Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for naturvitenskap
Institutt for fysikk

Gjert Magne Kahrs Knutsen

Parallelization of Coherent Point Drift for patient registration

Masteroppgave i fysikk og matematikk
Veileder: Erik Smistad
Juni 2019

NTNU
Kunnskap for en bedre verden

SINTEF

Parallelization of Coherent Point Drift for patient registration

A master's thesis in cooperation with SINTEF

Gjert Magne Kahrs Knutsen

Trondheim, June 2019

Supervisor NTNU: Pål Erik Goa
Supervisor SINTEF: Erik Smistad

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

NTNU

SINTEF Digital
Department of Health Research
Medical Technology

SINTEF

Abstract

Point set registration is a central part in any application where the correspondence between two data point sets is of interest, for instance patient data from medical examinations. There exists numerous different algorithms that aim at solving the registration problem, and one of which is the Coherent Point Drift (CPD) algorithm. In this thesis a fast implementation of this algorithm is developed with parallelization on both CPU and GPU, significantly reducing the time required to perform the registrations. The developed algorithm, called FAST CPD (FCPD), includes rigid, affine and elastic registration and the code will be freely available as open-source.

The rigid FCPD runtimes allow for real-time registration with the CPD algorithm, and the elastic registration runtimes make it possible to register relatively large point sets within a few seconds. The elastic FCPD is applied to data used to correct for the brain-shift occurring during brain surgeries.

Sammendrag

Punktskyregistrering er en sentral del i enhver anvendelse der det er ønskelig å vite hvordan to sett av punkter hører sammen, for eksempel data fra medisinske undersøkelser. Det finnes et stort utvalg av ulike algoritmer for å løse slike registreringsproblem, og en av dem er Coherent Point Drift (CPD). I denne oppgaven blir en rask implementasjon av denne algoritmen utviklet ved hjelp av parallellisering av beregninger både på CPU og GPU, som reduserer kjøretiden betydelig. Den utviklede algoritmen, kalt FAST CPD (FCPD), inkluderer rigid, affin og elastisk registrering og koden blir fritt tilgjengelig som åpen kildekode.

FCPD gir kjøretider for rigid registrering som gjør det mulig å gjøre slik registrering i sanntid med CPD, og elastisk registrering av relativt store datasett kan bli gjort i løpet av få sekunder. Elastisk FCPD ble testet på data for korrigering av forskyningen av hjernen som kan oppstå under hjerneoperasjoner (brain-shift correction).

Preface

This is a master's thesis in the field of medical technology, written as a part of my Master of Science in Physics and Mathematics at the Norwegian University of Science and Technology (NTNU), Trondheim. My specialization is in technical physics, topped with computer and medical imaging courses, at the Department of Physics (IFY).

The work with this thesis has been done with the group of medical technology at SINTEF Digital, Trondheim. This master's thesis is a continuation of the specialization project from the fall of 2018, also in cooperation with the same group at SINTEF.

I would like to start by thanking all the scientists and employees of the medical technology group at SINTEF for letting me work with you on my master's thesis. Thank you for inspiring me with your expertise, and for your humble and helpful appearances. In particular, I want to thank my supervisor Erik Smistad for your support and valuable feedback.

I would also like to thank my family and my friends for your unconditional support and motivation. And, to my beloved fiancée and wife-to-be Karina, thank you for always being there for me. You make me happy, and I am thankful for having you in my life.

Trondheim, June 7th, 2019
Gjert Magne Kahrs Knutsen

Contents

List of Figures

Abbreviations

CPD	Coherent Point Drift
EM	Expectation Maximization
FAST	FrAmework for heterogeneouS medical image compuTing and visualization
FCPD	FAST Coherent Point Drift
FGT	Fast Gauss Transform
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
MGSR	Modified Gram-Schmidt with Reorthogonalization
PSR	Point Set Registration
RKHS	Reproducing kernel Hilbert space
SRFT	Subsampled random Fourier transform

Mathematical Notation

$\mathbf{1}$	Column vector of ones.
$\boldsymbol{a}$	Vector in column form. Also used to represent points.
$\boldsymbol{a}^T$	Transpose of vector.
$\boldsymbol{A}_{M\times N}$	Matrix with M rows and N columns.
$\boldsymbol{A}(m,\cdot)$	Row m of matrix $\boldsymbol{A}$
$\boldsymbol{A}(\cdot,n)$	Column n of matrix $\boldsymbol{A}$
a_{ij}	Element (i,j) of matrix $\boldsymbol{A}$.
$\mathrm{d}(\boldsymbol{a})$	Diagonal matrix with the vector $\boldsymbol{a}$ on the diagonal.
$\boldsymbol{I}$	Identity matrix.
$\mathcal{X}$	Set of points, $\mathcal{X} = \{\boldsymbol{x_1}, \dots\}$.

1 Introduction

Registration is the process of finding the transformation between one or two data sets. These data sets can for instance be images or points or surfaces. With the help of registration, data sets can be combined and compared in a simple manner. Medical imaging is one of the numerous fields registration has proven useful, where patient data may be acquired with several different techniques and at different times. Applying registration on patient data can be a useful tool in the process of diagnosis, treatment and follow-up of the patient.

Registration methods are usually classified by the type of transformations they can describe and which type of data they can process. The focus in this thesis will be point set registration (PSR), limiting the data sets studied to point sets. In some cases the correspondences between the points can be collectively accounted for by a linear transformation, for instance a rotation. Restricting the transformation to a rigid or affine transformation, yield *rigid* and *affine* registration, respectively. In other cases each point must be described individually, for instance when a point set is deformed. A registration method able to account for this is called *elastic* or *deformable*. In some works the term non-rigid is imprecisely used about elastic methods even though it strictly speaking includes affine registrations as well.

A popular point set registration algorithm is Coherent Point Drift (CPD), developed by Andriy Myronenko and Xubo Song [2]. CPD includes methods for all the aforementioned registration types, rigid, affine and elastic, and it performs well even when the data sets are impaired with noise, outliers and missing points. The time required to run the computations in CPD, especially for larger point sets, can however, be limiting for some applications. It is therefore always of interest to reduce this runtime if possible. A couple of measures to accomplish this is already included in [2], yet parallelization of the CPD code was not covered.

Parallelization is to perform computations simultaneously, i.e. in parallel, which can reduce the overall runtime dramatically. This can be done on both the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU) on a computer, where especially GPUs can improve the computation time.

1.1 Project objectives

The purpose of this thesis was to develop and document an open-source optimized parallel version of the CPD algorithm, with the end goal of improving the CPD runtime without losing accuracy. The implementation was tested on both synthetic and real medical data, and compared with an ICP implementation and the C++ CPD implementation without GPU parallelization provided by Peter J. Gadomski [1]. The real data studied are blood vessel centerlines.

The parallel CPD implementation developed is included in the framework FAST [3], and will thus be referred to as FAST CPD (FCPD). The FAST CPD source code is available here: `https://github.com/smistad/FAST/tree/master/source/FAST/Algorithms/CoherentPointDrift`.

1.2 Related work

There exist plenty different PSR algorithms, both for rigid and non-rigid registration. A more extensive presentation of some of these will be presented in a later chapter, but for now the most relevant works for comparison with the parallel CPD are covered. Before CPD was published in 2009, the Thin plate spline Robust point matching (TPS-RPM) algorithm by Chui et. al. [4] from 2000, building upon the robust point matching (RPM) algorithm by Gold et. al. [5] from 1995, was the state-of-art elastic registration algorithm. In 2010, Mourning et. al. published an article describing GPU parallelization of the RPM-based algorithms [6].

In 2016, Golyanik and Taetz et. al. published the Extended Coherent Point Drift (ECPD) algorithm, in which they used parallelization on both CPU and GPU for some of the most computationally expensive calculations as one of several measures to reduce the runtime of their method.

The code for ECPD was not provided, so no direct comparison between ECPD and FCPD will be provided here, but the calculations parallelized in FCPD differ from the ones mentioned in the ECPD paper, and thus managing to clearly beat the runtimes reported by Golyanik and Taetz et. al. The rigid registration algorithm Maximum Likelihood Mixture Decoupling algorithm (MLMD) from 2015 by Eckart et. al. is also worth mentioning here. The method is based on much of the same mathematical foundation as CPD and is implemented using GPU computations to achieve state-of-the-art rigid registration results.

A couple of more recent PSR algorithms have shown to be very computationally efficient, providing real-time registration possibilities, at least for rigid registration of reasonable sized point sets. Both GMM-Tree by Eckart et. al. [7] (2018) and FilterReg by Gao and Tedrake (2018) have reported frame rates of between 20 and 30 frames per second for rigid registration of point sets of at least 3500×3 points. Both methods rely on GPU computation. Both of these methods build upon the same mathematical probabilistic foundation as CPD, but have their own algorithms, separating them from the CPD algorithm. Lastly, a novel approach PR-Net by Wang et. al. [8] using neural networks for non-rigid registration was published in April 2019. The registration time on a training sample beats the original CPD implementation runtime with several orders of magnitude (and comparable error) for point sets of size 10000, while the computation time with a pre-trained network is less than 10 seconds.

1.3 Master project and thesis outline

This thesis is a continuation of a specialization project by this writer from the fall of 2018, also together with the medical technology group at SINTEF. In that project rigid and affine CPD was implemented with CPU parallelization, and it was studied whether the algorithm could be used for patient registration. The runtimes achieved were significantly better than for the non-parallelized CPD. The relevant theory and results for the continued work are included in this thesis. The new contributions to the project are elastic (non-rigid) CPD registration, GPU parallelization and other optimizations, as well as an evaluation on data from a new application, namely brain-shift correction.

Following is the outline of this thesis. Chapter 2 describes the background of this project and puts registration in the context of medical imaging. The brain-shift correction problem is described, which will serve as a registration application later. At the end of this chapter the FAST framework is introduced. Chapter 3 contains a thorough introduction to the mathematical foundation of the CPD algorithm. In chapter 4, the attention is back at point set registration, describing several developed algorithms, before the derivation of the rigid, affine and elastic CPD algorithms are included in detail. Chapter 5 describes implementation details and CPD parallelization. In Chapter 6, the runtimes and error measurements are described and compared to ICP and Gadomski's CPD implementation. Results from the application cases are also presented. Chapter 7 discusses the results and puts them in a context of related work. At last, chapter 8 concludes and summarizes the work.

2 Project background

Registration can be used for numerous different purposes. This project is written in a context where medical applications are of particular interest. Registration is a topic the medical technology group at SINTEF has worked extensively with. The method and algorithmic steps of the registration algorithm considered in this thesis will however be general and not depend on the source from which the point set is acquired. In order to present a broader picture of registration and its possibilities, it is natural to start out with the setting of medical imaging. Then a specific problem known as brain-shift will be addressed and work as a test case for the registration implementation developed. The implementation developed in this thesis is included in the open-source framework FAST, which will be introduced at the end of this chapter.

2.1 Medical imaging

The purpose of medical imaging is to gather and visualize anatomical data of a patient. It is desirable to acquire as much data with as little intervention as possible. Medical imaging techniques allow one to study the inside of a patient without the need of any medical intervention. This information is useful for diagnosis, treatment and follow-up.

Different types of medical imaging are called modalities. Common modalities include X-ray, computer tomography (CT), magnetic resonance (MR), positron emission tomography (PET) and ultrasound (US). The principles behind the various modalities determine what kind of data is acquired and the application of the data. Since the application studied in this project involve data from US and MR, a short review of these specific modalities is a good place continue.

2.1.1 Ultrasound

Medical ultrasound is a medical image modality based on reflection of sound waves with frequencies higher than the human ear can hear (ultrasound), i.e. above 20 kHz. For medical purposes, the frequencies used are usually at the level of a few MHz. An ultrasound probe is used to initiate sound waves propagating in the body. Every time the sound waves goes from one tissue type to another with different acoustic impedance, some of the sound is reflected back to the probe. These echoes are recorded and used to create an ultrasound image.

The propagation speed of the sound waves depends on the density and compressibility of the medium it propagates through. Approximating this speed as constant, and by sending in pulses of ultrasound waves, one can estimate the depth of a reflection by measuring the time between sending and arrival of the signal. Sending multiple pulses in multiple directions one can create an image from the reflected signals. This method is called pulsed-wave ultrasound (PW-US), and there are also other methods of sending and interpreting the reflected ultrasound signals. As sound waves propagate faster in matter than air, a gel is applied on ultrasound probes ensuring the sound waves propagate well into the body.

One major advantage of ultrasound is the minimal risk related to it. The frequencies used for medical ultrasound provide practically no risk to the patient. In addition, ultrasound is relatively cheap and available, and simple to use. Ultrasound equipment is also portable and can be made very small, compared to MR or CT equipment, for instance. By using a special kind of ultrasound acquisition called Doppler, it is possible to estimate the blood flow through the blood vessels.

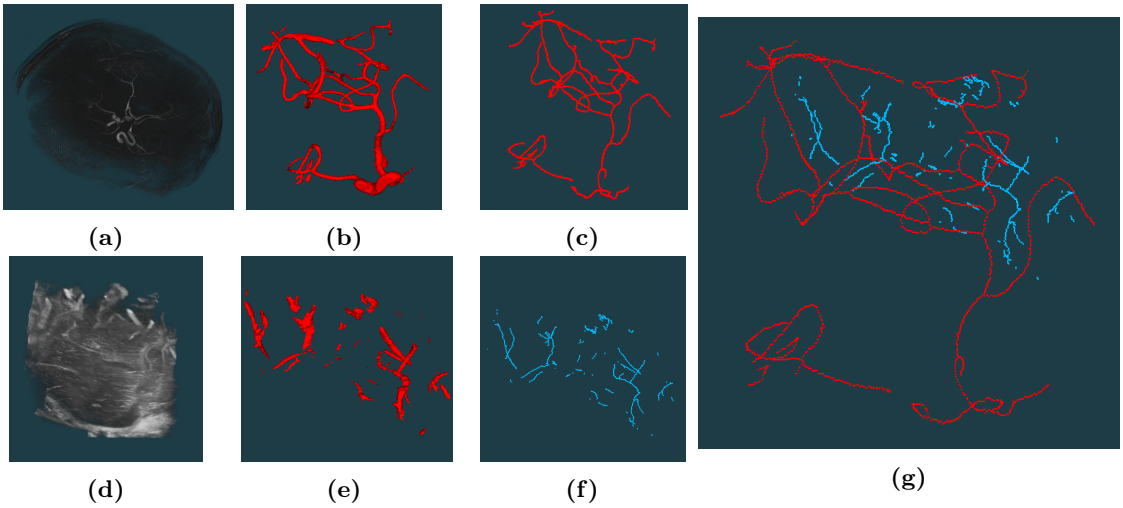


Figure 1: The data pre-processing steps for brain-shift correction with point set registration. From the acquired medical data (first column) the segmented blood vessels are segmented out (second column) and then the centerlines (third column) are extracted from the segmented vessels. The first and second row show examples from MR and US, respectively. The fourth column shows the centerlines from the MR and US data together

2.2 Magnetic resonance

In short, magnetic resonance (MR) uses a combination of strong magnetic fields and radio waves to produce images of the insides of the body. Different tissue types will respond differently to the radio waves, and due to the magnetic field it is possible to locate the different tissue types. The radiation used is non-ionizing, and thus harmless for the patient. With MR detailed 2D slices can be combined to give 3D images of the region of interest. A special kind of MR called MR angiography (MRA) is useful for medical imaging of blood vessels.

2.3 Brain-shift correction

The human brain is a complex and delicate organ protected by the solid skull. The brain itself is a not a rigid object, but rather soft with the possibility of moving slightly around inside the skull. Tumors and vessel aneurysms are two of many reasons a person might need brain surgery. Before such surgeries medical imaging techniques such as MRA are often used for planning before the operation and navigation during the operation. When the skull is opened, and as the surgery is ongoing, the brain can move around (shift) and deform, making it difficult for surgeons to navigate by the pre-operative images. This problem is known as *brain-shift*.

To account for the brain-shift, new brain images can be useful, but the brain-shift causes the orientation and form of the brain in these images to deviate from the pre-operative images. This effect must be corrected for, giving rise to the brain-shift correction problem. To solve this, registration is needed to align the intra-operative images with the pre-operative images. There are several approaches to do this. In [9], Ingerid Reinertsen et. al. describe a method for intra-operative correction of brain-shift using image registration of blood vessels segmented from pre-operative MRA and intra-operative US Doppler. Centerlines can then be extracted from these segmented vessels. Point set registration algorithms, which will be further explained later, can be used to register blood vessel centerlines, and thus correct the brain-shift. The process from MRA and US Doppler data to centerlines is shown in figure 1. The algorithm developed in this project is applied to data acquired before and during brain surgeries, trying to do brain-shift correction.

2.4 FAST Framework

For efficient processing and visualization of medical images in this project, the framework FAST (FrAmework for heterogeneouS medical image compuTing and visualization) developed by Erik Smistad was used [3]. FAST is an open-source cross-platform framework with the main goal of making it easier to do efficient processing and visualization on heterogeneous systems [10]. The source code for the framework is freely available online [11] under a BSD-licence, basically allowing anyone to do whatever they want with the code at their own risk. The term heterogeneous systems in computer science refers to combinations of different processors such as CPUs (Central Processing Unit) and GPUs (Graphic Processing Unit). Both of these processor types open up for parallelization, which can reduce computation times considerably, compared to traditional single-core computations. Later chapters will cover more details on how this is taken advantage of in the registration implementation. All image processing in this project was performed using FAST. Methods for segmentation and centerline extraction are included in FAST.

3 Mathematical background

For a satisfactory explanation of the workings of Coherent Point Drift, some mathematical prerequisites deserve attention before the details of registration and CPD is brought up. The mathematical level is attempted to be held at an understandable level, providing quite detailed derivations. This section is not required to understand the registration algorithm developed in this project or the results but will work as a mathematical foundation and motivation for the derivation of the CPD algorithm. A summary of the mathematical notation used throughout this thesis is provided right before chapter 1.CPD is a probabilistic algorithm, and thus a good place to start is with statistics and probability theory.

3.1 Bayesian probabilities

Bayesian probabilities are used to quantify how certain one can be of an observation. This view of probabilities is not quite the same as the *frequentist* view, where probabilities are interpreted as how likely a new event is to occur based on how frequent a random, repeatable event has happened before [12, p. 21]. The Bayesian perspective allows for combining prior knowledge with new observations. A probability distribution describes how the probabilities of an event are distributed among all the possible outcomes of the event.

3.1.1 Density estimation

A random variable $\boldsymbol{x}$ is a variable whose value is the outcome of a random process or event. The distribution of all possible outcomes of an event, and thus the possible values $\boldsymbol{x}$ can take, can be described by a function named the *probability distribution*, $p(\boldsymbol{x})$. In many experimental situations one only observe a set of N outcomes $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, yet the distribution itself is unknown. The problem of modelling the probability distribution based on the given set of observations is known as *density estimation* [12, p. 67]. It is worth noting that there are in principle infinitely many probability distributions that can be the cause of a finite amount of observations, in the same manner as there are infinitely many polynomials fitting a finite amount of data points, implying that the density estimation problem is fundamentally ill-posed. The challenge of determining an appropriate probability function will motivate much of the following theory.

3.1.2 The likelihood function

Consider a model with a set of m parameters $\boldsymbol{w} = (w_1, w_2, \ldots, w_m)$. An example of a model could be a polynomial to be fitted to a data set, where $\boldsymbol{w}$ represent the coefficients of the polynomial. These parameters will be distributed somehow, and this is described by the *prior distribution* $p(\boldsymbol{w})$. The distribution is given prior to any new observation is made. One can distribute the parameters by using existing data points, mere intuition or in any other way considered reasonable, depending on the current application. Let $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ be a set of N observations that should be compared to the model. A couple of new probabilities is now interesting to study. Firstly, the probability of the model to yield a result identical to the observations given its current parameters. This conditional probability is called the *likelihood function* and is denoted $p(\mathcal{D}|\boldsymbol{w})$. Secondly, the uncertainty of the model parameters given only the observations made, i.e. what is the likelihood of these parameters to be correct based on the set of new observations acquired after the parameters were set. This is called the *posterior probability* and denoted $p(\boldsymbol{w}|\mathcal{D})$. Bayes' theorem gives the relation between these as

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})}, \tag{1}$$

where $p(\mathcal{D})$ is the total probability of the model outputting a result equal to the observations among all possible parameters values. In words, the posterior probability is proportional to the product of the likelihood function and the prior probability,

$$\texttt{posterior} \propto \texttt{likelihood} \cdot \texttt{prior}, \tag{2}$$

being normalized by $p(\mathcal{D})$.

3.1.3 Maximizing the likelihood

Finding the parameters $\boldsymbol{w}$ that maximizes the likelihood $p(\mathcal{D}|\boldsymbol{w})$ is known as maximum likelihood estimation, or simply *maximum likelihood* [12, p. 23]. The negative of the natural logarithm of the likelihood (negative log-likelihood, NLL) is known in machine learning settings as the *loss* or *cost* function of the likelihood. As the machine should learn to find highest probability, a low probability is punished by a high cost in NLL, while high probabilities are assigned minimal cost. Since the negative logarithmic function is monotonically decreasing, maximizing the likelihood is equivalent to minimize the cost, i.e.

$$\max\{p(\mathcal{D}|\boldsymbol{w})\} \iff \min\{-\ln p(\mathcal{D}|\boldsymbol{w})\}. \tag{3}$$

The motivation for applying this logarithm to the likelihood is both to simplify mathematical expressions, exponential likelihood functions for instance, and to improve the numerical precision in cases where small probabilities are multiplied, but now rather can be summed [12, p. 26]. In some cases, it is possible to find explicit expressions for the parameters that minimize the cost, while in some cases maximum likelihood is more complicated to reach.

3.2 Gaussian Mixture Models

Linear combinations of basic probability distributions are called mixture distributions and can be used to model a set of observations originating from different probabilistic models, giving a *mixture model*. A mixture model of Gaussian probability distributions makes a Gaussian mixture model (GMM). By combining a sufficient number of Gaussians, and adjusting each distribution's parameters and the way the distributions are combined, any continuous distribution can be approximated to arbitrary accuracy [12, p. 111]. Each distribution in the mixture is called a *mixture component*.

3.2.1 Gaussian probability distributions

The Gaussian probability distribution in one dimension is a function with two parameters, namely its mean μ and variance σ^2,

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{4}$$

Generalizing to a multivariate Gaussian of dimension D the general corresponding expression is

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{5}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix and $|\boldsymbol{\Sigma}|$ its determinant [12, p. 78][13]. For simplification, one can assume that the variances in the different dimensions are independent, meaning that their covariances are equal to zero, $\text{cov}(\sigma_i,\sigma_j) = 0 \ \forall \ i,j \in \{1,2,\ldots,D\}, i \neq j$. This restricts and simplifies the covariance matrices $\boldsymbol{\Sigma}$ to diagonal matrices with diagonal $\boldsymbol{\sigma}^2 = (\sigma_1^2,\ldots,\sigma_D^2)$. Another restriction is to let the variance be identical in every dimension, $\sigma_1^2,\ldots,\sigma_D^2 = \sigma^2$. This results in so-called isotropic covariance $\boldsymbol{\Sigma} = \sigma^2\boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix [12, p. 84]. The isotropic covariance restriction limits a single Gaussian distribution's ability to approximate a general distribution, but within a mixture of Gaussians this can be accounted for by including more components in the model. This assumption on the variances will be used in the rest of this section, simplifying the expression for the multivariate Gaussian to

$$\mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu},\sigma^2\right) = \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(-\frac{1}{2\sigma^2}\left\|\boldsymbol{x}-\boldsymbol{\mu}\right\|^2\right), \tag{6}$$

and also reducing the number of parameters.

A D-dimensional GMM with K components and isotropic covariance will take the form

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \phi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_k}, \sigma_k^2), \tag{7}$$

where the mixing coefficients $\phi_k \in [0, 1]$ are normalized,

$$\sum_{k=1}^{K} \phi_k = 1. \tag{8}$$

3.2.2 Responsibilities

The mixing coefficients can be viewed as the prior probability of a random observation to come from the k^{th} component C_k among the K possible, $p(k) = \phi_k$. Each component $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_k}, \sigma_k^2) = p(\boldsymbol{x}|k)$ correspond to the likelihood function, describing the probability of the observing $\boldsymbol{x}$ when the component the observation is coming from is given. The posterior probability will in this case be the probability $p(C_k|\boldsymbol{x})$ yielding the likelihood of a given observation originating from component C_k. Applying Bayes' theorem, the posterior probability can be expressed as

$$p(C_k|\boldsymbol{x}) = \frac{p(C_k)p(\boldsymbol{x}|C_k)}{p(\boldsymbol{x})} = \frac{\phi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \sigma_k^2)}{\sum_{l=1}^{K} \phi_l \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_l, \sigma_l^2)}. \tag{9}$$

These posterior probabilities $p(C_k|\boldsymbol{x})$ are also known as *responsibilities*, as they can be interpreted as how much each component k is responsible for the result $\boldsymbol{x}$.

The results so far have only considered a single observation $\boldsymbol{x}$. By regarding the observations in a set of N observations $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ as independent and identically distributed (i.i.d.) random variables, the total GMM will be the product of probabilities in (7),

$$p(\boldsymbol{X}) = \prod_{n=1}^{N} p(\boldsymbol{x_n}) = \prod_{n=1}^{N} \left(\sum_{k=1}^{K} \phi_k \mathcal{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \sigma_k^2) \right). \tag{10}$$

The responsibility of component C_k for result $\boldsymbol{x}_n$ is denoted

$$\gamma_{nk} \equiv p(C_k|\boldsymbol{x}_n) = \frac{\phi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \sigma_k^2)}{\sum_{l=1}^{K} \phi_l \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_l, \sigma_l^2)}. \tag{11}$$

All these responsibilities can be stored in a $N \times K$ matrix $\boldsymbol{P}$ with elements $\boldsymbol{P}_{nk} = \gamma_{nk}$. Let such a matrix be named a *responsibility matrix*.

3.2.3 Maximum likelihood of GMM

In order to find the parameters that minimize the cost function, and maximize the likelihood, the derivatives of the cost with respect to the different parameters are set to zero. It will soon be clear that any direct analytic solution to the maximum likelihood approach is not available here, yet the responsibilities will appear often. An attempt at maximum likelihood of the GMM will now be given to look for alternative approaches.

The cost, i.e. the negative log-likelihood, NLL, for the set of observations $\boldsymbol{X}$ is

$$\begin{aligned} NLL &\equiv -\ln p(\boldsymbol{X}|\phi_k, \boldsymbol{\mu}_k, \sigma_k^2) \\ &= -\ln \left(\prod_{n=1}^{N} \left(\sum_{k=1}^{K} \phi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \sigma_k^2) \right) \right) \\ &= -\sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \phi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \sigma_k^2) \right). \end{aligned} \tag{12}$$

Inserting (4) into the right-hand side of (12) and differentiating it with respect to the mean $\boldsymbol{\mu}_k$ of component C_k give

$$\frac{\partial(NLL)}{\partial \boldsymbol{\mu}_k} = -\frac{1}{\sigma^2}\sum_{n=1}^{N}\left((\boldsymbol{x}_n - \boldsymbol{\mu}_k)\frac{\phi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \sigma_k^2)}{\sum_{l=1}^{K}\phi_l \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \sigma_k^2)}\right). \tag{13}$$

In deriving (13), $\frac{\partial}{\partial \boldsymbol{\mu}_k}\left\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\right\|^2 = -2(\boldsymbol{x}_n - \boldsymbol{\mu}_k)$ is used, which is is evident by applying the gradient component-wise to the norm: $(\frac{\partial}{\partial \mu_{k,1}}, \ldots, \frac{\partial}{\partial \mu_{k,D}})\left[(x_{n,1} - \mu_{k,1})^2 + \cdots + (x_{n,D} - \mu_{k,D})^2\right]$. Only the terms of the sum over k containing the index of the differentiation parameter in (12) will be kept. Notice that the expression for the responsibility γ_{nk} from (11) appears in (13). Equated to zero and re-organized, $\boldsymbol{\mu}_k$ in (13) can be expressed by the responsibility as

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N}\boldsymbol{x}_n\gamma_{nk}}{N_k}, \tag{14}$$

where the denominator N_k is defined as,

$$N_k \equiv \sum_{n=1}^{N}\gamma_{nk}. \tag{15}$$

Because the responsibilities depend on the mean this is not an analytic solution for the mean, but it is still an interesting expression to study. The form of the equation is similar to a general weighted mean. Remembering the interpretation of the responsibilities γ_{nk} as how much component C_k is responsible for the result $\boldsymbol{x}_n$, the sum of all the responsibilities related to component C_k, which is N_k, may be interpreted as the effective number of observations that component is taking responsibility for. The expression in (14) can then in turn be viewed as the weighted mean of all observations component C_k claims responsibility for, weighted by how responsible it considers itself for those observations.

