Point Cloud Registration in Augmented Reality using the Microsoft HoloLens

Kevin Kjellén
Master of Science Thesis in Electrical Engineering

Point Cloud Registration in Augmented Reality using the Microsoft HoloLens

Kevin Kjellén
LiTH-ISY-EX–18/5160–SE

Supervisor: Doktorand Felix Järemo Lawin
ISY, Linköpings universitet
Ingenjör Johan Falk
SICK IVP

Examiner: Maria Magnusson
ISY, Linköpings universitet

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Copyright © 2018 Kevin Kjellén
Abstract

When a Time-of-Flight (ToF) depth camera is used to monitor a region of interest, it has to be mounted correctly and have information regarding its position. Manual configuration currently require managing captured 3D ToF data in a 2D environment, which limits the user and might give rise to errors due to misinterpretation of the data. This thesis investigates if a real time 3D reconstruction mesh from a Microsoft HoloLens can be used as a target for point cloud registration using the ToF data, thus configuring the camera autonomously. Three registration algorithms, Fast Global Registration (FGR), Joint Registration Multiple Point Clouds (JR-MPC) and Prerejective RANSAC, were evaluated for this purpose.

It was concluded that despite using different sensors it is possible to perform accurate registration. Also, it was shown that the registration can be done accurately within a reasonable time, compared with the inherent time to perform 3D reconstruction on the Hololens. All algorithms could solve the problem, but it was concluded that FGR provided the most satisfying results, though requiring several constraints on the data.
Acknowledgments

I would like to thank SICK IVP AB for the opportunity to do this master’s thesis project, as well as a special thanks to my supervisor Johan Falk for his great help with the Universal Windows Platform and C# issues. Also, a big thank you to my supervisor Felix Järemo Lawin and examiner Maria Magnusson at Linköping University.

Last but not least, I would like to thank my family and friends for supporting me and helping me throughout my studies.

Linköping, June 2018
Kevin Kjellén
# Contents

Notation ix

1 Introduction 1
  1.1 Background ................................................. 1
  1.2 Problem Description ........................................ 2
  1.3 Aim ......................................................... 3
  1.4 Limitations ................................................ 3
  1.5 Hardware .................................................... 3
    1.5.1 The SICK Time-of-Flight Camera ....................... 3
    1.5.2 The Microsoft Hololens ............................... 4
  1.6 Related Literature .......................................... 6

2 Theory 7
  2.1 Stereo Camera 3D Reconstruction .......................... 7
    2.1.1 Finding Correspondences ............................... 7
    2.1.2 Triangulation ........................................... 9
    2.1.3 Disparity Maps ......................................... 11
  2.2 Features and Descriptors .................................... 11
    2.2.1 (Fast) Point Feature Histograms ...................... 11
  2.3 Registration Algorithms ..................................... 17
    2.3.1 Fast Global Registration ............................. 17
    2.3.2 Joint Registration Multiple Point Clouds ............ 21
    2.3.3 Prerejective RANSAC .................................. 25

3 Method 29
  3.1 Feature Selection ......................................... 30
  3.2 Registration Algorithm Selection .......................... 30
  3.3 Test Platform ............................................... 32
    3.3.1 Data Extraction ....................................... 32
    3.3.2 Data Pre-processing .................................... 33
    3.3.3 Normal and Feature estimation ........................ 34
    3.3.4 Registration ............................................. 36
  3.4 Hololens implementation ................................... 40
## Contents

3.5 Assumptions and Simplifications ........................................... 42

4 Results ................................................................. 43
  4.1 Data sets ........................................................................ 43
  4.2 FGR ............................................................................ 45
  4.3 JR-MPC ......................................................................... 45
  4.4 Prerejective RANSAC .................................................... 45
  4.5 Comparison ................................................................... 48

5 Discussion ................................................................. 51
  5.1 FGR ............................................................................ 51
  5.2 JR-MPC ......................................................................... 52
  5.3 Prerejective RANSAC .................................................... 53
  5.4 HoloLens implementation ................................................ 53

6 Conclusions and Future Work ................................................. 55
  6.1 Conclusions .................................................................... 55
  6.2 Future Work .................................................................... 56

A Timing Plots ................................................................. 59
  A.1 FGR ............................................................................ 59
  A.2 JR-MPC ......................................................................... 60
  A.3 Prerejective RANSAC .................................................... 61

Bibliography ................................................................. 63
# Notation

## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SICK</td>
<td>SICK IVP AB in Linköping, subsidiary of SICK AG</td>
</tr>
<tr>
<td>AR</td>
<td>Augmented Reality, merging real and virtual worlds</td>
</tr>
<tr>
<td>PCL</td>
<td>Point Cloud Library, open source C++ code library</td>
</tr>
<tr>
<td>ToF</td>
<td>Time-of-Flight, depth imaging method</td>
</tr>
<tr>
<td>HoloLens</td>
<td>The Microsoft HoloLens, holographic AR headset</td>
</tr>
<tr>
<td>FPFH</td>
<td>Fast Point Feature Histograms, geometric descriptor</td>
</tr>
<tr>
<td>ICP</td>
<td>Iterative Closest Point, local registration algorithm</td>
</tr>
<tr>
<td>FGR</td>
<td>Fast Global Registration</td>
</tr>
<tr>
<td>JR-MPC</td>
<td>Joint Registration Multiple Point Clouds</td>
</tr>
<tr>
<td>RANSAC</td>
<td>Random Sample Consensus, registration algorithm</td>
</tr>
<tr>
<td>GMM</td>
<td>Gaussian Mixture Model, probabilistic model</td>
</tr>
<tr>
<td>EM</td>
<td>Expectation Maximization, probabilistic update</td>
</tr>
</tbody>
</table>
This document will explain the problem, theory and method for aligning Time-of-Flight (ToF) data from a SICK camera to a stereo camera 3D reconstruction from a Microsoft HoloLens.

Aligning point clouds has been explored extensively in the last 20 years, but only now is it possible to integrate both data gathering and actual alignment in a headset, with the Microsoft HoloLens. This thesis will explore what methods of alignment work for the data of a ToF depth camera and the data from the HoloLens. Both speed and accuracy is required, as the processing power is more limited in a HoloLens than in stationary systems.

The goal is to integrate the alignment in an app which in real time will visualize the depth image layered above the real world.

1.1 Background

This thesis will be performed at SICK IVP AB (hereinafter called SICK) in Linköping, subsidiary of the global company SICK AG, originating in Germany. SICK AG produces everything from laser scanners and depth cameras to gas sensors and integrated circuitry, while IVP in Linköping is mostly focused on machine vision and software solutions utilizing the sensors.

Oftentimes when installing a SICK ToF depth camera, the goal is to monitor some volume for traffic. It might be to stop a machine if a hand enters, or to automate some manufacturing process. The camera has internal computing power that can process the image information, but to monitor a room specific volume the camera needs to be calibrated relative to its position. This is often done manually, which is labour intensive and risk being done incorrectly.
1.2 Problem Description

Normally, calibration of a SICK depth camera has to be done by manually placing the camera in some sort of virtual context. You then have to assign a virtual volume which to monitor, compute at what depths the volume lies relative to the camera’s sensor, and send this information to the camera. One of the issues with the current system is that there is no way of getting feedback of how the virtual context lies relative to the real room without having a 3D model of it. Calibration of the camera relies on the human operator to try to place a volume while seeing only the depth image of the camera. This is both time consuming and prone to errors.

The problem and mission of this thesis is to find an automatic way of calibrating a SICK ToF depth camera for monitoring a room-specific volume, without the need for tedious manual calibration. To accomplish this, the processing power and real-time 3D reconstruction capabilities of the Microsoft HoloLens are utilized. With the 3D-model that the Microsoft HoloLens creates, paired with the depth camera image of the ToF camera, we try to find the rigid transform which connects these Point Clouds; this is known as Point Cloud Registration.

Point cloud registration is the act of fitting a point cloud on to a 3D structure (the target), depicting (partially) the same scene, in an attempt to find the rigid transform which links their coordinate systems together. Assuming that the point clouds have a known scale correspondence, the desired transform will have six degrees of freedom; three for rotation and three for translation.

Figure 1.1: Illustration of the registration problem, with a depth camera point cloud (coloured with a jet colormap), and the Microsoft HoloLens 3D-reconstruction as a mesh. The point cloud is arbitrarily placed to the left, and manually aligned to the right.

Figure 1.1 shows the depth camera point cloud and the Microsoft HoloLens 3D-reconstruction, where the point cloud is arbitrarily placed to the left, and
correctly aligned to the right. The goal of this thesis is to use some method of point cloud registration to automatically place the point cloud as the right part of the figure demonstrates.

The point cloud from a stationary depth camera is often referred to as 2.5D data. This is because, while the points are in 3 dimensions, all points are viewpoint dependent, as they originate from a projection to a 2D plane (the camera plane).

Point cloud registration is a quite well explored subject, where the often referred baseline method ICP [2] was published in 1992 and several thousand papers have been released since. This thesis will focus on applying the current theory base on a new platform, the Microsoft HoloLens, and examine which methods produce satisfying results.

1.3 Aim

The main questions this thesis sets out to answer are:

- Is the HoloLens and the SICK ToF camera data reliable and dense enough to be subject to robust registration?
- Are the registration algorithms available today sufficient in adequately solving the problem?
- Can the performance of the above mentioned algorithms be optimized to the point where they can be used in real time on the HoloLens headset?

1.4 Limitations

This thesis will be limited by a few factors. Some methods of point cloud registration will not be available as the data which is used for registration lacks color information. The density of 3D points is also both varying and occasionally sparse, which might worsen the results of some algorithms that would otherwise be sufficient. Also, if the HoloLens were to malfunction in any way, repairs or replacement might be impossible to find, thus stopping the project entirely.

1.5 Hardware

For this project two main hardware systems will be used. A SICK ToF depth camera, generating depth images, and a Microsoft HoloLens, with processing, visualization and 3D-reconstruction capabilities.

1.5.1 The SICK Time-of-Flight Camera

The used depth camera is the V3S100-1AABAB model in the Visionary-T CX family of SICK cameras, with 176 x 144 pixels resolution and a 69° x 56° field of
view in the horizontal and vertical direction respectively. The camera is shown in figure 1.2.

\[ \text{Figure 1.2: The V3S100-1AABAAAB ToF camera} \]

The camera uses Time-of-Flight, where a phase shifted Continuous Wave method with 4 tabs, measuring the phase change of a light wave when it returns, is used. This approach has a modular error challenge; phase is periodic. This has been solved through property of SICK, resulting in measurements between 0.5 m and 7.2 m, where the phase is unambiguous; measurements in any other range is set at missing data and ignored.

### 1.5.2 The Microsoft Hololens

The Microsoft Hololens (hereinafter called the HoloLens), shown in figure 1.3, is an Augmented Reality (AR) headset utilizing holographic technology to merge reality with a virtual environment.

The headset has two pairs of stereo cameras, as well as a ToF camera. Stereo cameras are two ordinary RGB cameras placed at a very close, known distance from each other. The data from these sensors are processed in an internal computer inside the headset, running on Windows Universal Platform and Windows 10.

To uphold the illusion to the user that the virtual objects are somewhere in the scene the HoloLens needs to render them in stereo, e.g. from slightly different perspectives for the two eyes. Also, to make the objects appear spatially stationary, the coordinate system of the scene, rather than one relative to the headset, is needed. To accomplish this the HoloLens uses real time 3D reconstruction from two pairs of stereo cameras to create a 3D mesh of the scene and the objects in it (explained in section 2.1). Virtual objects can then be placed and kept stationary in the virtual scene, and as the HoloLens moves through the scene coordinate system rather than the scene moving around the HoloLens, the objects
Figure 1.3: The Microsoft HoloLens, shown with stereo 3D mapping cameras on each side, and the ToF hand gesture camera in the middle.

seem stationary in the actual scene as well. This system also allows occlusion of virtual objects by real ones, as their position and coverage is known from the 3D reconstruction.

To interact with the HoloLens there is a small ToF depth camera mounted on the headset, used to estimate the hand pose of the user. Different poses are assigned an action in the interface. There is also the possibility to connect a keyboard to the HoloLens through Bluetooth, which can serve as an alternative input method.

Applications on the HoloLens has the complete 3D mesh available for use, as well as built in ways of extracting specific parts of it. An example of the quality of the data from the HoloLens is shown in figure 1.4, where human perception can easily distinguish the features of the scene.

Figure 1.4: An example of a mapped scene extracted from the HoloLens on the left, with a zoomed in version to the right.
1.6 Related Literature

The HoloLens is still a fairly new technology, and the specific use of performing point cloud registration on it is entirely new. Looking beyond point cloud registration, several articles and theses has been written on the user aspect of the HoloLens and augmented reality, for example in medical training [1]. The base application from which this thesis is built, by I. Jansson [13], explored the user aspects of overlaying a depth data point cloud over the real world using AR.

