}))^2} \tag{2.24}$$

`itk::Euler2DTransform` is a direct implementation of Euler Transform (2.2.5), and is described by equation 2.12. As ITK places image origins in the corners of images, `itk::Euler2DTransform` is rarely used directly, as this transform always uses the origin of the coordinate system as the center of rotation.

`itk::IdentityTransform` instantiates equation 2.13, and is mainly used for debugging purposes, as it is a `NULL` operation. It is particularly useful when a transform should not affect the output of the process, or when methods that require a transform is used.

`itk::CenteredRigid2DTransform` is an implementation of `itk::Euler2DTransform` (2.4.6), with an additional specified center of rotation as a rigid transformation in two-dimensional space. Thus, the center of rotation may be placed in the middle of the image, in place of the corner. The initial parameters must be passed to the transform; center of rotation and angle of rotation. The rotations are measured in radians, and are in the range of $[-\pi, \pi]$. The center of rotation and the output translations are measured in millimeters, and their actual values vary depending on the image modality of the moving and fixed images. The transformation function on an input $(x, y)$, mapped to $(x', y')$ is expressed in equation 2.25

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x - C_x \\ y - C_y \end{bmatrix} + \begin{bmatrix} T_x + C_x \\ T_y + C_y \end{bmatrix} \tag{2.25}$$

Here, $\theta$ is the rotation angle, $(C_x, C_y)$ are the rotation center coordinates, and $(T_x, T_y)$ are the translation components.

`itk::ScaleTransform` implements equation 2.15, where $S_1$ is replaced with $S_i$ in the following way for different image types:

$$\vec{P}' = T(\vec{P}) : \vec{P}_i' = \vec{P}_i \cdot \vec{S}_i$$
$$\vec{V}' = T(\vec{V}) : \vec{V}_i' = \vec{V}_i \cdot \vec{S}_i$$
$$\vec{C}' = T(\vec{C}) : \vec{C}_i' = \vec{C}_i / \vec{S}_i$$

Where $\vec{P}'$ is point, $\vec{V}'$ is vector, and $\vec{C}'$ is covariant vector.

As can be seen, points and vectors are transformed by multiplication of each one of their coordinates by the scale factor. Covariant vectors are transformed by dividing. Thus, if a covariant vector was orthogonal to a vector, this orthogonality will be preserved after the transformation.

Scale transforms introduce some issues to the optimizer, as the optimizer typically manage the parameter space as a vector space, where the basic operation is addition. Gradient descent optimizers have trouble updating step length, since the effect of an additive increment on a scale factor diminishes as the factor grows.

Then, scaling is better treated in the frame of a logarithmic space where additions result in regular multiplicative increments of the scale.

`itk::ScaleLogarithmicTransform` is a small deviation of `itk::ScaleTransform` (2.4.6), as the parameters factors in this class are passed as logarithms. Multiplicative variations are thus additive variations in the logarithm of the scaling factors.

`itk::Similarity2DTransform` is a rigid transformation in two-dimensional space, like Rigid Transform (2.4.6), with an isotropic scaling factor. The transformation applied to an input $(x, y)$, mapped to $(x', y')$ is expressed in equation 2.26. $\lambda$ is a scaling factor, $\theta$ the angle of rotation, $(C_x, C_y)$ the coordinates of the rotation center, and $(T_x, T_y)$ the components of the translation.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x - C_x \\ y - C_y \end{bmatrix} + \begin{bmatrix} T_x + C_x \\ T_y + C_y \end{bmatrix} \tag{2.26}$$

As such, `itk::Similarity2DTransform` is a transformation that combines `itk::ScaleTransform` (2.4.6) and `itk::Euler2DTransform` (2.4.6) with an offset $(C_x, C_y)$ for the rotation center.

`itk::TranslationTransform` is a simple, yet very useful transformation. It maps all points by appending a vector to them. Vectors and covariant vectors remain unchanged as they are not associated with a specific position in space. As it is simple, it has the advantage of being fast to compute and have parameters that are easily interpretable. The transform work by resolving the translational misalignment between the input images.

## 2.5 Validation

In image registration, it is only possible to supply a measure of accuracy by reference to phantom studies, simulations or other registration methods. If a method for quantifying registration accuracy does get developed, that method should be used for registration processes, as the information to be verified is the information one wants to retrieve from the registration process. As such, it is rather paradoxical. The validation of a registration embodies more than verification of the accuracy, and includes the following[62];

- Precision
- Accuracy
- Stability
- Reliability
- Resource requirements
- Complexity
- Assumption verification
- Clinical use

Precision is defined as the systematic error, obtainable when the registration algorithm is supplied with idealized input. Thus, a simple optimization algorithm with a resolution of two pixels is expected to perform with a precision within two pixels given ideal inputs, such as two identical images. Systematic errors in general

are defined as errors that are not determined by chance, but are introduced by an inaccuracy in the system[99]. Systematic errors are predictable and typically constant or proportional to the true value. If the cause of systematic error is identifiable, it can usually be eliminated. Typical causes for systematic errors are imperfect measurement- or observation methods.

Whereas precision is a system property, accuracy applies to the specific instance of registration. Accuracy is a more direct measure, referring to the true error in the images. In hyperspectral imaging, complete accuracy is achieved when the exact same part of an object is represented in the same pixel coordinates in all image bands, for all pixel coordinates. In clinical practice, a ground truth for accuracy is unavailable, and must be emulated by reference to another measure. Ground truth may be achieved using simulated images (synthetic), or by simulating images which emulate the clinical acquisition (software phantom). Pre-clinical and clinical evaluations using cadavers or patient images may be used in order to approximate a ground truth.

Stability, or robustness, refers to the basic requirement that small variations in the input should result in small variations in the output. If two input images are aligned in a slightly varied orientation, the algorithm should converge to approximately the same result. Reliability is a requirement that the algorithm should behave as expected, given reasonable inputs. That is, inputs which is expected to work well with the components of the registration process. Resource requirements apply to the effort involved in the registration process, and is closely linked to the algorithm complexity. These items impact computation time and resource constraints. Any assumptions and clinical use should be verified before the registration is put into practice; Does the registration work satisfactory? Does it outweigh available alternatives?

Ideally, all validation criteria should be satisfied. As increased precision and accuracy generally lead to larger resource requirements and/or higher algorithm complexity, this ideal is unrealistic. The weighting of each criterion is highly dependent on the application, and whether the registration process is working optimally is hence a matter of judgement.

For the image set with simulated errors, it is possible to completely verify or discard the different methods, as the ideal output is known. Precision and accuracy will be evaluated by calculation of the standard deviation and mean offset versus the ideal output. Stability and reliability will be evaluated by changing some parameters and multiple tests. Resource requirements and complexity will be evaluated from the run time. For the other image sets, partly validation by reference to the simulated image set is possible using assumption verification.

# 3 Materials and methods

This chapter start by giving an overview of the images analyzed in this thesis. Next, the methods available in ITK is presented with examples. Lastly, the implementation of the methods available through ITK is described.

## 3.1 Image files

Due to the nature of HSI and MSI, all images presented in this thesis have been normalized post registration. The normalization was done using ImageMagick[47], using a filter that scales the intensity across the image, setting the maximum pixel intensity to 1.0.

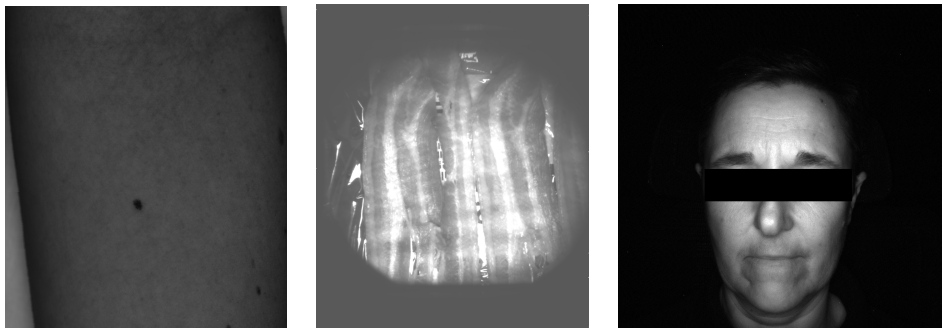
### 3.1.1 Hyperspectral images

Three sets of hyperspectral images are to be registered. The first set has the file format .img, and is BIL interleaved. The images origin from a HySpex VINIR-1600 camera[43], developed and manufactured by Norsk Elektro Optikk, which is a line-scanning camera using a push-broom technique. The camera features autofocusing[90], and each image file consists of 1600 pixels (samples), 2678 lines and 160 wavelengths (bands)[11]. This set is assumed to be perfectly aligned in the spectral regime in the first 70 bands, which allows this image set to be used as reference for validation. One of the image bands from this image set can be seen in figure 3.1a.

The second set has the file format .mat, and is BSQ interleaved. The images origin from a Zyla 5.5 sCMOS camera, produced by Andor technology from Belfast UK. The object in these images has been moved around at random. Thus, this image set is unsuited for validation, as perfect registration results are of unknown characteristics. Each image file consists of 6 to 11 image bands, with 2162x2560 pixels, where the last image band is a reference background image at wavelength $\lambda = 666$nm[59]. The image files consists of two matrices, where the first contain header information; number of images, image dimensions, wavelengths, use of compression and data type. The second matrix contain the images, and use `MATLAB` formatting; $x$ in `MATLAB` is $y$ in `C++`[66]. Example image can be seen in figure 3.1b.

The third set is also stored in the .img format with BIL interleave. Each image band consists of 1000 pixels (samples), 1000 lines and 91 wavelengths (bands).

These images origin from an Acousto-optical tunable filter (AOTF) camera system. Without precise knowledge of distance between camera and object, in addition to no knowledge about what correct registration in this set should be, the registration results from these images will not be verifiable. In conjunction with ethical standards and accordance, these images have been anonymized, inserting a black band across the eyes. The center image band from this set can be seen in figure 3.1c.



(a) Pushbroom
2678x1600 pixels

(b) Zyla
2162x2560 pixels

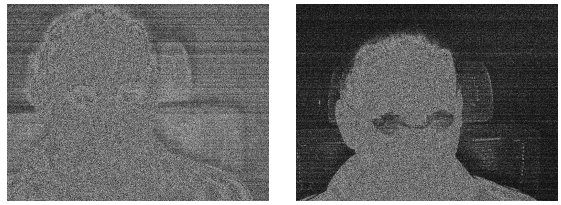
(c) AOTF
1000x1000 pixels

Figure 3.1: Hyperspectral images

### 3.1.2 Multispectral images

The multispectral images are stored in separate .raw containers. The images origin from a five camera system, described in [39], which is currently under peer review. The cameras have bandpass filters with center wavelengths of 360nm, 475nm, 560nm, 580nm, and 650nm. These images are `unsigned short`, with $1024x768$ pixels. Similar to the third hyperspectral image set, the object is between 30 and 60 cm away from the camera, and the registration results are similarly not verifiable. The images supplied are visible in figure 3.2.

(a) Image 1 (b) Image 2 (c) Image 3



(d) Image 4 (e) Image 5

Figure 3.2: Multispectral images

## 3.2 Choice of registration software

None of the available libraries directly supports the image formats dealt with in this task. That is, no native support for .img or .mat, and little to no support for .raw, in existing image registration software. This means that creating readers and handlers for these formats is a given, regardless of choice of library. Supported file formats may then be disregarded as a justification of use of library.

The implementation of alignment must be considered. Given from chapters 1 and 2.2.1, segmentation-based registration is the optimal solution for the hyperspectral images, wheres the multispectral images may be segmentation-based or deformable image registration problems, depending on how large the angles between the cameras and object are. Of the readily available registration software, ITK offers all of the functionability necessary, in addition to having better documentation than any other available image registration software.

## 3.3 Example methods

ITK may be compiled with example code. There are twenty image registration examples, and seventeen deformable image registration examples. The transform, optimizer and metric used in the different examples are summed up in table 3.1 and 3.2

| Name[1] | Transform | Optimizer | Metric |
|---|---|---|---|
| IR1.cxx | Translation | Gradient Descent | Mean Squares |
| IR2.cxx | Translation | Gradient Descent | Mutual Information |
| IR3.cxx | Translation | Gradient Descent | Mean Squares |
| IR4.cxx | Translation | Gradient Descent | Mutual Information |
| IR5.cxx | Rigid 2D | Gradient Descent | Mean Squares |
| IR6.cxx | Rigid 2D | Gradient Descent | Mean Squares |
| IR7.cxx | Similarity 2D | Gradient Descent | Mean Squares |
| IR8.cxx | Rigid 3D | Gradient Descent | Mean Squares |
| IR9.cxx | Affine | Gradient Descent | Mean Squares |
| IR10.cxx | Translation | Amoeba | Match Cardinality |
| IR11.cxx | Translation | One Plus One | Mutual Information |
| IR12.cxx | Rigid 2D[2] | Gradient Descent | Mean Squares |
| IR13.cxx | Rigid 2D | Gradient Descent | Mutual Information |
| IR14.cxx | Rigid 2D | One Plus One | NMIH[3] |
| IR15.cxx | Translation | One Plus One | NMIH[3] |
| IR16.cxx | Translation | Amoeba | NMIH[3] |
| IR17.cxx[4] | Translation | Amoeba | NMIH[3] |
| IR18.cxx | Translation | Gradient Descent | Gradient Difference |
| IR19.cxx | Affine | Amoeba | Match Cardinality |
| IR20.cxx | Affine[5] | Gradient Descent | Mean Squares |
| DR4.cxx | BSpline | LBFGS | Correlation |
| DR6.cxx | BSpline | LBFGS | Mean Squares |
| DR7.cxx | BSpline[5] | LBFGSB | Mean Squares |
| DR8.cxx | BSpline[5] | LBFGSB | Mutual Information |
| DR12.cxx | BSpline | LBFGSB | Mutual Information |
| DR13.cxx | BSpline | Gradient Descent | Mutual Information |
| DR14.cxx | BSpline[5] | Gradient Descent | Mutual Information |
| DR15.cxx | Multiple[56] | Gradient Descent | Mutual Information |

1. ImageRegistration shortened to IR.
2. The same as IR6.cxx, but with a mask applied before registration.
3. Normalized Mutual Information Histogram.
4. Small changes to parameters from IR16.cxx.
5. In three dimensional space.
6. This example makes use of Affine, Versor Rigid 3D and BSpline.

Table 3.1: Example image registrations with default ITKv4 registration methods

In table 3.1, all the examples make use of the default registration method shipped with ITKv4. Here, the transform, optimizer and metric of choice are listed, as it is these parameters that are the main source of difference between the examples. In table 3.2, histogram matching is used in all of the examples. The registration methods are listed, as they are the primary source of interest.

| Name[1] | Registration Method |
|---|---|
| DR1.cxx | FEM |
| DR2.cxx | Demons |
| DR3.cxx | Symmetric Forces Demons |
| DR5.cxx | Level Set Motion |
| DR11.cxx | FEM[2] |
| DR16.cxx | Multi Resolution, PDE and Symmetric Forces Demons[2] |

1. DeformableRegistration shortened to DR.
2. In three dimensional space.

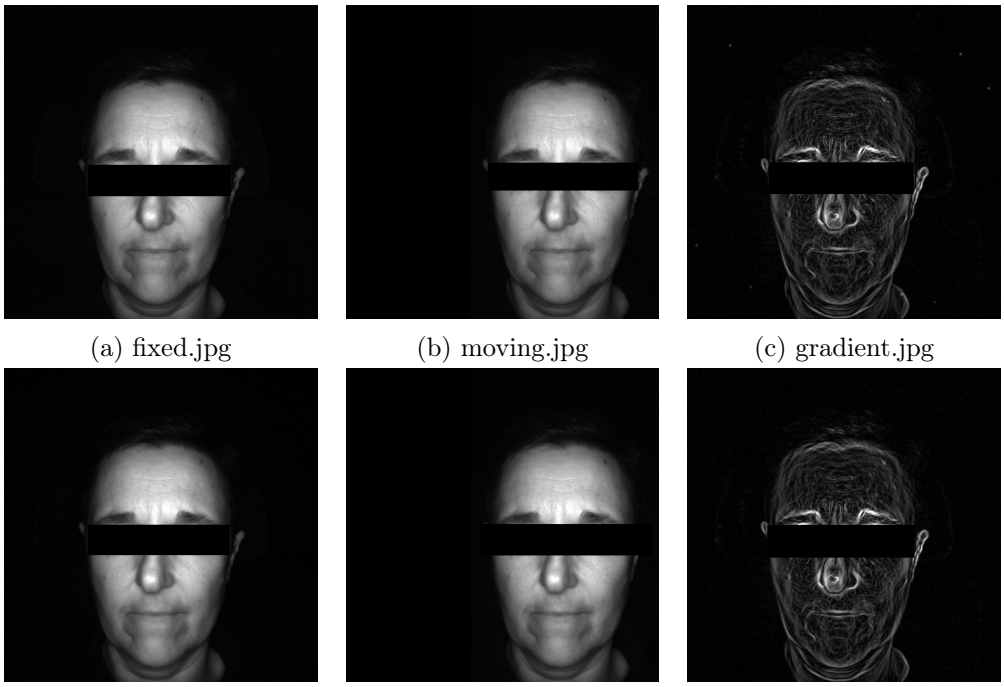
Table 3.2: Example image registrations with Histogram Matching

Where `DeformableRegistration9.cxx` and `DeformableRegistration10.cxx` have been omitted due to third party dependencies. `DeformableRegistration17.cxx` has been omitted, as it is essentially a duplicate of `DeformableRegistration16.cxx`.

`ImageRegistration1.cxx` and `ImageRegistration3.cxx` are essentially the same, where `ImageRegistration1.cxx` has a minimalistic setup, and should, according to the ITK book[54] be considered as a "Hello World" example.

## 3.4 Preliminary testing

For the preliminary tests, the example code described in chapter 3.3 were used. This proved to be an effective way to quickly prototype the different registration methods provided by ITK, but required some setup. The HSI was read into a `float` array using the code in appendix A.1. Now, a `float` array held the entire hyperspectral image. In this case, 90 bands. The center band was chosen and written to a file "fixed.tif", using the functions in lines 91 and 102 in A.3. The second image, "moving.tif", for the purpose of testing, was moved 200 pixels to the right. This was done with a modification of the same functions. An additional mask was created, as some of the examples required an additional mask input. This was chosen to be a gradient filter, "gradient.tif", as described in lines 94 and 277 in A.4. An additional test set was created, where a median filter, as described in lines 91 and 280 in A.4 was applied to the center band first. The output test images are aptly named "median_fixed.tif", "median_moving.tif" and "median_gradient.tif". The images are pictured in figure 3.3



(a) fixed.jpg (b) moving.jpg (c) gradient.jpg

(d) median_fixed.jpg (e) median_moving.jpg (f) median_gradient.jpg

Figure 3.3: Extracted images for preliminary testing

The output images from the registration process was then opened with the fixed input image in ImageJ, and the difference was calculated. The results from the area of interest are presented in table 4.4. The area of interest is defined as the

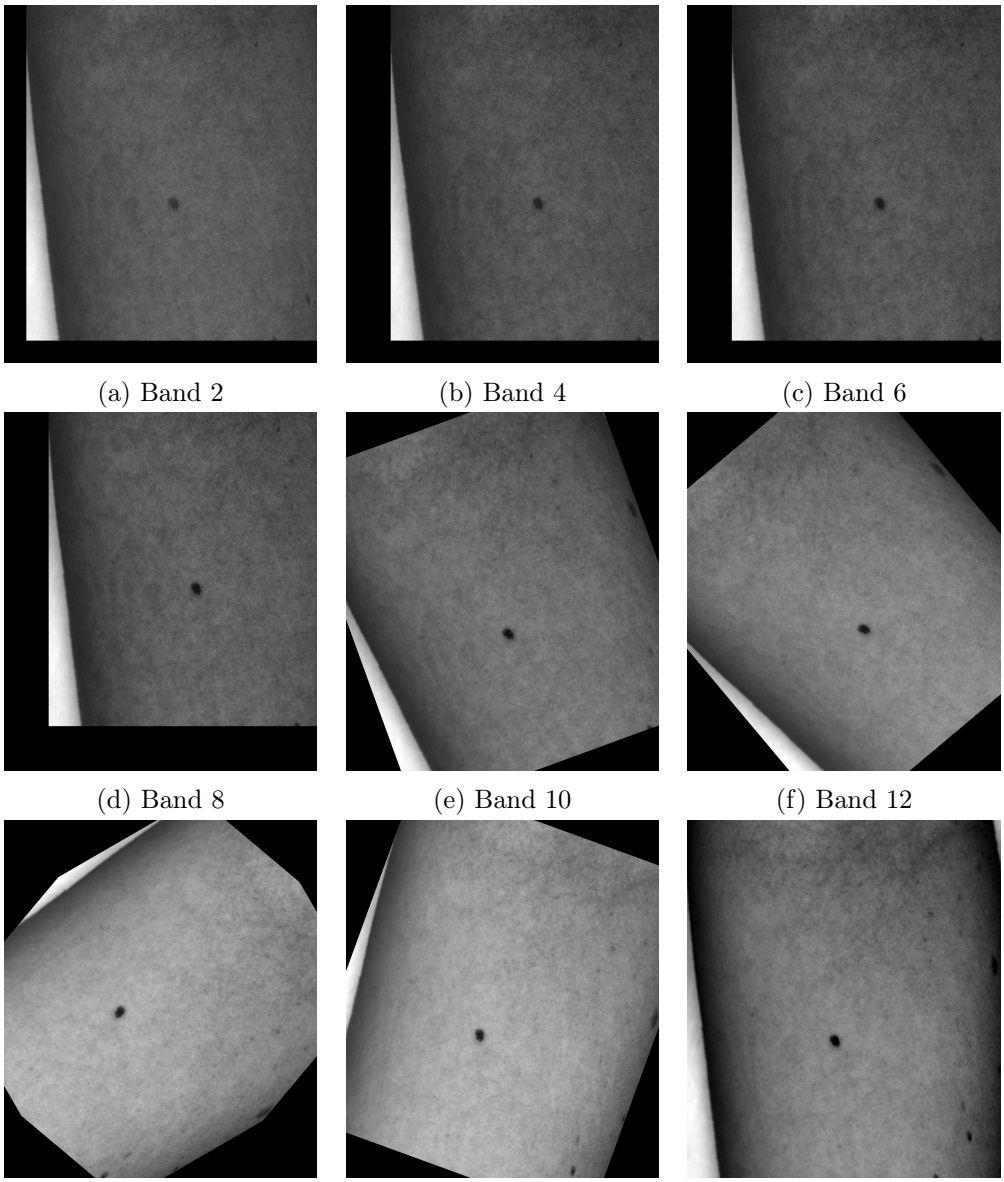
area between [0 + 100,image-width−100], as the moving image was moved 100 pixels to the right. The area between 0 and 100 pixels will have no equivalent point in the moving image, and the area between image-width−100 and image-width will be out of bounds for the moving image. No parameters were changed. These results give a starting point for further development. Note that some of these examples are legacy mode, described in 2.3.2, and will therefore not be considered. The hyperspectral images should in theory not be deformable problems, but the multispectral images are both rigid and deformable problems. Thus, rigid, similarity and affine transforms will be further investigated. BSpline transform will be investigated, as it was the deformable registration transform with the best results from the preliminary tests. Additionally, Demons transform will be tested further. The Demons transform has received a lot of attention in the last two decades.