Differentiating NLL with respect to the variance σ_k^2 result in

$$\frac{\partial(NLL)}{\partial \sigma_k^2} = \frac{1}{2\sigma_k^2}\sum_{n=1}^{N}\left[\left(\frac{1}{\sigma_k^2}\left\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\right\|^2 - D\right)\frac{\phi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \sigma_k^2)}{\sum_{l=1}^{K}\phi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \sigma_k^2)}\right], \tag{16}$$

where the expression for responsibilities γ_{nk} from (11) appears again, and the expression for N_k as well. Equated to zero and expressed by γ_{nk} and N_k, equation (16) can result in the following expression for the variance:

$$\sigma_k^2 = \frac{\sum_{n=1}^{N}\gamma_{nk}\left\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\right\|^2}{N_k D}. \tag{17}$$

The form of (17) for the variances is the same as for the means in (14) but with the norm of the distances between the observations and the means as weights. Because the variance is assumed to be equal in all dimensions (isotropic covariances) and σ_k represents the variance in each dimension, the factor of D also appears in the denominator. The same kind of interpreting as for the mean will describe the variances σ_k^2 here as the effective variance of all observations component C_k is taking responsibility for.

Moving on to the mixing coefficients, one must assure the normalization condition from (8) is still true. A popular method for differentiating where some of the variables have restrictions is to use the Lagrange multiplier λ. The differentiation problem can then be formulated as

$$\begin{aligned}\frac{\partial}{\partial \phi_k}(NLL) &= \lambda\frac{\partial}{\partial \phi_k}\left(\sum_{l=1}^{K}\phi_l - 1\right)\\ \sum_{l=1}^{K}\phi_l &= 1.\end{aligned} \tag{18}$$

Solving this system give $-N_k = \lambda\phi_k$ with $\lambda = \sum_{l=1}^{K}N_k = N$. Summing over k on both sides of the first equation and making use of the restriction, the mixing coefficients can be expressed as

$$\phi_k = \frac{N_k}{N}, \tag{19}$$

representing the effective fraction of the points component C_k is responsible for.

For a single Gaussian model, rather than a GMM, the sum inside the logarithm in (12) would not be there, and it would be possible to obtain a closed form analytic solution for the parameters μ_k, σ_k^2 and ϕ_k. In this case however, this is not the case, yet all parameters can be expressed recursively by the responsibilities γ_{nk}. In order to estimate maximum likelihood for GMMs one must resort to iterative methods, such as *expectation maximization*.

3.3 Expectation maximization for GMMs

Expectation maximization (EM) is an iterative method for maximum likelihood estimation, particularly useful where there is no analytic solution for parameter values that maximizes the likelihood. The application on GMMs will be a special case of the general EM Algorithm described by Dempster et. al in 1977 [14]. Initially the model parameters are set by guessing or at random. The EM algorithm then consists of two steps, repeated until convergence. As the name suggests, these steps are expectation (E-step) and maximization (M-step). In the E-step all the expected responsibilities are estimated based on the current model parameters. Then these responsibilities are used to estimate new values for the parameters in the M-step. It can be shown that an E-step followed by a M-step is guaranteed to increase the log-likelihood, or equivalently reduce the cost, implying that after a sufficient number of iterations the algorithm will converge to a local (or global) maximum likelihood [12, p. 451-452][13]. The EM algorithm is quite general and have many application areas, but in this project, EM applied to GMMs is of interest.

The model parameters to be estimated when dealing with GMMs are the mixing coefficients ϕ_k, the means $\boldsymbol{\mu}_k$ and the variances σ_k^2 for each of the K mixture components C_k. In section 3.2.3 all these parameters were expressed by the responsibilities γ_{nk}. A natural iterative approach is therefore to first estimate the responsibilities by the current parameters, and then estimate the new parameters by using that responsibility. For clarity, the terms involving the current parameter estimations be denoted "old", the new estimations "new".

Let the mixing coefficients be initialized by a uniform distribution, $\phi_1 = \phi_2, \ldots, \phi_K = 1/K$. The means can be initialized by setting each component mean $\boldsymbol{\mu}_k$ equal an arbitrary observation $\boldsymbol{x}_i$, keeping the means different from each other. The variances can all be initialized to the total variance of the observations, $\sigma_1 = \cdots = \sigma_K = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^2$, where $\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i$ is the mean of the observations. These initialization settings are only examples, and other settings may work as well. An initialization close to the true values may reduce the number of iterations needed afterwards.

In the expectation step, the responsibilities from equation (11) are computed with the current parameter estimates,

$$\gamma_{nk}^{\text{old}} = \frac{\phi_k p^{\text{old}}(\boldsymbol{x}_n|C_k)}{p^{\text{old}}(\boldsymbol{x}_n)} = \frac{\phi_k^{\text{old}}\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k^{\text{old}}, \sigma_k^{2\,\text{old}})}{\sum_{l=1}^{K}\phi_l^{\text{old}}\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_l^{\text{old}}, \sigma_l^{2\,\text{old}})}, \tag{20}$$

and can be stored as the elements of the responsibility matrix $\boldsymbol{P}^{\text{old}}$. The effective number of observations related to component C_k, namely N_k, can as well be calculated now,

$$N_k^{\text{old}} = \sum_{n=1}^{N}\gamma_{nk}^{\text{old}}. \tag{21}$$

In order to find update rules for the parameters for the maximization step it will be of use to take a closer look on the cost difference before and after the parameters are updated. Let the cost from (12) be be NLL^{old} at the beginning of a iteration and NLL^{new} after the parameters are

updated. The change in the cost function is then

$$\begin{aligned} NLL^{\text{new}} - NLL^{\text{old}} &= -\sum_{n=1}^{N} \ln p^{\text{new}}(\boldsymbol{x}_n) + \sum_{n=1}^{N} \ln p^{\text{old}}(\boldsymbol{x}_n) \\ &= -\sum_{n=1}^{N} \ln \frac{p^{\text{new}}(\boldsymbol{x}_n)}{p^{\text{old}}(\boldsymbol{x}_n)}. \end{aligned} \tag{22}$$

By using Jensen's inequality, it becomes clear that this iteration scheme will find a minimum for the cost. Jensen's inequality states that

$$\ln\left(\sum_j \lambda_j x_j\right) \geq \sum_j \lambda_j \ln(x_j), \tag{23}$$

when $\lambda_j \geq 0$ and $\sum_j \lambda_j = 1$. Introducing the factor $\gamma_n^{\text{old}} j / \gamma_n^{\text{old}} j = 1$ into (22), the responsibility γ_{nj}^{old} in the nominator may play the role as λ_j in Jensen's inequality, as it sums to one over all components C_j. The cost change in now satisfies

$$\begin{aligned} NLL^{\text{new}} - NLL^{\text{old}} &= -\sum_{n=1}^{N} \ln \left[\frac{\sum_{j=1}^{N} \phi_j^{\text{new}} p^{\text{new}}(\boldsymbol{x}_n | C_j)}{p^{\text{old}}(\boldsymbol{x}_n)} \frac{\gamma_{nj}^{\text{old}}}{\gamma_{nj}^{\text{old}}} \right] \\ &\leq -\sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_{nj}^{\text{old}} \ln \left[\frac{\phi_j^{\text{new}} p^{\text{new}}(\boldsymbol{x}_n | C_j)}{p^{\text{old}}(\boldsymbol{x}_n) \gamma_{nj}^{\text{old}}} \right]. \end{aligned} \tag{24}$$

Defining the right-hand side of the equality above as

$$\tilde{Q} \equiv -\sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_{nj}^{\text{old}} \ln \left[\frac{\phi_j^{\text{new}} p^{\text{new}}(\boldsymbol{x}_n | C_j)}{p^{\text{old}}(\boldsymbol{x}_n) \gamma_{nj}^{\text{old}}} \right], \tag{25}$$

it is evident that the new cost value $NLL^{\text{new}} \leq NLL^{\text{old}} + \tilde{Q}$, has an upper bound of $NLL^{\text{old}} + \tilde{Q}$. This means that the cost function in fact will be minimized for every iteration, unless it already is at a minimum. Minimizing the cost is done by minimizing $\tilde{Q}$. For the sake of finding equations describing how the "new" parameters should be updated to minimize the cost, the terms of $\tilde{Q}$ containing only "old" parameters can be discarded for now, leaving

$$Q = -\sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_{nj}^{\text{old}} \ln \left[\phi_j^{\text{new}} p^{\text{new}}(\boldsymbol{x}_n | C_j) \right], \tag{26}$$

which will be called the *objective function*.

Minimizing the objective function Q by the same procedure as for NLL in section 3.2.3, the following rules for parameter updates are found to maximize the likelihood of a GMM:

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{\sum_{n=1}^{N} \boldsymbol{x}_n \gamma_{nk}^{\text{old}}}{N_k}, \tag{27}$$

$$\sigma_k^{2\,\text{new}} = \frac{\sum_{n=1}^{N} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k^{\text{new}}\|^2 \gamma_{nk}^{\text{old}}}{N_k D}, \tag{28}$$

$$\phi_k^{\text{new}} = \frac{N_k^{\text{old}}}{N}. \tag{29}$$

These equations correspond well to equations (14), (17) and (19) found when looking at the cost function. Performing these parameter updates is the M-step of the EM algorithm. After E- and M-step the new parameter values overwrites the old ones, and the E- and M-steps are repeated until convergence. By calculating the cost function at each iteration, convergence can be defined by a threshold under which the cost change should be before the algorithm is stopped.

3.4 Transformations

As registration is the process of finding the spatial correspondence between points, it is convenient to think of the difference as a transformation. A transformation can cause a transition from one state to another. In mathematics the word mapping is also used for the same purpose; it describes a function that maps one set of points (objects) to another set of points, usually both of equal dimension. Applying a transformation to a set of points will give a new set of points. The outcome depends on the properties of the specific transformation. Some of the most fundamental transformations include translation, scaling, rotation, and shearing. Combinations of these transformations are also transformations.

3.4.1 Rigid, similarity and affine transformations

Transformations can be classified by their mathematical properties and what kind of geometric features they preserve. Three common classes of transformations are *rigid*, *similitude* and *affine* [15]. Rigid transformation includes translation and rotation of objects. Combinations of these operations will leave the transformed object in the same size and shape, but possibly with a new position and orientation. All geometric properties are preserved by a rigid transformation. The similarity transformation, also known as similitude, will in addition allow isotropic scaling of the object, i.e. the object is scaled equally in all directions, thus preserving the shape (similar geometries) but not its size. Adding arbitrary scaling, reflection and shearing to the mentioned transformations yield the affine transformation. Ratios of distances in any geometry is also preserved by affine transformations. An overview of some geometric properties preserved by the different kinds of transformations is shown in table 1.

Table 1: Table over some preserved properties in affine, similarity and rigid transformations.

Preserved property	**Affine**	**Similitude**	**Rigid**
Distances	No	No	Yes
Angles	No	Yes	Yes
Ratios of distances	Yes	Yes	Yes
Parallel lines	Yes	Yes	Yes
Straight lines	Yes	Yes	Yes

3.4.2 Matrix representation

Representing transformations by matrices make the representation consistent and transformations easy to compute. It also makes it easier to combine transformations, simply by multiplying the transformation matrices. Not all transformation can be described be in this way, but here the intention is to find matrix representations for the rigid, similarity and affine transformations. Consider a transformation $\mathcal{T}$ and a d-dimensional point described by the column vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^T$. Any transformation $\mathcal{T}(\boldsymbol{x})$ of point $\boldsymbol{x}$ that can be described by a $d \times d$ square matrix $\boldsymbol{A}$ as $\mathcal{T}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ is called a *linear transformation* and $\boldsymbol{A}$ is called the *transformation matrix*. Setting the transformation matrix equal to the identity matrix, $\boldsymbol{A} = \boldsymbol{I}$, the point remains unaltered, and therefore this will be the foundation for any transformation matrix.

Isotropic scaling can be described simply by multiplying the desired scaling factor s to the identity matrix, effectively scaling all diagonal matrix elements and all point components by s, $\mathcal{T}(\boldsymbol{x}) = s\boldsymbol{I} = (sx_1, \ldots, sx_d)^T$. For scaling in general, the diagonal element in a given row should

be scaled with the desired scaling factor for that component, resulting in the transformation

$$\mathcal{T}_{scale}(\boldsymbol{x}; s_1, \ldots, s_d) = \boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} s_1 & 0 & \ldots & 0 \\ 0 & s_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & s_d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}. \tag{30}$$

Rotations are specified by a rotation of an angle θ with respect to some kind of reference. For the 2D-case this reference is usually the origin of the Cartesian coordinate system, while in 3D it can be either of the three coordinate axes. In order for a matrix $\boldsymbol{R}$ to represent a rotation it must satisfy two constraints: i) orthogonality, $\boldsymbol{R}^{-1} = \boldsymbol{R}^T \iff \boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I}$, and ii) determinant equal unity, $\det \boldsymbol{R} = 1$. The matrix representation will vary depending on the rotation reference, but the elements of the 2D rotation matrix will reappear. A rotation transformation in 2D of angle θ around the origin is described by

$$\mathcal{T}_{rot}(\boldsymbol{x}; \theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \boldsymbol{x}. \tag{31}$$

Shearing is a displacement changing proportionally along a given direction, altering angles between lines, but preserving parallel lines. A parallelogram can for instance be made by applying a shear transformation to a rectangle. In a 2D shear transformation parallel to the x-axis the values of the x-components are increased by a scaling factor λ multiplied by the y-component, $\mathcal{T}(\boldsymbol{x}; \lambda_x) = (x + \lambda_x y, y)^T$. A general 2D shear matrix consist of a combination of shear in x- and y-direction giving the transformation

$$\mathcal{T}_{shear}(\boldsymbol{x}; \lambda_x, \lambda_y) = \begin{bmatrix} 1 & \lambda_x \\ \lambda_y & 1 \end{bmatrix} \boldsymbol{x}. \tag{32}$$

A translation is simply another vector added to every point vector, $\mathcal{T}_{transl}(\boldsymbol{x}; \boldsymbol{t}) = \boldsymbol{x} + \boldsymbol{t}$. Expressing this addition by a square transformation matrix is not possible, and hence translation is not a linear transformation. An affine transformation is therefore expressed as a sum of a linear transformation and a translation, $\mathcal{T}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{t}$. However, by introducing something called *homogeneous coordinates* it is possible to combine the linear transformation matrix and the translation vector into one transformation matrix by introducing one extra coordinate. 2D matrices become 3D, 3D matrices become 4D, etc. The advantage of that achievement is is both a matter of simplicity and implementation, and the concept is widely used in computer graphics.

The concept of homogeneous coordinates is to include an additional non-zero component to every point, effectively increasing the dimension by one. Letting this new component be equal to unity, a point with homogeneous coordinates is given by $\boldsymbol{x}_h = (\boldsymbol{x}^T, 1) = (x_1, x_2, \ldots, x_n, 1)^T$, with subscript h for clarity here. The corresponding transformation matrix is extended to a $(d+1) \times (d+1)$ square matrix. Letting this matrix be equal to the linear transformation matrix extended by one row (right) and column (bottom). Setting this new last column to a homogeneous translation column vector $\boldsymbol{t}_h = (\boldsymbol{t}^T, 1)^T$, the transformation matrix becomes

$$\boldsymbol{A}_h = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix}. \tag{33}$$

Applying this homogeneous transformation matrix to a point it become evident that an affine transformation can be expressed by merely matrix multiplication, $\mathcal{T}_{affine}(\boldsymbol{x}_h) = \boldsymbol{A}_h\boldsymbol{x}_h = (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{t}, 1)^T$.

3.5 Computational complexity

There are several factors relevant when discussing the performance of one algorithm compared to another, and one which is the time it takes for the algorithm to complete its calculations. Focusing here on algorithms performing calculations on and input matrix, the computation time of the algorithm will typically depend on the size of the input. A typical concern is what happens to the runtime when the input size changes. A common notation for such a *computational complexity*, or simply *complexity* is the *big O notation*, which describes an upper bound of the asymptotic runtime of an algorithm as the input size increases to infinity. A linear development for input size n is denoted $\mathcal{O}(n)$, while a second order development is denoted $\mathcal{O}(n^2)$.

The complexity of algorithms often breaks down to performing standard mathematical operations such as matrix multiplication and matrix inversion. It is thus useful to briefly comment on the complexities of these operations. The direct and straight-forward sequential algorithm to multiply two matrices $\boldsymbol{A}_{M\times P}$ and $\boldsymbol{B}_{P\times N}$ has a $\mathcal{O}(MPN)$ complexity, or $\mathcal{O}(N^3)$ if they are square $N \times N$ matrices. Over the past 50 years algorithms providing subcubic complexity for multiplication of square matrices have been proved and the current best result is the $\mathcal{O}(2^{2.373})$ complexity shown by Virginia V. Williams in 2014 [16]. The inverse of a matrix can be calculated in a way for which matrix multiplication is the complexity bottleneck, giving a matrix inversion algorithm the same complexity as the matrix multiplication algorithm used in it [17, p. 106-108].

3.6 Matrix decomposition and approximation

A matrix may be decomposed, or factorized, into a product of several matrices. There exist a large number of different methods to accomplish this. The matrix decomposition methods produce matrices of specific forms with the main goal of simplifying calculations or to reveal some kind of structure of the decomposed matrix. In fact, the concept of decomposing matrices is so important and useful that it was regarded as one of the "Top ten algorithms" of the 20th century in [18]. Whereas the matrix decomposition yields an equivalent description of the matrix, a matrix approximation method will reduce the complexity of the matrix at a cost of reduced accuracy. A low-rank matrix approximation can be considered as a form of compression of the matrix content, any usually depends on matrix decomposition. Now, two popular decomposition methods, singular value decomposition (SVD) and eigendecomposition, both relevant for CPD, will be covered before low-rank approximation is studied in more depth. Both SVD and eigendecomposition may be used to solve sets of linear equations and to simplify matrix inversion and computations.

3.6.1 Singular Value Decomposition

A set of linear equations can be written on the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ and $\boldsymbol{b}$ are given. An interpretation of this equation is that $\boldsymbol{A}$ works as a transformation from $\mathbb{R}^N$ to $\mathbb{R}^M$, when $\boldsymbol{A}$ has dimensions $M \times N$. If $\boldsymbol{A}$ is not a square matrix, the system is either degenerate (multiple solutions exist) or overdetermined (more equations than variables), which implies in general that there does not exist one unique, if any, solution for $\boldsymbol{x}$ that solves the system. Considering only square $N \times N$ matrices $\boldsymbol{A}$ simplifies the problem, yet the system can still be degenerate if multiple rows or columns in $\boldsymbol{A}$ are close to being linearly dependent. In this case the set of equations is called *singular*. The result of $\boldsymbol{A}\boldsymbol{x}$ will reside in the space spanned by the columns of $\boldsymbol{A}$, as the resulting vector can be expressed as a linear combination of the columns of $\boldsymbol{A}$. The number of independent columns will reveal the rank R of $\boldsymbol{A}$. The column space, also known as the range, is one of four fundamental subspaces relevant to understanding the transformation.

It can be shown that $\boldsymbol{A}$ can be decomposed into two orthogonal matrices $\boldsymbol{U}$ and a $\boldsymbol{V}$ and diagonal matrix Σ such that

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{34}$$

where the diagonal elements σ_{ii} of Σ are positive and in decreasing order [19]. This is known as the *Singular value decomposition* (SVD) of $\boldsymbol{A}$, with the σ_{ii} called the singular values. An orthogonal matrix will have a transpose equal to the inverse of the matrix, and its columns are rows are

orthogonal unit vectors. The SVD is here stated for real matrices but does generalize to complex matrices as well. In [20] it shown that $\boldsymbol{U}$ and $\boldsymbol{V}$ constitute bases of fundamental spaces related to $\boldsymbol{A}$. In the case of a square matrix $\boldsymbol{A}$, the orthogonal matrices can be seen as rotation matrices, and the diagonal matrix a scaling matrix, and thus the SVD decomposition consists of a series of geometrical transformations.

3.6.2 Eigendecomposition

Matrices that satisfy certain properties can be expressed by its eigenvectors and eigenvalues as

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{-1}, \tag{35}$$

where $\boldsymbol{D}$ is a square diagonal matrix with the eigenvalues on the diagonal and $\boldsymbol{Q}$ is a square matrix in which the i-th column is the eigenvector corresponding to eigenvalue $\boldsymbol{D}_{ii} = \lambda_i$. This decomposition is called the eigendecomposition, or diagonalization of $\boldsymbol{A}$. The effect the three matrices in the decomposition have something in common with the ones in SVD: firstly an (invertible) linear transformation is applied, then the eigenvalues provide scaling before the inverse of the initial transformation is applied.

In general, eigendecomposition and SVD are different methods of decomposition. Firstly, the eigendecomposition only applies to matrices that are square, and only square matrices that have linearly independent eigenvectors, whereas the SVD is applicable to all matrices. An interesting special case is symmetric matrices $\boldsymbol{A}$, for which the spectral theorem guarantees the matrix to be diagonalizable [21]. In addition, the eigenvalues will be real, and the eigenvectors can be chosen to be orthogonal to each other. For such a case the eigendecomposition will simplify to

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{T}. \tag{36}$$

Because the decomposition is unique, the eigendecomposition and the SVD will be identical and the eigenvalues and singular values will be equal [17, p. 569-570].

3.6.3 Low-rank approximation

After a matrix is decomposed, its effect as a transformation is clearer. A consequence of this is that it also becomes more clear which part of the (decomposed) matrix that contributes most to the transformation, and which parts that do not contribute that much. In both the decomposition methods mentioned above, the diagonal matrix may be interpreted as some form of scaling. If the scaling factors are very small, i.e. small singular values or eigenvalues for SVD and eigendecomposition, respectively, the contribution from some of the rows and columns of the non-diagonal matrices, will be minimal compared to the row and columns corresponding to larger scaling factors in the diagonal matrix.

Utilizing the observation that some rows and columns of the matrix decomposition components does not contribute much to the matrix, one can reduce the dimensions in the components while keeping most of the information of the matrix studied. The accuracy of the approximation will obviously depend on how much content is removed and how small the scaling factors are. Consider a decomposition like the eigendecomposition given in (35) in which all the involved matrices are square with dimensions $M \times M$. Assuming that the diagonal elements of the diagonal matrix $\boldsymbol{D}$ are ordered in decreasing order, the smallest eigenvalues will appear in the rightmost bottom corner, and these values will scale the rows at the bottom of the component to the right of $\boldsymbol{D}$. When the matrix to the left of $\boldsymbol{D}$ is multiplied to the rest, its rightmost columns will be scaled in the same manner and not contribute much. An approximation for $\boldsymbol{A}$ is then given by simply keeping only the largest eigenvalues and the corresponding rows and columns and discarding the rest. If K eigenvalues are kept, this approximation method effectively reduces the rank of the matrix components, from M at most, to $K < M$. Thus, this method is called *low-rank approximation*. The matrix components from left to right in the eigendecomposition will now have dimensions $M \times K$, $K \times K$ and $K \times M$, respectively,

$$\boldsymbol{A}_{M\times M} \approx \boldsymbol{P}_{M\times K}\boldsymbol{D}_{K\times K}\boldsymbol{P}^{-1}_{K\times M}. \tag{37}$$

3.7 Solving systems of linear equations

A set of linear equations can be written on the matrix form $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ and $\boldsymbol{b}$ are given. If $\boldsymbol{A}$ is not a square matrix the system is either degenerate (multiple solutions exist) or overdetermined (more equations than variables), which implies in general that there does not exist one unique, if any, solution for $\boldsymbol{x}$ that solves the system. Considering only square matrices $\boldsymbol{A}$ leaves out the overdetermined systems, yet the systems can still be degenerate if multiple rows or columns in $\boldsymbol{A}$ are close to being linearly dependent. In this case the set of equations is called *singular*, while the other sets are called non-singular. Non-singular sets of equations are guaranteed one solution (by the equivalent statements in linear algebra), while for singular problems this is not the case.

There are in principle at least two different approaches for solving a non-singular set of linear equations. The first is to solve the system of equations by the means of some kind of Gauss-Jordan elimination and/or by matrix decompositions. The other approach is to compute the inverse of $\boldsymbol{A}$, and pre-multiply both sides of the equation with it to get $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$. The methods of the solver-approaches do not necessarily produce the inverse matrix. Decomposition of $\boldsymbol{A}$ may also be used in the second approach to simplify the matrix inversion. For singular sets of equations, solutions exist if the right-hand side $\boldsymbol{b}$ lies in the range of $\boldsymbol{A}$. If this is the case, SVD can be used to find the solution $\boldsymbol{x}$ with the smallest norm, and if not, the same method can be used to find the $\boldsymbol{x}$ that is best possible result in the least-squares sense [17, p. 69-70].

3.8 Statistical learning and regularization

3.8.1 Inverse problems

A mathematical problem is considered to be well posed if there i) exists a solution to the problem, ii) the solution is unique, and iii) the solution is stable. A stable solution means that continuous changes in the initial conditions should result in continuous behavior of the solution. These three criteria are known as the Hadamard criteria after Jacques Hadamard. Any problem not satisfying the Hadamard criteria is said to be *ill-posed*. The problem can also be classified by how much small alterations in the input alter the output. If small error in the input causes a small error in the output, the problem is regarded well-conditioned, while the problem is regarded as ill-conditioned if the error in the output is large. This can be studied further by looking at the so-called condition number.

The purpose of a typical mathematical model for a physical process/phenomenon is to predict the outcome of the process given a set of inputs and parameters defining the model. However, in some cases only the results are known, and the model parameters that caused the of the observed result are unknown. Such problems are known as *inverse problems*. In medical imaging and computer vision there are many inverse problems. In the case of point set registration, it is clear how the points ended up, yet the parameters of the transformation they have been exposed to is unknown and must be determined. Inverse problems are often ill-posed due to lack of stability in the solutions. The fact that several parameter values might be consistent with the results, and that one might need to scan a potentially vast parameter space searching the optimal ones, contribute to making inverse problems difficult to solve [22].

3.8.2 Statistical learning

Learning something from data by statistical methods is the topic of the field statistical learning and machine learning. The learning is considered supervised or unsupervised depending on whether the learning model do or do not require some results to train on, respectively, in order to learn. A typical problem consists of some input data or variables $\boldsymbol{X}$, a function f taking $\boldsymbol{X}$ as input, and some output data $\boldsymbol{Y}$. The input variables are often called predictors, or independent variables, and the output of the model is known as the response. For supervised learning the model is given pairs of corresponding inputs and outputs from which the model learns.

In statistical learning one discerns between prediction and inference. A prediction is a qualified

guess on what a result might be for a given input. Inference, on the other hand, is not concerned with the result but how the input turned ended it up as a given result. One can also classify learning models as parametric and non-parametric. Using a parametric method, or model-based method, some assumptions on the form of f has been made, i.e. that the function can be described by a set of parameters. Non-parametric methods do not make this assumption and seek for any function that resembles the true function best. For inference parametric and relative inflexible methods are preferred [23].

