Minimal Deformation Methods for Loop Closure

Erik Tegler

Master's thesis
2021:E3

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Minimal Deformation Methods for Loop Closure

Student:
Erik Tegler
erik.tegler@gmail.com

Supervisor:
Gabrielle Flood
gabrielle.flood@math.lth.se

Supervisor:
Kalle Åström
kalle@maths.lth.se

Examiner:
Anders Heyden
anders.heyden@math.lu.se

Abstract

This thesis proposes a method for how to find duplicated 3D points in a single Structure from Motion point cloud. Together with related articles, this forms a possible solution to the loop closure problem.

The proposed method works by first selecting candidate pairs of 3D points by comparing BRIEF descriptors of all points. The second step consists of using RANSAC to select the best out of many possible deformations of the map. The deformations are created by finding the minimal increase in reprojection errors while fulfilling the added constraint that a selected candidate pair has to converge. Finding this minimal increase is done by assuming the residual is linear under small changes. The solution is then found by optimizing over the subspace spanned by the smallest eigenvectors to the Hessian for the loss function.

The method was tested on two different datasets. It managed to correctly identify the main duplication in both sets. In the first and simpler of the two datasets, which had 1000 points, all of the 33 pairs identified by the method were verified by hand to be correct. On the second dataset, which had 6000 points, the method found 153 pairs. Seven of the found pairs corresponded with the main split in the map. The rest of the found pairs were points which were already very close together in the original map. This might hint at a possible problem with the original map creation. One which this method could help to solve.

Contents

Acknowledgements

I first and foremost want to direct my thanks to my two mentors: Gabrielle Flood and Kalle Åström. Both have helped me a lot with their deep knowledge of the subject. This together with the fact that they are always positive and supportive makes me very grateful to have had them as mentors. Secondly, I also want to thank Patrik Persson for allowing me to use his maps, which I have have used continuously throughout the thesis project. Finally, I want to thank my friends and family who have supported me throughout the process of making this thesis and had to bear with me in my sometimes overeager attempts to explain what this work is about.

Popular Science Summary

We humans have two eyes. Because of this, we can perceive depth. This is commonly called stereo vision. In the same way, imagine we had two cameras of which we knew the positions, we could then triangulate the position of objects present in both images. As it turns out, if we have enough images we do not need to know the position of the cameras. It is actually possible to determine both a 3D map of what is being photographed and also find where the images were taken. This problem is called Structure from Motion and is a central part of the field of Computer Vision. An example of what such maps could look like is shown in Figure 1. Notice that we do not get complete surfaces, but rather a lot of points in 3D space. This is commonly referred to as a point cloud.

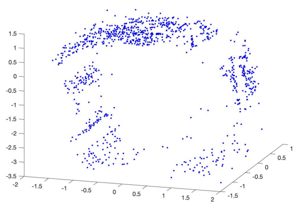


Figure 1: Example of a 3D map of a room. The four walls are what can be seen in the figure.

Usually, the images are collected by using a film camera which is moved around a path. Because of how the map is generated, if we walk in a closed loop the algorithm does not always recognize that a new point is the same 3D point as an old point. The algorithm then creates a second instance of the 3D point which is slightly shifted in space. This is called a Loop Closure problem. The goal for this thesis is to propose a method which solves a part of this problem. More precisely, this thesis is trying to find pairs of points which should actually be the same 3D point. The method requires searching for points which look very similar. However, just looking at similarity would result in too many false matches. Because of this, we propose to also filter for points which could be combined without altering the original map too much. As can be seen in Figure 2 the algorithm can successfully identify pairs which have been duplicated.

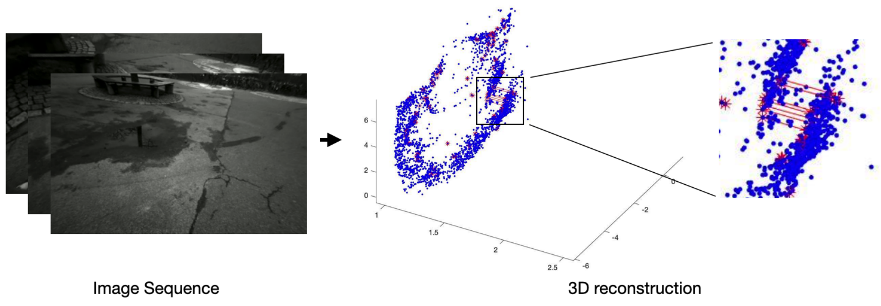


Figure 2: One the left side of this figure some example images from the image sequence are shown. On the right side of this figure is the 3D reconstruction shown together with the duplication caused by walking in a circle. The lines are drawn between points which the algorithm thinks should actually be the same point. Notice that there are red lines going between the duplicated part.

1 Introduction

The field of Computer Vision is the study of how to extract high level information from image and video. Phrased in a different way, it is the study of how to automate the function of the human eye and visual cortex. Together with image analysis these fields have a wide variety of applications such as self-driving cars and face recognition to mention a few. One large part of these fields is to solve the Structure from Motion (SfM) problem.

We humans have two eyes. Because of this, we can perceive depth. This is commonly called stereo vision. In the same way, if we have two cameras with known positions, we can triangulate the position of objects present in both images. As it turns out, if we have enough images we do not need to know the positions of the cameras. It is actually possible to determine both a 3D map of what is being photographed and also to find were the images were taken. This is the Structure from Motion problem, see Figure 3.

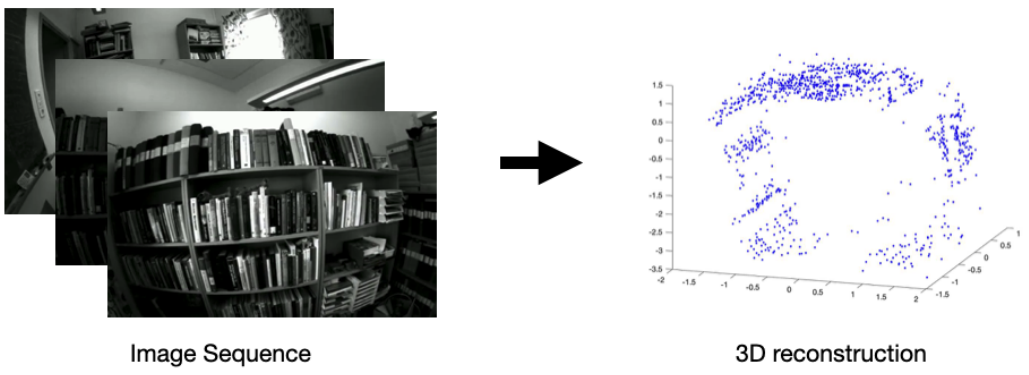


Figure 3: The SfM problem is making a 3D reconstruction from an image sequence.

With the method used in this thesis, we do not get complete surfaces but instead a lot of points in 3D space. This is sometimes referred to as a point cloud. The reason we only get discrete points is because we need to find similarities in the images and this is done by identifying interesting feature points in each of the images. When the same 3D point is identified in multiple images, it is possible to estimate its 3D position. This process will be further described in the following sections.

One problem which can occur if the image sequence loops back to a part of the scene previously depicted is that the algorithm does not realize that it sees the same 3D points again. This result in the creation of 3D points which are duplicates of points already in the map. This is called a loop closure problem, and will be further described in Section 6. However, one reason to care about this is that we could improve the entire 3D reconstruction if we use the information that we have returned to a previously known point. In fact, given that we know which points belong together, Flood et al. has already shown how to make this improvement, see [1, 2]. The goal of this thesis is to develop methods to identify the duplicated points so that they can combined with their method.

2 Random Sample Consensus Algorithm

The problem of fitting a model to some data is core to most of science. Probably the simplest form of this is fitting a line to a set of points $\begin{bmatrix} X & Y \end{bmatrix}$. There are multiple ways of doing this, for instance the total least square method, which minimizes the sum of the squared distance between the line and the points. One problem with this method is that it is very sensitive to outliers. In many cases in Computer Vision you might have the case that only a small percentage of points are correct, while the rest are simply noise, see Figure 4. For these kinds of problem total least squares will give a very poor result.