On the subject of point cloud registration many cite Iterative Closest Points (ICP) [2], as the ‘base’ algorithm. The ICP approach computes the rigid transformation that minimizes the distance between the assumed point correspondences, which is iterated with updating the assumed correspondences by looking for spatial nearest neighbours. While ICP is a robust algorithm for small translation and rotation errors, it is prone to getting stuck in local minima. Thus, it is very sensitive to how good the initial alignment is. It is often used as a finalizing refinement step, where other methods perform the initial alignment. There are also several attempts in increasing the robustness of ICP when faced with large initial errors [5], [21].

R. B. Rusu et al. created Point Feature Histograms (PFH), introduced in [18] and [17], which tries to extract invariant high dimensional features describing the geometry of local areas. Later R. B. Rusu et al. tried to improve the computational performance by removing redundancy in the descriptors with [19], known as Fast Point Feature Histograms (FPFH). These descriptors can be used by a registration algorithm looking for global correspondences, for example Fast Global Registration (FGR) [23], enabled by the high dimensional invariant features which are more likely to be unique. FGR seeks to avoid recomputation of correspondences while still finding global solutions.

There is also the probabilistic approach, which model the distribution of 3D-points as a density function. The registration is then performed by either maximizing a similarity measure between the density models [12], or applying the Expectation Maximization (EM) algorithm to iteratively find the registration parameters [8], [6]. The density models in [8] tracks the distribution means and isotropic variances. This was extended in [6] to also include color information.

A surprisingly simple and effective method often employed in pose estimation is Random Sample Consensus [9], more commonly known as RANSAC, which seeks the transform which maximizes the ratio of inliers through an iterative random correspondence search. The method was sped up in [4] by offering an earlier pose rejection, which eliminated the need to count inliers to perform an initial feasibility check of the proposed transform.
This chapter explains the necessary theory for performing point cloud registration with ToF depth images and a HoloLens 3D reconstruction. The theory included here will try to cover the basics of how the steps from image sensors to completed registration might work. As the HoloLens is a closed system, there are a few steps where we can only speculate what is used.

2.1 Stereo Camera 3D Reconstruction

Extracting 3D points from stereo cameras consists of two major parts. First, to find points in the two images which correspond to the same 3D point, and second, to find the 3D coordinates from the point pairs through triangulation.

2.1.1 Finding Correspondences

When looking for a correspondence for an image point \( y_L \) in the left camera of a stereo camera pair there are limitations as to where these correspondences might appear. Figure 2.1 shows a situation where the point \( y_L \) may originate from any of the 3D points \( x_1, x_2, x_3, x_4 \), while in the right image plane these 3D points correspond to different image points.

To express this in mathematical terms, some notation is needed. Given the camera matrices \( C_L, C_R \) and camera centres \( n_L, n_R \) of a stereo camera pair, one can define the epipolar points, or epipoles, of the system as:

\[
\begin{align*}
    e_{L,R} & \sim C_L n_R, \\
    e_{R,L} & \sim C_R n_L,
\end{align*}
\]

where \( \sim \) denotes the equivalence relation between projective elements, which is equality apart from scale for homogeneous coordinates. A complete explanation
Figure 2.1: Epipolar geometry of a stereo camera pair with cameras matrices $C_L, C_R$, camera centers $n_L, n_R$, image points $y_L, y_R$ and 3D points $x_1, x_2, x_3, x_4$. Here $e_{L,R}, e_{R,L}$ denotes the left-to-right and right-to-left epipolar points respectively, forming the baseline $L_B$.

of homogeneous coordinates can be found in [16, p.65]. The epipoles are the image points in the left and right image plane if a line is drawn from one camera centre to the other. This line is known as the baseline $L_B$. If any other point in the left image, say $y_L$, is chosen, the correspondence for $y_L$ in the right camera is not unambiguous. Instead the possible correspondences form a line in the right image plane, which can be expressed as

$$l_R \sim e_{R,L} \times (C_R C_L^+ y_L) = [e_{R,L}] \times C_R C_L^+ y_L, \quad (2.1)$$

where $[e_{R,L}] \times$ denotes the cross product operator of $e_{R,L}$ and $C_L^+$ the pseudoinverse of $C_L$. This can be condensed according to

$$F \sim [e_{R,L}] \times C_R C_L^+. \quad (2.2)$$

The matrix $F$ is called the fundamental matrix, and it describes the epipolar geometry of the stereo camera system. This lets equation (2.1) be written as:

$$l_R \sim F y_L. \quad (2.3)$$

If we define a corresponding point to $y_L$ as $y_R$, which has to lie on the corresponding epipolar line in the right image plane, it follows that

$$0 = y_R \cdot l_R = y_R^T l_R = y_R^T F y_L. \quad (2.4)$$

Equation (2.4) is known as the epipolar constraint, and it describes that any correspondence to an image point $y_L$ must lie on the epipolar line $l_R$. While this constraint is necessary, it is not sufficient in saying whether or not the points are true correspondences. Figure 2.1 shows that $y_L$ and all the points $y_{R,1\ldots4}$ fulfil
the epipolar constraint, but they do not necessarily originate from the same 3D point. This is due to the fact that all the 3D points lie in the same epipolar plane relative to the cameras.

So while the epipolar constraint is not a sufficient condition, it is necessary for true correspondences, and narrows the search for correspondences from the whole image to along a line. As with any real world application this condition will seldom be exact, and oftentimes the condition is relaxed to $y_RFy_L \approx 0$ due to noise and discrete sampling. This corresponds to searching in a narrow line strip in the image, rather than a perfect line.

Searching along these lines can be done in many different ways. A popular version is to extract points of interest in the images. If an area around a point has texture, or is not intrinsically 1D, it is often seen as a promising candidate. This might be a corner of a table, or a high frequency pattern on the ground. How to describe this point in a way that is easily matched with its potential correspondence in the other image is subject to much debate. A simple method could be to find interest points through Harris corner detection [10] in both images, and perform a correlation search (block matching) for all interest points in one image with every point along their respective epipolar lines in the other image. An even simpler method would be to do this for absolutely all points along the line, which is simple but very computationally expensive. This method is neither scale nor rotation invariant, but as the orientation of the cameras are known in this case it is not a necessity.

A more advanced method would be to extract SIFT descriptors [14], which are scale, translation and rotation invariant features made by creating image gradient histograms of an area around an interest point. This method might prove more robust when searching for correspondences, at the cost of complexity. For further reading of features and 2D image descriptors, see [15], which performs an in depth evaluation of several popular features.

The method used by the HoloLens is unknown, as their system is entirely hardware integrated in what they call the Holographic Processing Unit (HPU), and kept a secret. This section is merely meant to shine some light on the general process of finding correspondences, and what methods might be used.

### 2.1.2 Triangulation

When two tentative correspondences have been found, the first thing is to check the epipolar constraint (equation (2.4)). If this is (approximately) fulfilled we can proceed with the triangulation. If the epipolar constraint is exactly fulfilled, e.g. $y_RFy_L = 0$, the point to triangulate will simply be where the two projection lines of $y_L, n_L$ and $y_R, n_R$ meet. This point can be found by first extracting $x_L \sim C_L^x y_L$ and $x_R \sim C_R^x y_R$, which are points on the respective projection lines in the world coordinate system, in homogeneous coordinates. Solving the equation

$$s\bar{x}_L + (1-s)\bar{N}_L = t\bar{x}_R + (1-t)\bar{N}_R,$$

for $s, t$ in Cartesian coordinates will give the crossing point when inserted back into the equation on either side. In real applications this equation will rarely
have a solution. Noise in measurements will result in the lines never actually crossing. Figure 2.2 demonstrates a situation where the lines never cross, but we still want a solution to the problem; the point which is a "best approximation" of the crossing of the lines. There are several methods for doing this, where both geometric errors and algebraic errors can be defined for the problem. The method shown in figure 2.2 is the mid-point method. Here we try to find the middle point of the shortest line that can be drawn between the two projection lines. This can be solved by again computing \( x_L \sim C_L y_L \) and \( x_R \sim C_R y_R \), and instead defining two parameterizations of the projection lines as

\[
    x_L(s) = s x_L + (1 - s) n_L, \quad x_R(t) = t x_R + (1 - t) n_R.
\]

We can then define a geometric error function,

\[
    \varepsilon_{MP} = ||x_L(s) - \bar{x}_M||^2 + ||x_R(t) - \bar{x}_M||^2.
\]

Computing the tangent vectors of the projection lines \( \bar{t}_L = x_L - \bar{n}_L \) and \( \bar{t}_R = x_R - \bar{n}_R \) and solving for \( \begin{pmatrix} s_0 \\ t_0 \end{pmatrix} \) in the corresponding normal equation,

\[
    \begin{pmatrix} \bar{t}_L & -\bar{t}_R \end{pmatrix} \begin{pmatrix} \bar{t}_L & -\bar{t}_R \end{pmatrix} \begin{pmatrix} s_0 \\ t_0 \end{pmatrix} = \begin{pmatrix} \bar{t}_L & -\bar{t}_R \end{pmatrix} \begin{pmatrix} \bar{n}_L & -\bar{n}_R \end{pmatrix},
\]

gives us the two points, \( \bar{x}_L(s_0) \) and \( \bar{x}_R(t_0) \), that span the shortest line that can be drawn between the projection lines. The midpoint is thus given by:

\[
    \bar{x}_M = \frac{\bar{x}_L(s_0) + \bar{x}_R(t_0)}{2},
\]
which minimizes equation (2.7).

The mid-point method is just one (geometric) way of approximately triangulating 3D points from stereo correspondences. Additional geometric methods, such as Optimal Triangulation, and algebraic errors which can be minimized with for example the Inhomogeneous or Homogeneous methods, are described in [16].

What error function or triangulation method the HoloLens uses is once again unknown, and this section is merely meant to give some insight into the triangulation problem.

### 2.1.3 Disparity Maps

As the HoloLens creates a whole landscape of points, it is possible that it creates something called a Disparity Map. In the case of a pair of calibrated side-by-side stereo cameras, a disparity map will store the horizontal shift (in pixel coordinates) of each point in the scene, as seen from the two stereo cameras. The distance to any point can be directly extracted from its horizontal shift, similar to how humans see depth. This can be seen as a correspondence finding and triangulating for each point in the image, but it also lets us apply consistency checks between triangulated points.

There are additional smart ways of exploiting this method, such as triangulating points sparsely, and perform post-process edge filtering, as in [20]. This filters and interpolates depth values based on the few triangulated points, while preserving edges in the original image.

### 2.2 Features and Descriptors

A descriptor is a way to describe a certain data point in a data set in such a way that it contains more relevant information for the problem that needs solving. A local feature is an interest point combined with a descriptor. The 3D coordinate can be seen as a descriptor for a 3D point. Thus, it is possible to calculate the distance between points in 3D space. However, it is not possible to compare and see if it is the correct corresponding point of another point in a data set which represents the same scene. Instead we need something that does not depend on its 3D position, called translation and rotation invariant, but describes the surroundings very accurately. Below is a description of the features used in this thesis.

#### 2.2.1 (Fast) Point Feature Histograms

Point Feature Histograms (PFH), [17][18], was developed as a 3D point feature, with the intention of assigning a point to what kind of primitive shape it was a part of. This might be a plane, an edge, a cylinder etc.. For this reason the feature tries to describe what the local geometrical area around the point looks like. Consequently, the features were developed to be both translation and rotation invariant. To improve efficiency, the features were further developed in [19], attempting to remove redundancy. The new features, known as Fast Point Feature...
Histograms (FPFH), are popular among point cloud registration algorithms due to their speed, robustness as well as their transform invariance, which is needed for correspondence matching between unaligned point clouds. Below is a mathematical description of the original PFH features, followed by what changed and improved with FPFH.

**PFH**

This section is largely based on [17] and [18], which both describe the Point Feature Histograms. The idea is to give a complete theory of the features, while remaining relatively brief.

Point Feature Histograms are computed by extracting four features for each point pair in a neighbourhood around a target point, and binning the ratio of how many point pair features falls in each specified feature interval in a 16-bin histogram. This histogram then describes the geometrical neighbourhood around the target point.