As described in 2.4.6, translation transform does not include parameters for scaling or angles. This makes it fast to compute, but due to the lack of these parameters will never be an adequate registration method on its own. It should, however, be optimal as an optional coarse registration before the other registration methods are applied. In theory this should for certain image sets optimize both run time and results, as the larger translations in x- and y-coordinates are handled by this method.

For the different transforms, it is introduced errors to certain image bands in a known test set. "Lise_arm_before_occlusion_mnf_inversetransformed.img" is reduced to 40 image bands, and has 1601x1401 pixels. The following bands now has the following offsets, where the remaining bands are perfectly aligned:

| Band | Offset X | Offset Y | Angle |
|---|---|---|---|
| 2 | -100 | 100 | 0 |
| 4 | -200 | 100 | 0 |
| 6 | -200 | 100 | 0 |
| 8 | -200 | 200 | 0 |
| 10 | 0 | 0 | 20 |
| 12 | 0 | 0 | 40 |
| 14 | 0 | 0 | -60 |
| 16 | 0 | 0 | -20 |

Table 3.3: Introduced testset with known errors



(a) Band 2 (b) Band 4 (c) Band 6

(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16 (i) Band 20 (fixed)

Figure 3.4: Introduced offsets from table 3.3

## 3.5 Implementation

The `main` function takes in a number of arguments `*argv[]`. These arguments are presumed to be files, and using the function `strstr` from the `<string>` library, it recognizes the file format and appends the input arguments to the corresponding function. The entirety of the `main` function can be found in A.5, listing 1. All other parameters are read from a configuration file.

For Matlab hyperspectral files; `hyperspec_mat` (A.3, line 81), ENVI standard hyperspectral files; `hyperspec_img` (A.3, line 76), and raw multispectral files; `multispec_raw` (A.2, line 12). It is assumed that raw files are multispectral, and ENVI standard and Matlab files are hyperspectral, as this is a prerequisite. For arguments not affiliated with these functions, a `cerr` message is returned, and the program exits. Common for these corresponding functions, is that they make use of `conf_err_t params_read()` function, declared in A.3; lines 13, 54, 65 and 68, which returns values from a configuration file into a `struct`, containing all parameters and settings. In A.5, listing 2, these configuration parameters are read into `params`.

When the registration process is done, the results are written to files of the same format as the input. The flow of the program is illustrated in figure 3.5, where the use of filtering is optional, and metric, optimizer and interpolator depends on the chosen transform method through the configuration file.

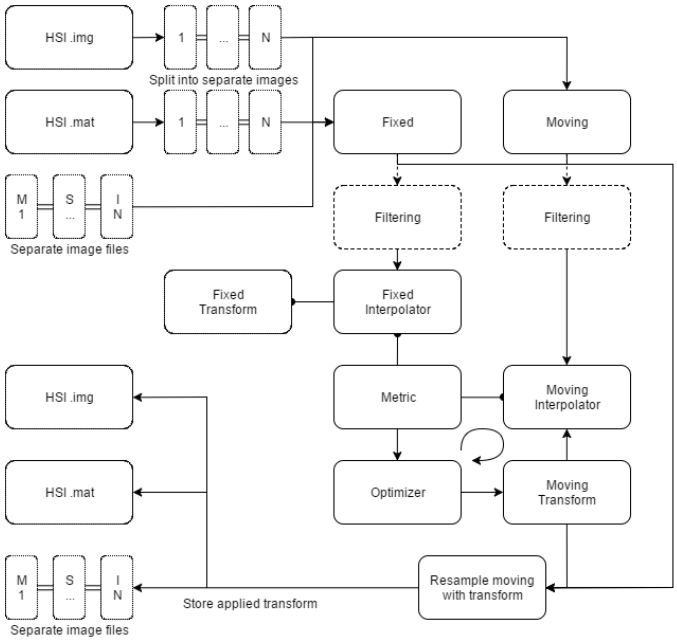



Figure 3.5: Components of the developed registration software

Making use of the functions available through `readimage.h` and `matio.h`, .img and .mat containers are read into a `float` array, respectively. In A.5, listing 3 this is done for .img, and in A.5, listing 4 this is done for .mat.

This array contain the full hyperspectral image, and will need to be split into separate images. This is done with functions `readITK` and `readMat`, in A.3 lines 91 and 118. These functions write a band `i` from the `float` array into an ITK image pointer of the same dimensions. In order to do any registration, two input images are needed, a fixed image and a moving image. For ease of use, the center band has been chosen as the fixed image for the registrations; If the fixed and moving images both were to be variable, a flaw in one registration would manifest itself in the next registration. Next, pre-defined settings contained in the configuration file decides whether a median filter and/or gradient filter is applied to the images, and the images are sent to the transform. The returned transform is applied to the unfiltered moving image and saved to a `float` array, using `writeITK` and `writeMat` in A.3 lines 102 and 130. If difference output is enabled in the configuration file, an additional `float` array is prepared, containing the difference between the transformed moving image and the fixed image. When all the bands are processed and registered, the `float` array, now containing all the registered bands, are saved to file using the functions available through `readimage.h` (A.5, listing 5) and `matio.h` (A.5, listing 6).

Unlike the hyperspectral images, the multispectral images does not contain any metadata information in a header, as described in chapter 2.1.5. The raw files are simply binary data files that can be read as long as the metadata is known. For the raw images, it is known that the images are 1024 pixels wide, and 768 pixels high. It is also known that the data type is `unsigned short`. Then, it is possible to read the image into an `unsigned short` array making use of the `iostream` library. This is done in the function `readRaw`, A.2, line 34, which reads a raw file into a temporary array, which is read into an ITK image pointer. This function is shown in full in A.5, listing 14.

The ITK image pointer is returned to the main function, but as this is of data type `unsigned short`, and the registration methods only handle `float` data types, this will need to be cast to a `float` image. This is done using `CastFloatImage`, declared in A.4 line 285, and shown in A.5, listing 10.

As there is only one image per file in this case, the first input to the program is chosen as the fixed image, and all other images are chosen as moving images. The output images are written to file with the `fstream` library, and the output names are set as the configuration file depicts, with an additional number corresponding to the order of the input. This is done using the function in A.2, line 47.

The general pipeline structure of ITK is visualized in figure 2.3, and depends heavily on `typedefs` and templates. The implementation is somewhat different in the developed program, with the optional filtering, as visualized in figure 3.5. Any chosen filtering only impacts the transform parameters, not the output transformed image. This is done by applying the unfiltered images when resampling the moving image with the moving transform. If the filtered images where to be used as base for the output, information loss would be introduced to the registration results.

All of the library modules are loaded, A.4 lines 12 to 59, and all transforms, registrations, optimizers and filters are `typedef`'d in lines 87 to 229. These `typedefs` depend on the image dimensions and image data types defined in lines 82 to 85. These definitions depend on the data types of the hyperspectral and multispectral images to be registered.

The registration process needs to be initialized with a `TransformationType::Pointer`, a `MetricType::Pointer`, and a `OptimizerType::Pointer`. The `TransformationType::Pointer` needs to be initialized with the fixed and moving images, and the parameters has to be set for the `OptimizerType::Pointer`. When this is done, the `RegistrationType::Pointer` can be initialized with the images, the optimizer, metric and other optional parameters such as smoothing and shrinking. A sample initialisation is presented in A.5, listing 7.

The registration process needs some further parameters, which is passed from the `params struct`. A sample preparation is presented in A.5, listing 8. Now the registration is ready, and can be passed to the `CommandIterationUpdate::Execute` class, appendix A.5, listing 9 and 11 lines 63 through 79, which handles the iteration process of the optimizer.

The different `RegistrationType` templates apply the current transformation to the `InitialTransform` when `registration->Update()` is run. This is done in a `try catch` block, in order to catch exceptions if anything should go wrong (A.5, listing 11). This process will run until `registration->GetOptimizer()->GetStopConditionDescription()` reaches the set parameters, defined in the configuration file.

When the stop condition has been reached, the `TransformationType::Pointer` set for the `RegistraionType::Pointer` will contain the necessary transformation in order to tranform the moving image to overlap the fixed image. The `TransformationType::Pointer` also contain all the translation parameters, which can be extracted using the available `->Get()` functions, as shown in A.5, listing 12.

When the transformation is found, it still needs to be applied to the moving image. This may be done with a `ResampleFilterType::Pointer`. This filter requires, as a minimum, the moving image and the transformation as inputs, but it is convenient to also pass the fixed image. The origin, spacing, size and direction of the output image is retrieved from the fixed image, and the transformation is applied to the moving image. Now, the output registered image may be retrieved using `resample->GetOutput()`, where `resample` is a `ResampleFilterType::Pointer`. A sample resample filter is available in A.5, listing 13.

The sample code in listings 7 through 13 will, given corresponding typedefs in 2.2 be directly applicable for `itk::CenteredRigid2DTransform`. Thus, this sample code is used as a foundation for the different transforms, except for `itk::BSplineTransform`, which require additional tweaking as this is a deformable registration method.

## 3.6 Transforms

### 3.6.1 Affine

In order to make use of `itk::AffineTransform`, an optimizer and metric must be chosen. From tables 3.1 and 4.4, the affine example with the best result makes use of Descent Gradient optimizer, and Mean Square metric. `itk::AffineTransform` is declared in A.4 lines 154 through 161.

In addition to the sample code in listings 7 through 13, `itk::AffineTransform` also supports scaling. Thus, the code in A.5, listing 12 needs to be replaced with the code in A.5, listing 15

### 3.6.2 BSpline

`itk::BSplineTransform` is a deformable registration method that makes use of an optimizer and metric, which must be chosen. From tables 3.1 and 4.4, the BSpline example with the best result makes use of the LBFGS optimizer, and Mean Square metric. `itk::BSplineTransform` is declared in A.4 lines 104 and 169 through 182.

Although some of the sample code in listings 7 through 13 may be used, `itk::BSplineTransform` makes use of the LBFGS optimizer, requiring the extra code in A.5, listing 16, and the parameters is aquired using the code in A.5, listing 17

### 3.6.3 Demons

`itk::DemonsRegistrationFilter` makes use of a filter which matches areas of the input fixed and moving images. With the matching filter, the demons registration calculates a displacement field which is passed to another filter, which warps the moving image onto the fixed image. This warper makes use of an interpolator, which removes the necessity for additional resampling of the output image. These filters are declared in A.4 lines 189 through 199, and the source code is available in 21.

### 3.6.4 Rigid

`itk::CenteredRigid2DTransform` also require an optimizer and metric to be chosen. From tables 3.1 and 4.4, the rigid example with the best result makes use of Descent Gradient optimizer, and Mean Square metric. `itk::CenteredRigid2DTransform` is declared in A.4 lines 130 through 136.

`itk::CenteredRigid2DTransform` directly makes use of the code in A.5, listings 7 through 13, where `TransformType` is defined as `itk::CenteredRigid2DTransform`.

### 3.6.5 Similarity

Similarly for `itk::CenteredSimilarity2DTransform`, an optimizer and metric must be chosen. From tables 3.1 and 4.4, the similarity example with the best result makes use of Descent Gradient optimizer, and Mean Square metric.`itk::CenteredSimilarity2DTransform` is declared in A.4 lines 142 through 148.

In addition to defining the `TransformType` as `itk::CenteredSimilarity2DTransform` in the code in A.5, listings 7 through 13, the additional parameters in the similarity transform must be accounted for. This is done using function `finalSimilarityParameters`, declared in A.4 line 293, available in A.5, listing 18.

### 3.6.6 Translation

From tables 3.1 and 4.4, the translation examples with the best results make use of Descent Gradient optimizer, Mean Square metric and Mattes Mutual Information. As this transform mainly is used for coarse transformations due to lack of angular translations, this transform needs an additional class for tracking the iterations. Although this is not strictly necessary, it is defined as good code ethics[54], and removes the possibility for mix ups in the iteration tracker between transforms.

The additional metric and transform is declared in A.4 lines 114 and 117. Choosing the metric is done in the configuration file, and is implemented using the code in A.5, listing 19. As this transform is intended as a coarse pre-registration step, the optimizer is passed a relaxation factor, available in A.5, listing 20. This massively decreases run time, but limits accuracy of the transformation.

The additional iteration class is initiated as shown in A.5, listing 22.

# 4 Results and discussion

An efficient and reliable registration method is necessary for information extraction in hyperspectral and multispectral imaging. The results of the proposed methods, with regards to the validation methods described in chapter 2.5, will here be presented. The image results have been brightened, as described in 3.1. Additionally, a grid has been added for visual inspection. The thickness of the grid is two pixels, and the grid size is set to 100x100 pixels.

Before the main results are presented, some equations regarding the rounding of the results will be evaluated. The result for each method is presented within its own chapter, with comparisons and precision evaluation in chapter 4.3.

Precision of distance in equation 2.18 is crucial for the accuracy of $\text{pixel}_n$. From the definitions used in ITK, chapter 2.4.1, the size of spacing between pixels are vital, as this will impact the interpolation. In the images to be registered, the spacing size is of unit spacing, or [1,1]. This simplifies further calculations.

The maximum impact is found, with pixel values of `float` type, when the pixel values for $n$ and $n+1$ are 0 and 1, respectively. In the transformations, angles impact the accuracy more than translations in $x$ and $y$ coordinates. Assuming an image of dimensions 1600x1600 pixels, and a transformation origin in (0,0), the translation in the image extremities can be calculated using trigonometric funtions (equation 4.1) and pythagoras (equation 4.2)

$$\tan\alpha = \frac{\text{opposite}}{\text{adjacent}} \tag{4.1}$$

$$h^2 = c_1^2 + c_2^2 \tag{4.2}$$

Combining equations 2.18 and 4.2, using $\text{pixel}_n = 0$, $\text{pixel}_{n+1} = 1$ yields;

$$\text{error} = \sqrt{c_1^2 + c_2^2} \tag{4.3}$$

Inserting for $c_1 = c_2 = c$ in equation 4.3 is presented in table 4.1

| c value | distance offset | % error in pixel value |
|---|---|---|
| 0.5000 | 0.707 | 70.7% |
| 0.0500 | 0.071 | 7.1% |
| 0.0050 | 0.007 | 0.7% |
| 0.0005 | 0.001 | 0.1% |

Table 4.1: Maximum accuracy error from pixel translation

Combining equations 2.18, 4.1 and 4.2, using $\text{pixel}_n = 0$, $\text{pixel}_{n+1} = 1$ and image dimensions XxY pixels yields;

$$\text{error} = \sqrt{(X * \tan x)^2 + (Y * \tan x)^2} \tag{4.4}$$

Insertions for $x$ in equation 4.4, using image dimensions 1600x1600, is presented in table 4.2.

| x value | distance offset | % error in pixel value |
|---|---|---|
| 0.05000 | 1.975 | 197.5% |
| 0.00500 | 0.198 | 20.0% |
| 0.00050 | 0.020 | 2.0% |
| 0.00005 | 0.002 | 0.2% |

Table 4.2: Maximum accuracy error from angle translation in image origin

In centered image rotations, this accuracy error is decreased, as the variables for $X$ and $Y$ in equation 4.4 is half the dimensions of the image. Insertions for $x$ in equation 4.4, using image dimensions 1600x1600 in a centered image operation is presented in table 4.3.

| x value | distance offset | % error in pixel value |
|---|---|---|
| 0.05000 | 0.987 | 98.7% |
| 0.00500 | 0.099 | 9.9% |
| 0.00050 | 0.010 | 1.0% |
| 0.00005 | 0.001 | 0.1% |

Table 4.3: Maximum accuracy error from angle translation in centered image operation

As can be seen in tables 4.1 and 4.3, using three decimal points for pixel coordinates and four decimal points for angle translations gives a maximum error rate of 1.7% for the pixel values in the image extremities, and 0.7% in the center of rotation.

## 4.1 Preliminary tests

The example code is written with casting and resampling filters, which makes some image conversion necessary before the code can be run. This is described in chapter 3.4. When the images are of `int` type, the results in table 4.4 are retrieved from running the example code with the images in figure 3.3, and should therefore be easily reproducable.

From table 4.1, presenting the offsets using two decimals yields a maximum additional error rate of 7.1%. For the preliminary testing, this is an acceptable possible error rate.

| Name[1] | Mean[2] | StdDev[3] | Max[4] | Name[1] | Mean[2] | StdDev[3] | Max[4] |
|---|---|---|---|---|---|---|---|
| IR1.cxx | 0.49 | 0.50 | 1 | IR15.cxx | 0.49 | 0.50 | 1 |
| IR3.cxx | 0.49 | 0.50 | 1 | IR16.cxx | 0.49 | 0.50 | 9 |
| IR4.cxx | 0.71 | 0.75 | 67 | IR17.cxx | 40.58 | 41.36 | 252 |
| IR5.cxx | 36.40 | 40.07 | 255 | IR18.cxx | 26.96 | 34.66 | 255 |
| IR6.cxx | 0.66 | 0.69 | 87 | IR19.cxx | 25.76 | 33.83 | 255 |
| IR7.cxx | 0.79 | 0.81 | 113 | DR2.cxx | 26.55 | 34.69 | 255 |
| IR9.cxx | 0.46 | 0.50 | 6 | DR3.cxx | 24.85 | 34.24 | 255 |
| IR10.cxx | 26.73 | 34.68 | 255 | DR4.cxx | 2.62 | 2.51 | 233 |
| IR11.cxx | 0.57 | 0.59 | 42 | DR5.cxx | 22.72 | 30.25 | 255 |
| IR12.cxx | 0.50 | 0.53 | 43 | DR6.cxx | 0.66 | 0.88 | 99 |
| IR13.cxx | 30.17 | 37.89 | 232 | DR12.cxx | 1.27 | 1.47 | 230 |
| IR14.cxx | 0.50 | 0.50 | 4 | DR13.cxx | 3.91 | 4.18 | 228 |

1. Image Registration = IR, Deformable Registration = DR
2. Mean deviation
3. Standard deviation
4. Maximum pixel value, 0-255

Table 4.4: Results from preliminary tests

Using the image analysis functions in ImageJ, the difference between the input fixed and the output transform was calculated. If the input moving image were to be completely registered, the difference image should be completely black. The histogram function in ImageJ was then used to extract information about mean pixel value, standard deviation of pixel values, and maximum pixel value.

## 4.2 Transformation methods

From table 4.4, using the overview of the example files in tables 3.1 and 3.2, the optimal transform is the translation transform. This is expected, as this transform handles rigid transforms in $(x, y)$ coordinates, which is the introduced errors to be corrected. Close behind are Affine transform, Rigid 2D transform and Similarity transform. The best deformable registration method is BSpline. Similar for all these methods is that they use the Gradient Descent optimizer and Mean Squares metric, with the exception of BSpline that makes use of the LBFGS optimizer, and translation transform using Mutual Information metric. The second best deformable registration method is Demons, which makes use of Histogram Matching. The program will therefore use these methods as base for further evaluation.

| Parameter | Value |
|---|---|
| Median[1] | 1 |
| Gradient[1] | 0 |
| Angle[2] | 0.0 |
| Scale[2] | 1.0 |
| Learning rate[3] | 0.1 |
| Step length[4] | 0.0001 |
| Iterations[4] | 10000 |
| Translation Scale[3] | 0.001 |
| numberOfLevels[3] | 1 |

1. Filtering of images, where 1 is on and 0 is off
2. Initial parameters applied to moving image
3. Speed and rate of the optimizer
4. Stop conditions; maximum iterations and step length

Table 4.5: Strict parameters

For qualitative results, the parameters in table 4.5 are set for all methods. Learning rate and step length are the stop conditions for the optimizer, and will be varied throughout the testing. Median filtering will be used with a small radius for noise removal. Initial angle and scale are the initial orientation of the initial transform, and is set equal to the input filtered image to be registered. Similarly, the number of level controls scaling through the optimizer process, and is set equal to the initial transform.

### 4.2.1 Translation

The results of the translation transform, using parameters as described in 4.5, is presented in 4.6, and visualized in 4.1.

The relaxation factor allows the metric to accept results which are considered "close enough". These can be seen in figures 4.1a, 4.1b, 4.1c and 4.1d. As expected, the introduced angular errors are not caught by translation transform. This leads to large errors in the resulting transforms, which can be seen in figures 4.1e, 4.1f, 4.1g and 4.1h.