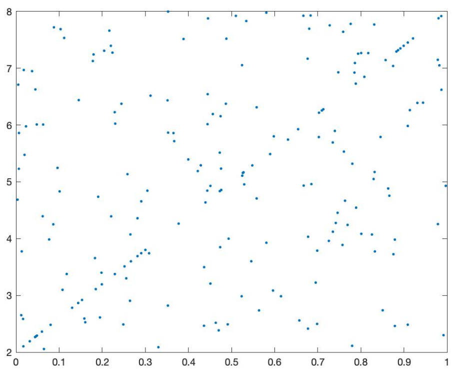


Figure 4: In this figure 80 % of the points in the image are not following the pattern but it still fairly easy to see the correct line.

One way of solving this problem is to use a Random Sample Consensus (RANSAC) algorithm [3]. The idea in RANSAC is to just study a small subset of points, with the hope that all of the points in the subset are inliers. We then compute our model from the subset. We now go back to the original set and see how many points agree with this model. Repeating this process many times and saving the model with which the most points agree gives us a good chance of finding the right model.

To use the data in Figure 4 as an example. We need two points to fit a line. Iterating the process of picking two points, fitting a line to them and seeing how many points are on the line should give us the correct model. The probability of finding the right model can be computed as

$$P(\text{finding subset of only inliers at least once}) = 1 - (1 - \epsilon^n)^l, \tag{1}$$

where ϵ is the proportion of all the points being inliers, n is the size of the subset and l is the number of iterations. Usually, we want to know how many iterations we need in order to have a high probability $P > 0.99$ chance of finding the right model. In the case of Figure 4, we would need $l \gtrsim 120$ iterations.

From this we can also see that we want to select as small a subset as possible, since that increases our chance of finding a subset of only inliers. In the example of Figure 4, the probability of selecting two inliers is $0.2^2 = 0.04$, while selecting three points gives us the probability $0.2^3 = 0.008$. Also, since we need to fit a model to a set of points many times, a quick method of computing the model is important. This can be achieved using minimal solvers, see [4].

In addition to many points being completely incorrect, we usually have some sort of measurement noise. This means that even the correct points will not fit the model perfectly. In this case we need to decide on some tolerance for how much the points can deviate from the model and still be considered an inlier, see Figure 5.

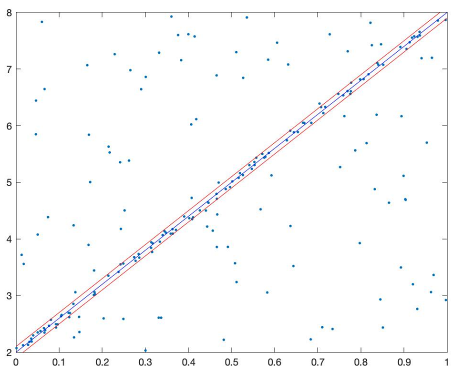


Figure 5: The blue line shows the model. The red lines show the tolerance bound. We count all the points that are between the red lines as inliers

3 Feature Detection

The field of Computer Vision aims to use a set of 2D images to recreate the 3D world they were taken in. How this is done will be discussed in Section 4. However, before this, an essential piece of image analysis is needed, a feature detector. What a feature detector should be able to do is finding 2D points (features) in different images which corresponds to the same 3D point, see Figure 6. There exists multiple different feature detectors such as SIFT [5] or ORB [6]. In this thesis ORB was used.

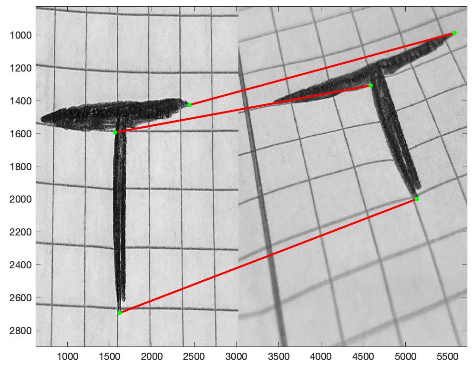


Figure 6: Example of the desired result of the feature detection. Features (green dots) which are paired together are marked with a red line.

The important part, in this report, is to understand that the feature detector pairs 2D points in different images which corresponds to the same 3D point. The remaining of this section discusses how the ORB detector achieves this based on [6].

3.1 ORB Detector Steps

The ORB detector has the following steps:

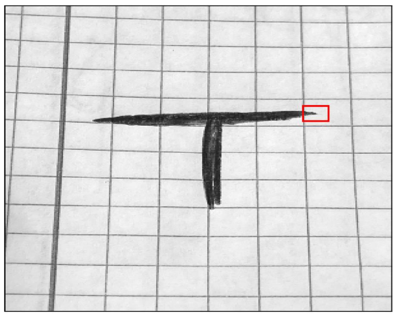
1. Identify the N most distinct corners and save a small patch around each one.
2. For each patch, rotate it so that image planar rotation of the initial image yields the same patch.
3. Compress the information for each patch into a descriptor.

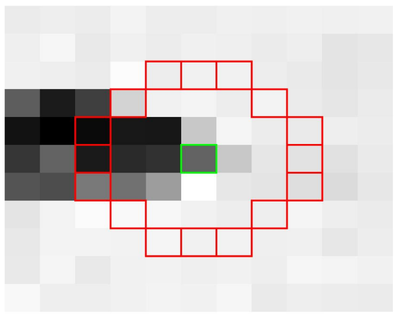
3.2 ORB: Identifying Corners

The first step is running the FAST corner detector. For each pixel, compare it with each pixel in a circle around it. Each of the pixels in the circle is assigned one of three labels based on the difference in intensity between the peripheral and the center pixel

$$L(I_c, I_p) = \begin{cases} \text{Light}, & I_p > I_c + tol, \\ \text{Dark}, & I_p < I_c - tol, \\ \text{Same}, & \text{else}, \end{cases} \tag{2}$$

where I_c and I_p respectively is the intensity of the center and peripheral pixel, and tol is some chosen tolerance. After labeling all the pixels along the periphery we check if there are m contiguous pixel along the circle which all have the label "Light" or all have the label "Dark". If this is the case the center pixel is considered a corner. Here m and tol are variables which can be tuned for different results. An example of this procedure can be found in Figure 7. The circle of pixels in the figure have a radius of 3 pixels in order to make it easier to visualize. However, in ORB they use a radius of 9 pixels. Depending on what values are chosen for tol and m we get different number of corners. To decide on the best corners ORB runs a Harris corner measure [7] and picks the chosen number of corners. For each of these pixels a patch around them is used for the computations in the following steps.


(a) Original image


(b) Zoomed in image. The green square marks the center pixel. And the pixels marked with red are pixels included in the periphery.

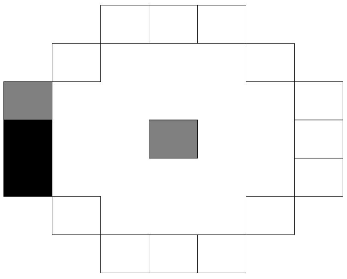

(c) The labels of the pixels are displayed with [Light,Same,Dark] being shown by [White,Gray,Black]. Because we have 13 contiguous "Light" pixels, the center pixel i categorized as a corner point.

Figure 7: FAST corner detector example

3.3 ORB: BRIEF Descriptor

A Binary Robust Independent Elementary Feature (BRIEF) descriptor is a binary vector where each bit corresponds to a comparison of intensities between two pixels. In other words, if the BRIEF descriptor has 256 bits, we are making 256 comparisons between pairs of pixels. The goal with the descriptor is to compress the data from each patch in such a way that similar descriptors likely corresponds to the same 3D point. Which pairs of pixels to compare in order to achieve this is a difficult problem. When ORB was developed, the developers ran a learning method to decide on which comparison to make [6]. We therefore have 256 pairs of pixels which should be compared. Gather them on the form

$$p_{i,j}, \quad 1 \leq i \leq 2, \quad 1 \leq j \leq 256, \quad i, j \in \mathbb{N}$$

where $p_{i,j}$ is a vector containing x and y coordinate of pixel (i, j) in the coordinate system with origin at the center pixel of the patch and comparisons are made between pixel $p_{1,j}$ and pixel $p_{2,j}$, see Figure 8.