Assume we have a point cloud \( \mathbf{P} = \{\mathbf{p}_i, \mathbf{n}_i| i = 1 \ldots N\} \), where \( \mathbf{p}_i \) is a point in the set and \( \mathbf{n}_i \) its normal. How to estimate normals can be found in section 3.3.3. Around each \( \mathbf{p}_i \), select the group of points within a radius \( r \) of the point, known as the \( k \)-neighbourhood (\( k \) refers to the number of neighbours inside the radius \( r \)). Figure 2.3 shows a radius \( r \) around point \( \mathbf{p}_i \), where 5 points are currently enclosed. For every pair of points \( (\mathbf{p}_j, \mathbf{p}_l), j \neq l, l < j \) within this radius, represented in the figure by a line connection, select a source point \( \mathbf{p}_s \) and a target point \( \mathbf{p}_t \), found by:

\[
\begin{align*}
\text{if } & n_j \cdot (\mathbf{p}_l - \mathbf{p}_i) < n_l \cdot (\mathbf{p}_j - \mathbf{p}_i) \text{ then} \\
& \mathbf{p}_s = \mathbf{p}_j, \: \mathbf{p}_t = \mathbf{p}_l \\
\text{else} & \\
& \mathbf{p}_s = \mathbf{p}_l, \: \mathbf{p}_t = \mathbf{p}_j
\end{align*}
\]

*Figure 2.3: PFH with the \( k \)-neighbourhood of a point \( \mathbf{p}_i \), where each point-pair combination that computes the 4 PFH features are represented by a line connection. Inspired by [19].*
which says that the source point is the one with a smaller angle between its normal the the line between the pair. From this, define a new coordinate system, a so called Darboux frame, with origin at \( p_s \):

\[
\mathbf{u} = \mathbf{n}_s, \quad \mathbf{v} = (\mathbf{p}_t - \mathbf{p}_s) \times \mathbf{u}, \quad \mathbf{w} = \mathbf{u} \times \mathbf{v},
\]

(2.10) followed by a normalization to form an orthonormal basis. A visualization of the new coordinate system can be seen in figure 2.4. From \( p_s \) and \( p_t \), as well as their

\[ \text{Figure 2.4: Darboux frame } (\mathbf{u}, \mathbf{v}, \mathbf{w}) \text{, an orthonormal coordinate system created from two points } \mathbf{p}_s \text{ and } \mathbf{p}_t \text{ with normals } \mathbf{n}_s \text{ and } \mathbf{n}_t \text{ respectively. The vector } \mathbf{p}_t - \mathbf{p}_s \text{ is in the plane spanned by } \mathbf{u} \text{ and } \mathbf{w}. \text{Inspired by [17].} \]

new Darboux coordinate system, four features can be calculated as:

\[
\begin{align*}
  f_1 &= \mathbf{v} \cdot \mathbf{n}_t \\
  f_2 &= \|\mathbf{p}_t - \mathbf{p}_s\| \\
  f_3 &= \frac{\mathbf{u} \cdot (\mathbf{p}_t - \mathbf{p}_s)}{\|\mathbf{p}_t - \mathbf{p}_s\|} \\
  f_4 &= \text{atan2}(\mathbf{w} \cdot \mathbf{n}_t, \mathbf{u} \cdot \mathbf{n}_t).
\end{align*}
\]

(2.11) Dividing the interval of each of the four features in two subdivisions gives the possibility of categorizing them in a 2\(^4\)-bin histogram. Each point \( \mathbf{p}_i \) will be described by such a histogram, where each bin in the histogram contains the percentage of point pairs \((\mathbf{p}_j, \mathbf{p}_l)\), \( j \neq l, \ l < j \) within radius \( r \) whose features, calculated as in (2.11), falls into the interval specified by that bin. Which bin to select for each calculated value can be specified as

\[
\text{bin} = \sum_{i=1}^{4} 2^{i-1} \text{step}(s_i, f_i),
\]

(2.12)
where \( \text{step}(s_i, f_i) \) is the step function, defined as 0 if \( f_i < s_i \), otherwise 1. Choose each \( s_i \) in the middle of the interval for each feature, which is 0 for \( f_1, f_3 \) and \( f_4 \), and \( r \) for \( f_2 \). This can be seen as defining a 4 bit number \( b_1 b_2 b_3 b_4 \) and letting each bit represent which interval was hit for that feature. The 4 bit number would represent which bin to choose for that point pair.

Intuitively, these features all represent some geometrical relationship between the points in close proximity to \( p_i \):

- \( f_1 \): Being a dot product of normalized vectors, \( f_1 \) is the cosine of the angle between vectors \( \mathbf{v} \) and \( n_t \). This gives an interval of \([-1, 1]\).
- \( f_2 \): A distance measure between point pairs. For all point pairs, this will give some sense of how scattered the neighbourhood is. This gives an interval of \([0, 2r]\).
- \( f_3 \): Similarly to \( f_1 \), \( f_3 \) is a normalized dot product, and represents the cosine of the angle between vectors \( \mathbf{u} \) and \( p_t - p_s \). This gives an interval of \([-1, 1]\).
- \( f_4 \): \( \text{atan2} \) of these dot products describes the angle between \( \mathbf{w} \) and the projection of \( n_t \) on the plane spanned by \( (\mathbf{u}, \mathbf{w}) \). This gives an interval of \((-\frac{\pi}{2}, \frac{\pi}{2})\).

The choice of using two subdivisions in each feature was motivated largely by looking at the very large increase in computations with increasing subdivisions. Increasing the subdivisions from 2, which gives \( 2^4 = 16 \) bins, to 3, we would instead end up with \( 3^4 = 81 \) bins, roughly five times as many. R. B. Rusu et al. [17] called the large increase in bins intractable.

### FPFH

This section describes Fast Point Feature Histograms [19], which tries to speed up the computation and remove redundancy of the previous features, Point Feature Histograms. Below is a description of FPFH, and what has changed from PFH.

First, define the concept Simple Point Feature Histograms, which reduces the pair combinations of the PFH to only those pair combinations that include the target point. Figure 2.5 shows the reduced number of combinations of the SPFH compared to PFH in figure 2.3, where each point-pair combination is illustrated by a line connection.

Let \( \text{SPFH}(\mathbf{p}_i) \) denote the SPFH calculated for a point \( \mathbf{p}_i \). After computing the SPFH of a point, compute the SPFH for each neighbour within radius \( r \) of the target points \( \mathbf{p}_i \). This is illustrated in figure 2.6, where each neighbour of \( \mathbf{p}_i \) is included in the first SPFH, and all the neighbours \( \mathbf{p}_{k1...5} \) subsequently compute their own SPFH. This means that several paths will be included twice, namely the connections between the target point and the first layer of neighbours, as well as the paths between those initial neighbours who lie within a radius \( r \) of each other.

\(^1\text{atan2}, \text{first defined for use in programming, is now commonly also used in academic work.} \text{atan2}(y, x) \text{ is the principal branch of arctan}(y/x) \text{ for } x > 0. \text{ For } x < 0 \text{ its value is instead taken from the appropriate branches of the inverse tan-function in the intervals } (-\pi, -\pi/2) \text{ and } (\pi/2, \pi).\)
2.2 Features and Descriptors

Figure 2.5: SPFH, with the $k$-neighbourhood of a point $p_i$, where each point-pair combination that computes the features are represented by a line connection. Inspired by [19]

In 2.6 this is illustrated as lines with colour gradients. The FPFH of a point $p_i$ is then defined as

$$FPFH(p_i) = SPFH(p_i) + \frac{1}{k} \sum_{j=1}^{k} \frac{SPFH(p_j)}{\omega_j},$$

(2.13)

where $k$ is the number of neighbours of target point $p_i$, and $\omega_j$ is a weight where $\omega_j \propto ||p_i - p_j||$. This means that FPFH ($p_i$) is a weighted sum of its own SPFH and the SPFH of its neighbours.

The full FPFH feature, despite having two steps of computations and neighbour calculations, reduces the complexity from $O(n \cdot k^2)$ of the PFH, to $O(n \cdot k)$. This is due to that the SPFH for each individual point in the whole set only needs to be computed once. If we want to compute the FPFH of a neighbouring point of $p_i$, say $p_1$, the SPFH of several neighbours has already been computed, and as long as it is cached, does not need recomputing. While this help greatly with how many computations are needed, having very large data sets or low memory capacity might require sorting according some 3D structure, like an octree\(^2\). This would let you keep only the features which are relevant, and compute the FPFH sweepingly across the octree.

The PFH features previously used a correlated binning of the probability of features appearing in a certain interval. According to analysis in [19], several of the correlated histogram bins remained empty, as no combination of certain intervals ever appeared. This introduces inefficiency as unnecessary zeroes are stored and used as descriptors. Instead, decorrelate the features and store the probability of each individual feature to appear in a certain interval in its own

\(^2\)An octree is a 3D sorting structure which at first covers the whole scene. When the scene node is considered 'full' by some measure, it splits into 8 (octo in latin) equally sized leaves, and redistributes its members amongst its leaves. The splitting is then performed recursively when a leaf is filled.
Figure 2.6: FPFH with the $k$-neighbourhood of a point $p_i$, where each point-pair combination that computes the features are represented by a line connection, and each neighbour subsequently computes their own SPFH from their own $k$-neighbourhood with radius $r$. Inspired by [19].

bin. Instead of $s^f$ bins, i.e. the number of subdivisions in each feature interval $s$ to the power of the number of calculated features $f$, we get $s \cdot f$ bins.

Reference [19] also presents evidence showing that $f_2 = \|p_i - p_s\|$ in (2.11) does little to help describe the local area, and seems to be redundant. This would let only 3 features from (2.11), $f_1, f_3, f_4$, to be computed. Due to the decorrelation of bins, the number of bins would in this case become $s \cdot 3$. This enables much more subdivisions of features without having an intractable amount of bins. PCL [11] in its implementation of FPFH uses 11 subdivisions for each feature, giving a total of 33 bins. This seems to be a common approach, and is what is used for this thesis. To summarize, the difference between PFH and FPFH are:

- FPFH does not fully interconnect all neighbours of $p_i$, thus missing some possible geometric describability, but increases performance.

- PFH uses a dense network of connections to precisely determine a surface within radius $r$ around $p_i$, while FPFH has sparser connections which reaches further, though a max range of $2r$.

- Due to the reweighting of neighbouring SPFH features, several connections of the FPFH will be taken into consideration twice.

- The second feature of (2.11), $f_2$, is regarded as redundant in FPFH, resulting in only 3 features for each point pair.
• PFH uses a correlated binning, where the number of bins for 2 subdivisions of 4 features would be $2^4 = 16$ bins. FPFH instead uses decorrelated bins, where each feature gets, for the proposed 11 subdivisions, 11 bins. This results in a total of $11 \cdot 3 = 33$ bins.

2.3 Registration Algorithms

A registration algorithm is a method which tries to align two or more data sets depicting (partially) the same data. In this thesis this refers to 3D point cloud registration, which searches for the rigid transformation $T$ that aligns two or more point clouds. Below is a description of the registration algorithms relevant to this thesis.

2.3.1 Fast Global Registration

This section is based heavily on [23], describing Fast Global Registration, and is meant to give a complete picture of the algorithm within, while still remaining relatively brief.

Fast Global Registration (FGR) is a semi-iterative registration algorithm which aims to provide a robust global solution while remaining computationally efficient, avoiding any update of correspondences in the inner update loop. The method contains two major steps; finding a set of correspondences between the two point clouds, and optimizing a transformation iteratively based on the rough correspondences. As the correspondences are not updated after the initial guess, it is critical for the optimization to be able to deal with very noisy correspondence sets.

**Correspondences**

To perform a correspondence search between two point clouds before doing any registration, features which are translation and rotation invariant are needed to be able to make any relevant comparison. This implementation uses FPFH features, explained in section 2.2.1, which uses an 33-dimensional feature vector that describes the local geometry around the target point. This feature was chosen as it can be computed in a fraction of a millisecond, and provides good accuracy for a broad range of scenes.

Let $F(P) = \{F(p) : p \in P\}$ be the set of features for point cloud $P$, where $F(p)$ is the FPFH for point $p$, and define $F(Q) = \{F(q) : q \in Q\}$ analogously for point cloud $Q$. For every $p \in P$, find the nearest neighbour of $F(p)$ among $F(Q)$ using a 33-dimensional euclidean distance measure; let $K_I$ be the set of the found correspondences. $K_I$ could be used directly as input to the transformation optimization, however, in practice the set has a very high fraction of outliers, e.g. not actual correspondences. Two tests are therefore used to improve the inlier ratio; these tests are:
• **Reciprocity test:** Ensure that every point pair \((p, q)\) from \(K_I\) fulfils that \(F(p)\) is the nearest neighbour of \(F(q)\) in \(F(P)\) and \(F(q)\) is the nearest neighbour of \(F(p)\) in \(F(Q)\). Let the correspondences that fulfil this requirement from \(K_I\) be called \(K_{II}\).

• **Tuple test:** Pick 3 random correspondence pairs from \(K_{II}\), \((p_1, q_1)\), \((p_2, q_2)\), \((p_3, q_3)\), and check if tuples \((p_1, p_2, p_3)\) and \((q_1, q_2, q_3)\) fulfil

\[
\forall i, j \in \{1, 2, 3\}, i \neq j, \quad \tau < \frac{||p_i - p_j||}{||q_i - q_j||} < \frac{1}{\tau},
\]

where \(\tau\) is a comparison constant (usually \(0.85 - 0.95\)). Intuitively, this makes sure the pairs are cross-compatible. The correspondences of \(K_{II}\) from all tuples who fulfil this requirement are gathered in a set \(K_{III}\).

The final correspondence set \(K_{III}\) will hereinafter be called \(K\), which is the set used in the registration process.

**Problem Formulation**

The task is to find the transformation \(T\) that best aligns \(Q\) to \(P\). This objective is analogously to minimize the distances between corresponding points, while negating spurious correspondences from \(K\). This can be formulated as

\[
E(T) = \sum_{(p, q) \in K} \rho(||p - Tq||),
\]

where \(\rho(\cdot)\) is a robust penalty function. The use of an appropriate robust penalty is crucial, as many of the terms in (2.14) contribute to spurious constraints. As the correspondences are never validated, pruned or recomputed, the estimator \(\rho\) will have to perform the validation and pruning automatically. To accomplish this a scaled Geman-McClure estimator,

\[
\rho(x) = \frac{\mu x^2}{\mu + x^2},
\]

is used. Figure 2.7 shows how this estimator differs for different \(\mu\), and how it affects one possible objective function \(E(T)\). This lets many correspondences participate for large \(\mu\), where a rough solution can be found. Sequentially decreasing \(\mu\) gives a tighter alignment while having avoided any local minima of the objective function.