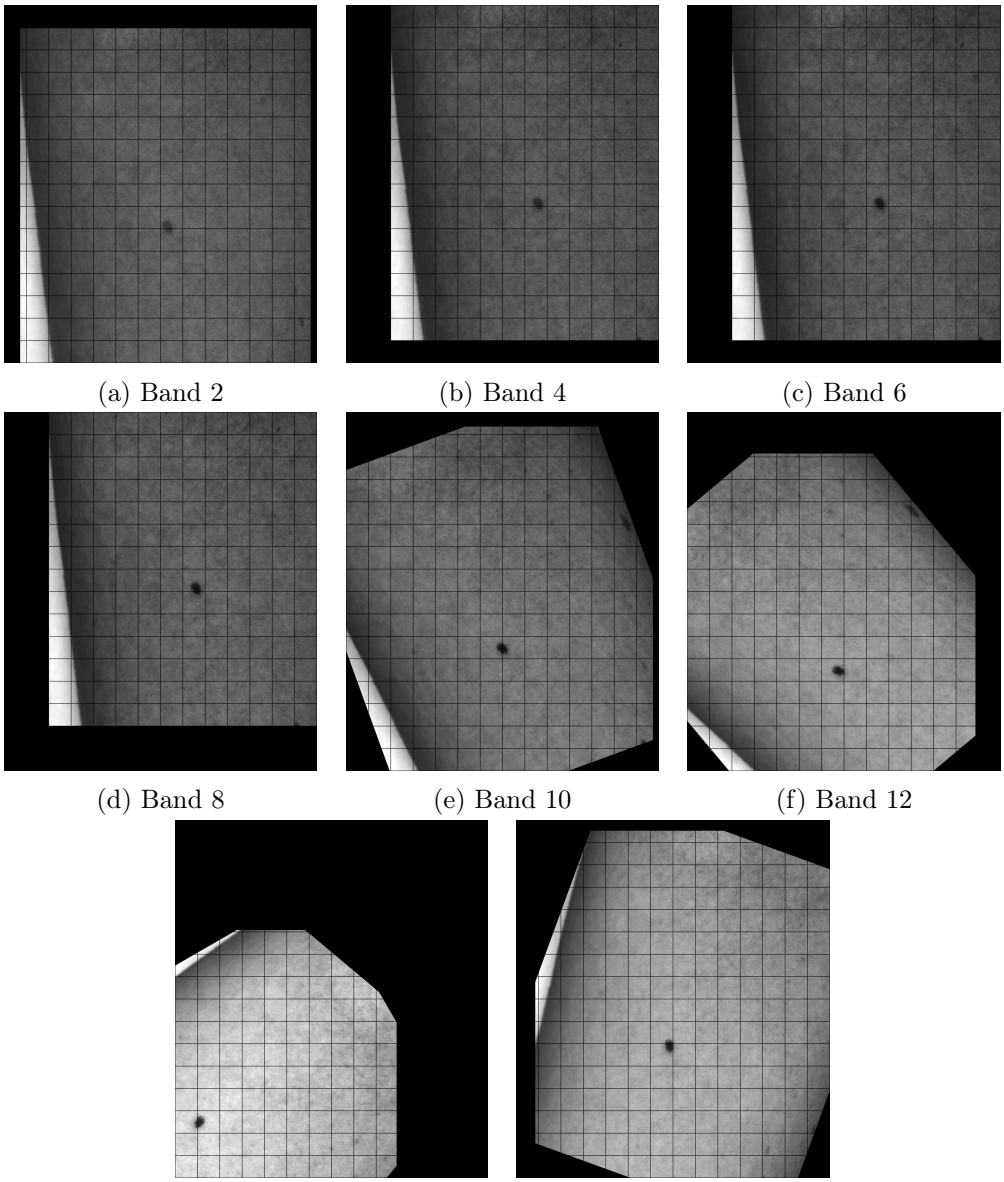
While the translation transform appears to be bad at correcting the introduced misplacement, it does not introduce new misplacement were none was created either.

With a run time of approximately 68 cpu hours, this transformation method is much faster to compute compared to the other transformation methods. In this way of calculating, it should take approximately 68 hours to complete the registration process when using a single core processor. However, this assumes that the speed of the single core processor is equal to the speed of the cores used when testing (appendix C). Similarily, using 30 cores should speed up the process, making it finish in less than three hours.

| Band | Iterations | X[1] | Y[2] |
|---|---|---|---|
| 1 | 10 | 0.999 | -0.037 |
| 2 | 1387 | 27.564 | -103.911 |
| 3 | 4 | 0.400 | -0.007 |
| 4 | 8 | 0.208 | -0.290 |
| 5 | 3 | 0.300 | -0.009 |
| 6 | 6 | 0.195 | -0.300 |
| 7 | 4 | 0.399 | -0.026 |
| 8 | 8 | 0.241 | -0.127 |
| 9 | 9 | 0.894 | -0.100 |
| 10 | 717 | 26.954 | -65.223 |
| 11 | 15 | 0.723 | -0.097 |
| 12 | 2239 | 113.833 | -185.393 |
| 13 | 10 | 0.445 | -0.069 |
| 14 | 7141 | 409.304 | -491.747 |
| 15 | 10 | 0.222 | -0.056 |
| 16 | 1005 | -85.749 | -48.004 |
| 17 | 2 | 0.088 | -0.026 |
| 18 | 5 | 0.057 | -0.024 |
| 19 | 8 | 0.026 | -0.019 |
| 20 | 10 | 0.003 | -0.017 |

1. Translation in X-direction
2. Translation in Y-direction

Table 4.6: Results from `TranslationTransform`



(a) Band 2 (b) Band 4 (c) Band 6

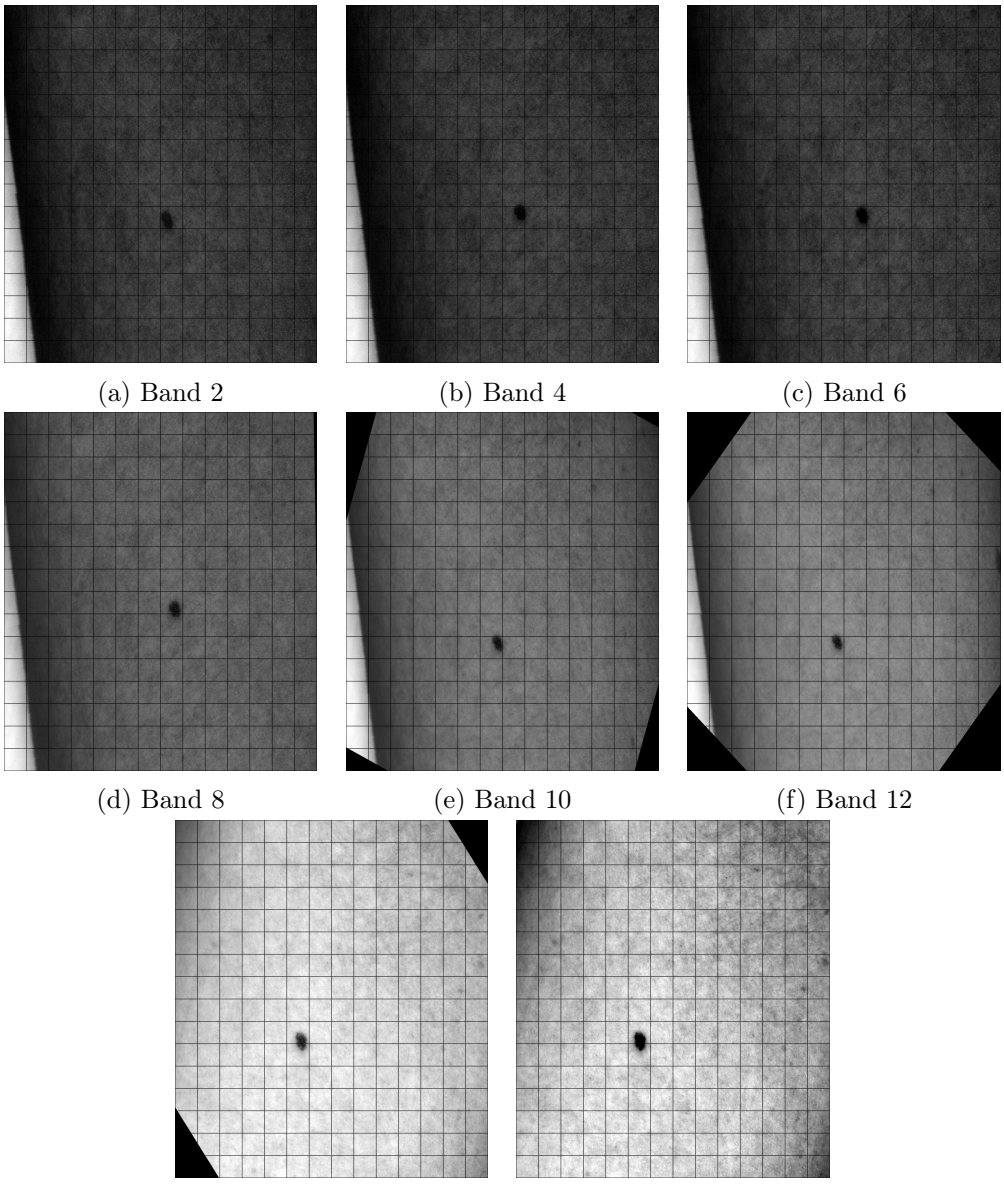
(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16

Figure 4.1: Registration results using `TranslationTransform`

### 4.2.2 Affine

Using the parameters defined in 4.5, the optimizer stop condition for `itk::AffineTransform` is "Maximum number of iterations exceeded". The fact that the optimizer stop due to the maximum iterations being met means that the registration has failed, as the metric does not meet the set criteria for recognized differences in the images. On a side-note, this also means very long run time.



(a) Band 2 (b) Band 4 (c) Band 6

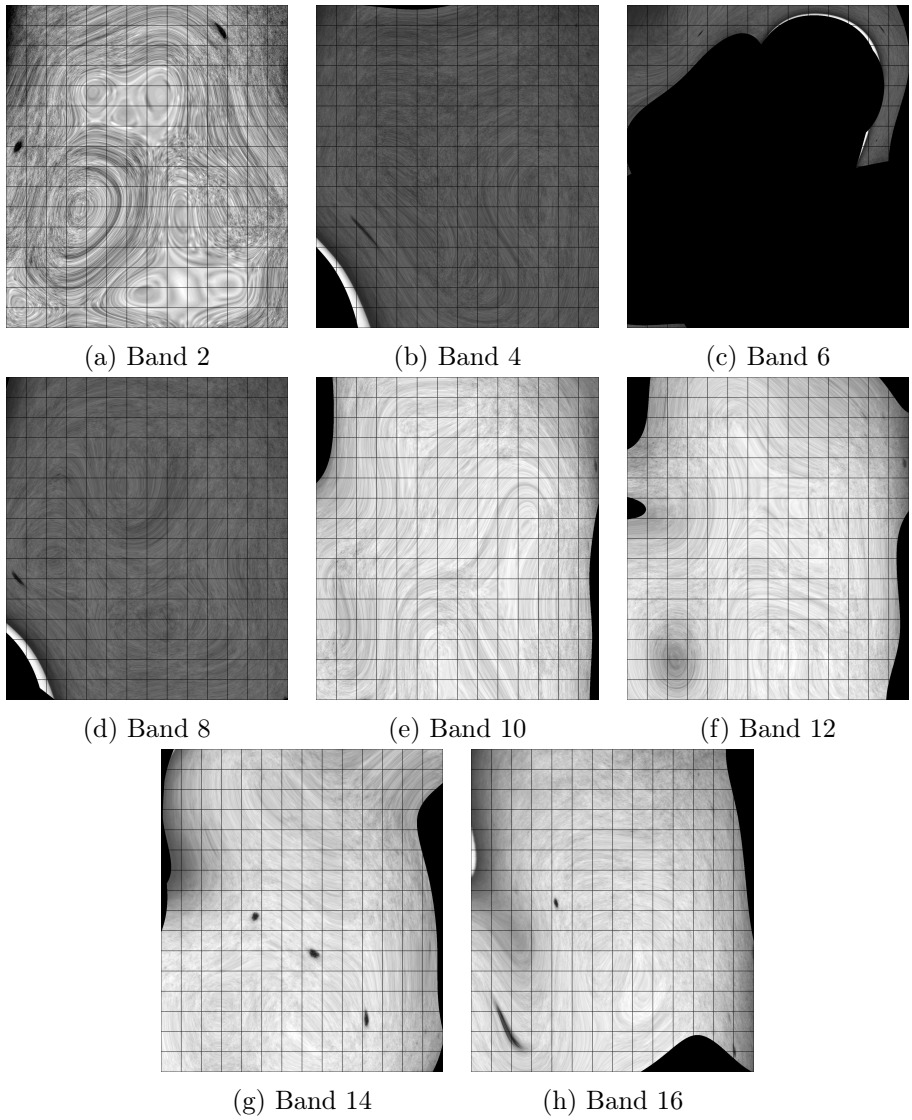
(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16

Figure 4.2: Registration results using `AffineTransform`

### 4.2.3 BSpline

Using the parameters defined in 4.5, `itk::BSplineTransform` reaches the stop condition for the optimizer at iterations between 7 and 400. The returned transform parameters are of the deformable field, not any translation in x- or y-direction, or angle translation. While the image-set to be registered are not deformable image problems, the preliminary tests did show some promise. The results from `itk::BSplineTransform` are illustrated in figure 4.3, and quite clearly show that this registration method is not suited for the testset.



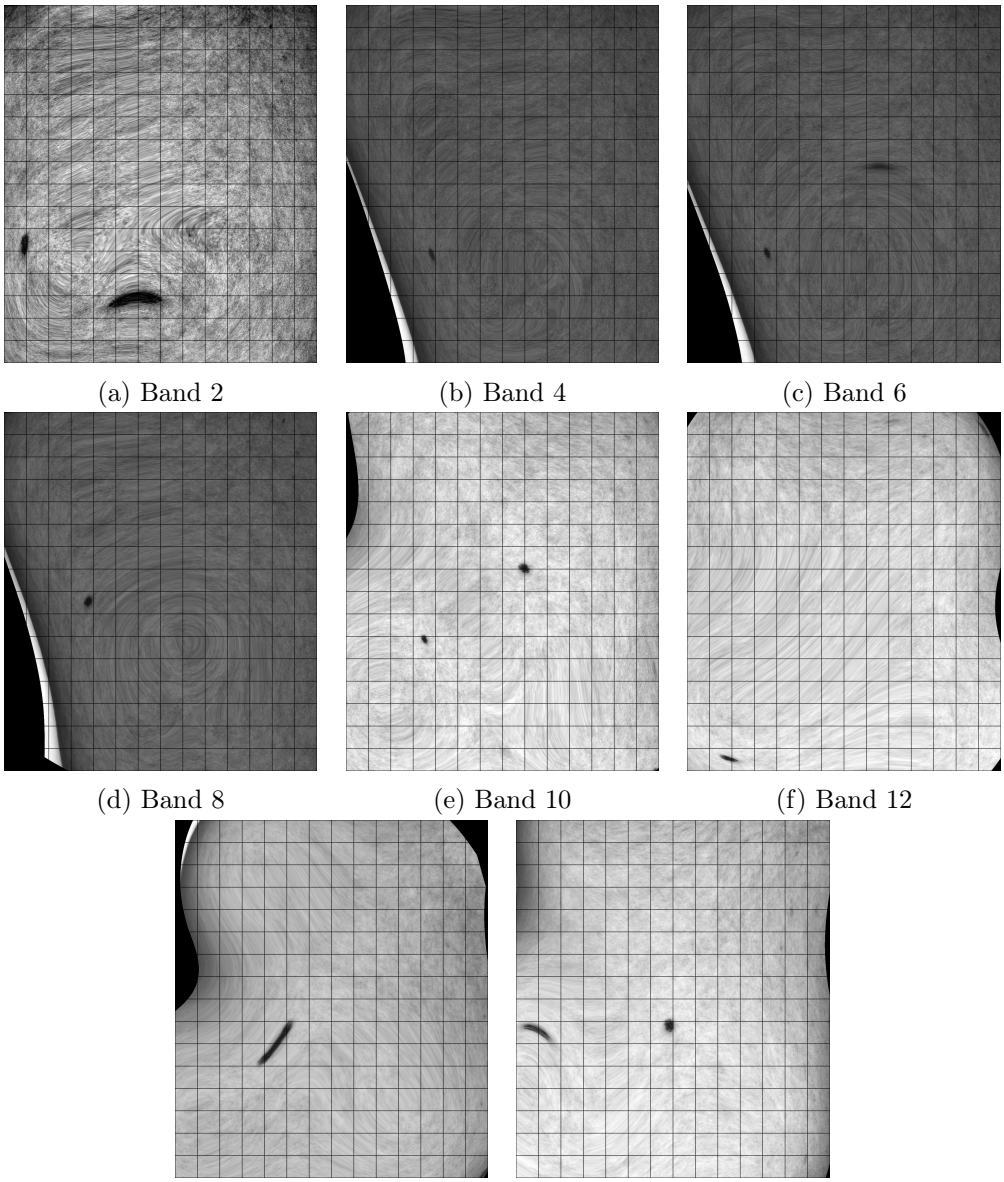
(a) Band 2 (b) Band 4 (c) Band 6

(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16

Figure 4.3: Registration results from `itk::BSplineTransform`

Using more relaxed parameters, specifically setting learning rate to 1.5, minimum step length to 0.05 and number of levels (scaling) to 3, shows better results, as shown in figure 4.4. Better results is a liberal term, but applies as the shape of the arm is somewhat maintained through the images, and the birthmark on the arm "only" doubles, not triples as evident in figure 4.3g.



(a) Band 2 (b) Band 4 (c) Band 6

(d) Band 8 (e) Band 10 (f) Band 12

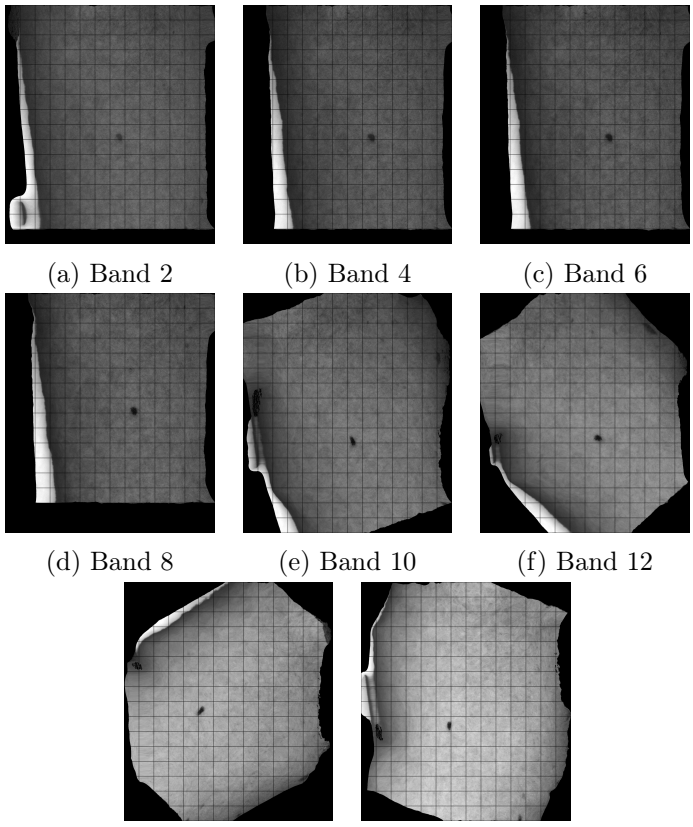
(g) Band 14 (h) Band 16

Figure 4.4: Relaxed registration results from `itk::BSplineTransform`

### 4.2.4 Demons

As with `itk::BSplineTransform`, `itk::DemonsRegistrationFilter` returns a deformable field. This deformable field is a warping field, and not comparable to the rigid transformation types. Comparing the left side of the registration output in figures 4.5e, 4.5f and 4.5h with the fixed image band in figure 3.4i, it is evident that this transformation method does work to some degree, but lacks the larger translation parameters necessary for complete registration. This is evident in the far left areas of these image bands, where the edge of the image object has been transformed to approximately the same angle and position as in the fixed image.

While this registration method does not output correctly aligned image bands, this method shows promise for use in multi-modal or deformable image registration problems, where the larger translations has prviously been handled by a rigid registration method.