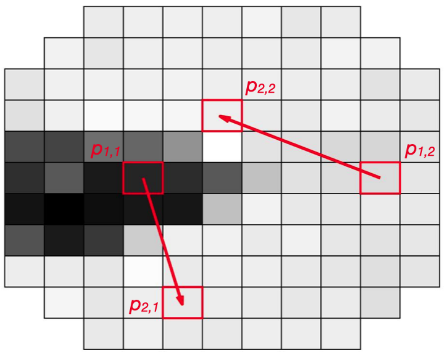
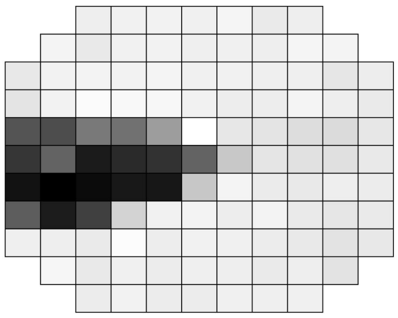


Figure 8: Shown in this figure are two example comparisons. When using ORB, 256 comparisons are instead used to create a BRIEF-descriptor.

Computing descriptors in this way would mean that a small change to the image would result in a small change in the descriptor. However, we would also like the descriptors to be similar if we rotate the image, see Figure 9.


(a) Original image

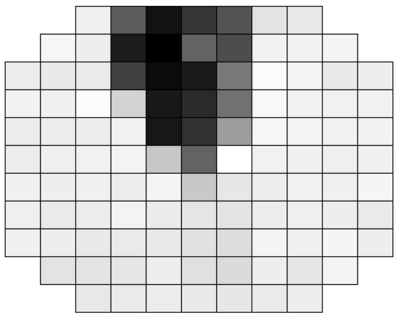

(b) Rotated version

Figure 9: We would like the descriptors for the two images to be similar.

3.4 ORB: Rotation Invariance

The goal with this step is to make features invariant to rotations in the image plane. Initially we compute

$$m_x = \sum_{x,y} xI(x,y) \quad \text{and} \quad m_y = \sum_{x,y} yI(x,y), \tag{3}$$

where $I(x,y)$ is the intensity of the pixel at position (x,y) and summing over all pixels in the patch. The rotation θ of the patch is then calculated as

$$\theta = atan2(m_y, m_x), \tag{4}$$

where *atan2* is the quadrant-aware version of arctangent, see Figure 10. If we view a pixel's intensity as its mass, we can describe this computation as: find the patch's center of mass and compute its argument.

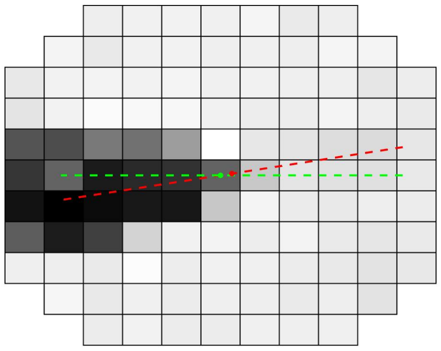


Figure 10: The center of mass is marked with a red point and the origin is marked with a green point. The angle we compute is the angle between the red and green lines, which in this case is $\theta = 11°$.

The rotation of the feature can now be handled by rotating the image by θ. This achieves the same result as what is really done, which is rotating the matrix of which pixels to compare

$$\tilde{p}_{i,j} = R_\theta p_{i,j}, \tag{5}$$

where R_θ is a rotation matrix. Because $\theta \in \mathbb{R}$ we might have non-integer values in $\tilde{p}_{i,j}$. Since pixel space is discrete, we need to decide exactly which pixels to compare. In ORB, this is solved by discretizing θ in increments of $\frac{2\pi}{30}$. What $\tilde{p}_{i,j}$ to use is then solved for each of these angles.

When each feature has its BRIEF descriptor, it is possible to compare features. Pairs of points can then be selected based on similarity of their BRIEF descriptor.

4 Structure from Motion

In the previous section we saw how to find pairs of features which likely belong to the same point. The next step is to, from these matches, compute the position of the 3D points and cameras.

4.1 Homogeneous Coordinates

Homogeneous coordinates are widely used in computer vision and is a part of projective geometry. Using homogeneous coordinates, points in $\mathbb{R}^n$ are represented using vectors in $\mathbb{R}^{n+1}$. To exemplify how to map from homogeneous coordinates to Cartesian coordinates, we will go through the case of homogeneous coordinates $X \in \mathbb{R}^3$ to the corresponding Cartesian coordinate $x \in \mathbb{R}^2$. Find the intersection between the line going through X and the origin and the plane defined by $(a, b, 1)^T$, where $a, b \in \mathbb{R}$, see Figure 11. The corresponding Cartesian point is first two coordinates of the intersecting point. Another way of saying the same thing is that the point $X = (a, b, c)$ corresponds to the point $x = (a/c, b/c)$. This also means that if the two points in homogeneous coordinates satisfy $X_1 = \lambda X_2$ then both X_1 and X_2 corresponds to the same point in $\mathbb{R}^2$.

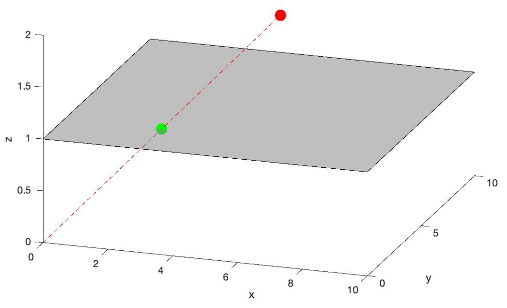


Figure 11: The red point is the homogeneous coordinate $(6, 4, 2)$ which corresponds to the Cartesian coordinate $(3, 2)$, the first two coordinates of the green point.

Homogeneous coordinates are a more complex representation but they also carry with them some advantages. For instance, we can represent points infinitely far away by setting the z-coordinate to zero. However, The main reason we want to use homogeneous coordinates is that they turn affine transformation such as

$$y = Ax + b, \tag{6}$$

into linear transformations

$$Y = \begin{bmatrix} y \\ 1 \end{bmatrix} = \begin{bmatrix} A & b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \tilde{A}X. \tag{7}$$

4.2 The Camera Matrix

We want a good representation to transform a point in 3D space to a 2D point in the image taken by a camera which we know the position and orientation of. What we mean by taking an image is a projection of the 3D-point onto a image plane. Here it is helpful to think of a pinhole camera, see Figure 12. Similar to above, the way we compute this projection is by taking the point where the line (going through the origin and 3D point) intersects the image plane. The difference is that the image plane is one unit in front of the origin in our homogeneous to Cartesian transformation, while it is behind the origin in a pinhole camera. This is also why the image is inverted in a pinhole camera. We can interpret going from homogeneous coordinates to Cartesian coordinates as taking an image with a camera situated in the origin and pointing along the z-axis, see Figure 13.