3.8.3 Empirical risk minimization

Let $f = f(\boldsymbol{x})$ be a function that estimates a mathematical distribution p, mapping an input data set $\boldsymbol{X} = \{\boldsymbol{x}_i\}$ to a set of observations (outputs) $\boldsymbol{Y} = \{y_i\}$. Consider this an inverse problem where p is assumed unknown and the goal is to estimate f based on the two given data sets. Ideally the outputs would be direct outputs from the true distribution, $y_i = p(\boldsymbol{x}_i)$, but in practical applications there will always be some noise ϵ, causing the data to be corrupted, giving the estimate $y_i = f(\boldsymbol{x}_i) = p(\boldsymbol{x}_i) + \epsilon$. In order to say something quantitatively about the estimate a loss function $L(\boldsymbol{x}, y)$ is defined. The loss function provides a deviance measurement between the observed output values and the true output values. A common and simple loss function is the mean square error, also known as the squared L_2 norm.

In statistics, the expected loss is called *risk*, and regarding this problem as an optimization problem, it desirable to find the estimate f that minimizes the risk. Let $R = R[f]$ be a functional ("a function of functions") expressing the average loss function with respect to the true distribution, which is called *expected risk*,

$$R[f] = \int_{X \times Y} L(\boldsymbol{x}, f(\boldsymbol{x}) p(\boldsymbol{x}, y) \mathrm{d}\boldsymbol{x} \mathrm{d}y. \tag{38}$$

Since the distribution p is unknown, the risk must be approximated by the observed data available. Let this be the *empirical risk* functional R_{emp} [24],

$$R_{\text{emp}}[f] = \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{x}_i, f(\boldsymbol{x}_i)). \tag{39}$$

The principle of making the empirical risk as small as possible is called *Empirical Risk Minimization* (ERM) and is a principle that many optimization methods, such as the method of least squares, build upon. Letting the function that minimizes the empirical risk be denoted $\hat{f}$, the ERM can be formulated as

$$\hat{f} = \arg\min_{f \in \mathcal{F}} R_{\text{emp}}[f], \tag{40}$$

where $\mathcal{F}$ is set of all functions f can take, also known as the *hypothesis space*.

3.8.4 Regularization

When solving mathematical problems numerically the problems should be well-posed to succeed. If the problem is ill-posed one need to make the problem well posed in order to proceed. One method to do this is to add more information to the problem, for instance by declaring some restrictions the model must obey. This process is known as *regularization*, and it is a technique used to solve ill-posed problems and to prevent overfitting, i.e. that a model become too specialized on the given input data and thus not able to develop a general model working for new inputs. An example of a much used restriction imposed by regularization is smoothness, implying that two similar inputs correspond to similar outputs [25]. Keeping in mind the Bayesian perspective the beginning of this chapter, regularization can be regarded as setting a prior distribution on the model parameters.

Mathematically, the regularization can be expressed as a regularization functional $\phi[f]$ added to the empirical risk yielding a regularized risk functional $H[f]$,

$$H[f] = R_{\text{emp}}[f] + \lambda \phi[f], \tag{41}$$

where $\lambda > 0$ is called the *regularization parameter*. The purpose of ϕ is often to impose smoothness to the function by penalizing large variations with higher risk, and then ϕ is called the *smoothness functional*. The regularization parameter controls the trade-off between a solution close to the given data in the first term of (41) and the imposed smoothness in the second term. In [25][p. 3-4], smoothness is described as a measure of the "oscillatory" behavior of a function, and it is stated that for a wide range of smoothness functionals the solutions to the risk minimization of (41) all have the same form. Referring to smoothness as oscillatory behavior it is convenient to consider the function in its Fourier (frequency) domain. Comparing two functions, the smoother one will there have less energy in the high frequency components than the other one. In intuitive way of determining the form of the smoothness functional can be derived by considering the function as a signal $\tilde{f}(\boldsymbol{s})$. In order to smoothen the signal, the higher frequencies should be weakened. The power of the signal may be measured by taking the L_2 norm of it. The smoothness functional can then take the form

$$\phi[f] = \int_{R^d} \frac{|\tilde{f}(\boldsymbol{s})|^2}{\tilde{G}(\boldsymbol{s})} \mathrm{d}\boldsymbol{s}, \tag{42}$$

where $\tilde{G}$ is a positive function decreasing towards zero as $||\boldsymbol{s}|| \to \infty$.

3.8.5 Reproducing Kernel Hilbert Space

A Hilbert space $\mathcal{H}$ a complete normed vector space (Banach space) with an inner product defined in it [26]. A normed vector space is simply a vector space in which a norm between vectors is defined. That the vector space is complete can informally be explained as a vector space without any points missing inside or from the boundary of the space[1]. The norm in a Hilbert space can be expressed by the inner product. A common example of a Hilbert space is the Euclidean space, where the inner product is known as the dot product.

A special kind of Hilbert spaces called *reproducing kernel Hilbert space* (RKHS) are of particular interest when solving inverse problems. Let k be symmetric, positive-definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ called the *reproducing kernel*. A RKHS $\mathcal{H}_k$ consists of a set of functions f and an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_k}$ such that [27]

1. k has the reproducing property

$$\langle f, k(x, \cdot) \rangle_{\mathcal{H}_k} = f(x), x \in \mathcal{X}. \tag{43}$$

2. $\mathcal{H}_k$ is the closure of the span of all $k(x, \cdot)$ with $x \in \mathcal{X}$.

These properties ensure that any function in a RKHS can be expressed as a linear combination of the kernel function. This can be related to the norm of the regularization, which will be useful later. The reader is referred to [28][ch. 6.2] for more details regarding this subject.

3.8.6 Representer Theorem

Concluding the mathematical background is a neat theorem motivating the introduction of the RKHS. Solving the regularized empirical risk minimization problem is by itself not straight forward to do as the hypothesis space may be an infinite dimensional space. However, the representer theorem states that when regularising with an empirical risk functional over a RKHS hypothesis space, the optimal risk functional can be represented as a finite linear combination of kernel products evaluated on the sample points from the data studied.

[1] More formally, the completeness of a vector space require that every Cauchy sequence of points in the vector space has a limit also included in the vector space.

4 Point set registration and Coherent Point Drift

In this section, the concept of registration is further explained, and the derivation of the Coherent Point Drift registration algorithm will be covered in depth. As explained in the introduction, registration is the process of finding the correspondences between two data sets. The data to be registered can be of various formats, where images and point sets are two of the most typical data types. The latter will be the concern of this project.

Given two point sets, point set registration (PSR) is the process of determining the geometrical transformation that aligns the points in one point set with the corresponding points in the other point set as optimal as possible. The result of a point set registration is thus a mathematical mapping relating the two point sets [29]. The point sets, also known as point clouds, can in general be of any dimension, but two- and three-dimensional point sets are most common in medical imaging where the point sets usually represent some anatomical structures. In this report, point set registration is the only registration type considered, so these terms might be used interchangeably.

For a more mathematical approach, let $\mathcal{F}$ and $\mathcal{M}$ be two point sets to be registered. Let $\mathcal{F} = \{x_i\}$ be called the *fixed* or *target* point set, and $\mathcal{M} = \{y_i\}$ be called the *moving* or *source* point set, consisting of (cardinality of the point sets) $|\mathcal{F}| = F$ and $|\mathcal{M}| = M$ points, respectively. One might consider the moving point set as a set of points to be moved to align with the fixed points. The problem of registration is to determine a transformation $\mathcal{T}$ that does this as good as possible, or to optimize some parameters $\boldsymbol{\theta}$ that describe the transformation. In other terms one want to make $\mathcal{T}(\mathcal{M}, \boldsymbol{\theta})$ as close to $\mathcal{F}$ as possible. As there are a lot of different ways of measuring the likeness of these point sets, as well as many approaches for determining the optimal parameters, there necessarily exist a lot of methods for registration. Before going into more details on that, it is better to look at different types of registration.

4.1 Types of point set registration

Registration methods are classified by what assumptions one makes about the transformation. It lies in the nature of the registration problem that the registration transformation and its properties is unknown a priori, yet one can put constraints on the result to determine the approach of the search for it. A rough classification divides registration methods into *rigid* and *non-rigid*, based on whether the transformation can be described as a rigid transformation or not. Rigid registrations can handle rotations and translations, and in some cases isotropic scaling is included here, even though it strictly speaking insinuates a similitude transformation. A common non-rigid registration type is *affine*, where the transformation is affine, which adds (anisotropic) scaling, reflection and shearing to rigid operations. A general non-rigid registration, omitting affine registrations, is the most general transformation and does not put any restrictions on the transformation matrix, making it possible to register deformed objects. A way to represent such non-rigid registration transformations is by the initial object plus a displacement function applied to it. In this project the rigid and affine registrations are implemented as in [2], where isotropic scaling is included in the rigid registration. In CPD and several other works on point set PSR algorithms, the term non-rigid is used registrations that strictly speaking are non-affine. Registration that are neither rigid nor affine will here be termed *elastic* or *deformable*.

In medical application the point sets usually represent anatomical structures of a patient and one can classify the registration as *intrapatient registration* or *interpatient registration*. When both point sets represent the same patient, the problem is *intrapatient registration*, while *interpatient registration* is when the point sets to be registered are acquired from different patients [29]. Using registration to combine images of a patient acquired with different medical imaging modalities, such as computed tomography (CT) and magnetic resonance (MR), is an example of intrapatient registration. Another example is registration of images taken from different angles. In this project the main application is the interpatient registration of a model of the abdominal region extracted from a CT scan to ultrasound images of a patient.

4.2 Evaluating a registration result

In order to evaluate registration results, determining how successful the registration was, one needs some kind of performance metric. The easiest case to consider is when the correspondences between the point sets are known in advance, typically applying a known transformation a point set and compare it itself without the transformation. A registration error can be defined either by comparing the resulting transformation with the applied transformation, or by comparing the point sets directly. There are several ways of defining the error, or distance, between the true and the registered result, including the euclidean distance and root-mean-squared error. In [30], it was found there has been no common metric for registration error among different registration algorithm tests. Following the practice of Myronenko and Song in [2], the norm of difference of the transformation matrices is used as a quantitative registration error measurement for rigid and affine registrations.

When noise and outliers is introduced in a controlled manner to the data sets, the registration error may be calculated by removing the noise and the outliers in the transformed data set. Thus, one can make an estimation of how the registration method performs in presence of true noise and outliers. For data sets from sources such as cameras and sensors, there will always be some noise, missing and non-existing points. Without any true data set representation, the task of determining the registration error is more difficult. In this project, this will not not be an area of focus, so a mere visual assessment will suffice as an evaluation of how well a registration method works.

4.3 Point set registration algorithms

The importance of the point set registration problem is evident considering the number of different algorithms to solve it. In [30] Maiseli, Gu and Gao presented an overview of the state-of-the-art point set registration methods in 2017. A brief introduction to some of these methods and the development of PSR algorithms will be presented here.

One of the most widespread PSR algorithms is Iterative Closest Point (ICP) developed by Besl and McKay [31] and Zhang [32]. ICP provides a simple rigid registration algorithm with low computational complexity. For every source point the closest target point is assumed to correspond to it. The transformation parameters (rotation, translation) are estimated by minimizing a point to point error quantity such as the squared euclidean norm. This process is repeated until convergence. In order to avoid local minima, ICP requires that the point sets are adequately close prior to the registration. Several modified ICP algorithms exist, improving different parts of the algorithm, including weighting of the correspondences and outlier rejection.

The rigid registration algorithm Robust Point Matching (RPM) introduced by Gold et. al [5] use a probabilistic method to remedy ICP's troubles with being trapped in local minima. RPM manages to make the minimization problem convex by a deterministic annealing method.[2] In this and other probabilistic algorithms the hard (binary) correspondence assignments are replaced with soft assignments, describing the probability of correspondence between a source point and any target point.

The thin plate spline robust point matching (TPS-RPM) algorithm developed by Chui et. al [4] goes beyond the rigid and affine limitations of the ICP and RPM algorithms and provides non-rigid registration in three dimensions. Here a parameterization of the source point set based on thin plate spline (TPS) is used because it can be decomposed into affine and non-affine components. This parameterization is however not possible in higher dimensions. Chui et. al's algorithm also required input on two free parameters, for regularization and outlier rejection, which can be difficult to determine.

In [33], Chui et. al show that point set registration can be interpreted as mixture density problem where the expectation maximization (EM) algorithm is used to solve the correspondences, for both rigid and non-rigid cases. They treat one point set as the centroids of a GMM with

[2] Annealing is a concept from statistical mechanics, involving free energy regarded as a temperature-depending cost function. The problem is to choose the optimal temperature (parameter) for minimal cost. Deterministic annealing is a optimization technique that is able to avoid local minima in the cost function.

isotropic covariances and the other point set as data points. Including an extra distribution to the mixture model to account for outliers and use of deterministic annealing to avoid local minima. The probabilistic approach based on GMM is the basis of most state-of-the-art PSR algorithms, and Maiseli, Gu and Gao [30] list three reasons for this: 1) several real-world problems can be effectively modeled as GMMs, 2) GMMs are easier to interpret and formulate, and 3) closed-form solutions exist for point sets in lower dimensions, hence making the algorithms computationally efficient.

One example of a GMM-based PSR method is the one proposed by Jian and Vemuri [34]. Their algorithm is a GMM version of the Kernel Correlation (KC) algorithm by Tsin and Kanade [35]. KC maximizes the correlation between the point sets by the use of a kernel function. For the non-rigid case the GMM version uses TPS for parameterization.

Coherent Point Drift (CPD), introduced by Myronenko and Song [2], is another PSR algorithm relying on GMMs. The source points act as GMM centroids and are iteratively aligned to the target points, maximizing the likelihood by EM. CPD forces the point sets to move coherently as a group, preserving the topological structure of the points. This algorithm generalized to any dimension and is robust even with noise and outliers present. A free parameter concerning the amount of noise must be estimated to obtain good results.

Stochastic global optimization (SGO) for robust PSR by Papazov and Burschka [36] converts to registration problem to the problem of minimizing a non-linear cost function. The cost function is based on an inverse distance kernel reducing the effect of noise and outliers to the model. Global optimum is found by a stochastical method developed by Parazov and Burschka. The SGO algorithm can struggle when there are many closely related local minima, for instance a smaller detail on a larger object, like a small handle on a big cup, but is in general very robust to noise and outliers.

Many registration algorithms do not manage to register complex surfaces missing clear structures or landmarks. Another GMM-based registration algorithm here is the direct point-based method using Student's-t mixture model (DSMM) by Zhou et. al [37], which is a non-rigid surface registration method aimed at solving this problem. The GMM from the methods previously mentioned is replaced by a Student's-t mixture model. The heavier tails of the Student's-t distribution, compared to the Gaussian, is found to better model the degradation of the point sets.

In the related work section 1.2, some of the registration most comparable to the FAST CPD algorithm developed here are introduced; Extended CPD [38], Maximum Likelihood Mixture Decoupling (MLMD) [39], GMM-Tree [7] and FilterReg [40], in addition to PR-Net [8], which is based on neural networks. Most of them provide some kind of parallelization, but the MLMD and GMM-Tree only provided for rigid registration. An advantage of the CPD algorithm is that rigid, affine and elastic registrations are unified by the same mathematical framework.

4.4 Coherent Point Drift

Coherent Point Drift (CPD) is a point set registration algorithm developed by Andriy Myronenko and Xubo Song who published a paper about it in 2009 [2]. CPD includes rigid, affine and elastic registration, and performs well on data with noise, outliers and missing points. The algorithm builds on GMMs and EM, which is covered in previous sections. The registration of two point sets is considered a density estimation problem (see section 3.1.1). One point set is regarded as the centroids of GMM components and the other point set as the observations generated by that GMM. The correspondences between points from the point sets are found by calculating the responsibilities with maximum likelihood estimation. The word *coherent* in CPD describe the movement of the centroids: They are forced to move coherently as a group during alignment. This preserves the topological structure of the point sets.

The main motivation for optimizing and parallelizing the Coherent Point Drift algorithm is that it is a well-tested and wide-spread registration algorithm which provides all the registration types: rigid, affine and elastic. In addition, the writer has not been able to find an open-source optimal CPD implementation. With existing open-source implementations without GPU parallelization

it is easy to compare the achieved speedup. As seen in the previous section, many other PSR algorithm build upon CPD as well.

4.4.1 Derivation of general responsibility matrix

Let $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M)^T$ be a $M \times D$ matrix representing the moving point set, and $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^T$ a $N \times D$ matrix representing the fixed points. The moving point set represent the centroids of a GMM with M components, described by (7), making $\boldsymbol{\mu}_m = \boldsymbol{y}_m$. The fixed point set is considered a set of N observations originating from the GMM. In addition to the M Gaussian components, a uniform distribution $p(\boldsymbol{x}|M+1) = 1/N$ is included in the model to account for noise and outliers. Outliers are points where there does not exist any correspondence to the other point set. All GMM components are assigned a common isotropic covariance $\sigma_1^2 = \cdots = \sigma_M^2 = \sigma^2$ and equal mixing coefficients $\phi_1 = \cdots = \phi_M = 1/M$. Letting the covariances be equal not only simplifies the calculations, it also prevents avoiding singularities where one component variance may come too close to zero and blow up the probability [41, p. 63]. The composite distribution GMM+uniform can itself be considered a mixture model with $k = 2$ components. In this mixture the uniform component is assigned a mixing coefficient, or weight, $\Phi_1 = w$, and following (8) the GMM component get a mixing coefficient $\Phi_2 = 1 - w$. The total mixture model is now given by

$$p(\boldsymbol{x}) = \sum_{k=1}^{2} \Phi_k p(x|k) = w\frac{1}{N} + (1-w)\sum_{m=1}^{M} \frac{1}{M}\mathcal{N}(\boldsymbol{x}|\boldsymbol{y}_k, \sigma). \tag{44}$$

Assuming the fixed data points are independent and identically distributed, the total model for whole set of fixed point $\boldsymbol{X}$ is given by

$$p(\boldsymbol{X}) = \prod_{N=1}^{N} p(\boldsymbol{x}_n). \tag{45}$$

In the process of registrating two point sets a transformation $\mathcal{T}$ will give the correspondences between them. The GMM centroids are reparameterized by this transformation with a set of parameters θ: $\boldsymbol{y}_m = \mathcal{T}(\boldsymbol{y}_m, \theta)$. These parameters vary for different registration types. The transformation parameters and the mixture variances are found by maximum likelihood estimation, which is equivalent to minimizing the cost

$$NLL(\theta, \sigma) = -\sum_{n=1}^{N} \ln p(\boldsymbol{x}_n). \tag{46}$$

For solving maximum likelihood for this system, the iterative approach of the Expectation Maximization (EM) algorithm is used. The responsibilities γ_{mn}, or posterior probabilities, calculated in the E-step give the probability of moving point $\boldsymbol{y}_m$ corresponding to fixed point $\boldsymbol{x}_n$. From equation (9) and (11), the responsibilities in this model is calculated to be

$$\begin{aligned}
\gamma_{mn} = \frac{p(C_m)p(\boldsymbol{x}_n|C_m)}{p(\boldsymbol{x}_n)} &= \frac{w\frac{1}{N} + (1-w)\frac{1}{M}\mathcal{N}(\boldsymbol{x}_n|C_m)}{w\frac{1}{N} + (1-w)\frac{1}{M}\sum_{k=1}^{M}\mathcal{N}(\boldsymbol{x}_n|C_k)} \\
&= \frac{\frac{N}{w}\frac{1}{c}\exp\left(\|\boldsymbol{x}_n - \mathcal{T}(\boldsymbol{y}_m, \theta)\|^2\right)}{\frac{N}{w}\frac{1}{c}\left[\sum_{k=1}^{M}\exp\left(\|\boldsymbol{x}_n - \mathcal{T}(\boldsymbol{y}_k, \theta)\|^2\right) + c\right]} \\
&= \frac{\exp\left(\|\boldsymbol{x}_n - \mathcal{T}(\boldsymbol{y}_m, \theta)\|^2\right)}{\sum_{k=1}^{M}\exp\left(\|\boldsymbol{x}_n - \mathcal{T}(\boldsymbol{y}_k, \theta)\|^2\right) + c},
\end{aligned} \tag{47}$$

where $c = (2\pi\sigma)^{D/2}\frac{w}{1-w}\frac{M}{N}$.

Calculating the change in the cost function before and after the parameters are updated, following equations (22)-(25), and discarding the terms not depending on the parameters θ or σ, as

in (26), the cost function Q for this model is

$$Q(\theta, \sigma^2) = \frac{1}{2\sigma^2} \sum_{n=1}^{N} \sum_{m=1}^{M} \left(\gamma_{mn}^{\text{old}} \left\| \boldsymbol{x}_n - \mathcal{T}(\boldsymbol{y}_m, \theta) \right\|^2 \right) + \frac{N_{\boldsymbol{P}} D}{2} \ln \sigma^2, \tag{48}$$

where

$$N_{\boldsymbol{P}} = \sum_{m=1}^{M} N_m \tag{49}$$

can be interpreted as the effective number of observations the components in sum are taking responsibility for. When the weight for the uniform distribution is zero, $w = 0$, and hence $c = 0$ in γ_{mn}, all the fixed points are taken responsibility for, i.e. $N_{\boldsymbol{P}} = N$. As w increases, fewer and fewer observations are included in the registration, $N_{\boldsymbol{P}} \xrightarrow{w \to 1} 0$, reflecting that w is an assumption on the amount of noise and outliers in the data set, which should not participate in finding the registration transformation.

To continue with the maximization step, the transformation $\mathcal{T}$ must be specified. This will now be done for rigid, affine and elastic transformations separately.

4.5 Rigid CPD

A rigid transformation in CPD may consist of rotation $\boldsymbol{R}$, translation $\boldsymbol{t}$, and isotropic scaling s. $\boldsymbol{R}$ is a $D \times D$ rotation matrix, $\boldsymbol{t}$ is a $D \times 1$ translation column vector and s is a scalar scaling factor. For $\boldsymbol{R}$ to represent a rotation it must belong to the special orthogonal group, implying that its transpose is equal to its inverse and that it has determinant $+1$. Applied to a point $\boldsymbol{y}_m$ the rigid transformation is $\mathcal{T}(\boldsymbol{y}_m; \boldsymbol{R}, \boldsymbol{t}, s) = s\boldsymbol{R}\boldsymbol{y}_m^T + \boldsymbol{t}$. The points are introduced as row vectors, but will be converted to vector representation by their transpose when pre-multiplied by matrices. Inserting this transformation into (48), the objective function for rigid registration is

$$Q(\boldsymbol{R}, \boldsymbol{t}, s, \sigma^2) = \frac{1}{2\sigma^2} \sum_{n=1}^{N} \sum_{m=1}^{M} \left(\gamma_{mn}^{\text{old}} \left\| \boldsymbol{x}_n^T - (s\boldsymbol{R}\boldsymbol{y}_m^T + \boldsymbol{t}) \right\|^2 \right) + \frac{N_{\boldsymbol{P}} D}{2} \ln \sigma^2, \tag{50}$$

where $\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I}, \det(\boldsymbol{R}) = 1$. These constraints on the rotation matrix complicates least squares minimization of $\sum_{n=1}^{N} \gamma_{mn} \left\| \boldsymbol{x}_n^T - (s\boldsymbol{R}\boldsymbol{y}_m^T + \boldsymbol{t}) \right\|^2$ in the objective function with respect to R, yet it is still possible to find a closed form solution. The approach of Myronenko and Song [2] requires that the objective function is on the form $\text{tr}(\boldsymbol{A}^T\boldsymbol{R})$, where tr is the trace of a matrix, $\boldsymbol{A}$ is a known $D \times D$ matrix and $\boldsymbol{R}$ is the D-dimensional rotation matrix.

At first, the objective function can be minimized with respect to the translation vector $\boldsymbol{t}$. By differentiating (50) with respect to $\boldsymbol{t}$,

$$\frac{\partial Q}{\partial \boldsymbol{t}} = -\frac{1}{\sigma^2} \sum_{n=1}^{N} \sum_{m=1}^{M} \gamma_{mn} (\boldsymbol{x}_n^T - s\boldsymbol{R}\boldsymbol{y}_m^T - \boldsymbol{t}), \tag{51}$$

equating it to zero and solving it for $\boldsymbol{t}$ the solution is

$$\boldsymbol{t} = \frac{1}{N_{\boldsymbol{P}}} \left[\sum_{n=1}^{N} \left(\sum_{m=1}^{M} \gamma_{mn} \right) \boldsymbol{x}_n^T \right] - s\boldsymbol{R} \frac{1}{N_{\boldsymbol{P}}} \left[\sum_{m=1}^{M} \left(\sum_{n=1}^{N} \gamma_{mn} \right) \boldsymbol{y}_m^T \right]. \tag{52}$$

The sum $\sum_{m=1}^{M} \gamma_{mn}$ of all the M component responsibilities corresponding to point n, appearing in the first term, is equivalent to take the sum of the elements in the n-th column in the responsibility matrix $\boldsymbol{P}$, representing the probability of $\boldsymbol{x}_n$ originating from the given model. Summing up all these N weighted positions $\sum_{m=1}^{M} \gamma_{mn} \boldsymbol{x}_n^T$ and dividing the sum by the effective number of points originating from the model $N_{\boldsymbol{P}}$, will thereby represent the effective mean of the fixed points resulting from the current mixture model. In matrix representation this mean can be defined as

$$\boldsymbol{\mu}_{\boldsymbol{x}} = \frac{1}{N_{\boldsymbol{P}}} \boldsymbol{X}^T \boldsymbol{P}^T \boldsymbol{1}, \tag{53}$$

where $\mathbf{1}$ is a column vector of ones, here with dimensions $M \times 1$. The second term of (52) contain a corresponding factor for the moving points, where the moving points $\boldsymbol{y}_m$ are weighted by N_m, describing the effective number of fixed points moving point m is taking responsibility of. Summing over all moving point and dividing by $N_{\boldsymbol{P}}$ this represent the effective mean of the centroids in the model. In matrix notation this mean can be defined as

$$\boldsymbol{\mu}_{\boldsymbol{y}} = \frac{1}{N_{\boldsymbol{P}}} \boldsymbol{Y}^T \boldsymbol{P} \mathbf{1}. \tag{54}$$

The moving mean in the second term of (52) is scaled and rotated according to the rigid transformation. The difference between these effective means is the new translation $\boldsymbol{t} = \boldsymbol{\mu}_{\boldsymbol{x}} - s\boldsymbol{R}\boldsymbol{\mu}_{\boldsymbol{y}}$. Inserting this solution into the objective function are rewriting it to

$$Q = \frac{1}{2\sigma^2} \sum_{n=1}^{N} \sum_{m=1}^{M} \gamma_{mn} \left(\left\| (\boldsymbol{x}_n^T - \boldsymbol{\mu}_{\boldsymbol{x}}) - s\boldsymbol{R}(\boldsymbol{y}_m^T - \boldsymbol{\mu}_{\boldsymbol{y}}) \right\|^2 \right) + \frac{N_{\boldsymbol{P}} D}{2} \ln \sigma^2, \tag{55}$$

one can see that the points in the point sets are shifted to be centered around their respective effective means. Let these centered point sets be denoted $\hat{\boldsymbol{X}} = \boldsymbol{X} - \mathbf{1}\boldsymbol{\mu}_{\boldsymbol{x}}^T$ and $\hat{\boldsymbol{Y}} = \boldsymbol{Y} - \mathbf{1}\boldsymbol{\mu}_{\boldsymbol{y}}^T$.