(a) Band 2 (b) Band 4 (c) Band 6

(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16

Figure 4.5: Registration results from `itk::DemonsTransform`

### 4.2.5 Rigid

Using the parameters defined in table 4.5 yields the results for `itk::CenteredRigid2DTransform` shown in table 4.7.

| Band | I[1] | X[2] | Y[3] | Angle[4] | I[1] | X[2] | Y[3] | Angle[4] |
|---|---|---|---|---|---|---|---|---|
| 1 | 875 | 0.18 | 0.37 | 0.020 | 839 | -0.82 | 0.41 | 0.023 |
| 2 | 1912 | 100.60 | -100.82 | 0.008 | 5000 | 72.28 | -44.37 | -0.004 |
| 3 | 1753 | 0.11 | 0.38 | 0.014 | 1761 | -0.29 | 0.39 | 0.014 |
| 4 | 2123 | 200.64 | -100.80 | 0.018 | 1870 | 200.43 | -100.54 | 0.017 |
| 5 | 2312 | -0.07 | 0.36 | 0.021 | 2323 | -0.37 | 0.37 | 0.021 |
| 6 | 1945 | 200.55 | -100.77 | 0.021 | 1964 | 200.35 | -100.47 | 0.021 |
| 7 | 2112 | -0.13 | 0.33 | 0.022 | 2103 | -0.55 | 0.36 | 0.022 |
| 8 | 2652 | 200.51 | -200.72 | 0.021 | 2640 | 200.27 | -200.59 | 0.020 |
| 9 | 1742 | -0.16 | 0.34 | 0.019 | 1599 | -1.06 | 0.44 | 0.019 |
| 10 | 472 | 14.12 | 0.42 | -20.053 | 1232 | -10.16 | 58.80 | -20.053 |
| 11 | 1316 | -0.19 | 0.30 | 0.015 | 1230 | -0.91 | 0.40 | 0.015 |
| 12 | 466 | 27.32 | -6.62 | -40.022 | 2208 | -58.36 | 130.81 | -40.019 |
| 13 | 1162 | -0.15 | 0.23 | 0.009 | 1089 | -0.59 | 0.30 | 0.008 |
| 14 | 1141 | -19.82 | -27.96 | 59.914 | 2344 | -74.77 | 92.85 | -139.556 |
| 15 | 973 | -0.13 | 0.15 | 0.005 | 967 | -0.36 | 0.20 | 0.005 |
| 16 | 617 | -11.89 | -7.05 | 20.035 | 1329 | 62.55 | 35.14 | 20.041 |
| 17 | 781 | -0.08 | 0.08 | 0.002 | 777 | -0.16 | 0.11 | 0.002 |
| 18 | 735 | -0.06 | 0.06 | 0.001 | 718 | -0.12 | 0.09 | 0.001 |
| 19 | 622 | -0.04 | 0.05 | -0.001 | 613 | -0.07 | 0.07 | 0.000 |
| 20 | 536 | -0.02 | 0.03 | -0.001 | 535 | -0.02 | 0.05 | -0.001 |

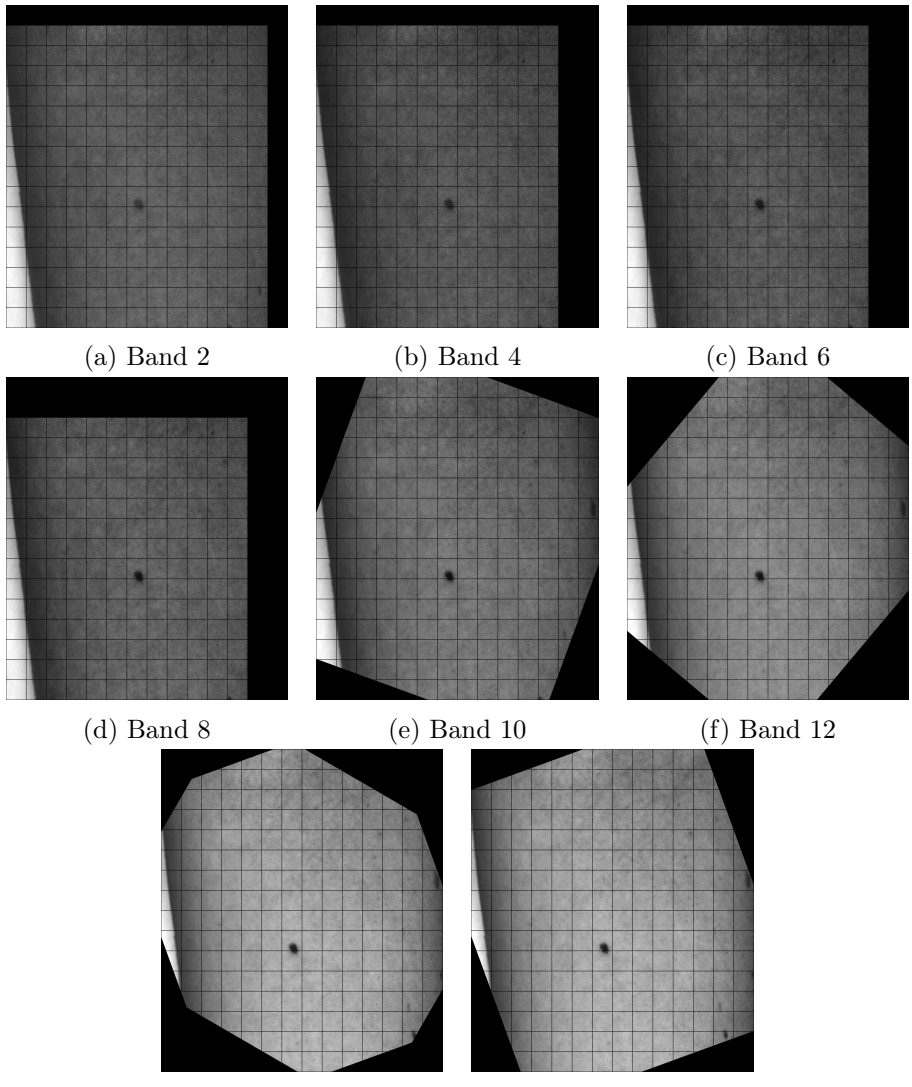
(a) `Rigid2D` (b) `Rigid2D` after `TranslationTransform`

1. Number of iterations
2. Translation in X-direction
3. Translation in Y-direction
4. Translation angle in degrees

Table 4.7: Results from `Rigid2DTransform`

As can be seen from table 4.7a, rigid transform using the parameters in table 4.5 correctly locate the introduced misplacement and corrects them on a sub-pixel level. This can be seen visually by noticing the placement of the birth mark compared to the overlaid grid in figure 4.6. Rigid transform also correctly locates the misplacement where translation transform has been applied first, in most of the image bands.

However, this transform is not able to locate the additionally introduced misplacement in the image bands where an angular offset was introduced, when translation transform has been applied first. That is, image bands 10, 12, 14 and 16. The translation transform moves these image bands partly out of the search space, making the optimizer stop due to the maximum iterations being met. This may be due to the mean square metric finding improvements for correlating points where part of the image still is partly outside the search space, effectively tricking it to search in the wrong direction. This is evident in figures 4.7e, 4.7f, 4.7g and 4.7h, where the metric has moved in the wrong direction the longest in figure 4.7g.

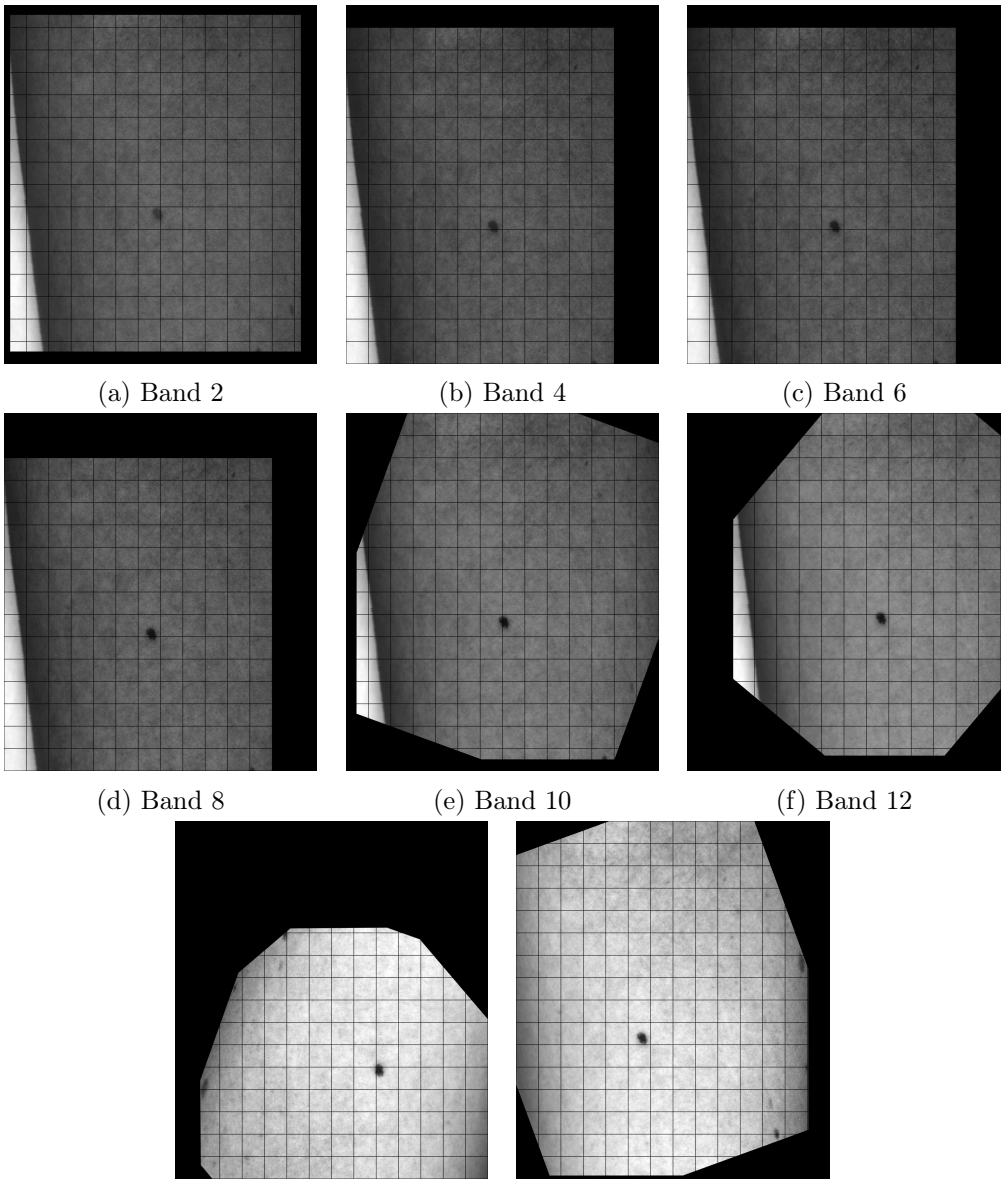


(a) Band 2 (b) Band 4 (c) Band 6

(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16

Figure 4.6: Registration results using `Rigid2DTransform`



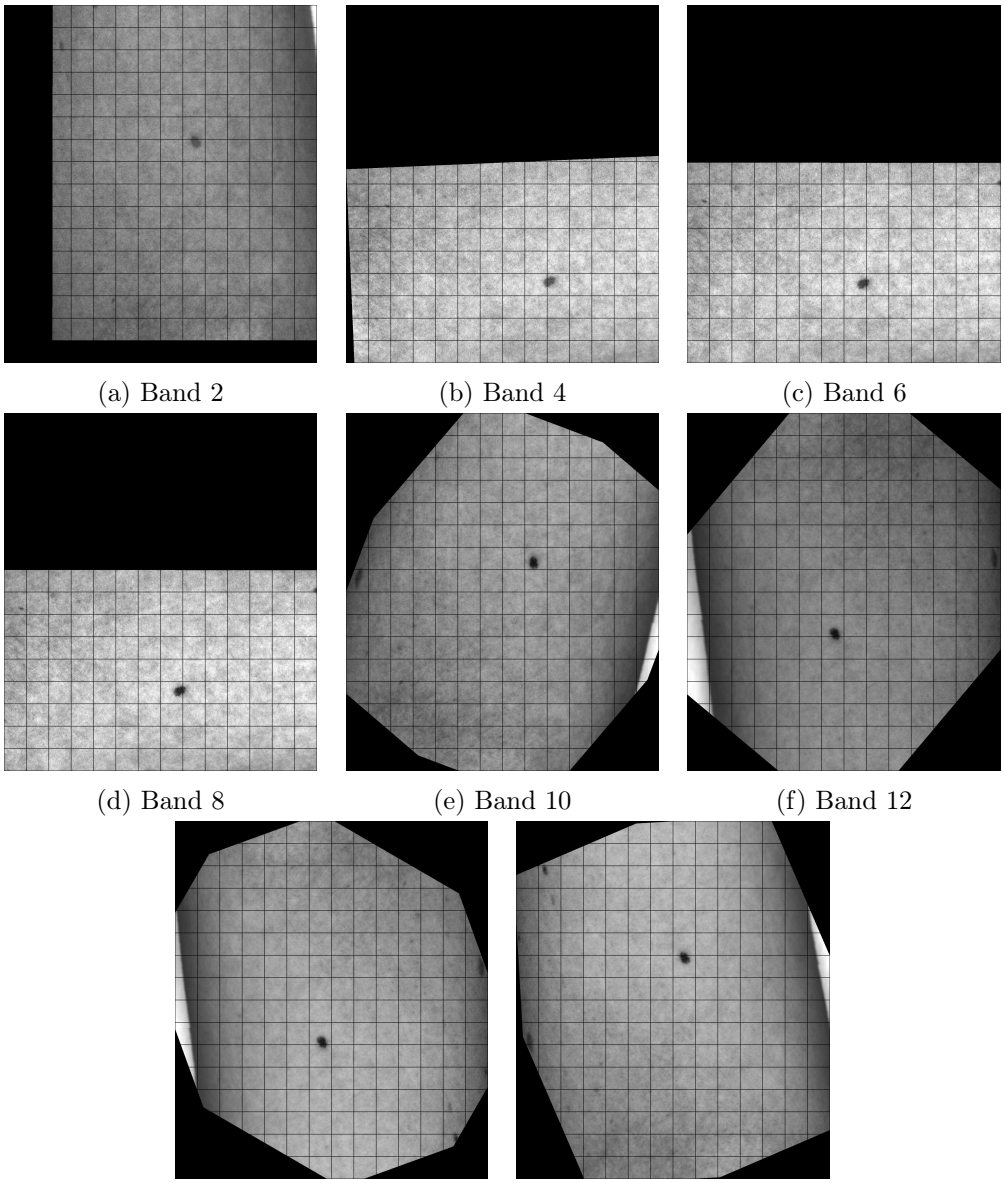
(a) Band 2 (b) Band 4 (c) Band 6

(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16

Figure 4.7: Registration results using `Rigid2DTransform`, after `TranslationTransform`

Results from `Rigid2DTransform` using parameters in table 4.5, but changing learning rate from 0.1 to 2.0 is seen in figure 4.8. Image bands 2, 10, 12, 14, and 16 are not registered correctly. What is interesting here, is that the failure to correctly register the image bands shown in figures 4.8b, 4.8c and 4.8d persists for rigid transformation using learning rates above 0.7. The high learning rate contributes to the metric diverging from the correct solution, resulting in transformations as seen in 4.8.



(a) Band 2 (b) Band 4 (c) Band 6

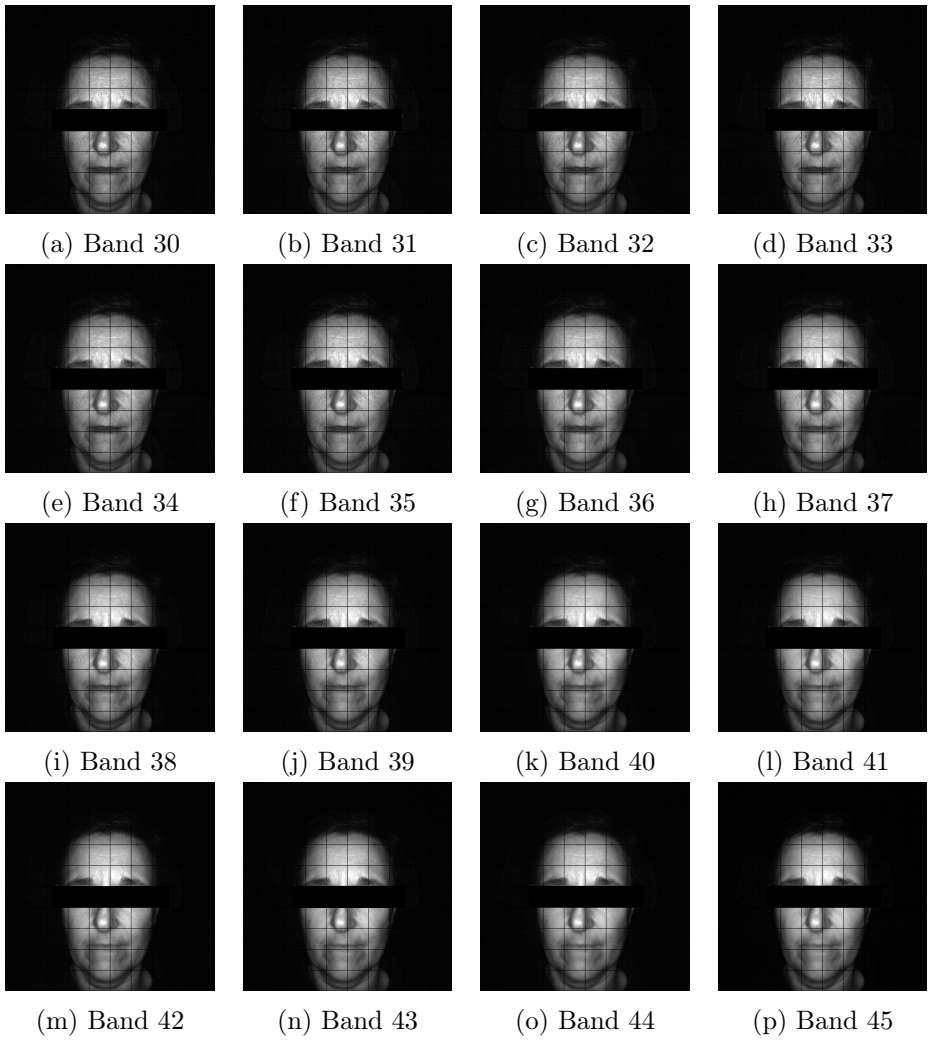
(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16

Figure 4.8: Registration results using `Rigid2DTransform`

#### 4.2.5.1 AOTF

The images from the AOTF camera was run with `Rigid2DTransform` using the parameters in table 4.5. Some of the results are presented in figure 4.9, as these consist of 91 image bands. The run time was 37 hours, or approximately 0.4 hours per image band with one cpu core. Some misalignment are detectable visually in the first 20 bands.



(a) Band 30 (b) Band 31 (c) Band 32 (d) Band 33

(e) Band 34 (f) Band 35 (g) Band 36 (h) Band 37

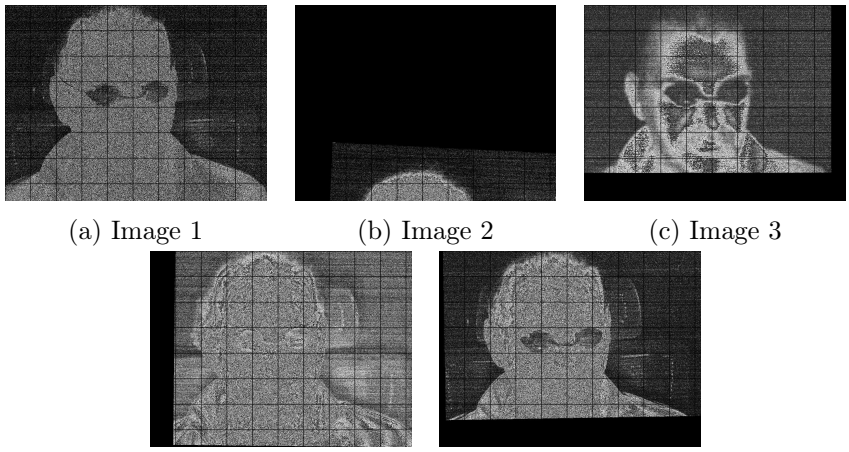
(i) Band 38 (j) Band 39 (k) Band 40 (l) Band 41

(m) Band 42 (n) Band 43 (o) Band 44 (p) Band 45

Figure 4.9: Subset of image bands from AOTF

#### 4.2.5.2 Five-camera

The raw images from the five-camera system in figure 3.2 has been tested with `Rigid2DTransform`, using a learning rate of 1.0 and minimum step length of 0.01. The results are presented in figure 4.10, where image 2 has been set as the first input, and thus becomes the reference (fixed) image. Image 2 in figure 3.2 becomes image 1 in 4.10, and vice versa. Lowering learning rate and minimum step length lead to poor registration results, effectively moving the entire image out of bounds.



(a) Image 1 (b) Image 2 (c) Image 3

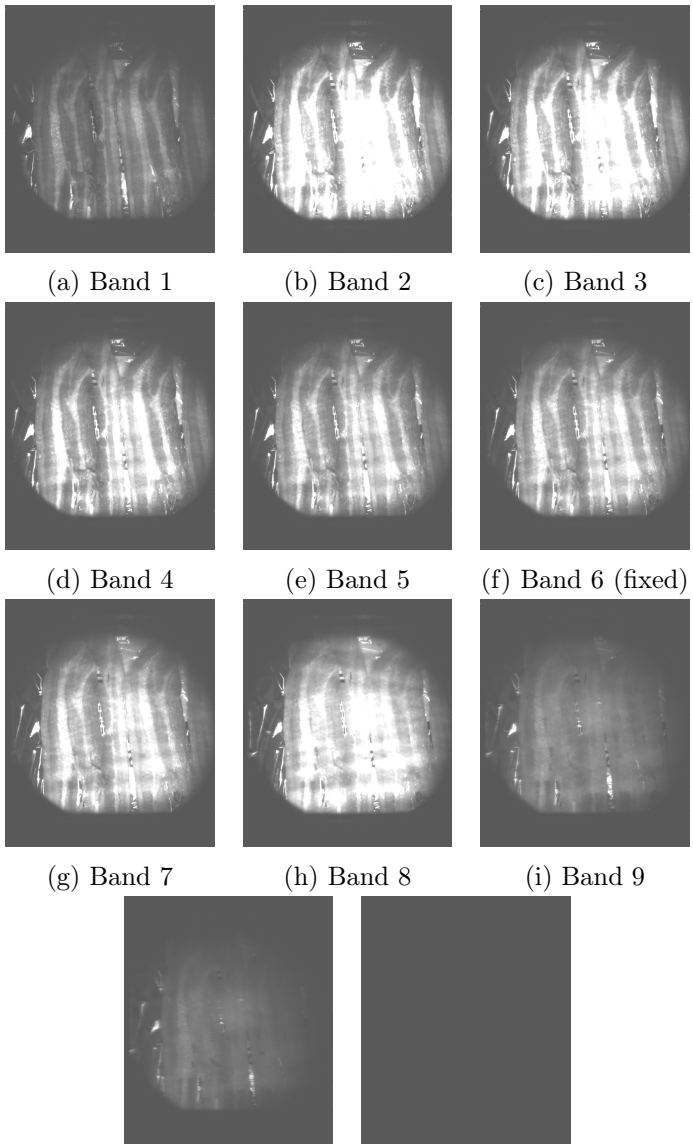
(d) Image 4 (e) Image 5

Figure 4.10: Multispectral images registered with `Rigid2DTransform`

These images are both rigid and deformable registration problems, as the cameras have different angles towards the object. Any rigid transform will thus never be completely accurate. With the exception of image 2, which is moved partly out of the image, the images are aligned approximately correctly at the ears, hairline and eye protection device. A decent solution would be rigidly registering the images, then correct the remainder with a deformable image registration method.