We can now imagine a procedure for computing images if we have a set of 3D points X_i and a camera situated at c pointing in the direction v. We then compute the image by first translating all the points by $-c$, so that the camera is at the origin. We then apply a rotation R_v so that the v is mapped on $(0, 0, 1)^T$. This can also be expressed as:

$$\tilde{X}_i = R_v(X_i - c). \tag{8}$$

The final step is to turn the homogeneous coordinates into Cartesian 2D coordinates. Since we are applying an affine transformation in the first step, a simpler way would be to store the original 3D coordinates in homogeneous coordinates, i.e. a vector in $\tilde{X}_i = (X_i, 1) \in \mathbb{R}^4$. We then multiply it with a matrix

$$P = \begin{bmatrix} R_v & -R_v c \end{bmatrix}, \tag{9}$$

which defines what camera we are using. This matrix is called the camera matrix. Note that this is just a rephrasing of the same computation, since

$$P\tilde{X}_i = \begin{bmatrix} R_v & -R_v c \end{bmatrix} \begin{bmatrix} X_i \\ 1 \end{bmatrix} = R_v X_i - R_v c = R_v(X_i - c). \tag{10}$$

If we then want the 2D Cartesian coordinates in the image we then divide each $x_i = P\tilde{X}_i$ with its last entry the same way it was done in the example above.

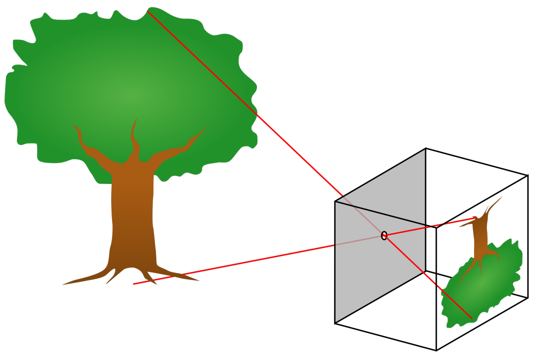
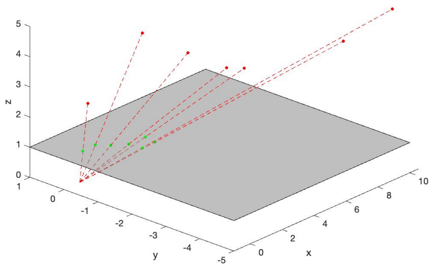


Figure 12: Points of the original tree is projected via a line through the origin (the small opening) onto the image plane at the back. Because the image plane is behind the origin, the image is flipped. Image: [8]



(a) The red 3D points being projected onto the gray image plane.

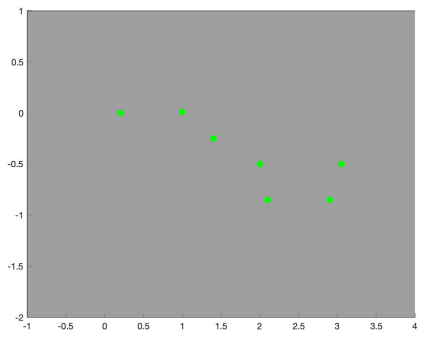

(b) Image result.

Figure 13: Projections of 3D points onto image plane.

When we are talking about real cameras we also have to take into account a few more factors such as what lens and sensor are used. Both this and the fact that the center of the image in pixel space is not $(0, 0)$ is dealt with by splitting the camera matrix into two factors

$$P = K \begin{bmatrix} A & t \end{bmatrix}, \tag{11}$$

where K is called the intrinsic parameters and $\begin{bmatrix} A & t \end{bmatrix}$ is the normalized camera matrix. If we use the same camera for all taken images we know that K is the same for all camera matrices and A is an orthogonal matrix.

When mentioning camera matrices in rest of this thesis, we will be talking about the normalized camera matrix. This is because, as a first step when performing SfM, we can normalize the 2D points, according to

$$x_i = PX \iff \tilde{x}_i = K^{-1}x_i = \begin{bmatrix} A & t \end{bmatrix} X. \tag{12}$$

Doing this we now know that the camera matrix consists of A which is an orthogonal 3×3 matrix representing rotation, and t is a 3×1 matrix representing translation.

4.3 The SfM Problem Statement

We can now understand the SfM problem statement. We have m images and n 3D points. What we want is to minimize the reprojection errors. This means that we want to find m camera matrices and n vectors in $\mathbb{R}^3$ such that when we compute the image, the distance between our projected points and the corresponding image point is as small as possible, see Figure 14.

This can also be formulated mathematically by gathering all the parameters from the cameras P and 3D points X into a vector $\mathbf{z}$. Because the image points are known, we can for a given $\mathbf{z}$ compute the residual vector

$$\mathbf{r} = f(\mathbf{z}), \tag{13}$$

where the function f is a short hand for: compute the reprojections of each of the 3D points and for each of the reprojections compute the distance to the corresponding image point. A more formal expression of this can be found in [9]. Each entry of the residual vector corresponds to the residual in one dimension such as the first entry corresponds to the residual in the x-direction for the first image point. When we have the residual vector, we can compute the loss function, which is the function we want to minimize. This is done by computing the sum of squared residuals

$$\min_{\mathbf{z}} g(\mathbf{z}) = \mathbf{r}(\mathbf{z})^T \mathbf{r}(\mathbf{z}). \tag{14}$$

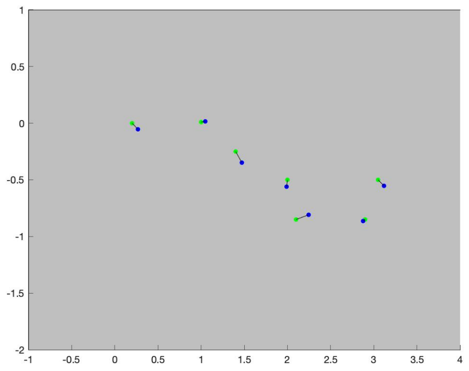


Figure 14: The green points are the reprojections. The blue points are the position of the features in the image. The goal is to minimize the sum of squares of the residuals, i.e. the sum of the squared lengths of the black lines.

4.4 Solving SfM

How to solve SfM is an entire field of study, so it will not be described in detail in this thesis. The 3D maps used in this thesis were created by using the methods described in the following sources: ORB [6], IMU data [10], filtering [11] and optimization [12].

There are multiple approaches to solve the problem. In order to give the reader an idea we will describe one possible approach, called the eight-point method [13]. This method will be easier to understand but is not what is commonly used nor used for the maps in this thesis. Further reading about solutions to SfM can be found in the following sources [14, 15].

4.4.1 Combining Two Images

First of all, instead of combining all the images at once, we instead start by looking at the case of adding just two images. Since rotation and translation of all the points and cameras together does not change the reprojection errors we fixate the first camera to be

$$P_1 = \begin{bmatrix} I & 0 \end{bmatrix}. \tag{15}$$

Remember, the structure of second camera is

$$P_2 = \begin{bmatrix} R & t \end{bmatrix}, \tag{16}$$

where R is an orthogonal matrix. We now define the essential matrix E to be

$$E = [t]_\times R, \tag{17}$$

where $[t]_\times$ is the cross product with the vector t written on matrix form,

$$[t]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}. \tag{18}$$

Given a 3D point (which is represented as a 4×1 matrix)

$$X = \lambda_i \begin{bmatrix} x_i \\ 1 \end{bmatrix}, \tag{19}$$

with the corresponding image points x_i,

$$x_i \lambda_i = P_i X, \tag{20}$$

E has the property

$$(x_2)^T E x_1 = 0. \tag{21}$$

This can be verified by rewriting the equation as:

$$\frac{1}{\lambda_2}(P_2 X)^T [t]_\times R x_1 = \frac{\lambda}{\lambda_2}(R x_1 + t)^T [t]_\times R x_1 = \frac{\lambda}{\lambda_2} x_1^T R^T [t]_\times R x_1 + \frac{\lambda}{\lambda_2} t^T [t]_\times R x_1. \tag{22}$$

Writing $Rx_1 = z$ gives us

$$\frac{\lambda}{\lambda_2} x_1^T R^T [t]_\times R x_1 + \frac{\lambda}{\lambda_2} t^T [t]_\times R x_1 = \frac{\lambda}{\lambda_2} z^T [t]_\times z + \frac{\lambda}{\lambda_2} t^T [t]_\times z. \tag{23}$$