A step toward a matrix representation is to rewrite the norm of the vector subtraction in the objective function as $\left\| \hat{\boldsymbol{x}}_n^T - s\boldsymbol{R}\hat{\boldsymbol{y}}_m^T \right\|^2 = \left\| \hat{\boldsymbol{x}}_n^T \right\|^2 - 2s\hat{\boldsymbol{x}}_n(\boldsymbol{R}\hat{\boldsymbol{y}}_m^T) + s^2 \left\| \boldsymbol{R}\hat{\boldsymbol{y}}_m^T \right\|^2 = \hat{\boldsymbol{x}}_n\hat{\boldsymbol{x}}_n^T - 2s\hat{\boldsymbol{x}}_n^T(\hat{\boldsymbol{y}}_m\boldsymbol{R}^T) + s^2\hat{\boldsymbol{y}}_m\hat{\boldsymbol{y}}_m^T$, where the orthogonality property $\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I}$ of $\boldsymbol{R}$ is used. [3] It is desirable to replace the sum signs in (55) by traces involving the point set matrices. The relation $\operatorname{tr}(\boldsymbol{X}^T\boldsymbol{X}) = \sum_{n=1}^{N} \boldsymbol{x}_n^T\boldsymbol{x}_n$ provides a starting point. [4] In this case one must also consider that the norm terms, $\|\hat{\boldsymbol{x}}_n\|^2$ and $\|\hat{\boldsymbol{y}}_m\|^2$, are weighted by a sums of responsibilities, $\sum_{m=1}^{M} \gamma_{mn}$ and $\sum_{n=1}^{N} \gamma_{mn}$ respectively. The first sum is the column-wise sum of the column belonging to fixed point $\boldsymbol{x}_n$ in the responsibility matrix. Repeating this for all N fixed points results in the column-wise sum of the responsibility matrix, which can be described by $\boldsymbol{P}^T\mathbf{1}$. A trick to include this weight in the trace is to express it as a diagonal matrix $\mathrm{d}(\boldsymbol{P}^T\mathbf{1})$ applied to one of the point sets. In this manner each norm $\hat{\boldsymbol{x}}_n\hat{\boldsymbol{x}}_n^T$ will be weighted by the appropriate responsibility. These last steps can be summarized as

$$\sum_{n=1}^{N} \sum_{m=1}^{M} \gamma_{mn} \left\| \hat{\boldsymbol{X}}_n \right\|^2 = \operatorname{tr}\left(\hat{\boldsymbol{X}}^T \mathrm{d}(\boldsymbol{P}^T\mathbf{1}) \hat{\boldsymbol{X}} \right). \tag{56}$$

The term in (55) regarding $\|\boldsymbol{R}\hat{\boldsymbol{y}}_m\|^2$ can be rewritten in a similar manner with a row-wise sum of the responsibility matrix, $\boldsymbol{P}\mathbf{1}$, as well as the square of the scaling parameter s. The mixed fixed and moving term is more complicated, but carefully writing out the elements one find

$$\operatorname{tr}\left(\hat{\boldsymbol{X}}^T \boldsymbol{P} \hat{\boldsymbol{Y}} \boldsymbol{R} \right) = \sum_{n=1}^{N} \sum_{m=1}^{M} \gamma_{mn} \hat{\boldsymbol{x}}_n^T \hat{\boldsymbol{y}}_m \boldsymbol{R} = \sum_{n=1}^{N} \sum_{m=1}^{M} \boldsymbol{P}_{mn} \hat{\boldsymbol{x}}_n^T \hat{\boldsymbol{y}}_m \boldsymbol{R}. \tag{57}$$

With this matrix representation the objective function is now given by

$$\begin{aligned} Q =& \frac{1}{2\sigma^2} \sum_{n=1}^{N} \sum_{m=1}^{M} \gamma_{mn} \left(\left\| \hat{\boldsymbol{x}}_n^T - s\boldsymbol{R}\hat{\boldsymbol{y}}_m^T \right\|^2 \right) + \frac{N_{\boldsymbol{P}} D}{2} \ln \sigma^2 \\ =& \frac{1}{2\sigma^2} \left[\operatorname{tr}\left(\hat{\boldsymbol{X}}^T \mathrm{d}(\boldsymbol{P}^T\mathbf{1}) \hat{\boldsymbol{X}} \right) - 2s \operatorname{tr}\left(\hat{\boldsymbol{X}}^T \boldsymbol{P}^T \hat{\boldsymbol{Y}} \boldsymbol{R}^T \right) + s^2 \operatorname{tr}\left(\hat{\boldsymbol{Y}}^T \mathrm{d}(\boldsymbol{P}\mathbf{1}) \hat{\boldsymbol{Y}} \right) \right] + \frac{N_{\boldsymbol{P}} D}{2} \ln \sigma^2. \end{aligned} \tag{58}$$

Returning the attention to the rotation matrix, there is only one term containing $\boldsymbol{R}$ in the last expression for the objective function, i.e. $-2s \operatorname{tr}(\hat{\boldsymbol{X}}^T \boldsymbol{P}^T \hat{\boldsymbol{Y}} \boldsymbol{R}^T)$. Remembering that the method of Myronenko and Song requires the form $\operatorname{tr}(\boldsymbol{A}^T\boldsymbol{R})$, it is convenient to define

$$A \equiv \hat{\boldsymbol{X}}^T \boldsymbol{P}^T \hat{\boldsymbol{Y}}. \tag{59}$$

[3] For two column vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of equal dimension, $\|\boldsymbol{a} - \boldsymbol{b}\|^2 = (\boldsymbol{a} - \boldsymbol{b})^T(\boldsymbol{a} - \boldsymbol{b}) = \boldsymbol{a}^T\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{b} + \boldsymbol{b}^T\boldsymbol{a} + \boldsymbol{b}^T\boldsymbol{b} = \|\boldsymbol{a}\|^2 - 2\boldsymbol{a}^T\boldsymbol{b} + \|\boldsymbol{b}\|^2$, and $(\boldsymbol{a}^T\boldsymbol{b})^T = \boldsymbol{a}\boldsymbol{b}^T$.

[4] By writing out the matrix product $\boldsymbol{X}^T\boldsymbol{X}$ it is clear that diagonal element i is equivalent to the norm of the corresponding row vector $\boldsymbol{x}_i$ of $\boldsymbol{X}$, i.e. $\boldsymbol{x}_i^T\boldsymbol{x}_i$. The trace of this matrix product thus gives the sum of the norms of row vectors.

The term then becomes $-2\,\mathrm{tr}\left(\hat{\boldsymbol{X}}^T\boldsymbol{P}^T\hat{\boldsymbol{Y}}\boldsymbol{R}^T\right) = -2\,\mathrm{tr}\left[\left(\hat{\boldsymbol{X}}^T\boldsymbol{P}^T\hat{\boldsymbol{Y}}\right)^T\boldsymbol{R}\right] = -2\,\mathrm{tr}\left(\boldsymbol{A}^T\boldsymbol{R}\right)$. Minimizing Q with respect to R imply minimizing this term, or equivalently maximizing $\mathrm{tr}\left(\boldsymbol{A}^T\boldsymbol{R}\right)$. This specific maximization problem is covered by the following lemma:

Lemma 1. *Let $\boldsymbol{R}_{D\times D}$ be an unknown rotation matrix and $\boldsymbol{A}_{D\times D}$ be a known real square matrix. Let $\boldsymbol{USSV}^T$ be a Singular Value Decomposition (SVD) of $\boldsymbol{A}$, where $\boldsymbol{UU}^T = \boldsymbol{VV}^T = \boldsymbol{I}$ and $SS = \mathrm{d}(s_i)$ with $s_1 \geq s_2 \geq, \ldots, \geq s_D \geq 0$. Then the optimal rotation matrix $\boldsymbol{R}$ that maximizes $\mathrm{tr}(\boldsymbol{A}^T\boldsymbol{R})$ is $\boldsymbol{R} = \boldsymbol{UCV}^T$, where $\boldsymbol{C} = \mathrm{d}(1, 1, \ldots, \det(\boldsymbol{UV}^T))$.*

A general proof of Lemma 1, as well as some history about the problem, is published in "On the closed-form solution of the rotation matrix arising in computer vision problems" (2009) by Myronenko and Song [42]. A crude outline of the proof consists of Lagrange multipliers are used to include the orthogonality of R and its positive determinant, and then Singular Value Decomposition as a part of the approach to find the solution for $\boldsymbol{R}$. Further details will not be provided here, yet this lemma provides an expression for the rotation matrix R that minimizes Q:

$$\boldsymbol{R} = \boldsymbol{UCT}^T, \tag{60}$$

where $\boldsymbol{USSV}^T = \mathrm{svd}(\boldsymbol{A})$, and $\boldsymbol{C} = \mathrm{d}(1, \ldots, 1, \det(\boldsymbol{UV}^T))$.

Minimizing the objective function with respect to s and σ^2 is fortunately more trivial. Differentiating Q with respect to s give

$$\frac{\partial Q}{\partial s} = -\frac{1}{\sigma^2}\,\mathrm{tr}\left(\boldsymbol{A}^T\boldsymbol{R}\right) + \frac{s}{\sigma^2}\,\mathrm{tr}\left(\hat{\boldsymbol{Y}}^T\mathrm{d}(\boldsymbol{P1})\hat{\boldsymbol{Y}}\right), \tag{61}$$

and equating this to zero and solving for s yield

$$s = \frac{\mathrm{tr}\left(\boldsymbol{A}^T\boldsymbol{R}\right)}{\mathrm{tr}\left(\hat{\boldsymbol{Y}}^T\mathrm{d}(\boldsymbol{P1})\hat{\boldsymbol{Y}}\right)}. \tag{62}$$

Using this expression for one of the s-es in (58) the objective function can now be written as

$$Q = \frac{1}{2\sigma^2}\left[\mathrm{tr}(\hat{\boldsymbol{X}}^T\mathrm{d}(\boldsymbol{P}^T\boldsymbol{1})\hat{\boldsymbol{X}}) - s\,\mathrm{tr}\left(\boldsymbol{A}^T\boldsymbol{R}\right)\right] + \frac{N_{\boldsymbol{P}}D}{2}\ln\sigma^2. \tag{63}$$

Differentiating with respect to the variance, equating to zero and solving for the variance, will result in

$$\sigma^2 = \frac{1}{N_{\boldsymbol{P}}D}\left[\mathrm{tr}(\hat{\boldsymbol{X}}^T\mathrm{d}(\boldsymbol{P}^T\boldsymbol{1})\hat{\boldsymbol{X}}) - s\,\mathrm{tr}\left(\boldsymbol{A}^T\boldsymbol{R}\right)\right]. \tag{64}$$

At this point there exist updated expressions for all model parameters t, R, s and σ^2. Finding these is the M step of the EM algorithm. The rigid point set registration algorithm of CPD is summarized in figure 2.

4.6 Affine CPD

The affine transformation of a point $\boldsymbol{y}$ is $\mathcal{T}(\boldsymbol{y}; \boldsymbol{R}, \boldsymbol{t}, s) = \boldsymbol{By} + t$, where $\boldsymbol{B}_{D\times D}$ is an affine transformation matrix, and $\boldsymbol{t}_{D\times 1}$ is a translation vector. The procedure for finding the maximization parameters in the affine case is the same as for rigid, but more straightforward, as there are no extra restrictions on the transformation matrix. The objective function now looks like

$$Q(\boldsymbol{B}, \boldsymbol{t}, \sigma^2) = \frac{1}{2\sigma^2}\sum_{n=1}^{N}\sum_{m=1}^{M}\left(\gamma_{mn}^{\mathrm{old}}\left\|\boldsymbol{x}_n^T - (\boldsymbol{By}_m^T + \boldsymbol{t})\right\|^2\right) + \frac{N_{\boldsymbol{P}}D}{2}\ln\sigma^2, \tag{65}$$

Differentiating (65) with respect to $\boldsymbol{t}$, equating it to zero and solving for $\boldsymbol{t}$ result in the translation vector

$$\boldsymbol{t} = \boldsymbol{\mu_x} - \boldsymbol{B}\boldsymbol{\mu_y}, \tag{66}$$

Rigid point set registration algorithm

Initialization: $\boldsymbol{R} = \boldsymbol{I}$, $\boldsymbol{t} = 0$, $s = 0$, $0 \leq w \leq 1$, $\sigma^2 = \frac{1}{DNM}\sum_{m=1}^{M} \|\boldsymbol{x}_n - \boldsymbol{y}_m\|^2$.

EM: Repeat until convergence

1. **Expectation**: Calculate responsibility matrix $\boldsymbol{P}$.
$$\gamma_{mn} = \frac{\exp\left(-\|\boldsymbol{x}_n - (s\boldsymbol{R}\boldsymbol{y}_m + \boldsymbol{t})\|^2/(2\sigma^2)\right)}{\sum_{k=1}^{M}\exp\left(-\|\boldsymbol{x}_n - (s\boldsymbol{R}\boldsymbol{y}_m + \boldsymbol{t})\|^2/(2\sigma^2)\right) + (2\pi\sigma^2)^{D/2}\frac{w}{1-w}\frac{M}{N}}$$

2. **Maximization**: Solve for $\boldsymbol{R}$, s, $\boldsymbol{t}$ and σ^2.
$N_{\boldsymbol{P}} = \boldsymbol{1}^T\boldsymbol{P}\boldsymbol{1}$, $\boldsymbol{\mu}_{\boldsymbol{x}} = \frac{1}{N_{\boldsymbol{P}}}\boldsymbol{X}^T\boldsymbol{P}^T\boldsymbol{1}$, $\boldsymbol{\mu}_{\boldsymbol{y}} = \frac{1}{N_{\boldsymbol{P}}}\boldsymbol{Y}^T\boldsymbol{P}\boldsymbol{1}$.
$\hat{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{1}\boldsymbol{\mu}_{\boldsymbol{x}}^T$, $\hat{\boldsymbol{Y}} = \boldsymbol{Y} - \boldsymbol{1}\boldsymbol{\mu}_{\boldsymbol{y}}^T$
$\boldsymbol{A} = \hat{\boldsymbol{X}}^T\boldsymbol{P}^T\hat{\boldsymbol{Y}}$
Compute SVD of $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{S}\boldsymbol{V}^T$
$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{C}\boldsymbol{V}^T$, where $\boldsymbol{C} = \mathrm{d}(1, \ldots, 1, \det(\boldsymbol{U}\boldsymbol{V}^T))$
$s = \frac{\mathrm{tr}\left(\boldsymbol{A}^T\boldsymbol{R}\right)}{\mathrm{tr}\left(\hat{\boldsymbol{Y}}^T\mathrm{d}(\boldsymbol{P}\boldsymbol{1})\hat{\boldsymbol{Y}}\right)}$
$\boldsymbol{t} = \boldsymbol{\mu}_{\boldsymbol{x}} - s\boldsymbol{R}\boldsymbol{\mu}_{\boldsymbol{y}}$
$\sigma^2 = \frac{1}{N_{\boldsymbol{P}}D}\left[\mathrm{tr}(\hat{\boldsymbol{X}}^T\mathrm{d}(\boldsymbol{P}^T\boldsymbol{1})\hat{\boldsymbol{X}}) - s\,\mathrm{tr}\left(\boldsymbol{A}^T\boldsymbol{R}\right)\right]$

Result: The aligned point set is $\mathcal{T}(\boldsymbol{Y}) = s\boldsymbol{Y}\boldsymbol{R}^T + \boldsymbol{1}\boldsymbol{t}^T$.

Figure 2: Rigid point set registration algorithm, CPD.

where $\boldsymbol{\mu}_{\boldsymbol{x}}$ and $\boldsymbol{\mu}_{\boldsymbol{y}}$ are defined as in the rigid case by (53) and (54), respectively. Inserting this into (65) yield a shift of the points to $\hat{\boldsymbol{x}}_n^T = \boldsymbol{x}_n^T - \boldsymbol{\mu}_{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}_m^T = \boldsymbol{y}_m^T - \boldsymbol{\mu}_{\boldsymbol{y}}$, and now the objective function may be expressed as

$$\begin{aligned} Q(\boldsymbol{B}, \boldsymbol{t}, \sigma^2) &= \frac{1}{2\sigma^2}\sum_{n=1}^{N}\sum_{m=1}^{M}\left(\gamma_{mn}^{\text{old}}\left\|\hat{\boldsymbol{x}}_n^T - (\boldsymbol{B}\hat{\boldsymbol{y}}_m^T)\right\|^2\right) + \frac{N_{\boldsymbol{P}}D}{2}\ln\sigma^2 \\ &= \frac{1}{2\sigma^2}\sum_{n=1}^{N}\sum_{m=1}^{M}\left[\gamma_{mn}^{\text{old}}\left(\left\|\hat{\boldsymbol{x}}_n^T\right\|^2 - 2\hat{\boldsymbol{x}}_n\boldsymbol{B}\hat{\boldsymbol{y}}_m^T + \left\|\boldsymbol{B}\hat{\boldsymbol{y}}_m^T\right\|^2\right)\right] + \frac{N_{\boldsymbol{P}}D}{2}\ln\sigma^2. \end{aligned} \tag{67}$$

Differentiating with respect to the affine transformation matrix $\boldsymbol{B}$,

$$\frac{\partial Q}{\partial \boldsymbol{B}} = \frac{1}{2\sigma^2}\sum_{n=1}^{N}\sum_{m=1}^{M}\gamma_{mn}\left[-2\frac{\partial}{\partial \boldsymbol{B}}\left(\hat{\boldsymbol{x}}_n\boldsymbol{B}\hat{\boldsymbol{y}}_m^T\right) + \frac{\partial}{\partial \boldsymbol{B}}\left\|\boldsymbol{B}\hat{\boldsymbol{y}}_m^T\right\|^2\right]. \tag{68}$$

To help simplify these matrix derivations a couple of useful relations are found in [43]. The differentiation in the first term of (68) is

$$\frac{\partial}{\partial \boldsymbol{B}}\left(\hat{\boldsymbol{x}}_n\boldsymbol{B}\hat{\boldsymbol{y}}_m^T\right) = \hat{\boldsymbol{y}}_m^T\hat{\boldsymbol{x}}_n, \tag{69}$$

using equation (71) in [43]. The second term is

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{B}}\left\|\boldsymbol{B}\hat{\boldsymbol{y}}_m^T\right\|^2 &= \frac{\partial}{\partial \boldsymbol{B}}\left[\left(\boldsymbol{B}\hat{\boldsymbol{y}}_m^T\right)^T\left(\boldsymbol{B}\hat{\boldsymbol{y}}_m^T\right)\right] \\ &= \frac{\partial}{\partial \boldsymbol{B}}\left[\hat{\boldsymbol{y}}_m\boldsymbol{B}^T\boldsymbol{B}\hat{\boldsymbol{y}}_m^T\right] \\ &= \boldsymbol{B}\hat{\boldsymbol{y}}_m^T\hat{\boldsymbol{y}}_m + \hat{\boldsymbol{y}}_m\hat{\boldsymbol{y}}_m^T \\ &= 2\boldsymbol{B}\left\|\hat{\boldsymbol{y}}_m^T\right\|^2, \end{aligned} \tag{70}$$

using equation (77) in [43] for the differentiation from the second to the third line. Inserting these two differentiation terms into (68) and equating it zero the result is

$$\sum_{n=1}^{N}\sum_{m=1}^{M}\left(\gamma_{mn}^{\text{old}}\hat{\boldsymbol{y}}_m^T\hat{\boldsymbol{x}}_n\right) = \sum_{n=1}^{N}\sum_{m=1}^{M}\left(\gamma_{mn}^{\text{old}}\boldsymbol{B}\hat{\boldsymbol{y}}_m\hat{\boldsymbol{y}}_m^T\right) \tag{71}$$

The matrix at the left-hand side is equivalent to $\hat{\boldsymbol{X}}^T\boldsymbol{P}^T\hat{\boldsymbol{Y}}$, and the matrix at the right-hand side equivalent to $\boldsymbol{B}\hat{\boldsymbol{Y}}^T\mathrm{d}(\boldsymbol{P}\boldsymbol{1})\hat{\boldsymbol{Y}}$. Postmultiplying both sides of the equation by the inverse of the last part of the latter expression to, the following expression for the optimal $\boldsymbol{B}$ is revealed:

$$\boldsymbol{B} = \left(\hat{\boldsymbol{X}}^T\boldsymbol{P}^T\hat{\boldsymbol{Y}}\right)\left(\hat{\boldsymbol{Y}}^T\mathrm{d}(\boldsymbol{P}\boldsymbol{1})\hat{\boldsymbol{Y}}\right)^{-1}. \tag{72}$$

At last, differentiating with respect to the variance and equating it to zero, the updated variance is given by

$$\sigma^2 = \frac{1}{N_{\boldsymbol{P}}D}\left[\mathrm{tr}(\hat{\boldsymbol{X}}^T\mathrm{d}(\boldsymbol{P}^T\boldsymbol{1})\hat{\boldsymbol{X}}) - \mathrm{tr}\left(\hat{\boldsymbol{X}}^T\boldsymbol{P}^T\hat{\boldsymbol{Y}}\boldsymbol{B}^T\right)\right]. \tag{73}$$

By now, expressions for all model parameters $\boldsymbol{t}$, $\boldsymbol{B}$ and σ^2 are derived. Combining all steps compose the affine CPD registration algorithm, summarized in figure 3.

Affine point set registration algorithm

Initialization: $\boldsymbol{B} = \boldsymbol{I}$, $\boldsymbol{t} = 0$, $0 \leq w \leq 1$, $\sigma^2 = \frac{1}{DNM}\sum_{m=1}^{M}\|\boldsymbol{x}_n - \boldsymbol{y}_m\|^2$.

EM: Repeat until convergence

1. **Expectation**: Calculate responsibility matrix $\boldsymbol{P}$.
$$\gamma_{mn} = \frac{\exp\left(-\|\boldsymbol{x}_n-(\boldsymbol{B}\boldsymbol{y}_m+\boldsymbol{t})\|^2/(2\sigma^2)\right)}{\sum_{k=1}^{M}\exp\left(-\|\boldsymbol{x}_n-(\boldsymbol{B}\boldsymbol{y}_m+\boldsymbol{t})\|^2/(2\sigma^2)\right)+(2\pi\sigma^2)^{D/2}\frac{w}{1-w}\frac{M}{N}}$$

2. **Maximization**: Solve for $\boldsymbol{B}$, s, $\boldsymbol{t}$ and σ^2.
$N_{\boldsymbol{P}} = \boldsymbol{1}^T\boldsymbol{P}\boldsymbol{1}$, $\boldsymbol{\mu}_{\boldsymbol{x}} = \frac{1}{N_{\boldsymbol{P}}}\boldsymbol{X}^T\boldsymbol{P}^T\boldsymbol{1}$, $\boldsymbol{\mu}_{\boldsymbol{y}} = \frac{1}{N_{\boldsymbol{P}}}\boldsymbol{Y}^T\boldsymbol{P}\boldsymbol{1}$.
$\hat{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{1}\boldsymbol{\mu}_{\boldsymbol{x}}^T$, $\hat{\boldsymbol{Y}} = \boldsymbol{Y} - \boldsymbol{1}\boldsymbol{\mu}_{\boldsymbol{y}}^T$
$\boldsymbol{B} = (\hat{\boldsymbol{X}}^T\boldsymbol{P}^T\hat{\boldsymbol{Y}})(\hat{\boldsymbol{Y}}^T\mathrm{d}(\boldsymbol{P}\boldsymbol{1})\hat{\boldsymbol{Y}})^{-1}$
$\boldsymbol{t} = \boldsymbol{\mu}_{\boldsymbol{x}} - \boldsymbol{B}\boldsymbol{\mu}_{\boldsymbol{y}}$
$\sigma^2 = \frac{1}{N_{\boldsymbol{P}}D}\left[\mathrm{tr}(\hat{\boldsymbol{X}}^T\mathrm{d}(\boldsymbol{P}^T\boldsymbol{1})\hat{\boldsymbol{X}}) - \mathrm{tr}\left(\hat{\boldsymbol{X}}^T\boldsymbol{P}^T\hat{\boldsymbol{Y}}\boldsymbol{B}^T\right)\right]$

Result: The aligned point set is $\mathcal{T}(\boldsymbol{Y}) = \boldsymbol{Y}\boldsymbol{B}^T + \boldsymbol{1}\boldsymbol{t}^T$.

Figure 3: Affine point set registration algorithm, CPD.

4.7 Elastic CPD

In the rigid and the affine case of registration considered so far, restrictions have been applied to the transformation matrix $\mathcal{T}$ in order to force the registration to conform to specific set of rules. When it is given, or at least likely, that the mapping between the two point sets considered can be described by such rules, the rigid and affine methods are solid. However, there are many cases when such an assumption on the mapping is not suitable, for instance when there might be a deformation on one or both of the point sets. For this case, a more general and flexible registration model is needed. When no assumptions on the form of $\mathcal{T}$ are made, the registration is referred to as elastic[5]. This case proves to be more mathematically advanced to solve than the rigid and affine cases. The main lines of the derivation will be drawn here, but for more details the reader is referred to for instance the supplementary material of ECPD [44].

[5]This method is called non-rigid registration in CPD, but because that term in principle include affine registration, the term elastic registration is used to be more clear

Let the moving points be described by a displacement function v representing the movement of each point. The elastic transformation can then be expressed as an addition of this displacement function to the initial position,

$$\mathcal{T}(\boldsymbol{Y}, v) = \boldsymbol{Y} + v(\boldsymbol{Y}). \tag{74}$$

As before, a function Q to be optimized, i.e. an objective function, must be defined. The displacement function must then be determined in a way such that the objective function is minimized.

Key to the CPD algorithm is the idea of points moving coherently, hence the name of the algorithm, implying that points should tend to move in the same direction as the points nearby. Mathematically, this means that the displacement function should be smooth. Smoothness can be ensured by the use of regularization, by penalizing the NLL cost by a regularization term $\phi(v)$. The objective function is then given by

$$Q(v, \sigma^2) = NLL(v, \sigma^2) + \frac{\lambda}{2}\phi(v), \tag{75}$$

where NLL is the negative log-likelihood given in eq. (46) and λ is the regularization parameter. In the Bayesian view, this regularization term corresponds to the prior $p(v) = \exp\left(-\frac{\lambda}{2}\phi(v)\right)$. See [25][p. 21-23] for an explanation of this.

Using a smoothness functional of the form introduced in section 3.8.4,

$$\phi[f] = \int_{R^d} \frac{|\tilde{f}(\boldsymbol{s})|^2}{\tilde{G}(\boldsymbol{s})} \mathrm{d}\boldsymbol{s}, \tag{76}$$

where $\tilde{G}$ is a positive and symmetric function decreasing towards zero as $||\boldsymbol{s}|| \to \infty$, it can be shown that the solutions to risk minimization problem all have the same form [25][p. 3]. The derivation will not be covered here, but a sketch of the derivation is presented in the appendices of [25] and [45]. In short, the risk functional is expressed in terms of $\tilde{v}$ and then calculus of variations is used to equate the functional derivatives to zero:

$$\frac{\partial Q(\tilde{v}(\boldsymbol{s})}{\partial \tilde{v}(\boldsymbol{t})} = 0, \quad \forall \boldsymbol{t} \in \mathbb{R}^D. \tag{77}$$

Carrying out this calculation leads to the result

$$v(\boldsymbol{x}) = \sum_{m=1}^{M} \boldsymbol{w}_m G(\boldsymbol{x} - \boldsymbol{y}_m) + \psi(\boldsymbol{x}), \tag{78}$$

with

$$\boldsymbol{w}_m = \frac{1}{\sigma^2 \lambda} \sum_{n=1}^{N} P^{\text{old}}(m|\boldsymbol{x}_n) \left[\boldsymbol{x}_n - (\boldsymbol{y}_m + v(\boldsymbol{y}_m))\right], \tag{79}$$

and where $\psi(\boldsymbol{x})$ is a term that lies in the null space of the smoothness functional.

In order to proceed now, the basis function G must be specified. There a many possible choices for the basis function, including radial functions, i.e. functions that are rotationally invariant and only depend on the norm of the input, $G(\boldsymbol{x}) = G(\|\boldsymbol{x}\|)$. In the CPD algorithm, G is chosen to be a Gaussian, $G(\boldsymbol{x}) = \exp\left(-\frac{1}{2\beta^2}\|\boldsymbol{x}\|^2\right)$, and Myronenko et al. list some motivations for this particular choice [45]: The null space term $\psi = 0$, the range of filtered frequencies, and thus the amount of spatial smoothening, can be controlled, and the Gaussian basis function makes the regularization term equivalent to the one used in Motion Coherence Theory (MCT) [46].