#### 4.2.5.3 Zyla

`Bacon.mat` (figure 4.11) from the Zyla 5.5 sCMOS camera [59] was run with `Rigid2DTransform` using the parameters in table 4.5, results in figure 4.12.



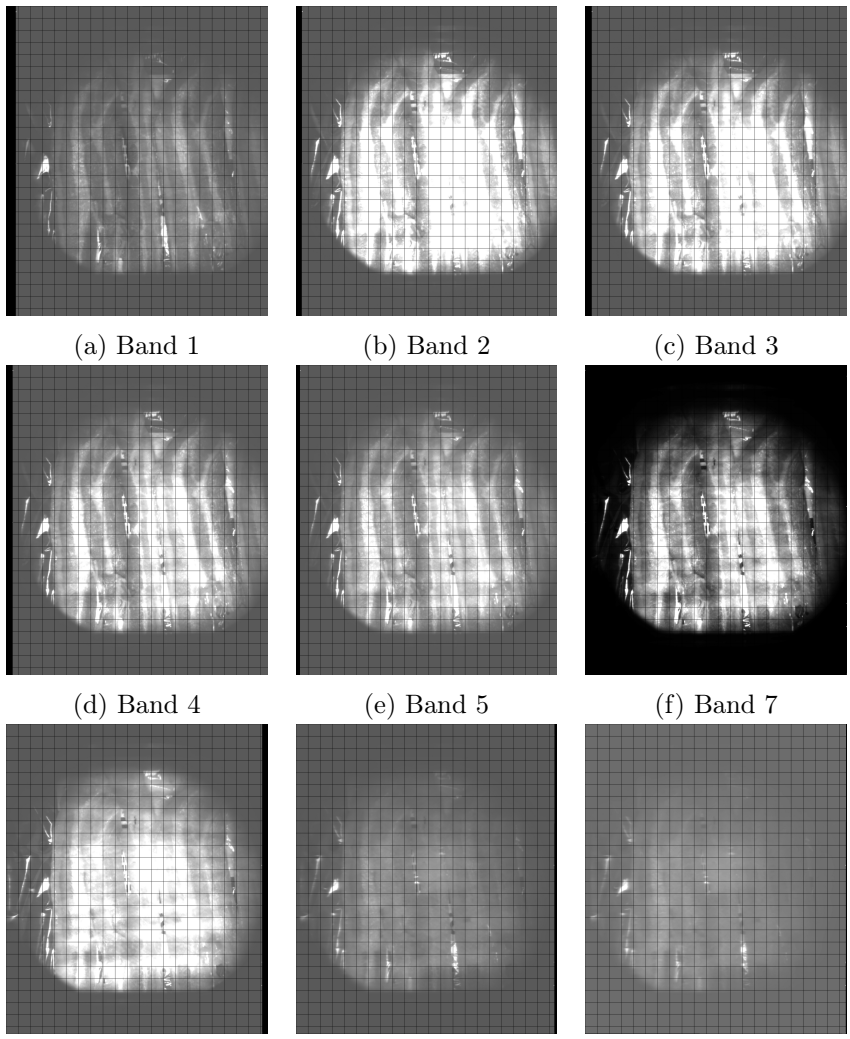
(a) Band 1 (b) Band 2 (c) Band 3

(d) Band 4 (e) Band 5 (f) Band 6 (fixed)

(g) Band 7 (h) Band 8 (i) Band 9

(j) Band 10 (k) Band 11

Figure 4.11: Input `Bacon.mat`



(a) Band 1 (b) Band 2 (c) Band 3

(d) Band 4 (e) Band 5 (f) Band 7

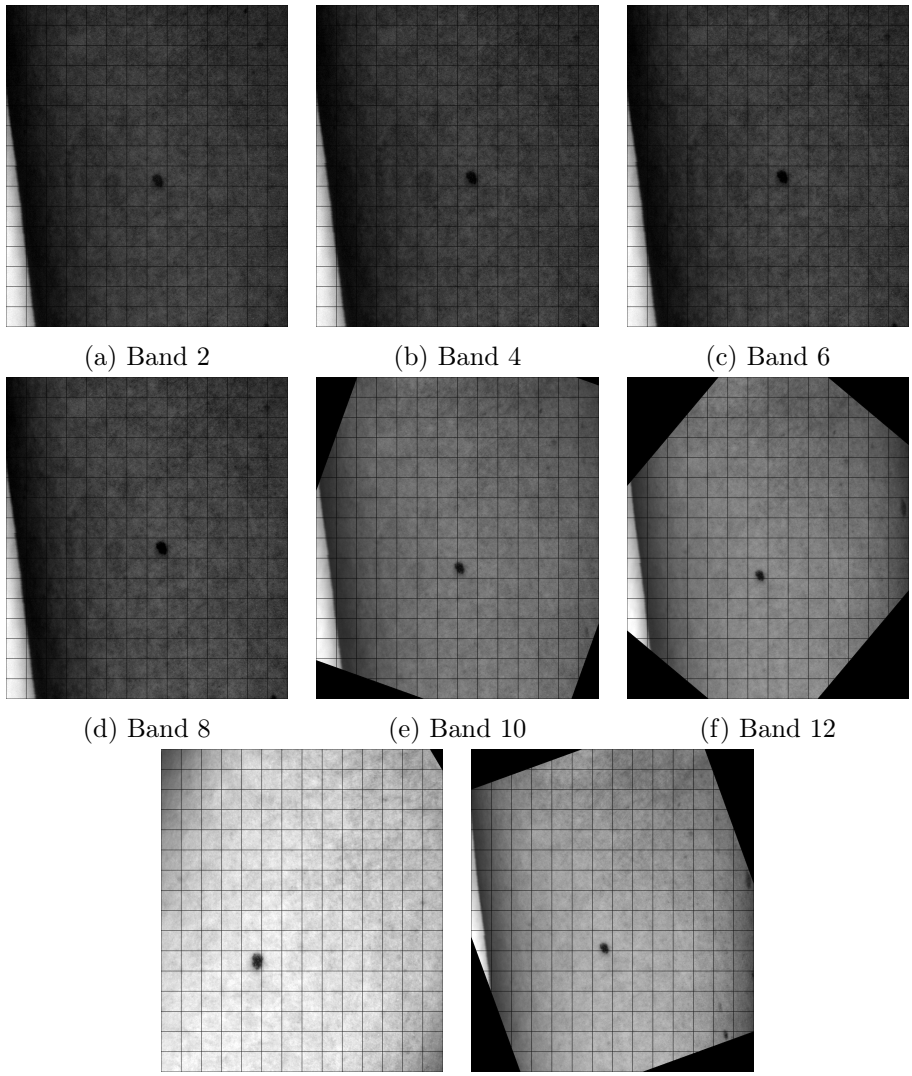
(g) Band 8 (h) Band 9 (i) Band 10

Figure 4.12: Registration results of `Bacon.mat` using `Rigid2DTransform`

The run time for this registration was 437 cpu hours, or roughly 44 hours per image band. While there are no visually detectable errors in these results, it is not possible to validate this registration further. As the grid size is 100x100 pixels, the images are too large to qualitatively judge whether deviations of one to ten pixels are present from a true registration. The accuracy and precision of the results must be judged through clinical use by the researcher using these images[59].

### 4.2.6 Similarity

Using the parameters in table 4.5, the results for `itk::Similarity2DTransform` are as shown in 4.13. As with Affine transform, discussed in 4.2.2, the stop condition is the number of iterations, however, the results are more accurate. Nonetheless, this highlights the issue of applying transformations that includes scale as a parameter, perhaps especially in registration problems where the scale is already correct. The affine transform has two variables for scaling, whereas similarity transform has one.



(a) Band 2 (b) Band 4 (c) Band 6

(d) Band 8 (e) Band 10 (f) Band 12

(g) Band 14 (h) Band 16

Figure 4.13: Registration results using `Similarity2DTransform`

## 4.3 Summary

The program was tested with different parameters. The results and specific run times are presented in table 4.8. While it becomes quite clear from the results presented in table 4.8 that rigid transform provide better results than the other transformation methods for this specific task, it is worth noticing that the run time decreases for smaller learning rates.

A two-layered approach has been implemented as an option in order to speed up the run time, where a coarse registration using translation transform is used before the true transformation is applied. This has shown to increase run time, as well as resulting in false registrations in the image bands with angular offsets. However, this method has provided faster run time in the image bands without angular offsets. In images with offsets in (x,y)-directions, and no other offsets, this approach is valuable in terms of reduced run time.

As previously stated in chapters 4.2.3 and 4.2.4, the final parameters for the BSpline and Demons transforms are not comparable to the rest, as these consist of a summary of the deformation field itself.

| Method | LR[1] | MSL[1] | Run time[2] | Pixels[3] | Pixels[4] | Angle[3] | Angle[4] |
|---|---|---|---|---|---|---|---|
| Affine | 1.0 | 0.0010 | >500 | 33.88 | 39.15 | 21.262 | 12.044 |
| Affine | 0.1 | 0.0001 | >500 | 37.59 | 33.52 | 7.699 | 8.739 |
| Rigid | 2.0 | 0.0001 | 296 | 20.45 | 28.96 | 75.860 | 82.924 |
| Rigid | 1.0 | 0.0001 | 287 | 8.64 | 25.11 | 11.258 | 29.749 |
| Rigid | 1.0 | 0.0010 | 275 | 7.90 | 26.64 | 11.282 | 29.750 |
| Rigid | 0.9 | 0.0001 | 269 | 1.16 | 3.31 | 0.025 | 0.020 |
| Rigid | 0.8 | 0.0001 | 322 | 0.96 | 2.68 | 0.026 | 0.020 |
| Rigid | 0.7 | 0.0001 | 247 | 0.34 | 0.22 | 0.025 | 0.020 |
| Rigid | 0.6 | 0.0001 | 245 | 0.34 | 0.22 | 0.024 | 0.018 |
| Rigid | 0.5 | 0.0001 | 244 | 0.33 | 0.22 | 0.024 | 0.019 |
| Rigid | 0.1 | 0.0001 | 255 | 0.33 | 0.23 | 0.024 | 0.019 |
| Similarity | 1.0 | 0.0010 | >500 | 71.20 | 56.67 | 5.711 | 9.735 |
| Translation | 0.1 | 0.0001 | 68 | 75.94 | 118.55 | - | - |
| Translation[5] | 0.1 | 0.0001 | 67 | 75.97 | 118.64 | - | - |

1. Parameters; Learning rate and Minimum step length
2. Run time based on a single core processor, measured in hours
3. Mean offset from true value
4. Standard deviation from true value
5. Using Mattes Mutual Information metric

Table 4.8: Results from multiple runs

The run time decreases for smaller learning rates, and the reasoning for this decrease in run time appears to be due to faster convergence on the true corrections when the learning rate is sufficiently low. With higher learning rates, the metric diverges around its stop condition, and thus spends a longer time, or even makes it impossible, to reach the correct value. This, however, does not hold true for the rigid transformation using 0.1 as learning rate. This may indicate that the optimal learning rate for this image-set is slightly below 0.5, or at 0.5. Another indication that 0.5 is a more optimal value for the learning rate than 0.1 is the slightly lower standard deviation when using 0.5 versus 0.1.

Initially it was assumed from the literature that increased accuracy and precision would also lead to increased run time. This, however, has been shown to not hold true. The metric has been shown to converge faster to the true registration results when using an optimally small learning rate. That is, when the learning rate is large, the metric will diverge around the true registration result, leading to an increased number of iterations, and thus resulting in an increased run time. This especially holds true for affine, similarity and translation, where the increased number of parameters without fault leads to divergence in the metric, resulting in the maximum number of iterations being met before the minimum step length.

As the computational cost of the program is quite high, the results presented in table 4.8 are the results available at time of writing this thesis, where the testing period started roughly two months before. The virtual machine supplied by Omega Verksted[78] roughly a month before the thesis deadline drastically sped the testing along.

It is also worth noting that the large mean offset and standard deviation from true value in rigid transforms using learning rate 0.8 through 1.0 is due to faulty registration results in image bands 4, 6 and 8, as seen in figures 4.8b, 4.8c and 4.8d, respectively, with only small margins of error in the other bands.

Although optimal results are possible, mean square metric may be unsuitable for this kind of hyperspectral imaging. As stated from the literature in chapter 2.2.4, mean square metric has a large capture radius. In images with similar intensities across the image, this may lead to erronous detections, as the average intensity change between the images may be very similar in different regions of the images, resulting in the faulty results mentioned in the previous paragraph. This is also evident in the translation transform, where the optimizer does reach the minimum step length, although it has no concept of angular translations.

The literature essentially agrees with these results; More transformation parameters increases the difficulty for the metric and optimizer to converge on the true transformation result. This is clearly shown in the results from the Affine and Similarity transforms. Furthermore, sub-pixel registration as achieved with the Rigid transform is expected with reliable and accurate registration methods.

Due to lack of computing power, further in-depth testing of affine and similarity transformations has been omitted, as even with 30 cores at disposal, a single run with affine transform takes five days to complete. With the exception of a few days, the virtual machine has been running the program without pause.

Whether the registration method has decent resource requirements, or is applicable in clinical use has to be weighted by the end-user, as this is highly dependent on the end-users goal and requirements of the method. In the test set with induced errors, any assumption verification is moot, as the errors to be corrected are known.

In the other image sets, however, assumption verification is very much relevant, as general information about camera distance, pixel size relative to real world scale, and how large movement in the objects are present between image bands are unknown. Thus, visual verification, combined with clinical use is necessary in these image sets. With clinical use, information extraction of the object in question after registration, comparing this information with information obtained through other means will give a strong indication whether these images are registered correctly.

# 5 Conclusion

The registration methods providing additional parameters have proven to be unsuccessful in correctly registering the image bands. Whether this is due to a bad choice of metric, or is due to no introduced errors along these parameters, or a combination, is not possible to conclude. However, it is reasonable to assume that a combination is the plausible source of error, as better metric values when applying these parameters will lead to divergence from the true registration result.

While BSpline deformable registration initially showed some promise in the preliminary testing, this method is unsuitable for the provided image sets. This may be due to the implementation of a metric which is highly unsuitable for HSI, but not enough available information exist in order to draw this conclusion. Demons deformable registration is similarly unsuited for the simulated image set, as the method does not handle large translations.

The rigid transformation satisfy the validation methods presented in chapter 2.5. With sub-pixel precision of the introduced errors, the registration method is accurate. Similarly, the introduced errors are variations in the input, and result in similar variations in the output, and the method is thus stable. The method is reliable for several parameter inputs, and the method has low complexity. That is, a relatively low number of variable parameters for the metric. The fastest computational time for the method which correctly registers the images was found to be approximately six hours with one processor core, using images of 1601x1401 pixels.

## 5.1 Future work

Mattes Mutual Information metric has been implemented for the translation transform, and show similar results as Mean Square metric. However, several additional metrics are available in the ITK library. It is possible that some of these provide better results when using the Affine-, Similarity- or BSpline transforms. While the computational time didn't allow for the testing of these metrics at this time, it is something worth looking into.

Some implementations of the ITK library has been developed for use with GPU processing. At this time, four filters has been implemented, where Demons Deformable registration is the only registration method[50]. Additionally, implementations using CUDA has been developed for use with ITK, although the latest available package is from 2010[19]. While this may require re-writing alot of code, this is something definitely worth looking into, as, according to NVidia, four GPUs in the Tesla 10 series is roughly equivalent to 230 processing cores[71].

# References


[1] J. Anderson, "A rapid and accurate method to realign pet scans utilizing image edge information.", *Journal of nuclear medicine: Official publication, Society of Nuclear Medicine*, vol. 36, no. 4, pp. 657–669, 1995.

[2] J. L. Andersson, A. Sundin, and S. Valind, "A method for coregistration of pet and mr brain images", 1995.

[3] E. Angelopoulou, "The reflectance spectrum of human skin", *Technical Reports (CIS)*, p. 584, 1999.

[4] *Antsr by stnava*, `http://stnava.github.io/ANTsR/`, (Accessed on 01/06/2017).

[5] *Arch linux*, `https://www.archlinux.org/`, (Accessed on 12/16/2016).

[6] T. Ault and M. Siegel, "Frameless patient registration using ultrasonic imaging", *Medical robotics and computer assisted surgery*, pp. 74–81, 1994.

[7] B. B. Avants, N. J. Tustison, M. Stauffer, G. Song, B. Wu, and J. C. Gee, "The insight ToolKit image registration framework", *Frontiers in Neuroinformatics*, vol. 8, Apr. 2014. DOI: `10.3389/fninf.2014.00044`. [Online]. Available: `http://dx.doi.org/10.3389/fninf.2014.00044`.

[8] P. K. Banerjee and A. Toga, "Image alignment by integrated rotational and translational transformation matrix", *Physics in medicine and biology*, vol. 39, no. 11, p. 1969, 1994.

[9] S. Baronti, A. Casini, F. Lotti, and S. Porcinai, "Multispectral imaging system for the mapping of pigments in works of art by use of principal-component analysis", *Applied optics*, vol. 37, no. 8, pp. 1299–1309, 1998.

[10] *Bil, bip, and bsq raster files*, `http://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/bil-bip-and-bsq-raster-files.htm`, (Accessed on 12/07/2016).

[11] A. Bjørgan, *Master's thesis; estimation of skin optical parameters for real-time hyperspectral imaging applications using gpgpu parallel computing*, 2013.

[12] G. Bonanno, G. Puy, Y. Wiaux, R. B. van Heeswijk, D. Piccini, and M. Stuber, "Self-navigation with compressed sensing for 2D translational motion correction in free-breathing coronary MRI: A feasibility study", *PLOS ONE*, vol. 9, no. 8, X. Yang, Ed., e105523, Aug. 2014. DOI: `10.1371/journal.pone.0105523`. [Online]. Available: `http://dx.doi.org/10.1371/journal.pone.0105523`.

[13] L. G. Brown, "A survey of image registration techniques", *ACM Computing Surveys*, vol. 24, no. 4, pp. 325–376, Dec. 1992. DOI: `10.1145/146370.146374`. [Online]. Available: `http://dx.doi.org/10.1145/146370.146374`.

[14] J. Burger and A. Gowen, "Data handling in hyperspectral image analysis", *Chemometrics and Intelligent Laboratory Systems*, vol. 108, no. 1, pp. 13–22, Aug. 2011. DOI: `10.1016/j.chemolab.2011.04.001`. [Online]. Available: `http://dx.doi.org/10.1016/j.chemolab.2011.04.001`.

[15] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. Springer, 2003, ISBN: 0306474832.

[16] ——, *Hyperspectral imaging: Techniques for spectral detection and classification*. Springer Science & Business Media, 2003, vol. 1.

[17] C.-T. Chen, C. A. Pelizzari, G. T. Chen, M. D. Cooper, and D. N. Levin, "Image analysis of pet data with the aid of ct and mr images", in *Information processing in medical imaging*, Springer, 1988, pp. 601–611.

[18] J.-S. Chou, S.-Y. J. Chen, G. S. Sudakoff, K. R. Hoffmann, C.-T. Chen, and A. H. Dachman, "Image fusion for visualization of hepatic vasculature and tumors", in *Medical Imaging 1995*, International Society for Optics and Photonics, 1995, pp. 157–163.

[19] *Citk*, `https://code.google.com/archive/p/cuda-insight-toolkit/`, (Accessed on 12/19/2016).

[20] *Cmake*, `https://cmake.org/`, (Accessed on 12/07/2016).

[21] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal, "Automated multi-modality image registration based on information theory", *In: Bizais*, 1995.

[22] M. Denstedt, B. S. Pukstad, L. A. Paluchowski, J. E. Hernandez-Palacios, and L. L. Randeberg, "Hyperspectral imaging as a diagnostic tool for chronic skin ulcers", in *SPIE BiOS*, International Society for Optics and Photonics, 2013, 85650N–85650N.

[23] Y. Diez, A. Oliver, X. Llado, and R. Marti, "Comparison of registration methods using mamographic images", in *2010 IEEE International Conference on Image Processing*, Institute of Electrical and Electronics Engineers (IEEE), Sep. 2010. DOI: `10.1109/ICIP.2010.5653325`. [Online]. Available: `http://dx.doi.org/10.1109/ICIP.2010.5653325`.

[24] *Doxygen: Main page*, `http://www.stack.nl/~dimitri/doxygen/`, (Accessed on 01/06/2017).

[25] *Dramms image registration software*, `https://www.cbica.upenn.edu/sbia/software/dramms/`, (Accessed on 01/06/2017).

[26] *Envi capabilities - image analysis - envi products | harris geospatial*, `http://www.harrisgeospatial.com/ProductsandSolutions/GeospatialProducts/ENVI/ENVICapabilities.aspx`, (Accessed on 12/07/2016).