Since $[t]_\times z$ is perpendicular to both t and z, both terms equal 0. Because the scale of P_2 is arbitrary, all matrices $E_2 = \lambda E$ are also valid essential matrices. Because of this we add the constraint that both of the non-zero eigenvalues of E has to be 1. The structure of E makes it so it only has 5 degrees of freedom. Because of this, it is possible solve for it with just 5 point-correspondences. However, then we need to be able to handle the non-linear constraint which we get from R being orthogonal. Instead, in the eight-point algorithm we just use the scale invariance of E together with 8 point-correspondences. If y_i and x_i are 2D points corresponding to the same 3D point, this is written as:

$$y_i^T E x_i = 0, \quad i = 1..8, \tag{24}$$

$$||E||_F = 1. \tag{25}$$

The solution corresponds to minimizing:

$$\min_{||v||=1} = ||Mv||, \tag{26}$$

where

$$v = \begin{bmatrix} E_{11} \\ E_{12} \\ \vdots \\ E_{33} \end{bmatrix}, \tag{27}$$

and each row of M corresponds to each equation. This can be solved by taking the singular values decomposition of $M = USV^T$ and extracting v as the last column of V, from which we compute an approximation of E called $\tilde{E}$. If the point correspondences are correct we will get a matrix $\tilde{E}$ which is close to one with the right structure. To enforce the correct structure we compute the singular value decomposition of $\tilde{E} = USV^T$ and find the correct matrix as:

$$E = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T. \tag{28}$$

We can now compute P_2 by factorizing $E = SR$, where S is skew-symmetric and R is orthogonal. Extracting t from S and combining it with R gives us P_2.

In order to find eight point-correspondences, a RANSAC loop is used. This gives us a set of inliers consisting of point-correspondences for which $y_i^T E x_i$ is small. When both P_1 and P_2 are known, the position of all the point-correspondences in the set of inliers can be computed through triangulation.

4.4.2 Combining Many Images

When we want to combine many images we first start by computing point-correspondences between all pairs of images. The pair of images with the largest overlap is then chosen and combined using the method in the section above. We now have two camera positions and a set of 3D points. Now, take a third image which has as large an overlap with the first two images as possible. Using matches between features in the third image and known 3D points we can find the position of the third camera.

When the position of the third camera is known, we can look at point-correspondences between the third and the first two images to triangulate the position of new 3D points. Repeating this process with the rest of the images will give us our final result, which hopefully is close to the global minimum. We can further improve on this solution using local optimization. [16]

4.5 Jacobian Matrix

Since we have an expression for the residual vector $\mathbf{r}$, it is possible to compute its Jacobian matrix J which show how $\mathbf{r}$ depends on z locally. Which means, if we have a solution $\mathbf{z}^*$, our first order approximation of $\mathbf{r}$ around that point is

$$\mathbf{r} \approx \mathbf{r}|_{\mathbf{z}^*} + J|_{\mathbf{z}^*} \Delta\mathbf{z} = \mathbf{r}_0 + J_0 \Delta\mathbf{z}. \tag{29}$$

This approximation turns the function we want to minimize $g(\mathbf{z})$ into

$$g(\mathbf{z}) = \mathbf{r}^T\mathbf{r} \approx \mathbf{r}_0^T\mathbf{r}_0 + 2\mathbf{r}_0^T J_0 \Delta\mathbf{z} + \Delta\mathbf{z}^T J^T J \Delta\mathbf{z}. \tag{30}$$

If $\mathbf{z}^*$ is a local minimum, then this expression further reduces into

$$g(\mathbf{z}) \approx \mathbf{r}_0^T\mathbf{r}_0 + \Delta\mathbf{z}^T J^T J \Delta\mathbf{z}, \tag{31}$$

which is a paraboloid centered at $\mathbf{z}^*$.

Another thing which one needs to notice is that every camera matrix is a 3×4 matrix, i.e. 12 values. But, since we have orthogonal constraint on part of the matrix the number of degrees of freedoms are actually just 6, 3 from translation and 3 from rotation. We can make a local parameterization of the orthogonal part using the fact that the exponent of a skew-symmetric matrix is orthogonal. This means that the size of $\mathbf{z}$ is actually,

$$\mathbf{z} \in \mathbb{R}^{6m+3n}, \tag{32}$$

where m is the number of cameras and n is the number of 3D points.

4.6 Jacobian Eigenvectors

We now want to study the eigenvalues and vectors of the matrix J^TJ. The first thing to notice is that there exist 7 eigenvalues which are 0. This is because the residual vector does not change if the map is scaled, rotated or translated. Secondly, the eigenvectors which have large eigenvalues corresponds to directions in which $g(\mathbf{z})$ increases quickly. In other words, directions in which we are more confident the map is correct. In the same way, the directions with small eigenvalues corresponds to small increases in $g(\mathbf{z})$. In Section 8, when we have some extra information on what the map should look like, these are the directions in which we should try to modify the map.

Another way of phrasing this is with the covariance matrix

$$C = \sigma^2(J^TJ)^{-1}, \tag{33}$$

where

$$\sigma^2 = \frac{\mathbf{r}_0^T\mathbf{r}_0}{a-b}, \tag{34}$$

where $a \times b$ is the size of J. We divide by $a-b$ since we have a residuals and b parameters. The directions where C has large eigenvalues corresponds to low uncertainty in our model and small eigenvalues corresponds to high certainty. Note that since we get the same residuals if the map is scaled, translated or rotated, the eigenvalues are infinite in these directions. Because of this, we cannot invert the matrix in equation 33 until we have removed these dimensions from the space.

The reader might find it helpful here with a slight pause to think about what a change along a direction in $\mathbb{R}^{6m+3n}$ means. An example which is easier to visualize is to just think of two points in 2D space:

$$x = (x_1, x_2), y = (y_1, y_2). \tag{35}$$

We now collect both of these into a vector $\mathbf{z} \in \mathbb{R}^4$:

$$\mathbf{z} = \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix}. \tag{36}$$

What direction in $\mathbb{R}^4$ corresponds to is moving each point along some separate line, see Figure 15. Similarly a change in $\mathbb{R}^{6m+3n}$ corresponds to moving every point and camera, every degree of freedom some separate amount. This will be discussed further in Section 8.

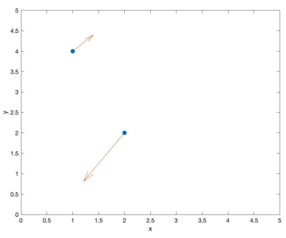


Figure 15: The points corresponds to the vector $\mathbf{z} = \begin{bmatrix}1 & 4 & 2 & 2\end{bmatrix}^T$ and the arrows indicate the direction $\Delta\mathbf{z} = \begin{bmatrix}1 & 1 & -2 & -3\end{bmatrix}^T$. Notice that each point can move a separate distance and direction. If the change is scaled into $2\Delta\mathbf{z}$ both the point move twice the current change.

5 The Absolute Orientation Problem

When solving SfM, the resulting map tells us what the relative position of all the features we have detected in 3D space. We choose the position of the origin and the orientation of the coordinate system, usually based on where the first image in the sequence is taken from. We also cannot know the scale of the map without a reference. This is because an image taken twice as far away of something twice the size looks the same. What this means when studying two different maps of the same object is that they differ by: translation, rotation and scale. In other words, the two maps differ by a similarity transformation, example in Figure 16. If we want to map the first map into the second map we are solving a minimization problem on the form:

$$\min_{s,R,t} \left| \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix} X - Y \right|, \tag{37}$$

where X and Y are the homogeneous coordinates in the first and second map and R is an orthogonal matrix, $R^T R = I$. This is a point set registration problem, the solution of which have already been demonstrated in [17] . A brief summary of the solution would be:

1. Translation is the difference between the two centers of mass of the two point clouds.
2. Scale is the ratio between the sum of the norms.
3. Rotation is found by UV^T from the singular value decomposition of

   $$M = \sum_{i=1}^{n} = (\mathbf{r}_l)'_i (\mathbf{r}'_r)^T_i,$$

   where $\mathbf{r}'_l$ ($\mathbf{r}'_r$) is the position of the points in the first (second) map after adjusting for scale and translation.