Equation (2.14) is difficult to optimize directly. Instead the Black-Rangarajan duality between robust estimation and line processes [3] is used. Define \(L = \{l_{p,q}\}\) as a line process over the correspondences \(K\). You can now formulate a joint objective over \(T\) and \(L\):

\[
E(T, L) = \sum_{(p, q) \in K} l_{p,q}||p - Tq||^2 + \sum_{(p, q) \in K} \mu(\sqrt{l_{p,q}} - 1)^2.
\]
2.3 Registration Algorithms

For (2.16) to be minimized, the partial derivative with respect to each $l_{p,q}$ must be zero:

$$\frac{\partial E}{\partial l_{p,q}} = \|p - Tq\|^2 + \mu \frac{\sqrt{l_{p,q}} - 1}{\sqrt{l_{p,q}}} = 0,$$

which yields the solution

$$l_{p,q} = \left(\frac{\mu}{\mu + \|p - Tq\|^2}\right)^2.$$

(2.17)

Inserting this $l_{p,q}$ into equation (2.16) produces our original optimization problem (2.14). This shows that optimizing (2.16) provides a solution $T$ that also minimizes (2.14).

### Optimization

The main benefit of the optimization formulation in (2.16) is how you can alternate between optimizing $L$ and $T$, giving us very high computational efficiency. By fixing $L$ when optimizing $T$, and vice versa, and because both steps optimizes the same global objective, the algorithm is guaranteed to converge.

When $T$ is fixed, and $L$ is being updated, the problem has a closed-form solution; when equation (2.17) is satisfied. When $L$ is fixed (2.16) turns into a weighted sum of $L^2$ penalties on distances between point-to-point correspondences. While this problem has a closed-form solution, it does not extend to joint registration of more than two point clouds. Because of this another approach is used. Linearize $T$ as a 6-dimensional vector $\xi_T = (\omega, t) = (\alpha, \beta, \gamma, a, b, c)$, including 3 parameters ($\alpha, \beta, \gamma$) as a rotational component $\omega$, and 3 parameters ($a, b, c$) as a translational component $t$. $T$ can now be iteratively updated as a function of
\( \xi_T \) by

\[
T^q \approx \begin{pmatrix}
1 & -\gamma & \beta & a \\
\gamma & 1 & -\alpha & b \\
-\beta & \alpha & 1 & c \\
0 & 0 & 0 & 1
\end{pmatrix} T^{q-1}, \quad (2.18)
\]

where \( q \) is the current iteration, and thus \( T^{q-1} \) is the transformation from the previous iteration. To update the parameters of \( \xi_T \), notice that (2.16) becomes a least-squares objective on \( \xi_T \). Thus, using the Gauss-Newton method, \( \xi_T \) can be found by solving the linear equation system

\[
J^T_r r = -J^T_r r, \quad (2.19)
\]

where \( r \) is the residual vector of (2.16), and \( J_r \) is its Jacobian matrix with respect to \( \xi_T \).

These steps can be summarized in the following algorithm:

**Data:** A pair of point clouds \( P \) and \( Q \), with normals

**Result:** Rigid transformation matrix \( T \) that aligns \( Q \) to \( P \)

1. Compute FPFH features \( F(P) \) and \( F(Q) \) (with the point normals).
2. Build \( K_I \) through a nearest neighbour search of \( F(P) \) and \( F(Q) \).
3. Perform the reciprocity test on \( K_I \) to get \( K_{II} \).
4. Perform the tuple test on \( K_{II} \) to get \( K_{III} = K \).
5. \( q = 0, T^q = I, \mu = D^2 \).
6. repeat
7. \( q = q + 1 \).
8. reset \( J_r, r \).
9. for \((p, q) \in K\) do
10. Compute \( l_{p,q} \) as in equation (2.17).
11. Add to \( r \) and \( J_r \) of equation (2.19) using (2.16).
12. end
13. Solve equation (2.19) and update \( T^q \) with \( \xi_{T^q} \) and \( T^{q-1} \) in (2.18)
14. if \( q \mod 4 = 0 \) and \( \mu > \mu_{\text{min}} \) then
15. \( \mu = \mu/\alpha_f \)
16. end
17. until \( T^q \approx T^{q-1} \) or max iterations reached;
18. return \( T^q \)

**Algorithm 1:** FGR, a computationally efficient global registration algorithm without correspondence recomputations.
2.3.2 Joint Registration Multiple Point Clouds

This section is based heavily on [8], describing *Joint Registration Multiple Point Clouds*, and is meant to give a complete picture of the algorithm within, while still remaining relatively brief.

Joint Registration Multiple Point Clouds, or JR-MPC, is a probabilistic generative registration method aimed at removing the bias of assigning one point cloud in a set as the *model* in a probabilistic registration framework. There is no guarantee that a model-set is free of noise and outliers, thus running the risk of contaminating an estimation of registration parameters. Figure 2.8 demonstrates the first and last iteration of an attempted JR-MPC registration between two point clouds. Shown in yellow are the means of the gaussian distributions which JR-MPC uses to estimate the registration parameters. The method employs a joint *Gaussian Mixture Model* (GMM), which is updated based on all existing point clouds through a modified derived *Expectation Maximization* (EM) algorithm. This estimates both the GMM parameters and the transformations which map each individual set to the central GMM-model. The modified algorithm is called *Expectation Conditional Maximization* (ECM), which replaces the M-step of EM with a series of conditional maximization (CM) steps; performing an M-step for each model parameter.

**Toy Example**

JR-MPC works for an arbitrary number of point clouds. However, this example only includes two.

Figure 2.9 shows the initial state of a toy example, with 5 black dots in point cloud $j = 1$, and another point cloud $j = 2$ with 4 green ones. The yellow dots are three gaussian distributions $k = 1, 2, 3$, initialized at random locations in point cloud $j = 1$. The initial state of a real example can be seen in the left of figure 2.8.

Visible in figure 2.9 are three probabilities $\alpha_{ijk}$, which are the probabilities of point $i$ in cloud $j$ being a realisation of distribution $k$. This probability depends on the distance. Each point in both point clouds calculate their probabilities towards each distribution.

Through applying an EM algorithm on these data a rigid transformation can be estimated, with the intention of moving the green point cloud closer to the...
black. Afterwards, the distributions’ means and variances are updated using the calculated probabilities.

After a number of iterations the point clouds will have converged to the most probable alignment, which may or may not be the correct registration. The right part of figure 2.8 show a real example of a converged state.

![Figure 2.9: Initial state of a JR-MPC toy example.](image)

### Problem Formulation

Let $\Theta$ denote the model parameters

$$\Theta = (\{p_k, x_k, \sigma_k\}_{k=1}^K, \{\phi_j\}_{j=1}^M),$$

where $x_k$ and $\sigma_k$ is the mean and standard deviation of distribution $k$ with prior $p_k$, and the transformation $\phi_j(v_{ij}) = R_j v_{ij} + t_j$, for each point $v_{ij}$ in point-set $j$. Also, define hidden variables

$$Z = \{z_{ij} | i = 1 \ldots N_j, j = 1 \ldots M\},$$

such that $z_{ij} = k$ assigns the transformed observation $\phi(v_{ij})$ to the $k$-th component of the GMM. The expected complete-data log-likelihood we seek to maximize in order to estimate the parameters $\Theta$ can now be written as

$$\mathcal{E}(\Theta|V, Z) = \mathbb{E}_Z[\log P(V, Z; \Theta)|V] = \sum_Z P(Z|V, \Theta) \log \left(P(V, Z; \Theta)\right). \quad (2.20)$$

If we assume the data $V = \{v_{ij} | i = 1 \ldots N_j, j = 1 \ldots M\}$ are independent and identically distributed, equation (2.20) can be written as

$$\mathcal{E}(\Theta|V, Z) = \sum_{i,j,k} \alpha_{ijk} \left( \log p_k + \log P(\phi_j(v_{ij})|z_{ij} = k; \Theta) \right). \quad (2.21)$$

where $\alpha_{ijk} = P(z_{ij} = k | v_{ij}; \Theta)$; the probability of $v_{ij}$ being a true observation of distribution $k$ with GMM model parameters $\Theta$. By replacing the standard expressions of the likelihoods, and ignoring the constant terms, equation (2.21)
can be formulated as an objective function;

\[ f(\Theta) = -\frac{1}{2} \sum_{i,j,k} \alpha_{ijk}(\frac{\|\phi_j(v_{ij}) - x_k\|^2}{\sigma_k} + 6 \log \sigma_k - 2 \log p_k) + \log p_{K+1} \sum_{i,j} \alpha_{ij(K+1)}, \]  

(2.22)

where \( p_{K+1} = \gamma \sum_{k=1}^{K} p_k \), with \( \gamma \) as the ratio between outliers and inliers, and \( \alpha_{ij(K+1)} = 1 - \sum_{k=1}^{K} \alpha_{ijk} \), the probability of \( v_{ij} \) being an outlier. The following constrained optimization problem has thus been defined:

\[
\begin{cases}
\max_{\Theta} f(\Theta), \\
\text{s.t.} \quad R_j^\top R_j = I, |R_j| = 1, \forall j = 1 \ldots M.
\end{cases}
\]

(2.23)

**Applying ECM**

Performing direct optimization of (2.23) via a closed-form solution is difficult due to the induced non-linearities. Instead the mentioned ECM algorithm is applied to iteratively update all model parameters in \( \Theta \) individually, until they converge. The algorithm looks like this:

**Data:** M point clouds of the same scene  
**Result:** Transformations for each point cloud registered to the joint GMM

1. \( q = 0 \).
2. Initialize \( \Theta^q \).
3. repeat
   4. \( q = q + 1 \).
   5. E-step: Update \( \alpha_{ijk}^q \) with previous state \( \Theta^{q-1} \).
   6. CM-step-A: Update \( R_j^q \) and \( t_j^q \) with \( \alpha_{ijk}^q, x_k^{q-1}, \sigma_k^{q-1} \).
   7. CM-step-B: Update \( x_k^q \) with \( \alpha_{ijk}^q, R_j^q, t_j^q \).
   8. CM-step-C: Update \( \sigma_k^q \) with \( \alpha_{ijk}^q, R_j^q, t_j^q, x_k^q \).
   9. CM-step-D: Update \( p_k^q \) with \( \alpha_{ijk}^q \).
4. until \( \Theta^q \approx \Theta^{q-1} \);
5. return \( \Theta^q \).

**Algorithm 2:** JR-MPC, an iterative ECM-algorithm applied to a joint GMM.

where each step in the ECM-algorithm is explained below.
E-step

The E-step of the ECM algorithm updates the probabilities for each point, where \( \alpha_{ijk}^q \) is computed using the previous \( \Theta^{q-1} \) as seen in algorithm 2, as

\[
\alpha_{ijk} = \frac{p_k \sigma_k^{-3} e^{-\frac{1}{2\sigma_k^2} \|\phi_j(v_{ij}) - x_k\|^2}}{\sum_{s=1}^{K} (p_s \sigma_s^{-3} e^{-\frac{1}{2\sigma_s^2} \|\phi_j(v_{ij}) - x_s\|^2})} + \beta, \quad k = 1, \ldots, K, \tag{2.24}
\]

where the constant \( \beta = \gamma/h(\gamma + 1) \) accounts for the outlier term. Here \( h \) is the volume of the convex hull that can contain all (centered, not registrated) point clouds.

CM-step-A

The first step of the CM sequence is to update the rotation and translation estimations based on the new probabilities \( \alpha_{ijk}^q \), and the distribution parameters of the previous iteration. By setting the GMM parameters to their values given in equation (2.24), the optimal parameters for (2.23) can be shown to coincide with the reformulated optimization problem

\[
\begin{aligned}
\min_{R_j, t_j} & \quad \| (R_j W_j + t_j e^T - X) \Lambda_j \|^2_F \\
\text{s.t.} & \quad R_j^T R_j = I, \quad |R_j| = 1
\end{aligned}
\tag{2.25}
\]

where \( \Lambda_j \) is a \( K \times K \) diagonal matrix with elements \( \lambda_{jk} = \frac{1}{\sigma_k} \sqrt{\sum_{i=1}^{N_j} \alpha_{ijk}} \), matrix \( X = [x_1, \ldots, x_K] \) is the means of all distributions, \( e \) is a vector of ones and \( W_j = [w_{j1}, \ldots, w_{jK}] \) is a matrix of virtual points, where each point \( w_{jk} \) is given by

\[
\sum_{i=1}^{N_j} \frac{\alpha_{ijk} v_{ij}}{\sum_{i=1}^{N_j} \alpha_{ijk}}.
\]

This is an average of all points in set \( j \), weighed with their probabilities. The optimization problem in (2.25) is a weighed version of the problem solved in [22], and it still has an analytic solution. The expression for the optimal rotation is given by

\[
R_j^T = U_j^L S_j U_j^R, \tag{2.26}
\]

where \( U_j^L \) and \( U_j^R \) are the left and right matrices of a Singular Value Decomposition (SVD) of the matrix \( W_j \Lambda_j P_j \Lambda_j X^T \), with \( P_j = I - \frac{\Lambda_j e (\Lambda_j e)^T}{tr(\Lambda_j^2)} \) as a projection matrix, and \( S_j \) as an identity matrix with the bottom right element set to \( |U_j^L||U_j^R| \).