[27] A. C. Evans, S. Marrett, J. Torrescorzo, S. Ku, and L. Collins, "Mri-pet correlation in three dimensions using a volume-of-interest (voi) atlas", *Journal of Cerebral Blood Flow & Metabolism*, vol. 11, no. 1 suppl, A69–A78, 1991.

[28] J. M. Fitzpatrick, D. L. Hill, and C. R. Maurer Jr, "Image registration", *Handbook of medical imaging*, vol. 2, pp. 447–513, 2000.

[29] S. Fox, *Fundamentals of Human Physiology*. McGraw-Hill Science/Engineering/-Math, 2008, ISBN: 0077226356.

[30] V. Fresse, D. Houzet, and C. Gravier, "Evaluation of cpu and gpu architectures for spectral image analysis algorithms", in *IS&T/SPIE Electronic Imaging*, International Society for Optics and Photonics, 2011, pp. 78720M–78720M.

[31] V. Fresse, D. Houzet, and C. Gravier, "Gpu architecture evaluation for multispectral and hyperspectral image analysis", in *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*, IEEE, 2010, pp. 121–127.

[32] *Gcc, the gnu compiler collection - gnu project - free software foundation (fsf)*, `https://gcc.gnu.org/`, (Accessed on 12/07/2016).

[33] Y. Ge, J. M. Fitzpatrick, J. R. Votaw, S. Gadamsetty, R. J. Maciunas, R. M. Kessler, and R. A. Margolin, "Retrospective registration of pet and mr brain images: An algorithm and its stereotactic validation.", *Journal of computer assisted tomography*, vol. 18, no. 5, pp. 800–810, 1994.

[34] C. Gonzalez, J. Resano, A. Plaza, and D. Mozos, "Fpga implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 1, pp. 248–261, 2012.

[35] A. Hamadeh, P. Sautot, S. Lavallée, and P. Cinquin, "Towards automatic registration between ct and x-ray images: Cooperation between 3d/2d registration and 2d edge detection", *Medical robotics and computer assisted surgery*, pp. 39–46, 1995.

[36] W. R. Hamilton, *Elements of quaternions*. Longmans, Green, & Company, 1866.

[37] P. F. Hemler, P. A. van den Elsen, T. S. Sumanaweera, S. Napel, J. Drace, and J. R. Adler, "A quantitative comparison of residual error for three different multimodality registration techniques", in *Information processing in medical imaging*, Ile Berder, France: IPMI, Kluwer, 1995, pp. 251–62.

[38] P. F. Hemler, T. Sumanaweera, P. A. van den Elsen, S. Napel, and J. Adler, "A system for multimodality image fusion", in *Computer-Based Medical Systems, 1994., Proceedings 1994 IEEE Seventh Symposium on*, IEEE, 1994, pp. 335–340.

[39] P. Henriquez, B. J. Matuszewski1, Y. Andreu-Cabedo1, L. Bastiani, S. Colantonio, G. Coppini, M. D'Acunto, R. Favilla, D. Germanese, D. Giorgi, P. Marraccini, M. Martinelli, M.-A. Morales, M. A. Pascali, M. Righi, O. Salvetti, M. Larsson, T. Stromberg, L. L. Randeberg, A. Bjorgan, G. Giannakakis, M. Pediaditis, F. Chiarugi, E. Christinaki, K. Marias, and M. Tsiknakis, "Mirror mirror on the wall... an unobtrusive intelligent multisensory mirror for well-being status self-assessment and visualization", 2017.

[40] D. L. Hill, *Combination of 3D medical images from multiple modalities*. University of London, 1994.

[41] D. L. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration", *Physics in medicine and biology*, vol. 46, no. 3, R1, 2001.

[42] M. Holden, D. L. Hill, E. R. Denton, J. M. Jarosz, T. C. Cox, and D. J. Hawkes, "Voxel similarity measures for 3d serial mr brain image registration", in *Biennial International Conference on Information Processing in Medical Imaging*, Springer, 1999, pp. 472–477.

[43] *Hyspex, norsk elektro optikk*, `http://www.hyspex.no/`, (Accessed on 12/19/2016).

[44] *Hyview*, `https://ntnu-bioopt.github.io/software/hyview.html`, (Accessed on 12/07/2016).

[45] (2015). Image registration open source computer vision, OpenCV, [Online]. Available: `http://docs.opencv.org/3.1.0/db/d61/group__reg.html` (visited on 09/27/2017).

[46] *Imagej documentation*, `https://imagej.nih.gov/ij/docs/index.html`, (Accessed on 12/07/2016).

[47] *Imagemagick*, `https://www.imagemagick.org/script/index.php`, (Accessed on 12/22/2016).

[48] *Introduction to gimias*, http://gimias.org/index.php/whatisgimias/introduction, (Accessed on 12/08/2016).

[49] *Itk/examples - kitwarepublic*, https://itk.org/Wiki/ITK/Examples, (Accessed on 12/08/2016).

[50] *Itk/release 4/gpu acceleration - kitwarepublic*, https://itk.org/Wiki/ITK_Release_4/GPU_Acceleration, (Accessed on 12/19/2016).

[51] C. G. J. Jacobi, *Gesammelte Werke, Herausgegeben auf Veranlassung der Königlich Preussischen Akademie der Wissenschaften*, 2. New York: Chelsea Publishing Co, 1969 (1881).

[52] H. J. Johnson, M. M. McCormick, and L. Ibanez, *Introduction and Development Guidelines*, 4th ed., T. I. S. Consortium, Ed., ser. The ITK Software Guide. The Insight Toolkit (ITK), May 2016. [Online]. Available: https://itk.org/ItkSoftwareGuide.pdf.

[53] H. J. Johnson, M. M. McCormick, and L. Ibanez, *The itk software guide book 1: Introduction and development guidelines*, Kitware, Inc., 2015.

[54] H. J. Johnson, M. M. McCormick, and L. Ibanez, *The itk software guide book 2: Design and functionality*, Kitware, Inc., 2015.

[55] A. D. Kim and M. Moscoso, "Light transport in two-layer tissues", *Journal of Biomedical Optics*, vol. 10, no. 3, 2005. DOI: 10.1117/1.1925227. [Online]. Available: http://dx.doi.org/10.1117/1.1925227.

[56] S. Klein, M. Staring, K. Murphy, M. Viergever, and J. Pluim, "Elastix: A toolbox for intensity-based medical image registration", *IEEE Transactions on Medical Imaging*, vol. 29, no. 1, pp. 196–205, Jan. 2010. DOI: 10.1109/tmi.2009.2035616. [Online]. Available: http://dx.doi.org/10.1109/TMI.2009.2035616.

[57] S. Klein and M. Staring, "Elastix the manual", in *Elastix the manual*, Elastix, Sep. 2015. [Online]. Available: http://elastix.isi.uu.nl/download/elastix_manual_v4.8.pdf.

[58] H. Lester and S. R. Arridge, "A survey of hierarchical non-linear medical image registration", *Pattern recognition*, vol. 32, no. 1, pp. 129–149, 1999.

[59] M. Liland, *Characterization of atopic dermatitis in children's health (working title)*, Feb. 2017.

[60] 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.

[61] J. A. Maintz, P. A. van den Elsen, and M. A. Viergever, "Registration of spect and mr brain images using a fuzzy surface", in *Medical Imaging 1996*, International Society for Optics and Photonics, 1996, pp. 821–829.

[62] J. Maintz and M. A. Viergever, "A survey of medical image registration", *Medical Image Analysis*, vol. 2, no. 1, pp. 1–36, Mar. 1998. DOI: 10.1016/S1361-8415(01)80026-8. [Online]. Available: http://dx.doi.org/10.1016/S1361-8415(01)80026-8.

[63] T. Makela, P. Clarysse, O. Sipila, N. Pauna, Q. C. Pham, T. Katila, and I. Magnin, "A review of cardiac image registration methods", *IEEE Transactions on Medical Imaging*, vol. 21, no. 9, pp. 1011–1021, Sep. 2002. DOI: 10.1109/TMI.2002.804441. [Online]. Available: http://dx.doi.org/10.1109/TMI.2002.804441.

[64] P. Markelj, D. Tomaževič, B. Likar, and F. Pernuš, "A review of 3D/2D registration methods for image-guided interventions", *Medical Image Analysis*, vol. 16, no. 3, pp. 642–661, Apr. 2012. DOI: 10.1016/j.media.2010.03.005. [Online]. Available: http://dx.doi.org/10.1016/j.media.2010.03.005.

[65] *Mat file i/o library*, https://sourceforge.net/projects/matio/, (Accessed on 01/11/2017).

[66] *Matlab documentation*, https://www.mathworks.com/help/matlab/, (Accessed on 12/19/2016).

[67] 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.

[68] D. Mattes, D. R. Haynor, H. Vesselle, T. K. Lewellyn, and W. Eubank, "Nonrigid multimodality image registration", in *Medical Imaging 2001*, International Society for Optics and Photonics, 2001, pp. 1609–1620.

[69] H. Matthies and G. Strang, "The solution of nonlinear finite element equations", *International Journal for Numerical Methods in Engineering*, vol. 14, no. 11, pp. 1613–1626, 1979, ISSN: 1097-0207. DOI: 10.1002/nme.1620141104. [Online]. Available: http://dx.doi.org/10.1002/nme.1620141104.

[70] C. R. Maurer Jr, G. B. Aboutanos, B. M. Dawant, R. A. Margolin, R. J. Maciunas, and J. M. Fitzpatrick, "Registration of ct and mr brain images using a combination of points and surfaces", in *Medical Imaging 1995*, International Society for Optics and Photonics, 1995, pp. 109–123.

[71] *Medical imaging | nvidia*, http://www.nvidia.com/object/medical_imaging.html, (Accessed on 12/19/2016).

[72] E. Meijering, "A chronology of interpolation: From ancient astronomy to modern signal and image processing", *Proceedings of the IEEE*, vol. 90, no. 3, pp. 319–342, Mar. 2002. DOI: 10.1109/5.993400. [Online]. Available: https://doi.org/10.1109%5C%2F5.993400.

[73] *Miniature spectral imaging camera - gooch & housego*, https://goochandhousego.com/miniature-spectral-imaging-camera/, (Accessed on 01/23/2017).

[74] P. Neelin, J. Crossman, D. Hawkes, Y. Ma, and A. Evans, "Validation of an mri/pet landmark registration method using 3d simulated pet images and point simulations", *Computerized medical imaging and graphics*, vol. 17, no. 4, pp. 351–356, 1993.

[75] O. Neugebauer, *A history of ancient mathematical astronomy*. Springer Science & Business Media, 2012, vol. 1.

[76] *Niftyreg*, http://cmictig.cs.ucl.ac.uk/wiki/index.php/NiftyReg, (Accessed on 12/08/2016).

[77] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[78] *Omega verksted*, https://omegav.no/, (Accessed on 01/13/2017).

[79] Y. Ou, A. Sotiras, N. Paragios, and C. Davatzikos, "Dramms: Deformable registration via attribute matching and mutual-saliency weighting", *Medical Image Analysis*, vol. 15, no. 4, pp. 622–639, Aug. 2011. DOI: 10.1016/j.media.2010.07.002. [Online]. Available: http://dx.doi.org/10.1016/j.media.2010.07.002.

[80] J. Pluim, J. Maintz, and M. Viergever, "Mutual-information-based registration of medical images: A survey", *IEEE Transactions on Medical Imaging*, vol. 22, no. 8, pp. 986–1004, Aug. 2003. DOI: 10.1109/tmi.2003.815867. [Online]. Available: http://dx.doi.org/10.1109/TMI.2003.815867.

[81] *R: The r project for statistical computing*, https://www.r-project.org/, (Accessed on 01/06/2017).

[82] L. L. Randeberg, I. Baarstad, T. Løke, P. Kaspersen, and L. O. Svaasand, "Hyperspectral imaging of bruised skin", *Proc. SPIE*, vol. 6078, 2006. DOI: 10.1117/12.646557. [Online]. Available: http://dx.doi.org/10.1117/12.646557.

[83] L. L. Randeberg, O. A. Haugen, R. Haaverstad, and L. O. Svaasand, "A novel approach to age determination of traumatic injuries by reflectance spectroscopy", *Lasers in surgery and medicine*, vol. 38, no. 4, pp. 277–289, 2006.

[84] L. L. Randeberg, E. L. P. Larsen, and L. O. Svaasand, "Characterization of vascular structures and skin bruises using hyperspectral imaging, image analysis and diffusion theory", *Journal of Biophotonics*, vol. 3, no. 1-2, pp. 53–65, 2010, ISSN: 1864-0648. DOI: 10.1002/jbio.200910059. [Online]. Available: http://dx.doi.org/10.1002/jbio.200910059.

[85] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. 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.

[86] J. Salvi, C. Matabosch, D. Fofi, and J. Forest, "A review of recent range image registration methods with accuracy evaluation", *Image and Vision Computing*, vol. 25, no. 5, pp. 578–596, May 2007. DOI: 10.1016/j.imavis.2006.05.012. [Online]. Available: http://dx.doi.org/10.1016/j.imavis.2006.05.012.

[87] S. Sanchez and A. Plaza, "Real-time implementation of a full hyperspectral unmixing chain on graphics processing units", in *SPIE Optical Engineering+ Applications*, International Society for Optics and Photonics, 2011, 81570F–81570F.

[88] K. Sayood, "Digital image formats", in *Digital Image Forensics: There is More to a Picture than Meets the Eye*, H. T. Sencar and N. Memon, Eds. New York, NY: Springer New York, 2013, pp. 79–121, ISBN: 978-1-4614-0757-7. DOI: 10.1007/978-1-4614-0757-7_3. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-0757-7_3.

[89] R. A. Schowengerdt, *Remote Sensing: Models and Methods for Image Processing*, 3rd ed. Academic Press, Sep. 2006, ISBN: 9780123694072.

[90] S. T. Seljebotn, *Master's thesis; continuous autofocus for line scanning hyperspectral camera*, 2012.

[91] P. Seroul, M. Hébert, M. Cherel, R. Vernet, R. Clerc, and M. Jomier, "Model-based skin pigment cartography by high-resolution hyperspectral imaging", *Journal of Imaging Science and Technology*, 2016.

[92] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado, "Gpu for parallel on-board hyperspectral image processing", *International Journal of High Performance Computing Applications*, vol. 22, no. 4, pp. 424–437, 2008.

[93] J. Setoain, C. Tenllado, M. Prieto, D. Valencia, A. Plaza, and J. Plaza, "Parallel hyperspectral image processing on commodity graphics hardware", in *2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, IEEE, 2006, 8–pp.

[94] C. Shannon, "A mathematical theory of communication, bell system technical journal 27: 379-423 and 623–656", *Mathematical Reviews (MathSciNet): MR10, 133e*, 1948.

[95] T. Skauli, T. V. Haavardsholm, I. Kåsen, G. Arisholm, A. Kavara, T. O. Opsahl, and A. Skaugen, "An airborne real-time hyperspectral target detection system", in *SPIE Defense, Security, and Sensing*, International Society for Optics and Photonics, 2010, 76950A–76950A.

[96] *Stigvis/registration*, `https://github.com/stigvis/registration`, (Accessed on 12/11/2016).

[97] C. Studholme, D. L. Hill, and D. J. Hawkes, "An overlap invariant entropy measure of 3d medical image alignment", *Pattern recognition*, vol. 32, no. 1, pp. 71–86, 1999.

[98] L. Svaasand, L. Norvang, E. Fiskerstrand, E. Stopps, M. Berns, and J. Nelson, "Tissue parameters determining the visual appearance of normal skin and port-wine stains", *Lasers in Medical Science*, vol. 10, no. 1, pp. 55–65, 1995.

[99] J. R. TAYLOR, *Introduction To Error Analysis. Palgrave. 1997.* Palgrave, 1997, ISBN: 093570275X.

[100] J.-P. Thirion, "Image matching as a diffusion process: An analogy with maxwell's demons", *Medical image analysis*, vol. 2, no. 3, pp. 243–260, 1998.

[101] J.-P. Thirion, "Non-rigid matching using demons", in *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*, IEEE, 1996, pp. 245–251.

[102] D. Valencia and A. Plaza, "Fpga-based hyperspectral data compression using spectral unmixing and the pixel purity index algorithm", in *International Conference on Computational Science*, Springer, 2006, pp. 888–891.

[103] M. A. Viergever, J. A. Maintz, S. Klein, K. Murphy, M. Staring, and J. P. Pluim, "A survey of medical image registration–under review", *Medical Image Analysis*, vol. 33, pp. 140–144, 2016.

[104] *Vimdoc : The online source for vim documentation*, `http://vimdoc.sourceforge.net/`, (Accessed on 12/07/2016).

[105] P. Viola and W. M. Wells III, "Alignment by maximization of mutual information", *International journal of computer vision*, vol. 24, no. 2, pp. 137–154, 1997.

[106] M. Y. Wang, J. M. Fitzpatrick, and C. R. Maurer Jr, "Design of fiducials for accurate registration of ct and mr volume images", in *Medical Imaging 1995*, International Society for Optics and Photonics, 1995, pp. 96–108.

[107] *Yed - graph editor*, `https://www.yworks.com/products/yed`, (Accessed on 01/09/2017).

[108] M. M. Yeung, B.-L. Yeo, S.-P. Liou, and A. Banihashemi, "Three-dimensional image registration for spiral ct angiography", in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, IEEE, 1994, pp. 423–429.

[109] T. S. Yoo, Ed., *Insight into Images: Principles and Practice for Segmentation, Registration, and Image Analysis*, 1st ed. A K Peters/CRC Press, Aug. 2004, ISBN: 9781568812175.

[110] B. Zitová and J. Flusser, "Image registration methods: A survey", *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, Oct. 2003. DOI: 10.1016/S0262-8856(03)00137-9. [Online]. Available: http://dx.doi.org/10.1016/S0262-8856(03)00137-9.

# A Code

The source code is available as a zipped archive, or directly on github[96]. All declarations happen in the header files, which are appended. They are also available in the zipped archive.

## A.1 Read image

```
/*
 * Copyright 2015 Asgeir Bjorgan, Lise Lyngsnes Randeberg, Norwegian
    University of Science and Technology
 * Distributed under the MIT License.
 * (See accompanying file LICENSE or copy at
 * http://opensource.org/licenses/MIT
 */

#ifndef READIMAGE_H_DEFINED
#define READIMAGE_H_DEFINED
#include <vector>
#include <boost/regex.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <string>
#include <iostream>

/* Interleave, BIL or BIP. BSQ not supported. */
enum interleave_t{BIL_INTERLEAVE, BIP_INTERLEAVE};

/* Container for hyperspectral header file. */
struct hyspex_header {
  ///Interleave of image
  interleave_t interleave;
  ///Number of pixels in the across-track axis
  int samples;
  ///Number of wavelength bands
  int bands;
  ///Number of lines in the image (along-track)
  int lines;
  ///Offset of the image data from start of the hyperspectral file
  int offset;
  ///Wavelengths
  std::vector<float> wlens;
  ///Datatype of values in hyperspectral file
  int datatype;
};
```

```
39 /* Image subset convenience struct for specifying a subset of the
       hyperspectral image file for reading. */
   struct image_subset {
41   ///Start pixel in the sample direction
     int start_sample;
43   ///End pixel in the sample direction
     int end_sample;
45   ///Start pixel in the line direction
     int start_line;
47   ///End pixel in the line direction
     int end_line;
49   ///Start wavelength band
     int start_band;
51   ///End wavelength band
     int end_band;
53 };

55 /* Errors. */
   enum hyperspectral_err_t {
57   ///Successful
     HYPERSPECTRAL_NO_ERR,
59   ///File not found
     HYPERSPECTRAL_FILE_NOT_FOUND,
61   ///Could not find the requested property in the header file
     HYPERSPECTRAL_HDR_PROPERTY_NOT_FOUND,
63   ///Interleave in header file not supported by this software
     HYPERSPECTRAL_INTERLEAVE_UNSUPPORTED,
65   ///Datatype in hyperspectral image not supported
     HYPERSPECTRAL_DATATYPE_UNSUPPORTED,
67   ///???
     HYPERSPECTRAL_FILE_READING_ERROR
69 };

71 /** Read header information from file.
    * \param filename Filename
73  * \param header Output header container
    * \return HYPERSPECTRAL_NO_ERR on success */
75 hyperspectral_err_t hyperspectral_read_header(const char *filename,
       struct hyspex_header *header);

77 /** Read hyperspectral image from file.
    * \param filename Filename
79  * \param header Header, already read from file using
       hyperspectral_read_header
    * \param image_subset Specified image subset
81  * \param data Output data, preallocated to neccessary size (bands*
       samples*lines, get it from header (or in this case, calculate it
       from the specified subset)). Pixels are accessed by data[SAMPLES*
       BANDS*line_number + SAMPLES*band_number + sample_number].
    * \return HYPERSPECTRAL_NO_ERR on success */
83 hyperspectral_err_t hyperspectral_read_image(const char *filename,
       struct hyspex_header *header, struct image_subset subset, float *
       data);
```

```
85 /** Overloaded version of hyperspectral_read_image where it isn't
       neccessary to supply subset information, the full image will be
       read.
87  * \param filename Filename
    * \param header Header information
89  * \param data Output image data
    * \return HYPERSPECTRAL_NO_ERR on success */
   hyperspectral_err_t hyperspectral_read_image(const char *filename,
       struct hyspex_header *header, float *data);
91
   /** Write header information to file.
93  * \param filename Filename
    * \param bands Number of bands
95  * \param samples Number of samples (across-track)
    * \param lines Number of lines (along-track)
97  * \param wlens Wavelength array */
   void hyperspectral_write_header(const char *filename, int bands, int
       samples, int lines, std::vector<float> wlens);
99
   /** Write hyperspectral image to file.
101 * \param filename Filename
    * \param bands Bands
103 * \param samples Samples
    * \param lines Lines
105 * \param data Image data */
   void hyperspectral_write_image(const char *filename, int bands, int
       samples, int lines, float *data);
107 #endif
```

includes/readimage.h

## A.2 Multispectral images

```
/*
 * Copyright 2016 Stig Viste, Norwegian University of Science and
   Technology
 * Distributed under the MIT License.
 * (See accompanying file LICENSE or copy at
 * http://opensource.org/licenses/MIT
 */

#ifndef MULTISPEC_H_DEFINED
#define MULTISPEC_H_DEFINED

// Read .raw files
void                    multispec_raw(
                    // Number of inputs
                    int argc,
                    // Inputs
                    char *argv[] );

#include "registration.h"
// Create raw itk container
UintImageType::Pointer  rawContainer(
                    // Image width
                    int xsize,
                    // Image height
                    int ysize );

// Create float itk container
ImageType::Pointer      imgContainer(
                    // Image width
                    int xsize,
                    // Image height
                    int ysize );

// Read raw image to itk container
UintImageType::Pointer  readRaw(
                    // Pointer to write to
                    UintImageType* const itkimg,
                    // Image number
                    int i,
                    // Image width
                    int xsize,
                    // Image height
                    int ysize,
                    // File to read from
                    char *argv );