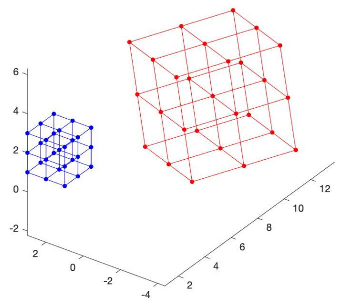


Figure 16: The red cube is the result of the blue cube after a similarity transformation, i.e. they differ by translation, rotation and scale.

6 Loop Closure

The usual practice when solving SfM is to only compare images which are close after each other in the image sequence. The reason we need to create separate bundles is that performing SfM over all the pictures would be too computationally costly. There are multiple reasons why the bundles are chosen based on the sequence. One reason is that images taken right after each other likely have a large overlap of 3D points.

This however means that if we have images from a loop and come back to the same place later in the sequence we do not always realize that we are viewing the same points again, see Figure 17. Because of small errors which accumulate during the loop there might be quite a distance between the two versions of the same point.

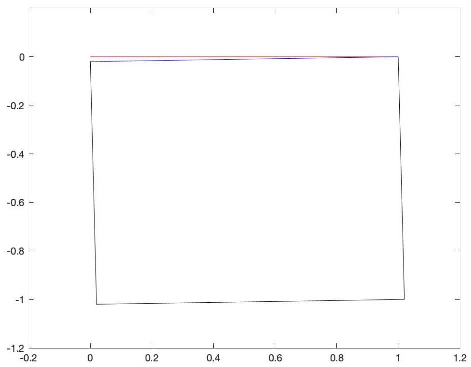


Figure 17: Because of small errors in earlier estimates the blue and red wall at the top do not perfectly align.

We know that some of the features will be the detected on both the red and blue walls. If we can find these pairs of points we should be able to improve our map by enforcing the points to have the same coordinates. Using Figure 17 as an example, we would add the condition that the red and blue line has to be on top of each other. Adding this constraint would hopefully make our estimation of the entire square more accurate.

6.1 Loop Closure in Practice

In the first map of the office dataset, see appendix A.1, one of the walls has been duplicated. However, because there are a lot of points, identifying the split by just looking at the map is quite difficult. In order to see the split we will first look at another metric. In the office dataset we have two maps. A first step in merging these maps would be to just do it with a similarity transform, see Section 5. This results in the maps shown in Figure 18.

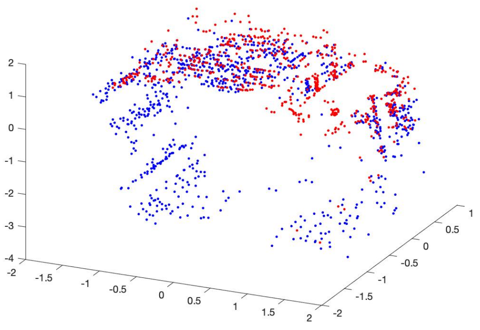


Figure 18: The result after transforming the second map (red) into the first map's (blue) coordinate system using a similarity transform.

After the similarity transform, for every point in map 1 we compare it with every point from map 2 and compute the geometric distance between the two points, see Figure 19.

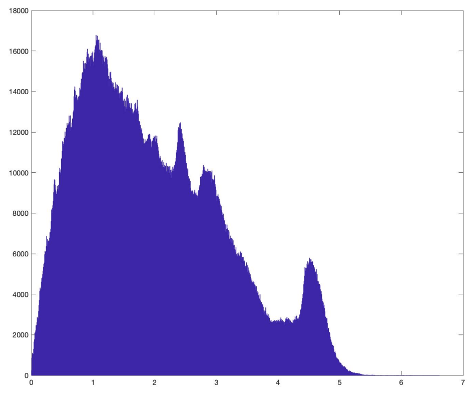


Figure 19: Histogram of geometrical distance between pairs of points in the first and second map of the office dataset.

In addition to computing the geometric distance we also compute the BRIEF distance. What we mean by BRIEF distance is the number of places where the two descriptors have different bits. Computing this distance for all pairs of points gives us the histogram shown in Figure 20.

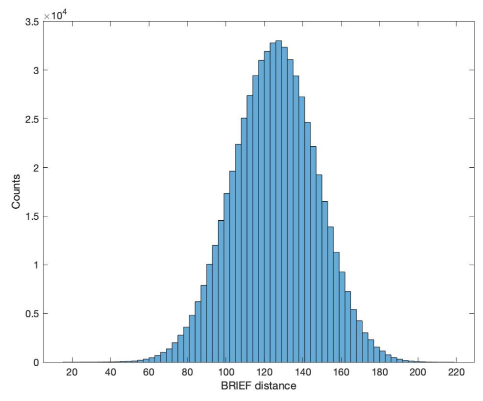


Figure 20: Histogram for BRIEF distance between pairs of points in the first and second map of the office dataset.

One thing we also need to consider is that each 3D point corresponds to multiple 2D points, each of which have their own BRIEF descriptor. This was dealt with by giving each 3D point a single descriptor which was the mode of all its descriptors.

If a pair of points is actually the same point we expect them to have a small BRIEF distance, and we also expect them to have a relatively small geometrical distance. Plotting the geometrical distance against the BRIEF distance for all the pairs of points gives us Figure 21. Looking at the bottom left of the figure we see good candidates for pairs which might actually be the same point. Pairs close to the region of interest were labeled by hand. The result of this labeling can be seen in Figure 22.

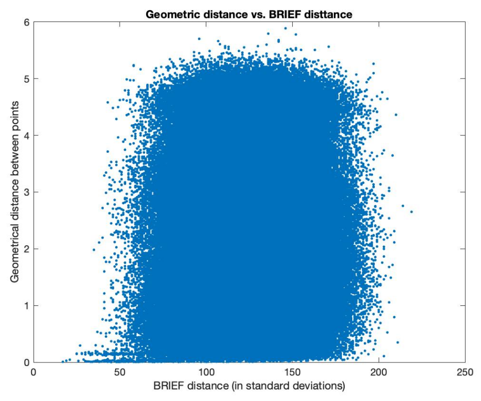
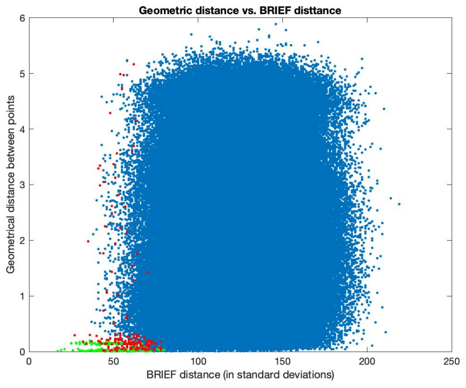
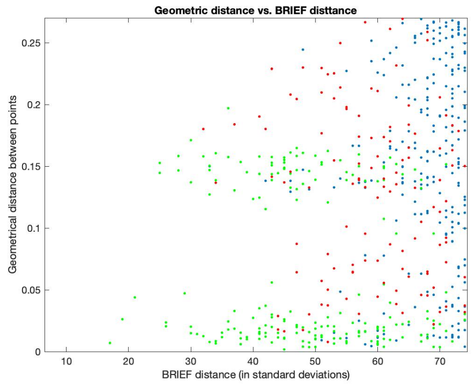


Figure 21: Geometrical distance plotted against BRIEF distance. Notice that we have a lot more pairs with similar BRIEF descriptors when the geometrical distance is small (bottom left of figure).



(a) Original plot



(b) Zoomed

Figure 22: Geometrical distance plotted against BRIEF distance. Labeled pairs are plotted in green if the pair is the same point and red if they are different. Unlabeled pairs are blue. Notice that there are two clusters of correct pairs, ones being just a few cm wrong while the other one is around 15 cm wrong.