Using the optimal rotation, the translation is subsequently calculated as

\[
t_j = \frac{(X - R_j W_j) \Lambda_j^2 e}{tr(\Lambda_j^2)}. \tag{2.27}
\]
2.3 Registration Algorithms

CM-step-B and CM-step-C

The B and C steps update the means and variances of the GMM distributions using \( R^q_j, t^q_j \) and \( \alpha^q_{ijk} \). By setting \( \partial f / \partial x_k = 0 \) in (2.22) the expression for the optimal means can be acquired as

\[
x_k = \frac{\sum_{j=1}^{M} \sum_{i=1}^{N_j} \alpha_{ijk} (R_j v_{ij} + t_j)}{\sum_{j=1}^{M} \sum_{i=1}^{N_j} \alpha_{ijk}}.
\]

(2.28)

Inserting the means from (2.28) and instead setting \( \partial f / \partial \sigma_k = 0 \), the expression for the variance becomes

\[
\sigma^2_k = \frac{\sum_{j=1}^{M} \sum_{i=1}^{N_j} \alpha_{ijk} ||R_j v_{ij} + t_j - x_k||^2}{3 \sum_{j=1}^{M} \sum_{i=1}^{N_j} \alpha_{ijk}} + \epsilon^2,
\]

(2.29)

where \( \epsilon^2 \) is a very small value to avoid singularities.

CM-step-D

The last sub-step in the sequence updates the priors \( p_k \) with the probabilities \( \alpha^q_{ijk} \). By formulating a dual objective function of (2.22),

\[
f_L(p_1, \ldots, p_K, \mu) = \sum_{k=1}^{K} \left( \log p_k \sum_{i,j} \alpha_{ijk} \right) + \mu \sum_{k=1}^{K} \left( p_k - \frac{1}{1 + \gamma} \right)
\]

(2.30)

only with respect to terms dependent on \( p_k \), and setting \( \partial f_L / \partial p_k = 0 \), the optimal priors becomes

\[
p_k = \frac{\sum_{i,j} \alpha_{ijk}}{\mu}, \quad k = 1, \ldots, K \quad \text{and} \quad p_{K+1} = 1 - \sum_{k=1}^{K} p_k,
\]

(2.31)

where \( \mu = (\gamma + 1)(N - \sum_{i,j} \alpha_{ijk(K+1)}) \), and \( N = \sum_j N_j \) is the cardinality of \( V \).

2.3.3 Prerejective RANSAC

This section explains the modification of the Random Sample Consensus (RANSAC) algorithm in [4], Prerejective RANSAC, which applies an earlier pose-rejection condition in the RANSAC pipeline in an attempt to speed up the process.
Consider two point clouds \( P \) and \( Q \). We seek a rigid transform \( T \) which aligns \( P \) and \( Q \). Let \( F(P) = \{ F(p) : p \in P \} \) be the set of features for point cloud \( P \), where \( F(p) \) is some extracted feature for point \( p \). Analogously, define \( F(Q) = \{ F(q) : q \in Q \} \) for point cloud \( Q \). To align these two, a generic optimization problem can be formulated as:

\[
\varepsilon(T) = \sum_{p \in P, q \in Q} \| p - Tq \|^2. \tag{2.32}
\]

Here \( T \) is the transformation to estimate in order to minimize the sum of squared euclidean distances between corresponding points in the two sets \( P \) and \( Q \). This can be estimated using a RANSAC pipeline. The idea is to sample \( n \geq 3 \) completely random points from \( P \) and find correspondences of these in \( Q \). These correspondences can either be completely random, or found through a nearest neighbour search among the invariant features in \( F(P) \) and \( F(Q) \). In this version, the features are utilized for finding the correspondences. Figure 2.10 shows an example of \( P \) and \( Q \), with three point correspondences illustrated as a line connection. Consequently, a transform \( T \) can be estimated using the correspondences which best align the two point clouds. If the found transformation is better than any seen so far, i.e. if the ratio of inliers is high enough and the sum of squared distances between inliers is the lowest seen so far, make that the new best guess. Repeat the process until a maximum number of iterations is reached. This can be seen as a guessing algorithm, where you ‘brute force’ an optimal transformation through trial and error. A pseudo-code algorithm can be seen in algorithm 3.

**Figure 2.10:** A toy example of a RANSAC iteration. Two point clouds \( P \) and \( Q \) are shown with three drawn correspondences, illustrated as line connections. A transformation \( T \) can consequently be estimated, and the ratio of inliers versus outliers computed.
2.3 Registration Algorithms

Data: A pair of point clouds \( P \) and \( Q \)

Result: Transformation \( T \) that aligns \( Q \) to \( P \)

1. Compute features \( F(P) \) and \( F(Q) \).
2. \( T^{\text{best}} = I \), \( \varepsilon^{\text{best}} = \infty \).
3. repeat
   4. Pick \( n \geq 3 \) random points from \( P \).
   5. Find correspondences of these points in \( Q \) by nearest neighbour search in \( F(P) \) and \( F(Q) \).
   6. Estimate \( T \) from the correspondences (with suitable method, see [16]).
   7. Apply \( T \) to \( Q \).
   8. Search for inliers between \( P \) and the transformed \( Q \) by spatial nearest neighbour search, with a max distance to be considered an inlier.

   if ratio of inliers is too low then
      10. Skip to next iteration.
   end

   Reestimate \( T \) with the found inlier correspondences.
   12. Compute the error \( \varepsilon \) from (2.32) using the estimated \( T \) and inliers.

   if \( \varepsilon < \varepsilon^{\text{best}} \) then
      15. \( T^{\text{best}} = T \).
      16. \( \varepsilon^{\text{best}} = \varepsilon \).
   end

18. until max iterations reached;
19. return \( T^{\text{best}} \)

Algorithm 3: RANSAC, an iterative ‘guessing’ algorithm for pose estimation.

An issue with the simple algorithm 3 is a slow execution time, which random search algorithms often struggle with. Reference [4] tries to solve this by offering an earlier pose-rejection step, where you do not need to transform the entire \( Q \) with \( T \) and compute inliers to be able to reject the pose.

Denote the \( n \) sampled points from \( P \) and their correspondences in \( Q \), found through \( F(P) \) and \( F(Q) \), as \( p_i, q_i, i \in \{1, \ldots, n\} \). We can utilize that distances are preserved under affine transformations (which is a stronger requirement than our rigid transformer). Specifically, make a simple check to make sure the ratio of the edge length of the virtual polygons formed by the points \( p_i, q_i, i \in \{1, \ldots, n\} \) is relatively similar. Denote the edge lengths of the object polygon as \( d_{p,i} = \| p_{i+1 \mod n} - p_i \| \), and analogously for \( d_{q,i} \). We can then compute a relative dissimilarity vector of ratios between these \( n \) polygon edge lengths as

\[
\delta = \left[ \frac{|d_{p,1} - d_{q,1}|}{\max(d_{p,1}, d_{q,1})}, \ldots, \frac{|d_{p,n} - d_{q,n}|}{\max(d_{p,n}, d_{q,n})} \right]
\]  

(2.33)

A perfect match of two polygons would give a \( \|\delta\| \) of 0. In practice, the largest deviation, while still remaining an actual match, can be expected to be some constant \( \|\delta\|_{\infty} \leq t_{\text{poly}} \). Inserting this check after finding the initial correspondences (row 5 of algorithm 3) should speed up the process significantly, as we can reject
the pose without computing an inlier ratio, which requires a neighbour search for the whole data set. Reference [4] suggests a $t_{poly}$ of around 0.25, allowing an edge length dissimilarity of 25%. The modified algorithm can be written as:

**Data:** A pair of point clouds $P$ and $Q$

**Result:** Transformation $T$ that aligns $Q$ to $P$

1. Compute features $F(P)$ and $F(Q)$.
2. $T_{best} = I$, $\varepsilon_{best} = \infty$.
3. repeat
   4. Pick $n \geq 3$ random points $p_i, i \in \{1 \ldots n\}$ in $P$.
   5. Find correspondences $q_i, i \in \{1 \ldots n\}$ of these points in $Q$ by nearest neighbour search in $F(P)$ and $F(Q)$.
   6. Calculate $\delta$ according to equation (2.33).
   7. if $||\delta||_\infty > t_{poly}$ then
      8. Skip to next iteration.
   end
   9. Estimate $T$ from the correspondences (with suitable method, see [16]).
10. Apply $T$ to $Q$.
11. Search for inliers between $P$ and the transformed $Q$ by spatial nearest neighbour search, with a max distance to be considered an inlier.
12. if ratio of inliers is too low then
    13. Skip to next iteration.
   end
14. Reestimate $T$ with the found inlier correspondences.
15. Compute the error $\varepsilon$ from (2.32) using the estimated $T$ and inliers.
16. if $\varepsilon < \varepsilon_{best}$ then
    17. $T_{best} = T$.
    18. $\varepsilon_{best} = \varepsilon$.
   end
20. until max iterations reached;
21. return $T_{best}$

**Algorithm 4:** Prerejective RANSAC, an iterative ‘guessing’ algorithm for pose estimation, with an early pose-rejection step to speed up the search.
The problem of aligning the data from theToF camera with the HoloLens mesh is in this thesis split into two parts. First, to set up a test platform in which to try several different methods and evaluate them on the data from the two different sensors. While an algorithm might work very well on two data sets generated from the same sensor, it may not for this setup. Especially since the depth camera and HoloLens uses entirely different depth imaging methods, as sections 1.5.1 and 1.5.2 explains. The second part is to find the best method, and implement this to run on the HoloLens headset. This is intended to run in real time to align the two data sets. Because of this, both accuracy, consistency and computation time is taken into account when evaluating the methods on the test platform.

When performing point cloud registration there are always two major parts to solve:

- how to describe the data points in a relevant way. Section 2.2 explains descriptors, which are, most commonly, a vector of extracted information about any given data point. A specific point coupled with its descriptor is sometimes also called a feature. A descriptor can include more informative data than only the 3D coordinates; a coordinate only lets us find a spatial distance between points. There are others which may be more descriptive for the problem we want to solve. For the problem in this thesis for example, it is very beneficial to be able to compare if neighbourhoods around points are similar, and thus compare if two descriptors represent the same area in the actual scene.

- how to align the point clouds, using improved descriptors of the data points. Data alignment, also called registration, explained in more detail in section 2.3, seeks the rigid transformation which best align two or more data sets. The method to estimate this transformation varies greatly, but two general approaches exist. First, to construct some probabilistic density functions
which are updated iteratively, either through maximizing a similarity mea-
sure, or applying the Expectation Maximization (EM) algorithm to update
the registration parameters. Second, an error measure from tentative point
correspondences, and use this error to update the transformation iteratively.
A point correspondence is a guess of which two points in two different data
sets represents the same point in a scene. As these suggests, most meth-
ods uses iterative approaches, as many non-linearities in any error function
prevents a closed form solution.

3.1 Feature Selection

In this thesis the goal is to evaluate different registration algorithms, not different
descriptors. In the initial literature search Fast Point Feature Histograms (FPFH)
appeared to be a popular choice, and is the recommended feature to use together
with Fast Global Registration [23], which is evaluated later in the thesis. There-
fore the FPFH will be considered the baseline choice for all algorithms where
additional features are beneficial.

FPFH [19], explained in 2.2.1, are transform invariant features made to de-
scribe the structure of the local area around any given point. This means that a,
in this case 33-dimensional, euclidean distance measure can be seen as a check to
see how similar two point-neighbourhoods are. FPFH captures the geometrical
neighbourhood of the point through extraction of different angles between point
pairs near the target point. The descriptor is then formed by binning the proba-
bility of each angle appearing within a sub-interval of its range for all point pairs
in a histogram, covering all different angles and their sub-intervals.

3.2 Registration Algorithm Selection

The first imposed limitation in selecting suitable registration algorithms are whether
or not they can find global solutions. As mentioned in section 1.6 the common
method Iterative Closest Point (ICP) is very good for small errors, but fails when
the initial error increases. ICP is thus reserved for refinement purposes, where
another method supplies the first alignment.

The three algorithms that were chosen to be analyzed were so due to them
being promising candidates for both speed and accuracy. They also represent
the overall registration scene well, with all methods employing vastly different
approaches. The algorithms to be tested can be seen below.

• Fast Global Registration: FGR [23], explained in section 2.3.1, is a global
registration algorithm which tries to avoid updating any correspondences it
finds in an initial search, in the inner update loop. This requires some other
way of disregarding spurious correspondences, which was solved in [23] us-
ing a Geman-McClure estimator enclosing the terms of the loss function.
The lack of correspondence updates in an inner loop makes this method
very fast. The main issue with the algorithm is to find enough true correspondences in the initialization step of building the correspondence set. This is where FPFH comes in. As the features are transform invariant, a euclidean distance measure can represent the neighbourhood similarity of two points.