```

```
// Write images from itk container to raw format
void                        writeRaw(
                        // Pointer to read from
                        UintImageType* const itkimg,
                        // Image number
                        int i,
                        // Image width
                        int xsize,
                        // Image height
                        int ysize,
                        // Output name
                        std::string name );
#endif // MULTISPEC_READ_H_DEFINED
```

includes/multispec.h

## A.3 Hyperspectral images

```
/* Copyright 2016 Stig Viste, Norwegian University of Science and
    Technology
 * Distributed under the MIT License.
 * (See accompanying file LICENSE or copy at
 * http://opensource.org/licenses/MIT */
#include <iostream>
#include <sstream>
#include <string>
#include <string.h>
#ifndef HYPERSPEC_H_DEFINED
#define HYPERSPEC_H_DEFINED

// Container for registration parameters
struct reg_params {
  // Registration method
  int regmethod;
  // Registration output name
  std::string reg_name;
  // Output from diff
  int diff_conf;
  // Diff output name
  std::string diff_name;
  // Median filter
  int median;
  // Level of median filtering
  int radius;
  // Gradient filter
  int gradient;
  // Level of gradient filtering
  int sigma;
  // Initial transform angle
  float angle;
  // Initial transform scale
  float scale;
  // Learning rate of registration
  float lrate;
  // Minimum step length before completion
  float slength;
  // Maximum number of iterations
  int niter;
  // Resizing
  unsigned int numberOfLevels;
  // Translation scale
  double translationScale;
  // Initial Translation transform
  int translation;
  // Choose between mutual information and mean squares
  int metric;
  // Option for suppressing iteration outputs
  int output;
};
```

```
52
// Errors
54 enum conf_err_t {
  // Successful
56   CONF_NO_ERR,
  // File not found
58   CONF_FILE_NOT_FOUND,
  // Read error
60   CONF_FILE_READING_ERROR
};
62
// Functions
64 // Read config (params.conf)
conf_err_t params_read( struct reg_params *params);
66
// Retrieve variable from config
68 std::string          getParam(
                            // Variable name
70                          std::string confText,
                            // Variable value
72                          std::string property );

74 // Read a hyperspectral .img file and
// output a registrated .img file
76 void                 hyperspec_img(
                            const char *filename );
78
// Read a hyperspectral .mat file and
80 // output a registrated .mat file
void                 hyperspec_mat(
82                          const char *filename );

84 #include "registration.h"
// Create an image pointer for .img
86 ImageType::Pointer  imageContainer(
                            // Get size of image from .hdr
88                          struct hyspex_header header );

90 // Read an image into an image pointer from .img
ImageType::Pointer  readITK(
92                          // Pointer to write to
                            ImageType* const itkimg,
94                          // Float to read from
                            float *img,
96                          // Image band
                            int i,
98                          // Image storage format
                            struct hyspex_header header );
100
```

```
    // Write an image from an image pointer to a float*
102 float*                writeITK(
                                // Pointer to read from
104                             ImageType* const itkimg,
                                // Float to write to
106                             float *image,
                                // Image band
108                             int i,
                                // Image storage format
110                             struct hyspex_header header );
    // Create an image pointer for .mat
112 ImageType::Pointer  imageMatContainer(
                                // Image width
114                             unsigned xSize,
                                // Image height
116                             unsigned ysize );
    // Read an image into an image pointer from .mat
118 ImageType::Pointer  readMat(
                                // Pointer to write to
120                             ImageType* const itkmat,
                                // Image band
122                             int i,
                                // Image width
124                             unsigned xSize,
                                // Image height
126                             unsigned ySize,
                                // Float to read from
128                             float *hData );
    // Write an image from an image pointer to a float*
130 float*                writeMat(
                                // Pointer to read from
132                             ImageType* const itkmat,
                                // Float to write to
134                             float *hData,
                                // Image band
136                             int i,
                                // Image width
138                             unsigned xSize,
                                // Image height
140                             unsigned ySize );
    // Write float* to .mat
142 void                  outMat(
                                // Output float
144                             float *hData,
                                // Output name
146                             std::string outname,
                                // Wavelengths
148                             matvar_t *wavelengthsd,
                                // Dimensions
150                             matvar_t *HSId );
    #endif // HYPERSPEC_READ_H_DEFINED
```

includes/hyperspec.h

## A.4 Image registration

```
/*
 * Copyright 2016 Stig Viste, Norwegian University of Science and
   Technology
 * Distributed under the MIT License.
 * (See accompanying file LICENSE or copy at
 * http://opensource.org/licenses/MIT
 */

#ifndef REGISTRATION_H_DEFINED
#define REGISTRATION_H_DEFINED

// Insight Toolkit
#include "itkImage.h"

// Image registration
#include "itkImageRegistrationMethodv4.h"
#include "itkMattesMutualInformationImageToImageMetricv4.h"
#include "itkMeanSquaresImageToImageMetricv4.h"
#include "itkRegularStepGradientDescentOptimizerv4.h"

// Deformable image registration
#include "itkBSplineTransform.h"
#include "itkBSplineTransformParametersAdaptor.h"
#include "itkCorrelationImageToImageMetricv4.h"
#include "itkLBFGSOptimizerv4.h"
#include "itkMemoryProbesCollectorBase.h"
#include "itkTimeProbesCollectorBase.h"

// Demons
#include "itkImageRegionIterator.h"
#include "itkDemonsRegistrationFilter.h"
#include "itkHistogramMatchingImageFilter.h"
#include "itkWarpImageFilter.h"

// Filtering
#include "itkBinaryThresholdImageFilter.h"
#include "itkMedianImageFilter.h"
#include "itkGradientMagnitudeRecursiveGaussianImageFilter.h"

// Transform
#include "itkAffineTransform.h"
#include "itkBSplineTransformInitializer.h"
#include "itkCenteredSimilarity2DTransform.h"
#include "itkCenteredRigid2DTransform.h"
#include "itkCenteredTransformInitializer.h"
#include "itkCompositeTransform.h"
#include "itkIdentityTransform.h"
#include "itkTransformToDisplacementFieldFilter.h"
#include "itkTranslationTransform.h"

```

```
// Image I/O
#include "itkCastImageFilter.h"
#include "itkImageFileWriter.h"
#include "itkImageMaskSpatialObject.h"
#include "itkResampleImageFilter.h"

// Image operations
#include "itkRescaleIntensityImageFilter.h"
#include "itkSquaredDifferenceImageFilter.h"
#include "itkSubtractImageFilter.h"

// Introduce a class that will keep track of the iterations
#include "itkCommand.h"
class CommandIterationUpdate : public itk::Command {
public:
  typedef  CommandIterationUpdate    Self;
  typedef  itk::Command              Superclass;
  typedef  itk::SmartPointer<Self>   Pointer;
  itkNewMacro( Self );

protected:
  CommandIterationUpdate() {};

public:
  typedef itk::RegularStepGradientDescentOptimizerv4<double>
    OptimizerType;
  typedef const OptimizerType *
    OptimizerPointer;

  void Execute(itk::Object *caller, const itk::EventObject & event )
    ITK_OVERRIDE;
  void Execute(const itk::Object * object, const itk::EventObject &
    event ) ITK_OVERRIDE;
};

// Instantiation of input images
const    unsigned int     Dimension = 2;
typedef float             PixelType;
typedef unsigned char     CharPixelType;
typedef unsigned short    UintPixelType;

typedef itk::Image< PixelType, Dimension >                    ImageType;

typedef itk::Image< UintPixelType, Dimension >
    UintImageType;

```

```
// Filters
typedef itk::MedianImageFilter<
                              ImageType,
                              ImageType >
    MedianFilterType;
typedef itk::GradientMagnitudeRecursiveGaussianImageFilter<
                              ImageType,
                              ImageType >
    GradientFilterType;
typedef itk::ShrinkImageFilter<
                              ImageType,
                              ImageType >
    ShrinkFilterType;

// Registration
typedef itk::RegularStepGradientDescentOptimizerv4<
                              double>
    OptimizerType;
typedef itk::LBFGSOptimizerv4
    OptimizerBSplineType;
typedef itk::MeanSquaresImageToImageMetricv4<
                              ImageType,
                              ImageType >                    MetricType
    ;

// Initialization of transform types

// Translation
typedef itk::RegularStepGradientDescentOptimizerv4<
                              double >
    TOptimizerType;
typedef itk::TranslationTransform<
                              double,
                              Dimension >
    TTransformType;
typedef itk::MattesMutualInformationImageToImageMetricv4<
                              ImageType,
                              ImageType >
    TMetricType;
typedef itk::ImageRegistrationMethodv4<
                              ImageType,
                              ImageType,
                              TTransformType >
    TRegistrationType;
typedef  OptimizerType::ParametersType
    TParametersType;
typedef itk::CompositeTransform<
                              double,
                              Dimension >
    CompositeTransformType;
```

```
129 // Rigid
    typedef itk::CenteredRigid2DTransform<
131                                 double >
        TransformRigidType;
    typedef itk::CenteredTransformInitializer<
133                                 TransformRigidType,
                                    ImageType,
135                                 ImageType >
        TransformRigidInitializerType;
    typedef itk::ImageRegistrationMethodv4<
137                                 ImageType,
                                    ImageType,
139                                 TransformRigidType >
        RegistrationRigidType;

141 // Similarity
    typedef itk::CenteredSimilarity2DTransform<
143                                 double >
        TransformSimilarityType;
    typedef itk::CenteredTransformInitializer<
145                                 TransformSimilarityType,
                                    ImageType,
147                                 ImageType >
        TransformSimilarityInitializerType;
    typedef itk::ImageRegistrationMethodv4<
149                                 ImageType,
                                    ImageType,
151                                 TransformSimilarityType >
        RegistrationSimilarityType;

153 // Affine
    typedef itk::AffineTransform<
155                                 double,
                                    Dimension >
        TransformAffineType;
157 typedef itk::CenteredTransformInitializer<
                                    TransformAffineType,
159                                 ImageType,
                                    ImageType >
        TransformAffineInitializerType;
161 typedef itk::ImageRegistrationMethodv4<
                                    ImageType,
163                                 ImageType,
                                    TransformAffineType >
        RegistrationAffineType;
165
```

```
// BSpline
const unsigned int SplineOrder = 3;
typedef double CoordinateRepType;
typedef itk::BSplineTransform<
                            CoordinateRepType,
                            Dimension,
                            SplineOrder >          TransformBSplineType;
typedef itk::ImageRegistrationMethodv4<
                            ImageType,
                            ImageType >
    RegistrationBSplineType;
typedef itk::BSplineTransformInitializer<
                            TransformBSplineType,
                            ImageType >
    InitializerBSplineType;
typedef TransformBSplineType::ParametersType      ParametersBSplineType
    ;
typedef itk::BSplineTransformParametersAdaptor<
                            TransformBSplineType >
    BSplineAdaptorType;
typedef itk::RegistrationParameterScalesFromPhysicalShift<
                            MetricType >           ScalesEstimatorType;

// Demons
typedef itk::DemonsRegistrationFilter<
                            ImageType,
                            ImageType,
                            DisplacementFieldType >
    DemonsFilterType;
typedef itk::HistogramMatchingImageFilter<
                            ImageType,
                            ImageType >
    MatchingFilterType;
typedef itk::WarpImageFilter<
                          ImageType,
                          ImageType,
                          DisplacementFieldType >              WarperType
    ;
typedef itk::LinearInterpolateImageFunction<
                          ImageType,
                          double >
    LinInterpolatorType;
```

```
201 // Image casting, because registrations only supports float
    typedef itk::CastImageFilter<
203                             UintImageType,
                                ImageType >         CastFilterFloatType;
205 typedef itk::CastImageFilter<
                                ImageType,
207                             UintImageType >     CastFilterUintType;
    typedef itk::RescaleIntensityImageFilter<
209                             UintImageType,
                                UintImageType >     RescalerUintType;
211 typedef itk::RescaleIntensityImageFilter<
                                ImageType,
213                             ImageType >         RescalerFloatType;

215 // Set up outputs and writers
    typedef itk::SubtractImageFilter<
217                             ImageType,
                                ImageType,
219                             ImageType >     DifferenceFilterType;
    typedef itk::ResampleImageFilter<
221                             ImageType,
                                ImageType >     ResampleFilterType;
223 typedef itk::ImageFileWriter<
                                ImageType >                     WriterType
        ;
225 typedef itk::ImageFileWriter<
                                UintImageType >
        UintWriterType;
227
    // Set up optimizer
229 typedef OptimizerType::ScalesType           OptimizerScalesType;

231 // Generic handlers, float
    RegistrationRigidType::Pointer registrationRigidContainer(
233                             ImageType* const fixed,
                                ImageType* const moving,
235                             OptimizerType::Pointer optimizer );
    RegistrationSimilarityType::Pointer registrationSimilarityContainer(
237                             ImageType* const fixed,
                                ImageType* const moving,
239                             OptimizerType::Pointer optimizer );
    RegistrationAffineType::Pointer registrationAffineContainer(
241                             ImageType* const fixed,
                                ImageType* const moving,
243                             OptimizerType::Pointer optimizer );
    TransformRigidInitializerType::Pointer initializerRigidContainer(
245                             ImageType* const fixed,
                                ImageType* const moving,
247                             TransformRigidType::Pointer transform );
```

```
    TransformSimilarityInitializerType::Pointer
        initializerSimilarityContainer(
249                             ImageType* const fixed,
                                ImageType* const moving,
251                             TransformSimilarityType::Pointer transform
        );
    TransformAffineInitializerType::Pointer initializerAffineContainer(
253                             ImageType* const fixed,
                                ImageType* const moving,
255                             TransformAffineType::Pointer transform );
    ResampleFilterType::Pointer resampleRigidPointer(
257                             ImageType* const fixed,
                                ImageType* const moving,
259                             TransformRigidType::Pointer transform );
    ResampleFilterType::Pointer resampleSimilarityPointer(
261                             ImageType* const fixed,
                                ImageType* const moving,
263                             TransformSimilarityType::Pointer transform
        );
    ResampleFilterType::Pointer resampleAffinePointer(
265                             ImageType* const fixed,
                                ImageType* const moving,
267                             TransformAffineType::Pointer transform );
    ResampleFilterType::Pointer resampleBSplinePointer(
269                             ImageType* const fixed,
                                ImageType* const moving,
271                             TransformBSplineType::Pointer transform );
    DifferenceFilterType::Pointer diffFilter(
273                             ImageType* const moving,
                                ResampleFilterType::Pointer resample );
275
    // Image filtering
277 ImageType::Pointer            gradientFilter(
                                ImageType* const fixed,
279                             int sigma );
    ImageType::Pointer            medianFilter(
281                             ImageType* const fixed,
                                int radius );
283
```

```
// Image I/O
CastFilterFloatType::Pointer  castFloatImage(
                              UintImageType* const img );
CastFilterUintType::Pointer   castUintImage(
                              ImageType* const img );

// Printing parameters
void finalRigidParameters( TransformRigidType::Pointer transform,
                        OptimizerType::Pointer optimizer);
void finalSimilarityParameters( TransformSimilarityType::Pointer
    transform,
                        OptimizerType::Pointer optimizer);
void finalAffineParameters( TransformAffineType::Pointer transform,
                        OptimizerType::Pointer optimizer);

// Image registrations
#include "hyperspec.h"
TransformRigidType::Pointer registration1(
                              ImageType* const fixed,
                              ImageType* const moving,
                              reg_params params );
TransformSimilarityType::Pointer registration2(
                              ImageType* const fixed,
                              ImageType* const moving,
                              reg_params params );
TransformAffineType::Pointer registration3(
                              ImageType* const fixed,
                              ImageType* const moving,
                              reg_params params );
TransformBSplineType::Pointer registration4(
                              ImageType* const fixed,
                              ImageType* const moving,
                              reg_params params );
CompositeTransformType::Pointer translation(
                              ImageType* const fixed,
                              ImageType* const moving,
                              reg_params params );

#endif // REGISTRATION_H_DEFINED
```

includes/registration.h

## A.5 Source code

Part of the source code is included here, and illustrates a typically pipeline structure in ITK

```
int main(int argc, char *argv[]){

  if (argc < 2) {
    cerr << "Usage: " << argv[0] << " hyperspectral_image_path" <<
    endl;
    exit(1);
  }

  char *filename = argv[1];

  // File format recognition and run correct function
  if (strstr(filename, "raw") ){
    // File is .raw, see src/multispec.cpp
    multispec_raw( argc, argv );
  } else if (strstr(filename, "img") ){
    // File is .img, see src/hyperspec.cpp
    hyperspec_img(filename);
  } else if (strstr(filename, "mat") ){
    // File is .mat, see src/hyperspec.cpp
    hyperspec_mat(filename);
  } else {
    // Unknown format
    cerr << "Currently supported file formats: img, mat, raw" << endl;
    exit(1);
  }
}
```

Listing 1: main()

```
    struct reg_params params;
    conf_err_t reg_errcode = params_read( &params );
```

Listing 2: Read configuration

```
  struct hyspex_header header;
  hyperspectral_err_t hyp_errcode
    = hyperspectral_read_header(filename, &header);

  // Read hyperspectral image
  float *img  = new float[header.samples*header.lines*header.bands]();
  hyp_errcode = hyperspectral_read_image(filename, &header, img);
```

Listing 3: Read .img to float

```
// Read mat pointer
mat_t *matfp;

// Open file
matfp = Mat_Open(filename,MAT_ACC_RDONLY);
if ( NULL == matfp ) {
  fprintf(stderr,"Error opening MAT file %s\n",filename);
  exit(1);
}

// Read mat matrix
matvar_t *HSId = Mat_VarRead(matfp, "HSI");
// Read to float array
float *hData = static_cast<float*>(HSId->data);
```

Listing 4: Read .mat to float

```
hyperspectral_write_header( params.reg_name.c_str(), header.bands,
  header.samples, header.lines, header.wlens );
hyperspectral_write_image( params.reg_name.c_str(), header.bands,
  header.samples, header.lines, out );
```

Listing 5: Write .img to file

```
// Prepare
size_t dim3d[3] = { HSId->dims[0], HSId->dims[1], HSId->dims[2] };
outname += ".mat";

mat_t *matout;

// Write
matout = Mat_CreateVer(outname.c_str(),NULL,MAT_FT_MAT5);
Mat_VarWrite( matout, wavelengthsd, MAT_COMPRESSION_ZLIB );

matvar_t *HSIout = Mat_VarCreate("HSI", MAT_C_SINGLE, MAT_T_SINGLE,
  HSId->rank, dim3d, static_cast<void*>(hData), 0);
Mat_VarWrite( matout, HSIout, MAT_COMPRESSION_ZLIB );

Mat_VarFree(wavelengthsd);
Mat_VarFree(HSIout);
Mat_VarFree(HSId);
Mat_Close(matout);
```

Listing 6: Write .mat to file

```
    OptimizerType::Pointer      optimizer     = OptimizerType::New();
    MetricType::Pointer         metric        = MetricType::New();
    RegistrationType::Pointer   registration  = RegistrationType::New()
    ;
    TransformType::Pointer      transform     = TransformType::New();
    TransformInitializerType::Pointer
                        initializer   = TransformInitializerType::New();

    registration->SetMetric(        metric     );
    registration->SetOptimizer(    optimizer );
    registration->SetFixedImage(     fixed    );
    registration->SetMovingImage(    moving   );

    initializer->SetTransform(    transform   );
    initializer->SetFixedImage(    fixed      );
    initializer->SetMovingImage(   moving     );

    // Select center of mass mode
    initializer->MomentsOn();

    // Compute the center and translation
    initializer->InitializeTransform();

    transform->SetAngle( params.angle );
    registration->SetInitialTransform( transform );
    registration->InPlaceOn();
```

Listing 7: Initialize registration

```
OptimizerScalesType optimizerScales( transform->
  GetNumberOfParameters() );

optimizerScales[0] = 1.0;
optimizerScales[1] = params.translationScale;
optimizerScales[2] = params.translationScale;
optimizerScales[3] = params.translationScale;
optimizerScales[4] = params.translationScale;

optimizer->SetScales(        optimizerScales      );
optimizer->SetLearningRate(      params.lrate     );
optimizer->SetMinimumStepLength( params.slength );
optimizer->SetNumberOfIterations( params.niter   );

// Create the command observer and register it with the optimizer
CommandIterationUpdate::Pointer observer = CommandIterationUpdate::
  New();
optimizer->AddObserver( itk::IterationEvent(), observer );

// Optional: Shrinking and/or smoothing, set to 0
RegistrationRigidType::ShrinkFactorsArrayType shrinkFactorsPerLevel;
shrinkFactorsPerLevel.SetSize( 1 );
shrinkFactorsPerLevel[0] = 1;

RegistrationRigidType::SmoothingSigmasArrayType
  smoothingSigmasPerLevel;
smoothingSigmasPerLevel.SetSize ( 1 );
smoothingSigmasPerLevel[0] = 0;

registration->SetNumberOfLevels(        params.numberOfLevels         );
registration->SetSmoothingSigmasPerLevel( smoothingSigmasPerLevel );
registration->SetShrinkFactorsPerLevel(    shrinkFactorsPerLevel    );
```