Inserting the expression for v into the risk functional results in the objective function of this algorithm:

$$Q(\boldsymbol{W}) = \frac{1}{2\sigma^2} \sum_{n=1}^{N} \sum_{m=1}^{M} \gamma_{mn} \|\boldsymbol{x}_n - \boldsymbol{y}_m - \boldsymbol{G}(m, \cdot)\boldsymbol{W}\|^2 + \frac{\lambda}{2} \operatorname{tr}\left(\boldsymbol{W}^T \boldsymbol{G} \boldsymbol{W}\right), \tag{80}$$

where $\boldsymbol{G}_{M\times M}$ is a square and symmetric Gaussian kernel matrix and $\boldsymbol{W}_{M\times D} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_M)^T$. More precisely, $\boldsymbol{G}$ is a matrix consisting of all inner products between all the moving points (known

as the Gram matrix). Its elements need only be calculated once with the initial moving points of each registration,

$$g_{ij} = \exp\left(-\frac{1}{2\beta^2}\left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\|^2\right), \tag{81}$$

with a chosen variance β^2. $\boldsymbol{G}(m,\cdot)$ refers to the m-th row of $\boldsymbol{G}$.

The matrix W depends on the displacement function v. To determine this matrix the objective function is differentiated with respect to $\boldsymbol{W}$ and the results is equated to zero, $\frac{\partial Q}{\partial \boldsymbol{W}} = 0$. Rearranging the resulting equation yield the following system of equations to be solved:

$$\left(\boldsymbol{G} + \lambda\sigma^2 \mathrm{d}(\boldsymbol{P}\boldsymbol{1})^{-1}\right)\boldsymbol{W} = \mathrm{d}(\boldsymbol{P}\boldsymbol{1})^{-1}\boldsymbol{P}\boldsymbol{X} - \boldsymbol{Y}. \tag{82}$$

This matrix equation must be solved for $\boldsymbol{W}$, and such methods will be discussed in the next chapter. The transformation will then be given by $\boldsymbol{T} = \mathcal{T}(\boldsymbol{Y}, \boldsymbol{W}) = \boldsymbol{Y} + \boldsymbol{G}\boldsymbol{W}$. Solving $\frac{\partial Q}{\partial \sigma^2} = 0$ for the variance, the variance that minimizes the risk is found to be

$$\begin{aligned}\sigma^2 &= \frac{1}{N_{\boldsymbol{P}} D}\sum_{n=1}^{N}\sum_{m=1}^{M}\left\|\boldsymbol{x}_n - \mathcal{T}(\boldsymbol{y}_m, \boldsymbol{W})\right\|^2 \\ &= \frac{1}{N_{\boldsymbol{P}} D}\left[\mathrm{tr}\left(\boldsymbol{X}^T \mathrm{d}(\boldsymbol{P}^T\boldsymbol{1})\boldsymbol{X}\right) - 2\,\mathrm{tr}\left((\boldsymbol{P}\boldsymbol{X})^T\boldsymbol{T}\right) + \mathrm{tr}\left(\boldsymbol{T}^T \mathrm{d}(\boldsymbol{P}\boldsymbol{1})\boldsymbol{T}\right)\right].\end{aligned} \tag{83}$$

Calculating $\boldsymbol{W}$ and the updated σ^2 is the maximization step for the elastic CPD algorithm. All the steps are summarized in figure 4.

Elastic point set registration algorithm

Initialization: $\boldsymbol{W} = 0$, $\sigma^2 = \frac{1}{DNM}\sum_{m=1}^{M}\left\|\boldsymbol{x}_n - \boldsymbol{y}_m\right\|^2$.
Set parameters: $0 \le w \le 1$, $\beta > 0$, $\lambda > 0$.
Calculate $\boldsymbol{G}$: $g_{ij} = \exp\left(-\frac{\left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\|^2}{2\beta^2}\right)$
EM: Repeat until convergence

1. **Expectation**: Calculate responsibility matrix $\boldsymbol{P}$.
   $\gamma_{mn} = \frac{\exp\left(-\left\|\boldsymbol{x}_n - (\boldsymbol{y}_m + \boldsymbol{G}(m,\cdot)\boldsymbol{W})\right\|^2/(2\sigma^2)\right)}{\sum_{k=1}^{M}\exp\left(-\left\|\boldsymbol{x}_n - (\boldsymbol{y}_k + \boldsymbol{G}(k,\cdot)\boldsymbol{W})\right\|^2/(2\sigma^2)\right) + (2\pi\sigma^2)^{D/2}\frac{w}{1-w}\frac{M}{N}}$
2. **Maximization**: Solve for $\boldsymbol{W}$ and σ^2.
   Solve for $\boldsymbol{W}$: $\left(\boldsymbol{G} + \lambda\sigma^2\mathrm{d}(\boldsymbol{P}\boldsymbol{1})^{-1}\right)\boldsymbol{W} = \mathrm{d}(\boldsymbol{P}\boldsymbol{1})^{-1}\boldsymbol{P}\boldsymbol{X} - \boldsymbol{Y}$
   $N_{\boldsymbol{P}} = \boldsymbol{1}^T\boldsymbol{P}\boldsymbol{1}$, $\boldsymbol{T} = \boldsymbol{Y} + \boldsymbol{G}\boldsymbol{W}$
   $\sigma^2 = \frac{1}{N_{\boldsymbol{P}} D}\left[\mathrm{tr}(\hat{\boldsymbol{X}}^T \mathrm{d}(\boldsymbol{P}^T\boldsymbol{1})\hat{\boldsymbol{X}}) - 2\,\mathrm{tr}\left((\boldsymbol{P}\boldsymbol{X})^T\boldsymbol{T}\right) + \mathrm{tr}\left(\boldsymbol{T}^T\mathrm{d}(\boldsymbol{P}\boldsymbol{1})\boldsymbol{T}\right)\right]$

Result: The aligned point set is $\mathcal{T}(\boldsymbol{Y}, \boldsymbol{W}) = \boldsymbol{Y} + \boldsymbol{G}\boldsymbol{W}$.

Figure 4: Elastic point set registration algorithm, CPD.

5 Methods and implementation

In this project the optimized and parallelized version of the rigid, affine and elastic CPD algorithm were implemented and tested. This section will provide details regarding the implementation used in FAST CPD and motivate the different optimizations and parallelizations included. The data acquired for testing FCPD is also introduced in this section. The source code of the FCPD implementation is available in FAST repository at GitHub: `https://github.com/smistad/FAST/tree/master/source/FAST/Algorithms/CoherentPointDrift`. The rigid, affine and elastic implementations are summarized at the end of this chapter.

5.1 Structure of the implementation

The implementation is written in C++ as a part of the framework FAST (introduced in 2.4). The C++ language is an extension of C and includes support for object-oriented programming. This implementation is structured in one virtual base class called CoherentPointDrift, and three subclasses, CoherentPointDriftRigid, CoherentPointDriftAffine and CoherentPointDriftElastic, covering rigid, affine and elastic registration, respectively. A virtual base class implies that no objects are made of that class, but it defines one or more functions that it demands any subclass to have. The three registration algorithms from the original CPD paper [2] are summarized in figures 2, 3 and 2 in the previous chapter, respectively.

The algorithm takes two point sets as input, one defined to be the fixed point set, and the other to be the moving point set. The E-step of the CPD algorithm is common for all three supported registration types. This is therefore implemented in the base class. The subclasses all include an initialization step, and a maximization function to solve the M-step. The output of the algorithm is a homogeneous transformation matrix in the case of rigid and affine registration, and a mesh with updated moving points for the elastic case.

As a stopping criterion for the EM algorithm, the change in variance from iteration to iteration was compared to a defined tolerance. This was motivated by [47] where it was found that the variance is the last parameter to stabilize in the EM algorithm for GMMs.

Note that two optimization techniques were provided in [2], namely the fast Gauss transform, and the low-rank approximation. In the FCPD implementation FGT is replaced by other means and the low-rank approximation is included. More details on these methods will follow in this chapter.

The implementation only depend on one external library, namely Eigen. Eigen is an open source library for linear algebra operations. Calculating for instance the responsibility matrix $\boldsymbol{P}$ and the reduced versions $\boldsymbol{P}^T\mathbf{1}$ and $\boldsymbol{P}\mathbf{1}$ can efficiently be computed by the optimized linear algebra implementations in Eigen. However, matrix multiplications and other matrix operations can always be expressed by for-loops calculating the resulting matrix element by element. This approach is usually much slower than using optimized linear algebra tools, but it has the advantage of being easy parallelizable.

5.2 Parallelization

In short, parallelization is to perform computations simultaneously. In other words, several computations are performed in parallel. Parallelization can reduce computation times quite dramatically and is obviously of special interest in any program where the computational speed matters, for instance processing real-time data. Not any program can be parallelizable. The computations to be performed must be independent of each other. Two types of parallelization is task- and data-parallelism [48, p. 6]. In the first, different tasks are distributed among the computational units and in the latter all units perform the same computations but on different data. The parts of a program that is not parallel is referred to as serial.

In this project a serial implementation of CPD, using linear algebra operations with Eigen is

compared to FCPD which is a partially parallelized implementation of CPD. Parallelization can be performed on both CPUs and GPUs. For CPU parallelization the application programming interface (API) OpenMP has been used, and for GPU OpenCL is used, both of which will be further explained now. Notice that there exist other APIs/frameworks for parallelization as well, but these are well-known.

5.2.1 CPU parallelization and OpenMP

Making an existing serial implementation of a program parallel can be done incrementally using OpenMP (Open Multi-Processing) [48, p. 209]. OpenMP uses several processing units (threads) sharing memory to perform parallel computing. Most computers today have multiple cores that can process data in parallel. Any system with a C/C++ compiler can compile and run code with OpenMP. With OpenMP one can easily, literally by adding a single line of code, parallelize a block of code, for instance for-loops. When the computations performed in a for-loop are independent for every iteration, it is a straightforward code block to parallelize. OpenMP provide mechanisms for distributing data in a for-loop simply by adding the line *#pragma omp parallel for* above the loop. This is used in the CPD implementation when calculating the responsibility matrix, for instance.

5.2.2 GPU parallelization

The use of graphics processing units (GPU) has increased in popularity the last decade due to its superior parallelization abilities. GPUs are built specifically to handle repetitive and relative simple computations rapidly for graphics data. GPUs are therefore built in a parallel structure, with thousands of processing units, compared to CPUs with just a few cores. The use of GPUs for general-purpose computing (GPGPU) have increased in later years. There are several application programming interfaces making it possible to perform desired computations in parallel on GPU. The most known are probably CUDA and OpenCL.

GPUs have their own memory and does not automatically have access to the same data accessed by the CPU. All data that is to be processed on the GPU must therefore explicitly be transferred to the GPU by allocating memory space for a buffer to contain the data on the GPU. The time required to perform this data obviously increases as the size of the data increases. One must therefore be careful to not lose the time saved by parallelization on data transfer. Regarding runtimes, the runtimes of calculations perform in parallel will vary depending on other tasks, such as rendering etc., also performed by the GPU.

5.2.3 OpenCL

Open Computing Language (OpenCL) is a framework for writing program that can run on heterogeneous platforms. The framework consists of a the OpenCL C language and the OpenCL runtime API. The former is a C-based programming language used for writing *kernels* containing the code that will be executed on the compute devices such as the GPU. An algorithm that can perform the same instructions on multiple data elements in parallel is said to be data parallel, and the set of instructions to be executed for each element is called a kernel. [49]. The OpenCL runtime API is used to control and manage the different available devices.

When using GPU kernels, the user should describe which access rights the kernel shall have to the data transferred to it. Typical access rights are read only, write only and read-write. These terms mean exactly as they sound like, implying that a buffered with read-only cannot be written to by the device.

5.3 Data

For the purpose of testing the registration algorithm it is wise to use known data sets, free of noise and outliers. Noise are considered random disturbances of points, making them deviate from

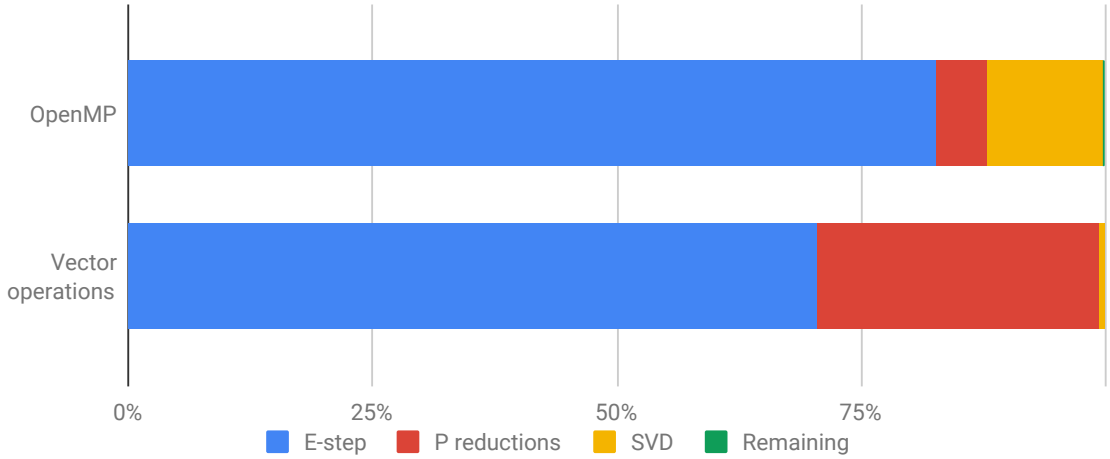


Figure 5: Profiling of rigid CPD implementation without and without OpenMP parallelization.

their true positions. Outliers are point in the point sets not corresponding to any true point of the object. Data sets with only true points are called synthetic data sets. Data acquired from any source will contain some degree of noise and outliers making the registration more difficult to perform and more difficult to evaluate.

5.3.1 Synthetic data

Synthetic data sets are great for evaluating the performance of registration. In a controlled manner one can apply a known transformation to a point set and use registration to try to retrieve the original data. In this project the Stanford bunny data set [50] is used for testing. This is a data set often used for registration evaluation, for instance in the CPD paper [2].

5.3.2 Brain shift data

The data used for brain-shift correction stem from MRA and US Doppler acquisitions. In order to use a point set registration algorithm, the data must be converted by point sets. This can be done by segmenting out the blood vessels from the MRA and US Doppler data, and then extracting the centerlines of the segmented vessels. The two resulting centerlines can be used as moving and fixed data sets. This process is described in figure 1.

5.3.3 Data processing

To make the registration is easy as possible the sometimes pre-processing steps are needed before registration can commence. An example is provided by the brain-shift application, where segmentation and centerline extraction is needed to end up with the registration point sets. In order cases, the data perhaps need to be filtered to remove noise and irrelevant parts of the point sets. If the point sets are very large, they can be downsampled to improve the registration speed. Downsampling can often be done without losing accuracy if the sampling is dense.

5.4 CPD Expectation step: Responsibility kernel calculations

From the code profiling of the rigid CPD, shown in figure 5 from the master project study it is clear that the most time-consuming part of the (rigid) CPD algorithm is the calculating the matrix-vector products $\boldsymbol{P}\mathbf{1}$, $\boldsymbol{P}^T\mathbf{1}$ and $\boldsymbol{PX}$, including the calculation of the responsibility matrix $\boldsymbol{P}$. These calculations are therefore the natural place to start with parallelization in order to minimize the

algorithmic runtime. At first, a glance at the definition of $\boldsymbol{P}$ in (11) reveals that each element of the matrix only depends on the fixed and moving point sets, in addition to determined variables, and thus can be computed independently of each other. Since this is a data parallel operation it is a perfect job for a GPU. Kernel input would then be $N \times D$ fixed point matrix, $M \times D$ moving point matrix, as well as the variance and constants, and the kernel output would be the $M \times N$ responsibility matrix. For large point set sizes the time consumed by reading output from GPU memory is substantial enough to avoid this if possible.

By reviewing the role of the responsibility matrix in the algorithm it is evident that $\boldsymbol{P}$ is never used directly in either of the CPD variants in figures 2, 3, 4. It only appears as the reduced forms $\boldsymbol{P}\mathbf{1}$ and $\boldsymbol{P}^T\mathbf{1}$, of size $M \times 1$ and $N \times 1$, respectively, or in the $M \times D$ matrix-matrix product $\boldsymbol{P}\boldsymbol{X}$. Taking advantage of this, there might be more to gain by performing these computations directly from the fixed and moving point sets $\boldsymbol{X}$ and $\boldsymbol{Y}$ without relying on a pre-calculated responsibility matrix. Beware that the explicit insights of point correlations provided by the responsibility matrix is the cost of this modification.

The denominator in the expression for the responsibility includes a sum over the Gaussians of all the moving points. This denominator will does not depend on the moving point considered, and it therefore not be necessary to calculate this sum more than once for each of the fixed points. Each of the elements of the vector $\boldsymbol{P}^T\mathbf{1}$ calculated by the sum of all elements in one column of $\boldsymbol{P}$. The sum of Gaussians in the nominator is the same as the sum of Gaussian needed in the denominator. Let the Gaussian affinity be defined as

$$K_{mn} = \exp\left(-\frac{1}{2\sigma^2}\left\|\boldsymbol{x}_n - \boldsymbol{y}_m\right\|^2\right). \tag{84}$$

The components of $\boldsymbol{P}^T\mathbf{1}$ can then be expressed as

$$(\boldsymbol{P}^T\mathbf{1})_n = \sum_{m=1}^{M}\left(\frac{K_{mn}}{\sum_{k=1}^{M} K_{kn} + c}\right) = \frac{\sum_{m=1}^{M} K_{mn}}{\sum_{k=1}^{M} K_{kn} + c}, \tag{85}$$

with $n \in \{1, \ldots, N\}$ and $c = (2\pi\sigma^2)^{D/2}\frac{w}{1-w}\frac{M}{N}$. As the sum in the nominator and the denominator is identical the expression in the denominator may be calculated along with the calculations of $\boldsymbol{P}^T\mathbf{1}$ and saved for the calculations of $\boldsymbol{P}\mathbf{1}$. Let the denominator be stored in a N-dimensional vector $\boldsymbol{a}$ such that its n-th component is given by

$$a_n = \sum_{k=1}^{M} K_{kn} + c. \tag{86}$$

Because all the N components of vectors $\boldsymbol{P}^T\mathbf{1}$ and $\boldsymbol{a}$ can be computed independently they can parallelized to N cores. These computations make up the first kernel, simply referred to as GPU Kernel 1, as shown in figure 6.

The m-th component of $\boldsymbol{P}\mathbf{1}$ can now be expressed as

$$(\boldsymbol{P}\mathbf{1})_m = \sum_{n=1}^{N}\left(\frac{K_{mn}}{\sum_{k=1}^{M} K_{kn} + c}\right) = \sum_{n=1}^{N}\left(K_{mn}/a_n\right), \tag{87}$$

with $m \in \{1, \ldots, M\}$. Writing out the matrix-matrix product $\boldsymbol{P}\boldsymbol{X}$ it is clear that its elements are computed in the same manner as the elements of $\boldsymbol{P}\mathbf{1}$ except the affinity terms are weighted by the fixed points as they are summed up:

$$(\boldsymbol{P}\boldsymbol{X})_{md} = \sum_{n=1}^{N}\left(\frac{K_{mn}}{\sum_{k=1}^{M} K_{kn} + c} \cdot x_{nd}\right) = \sum_{n=1}^{N}\left(K_{mn}x_{nd}/a_n\right). \tag{88}$$

The similarity between the calculation of the elements of $\boldsymbol{P}\mathbf{1}$ and $\boldsymbol{P}\boldsymbol{X}$ makes it easy and efficient to distribute these calculations to a kernel with M cores on the GPU. It is also evident that for each moving point m considered all of the dimensions $d \in \{1, dots, D\}$ can be computed in parallel as well, demanding $M \times D$ cores. For the practical cases studied here the number of dimensions

is fixed to $D = 3$. For fixed and smaller dimensions loop unrolling be an efficient alternative, implying that M cores is adequate. Loop unrolling simply means writing out the sum explicitly. The kernel calculations of $\boldsymbol{P1}$ and $\boldsymbol{PX}$ will be denoted GPU Kernel 2, and is summarized in figure 7, using loop unrolling in a 3D-case.

GPU Kernel 1: Calculating $\boldsymbol{P}^T\boldsymbol{1}$ and $\boldsymbol{a}$

Kernel size: N
Input buffers:

- Fixed points $\boldsymbol{X}$ (READ ONLY)
- Moving points $\boldsymbol{Y}$ (READ ONLY)
- Responsibility denominator $\boldsymbol{a}$ (READ-WRITE)
- $\boldsymbol{P}^T\boldsymbol{1}$ (WRITE ONLY)

Input parameters:

- Variance σ^2
- Constant c
- Number of moving points M

Kernel:

n = get_core_id(0)
N = get_kernel_size(0)
affinitySum = 0.0
for $m = 0, \ldots, M-1$ **do**
    norm = $\|\boldsymbol{x}_n - \boldsymbol{y}_m\|^2$
    affinitySum += exp$\left(-\text{norm}/(2 * \sigma^2)\right)$
a_n = affinitySum + c
$(\boldsymbol{P}^T\boldsymbol{1})_n$ = affinitySum / (affinitySum + c)

Figure 6: Kernel for calculating $\boldsymbol{P}^T\boldsymbol{1}$ and $\boldsymbol{a}$ in parallel on the GPU.

5.5 Maximization step rigid and affine CPD

In the CPD algorithm the effective responsibility-weighted means $\boldsymbol{\mu_x}$ and $\boldsymbol{\mu_y}$ of the fixed and moving point sets, respectively, are calculated and then the points sets are centered around these means,

$$\begin{aligned}\hat{\boldsymbol{X}} =& \boldsymbol{X} - \boldsymbol{1}\boldsymbol{\mu}_{\boldsymbol{x}}^T \\ \hat{\boldsymbol{Y}} =& \boldsymbol{Y} - \boldsymbol{1}\boldsymbol{\mu}_{\boldsymbol{y}}^T.\end{aligned} \tag{89}$$

The centered point sets are used for the further calculations of the M step. Firstly, to calculate $\boldsymbol{A} = \hat{\boldsymbol{X}}\boldsymbol{P}^T\hat{\boldsymbol{Y}}$ in the rigid registration, and $\boldsymbol{B} = \boldsymbol{A}\left(\hat{\boldsymbol{Y}}^T \mathrm{d}(\boldsymbol{P1})\hat{\boldsymbol{Y}}\right)^{-1}$. However, these calculations can be made more efficient by omitting the centring and expressing them by the matrix products already calculated in the E-step:

$$\begin{aligned}\boldsymbol{A} =& (\boldsymbol{PX})^T - N_{\boldsymbol{P}}\boldsymbol{\mu_x}\boldsymbol{\mu}_{\boldsymbol{y}}^T \\ \boldsymbol{B} =& \boldsymbol{A}\left(\boldsymbol{Y}^T \mathrm{d}(\boldsymbol{P1})\boldsymbol{Y} - N_{\boldsymbol{P}}\boldsymbol{\mu_y}\boldsymbol{\mu}_{\boldsymbol{y}}^T\right)^{-1}.\end{aligned} \tag{90}$$

Note that the end result of these calculations is identical as the one calculated in the original CPD implementation, just making some of the steps more efficient. After calculating $\boldsymbol{A}$ and $\boldsymbol{B}$, the registration parameters s and $\boldsymbol{t}$ are calculated for the rigid version, and $\boldsymbol{t}$ for the affine registration algorithm, in addition to update the variance in both cases. These calculations can be slightly rewritten using similar re-writings, and these are included in the figures 11 and 12, describing the rigid and affine FAST CPD implementations, respectively.

GPU Kernel 2: Calculating $\boldsymbol{P1}$ and $\boldsymbol{PX}$

Kernel size: M
Input buffers:

- Fixed points $\boldsymbol{X}$ (READ ONLY)
- Moving points $\boldsymbol{Y}$ (READ ONLY)
- Responsibility denominator $\boldsymbol{a}$ (READ-WRITE)
- $\boldsymbol{P1}$ (WRITE ONLY)
- $\boldsymbol{PX}$ (WRITE ONLY)

Input parameters:

- Variance σ^2
- Constant c
- Number of fixed points N

Kernel:

m = get_core_id(0)
M = get_kernel_size(0)
rowSum = 0.0
rowSumX1 = 0.0
rowSumX2 = 0.0
rowSumX3 = 0.0
for $n = 0, \ldots, N-1$
    norm = $\|\boldsymbol{x}_n - \boldsymbol{y}_m\|^2$
    responsibility = $\exp\left(-\text{norm}/(2 * \sigma^2)\right)$
    rowSum += responsibility
    rowSumX1 += responsibility $\cdot\, x_{n1}$
    rowSumX2 += responsibility $\cdot\, x_{n2}$
    rowSumX3 += responsibility $\cdot\, x_{n3}$
$(\boldsymbol{P1})_m$ = rowSum
$(\boldsymbol{PX})_{m1}$ = rowSumX1
$(\boldsymbol{PX})_{m2}$ = rowSumX2
$(\boldsymbol{PX})_{m3}$ = rowSumX3

Figure 7: Kernel for calculating $\boldsymbol{P1}$ and $\boldsymbol{PX}$ in parallel on the GPU.

5.6 Low-rank approximation of affinity matrix

In the elastic CPD algorithm the affinity matrix $\boldsymbol{G}$ is involved in two processes that both act as bottlenecks in the M-step. Firstly, $\boldsymbol{G}$ is a part of the system of equations to be solved in order to calculate $\boldsymbol{W}$. Secondly, $\boldsymbol{G}$ is involved in the matrix product with $\boldsymbol{W}$. The latter case will now be addressed first, assuming $\boldsymbol{W}$ is calculated, and then the problem of calculating $\boldsymbol{W}$ will be discussed in the following section.

5.6.1 A simple complexity analysis

A simple complexity analysis will underline the computational benefit provided by low-rank approximation, as discussed in section 3.6.3. In the elastic CPD algorithm the square affinity matrix $\boldsymbol{G}_{M\times M}$ is multiplied by $\boldsymbol{W}_{M\times D}$. By the naive approach this product has complexity $\mathcal{O}(M^2D)$. Since $\boldsymbol{G}$ is symmetric it can decomposed by the eigendecomposition in (36) as

$$\boldsymbol{G} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^T, \tag{91}$$

where $\boldsymbol{D}$ contain the eigenvalues of $\boldsymbol{G}$ on its diagonal ordered decreasingly. Now, performing a low-rank approximation the affinity matrix is approximated by

$$\boldsymbol{G} \approx \tilde{\boldsymbol{G}} = \boldsymbol{U}_{M\times K}\boldsymbol{\Lambda}_{K\times K}\boldsymbol{U}_{K\times M}^T, \tag{92}$$

where $\boldsymbol{\Lambda}$ contain the $K < M$ largest eigenvalues from $\boldsymbol{D}$, and $\boldsymbol{U}$ contain their corresponding eigenvectors. The rank K is called *numerical rank*. At first it might not be clear that performing more matrix multiplications will help, yet in total it will be faster because of the reduced complexity, especially when K is relatively small compared to M. Studying the matrix products in $\tilde{\boldsymbol{G}}\boldsymbol{W}$ the complexity sums up to $\mathcal{O}(2MKD + K^2D) \approx \mathcal{O}(MKD)$, by calculating the products from the right end first. Compared to the naive approach, the matrix multiplications with the low-rank approximation will be faster as long as the numerical rank is less than roughly 40% the size of M. As the results will reveal later, the approximation error is small even for $K \ll M$, so the computational runtime of this matrix multiplication will be significantly improved by the low-rank approximation.

The time consumed by the decomposition and the low-rank approximation itself is not accounted for yet, so the picture is a bit more complicated. However, notice that the affinity matrix is only calculated once, while the matrix product $\boldsymbol{G}\boldsymbol{W}$ is calculated for each iteration of the EM algorithm. Thus, the low-rank approximation needs only be performed once at the beginning of the CPD algorithm, and the components of the decomposition are stored during the registration.