- Joint Registration Multiple Point Clouds: JR-MPC [8], explained in section 2.3.2, is a probabilistic registration method which forms a joint Gaussian Mixture Model (GMM) for all point clouds which are to be registered. This means a certain number of gaussian distributions with unique mean and variance are placed in the coordinate system of the point clouds. All points are assigned a probability to be a realisation of each individual distribution. This probability can then be used to update the mean and variances, which in turn can be used to estimate a transformation for each point cloud which increases the probabilities.

The iterative update approach uses a modified EM algorithm, ECM (where C stands for conditional), to let the system converge into the most likely placement. The main weakness of this method is how it only takes point density into consideration. Normal directions and local geometry are disregarded, which otherwise might give more information about the data. It is also not a completely global registration algorithm. Large prominent structures in the data may make the algorithm converge to incorrect solutions. An example is if a wall makes up the backdrop of a 2.5D image; with an incorrect initial rotation it is likely the algorithm converges to a 180° rotation compared to the true alignment. Despite this, given the assumptions we can make about the data in this setup, it is sufficiently 'global'.

- Prerejective RANSAC: Explained in section 2.3.3 is the extension of the widely used RANSAC algorithm, Prejective RANSAC [4]. The method is intended to speed up the random search of RANSAC by offering a earlier pose rejection step, and thus not require the whole point cloud to be transformed and checked for each iteration. RANSAC picks a small set of random points in one point cloud, which it then finds correspondences for, either randomly or through some neighbour search. The few correspondences are used to find the transformation which aligns them. The ratio of inliers, e.g. how many points have a neighbour within a radius $r$ in the other point cloud, is then extracted. If the ratio is large enough, and the sum of squared distances between the neighbours is smaller than any other previous sum, pick that transformation as the best. This is iterated for a predetermined number of iterations.

As with any random search algorithm it suffers from consistency errors, especially if we have a large search space. This can be remedied by having more iterations, which of course increases computation times.

As mentioned, all three of the algorithm can be followed by ICP to fix any small errors which might remain, without severely slowing down the registration.
3.3 Test Platform

The test platform is implemented as an offline system, built around the open source library PCL [11]. PCL offers a wide variety of functionality, where several previously mentioned methods, namely the FPFH features and the ICP registration algorithm, are already implemented. This makes testing and evaluation significantly faster. PCL operates in C++, which suits the need for low computation times. To test all the methods effectively and equally, it is important to make a pipeline which allows switching of the registration method without much effort.

Because the testing is offline, a data set needs to be extracted and saved. Also, as the data is not perfect, and artifacts may appear due to lighting conditions or weird angles, pre-processing of data is required.

3.3.1 Data Extraction

This section explains how to extract the needed information from the sensors. Figure 3.1 shows a partial flowchart, including the step of extracting a mesh with normals from the HoloLens, as well as a point cloud from the ToF camera.

![Figure 3.1: Partial flow chart of the registration pipeline. Here the two different data extraction steps are shown.](image)

The HoloLens

The data needed from the HoloLens is its mesh of triangles it creates in real time. Fortunately, Microsoft offers an online interface when connected to the same network as the HoloLens. With adequate login information there is access to many features to see the data the HoloLens generates. There is screen capture functionality for visualization and presentation applications, where we can see what the person wearing the HoloLens sees. There is also a visualization of the active mesh, as well as a download option in the common .obj file format.

The Visionary-T ToF camera

The typical setup in the experiments of this thesis is with 2-4 meters to the target at which the ToF camera is pointed. The Visionary-T camera can be accessed through a C++ program which connects to the camera through ethernet with a known static IP. The sent data arrives as a byte buffer, with a header of meta data. The data arrives as distance values, confidence values and intensity values in the
176x144, grid which is the camera resolution. To get a point cloud with \((x, y, z)\) coordinates we need to undistort, shift and scale the distance values using the camera parameters. The transform from \(z\)-values to \((x, y, z)\) is provided by the camera. The point cloud can then be saved in a .pcd format, which saves position and color data. As only the intensity is available, a dynamic jet-colormap is used for color, where the intensity dims the color of an ordinary jet-colormap depending on how far from the max intensity each pixel is. A normal jet-colormap is a remapping from distance values, near to far, to red, green and blue color, in that order. The color is used for visualization purposes only, such as in figure 3.4.

### 3.3.2 Data Pre-processing

To prepare the data for registration several pre-processing steps are taken, in order to improve the robustness of the data. Figure 3.1 illustrates the steps; first to refine the data, and second, to downsample for an even distribution of points as well as reducing the complexity.

**Figure 3.2:** Partial flow chart of the registration pipeline. This shows steps to refine and improve the data, which is later downsampled to manageable quantities.

**Refinement of existing data**

The ToF camera depth image suffers from noise, and artefacts may appear due to lighting conditions. As all depth images are projections from different surfaces, we can estimate the surface a neighbourhood of points originated from. Using this, we can move each data point closer to the estimated original surface, thus removing some noise. This is called *Mean Least Squares* resampling, shown as *MLS resampling* in figure 3.2. Similarly to the normal estimation in section 3.3.3, MLS resampling estimates a plane for a neighbourhood of points through a covariance matrix, which can be used as an estimation of the original surface.

Points from the HoloLens mesh come with varying spatial density. This is reasonable from a data standpoint, as a flat surface can be represented by very few polygons. For this application however, we need a roughly uniform point density. As the points are actually vertices of a mesh, additional points can be picked
from inside the polygons to fill in gaps where there is low point density. In figure 3.2 this is the **Mesh upsampling** block. Having an even point distribution helps with having consistent normal and feature estimation, as each neighbourhood in a radius $r$ around any point will have roughly the same number of neighbours as any other.

**Downsampling to manageable amounts**

The ToF camera produces $176 \times 144 = 25344$ points, and the HoloLens data with mesh upsampling ends up with roughly twice that. Almost 80 000 points are generally too many to handle, both in terms of computation time as well as the extremely large search space it creates. Because of this the data is downsampled using a **Voxel grid downsampling**, included in figure 3.2. This partitions the entire scene into small square blocks with a common side length $d$. The blocks are aligned with the $x, y, z$ axes. To downsample using this, each point will be assigned to the block that covers its 3D position in the scene. Many points might end up in the same box. When all points are assigned, remove any points where there are two or more in the same box, keeping only the one closest to the centroid of the box. Another alternative is to make each block result in the mean position of all its inhabitants. Both methods were tested, and provided very similar registration results. Thus, the first method was prioritized due to computational efficiency. The result is in both cases an evenly distributed point cloud of points with an approximate distance $d$ between them. This step renders the mesh invalid, but the HoloLens points and their normals can still be used as an ordinary point cloud.

### 3.3.3 Normal and Feature estimation

To be able to calculate the FPFH features for both the data sets, normals are needed. As the HoloLens data originated from a mesh, the normals are already available there, where as for the ToF data we need to estimate them. After normal estimation the features can be calculated using the point clouds and their normals, as shown in the partial flow chart in 3.3.

**Normal Estimation**

The recommended radius for normal estimation of FGR [7] is twice that of the downsampling range $d$. This means that each point will use a set of neighbours within the radius, in this case $2d$, to estimate its normal. If we assume the entire ToF camera data was part of a single planar surface, a radius of $2d$ around a point would include roughly 12 neighbours ($2^2 \pi \approx 12$). This seems like a reasonable amount to approximate a surface with, and subsequently its normal.

With a set of neighbours to a target point, the normal of the target can be estimated by first computing a $3 \times 3$ covariance matrix of the set of points, defined
3.3 Test Platform

**Figure 3.3:** Partial flow chart showing the steps of normal estimation for the ToF data, and FPFH estimation for both the data sets.

![Flow chart](image)

\[
\Sigma_{3x3} = \begin{bmatrix}
E[(X - \mu_x)(X - \mu_x)] & E[(X - \mu_x)(Y - \mu_y)] & E[(X - \mu_x)(Z - \mu_z)] \\
E[(Y - \mu_y)(X - \mu_x)] & E[(Y - \mu_y)(Y - \mu_y)] & E[(Y - \mu_y)(Z - \mu_z)] \\
E[(Z - \mu_z)(X - \mu_x)] & E[(Z - \mu_z)(Y - \mu_y)] & E[(Z - \mu_z)(Z - \mu_z)]
\end{bmatrix}, \quad (3.1)
\]

where \( E \) denotes the expected value, \( X, Y, Z \) is the distribution of \( x, y, z \) coordinates of the point set and \( \mu_x = E(X), \mu_y = E(Y), \mu_z = E(Z) \) their means. The mean \( E(A) \) and covariance \( E[(A - \mu_a)(B - \mu_b)] \) is here approximated as:

\[
E(A) \approx \frac{1}{N} \sum_{i=1}^{N} a_i, \quad E[(A - E(A))(B - E(B))] \approx \frac{1}{N} \sum_{i=1}^{N} (a_i - E(A))(b_i - E(B)),
\]

where \( N \) is the number of points in set \( A \) as well as in set \( B \), and \( a_i, b_i \) points from the \( A \) and \( B \) sets.

The two eigenvectors of \( \Sigma_{3x3} \), which correspond to its two largest eigenvalues will span a plane in the directions in which the point set varies the most. This means the plane is a planar approximation of the surface that the points originated from. The normal of the plane is thus approximately the normal of the point, but the sign of a plane normal is ambiguous. To find the sign, simply make sure the normal point towards the camera position.

Figure 3.4 shows a point cloud with approximated normals as ‘spikes’ from the surface, in black color.

**FPFH estimation**

FPFH estimation, explained in section 2.2.1, uses a point cloud and its normals to estimate 33-dimensional descriptors for each point. These descriptors are important as they allow us to see whether or not two points are likely to correspond to the same location in a scene. Any correspondence search before alignment requires some form of transform invariant features.

Similarly to the radius of the normal estimation, the radius for the FPFH estimations is also mentioned by [7] to be five times the downsampling distance \( d \).
Initial tests with varying radii showed the same, and the 1:2:5 ratio of downsampling, normal estimation and FPFH estimation is hereinafter what is used.

### 3.3.4 Registration

With the FPFH features available, the procedure of estimating a transformation can be started, see figure 3.5. Each of the different algorithms do not use all of the information. None of them use the estimated normals other than indirectly through the FPFH features. Each of the registration algorithms produces a 4x4 transformation matrix, which transforms each point in the ToF data to the coordinate system of the mesh. In order to quantitatively evaluate the transformations, some sort of ground truth is needed.

![Figure 3.5: Partial flow chart regarding registration.](image-url)

For each registration algorithm several grid parameter searches were done to
find the parameters which gave the best and most robust results. As the number of adjustable parameters vary for the algorithms, the parameter space has to be reduced to the few most important ones. Also, an algorithm with a high dimensional parameter space is infeasible to do a complete grid search for. For all the algorithms the downsampling radius $d$ (which affects the normal estimation radius and features estimation radius using the 1:2:5 ratio), is the most important parameter to tune, either to improve accuracy or reduce execution time.

### Evaluating transformations

To be able to evaluate a transformation $T$ there needs to be a ground truth transformation to compare with.

The ground truth was generated by manually merging a visualization of both the mesh and the ToF point cloud. Small translations and rotations were performed to slowly shift the cloud into place. Each key press generated a transformation which, when matrix-multiplied from the left, was joined with a cumulative transformation $T_{gt}$ that served as ground truth when the data eventually aligned. The translational resolution was 2 cm in the $x$, $y$, and $z$ directions, and the rotational resolution was $1^\circ$ in the yaw, pitch and roll directions. The ToF data and the mesh had a known scale correspondence, where their coordinates were expressed in (real world) meters.

The metric to compare $T$ and $T_{gt}$ was split into two. First, there is the sum of absolute translation errors. Any $4 \times 4$ transformation matrix $T$ can be represented by a $3 \times 3$ rotation $R_T$ and $3 \times 1$ translation $t_T$ matrix, $T = \begin{bmatrix} R_T & t_T \\ 0 & 1 \end{bmatrix}$. The absolute translation error between $T$ and $T_{gt}$ is thus defined as

$$\varepsilon_{\text{trans}} = \|t_T - t_{T_{gt}}\|_1,$$

where $\| \cdot \|_1$ is the $L^1$ norm. Arguably, the $L^2$ norm might have better described the translation errors, but this was not investigated.

Secondly is the rotation error. Using Rodrigues’ formula we can find the angle of an axis angle representation of the rotation difference between the two rotation matrices $R_T$ and $R_{T_{gt}}$, $R_T^T R_{T_{gt}}$. This is calculated as

$$\varepsilon_{\text{rot}} = |\arccos\left(\frac{\text{tr}(R_T^T R_{T_{gt}}) - 1}{2}\right)|,$$

where $\text{tr}(\cdot)$ denotes the trace of a matrix, which is the sum of the diagonal. $\varepsilon_{\text{rot}}$ is expressed in radians.