Listing 8: Passing parameters to the optimizer

```
void CommandIterationUpdate::Execute(itk::Object *caller, const itk::
    EventObject & event){
  Execute( (const itk::Object *)caller, event);
}

void CommandIterationUpdate::Execute(const itk::Object * object, const
     itk::EventObject & event){
  OptimizerPointer optimizer = static_cast< OptimizerPointer >( object
    );
  if( ! itk::IterationEvent().CheckEvent( &event ) ){
    return;
  }
  std::cout << optimizer->GetCurrentIteration() << "   ";
  std::cout << optimizer->GetValue() << "   ";
  std::cout << optimizer->GetCurrentPosition() << std::endl;
  }
}
```

Listing 9: CommandIterationUpdate

```
  CastFilterFloatType::Pointer castFilter = CastFilterFloatType::New()
    ;
  castFilter->SetInput( image );
  castFilter->Update();
```

Listing 10: Cast `unsigned short` to `float`

```
try {
    registration->Update();
    cout     << "Optimizer stop condition: "
             << registration->GetOptimizer()->
   GetStopConditionDescription()
             << endl;
}
catch( itk::ExceptionObject & err ){
    cerr << "ExceptionObject caught !" << endl;
    cerr << err << endl;
    exit(1);
}
```

Listing 11: Registration update

```
  TransformType::ParametersType
                          finalParameters = transform->GetParameters();

  const double finalAngle                 = finalParameters[0];
  const double finalRotationCenterX       = finalParameters[1];
  const double finalRotationCenterY       = finalParameters[2];
  const double finalTranslationX          = finalParameters[3];
  const double finalTranslationY          = finalParameters[4];

  const unsigned int  numberOfIterations = optimizer->
    GetCurrentIteration();
  const double        bestValue          = optimizer->GetValue();
```

Listing 12: Get parameters from transform and optimizer

```
  ResampleFilterType::Pointer  resample = ResampleFilterType::New();
  resample->SetTransform(                transform                );
  resample->SetInput(                      moving                 );
  resample->SetSize( fixed->GetLargestPossibleRegion().GetSize()   );
  resample->SetOutputOrigin(         fixed->GetOrigin()            );
  resample->SetOutputSpacing(        fixed->GetSpacing()           );
  resample->SetOutputDirection(      fixed->GetDirection()         );
  resample->SetDefaultPixelValue(              0.0                 );
  resample->Update();
```

Listing 13: Apply transform to moving image

```
  // Open images
  FILE *fid = fopen(argv, "rb");
  unsigned short *in_data = new unsigned short[xsize*ysize]();
  int read_bytes = fread(in_data, sizeof(uint16_t), xsize*ysize, fid);

  // One pixel at a time
  for ( int j=0; j<ysize; j++){
    for (int k=0; k<xsize; k++){
      UintImageType::IndexType pixelIndex;
      pixelIndex[0] = k;
      pixelIndex[1] = j;
      itkraw->SetPixel(pixelIndex, in_data[xsize*j + k]);
    }
  }

  fclose( fid );
  delete [] in_data;
  return itkraw;
}
```

Listing 14: Read raw files

```
  const TransformAffineType::ParametersType
                    finalParameters = transform->GetParameters();

  const double finalRotationCenterX = transform->GetCenter()[0];
  const double finalRotationCenterY = transform->GetCenter()[1];
  const double finalTranslationX    = finalParameters[4];
  const double finalTranslationY    = finalParameters[5];

  const unsigned int numberOfIterations = optimizer->
    GetCurrentIteration();
  const double bestValue = optimizer->GetValue();

  // Compute rotation angle and scaling
  vnl_matrix<double> p(2, 2);
  p[0][0] = (double) finalParameters[0];
  p[0][1] = (double) finalParameters[1];
  p[1][0] = (double) finalParameters[2];
  p[1][1] = (double) finalParameters[3];
  vnl_svd<double> svd(p);
  vnl_matrix<double> r(2, 2);
  r = svd.U() * vnl_transpose(svd.V());
  double angle = std::asin(r[1][0]);
```

Listing 15: Parameters for `itk::AffineTransform`

```
unsigned int numberOfGridNodesInOneDimension = 8;
TransformBSplineType::MeshSizeType                    meshSize;
meshSize.Fill( numberOfGridNodesInOneDimension - SplineOrder );
transformInitializer->SetTransformDomainMeshSize( meshSize  );

// Scale estimator
ScalesEstimatorType::Pointer scalesEstimator = ScalesEstimatorType::
  New();
scalesEstimator->SetMetric( metric );
scalesEstimator->SetTransformForward( true );
scalesEstimator->SetSmallParameterVariation( 1.0 );

// Set Optimizer
optimizer->SetGradientConvergenceTolerance( params.slength );
optimizer->SetLineSearchAccuracy( 1.2 );
optimizer->SetDefaultStepLength( 1.5 );
optimizer->TraceOn();
optimizer->SetMaximumNumberOfFunctionEvaluations( params.niter );
optimizer->SetScalesEstimator( scalesEstimator );

// Add time and memory probes
itk::TimeProbesCollectorBase      chronometer;
itk::MemoryProbesCollectorBase    memorymeter;

cout << "Starting Registration"       << endl;

try
  {
  memorymeter.Start( "Registration" );
  chronometer.Start( "Registration" );

  registration->Update();

  chronometer.Stop( "Registration" );
  memorymeter.Stop( "Registration" );
}
```

Listing 16: Additional parameters for `itk::BSplineTransform`

```
// Report the time and memory taken by the registration
chronometer.Report( cout );
memorymeter.Report( cout );

OptimizerType::ParametersType finalParameters = transform->
  GetParameters();

cout << "Last Transform Parameters"   << endl;
cout << finalParameters               << endl;
```

Listing 17: Retrieving parameters for `itk::BSplineTransform`

```
  TransformSimilarityType::ParametersType finalParameters =
  transform->GetParameters();

  const double finalScale           = finalParameters[0];
  const double finalAngle           = finalParameters[1];
  const double finalRotationCenterX = finalParameters[2];
  const double finalRotationCenterY = finalParameters[3];
  const double finalTranslationX    = finalParameters[4];
  const double finalTranslationY    = finalParameters[5];

  const unsigned int  numberOfIterations = optimizer->
  GetCurrentIteration();
  const double        bestValue          = optimizer->GetValue();

  // Print results
  const double finalAngleInDegrees = finalAngle * 180.0 / itk::Math
  ::pi;

  std::cout << "Result ="          << std::endl;
  std::cout << "Scale         = " << finalScale           << std::
  endl;
  std::cout << "Angle (radians) " << finalAngle  << std::endl;
  std::cout << "Angle (degrees) " << finalAngleInDegrees  << std::
  endl;
  std::cout << "Center X      = " << finalRotationCenterX << std::
  endl;
  std::cout << "Center Y      = " << finalRotationCenterY << std::
  endl;
  std::cout << "Translation X = " << finalTranslationX    << std::
  endl;
  std::cout << "Translation Y = " << finalTranslationY    << std::
  endl;
  std::cout << "Iterations    = " << numberOfIterations   << std::
  endl;
  std::cout << "Metric value  = " << bestValue            << std::
  endl;
```

Listing 18: Retrieving parameters for `itk::CenteredSimilarity2DTransform`

```
    if ( params.metric == 1 ){
      TMetricType::Pointer                    transMetric        =
                                              TMetricType::New();
      transRegistration->SetMetric(           transMetric        );
      transMetric->SetNumberOfHistogramBins( 24 );
    } else {
      MetricType::Pointer                     transMetric        =
                                              MetricType::New();
      transRegistration->SetMetric(           transMetric        );
    }
```

Listing 19: Choose metric for translation transform

```
    transOptimizer->SetRelaxationFactor( 0.1 );
```

Listing 20: Relaxation factor passed to optimizer

```
// Matcher
MatchingFilterType::Pointer matcher = MatchingFilterType::New();
matcher->SetInput( moving );
matcher->SetReferenceImage( fixed );
matcher->SetNumberOfHistogramLevels( 2048 );
matcher->SetNumberOfMatchPoints( 9 );
matcher->ThresholdAtMeanIntensityOn();

// Filter
DemonsFilterType::Pointer filter = DemonsFilterType::New();
CommandIterationUpdate2::Pointer observer = CommandIterationUpdate2::
    New();
filter->AddObserver( itk::IterationEvent(), observer );
filter->SetFixedImage( fixed );
filter->SetMovingImage( matcher->GetOutput() );
filter->SetNumberOfIterations( params.niter );
filter->SetStandardDeviations( 1.0 );
filter->Update();

// Warper
WarperType::Pointer warper = WarperType::New();
LinInterpolatorType::Pointer interpolator = LinInterpolatorType::New()
    ;

warper->SetInput( moving );
warper->SetInterpolator( interpolator );
warper->SetOutputSpacing( fixed->GetSpacing() );
warper->SetOutputOrigin( fixed->GetOrigin() );
warper->SetOutputDirection( fixed->GetDirection() );
warper->SetDisplacementField( filter->GetOutput() );
```

Listing 21: Demons transform

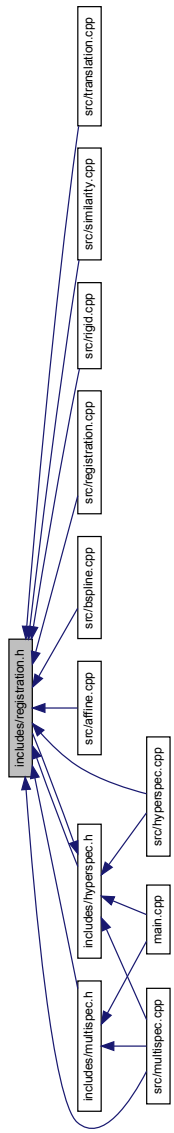
```
template <typename TRegistration>
class RegistrationInterfaceCommand : public itk::Command{
public:
    typedef  RegistrationInterfaceCommand   Self;
    typedef  itk::Command                   Superclass;
    typedef  itk::SmartPointer<Self>        Pointer;
    itkNewMacro( Self );

protected:
    RegistrationInterfaceCommand() {};

public:
    typedef   TRegistration                 RegistrationType;

    void Execute( itk::Object * object, const itk::EventObject & event
    ) ITK_OVERRIDE{
        Execute( (const itk::Object *) object , event );
    }

    void Execute(const itk::Object * object, const itk::EventObject &
    event) ITK_OVERRIDE{
        if( !(itk::MultiResolutionIterationEvent().CheckEvent( &event
    ))){
            return;
        }

        cout << "\nObserving from class " << object->GetNameOfClass();
        if (!object->GetObjectName().empty()){
            cout << " \"" << object->GetObjectName() << "\"" << endl;
        }

        const RegistrationType * registration = static_cast<const
    RegistrationType *>( object );
        unsigned int currentLevel = registration->GetCurrentLevel();
        typename RegistrationType::
    ShrinkFactorsPerDimensionContainerType shrinkFactors =
                                                registration->
    GetShrinkFactorsPerDimension( currentLevel );
        typename RegistrationType::SmoothingSigmasArrayType
    smoothingSigmas =

    registration->GetSmoothingSigmasPerLevel();

        cout << "-------------------------------------" << endl;
        cout << " Current multi-resolution level = " << currentLevel
    << endl;
        cout << "    shrink factor = " << shrinkFactors << endl;
        cout << "    smoothing sigma = " << smoothingSigmas[
    currentLevel] << endl;
        cout << endl;
    }
};
```

Listing 22: Iteration tracking for translation transform

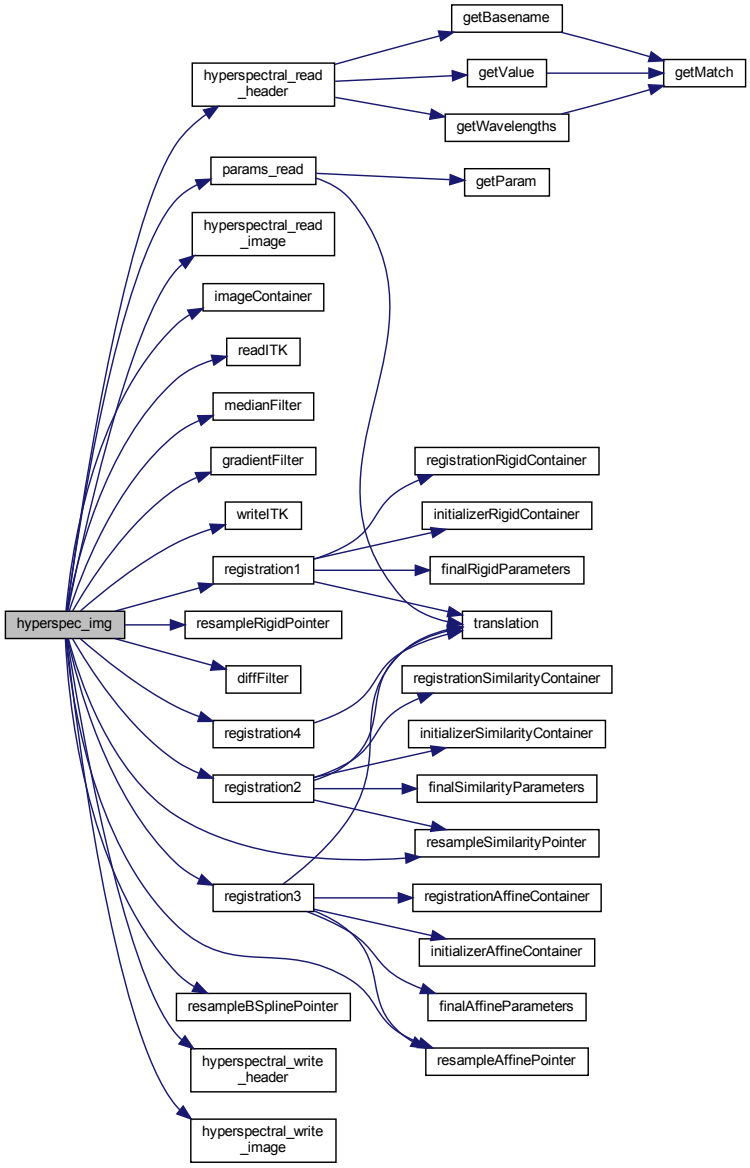
# B Inheritance diagrams

The diagrams was generated using Doxygen[24].

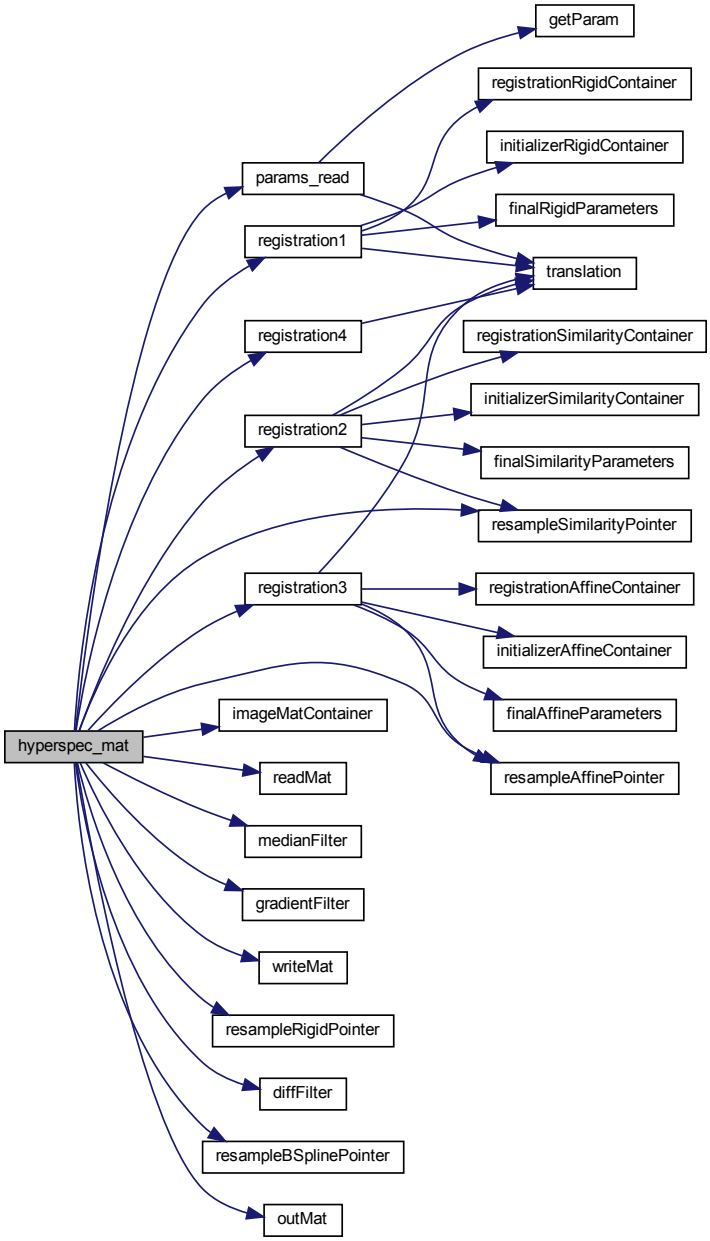
## B.1 Includes



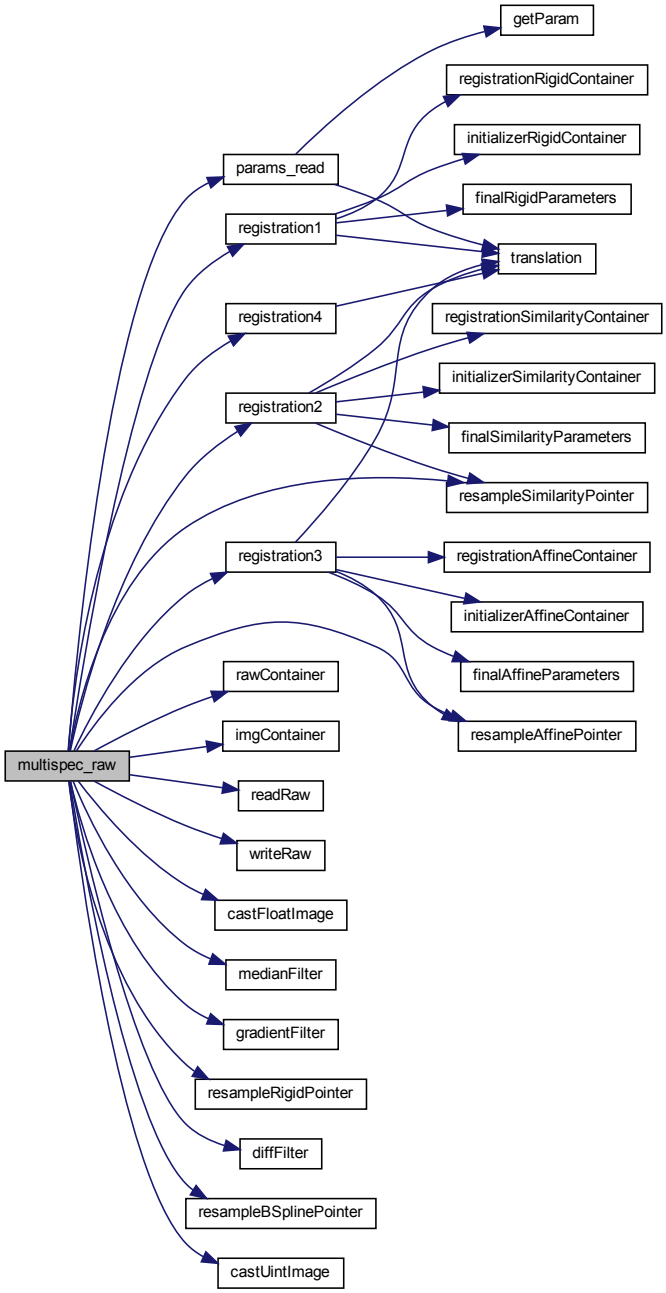
## B.2 hyperspec_img functions



## B.3 hyperspec_mat functions



## B.4 multispec_raw functions



# C Hardware

| | |
|---|---|
| **Laptop:** | Asus Zenbook UX302LG |
| Memory:<br>CPU: | 8GB DDR3L 1300MHz<br>Intel i7-4500U 1.8GHz dual core |
| **Server:** | Unbranded |
| Memory:<br>CPU: | 6GB DDR3 2133MHz<br>Intel i7-930 2.8GHz quad core |
| **Virtual Machine:** | Xenserver 7 |
| Memory:<br>CPU: | 32GB DDR2 Sun Fire X4600 M2<br>AMD Opteron 8356 2.3GHz 30 core |

Table C.1: Hardware

# D Software

| | |
|---|---|
| **Arch Linux:** | Unix based operating system, 64-bits.[5] |
| **CMake:** | Open source, cross platform, build process manager.[20] |
| **GCC:** | Open source compiler system.[32] |
| **Doxygen:** | Tool for generation of documentation[24] |
| **Hyview:** | Hyperspectral image viewer, available from NTNU IET.[44] |
| **ImageJ:** | Java-based image processing application.[46] |
| **ImageMagick:** | Image manipulation and conversion program[47] |
| **Matlab:** | Programming language developed by Mathworks[66] |
| **Vim:** | Vi IMproved, text editor for Unix systems.[104] |
| **yEd:** | Open source, cross platform, graph editor.[107] |

Table D.1: Software