5.6.2 Approximate basis and decomposition

In Eigen, the linear algebra library mentioned at the beginning of this chapter, there are several decomposition methods implemented, but decomposing the full $M \times M$ affinity matrix will still be a time-consuming step, despite being performed only once. Due to the following low-rank approximation it would be more efficient if one could calculate only the K eigenvalues and eigenvectors of interest, rather than calculating all M and then discard $M - K$ of them. Halko, Martinsson and Tropp present a framework dealing with this problem in [51], from which the low-rank approximation implementation in this thesis will be based on. They regard the low-rank approximation as a process with two stages:

A) Computing an approximate basis for the range of the matrix of interest, and

B) use the approximate basis to help calculate a standard matrix decomposition.

The framework they present uses random samples to identify a subspace of the matrix of interest that captures most of the action of the matrix, and by this solves the first step a low-rank approximation process. For the second stage, any decent QR-decomposition method will suffice, for instance those in Eigen. A catalogue of the decompositions and linear solvers implemented in Eigen can be found in the Eigen documentation.

The goal of stage A in a low-rank approximation of matrix $\boldsymbol{A}$ is to compute an orthogonal matrix $\boldsymbol{Q}$ such that

$$\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A}, \tag{93}$$

with as few columns, i.e. numerical rank, in $\boldsymbol{Q}$ as possible yet enough to keep the approximation accurate. An example with SVD will help to better see how the result of stage A aids the low-rank approximation in stage B. Let the basis matrix $\boldsymbol{Q}$ have numerical rank K much smaller than the number of rows in $\boldsymbol{A}$. First, the assisting matrix $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$ is formed, which provides a low-rank approximation $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{B}$. Then this assisting matrix is decomposed $\boldsymbol{B} = \tilde{\boldsymbol{U}}\boldsymbol{\Sigma}\boldsymbol{V}^T$. The dimension of B will be lower than of $\boldsymbol{A}$, thus the decomposition will be faster now than if $\boldsymbol{A}$ was decomposed directly. Setting $\boldsymbol{U} = \boldsymbol{Q}\tilde{\boldsymbol{U}}$, the input matrix is decomposed as approximated by $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$.

Halko et. al [51] thoroughly discusses and justifies the randomization approach of matrix approximation, and they summarizes the basic patterns of such schemes:

1. Preprocess the matrix, usually to calculate sampling probabilities.
2. Take random samples from the matrix, where the term sample refers generically to a linear function of the matrix.
3. Postprocess the samples to compute a final approximation, typically with classical techniques from numerical linear algebra. This step may require another look at the matrix.

A technique using randomization to estimate the approximate basis in stage A is described in Algorithm 4.5 in [51]. This algorithm uses *subsampled random Fourier transform* (SRFT) which involves taking random samples from the discrete Fourier transform. An orthogonal basis is then calculated for the resulting sampled matrix, concluding stage A. This algorithm is explained in more detail, together with the eigendecomposition algorithm 5.3 in [51] as stage B, in figure 8, providing a full low-rank approximation algorithm.

Low-rank approximation algorithm

Input parameter: Numerical rank K.
Input matrix: Matrix $\boldsymbol{A}_{M\times N}$ to be approximated.
Output matrices: $\boldsymbol{U}_{M\times K}, \boldsymbol{\Lambda}_{K\times K}$, such that $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$.
Stage A:

1. **SRFT**: Calculate the $N \times K$ SRFT matrix $\boldsymbol{\Omega} = \sqrt{\frac{N}{K}}\boldsymbol{D}\boldsymbol{F}\boldsymbol{R}$, where
   $\boldsymbol{D}$ is an $N \times N$ diagonal matrix whose elements are independent random variables uniformly distributed on the unit circle,
   $\boldsymbol{F}$ is an $N \times N$ unitary discrete Fourier transform (DFT), with elements $f_{pq} = N^{-1/2}\exp(-2\pi ipq/n)$, $p, q = 0, 1, 2, \ldots, N-1$,
   $\boldsymbol{R}$ is an $N \times K$ matrix that samples K coordinates from N uniformly at random, i.e. its K columns are drawn randomly without replacement from the columns of the $N \times N$ identity matrix.
2. **Sample matrix**: Form the $M \times K$ sample matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$.
3. **Orthogonal basis of sample matrix**: Use a factorization method to find an $M \times K$ orthogonal basis $\boldsymbol{Q}$ for the range of $\boldsymbol{Y}$, e.g. QR factorization: $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$.

Stage B:

1. **Assisting matrix**: Form the $K \times K$ matrix $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{\Lambda}\boldsymbol{Q}$.
2. **Eigendecomposition**: Use an eigendecomposition method to find $\boldsymbol{V}$ and $\boldsymbol{\Lambda}$ such that $\boldsymbol{B} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$.
3. **Orthogonal matrix**: Form the orthogonal matrix $\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{V}$.

Figure 8: Algorithm of low-rank approximation for a numerical desired rank K.

For the low-rank approximation the affinity matrix $\boldsymbol{G}$ in the elastic FCPD implementation, the algorithm described in figure 8 is implemented. There are some simplifications that can be made in the implementation of this algorithm in order to speed up the calculations. Notice is that neither the full discrete Fourier transform (DFT) matrix $\boldsymbol{F}$ nor the full matrix $\boldsymbol{D}$ will be need, as only the K columns corresponding to the random indices in $\boldsymbol{R}$ will be need. Figure shows the complete low-rank approximation algorithm used for $\boldsymbol{G}$ where this simplification is taken into account.

5.6.3 An alternative QR-decomposition algorithm

The decompositions involved in the low-rank approximation of $\boldsymbol{G}$ are accurately solved by methods implemented in Eigen, for instance the Householder QR decomposition. When running the given low-rank approximation algorithm, the QR-decomposition will prove to be the numerical bottleneck. The full QR-decomposition is not demanded in itself, as the purpose is simply to obtain an orthogonal basis for the sample matrix, as provided by the $\boldsymbol{Q}$ matrix. More specifically, K orthogonal column vectors (bases) spanning the column space of the sample matrix. A well-known method for this exact problem is the Gram-Schmidt process. Concluding this low-rank approximation discussion, an alternative approach to calculate $\boldsymbol{Q}$ using Gram-Schmidt will now briefly be explored.

In the report "Algorithms for QR-Decomposition", Walter Gander [52] reviews some QR-

Low-rank approximation of G implementation

This is a low-rank approximation implementation of the algorithm given in figure 8. The methods assumes that there exist a function rand(a, b) that output a random integer in the range $[a, b]$, and a function shuffle(v) that shuffles the order of the elements of a list v. Methods for QR-decomposition qr($\boldsymbol{A}$) and Eigendecomposition eigensolver($\boldsymbol{A}$) are needed as well.
Input parameter: Numerical rank K.
Input matrix: Affinity matrix $\boldsymbol{G}_{M\times M}$
Output matrices: $\boldsymbol{U}_{M\times K}, \boldsymbol{\Lambda}_{K\times K}$, such that $\boldsymbol{G} \approx \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$.

/ Find SRFT matrix $\boldsymbol{\Omega}$ and use it to sample $\boldsymbol{G}$, yielding sample matrix $\boldsymbol{Y}$ */*
for $i = 0, \ldots, M-1$ **do**
  $randomColIdx[i] = i$
  $r = \sqrt{\text{random}(0,1)}$
  $\theta = 2\pi \cdot$ random(0,1)
  $randomUniformSample[i] = r \cdot \cos(\theta)$
shuffle($randomColIdx$)

for $col = 0, \ldots, K-1$ **do**
  $colIdx = randomColIdx[col]$
  for $row = 0, \ldots, M-1$ **do**
    $\boldsymbol{\Omega}(row, col) = randomUniformSample[row] \cdot \cos\left(2\pi \cdot row \cdot colIdx/M\right)/\sqrt{M}$
$\boldsymbol{Y} = \boldsymbol{G} \cdot \boldsymbol{\Omega}$

/ Find orthogonal basis for sampleMatrix */*
$\boldsymbol{Q} = \text{qr}(\boldsymbol{Y})$

/ Eigendecomposition */*
$\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q}$
$\boldsymbol{V}, \boldsymbol{\Lambda} = \text{eigensolver}(\boldsymbol{A})$
$\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{V}$

Figure 9: Implementation of low-rank approximation of the affinity matrix $\boldsymbol{G}$ in elastic FCPD.

decomposition algorithms based on the Gram-Schmidt process, and also compares them with the a Householder algorithm. Gander states that simplest Gram-Schmidt QR-decomposition may result in a matrix $\boldsymbol{Q}$ not orthogonal at all, but by a process called *reorthogonalization* orthogonality can be assured. In the appendices of [52] the error related to computed $\boldsymbol{Q}$s not being orthogonal are compared. One of the algorithm with promising results is the *Modified Gram-Schmidt with Reorthogonalization* (Example 9.3). This algorithm is said to not work for (column-)rank deficient matrices, but this will not be a problem for the rank-reduced sample matrix in the case of the low-rank approximation in figure 8. Because the $\boldsymbol{R}$ is not needed for this purpose, some lines of the algorithm related to the calculation of $\boldsymbol{R}$ may be omitted. This QR-decomposition algorithm is described in figure 10 and will later be referred to as MGSR.

5.7 Calculate W

A central part of the maximization step in the elastic FCPD algorithm is the calculation of $\boldsymbol{W}$ which is the solution of equation (82). For simplicity, let both sides of the matrix equation be pre-multiplied by d($\boldsymbol{P}\boldsymbol{1}$) in order to avoid unnecessary inverses, and then rewrite it as

$$\boldsymbol{K}_{LHS}\boldsymbol{W} = \boldsymbol{K}_{RHS}, \tag{94}$$

where $\boldsymbol{K}_{LHS} = \text{d}(\boldsymbol{P}\boldsymbol{1})\boldsymbol{G} + \lambda\sigma^2\boldsymbol{I}$ and $\boldsymbol{K}_{RHS} = \boldsymbol{P}\boldsymbol{X} - \text{d}(\boldsymbol{P}\boldsymbol{1})\boldsymbol{Y}$. As discussed in section 3.7 the possible methods for solving the sets of equations for $\boldsymbol{W}$ include using a solver and solving the

QR-Decomposition by Modified Gram-Schmidt with Reorthogonalization

This is an implementation for QR-decomposition based on Gram-Schmidt, but where only the $\boldsymbol{Q}$ matrix is of interest.
Input matrix: Matrix $\boldsymbol{A}_{M\times K}$ to be decomposed.
Output matrices: Orthogonal basis $\boldsymbol{Q}$ for $\boldsymbol{A}$ s.t. $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_{K\times K}$

$\boldsymbol{Q} = \boldsymbol{A}$
for $k = 0, \ldots, K-1$ **do**
    $tt = 0$
    $t = \|\boldsymbol{Q}.\text{col}(k)\|$
    reorthogonalize = true
    while reorthogonalize **do**
        for $i = 0, \ldots, k-1$ **do**
            $s = \boldsymbol{Q}.\text{col}(i) \cdot \boldsymbol{Q}.\text{col}(k)$
            $\boldsymbol{Q}.\text{col}(k) -= s\boldsymbol{Q}.\text{col}(i)$
        $tt = \|\boldsymbol{Q}.\text{col}(k)\|$
        reorthogonalize = false
        if $(tt < t/10)$ **do**
            $t = tt$
            reorthogonalize = true
    $\boldsymbol{Q}.\text{col}(k) /= tt$

Figure 10: Implementation of the modified Gram-Schmidt algorithm with reorthonomalization QR-decomposition algorithm (MGSR), where only the orthogonal component $\boldsymbol{Q}$ is calculated.

system by computing the inverse of the right-hand side. Because the affinity matrix is given by distances between different points that may be close or even overlapping due to measurement inaccuracies or errors, it will probably be singular. Thus, K_{LHS} will also be singular, a fact that should be considered when proceeding with the problem of determining $\boldsymbol{W}$. In any case, using solvers or computing the inverse of the possibly large $M \times M$ affinity matrix will be slow when repeated in each EM iteration. Luckily, a low-rank approximation of $\boldsymbol{G}$ is useful for the calculation of $\boldsymbol{GW}$, as discussed above, and it will prove useful in the calculation of $\boldsymbol{W}$ as well.

As the left-hand side in this case consists of more than simply $\boldsymbol{G}$, and its dimensions is still $M \times M$, a solver used to find $\boldsymbol{W}$ is not helped much by the low-rank approximation. However, there is a mathematical identity that will take advantage over the low-ranked decomposition, namely the *Woodbury identity*, stating

$$(\boldsymbol{A} + \boldsymbol{UCV})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}\left(\boldsymbol{C}^{-1} + \boldsymbol{VA}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{VA}^{-1}, \tag{95}$$

for any matrices $\boldsymbol{A}$, $\boldsymbol{U}$, $\boldsymbol{C}$ and $\boldsymbol{V}$ of conformable sizes, where $\boldsymbol{A}$ and $\boldsymbol{C}$ are invertible. Applied to $\boldsymbol{K}_{LHS}$ the Woodbury identity reads

$$\begin{aligned} \boldsymbol{K}_{LHS}^{-1} = \left[d(\boldsymbol{P1})\boldsymbol{G} + \frac{1}{k}\boldsymbol{I}\right]^{-1} &\approx \left[d(\boldsymbol{P1})\boldsymbol{Q\Lambda Q}^T + \frac{1}{k}\boldsymbol{I}\right]^{-1} \\ &= k\boldsymbol{I} - k^2 d(\boldsymbol{P1})\boldsymbol{Q}\left[\boldsymbol{\Lambda}^{-1} + k\boldsymbol{Q}^T d(\boldsymbol{P1})\boldsymbol{Q}\right]^{-1}\boldsymbol{Q}^T, \end{aligned} \tag{96}$$

with $k = 1/(\lambda\sigma^2)$. As long as the numerical rank K is less than or equal to the true rank of $\boldsymbol{G}$, the eigenvalue-matrix $\boldsymbol{\Lambda}$ only contains positive eigenvalues and will be invertible; its inverse is given simply by replacing the eigenvalues by their reciprocals. A matrix inversion is still needed, as well as some matrix multiplications, yet the complexity of this matrix inverse is $\mathcal{O}(K^3)$ by direct computation, which is considerably better than the initial $\mathcal{O}(M^3)$. For $K = \sqrt[3]{M}$ the complexity is reduced to linear for this step. Pre-multiplying equation (94) by the inverse of the left-hand side, $\boldsymbol{W}$ is acquired by

$$\boldsymbol{W} = \boldsymbol{K}_{LHS}^{-1}\boldsymbol{K}_{RHS}. \tag{97}$$

5.8 Fast Gauss Transform

In order to speed up the responsibility products $\boldsymbol{P}^T\mathbf{1}$, $\boldsymbol{P}\mathbf{1}$ and $\boldsymbol{P}\boldsymbol{X}$, Myronenko and Song proposed to use the fast Gauss transform (FGT). The FGT was first introduced by Greengard and Strain in 1991 [53]. The algorithm reduces the complexity for computing a sum of exponentials from $\mathcal{O}(MN)$ to $\mathcal{O}(M+N)$. In CPD, FGT is used with the Gaussian kernel $\boldsymbol{K}$ matrix, with elements

$$k_{mn} = \exp\left(-\frac{1}{2\sigma^2}\left\|\boldsymbol{x}_n - \boldsymbol{y}_m\right\|^2\right). \tag{98}$$

Because the calculation of the kernel matrix will be a computational bottleneck FCPD avoids computing it and does not use FGT. The interested reader is referred to the CPD paper [2] for details on how FGT was used to calculate the responsibility products in the original CPD implementation.

5.9 Algorithms for comparison

The open-source C++ CPD implementation by Gadomski [1] will be used to evaluate the performance of the FAST CPD algorithm. Gadomski's implementation is based on the Matlab implementation developed by the authors of the CPD algorithm [2], using the provided optimizations, namely fast Gauss transform and low-rank approximation. In addition some CPU parallelization with OpenMP is included.

A few modifications was done by the writer to Gadomski's implementation in order to make the comparison as fair as possible. Firstly, the default parameters have been updated to match with the one used for the different registration cases studied. Secondly, the same low-rank approximation algorithm was provided to the CPD code as for the FCPD. The newest version of Gadomski's CPD did not include low-rank approximation, and the older version with that included was based on a linear algebra library named Armadillo, rather than Eigen. Thirdly, the Jocobi SVD algorithm from Eigen used by Gadomski was replaced by the BDC version. According to Eigen, the latter should be faster for large data sets [54]. Lastly, the convergence criterion in Gadomski's CPD was updated to match the one used in FCPD.

5.10 FAST CPD algorithms

The FCPD rigid, affine and elastic implementations are summarized in figure 11, 12 and 13, respectively.

Rigid FCPD implementation

Initialization: $\boldsymbol{R} = \boldsymbol{I}$, $\boldsymbol{t} = 0$, $s = 0$, $0 \leq w \leq 1$, $\sigma^2 = \frac{1}{DNM} \sum_{m=1}^{M} \|\boldsymbol{x}_n - \boldsymbol{y}_m\|^2$.

EM: Repeat until convergence

1. **Expectation**: Calculate responsibility products $\boldsymbol{P}\mathbf{1}$, $\boldsymbol{P}^T\mathbf{1}$, and $\boldsymbol{P}\boldsymbol{X}$.

   Kernel 1: Calc. $\boldsymbol{P}^T\mathbf{1}$.

   Kernel 2: Calc. $\boldsymbol{P}\mathbf{1}$ and $\boldsymbol{P}\boldsymbol{X}$.

   $N_{\boldsymbol{P}} = \sum_{n=1}^{N} (\boldsymbol{P}^T\mathbf{1})_n = \sum_{m=1}^{M} (\boldsymbol{P}\mathbf{1})_m$.

2. **Maximization**: Solve for $\boldsymbol{R}$, s, $\boldsymbol{t}$ and σ^2.

   $\boldsymbol{\mu}_{\boldsymbol{x}} = \frac{1}{N_{\boldsymbol{P}}} \boldsymbol{X}^T \boldsymbol{P}^T \mathbf{1}$, $\boldsymbol{\mu}_{\boldsymbol{y}} = \frac{1}{N_{\boldsymbol{P}}} \boldsymbol{Y}^T \boldsymbol{P}\mathbf{1}$,

   $\boldsymbol{A} = (\boldsymbol{P}\boldsymbol{X})^T \boldsymbol{Y} - N_{\boldsymbol{P}} \boldsymbol{\mu}_{\boldsymbol{x}} \boldsymbol{\mu}_{\boldsymbol{y}}^T$,

   Compute SVD of $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{S}\boldsymbol{V}^T$,

   $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{C}\boldsymbol{V}^T$, where $\boldsymbol{C} = \mathrm{d}(1, \ldots, 1, \det(\boldsymbol{U}\boldsymbol{V}^T))$,

   $s = \operatorname{tr}\left(\boldsymbol{A}^T \boldsymbol{R}\right) / \left(\sum_{m=1}^{M} (\boldsymbol{P}\mathbf{1})_m \|\boldsymbol{y}_m\|^2 - N_{\boldsymbol{P}} \|\boldsymbol{\mu}_{\boldsymbol{y}}\|^2\right)$,

   $\boldsymbol{t} = \boldsymbol{\mu}_{\boldsymbol{x}} - s\boldsymbol{R}\boldsymbol{\mu}_{\boldsymbol{y}}$,

   $\sigma^2 = \frac{1}{N_{\boldsymbol{P}} D} \left[\left(\sum_{n=1}^{N} (\boldsymbol{P}^T\mathbf{1})_n \|\boldsymbol{x}_n\|^2\right) - N_{\boldsymbol{P}} \|\boldsymbol{\mu}_{\boldsymbol{y}}\|^2 - s \operatorname{tr}\left(\boldsymbol{A}^T \boldsymbol{R}\right)\right]$.

Result: The aligned point set is $\mathcal{T}(\boldsymbol{Y}) = s\boldsymbol{Y}\boldsymbol{R}^T + \mathbf{1}\boldsymbol{t}^T$.

Figure 11: Implementation of the rigid FCPD algorithm.

Affine FCPD implementation

Initialization: $\boldsymbol{B} = \boldsymbol{I}$, $\boldsymbol{t} = 0$, $0 \leq w \leq 1$, $\sigma^2 = \frac{1}{DNM} \sum_{m=1}^{M} \|\boldsymbol{x}_n - \boldsymbol{y}_m\|^2$.

EM: Repeat until convergence

1. **Expectation**: Calculate responsibility products $\boldsymbol{P}\mathbf{1}$, $\boldsymbol{P}^T\mathbf{1}$, and $\boldsymbol{P}\boldsymbol{X}$.

   GPU Kernel 1: Calc. $\boldsymbol{P}^T\mathbf{1}$.

   GPU Kernel 2: Calc. $\boldsymbol{P}\mathbf{1}$ and $\boldsymbol{P}\boldsymbol{X}$.

   $N_{\boldsymbol{P}} = \sum_{n=1}^{N} (\boldsymbol{P}^T\mathbf{1})_n = \sum_{m=1}^{M} (\boldsymbol{P}\mathbf{1})_m$.

2. **Maximization**: Solve for $\boldsymbol{B}$, s, $\boldsymbol{t}$ and σ^2.

   $\boldsymbol{\mu}_{\boldsymbol{x}} = \frac{1}{N_{\boldsymbol{P}}} \boldsymbol{X}^T \boldsymbol{P}^T \mathbf{1}$, $\boldsymbol{\mu}_{\boldsymbol{y}} = \frac{1}{N_{\boldsymbol{P}}} \boldsymbol{Y}^T \boldsymbol{P}\mathbf{1}$,

   $\boldsymbol{A} = (\boldsymbol{P}\boldsymbol{X})^T \boldsymbol{Y} - N_{\boldsymbol{P}} \boldsymbol{\mu}_{\boldsymbol{x}} \boldsymbol{\mu}_{\boldsymbol{y}}^T$,

   $\boldsymbol{B} = \boldsymbol{A} \left[\boldsymbol{Y}^T \mathrm{d}(\boldsymbol{P}\mathbf{1}) \boldsymbol{Y} - N_{\boldsymbol{P}} \boldsymbol{\mu}_{\boldsymbol{x}} \boldsymbol{\mu}_{\boldsymbol{y}}^T\right]^{-1}$,

   $\boldsymbol{t} = \boldsymbol{\mu}_{\boldsymbol{x}} - \boldsymbol{B}\boldsymbol{\mu}_{\boldsymbol{y}}$,

   $\sigma^2 = \frac{1}{N_{\boldsymbol{P}} D} \left[\left(\sum_{n=1}^{N} (\boldsymbol{P}^T\mathbf{1})_n \|\boldsymbol{x}_n\|^2\right) - N_{\boldsymbol{P}} \|\boldsymbol{\mu}_{\boldsymbol{y}}\|^2 - \operatorname{tr}\left(\boldsymbol{A}\boldsymbol{B}^T\right)\right]$.

Result: The aligned point set is $\mathcal{T}(\boldsymbol{Y}) = \boldsymbol{Y}\boldsymbol{B}^T + \mathbf{1}\boldsymbol{t}^T$.

Figure 12: Implementation of the affine FCPD algorithm.

Elastic FCPD implementation

Initialization: $\boldsymbol{W} = 0$, $\sigma^2 = \frac{1}{DNM}\sum_{m=1}^{M} \|\boldsymbol{x}_n - \boldsymbol{y}_m\|^2$.
Set parameters: $0 \leq w \leq 1$, $\beta > 0$, $\lambda > 0$.
Calculate $\boldsymbol{G}$: $g_{ij} = \exp\left(-\frac{\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2}{2\beta^2}\right)$
Low rank approximation: $\boldsymbol{G} \approx \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$
EM: Repeat until convergence

1. **Expectation**: Calculate responsibility products $\boldsymbol{P}\mathbf{1}$, $\boldsymbol{P}^T\mathbf{1}$, and $\boldsymbol{P}\boldsymbol{X}$.
   GPU Kernel 1: Calc. $\boldsymbol{P}^T\mathbf{1}$.
   GPU Kernel 2: Calc. $\boldsymbol{P}\mathbf{1}$ and $\boldsymbol{P}\boldsymbol{X}$.
   $N_{\boldsymbol{P}} = \sum_{n=1}^{N}(\boldsymbol{P}^T\mathbf{1})_n = \sum_{m=1}^{M}(\boldsymbol{P}\mathbf{1})_m$.

2. **Maximization**: Solve for $\boldsymbol{W}$ and σ^2.
   Calc. $\boldsymbol{K}_{P1Q} = \mathrm{d}(\boldsymbol{P}\mathbf{1})\boldsymbol{Q}$.
   GPU Kernel 3: Calc. $\boldsymbol{K}_{RHS} = \boldsymbol{P}\boldsymbol{X} - \mathrm{d}(P1)\boldsymbol{Y}$.
   $k = 1/(\lambda\sigma^2)$.
   Approximate $\boldsymbol{K}_{LHS}^{-1} = \left[\mathrm{d}(\boldsymbol{P}\mathbf{1})\boldsymbol{G} + k^{-1}\boldsymbol{I}\right]^{-1} \approx \left[\boldsymbol{K}_{P1Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T + k^{-1}\boldsymbol{I}\right]^{-1}$.
   $\boldsymbol{W} = \boldsymbol{K}_{LHS}^{-1}\boldsymbol{K}_{RHS}$.
   Approximate $\boldsymbol{G}\boldsymbol{W} \approx \boldsymbol{Q}(\boldsymbol{\Lambda}(\boldsymbol{Q}^T\boldsymbol{W}))$.
   Update $\boldsymbol{Y}$: $\tilde{\boldsymbol{Y}} = \boldsymbol{Y} + \boldsymbol{G}\boldsymbol{W}$.
   $\sigma^2 = \frac{1}{N_{\boldsymbol{P}}D}\left[\left(\sum_{n=1}^{N}(\boldsymbol{P}^T\mathbf{1})_n \|\boldsymbol{x}_n\|^2\right) + \left(\sum_{m=1}^{M}(\boldsymbol{P}\mathbf{1})_m \|\tilde{\boldsymbol{y}}_m\|^2 - 2(\boldsymbol{P}\boldsymbol{X})_m \cdot \tilde{\boldsymbol{y}}_m^T\right)\right]$.

Result: The aligned point set is $\mathcal{T}(\boldsymbol{Y}, \boldsymbol{W}) = \tilde{\boldsymbol{Y}} = \boldsymbol{Y} + \boldsymbol{G}\boldsymbol{W}$.

Figure 13: Implementation of the elastic FCPD algorithm.

6 Results

All the results presented in this chapter have been generated on a computer running Linux (Ubuntu 18.04 LTS) with 12-core 3.30GHz CPU (Intel i7-5820K) and 16 GB RAM. The computer is equipped with an AMD Radeon R9 FURY graphics processing unit (GPU), which has 3584 cores, 4 GB memory and 1 GHz clock speed.

Results are only presented for rigid and elastic registrations. These two are the most common, and thus it is easier to compare the results with the works published by others. In addition, the optimization and parallelization steps for affine FCPD are similar to those in the rigid FCPD, so the speedup will also be similar.

6.1 Data preprocessing and default parameters

All the synthetic data sets for rigid and elastic registration have been normalized (zero mean, unit variance) before registration. This is not done for the brain-shift data, where the moving point set only covers a part of the fixed data set. For the synthetic data, only 3D data is considered and the fixed and moving point sets are of the same size, $M = N$. All registrations are said to converge when the variance changes less then a given tolerance between two consecutive iterations, or a roof of maximum 100 iterations is reached. The tolerance is set to 10^{-7} for rigid registration and 10^{-6} for elastic registration. FCPD will always refer to the CPU and GPU parallelized CPD implementation in FAST.

6.2 Rigid registration

In the rigid registration results, $\omega = 0$ when no noise is added, and $\omega = 0.5$ otherwise. The rigid FCPD implementation does not change any of the calculations in the original CPD algorithm but parallelize and optimize some of the calculations. Thus, the results regarding registration error and parameter sensitivity will not be altered by the FCPD implementation. The point of interest here will be the rigid runtime results. The rigid CPD profiling shown previously in figure 5 can be compared to the corresponding rigid FCPD profiling provided in figure 15, where GPU Kernel 1 and GPU Kernel 2 constitute the E-step as well as the $\boldsymbol{P}$ reductions in the former profiling.