### FGR

The FGR algorithm, explained fully in section 2.3.1, uses several parameters in order to be able to adjust its performance depending on the incoming data apart from the downsampling distance $d$:
• A division factor $d_{df}$, which is how fast the Geman McClure estimator narrows, seen in row 15 of FGR in algorithm 1. A too large division factor risks getting stuck in local minima, while a too small factor might give a sub-par alignment. This is kept constant at 1.2.

• The tuple scale constant and tuple count. These refer to the initial correspondence search in the tuple test on row 4 of algorithm 1. The tuple scale constant $\tau$ is how similar the randomly formed tuples needs to be to be considered a match, and the tuple count is the maximum number of tuples formed. $\tau$ is kept constant at 0.85, and a constant tuple count of 1500.

• The smallest allowed $\mu$, $\mu_{\text{min}}$, which should correspond to the maximum distance between any true correspondences (after alignment). This is seen at row 14 of algorithm 1, kept constant at 0.1 m.

Figure 3.6: An example registration using FGR. The black dots on the mesh represents the actual points used for registration, while the mesh is only there for visualization purposes. The colored data is the ToF depth image registered onto the mesh.

None of the listed parameters were found to have a huge effect, or rather have one optimal value regardless of other settings. An example of a registration done with FGR can be seen in figure 3.6. The FGR algorithm was iterated for 64 iterations to converge.

JR-MPC

The probabilistic algorithm JR-MPC, explained fully in section 2.3.2, has only a few adjustable parameters apart for the inherent downsampling distance $d$:

• How many distributions to generate. This parameter improves performance, at the cost of execution time. The idea is to find a balance between reasonable computation time and accuracy. This value was varied between 8 – 210 in the parameter searches presented in chapter 4.
• $\gamma$, appearing in the probability update of equation (2.24) through $\beta$. $\gamma$ is here set to depend on the number of distributions, where $\gamma = \frac{1}{\text{number of distributions}}$.

• $\epsilon$, found in the variance update in equation (2.29). This is normally a small number to avoid any singularities when using the variances. $\epsilon$ here is a constant set to $10^{-5}$.

Out of these there was only the number of distributions that provide a notable difference (and $\gamma$, as they are correlated). Thus, only $d$ and the number of distributions are varied in the parameter searches.

An important observation is how to handle the first few iterations; if we allow the distributions to update their means in the first few iterations they will always converge towards the middle before wandering outwards again. This is due to the large initial variance of the distributions, which makes the probabilities of every point toward every distribution very similar. Therefore, the distributions were limited to only update their variance, not the means, for the first few iterations, which stabilizes the convergence.

In figure 3.7 an example of an alignment using JR-MPC is shown. The yellow dots are the means of each distribution at the last iteration. The algorithm was allowed to run for 100 iterations for this example.

![Figure 3.7: A registration using JR-MPC, where the final distribution means are shown in yellow. The black dots on the mesh represents the actual points used for registration, while the mesh is only there for visualization purposes. The colored data is the ToF depth image registrated onto the mesh.](image)

Prerejective RANSAC

The random search algorithm Prerejective RANSAC, explained fully in section 2.3.3, shares its parameters with the ordinary RANSAC algorithm, as well as adding one extra. Apart from the downsampling distance $d$, Prerejective RANSAC has:
• The maximum number of iterations (RANSAC parameter). As the algorithm is a random search, this parameter should improve performance regardless of circumstances. Choosing an appropriate value as a trade off between computation time and accuracy is important, but when a middle ground is found there is little need to adjust it because of changes in other parameters. This parameter is found at row 18 of algorithm 4 in section 2.3.3. This is kept at 10 000 to keep a low execution time while remaining relatively robust.

• The \( n \) random points (RANSAC parameter) picked at the start of each iteration, found on row 4 of algorithm 4. It is hard to pinpoint how this affects the results intuitively, but a larger \( n \) might give more consistent guesses of transformations. Values at or slightly above 3 is recommended, where 3 is used in this work.

• The inlier ratio to not reject the pose (RANSAC parameter). This parameter, used at row 9 of algorithm 4, lets us adjust the number of points that has to have a spatial nearest neighbour after the first transformation. This will correlate with how much overlap the data sets has, and a value somewhat lower than the actual overlap percentage is recommended. A value of 0.4 is used here.

• The max correspondence distance (RANSAC parameter). Referenced at row 8 of algorithm 4, this is connected heavily with the inlier ratio, and determines how far away the nearest neighbour of a point can be while remaining an inlier. A large value here lets us increase the required inlier ratio, as we might get more false positives. This value should roughly correspond to how much noise is present. Here, it was set to depend on the downsampling distance, i.e. \( 2d \).

• The comparison constant \( t_{\text{poly}} \) for polygonal edge lengths (new parameter). Referenced at row 7 of algorithm 4, this parameter determines how easily we reject a pose after the first transformation guess. A low \( t_{\text{poly}} \) might risk rejecting good poses, while a too large \( t_{\text{poly}} \) will make the algorithm run like a normal (slow) RANSAC. Here, it was set to 0.25, allowing an edge dissimilarity of 25%.

Most of these parameters do have an effect, but as with the previous algorithms these effects are small compared to the changes of the downsampling distance. Therefore these parameters are given constant values which seem to provide the best general results. In figure 3.8 is an example of an alignment with Prerejective RANSAC is shown, allowed to run for 10 000 iterations.

### 3.4 HoloLens implementation

The HoloLens implementation is slightly different than the one explained above. This is partially because of time constraints, where the previously used PCL code
library cannot be integrated easily with the *Universal Windows Platform* (UWP) which the HoloLens runs on. Thus, every step in the final algorithm had to be re-implemented for the UWP. In figure 3.9 the modified complete flowchart is shown. Some notable differences:

- The preprocessing step, where the mesh is upsampled and the ToF data MLS resampled, is removed. Both these steps were deemed time sinks both in terms of computation time and implementation time, compared with the improvements they provided.

- The mesh already had normals for every vertex through the online HoloLens interface. When extracting the mesh in real time the normals are instead sparse, with roughly 80% of vertices arriving with precomputed normals. This is solved by noting the indices which lack a computed normal, and computing the normals only for those points.

An important difference in this case versus the ToF data normal estimation is the sign of the plane normal. Before, we could assume each normal pointed toward the camera position. As the mesh is generated over time, there is no ‘one’ camera position. The sign is instead found by making sure that the sign of the newly estimated normal is consistent with the direction of normals of the neighbourhood. As a neighbourhood is already computed when we estimated a plane, and most of the points already had normals, we use the neighbour set to find their mean normal.

- The registration block only includes one algorithm, the FGR algorithm, as this is meant to be a running application and not a test platform. A motivation for the choice can be found in chapter 5.
3.5 Assumptions and Simplifications

Originally the HoloLens supplies the entire mesh it creates, which can be used as input to the registration. This is often a complete room, with all walls, floor and roof. However, this would make the problem very hard to solve, as it introduces more ambiguities for true corresponding points. As the position and the direction the HoloLens is facing is available, a frustum can be used to cull any irrelevant data. This imposes the restriction that whomever is wearing the HoloLens needs to look at the part of the room the ToF camera is looking at, which seems like a reasonable requirement. On the test platform this is mimicked by manually reducing the mesh to the portion of the scene where the camera is faced.

With only a portion of the scene, a rough initial rotation can be found by making sure the mean normal of the ToF data faces the same direction as the mean normal of the culled mesh. This can be expressed as

\[
\frac{\sum_{i \in \text{ToF}} \mathbf{n}_i}{\left\| \sum_{i \in \text{ToF}} \mathbf{n}_i \right\|} = \frac{\sum_{j \in \text{mesh}} \mathbf{n}_j}{\left\| \sum_{j \in \text{mesh}} \mathbf{n}_j \right\|}.
\]

This is hereinafter referred to as transforming the data to a **common normal**.
This chapter presents the acquired results from the experiments made with the different registration algorithms.

Each evaluation of a transformation is done by comparing the ground truth transformation with the one produced by the registration algorithm, according to section 3.3.4. There, equations (3.2) and (3.3) define $\varepsilon_{\text{trans}}$ and $\varepsilon_{\text{rot}}$, which are a two part error metric for how close the estimated transformation is compared with ground truth. All translations are in meters, and rotations in radians.

All the three algorithms were evaluated for accuracy and execution time. Every data point for each parameter search is a mean value for many registrations using the same parameters.

Each plot presented in this chapter shows the translation and rotation errors from registration experiments using the different algorithms, mainly varying the downsampling distance $d$. The mean of 40 registrations with identical parameters were used to generate each data point. 30 points were generated for every x-axis value, where the set of points in a 'column' indicate how the error might vary between registrations due to nondeterministic properties of the registration algorithms. In the case of JR-MPC, which is very consistent (but not quite deterministic), a 'column' of points is instead acquired by altering a second parameter apart from $d$. All other parameters are set according to the corresponding description in section 3.2.

A rotation error of below $\pi/6$ and a translation error below 1 meter should be good enough to be fixed by a refinement step using ICP.

### 4.1 Data sets

The experiments presented in this chapter were performed on data gathered from the HoloLens and the depth camera in a furnished 7x8 meter room.
8 different depth images were produced, where each image consisted of $176 \times 144 = 25344$ points before downsampling. These were images facing walls or corners of the scene, always including sections of floor and roof.

As the HoloLens can capture a complete room, many ToF images can be tested on the same HoloLens mesh. Three differently furnished room meshes were produced, each at roughly 200 000 points. Each registration was performed on a subset of the complete room mesh; either of a section of a wall, or a corner where the ToF camera was facing. Such a subset includes roughly 50 000 points before downsampling. This is to replicate the frustum culling which can be done on the HoloLens. There is a large expected overlap between the data. The plots presented in the chapter are from one specific ToF image registrated on one specific room mesh, deemed to be a good representation of the overall registration performance for the methods. Extensive tested were performed on all data sets, but it is not presented here.

A complete HoloLens room mesh can be seen in the top of figure 4.1, as well as two examples of ToF data images from inside the room at the bottom. The bottom left colored ToF image is the data set used to create the result plots.

The mesh is considered to be in the world coordinate system, which the ToF point cloud is to be transformed to. A randomized transformation matrix is applied to the ToF data before registration as an initialization. Thus, the transformation to be estimated may be any rigid transformation matrix. As a part of the registration algorithms, the centerpoint of the ToF data was shifted to the centerpoint of the partial mesh, thus ensuring some sort of adjacency. Because of this the randomized translation part of the initialization transformation was not
needed.

4.2 FGR

Figure 4.2 show eight different plots with four different sets of methods applied before the registration. Each row include the translation and rotation errors produced by a parameter search using FGR. MLS resampling is done only on rows 3 and 4, and transforming to a common normal for the mesh and ToF data is done only on rows 2 and 4. Each point is a mean of several registrations, where many points are produced for each parameter value. The remaining parameters were set according to the description of FGR in section 3.2.

Figures A.1 and A.2 in appendix A show the execution time difference between excluding and including MLS resampling. The time it takes to transform to a common normal is considered negligible.

4.3 JR-MPC

JR-MPC suffers from getting stuck in local minima. As much of the ToF data here is depth images of walls, or planar surfaces, an initial rotation error larger than 180° fails consistently. Because of this, a rotation to a common normal is included for all parameter searches.

MLS resampling is not used for JR-MPC; because the fine details it corrects does not affect an algorithm which only considers point density, its impact would be minimal.

In figure 4.3 the translation and rotation error can be seen for a parameter search using JR-MPC. Each column in the graphs represents the different errors achievable when changing how many distributions are used during registration, varying from 8 to 210. In general, more distributions will give a tighter alignment, but increase execution time. The remaining parameters were set according to the description of JR-MPC in section 3.2. The same data but along the distribution axis, and zoomed 8 times, can be seen in figure 4.4, where each column instead represents the error while changing the downsampling radius. The corresponding execution time plots of figures 4.3 and 4.4 can be found in figure A.3 and A.4 in the appendix.

4.4 Prerejective RANSAC

Results from a parameter search for Prerejective RANSAC can be seen in figure 4.5. For this parameter search MLS resampling was included, as well as a transformation to a common normal direction. Each column in the graph represent the variance of error between registrations, changing no parameters. This gives an indication of the robustness of the registration algorithm. Important to note is the larger downsampling distances than any other of the registration algorithms.
(a) Translation error of FGR with no data refinement.

(b) Rotation error of FGR with no data refinement.

(c) Translation error of FGR, only transforming to a common normal.

(d) Rotation error of FGR, only transforming to a common normal.

(e) Translation error of FGR, using only MLS resampling.

(f) Rotation error of FGR, using only MLS resampling.

(g) Translation error of FGR, using both MLS resampling and transforming to a common normal.

(h) Rotation error of FGR, using both MLS resampling and transforming to a common normal.

Figure 4.2: Four parameter searches with FGR, each with a different set of methods applied before the registration. Each data point is the mean error of 40 registrations, where each column in a graph consists of 30 data points.
4.4 Prerejective RANSAC

(a) The translation error for a parameter search using JR-MPC.

(b) The rotation error for a parameter search using JR-MPC.