If we see many pairs which have similar errors and the error is larger than a few centimeters, we should suspect a problem with a wall being duplicated. The error which we see a lot of should be the accumulated error. This seems very similar to what we see in Figure 22b. There is a cluster of correct pairs which have a geometrical distance of 15 cm. This corresponds to the duplicated wall having been shifted 15 cm, see Figure 23 for a visual example.

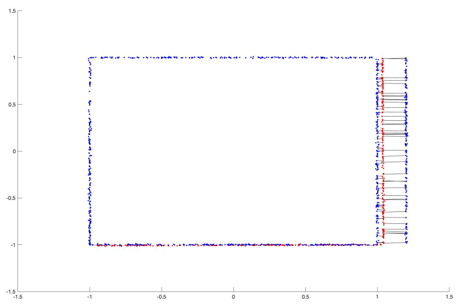
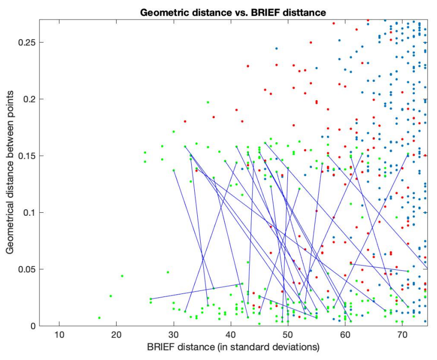


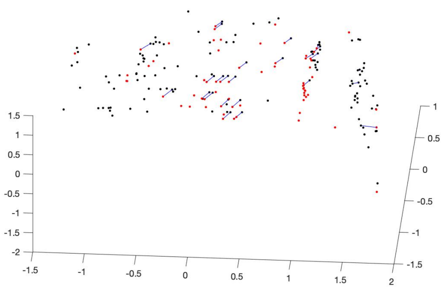
Figure 23: In this figure there is a black line between points in the first map (blue) and the second map (red) which have small BRIEF distances. Each of the black lines corresponds to a green dot in Figure 22. Because of the duplication of the right wall there is a collection of long black lines which have a similar length. Notice that this is not the office dataset but just an example.

Each of the pairs are made of one 3D point from the first map and one from the second. In some cases a single 3D point from the second map corresponds to multiple points in the first map. One of the more probable cases where this occurs is when the two points in the first map are on different versions of the same wall. We can use this relation to connect two dots in Figure 22. If we draw a line between each of these dots which have the same 3D point in the second map we get Figure 24a. Notice that most of the lines are drawn between the two clusters, this matches with the description above. Another way of showing this is to pair up the points in the first map using the above described method. We can then plot all the points from the first map in 3D space and draw a vector between the points which match with the same point in the second map, see Figure 24b. As we can see many of these vectors are similar, which matches with the accumulated error.

Because of Figure 24a we now see the problem. We also have an indicator for when the problem is solved. This is when the cluster of correctly labeled points with a geometrical distance of 15 cm has a much lower geometrical distance.



(a) Lines between matching pairs



(b) Points in 3D

Figure 24: In the left figure lines are drawn between dots which reference the same point in the second map. Notice that most lines goes between the two clusters. In figure on the right, the points are plotted in 3D space and all are from the first map. The points marked in black are from the bottom cluster on the left and the red from the top cluster. The cutoff for the separation is geometrical distance < 0.1. Notice that most of the errors (lines connecting red and black points) are not just the same length but also the same vector, this seems likely to be the accumulated error.

7 Method

The goal is to find pairs of points which are actually the same point in order to solve the loop closure problem. Given one map which has a duplication problem, this thesis proposes the following algorithm as a solution:

1. Find candidate pairs of points which should be the same point.
2. Using RANSAC, combine together the candidate pairs while increasing the reprojection errors as little as possible.

7.1 Finding Candidate Pairs

Pairs which are actually the same point, likely have similar BRIEF descriptors. One approach to finding candidate pairs is to compute the BRIEF distance for all pairs of points in the map. We can then select the pairs which are the most similar, i.e. have a short BRIEF distance, see Figure 25.

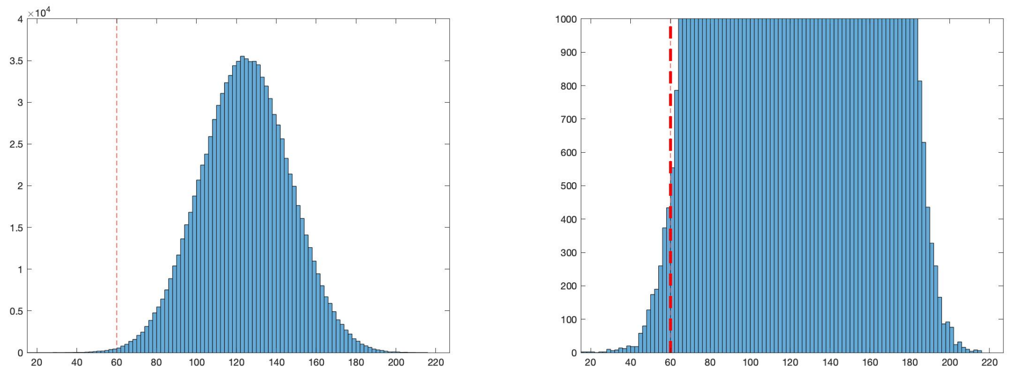


Figure 25: Histogram showing the BRIEF distance between all pairs of points. A cutoff at 60 errors for the filter is also shown in the figure.

The main drawback with this approach is that we have to compute the distance between all pairs of descriptors. This makes it slow with a time-complexity of $O(n^2)$. Because of this, the method used in this thesis is: for each point, find the point which has the shortest BRIEF distance to it. If two points have each other as their closest neighbor, consider them a candidate pair. Finding the nearest neighbor is a faster method and instead runs in $O(n \log(n))$, see [18].

7.2 Methods for Deforming the Map

We now want to deform the map to combine some of the candidate pairs. Since we want to increase the reprojection errors as little as possible, the map is deformed along the directions in which $J^T J$ has the smallest non-zero eigenvalues, see Section 4.6. There will be many incorrect pairs among the selected candidate pairs. In order to find the correct solution among all the noise, a RANSAC-loop is used. For the efficiency of the RANSAC-loop, we want to select as small a subset from the candidate pairs as possible. It is this subset to which we are fitting our model. The smallest subset we can select is a single candidate pair. Two different models were used to combine this pair in this thesis.

7.2.1 Deformation in Three Directions

If we select one pair which we want to force together as our subset, we need a linear combination of at least three eigenvectors for one and only one solution to exist, i.e:

$$\begin{cases} \tilde{U} = U + b_1B_1 + b_2B_2 + b_3B_3 \\ \tilde{U}_{p_1} = \tilde{U}_{p_2} \end{cases}, \tag{38}$$

where $\tilde{U}$ is the map after the deformation, $\tilde{U}_{p_1}$ refers to the coordinates of point p_1, and p_1, p_2 are the two points making up the pair which we want to force together, B_i is the i:th smallest eigenvector and b_i is how far we need to go in the B_i direction. Note that to solve the equation above we only need to use the part of U and B_i that correspond to the points p_1 and p_2. Both U and B_i are known and we solve for b_i. We can then deform the entire map according to,

$$\tilde{U} = U + b_1B_1 + b_2B_2 + b_3B_3. \tag{39}$$

We then want to compute which of the candidate pairs are consenting with this model. For each pair in the candidate pair we compute the distance between the two points in the pair. If the distance is less than some threshold ≈ 2 cm, we consider it agreeing with the model.

A problem with this method is that it has no prior on choosing a solution which is close to $b_i = 0$. When making the assumption $r = r_0 + J_0\Delta z$ we are assuming we will only alter the solution slightly. Because of this we should prefer solutions which make smaller changes to the map. One way of fixing this is if a solution in the RANSAC-loop resulted in an increase of the reprojection errors which were greater than some tolerance, then the solution was considered invalid and had 0 consenting points. In other words we introduced the constraint

$$\sum_{i=1}^{3} \lambda_i b_i^2 < \text{tol}, \tag{40}$$

where λ_i is the i:th smallest eigenvalue. However, a good method for selecting this tolerance was not found. Because of this, the method presented in the next section seems preferable.