The achieved runtimes for rigid FCPD for different (downsampled) point set sizes of the Stanford bunny data set are showed in figure 14 and compared to FAST ICP and rigid CPD. The moving data set equals the fixed but a rotation of 50 degrees about the Y-axis. Table 2 states the achieved runtimes from the figure. The registration times are averaged over 25 runs to account for variations in GPU computations, but the standard deviations are found to be negligible. The estimated slopes in the corresponding log-log-plot are 0.94, 1.98 and 2.25 for FCPD, CPD and ICP, respectively, implying a linear speedup from $\mathcal{O}(M^2)$ to $\mathcal{O}(M)$ with FCPD. Using the difference between the true and the registered rotation matrix as an error metric, the resulting error is in the order of 10^{-10} for all three methods. The FCPD registration time of 47 ms for $M = N = 1600$ is enough to provide a frame rate of about 20 FPS.

Table 2: Rigid registration times in seconds for CPD Gadomski and FCPD for various point set sizes.

Point set sizes (M, N)	**CPD Gadomski**	**FCPD**
800×3	0.090	0.024
1600×3	0.313	0.047
3200×3	1.25	0.084
6400×3	4.72	0.158
12800×3	19.7	0.334

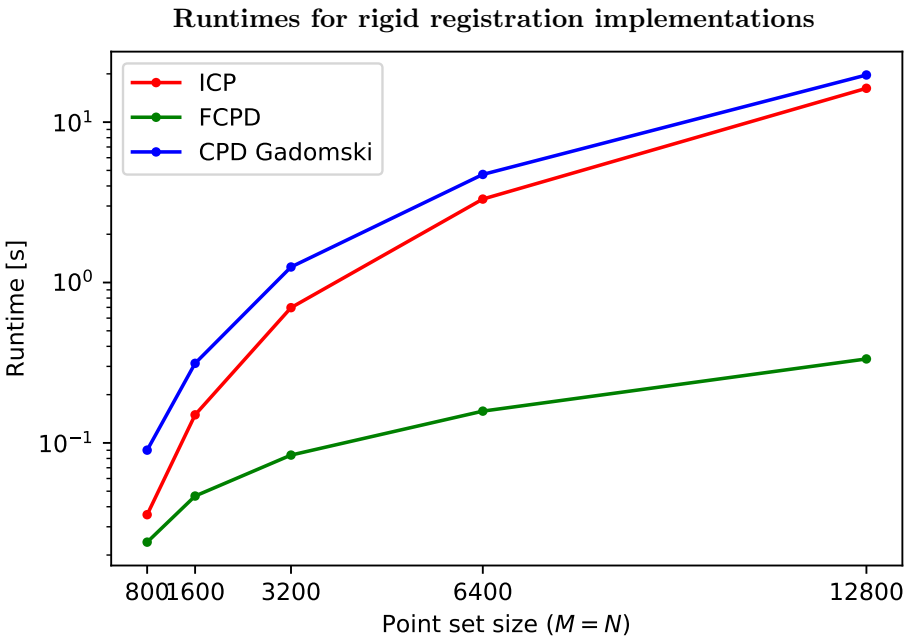


Figure 14: Runtimes for rigid registration for different point set sizes.

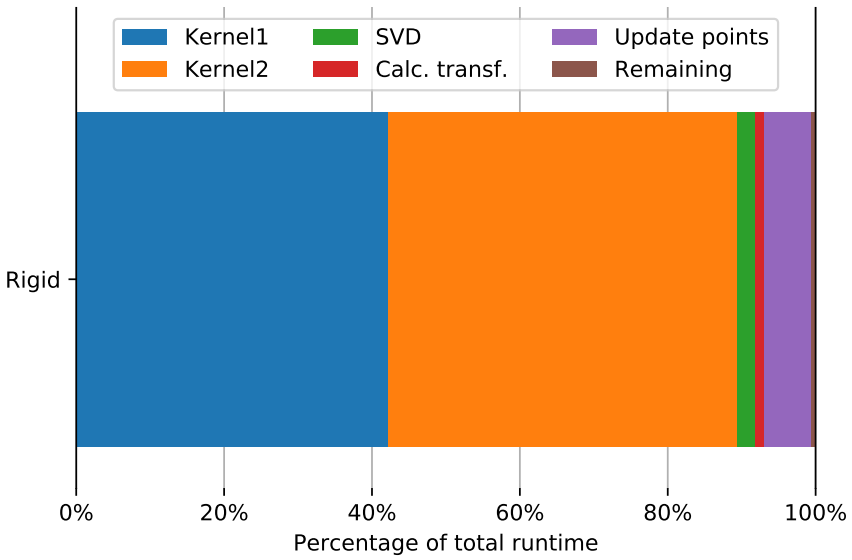


Figure 15: Profiling of computation time for rigid FCPD.

6.3 Low-rank approximation

To account for the randomness introduced in the low-rank approximation, the results for each set of parameters are averaged over 50 runs. The error of the low-rank approximation is here set to the squared norm of the difference between the true affinity matrix $\boldsymbol{G}$ and its approximation, i.e. $\left\|\boldsymbol{G} - \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T\right\|^2$. This an error measure similar in form as the one used for the rotation matrix in rigid registration, but note that the size of the affinity matrix vary as different point set sizes are tested.

In section 5.6, two different QR-decomposition was introduced as a part of the low-rank approximation, one a Householder QR-decomposition included in Eigen, and the other a modified Gram-Schmidt with reorthogonalization (MGSR). At first, the Householder variant is used to calculate the approximation error for different β values, shown in figure 16 for different sizes of the Bunny data set with the same deformation applied as will be introduced in the next section. Figure 17 and 18 show the errors and runtimes, respectively, for the same data sets, for both QR-decompositions mentioned, and for different numerical ranks, K. The MGSR implementation is clearly faster but suffers from a slightly higher and more fluctuating error. Notice that the effect of increasing the rank is minimal on the approximation after the rank exceeds a certain value. Looking at the standard deviation of the error, see figure 19, it is higher for MGSR than for the Eigen variant. Different QR-decomposition implementations in Eigen were tested among which the Householder implementation is used for these results.

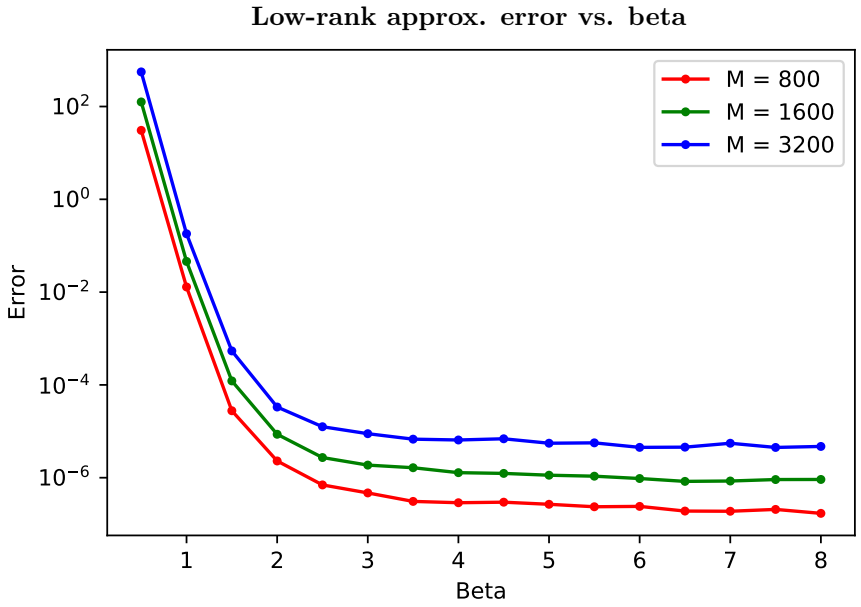


Figure 16: Approximation error in low-rank approximation of $\boldsymbol{G}$ for different values of β and point set sizes M.

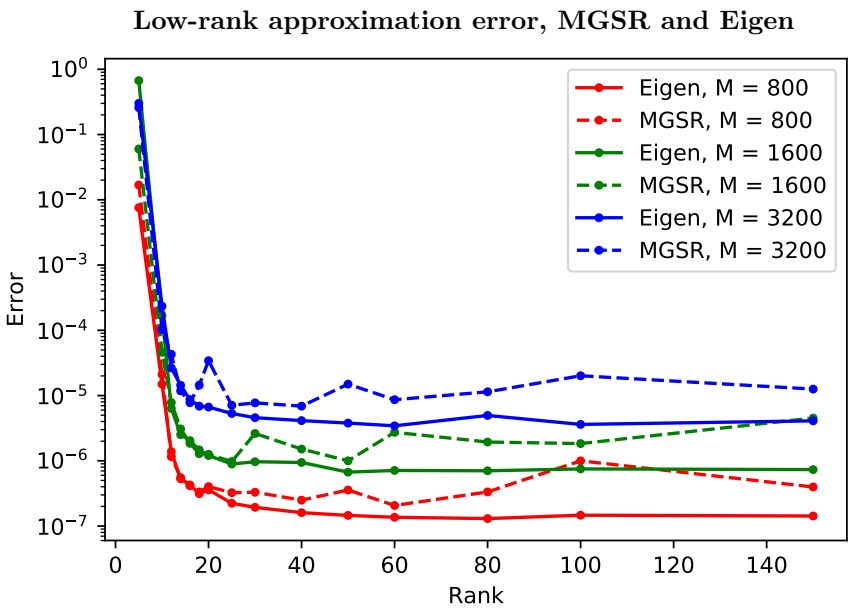


Figure 17: Approximation error in low-rank approximation of $\boldsymbol{G}$ for different ranks and point set sizes.

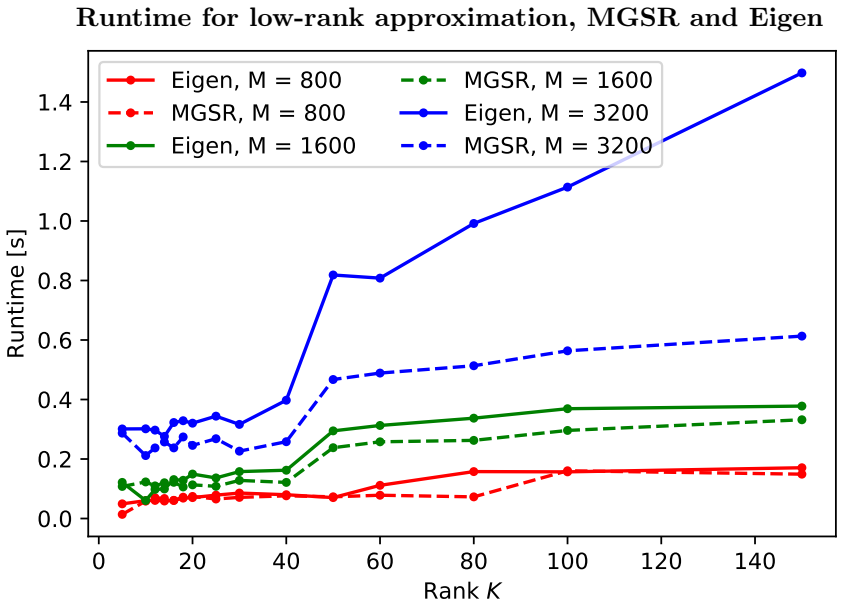


Figure 18: Computation time of low-rank approximation of $\boldsymbol{G}$ for different ranks and point set sizes.

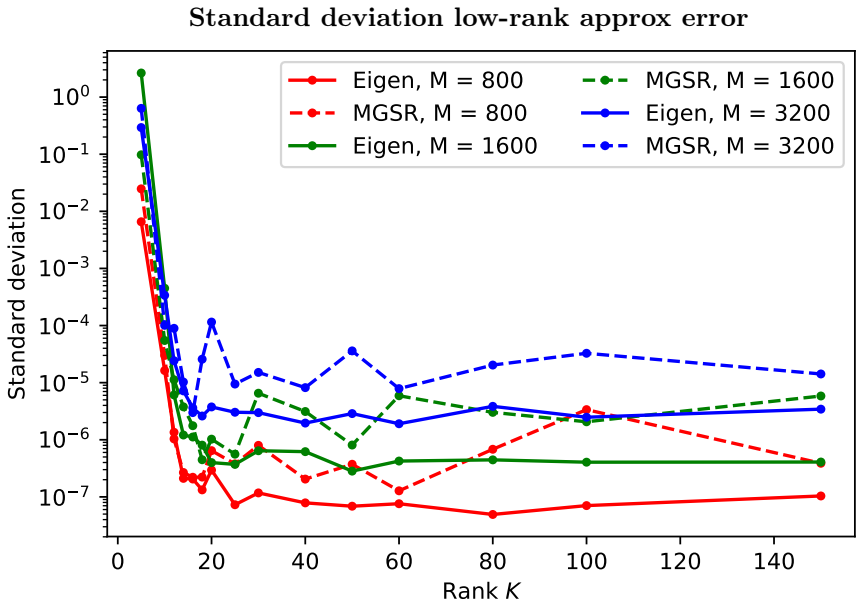
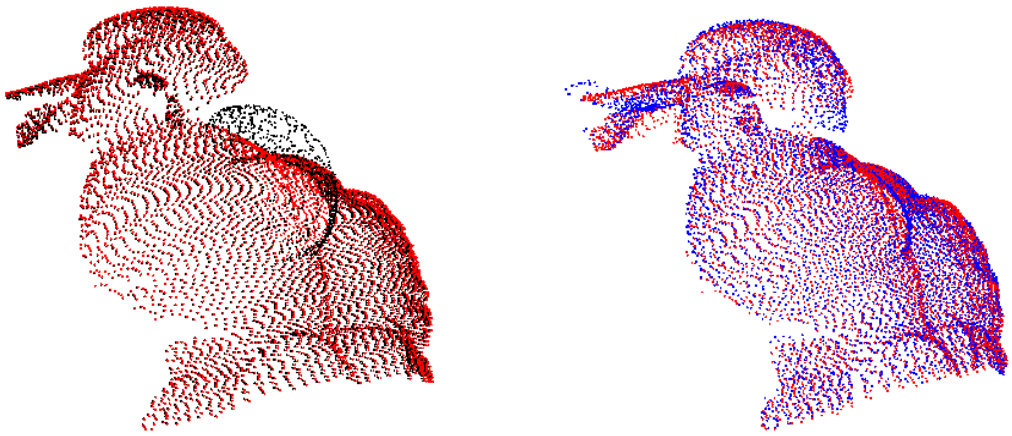


Figure 19: Standard deviation for the low-rank approximation errors for different ranks for both QR-decomposition considered.

6.4 Elastic registration

To account for the randomness introduced by the low-rank approximation, all elastic results are averaged over 25 independent runs. A spherical deformation, pushing all points within a distance of 0.5 from the origin out to that radius, is introduced to the point sets for the elastic registration. The deformation described is visualized on the Bunny data set in figure 20, together with the result from elastic FCPD registration. Unless otherwise stated, this is the data set, with various downsamplings, all the elastic registration results are tested on. Regarding the numerical rank K for the low-rank approximation, the default from Gadomski's CPD, $K = \sqrt{M}$, is used for all cases.



(a) Before registration

(b) After registration

Figure 20: The figure to the left shows the deformation applied the Bunny data set in black, compared to the fixed non-deformed point set in red. In the figure to the right the result after elastic CPD is shown in blue.

6.4.1 Parameter sensitivity and registration error

In the study of elastic FCPD registration parameter sensitivity, the mean squared distances between the fixed and registered moving point set are used as error metrics.

The effect of different combinations of β and λ in the range from 1.0 to 6.0, is visualized with a heatmap in figure 21. In figure 22 the error is decomposed to study the error contribution from the points exposed to the deformation and the non-deformed points separately. Corresponding heatmaps of the relative standard deviation of the error and the computational runtime are included in figures 23a and 23b, respectively.

The registration results for two different β values are showed in figure 24, where a rotation of 50 degrees and Gaussian noise with standard deviation of 0.1 is added to the data sets, in addition to the deformation introduced above. The registration error was lower for $\beta = 8.0$ than for $\beta = 2.0$, consistent with the heatmaps above, yet the result appear noisier.

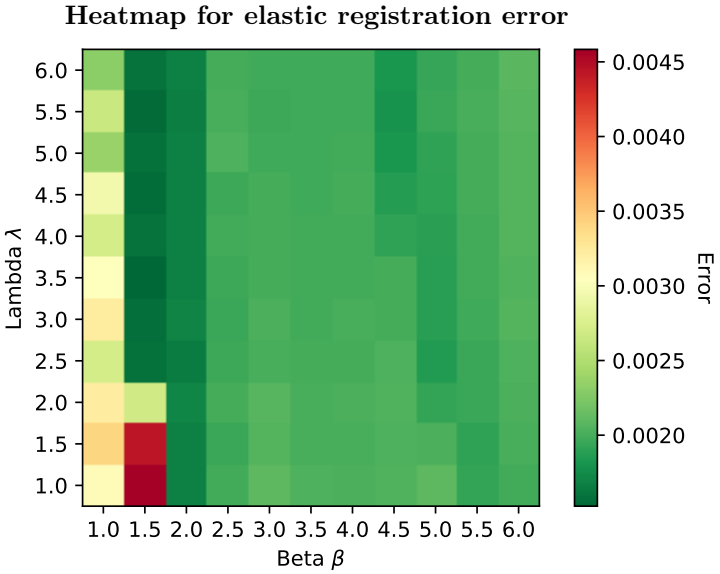
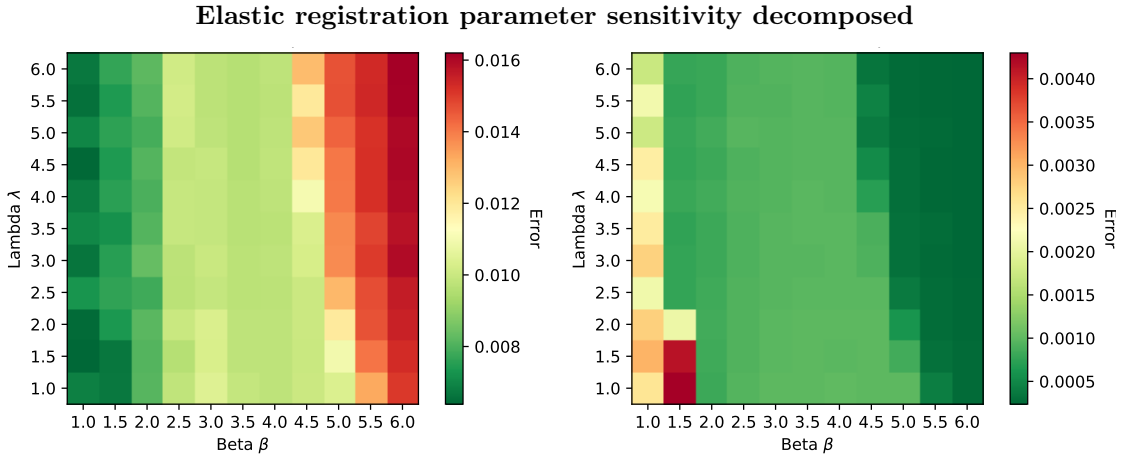


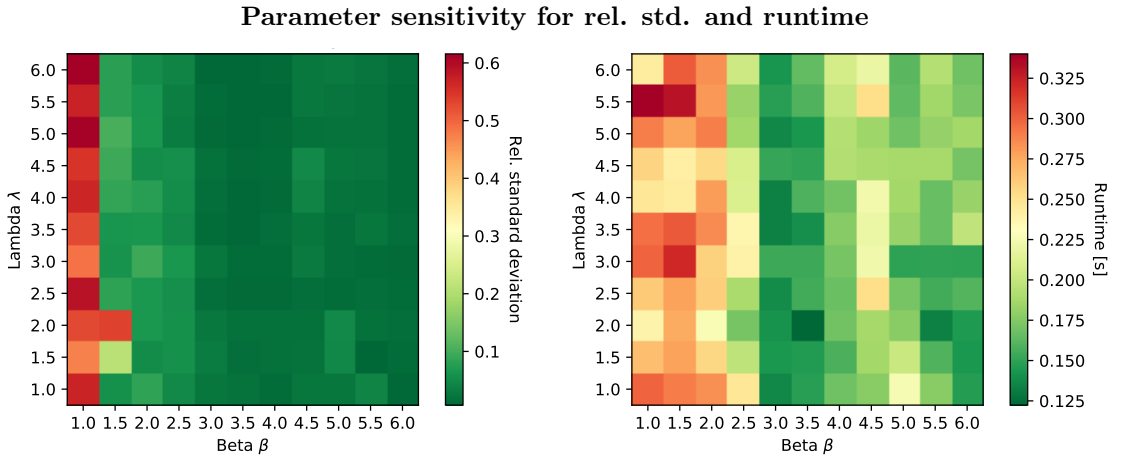
Figure 21: Total elastic registration error for different values of the parameters β and λ.



(a) Deformed points

(b) Non-deformed points

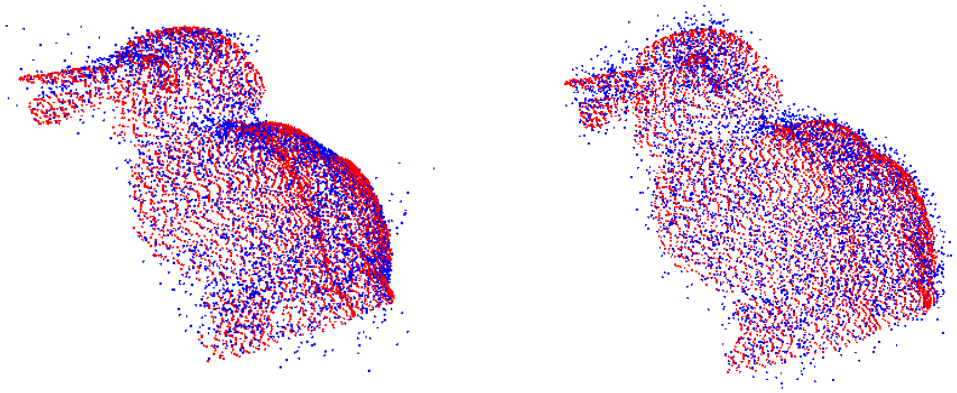
Figure 22: Elastic registration error for different values of the parameters β and λ, decomposed to only deformed points (22a) and non-deformed (the remaining) points (22b.



(a) Rel. std. of error

(b) Runtimes

Figure 23: Relative standard deviation of the elastic registration error and runtimes for different values of the parameters β and λ.



(a) $\beta = 2.0$

(b) $\beta = 8.0$

Figure 24: Elastic registration result for two different β-values, and $\lambda = 2.0$. The data sets were transformed, rotated 50 degrees and Gaussian noise with $\mu = 0$, $\sigma = 0.1$ was added before registration.

6.4.2 Runtime and profiling

In figure 25 the elastic FCPD implementation is profiled for three different point set sizes of the deformed Bunny with the Eigen QR-decomposition in the low-rank approximation, and one implemented with MGSR. For large point sets it is evident that the low-rank approximation is the primary bottleneck in the first cases, but MGSR resolves this. Then, solving for $\boldsymbol{W}$ become the most time-consuming step of this method.

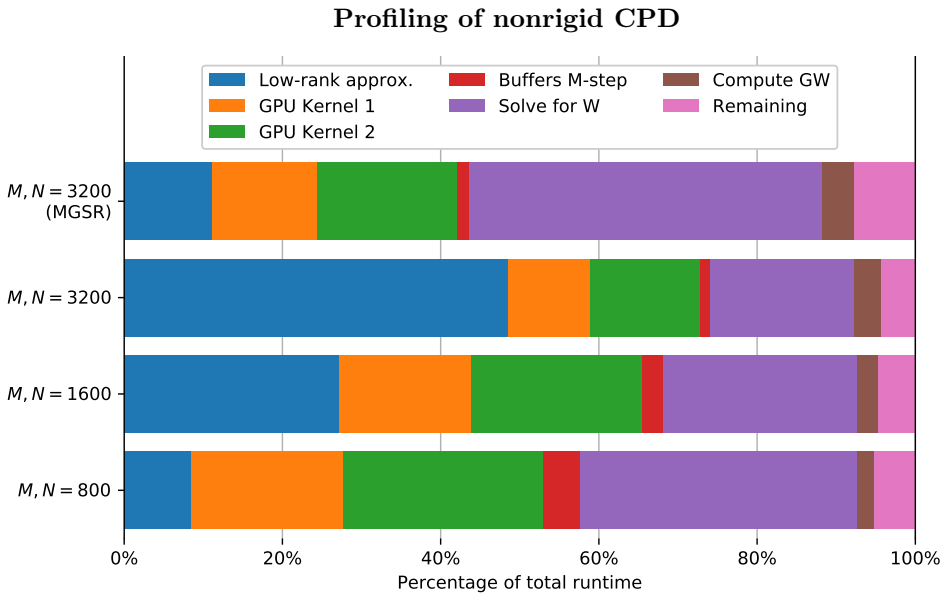


Figure 25: Profiling of computation time for elastic FCPD registration. The registration is performed on the data set from figure 20 for different point set sizes. For the row at the top, MGSR was used for QR-decomposition in the low-rank approximation, while a method in Eigen used for the rest.

For the following elastic registration results, the FCPD implementation using MGSR will be used. A comparison of runtimes of elastic FCPD versus the CPD implementation without GPU parallelization for different point set sizes is shown in figure 26. The point set sizes $M, N = \{800, 1600, 3200, 6400, 12800\}$ have been used here. The FCPD registration did converge at a lower number of iterations than the other (roughly 40 vs 90 for $M = 1600$), but studying the average runtime per EM iteration, see figure 27, the average EM iteration complexity is improved from $\mathcal{O}(M^{2.31})$ to $\mathcal{O}(M^{1.34})$. The similar results for the overall complexity is an improvement from $\mathcal{O}(M^{2.30})$ to $\mathcal{O}(M^{1.56})$ with the FCPD implementation compared to Gadomski's CPD with the same low-rank approximation method.

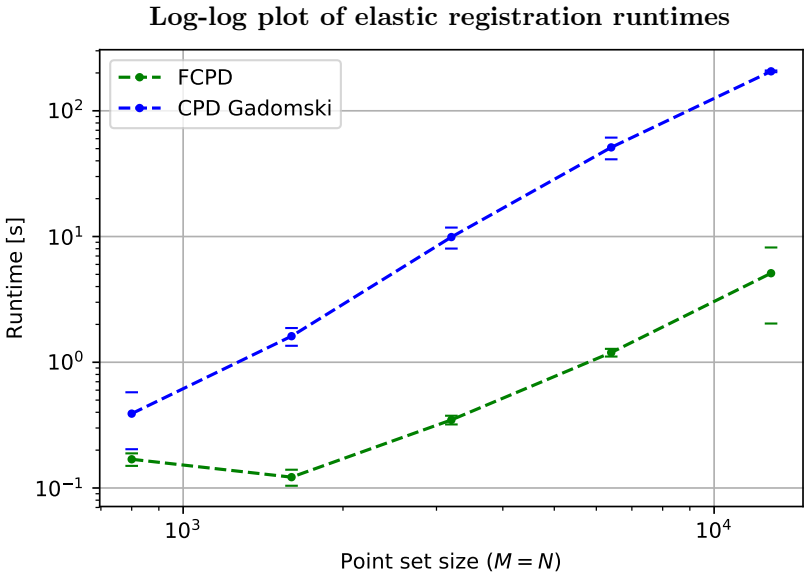


Figure 26: The runtime of elastic FCPD compared with and the pure CPU implementation CPD by Gadomski [1] for various point set sizes.