Figure 4.3: The translation and rotation errors of a parameter search using JR-MPC. Each column shows the different errors when varying the number of used distributions, varying from 8 to 210. Each data point is the mean error of 40 registrations (with identical parameters), where each column consists of 30 data points.

(a) The translation registration error for a parameter search using JR-MPC.

(b) The rotation registration error for a parameter search using JR-MPC.

Figure 4.4: The translation and rotation errors of a parameter search using JR-MPC. Each column shows the different errors when changing the down-sampling radius, varying from 3cm to 16cm. Each data point is the mean error of 40 registrations (with identical parameters), where each column consists of 30 data points.
Results

(a) The translation registration error for a parameter search using Prerejective RANSAC.

(b) The rotation registration error for a parameter search using Prerejective RANSAC.

Figure 4.5: The translation registration and rotations errors for a parameter search using Prerejective RANSAC. Each column represents the variance of error when performing registrations using identical parameters. Each data point is the mean error of 40 registrations, where each column consists of 30 data points.

This is because, as a random search algorithm, a too large search space will require an intractable number of iterations to consistently find good solutions. Reducing the number of points both increases the probability of finding a good transformation, as well as reduces the execution time. The remaining parameters were set according to the description of Prerejective RANSAC in section 3.2.

Figure A.5 in appendix A shows the corresponding execution time graph.

4.5 Comparison

In table 4.1 a summary of the most prominent results can be seen. For FGR and Prerejective RANSAC the values are the mean errors of all registrations using the specified parameters. For JR-MPC, the error values are instead from one registration, but as the algorithm is very consistent the same errors are expected for every registration. The error values displayed include the smallest error achieved by all the algorithms (regardless of computation time), the fastest execution time (within reasonable downsampling distances), and the best combination of time versus accuracy.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$d$ (cm)</th>
<th>$\varepsilon_{\text{trans}}$ (m)</th>
<th>$\varepsilon_{\text{rot}}$ (rad)</th>
<th>Time (s)</th>
<th>Misc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGR (smallest error)</td>
<td>3.8</td>
<td>0.35</td>
<td>0.13</td>
<td>3.31</td>
<td>MLS, com. n.</td>
</tr>
<tr>
<td>FGR (fastest time)</td>
<td>15.7</td>
<td>6.18</td>
<td>1.51</td>
<td>0.31</td>
<td>com. n.</td>
</tr>
<tr>
<td>FGR (best combination)</td>
<td>8.1</td>
<td>0.68</td>
<td>0.16</td>
<td>0.99</td>
<td>MLS, com. n.</td>
</tr>
<tr>
<td>JR-MPC (smallest error)</td>
<td>3.0</td>
<td>0.20</td>
<td>0.08</td>
<td>975.5</td>
<td>210 distrib.</td>
</tr>
<tr>
<td>JR-MPC (fastest time)</td>
<td>15.7</td>
<td>0.61</td>
<td>0.25</td>
<td>2.28</td>
<td>8 distrib.</td>
</tr>
<tr>
<td>JR-MPC (best combination)</td>
<td>15.7</td>
<td>0.16</td>
<td>0.13</td>
<td>5.16</td>
<td>36 distrib.</td>
</tr>
<tr>
<td>Pr. RAN. (smallest error)</td>
<td>11.6</td>
<td>2.39</td>
<td>1.24</td>
<td>0.91</td>
<td>10 000 iter.</td>
</tr>
<tr>
<td>Pr. RAN. (fastest time)</td>
<td>22.6</td>
<td>5.33</td>
<td>1.71</td>
<td>0.36</td>
<td>10 000 iter.</td>
</tr>
<tr>
<td>Pr. RAN. (best combination)</td>
<td>11.6</td>
<td>2.39</td>
<td>1.24</td>
<td>0.91</td>
<td>10 000 iter.</td>
</tr>
</tbody>
</table>

**Table 4.1:** Table of the three most prominent registrations for each of the algorithms. The registration with smallest error, the one with fastest time, and the best combination of execution time versus accuracy.
Discussion

This chapter discusses the results presented in chapter 4; what worked and what did not, as well as possible reasons for some shortcomings. The algorithm which turned out to be the best candidate for implementation on the HoloLens is FGR, but not without issues, which are explored below.

As all data points from the graphs shown in chapter 4 is the mean error of 40 registrations, the errors will be both smaller and larger for individual registrations at those same parameters, due to nondeterministic properties. The registration algorithms include some random elements, which affect the results slightly. What this means is that the threshold for ICP from chapter 4 (1 meter translation and $\pi/6$ rotation error) to be able to correctly align the point clouds may be exceeded for individual registrations, i.e. we cannot guarantee success. As an example, if an algorithm produced a translation error of 0 for 38 registrations, and 20 meters for 2 registrations, the mean error would be 1 meter.

5.1 FGR

The FGR algorithm showed the most promise in the initial search for algorithm candidates, and seemed like a very fast, robust solution. What quickly appeared during testing was how it required much help to reach a good global solution. As can be seen in figure 4.2 the initial rotation to a common normal was of utmost importance. Without a rotation to a common normal the results were unusable. MLS resampling helped slightly with accuracy, but also, as can be seen by the spread of points, added to the variance of the error.

While FGR is a truly global registration algorithm, it relies heavily on a good initial correspondence set found through the FPFH features. Surprisingly, FGR is unable to find the correct transformation with a bad initial rotation as figure 4.2 shows, while a small initial rotation error (but a large translation error) had
high success rate (on rows 2 and 4 of the figure). The only difference for the FPFH features with different initial rotations should be the downsampling. The features are transform invariant, but the downsampling is axis aligned, which means that a slightly rotated point cloud would give a slightly different distance between points. While this certainly gives a different downsampling, it is unclear why it affects the results to the point where the algorithm is unusable without an initial correction.

Table 4.1 and figures A.1, A.2 show that FGR can produce good results within a reasonable time, but occasionally individual registrations might exceed the ICP threshold from chapter 4 and fail.

This means that as long as the assumptions made about the data during testing still holds, the presented accuracy versus execution time is the best among all the algorithms. The contender would be Prerective RANSAC, which also had very low execution time. The issue there is the consistency problem, which increases the average error. While this can be fixed by having more iterations, this increases execution time.

5.2 JR-MPC

While the graph in 4.3 certainly looks promising regarding accuracy, this is for a scene well suited for the JR-MPC algorithm. A scene with fewer large objects (such as tables) would make it significantly harder to gradually converge to a correct solution. The figure also unveils the downsampling distance having little to no effect on the results, which is good when considering execution time, as a lot of points can be pruned without affecting the results significantly.

The number of distributions in figure 4.4 show that the translation error seems to decrease to a minimum for a distribution count of around 30. This is probably due to differences in the mesh and depth image data, where artefacts in the data fools the probability distributions. The corresponding rotation error only decreases with more distributions. JR-MPC suffers when small details are important, as only point density is taken into account, which more distributions remedies somewhat. It was brought up in section 3.1 that including normals might increase accuracy robustness, but also using color information or at least having anisotropic variance of the distributions should help. Also, as JR-MPC is not a true global solution, the assumptions that could be made here, that a mean normal for a data set held information regarding the general placement of the data, may not be present in other situations. This would severely damage the registration, as it would be sensitive to any initial rotation error > 180°. Regardless, JR-MPC was the most consistent of all tested algorithms, where all registrations using more than 8 distributions could be aligned using an ICP refinement step.

Figure A.3 and A.4 show that the execution time becomes very large for a small downsampling distance and many distributions. To be comparable to the time it takes to do an adequate 3D reconstruction with the HoloLens, roughly 20-30 seconds, a low number of distributions and points are needed. Despite choosing the absolute fastest parameters, there is a large difference between JR-
MPC and the other algorithms. Taking into consideration that the HoloLens runs registration 10-20 times slower than the testing PC, anything over 2 seconds during tests is considered too long.

Although JR-MPC does not fit this problem, it proved to be a very robust method with the right input data, as table 4.1 and figure 4.3 shows. The time a registration takes could arguably be reduced to a tractable duration with enough pruning of points, improvements in the implementation and optimization of the number of distributions.

5.3 Prerejective RANSAC

Prerejective RANSAC provided surprisingly robust results early in the testing. While RANSAC is a simple method in general, it often solves problems where more complicated solutions would only improve results slightly. Despite this, it suffers from inconsistency, as figure 4.5 illustrates, and to improve it a lot of iterations are needed. As Prerejective RANSAC uses FPFH features to search for correspondences for the initial $n$ points drawn from one of the data sets, other features which provide better transform invariant descriptors would improve both robustness and execution time.

Figure 4.5 shows a parameter search for rather large downsampling distances. When decreasing the number of points the accuracy actually increases somewhat. This might seem weird, but it makes sense for a random search algorithm, since the search space is reduced. This incidentally also reduces computation times, so a well chosen $d$ is essential. Because of this the parameter search only includes larger values of $d$. A smaller downsampling distance with a lot more iterations would probably give similar, or better accuracy, but would increase execution time significantly. Currently the execution times are very competitive with FGR, seen in figure A.5.

Increasing the number of iterations used would most likely reduce the randomness figure 4.5 illustrates, but would put the execution time at several times slower than FGR, and would still not guarantee sufficient results. The figure shows mean errors way above the ICP threshold from chapter 4, but this does not mean that every registration fails. Rather, there is a higher probability of producing registrations with insufficient, or large errors. Especially since a RANSAC algorithm might produce registrations with very large errors, thus skewing the mean errors.

5.4 HoloLens implementation

The HoloLens implementation, despite using re-implementations of all the steps in the registration pipeline, ran at roughly the same number of computations as the test platform program. Important to note is that the HoloLens has much lower processing power than most stationary computers, which prolongs the execution time of the registration. Despite this, a complete registration pipeline could be done within 10-30 seconds, depending on parameters, which is comparable to the
time is would take to create a mesh from scratch while walking around with the HoloLens. This means that the long time to perform the registration would barely be noticed, as it is time being spent waiting anyway. The algorithm chosen to be implemented on the HoloLens, FGR, was fast enough to not be very noticeable, while still producing mostly good results.
Conclusions and Future Work

This report has explored the topic of point cloud registration on a Microsoft HoloLens, using the mesh it creates together with a ToF image of the same scene. The point cloud registration algorithms FGR, JR-MPC and Prerejective RANSAC have been tested for this purpose, where FGR proved to be the most fitting solution as discussed in chapter 5. This chapter seeks to answer the questions posed at the beginning of this document, as well as give directions for future work.

6.1 Conclusions

- *Is the HoloLens and the SICK ToF camera data reliable and dense enough to be subject to robust registration?*

The data from the HoloLens mesh and ToF depth data definitely differs, especially where the ToF camera encounters steep angles or reflective surfaces. Despite this, the data is deemed *similar enough* to give adequate results. Further pre-processing or assumptions about the data may improve the results even more.

- *Are the registration algorithms available today sufficient in adequately solving the problem?*

While the evaluated algorithms required some setup and assumptions about the data, it is possible to acquire good results most of the time, with good tuning. While JR-MPC was the only really consistent algorithm, FGR could be tuned to align correctly almost every registration. How well the algorithms would operate using different features may differ, and improve results even further.
6 Conclusions and Future Work

• Can the performance of the above mentioned algorithms be optimized to the point where they can be used in real time on the HoloLens headset?

The duration of a registration with the chosen algorithm FGR, while having good enough parameters to ensure adequate registration, is comparable with the time it takes to create a 3D reconstruction with the HoloLens. A rough mesh is created in about 30 seconds, where an FGR registration with the found optimal parameters on the HoloLens takes 10-15 seconds. This means that the time spent doing the registration would need to be spent standing around anyway, to wait for a sufficiently large mesh to be created.

6.2 Future Work

This thesis only examined registration algorithms, where they used the FPFH features (where applicable) and 3D points as input. It would be interesting to do a similar survey, but for several other transform invariant features, which might work better given that the data from the two sensors are slightly different.

There is also the chance to expand the number of algorithms which are searched and evaluated. As a whole test platform has been set up, it would require little work apart from having a C++ implementation to start evaluating their performance. Any additional algorithms must of course still fulfil the global solution requirement, or expand the possible assumptions to the point where local methods also solves the problem.

A more product focused future line of work would be exploring the HoloLens application, where instead of a test application a finished product is developed for camera calibration. This may lie more in the interest of SICK, and not academic work.
Appendix
This appendix includes the time plots of each of the different registration algorithms.

### A.1 FGR

*Figure A.1:* Plot of the execution time of the FGR algorithm, with the pre-processing pipeline excluding the MLS resampling step.
Figure A.2: Plot of the execution time of the FGR algorithm, with the preprocessing pipeline including the MLS resampling step.

A.2 JR-MPC

Figure A.3: Plot of the execution time of the JR-MPC algorithm. Each column represents the time variance when changing how many distributions are used in the registration, varying from 8 to 210. A large number of distributions and a low downsampling distance gives an intractable execution time.
A.3 Prerejective RANSAC

Figure A.4: Plot of the execution time of the JR-MPC algorithm. Each column represents the time variance when changing the downsampling radius, varying from 3 to 16 cm.

Figure A.5: Plot of the execution time of the Prerejective RANSAC algorithm.