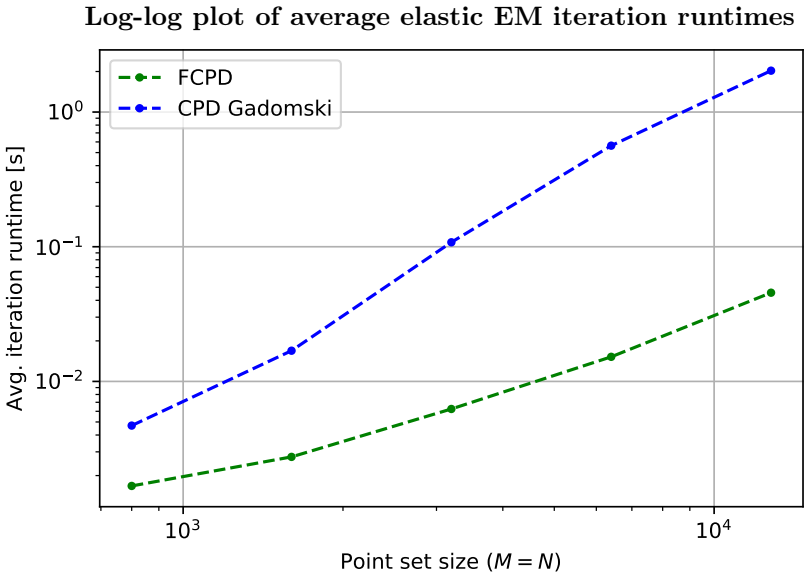
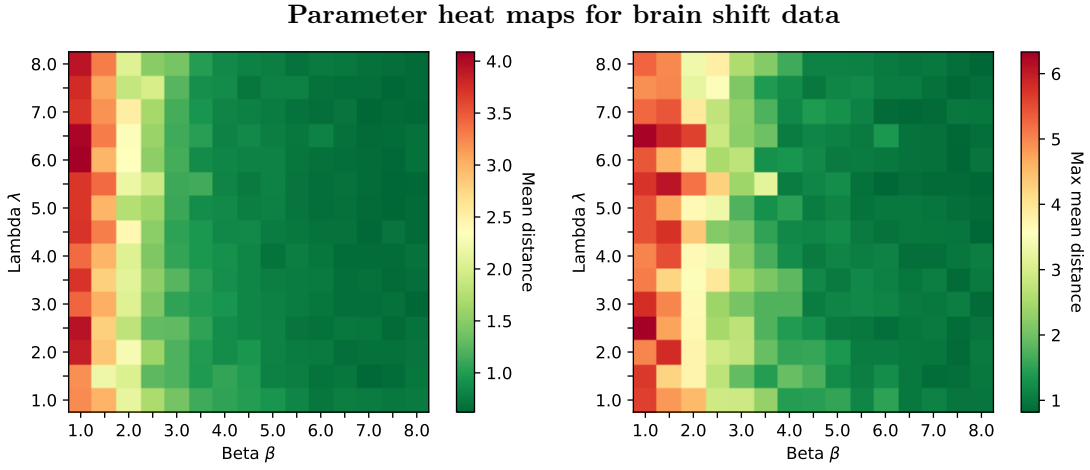


Figure 27: Average EM iteration runtime for elastic registration for different point set sizes.

6.5 Blood vessel registration for brain-shift correction

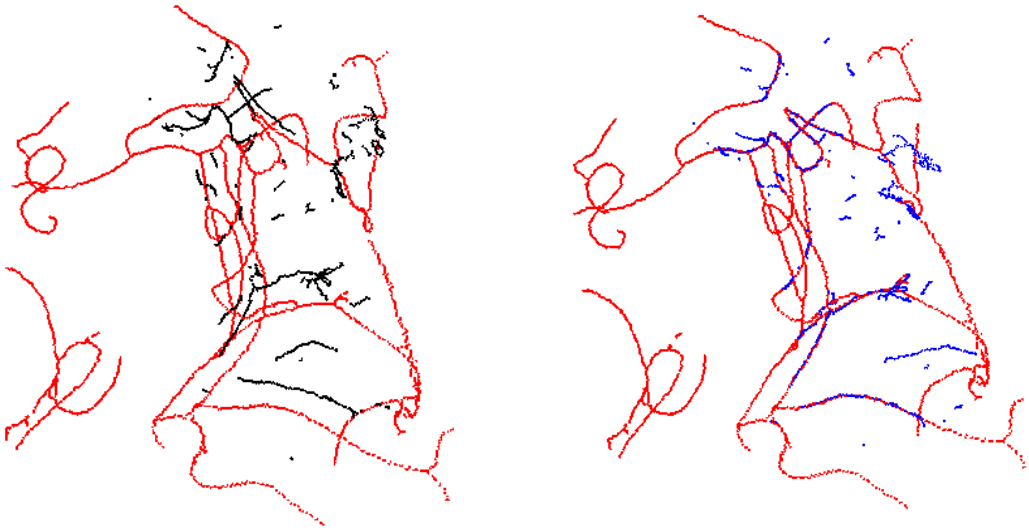
In the case of blood vessel registration for brain-shift correction 3-5 corresponding landmarks have been chosen in the centerlines from MRA and US. The elastic registration error is the mean distance between these landmarks after registration, and the maximum distance among the landmarks was also considered. The landmarks have not been assessed by any clinicians, but the purpose is here to get an impression of whether the elastic CPD registration works for blood vessel centerline data, and how sensitive it is for parameter settings, rather than to provide accurate error measurements.

Heatmaps showing the mean and maximum error for the elastic registration parameters are included in figure 28. Figure 29 shows a brain-shift correction result using elastic FCPD with $\beta = \lambda = 8.0$.



(a) Mean distance **(b)** Max distance

Figure 28: Heatmaps showing the mean (left) and max (right) of the mean landmark distances calculated over 20 runs for each pair of the parameters β and λ.



(a) Before registration **(b)** After registration

Figure 29: Elastic FCPD registration result for brain-shift correction, $\beta = \lambda = 8.0$.

7 Discussion

The FAST CPD (FCPD) implementation clearly succeeds in improving the runtime for rigid, affine and elastic registration with the Coherent Point Drift algorithm compared to the original fast implementation. The original CPD paper [2] introduced and demonstrated the fast Gauss transform (FGT) and the low-rank approximation as efficient means to reduce the runtime of the naive CPD implementation, providing a fast implementation of their own algorithm. The FGT reduces the complexity of the matrix-vector product needed to calculate the responsibility products $\boldsymbol{P}^T\mathbf{1}$, $\boldsymbol{P}\mathbf{1}$ and $\boldsymbol{PX}$, but depends on the calculation of the kernel matrix $\boldsymbol{K}_{M\times N}$ to do so. The FCPD implementation avoids the calculation of both the responsibility matrix $\boldsymbol{P}$ and the kernel matrix $\boldsymbol{K}$, and instead calculates the responsibility products on the GPU directly from the point sets. This is the main contribution to the speedup in FCPD, especially for rigid and affine registration. A strength of the CPD and FCPD algorithm is the possibility for rigid, affine and elastic registration within the same algorithmic framework, making it easy to test and compare the different registration types.

7.1 Rigid registration

Rigid FCPD is able to register 3D point sets of sizes in the order of 10^4 points within a few seconds, whereas the non-optimized CPD quickly will need a minute or more. Considering relatively small point set sizes (up to a few thousand points) multiple registrations can be performed every second, opening up for possible real-time registration applications. The Extended CPD (ECPD) algorithm [38] relies on a parallel FGT implementation to optimize the EM algorithm. No runtimes for ECPD were reported for rigid registration, but the FCPD implementation will most likely be faster as the computation of the $\boldsymbol{K}$ matrix is avoided.

The FCPD rigid registration runtimes make real-time rigid registration with CPD possible. For 3D point set sizes up to at least 3200, more than 10 registrations can be performed per second. Some additional time for preprocessing or downsampling may be required, depending on the specific application, but the possibility to use rigid FCPD for real-time registration seems promising. However, it does not match the newer state-of-the-art PSR registrations algorithms GMMReg [7] and FilterReg [40]. In [40], frame rates achieved by several rigid registration algorithms were reported for $M = N = 5000$, where GMMTree performed close to 20 frames per second (FPS), the different FilterReg implementations performed in the range of 20-40 FPS and CPD is reported to perform 0.23 FPS. The corresponding result for rigid FCPD is about 5 FPS, more than an order of magnitude faster than CPD, calculated on the computer stated at the beginning of the result chapter. It is worth noting that the GPU used by them is newer and faster than the one used for this project [55]. Thus, the FCPD is not the fastest, but still comparable and better than many of the ICP implementations also included in the test. The MLMD algorithm [39], which is based on EM and maximum likelihood and parallelized on GPU, is able to scale better for large point sets, but is beaten by rigid FCPD regarding runtime for 3D point sets up to roughly 3000 points.

7.2 Low-rank approximation

In the CPD paper [2], Myronenko and Song reported that about 60% of the total elastic registration runtime was consumed by the low-rank approximation. The specific low-rank approximation they used was not provided, so the details are unknown. However, it is clear from the elastic FCPD profiling (see figure 25) that the low-rank approximation scales poorly using Eigen's QR-decomposition methods, and becomes the computational bottleneck for large point sets. This is remedied by using a modified Gram-Schmidt QR-decomposition with reorthogonalization (MGSR). The cost seems to be a higher approximation error and an increased standard deviation, but is still good enough to achieve qualitative good registration results. Note that the reported errors are squared matrix norms and not corrected for increasing matrix sizes. A more thorough quantitative study should be done to determine whether the reduced accuracy is acceptable for any specific application. There might also exist more stable QR-decompositions with similar computational speed as MGSR.

The elastic registration results documented in this report were generated using a numerical rank $K = \sqrt{M}$. Studying figure 17, one can observe that the low-rank approximation error for increasing numerical ranks is found to not improve noticeably beyond a certain rank. In fact, it seems to be a threshold rank beyond which the approximation does not improve. Setting $K = \sqrt{M}$ is apparently very conservative for large point sets. Limiting the numerical rank will help prevent the low-rank approximation becoming a bottleneck in elastic registrations.

The Extended CPD (ECPD) algorithm [38] relies on a parallel eigendecomposition provided by the linear algebra library ARPACK [56] and FGT for low-rank approximation. It would be interesting to compare the low-rank approximation in FCPD directly with that implementation. The results for elastic registration will be compared in the next paragraph, but those results does unfortunately not settle which low-rank approximation is the most efficient.

7.3 Elastic registration

Reviewing the elastic FCPD profiling in figure 25, the calculation of the $\boldsymbol{W}$ matrix contributes most to the total runtime, along with the responsibility product calculations. The main contribution in FCPD specific for elastic registration is the GPU kernel introduced to calculate one of the matrix-matrix products in the calculation of $\boldsymbol{W}$. The elastic FCPD runtime significantly beats the fast CPD implementation, even when they are equipped with the same efficient low-rank approximation described in this thesis. In the ECPD paper [38], a downsampled moving point set was elastically registered to a large fixed point set. The runtime was 5 minutes and 40 seconds for $N = 2.9 \cdot 10^5$ and $M = 5625$, including downsampling by the method they describe. For comparison, elastic FCPD is able to register point sets with $M = 3.1 \cdot 10^5$ and $N = 5600$ in 20 seconds, excluded downsampling. The runtimes provided by elastic FCPD allow the registration of few point sets of reasonable size per second.

Because all the calculations in the FCPD implementation should be equivalent to the corresponding calculations in the CPD algorithm, with an exception of the low-rank approximation, there should be no discrepancies between the errors and parameter sensitivity of these implementations. For this reason little effort was put into error analysis and parameter settings, as this was covered by the original paper. Still, a sensitivity analysis of the elastic registration parameters β and λ is included. From this analysis, it appears that β, which weights the smoothness regularization of the algorithm, is the parameter that affects the results the most. With the squared distance error metric, the analysis suggests that small betas, $\beta = [1, 2]$, give the best registration results for the deformed points, while higher betas, $\beta > 5$, work best for the non-deformed points. Setting $\beta = \lambda = 2$ seemed to be the best compromise here. However, by visual assessment of the registration results (see the example showed in figure 24), the non-deformed points appeared to be noisier for large values of β. This may be caused by the specific deformation applied.

7.4 Applications and brain-shift correction

FAST CPD was finally tested on blood vessel centerlines from brain scans with the goal of providing a brain-shift correction. The most relevant comparable registration results were found in [9], where a rigid ICP algorithm was used for brain-shift correction, and [57], where a landmark-based variant of the CPD algorithm was used for elastic blood vessel registration. If one can get good results with an elastic PSR algorithm not depending on landmark priors, it will be an improvement for such applications. The results with elastic FCPD look promising. Unfortunately, the algorithm was only tested on a couple of patient data sets. A more extensive study using properly chosen landmarks should be tested to see if elastic CPD methods can compete with the state-of-the-art brain-shift correction methods.

There already exist CPD variants where the registration parameters are automatically tuned during registration. It would be interesting to study the possibility to include such methods in FCPD without reducing the achieved registration speed. Especially for new applications, or in applications with large variations in the point sets, automatic parameter optimization may provide better registration accuracy. The significantly improved runtime on the CPD algorithm by FCPD

can also open up a new range of applications, including real-time rigid registration.

8 Conclusion

FAST Coherent Point Drift (FCPD) provides runtimes for rigid, affine and elastic point set registration that are significantly faster than the original CPD algorithm. The speedup was achieved by optimization and parallelization on both the CPU and the GPU, without altering the properties and robustness of the original CPD algorithm. Registration runtime results were compared directly with open-source CPD and ICP implementations, as well as compared to some state-of-the-art point set registration methods. The rigid registration runtimes are fast enough to allow for real-time registration, and the elastic registration runtimes beats other comparable elastic registration algorithms. FCPD was tested on synthetic data, and on real centerline data for brain-shift correction. The FCPD implementation will be published open-source under a very permissive (BSD) licence, allowing anyone to use the code for any purpose they want.

References

[1] P. J. Gadomski, "Coherent point drift: C++ library for point set registration." https://github.com/gadomski/cpd, 2017. Accessed: 19.02.19.

[2] A. Myronenko and X. Song, "Point set registration: Coherent point drift," *Computing Research Repository (CoRR)*, 2009. http://arxiv.org/abs/0905.2635. Accessed: 13.08.18.

[3] E. Smistad, "Fast - framework for heterogeneous medical image computing and visualization." https://www.eriksmistad.no/fast/. Accessed: 17.11.18.

[4] H. Chui and A. Rangarajan, "A new algorithm for non-rigid point matching," *Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 44–51, 2000.

[5] S. Gold, A. Rangarajan, C. P. Lu, S. Pappu, and E. Mjolsness, "New algorithms for 2d and 3d point matching: pose estimation and correspondence," *Conference on Neural Information Processing Systems (NIPS)*, vol. 7, pp. 957–964, 1995.

[6] C. Mourning, S. Nykl, H. Xu, D. Chelberg, and J. Liu, "Gpu acceleration of robust point matching," 11 2010.

[7] B. Eckart, K. Kim, and J. Kautz, "Fast and accurate point cloud registration using trees of gaussian mixtures." https://arxiv.org/pdf/1807.02587.pdf, 07 2018. Accessed: 30.05.19.

[8] L. Wang, J. Chen, X. Li, and Y. Fang, "Non-rigid point set registration networks." https://arxiv.org/pdf/1904.01428.pdf, 04 2019. Accessed: 03.06.19.

[9] I. Reinertsen, F. Lindseth, C. Askeland, D. H. Iversen, and G. Unsgård, "Intra-operative correction of brain-shift," *Acta Neurochirurgica*, vol. 156, no. 7, 2014.

[10] E. Smistad, M. Bozorgi, and F. Lindseth, "FAST: framework for heterogeneous medical image computing and visualization," *International Journal of Computer Assisted Radiology and Surgery (CARS)*, vol. 10, no. 11, pp. 1811–1822, 2015. https://doi.org/10.1007/s11548-015-1158-5. Accessed: 19.11.18.

[11] E. Smistad, "FAST Source code." https://github.com/smistad/FAST. Accessed: 19.11.18.

[12] C. M. Bishop, *Pattern recognition and machine learning*. Information science and statistics, New York: Springer, 2006.

[13] J. McGonagle, G. Pilling, V. Tembo, A. Chumbley, E. Ross, and J. Khim, "Gaussian mixture model." https://brilliant.org/wiki/gaussian-mixture-model/. Accessed: 15.08.18.

[14] A. P. Dempster, N. M. Laird, and C. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society*, vol. 39, No. 1, 1977. http://web.mit.edu/6.435/www/Dempster77.pdf. Accessed: 29.11.18.

[15] T. Theoharis, G. Papaioannou, N. Platis, and N. M. Patrikalakis, *Graphics & Visualization: Principles and Algorithms*. Wellesley, Mass: A. K. Peters, Ltd, 2008.

[16] V. V. Williams, "Multiplying matrices in $O(n^{2.373})$ time." http://theory.stanford.edu/~virgi/matrixmult-f.pdf. Accessed: 18.05.19.

[17] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes (3rd ed.)*. Cambridge University Press, 2007.

[18] J. Dongarra and F. Sullivan, "The top 10 algorithms," *Comp. Sci. Eng.*, vol. 2, pp. 22–23, 2000. Accessed: 15.05.19.

[19] G. Golub and C. V. Loan, *Matrix Computations*. Johns Hopkins University Press, 1989.

[20] G. Strang, "The fundamental theorem of linear algebra," *The American Mathematical Monthly*, vol. 100, No. 9, pp. 848–855, 1993. Accessed: 16.05.19.

[21] C. Clement, S. Kailasa, and J. Khim, "Spectral theorem." https://brilliant.org/wiki/spectral-theorem/#proof-of-spectral-theorem. Accessed: 16.05.19.

[22] A. Tarantola, "Inverse problem." http://mathworld.wolfram.com/InverseProblem.html. Accessed: 27.03.19.

[23] A. Rathi, "Statistical learning." https://towardsdatascience.com/statistical-learning-for-data-science-b61b263c1196. Accessed: 30.03.19.

[24] V. Vapnik, "Principles of Risk Minimization for Learning Theory," *Neural Information Processing Systems (NIPS)*, 1992. https://papers.nips.cc/paper/506-principles-of-risk-minimization-for-learning-theory, Accessed: 27.03.19.

[25] F. Girosi, M. Jones, and T. Poggio, "Regularization Theory and Neural Networks Architectures," *Neural Computation*, vol. 7, pp. 219–269, 1995.

[26] J. Hunter and B. Nachtergaele, "Hilbert spaces." https://www.math.ucdavis.edu/~hunter/book/ch6.pdf. Accessed: 02.04.19.

[27] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online Learning with Kernels," *IEEE Transactions on signal processing*, vol. 52, no. 8, 2004. https://alex.smola.org/papers/2004/KivSmoWil04.pdf. Accessed: 02.04.19.

[28] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006. http://www.gaussianprocess.org/gpml/chapters/RW.pdf. Accessed 02.05.19.

[29] J. M. Fitzpatrick, D. L. G. Hill, and C. R. Maurer Jr., *Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis, Ch. 8*. SPIE Press, 2000.

[30] B. Maiseli, Y. Gu, and H. Gao, "Recent developments and trends in point set registration methods," *Journal of Visual Communication and Image Representation*, vol. 46, pp. 95–106, 2017. https://doi.org/10.1016/j.jvcir.2017.03.012. Accessed: 16.12.18.

[31] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE PAMI*, vol. 14, no. 2, pp. 239–256, 1992.

[32] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *International Journal of Computer Vision (IJCV)*, vol. 13, no. 2, pp. 119–152, 1994.

[33] H. Chui and A. Rangarajan, "A feature registration framework using mixture models," *IEEE Workshop on MMBIA*, pp. 190–197, 2000.

[34] B. Jian and B. Vemuri, "A robust algorithm for point set registration using mixture of gaussians," *Tenth IEEE International Conference on Computer Vision (ICCV'05)*, vol. 1 and 2, pp. 1246–1251, 2005.

[35] Y. Tsin and T. Kanade, "A correlation-based approach to robust point set registration," *European Conference on Computer Vision*, p. 558–569, 2004.

[36] C. Papazov and D. Burschka, "Stochastic global optimization for robust point set registration," *Computer Vision and Image Understanding*, vol. 115, pp. 1598–1609, 2011.

[37] Z. Zhou, B. Tong, C. Geng, J. Hu, J. Zheng, and Y. Dai, "Direct point-based registration for precise non-rigid surface matching using Student's-t mixture model," *Biomedical Signal Processing and Control*, vol. 33, pp. 10–18, 2017.

[38] V. Golyanik, B. Taetz, G. Reis, and D. Stricker, "Extended coherent point drift algorithm with correspondence priors and optimal subsampling," 03 2016.

[39] B. Eckart, K. Kim, A. Troccoli, A. Kelly, and J. Kautz, "MLMD: Maximum Likelihood Mixture Decoupling for Fast and Accurate Point Cloud Registration," *2015 International Conference on 3D Vision*, pp. 241–249, Oct 2015. Accessed: 06.06.19.

[40] W. Gao and R. Tedrake, "Filterreg: Robust and efficient probabilistic point-set registration using gaussian filter and twist parameterization," *CoRR*, vol. abs/1811.10136, 2018. Accessed: 03.06.19.

[41] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.

[42] A. Myronenko and X. Song, "On the closed-form solution of the rotation matrix arising in computer vision problems," *CoRR*, 2009. http://arxiv.org/abs/0904.1613. Accessed: 04.12.18.

[43] K. B. Petersen and M. S. Pedersen, "The matrix cookbook." http://www.math.uwaterloo.ca/~hwolkowi//matrixcookbook.pdf. Accessed: 14.12.18.

[44] V. Golyanik, B. Taetz, G. Reis, and D. Stricker, "Extended Coherent Point Drift Algorithm with Correspondence Priors and Optimal Subsampling, Supplementary Material." https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7477719, 2016. Accessed: 30.04.19.

[45] A. Myronenko, X. Song, and M. Á. Carreira-Perpiñán, "Non-rigid point set registration: Coherent point drift," *NIPS*, vol. 19, pp. 1009–1016, 2006. Accessed: 25.04.19.

[46] A. L. Yuille and N. M. Grzywacz, "The motion coherence theory," pp. 344–353, Dec 1988.

[47] R. Abbi, E. El-Darzi, C. Vasilakis, and P. Millard, "Analysis of stopping criteria for the em algorithm in the context of patient grouping according to length of stay," *2008 4th International IEEE Conference Intelligent Systems, IS 2008*, vol. 1, pp. 3–9, 2008.

[48] P. Pacheco, *An Introduction to Parallel Programming*. Elsevier Inc., 2011.

[49] E. Smistad, T. L. Falch, M. Bozorgi, A. C. Elster, and F. Lindseth, "Medical image segmentation on gpus – a comprehensive review," *Medical Image Analysis*, vol. 20, no. 1, pp. 1 – 18, 2015.

[50] S. U. C. G. Laboratory, "Stanford bunny data set." http://graphics.stanford.edu/data/3Dscanrep/.

[51] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev., Survey and Review section*, vol. 53, no. 2, pp. 217–288, 2011. https://arxiv.org/abs/0909.4061, Accessed: 24.04.19.

[52] W. Gander, "Algorithms for QR-Decomposition." http://people.inf.ethz.ch/gander/papers/qrneu.pdf, 1980. Accessed: 31.05.19.

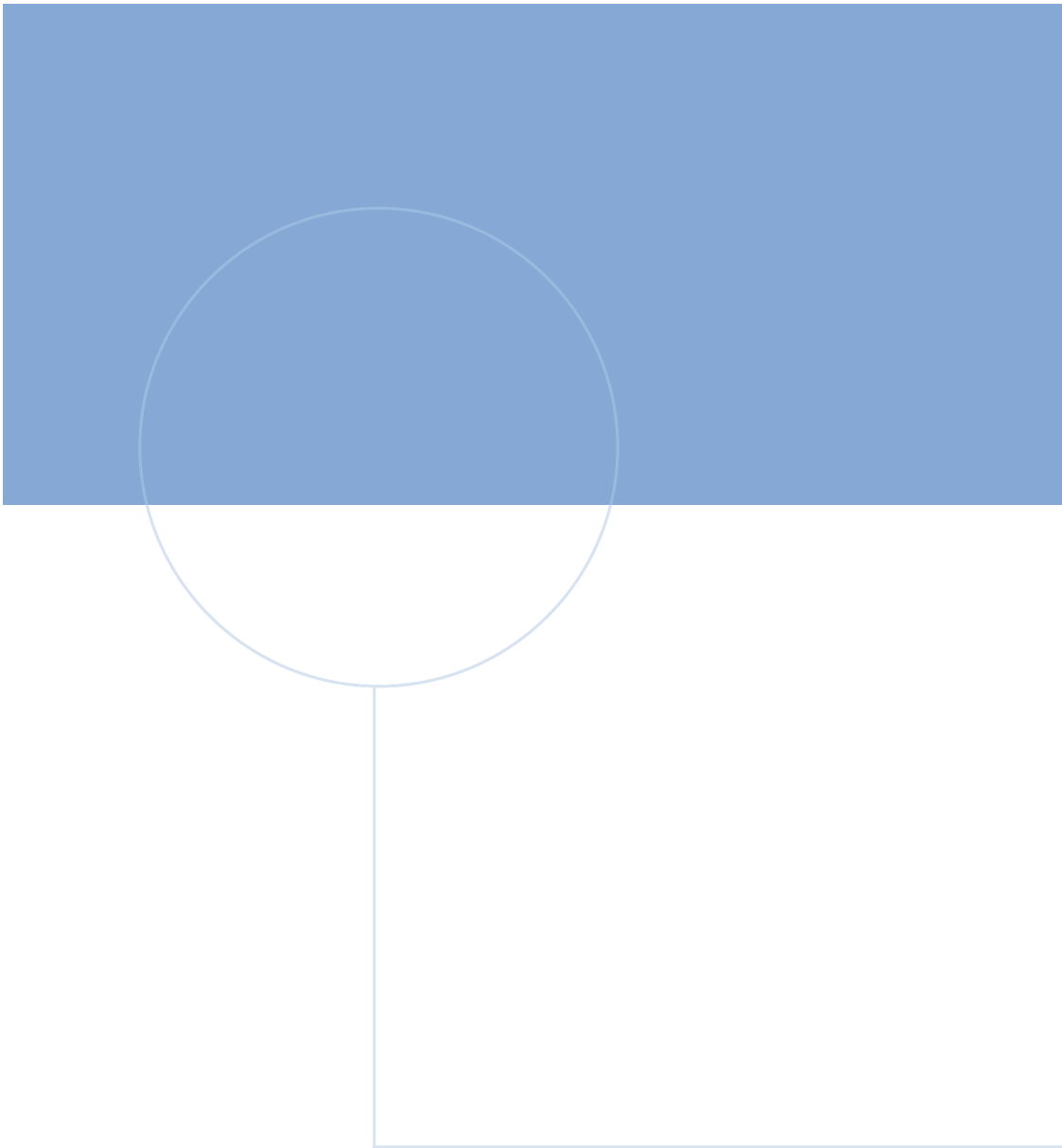
[53] "The fast gauss transform," *Society for Industrial and Applied Mathematics (SIAM)*, vol. 12, no. 1, pp. 79 – 94, 1991.

[54] "Eigen: Benchmark of dense decompositions." https://eigen.tuxfamily.org/dox/group__DenseDecompositionBenchmark.html. Accessed: 06.05.19.

[55] "GPU Benchmark." https://gpu.userbenchmark.com/Compare/Nvidia-Titan-Xp-vs-AMD-R9-Fury/m265423vs3509. Accessed: 06.06.19.

[56] "ARPACK - Arnodli Package." https://www.caam.rice.edu/software/ARPACK/. Accessed: 06.06.19.

[57] Y. Hu, E. Rijkhorst, R. Manber, D. Hawkes, and D. Barratt, "Deformable Vessel-Based Registration Using Landmark-Guided Coherent Point Drift.," *Medical Imaging and Augmented Reality (MIAR)*, 2010.



NTNU
Kunnskap for en bedre verden

SINTEF