7.2.2 Deformation in Many Directions

When combining a single pair, three eigenvectors are usually needed to get a single solution. The idea with only allowing changes along the three eigenvectors with the smallest eigenvalues was to only have a small increase of the reprojection error. We can likely improve on the previous method by allowing changes along all eigenvectors. Using many eigenvectors will give us many solutions. We could then choose the solution which minimizes the reprojection error. The formulation of this problem is:

$$\begin{cases} \min\limits_{b} \quad b^T\Lambda b \\ \text{s.t} \quad \tilde{U}_{p_1} = \tilde{U}_{p_2} \end{cases}, \tag{41}$$

where b is how much of each eigenvector B we use, i.e

$$\tilde{U} = U + \sum_i b_iB_i \quad , \tag{42}$$

Λ is a diagonal matrix containing the eigenvalues and $\tilde{U}_{p_1}$, $\tilde{U}_{p_2}$ are the coordinates of the chosen pair. For simpler notation, we write $\beta_i = B_{i,p_1} - B_{i,p_2}$, i.e. the relative movement between the points in the chosen pair when moving in the direction of the i:th eigenvector. To solve this we first find a solution to the constraint among the first three eigenvectors,

$$b^* = \begin{bmatrix} b_1^* & b_2^* & b_3^* & 0 & \dots & 0 \end{bmatrix}^T, \tag{43}$$

which is the same solution we found in the previous section before the constraints were used. Changing coordinates to

$$a = b + b^*. \tag{44}$$

We can now rewrite the constraint as

$$\sum_i a_i \beta_i = 0, \tag{45}$$

or on matrix form

$$\beta a = 0, \tag{46}$$

where

$$\beta = \begin{bmatrix} \beta_1 & \dots & \beta_k \end{bmatrix}. \tag{47}$$

This means we are looking for an a which is in the nullspace of β. We can therefore change coordinates again to

$$a = Hc, \tag{48}$$

where the columns of H is a basis in the nullspace of β. Our optimization problem is then

$$\min_c \quad c^T H^T \lambda H c + 2 b_s^* \Lambda_s H_s c, \tag{49}$$

where the subscript s refers to only the first three eigenvectors i.e. b_s^* is the first three element of b^*, Λ_s is a diagonal matrix containing the first three eigenvalues and H_s contains only the first three rows of H. The solution to this minimization problem is given by

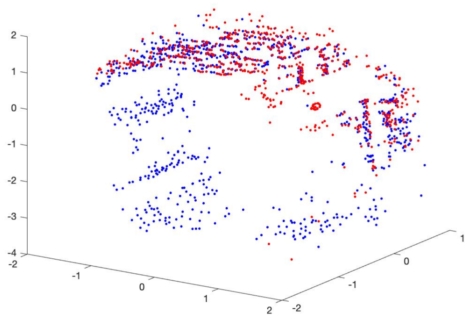
$$b = -H(H^T \Lambda H)^{-1}(H_s^T \Lambda_s b_s^*) - b^*. \tag{50}$$

Since we do not know which initial pair is good, RANSAC is used. However, if we optimize with all eigenvectors we need to invert $H^T \Lambda H$, the size of which scales with the number of eigenvectors. For the first map in the office dataset, the number of eigenvectors is 3600. Which makes $H^T \Lambda H$ too slow to invert. on every iteration. However, since the factors b_i depends on the size of the corresponding eigenvalue, we can use something like the smallest 100 eigenvectors, knowing that contribution of the rest of the eigenvectors is small.

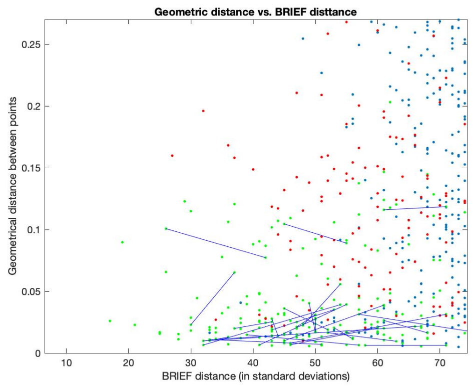
8 Results & Discussion

8.1 Office Results

The two methods that were presented in the previous section were used on the first map of the office dataset. The resulting modified maps were then aligned with the second map of the office data set with a similarity transform. The results using the method with three eigenvectors are shown in Figure 26. The results using the method with many eigenvectors are shown in Figure 27. 89 pairs were verified by hand to be the same point. One way of evaluating the methods is to look at the distance between the two points in the verified pairs, see Figure 28. All of the pairs the methods returned were checked by hand, this result can be seen in Table 1.

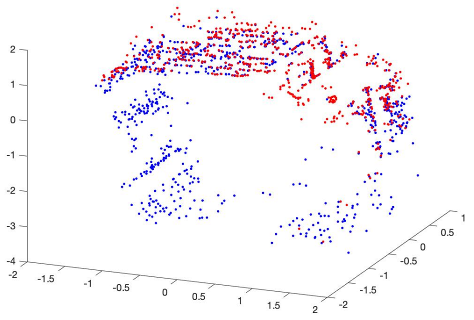


(a) 3D plot of the modified first map (blue) together with the second map (red). The maps were merged using a similarity transform.

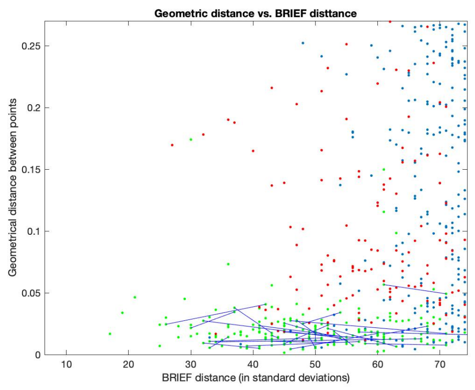


(b) The same figure as Figure 24 but using the modified version of the first map for geometrical distance.

Figure 26: Result for the method with three eigenvalues.

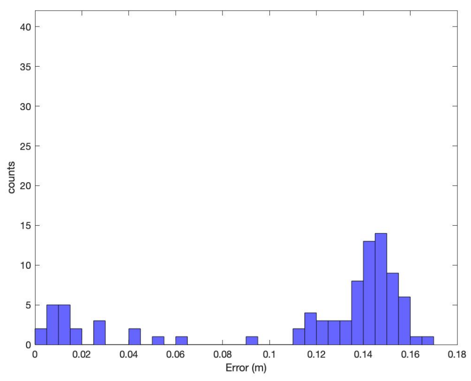


(a) 3D plot of the modified first map (blue) together with the second map (red). The maps were merged using a similarity transform.

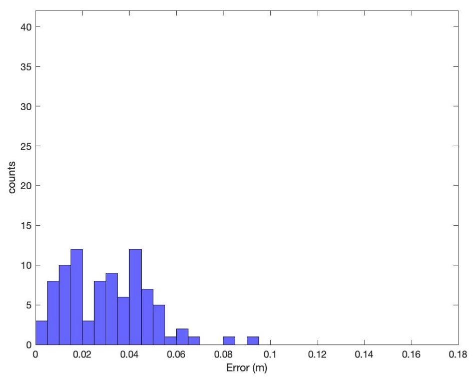


(b) The same figure as Figure 24 but using the modified version of the first map for geometrical distance.

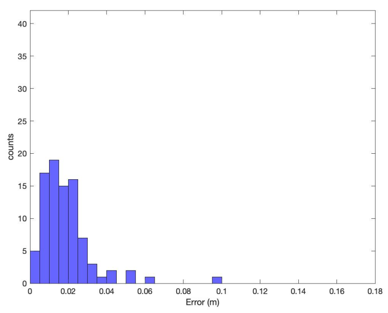
Figure 27: Result for the method with all eigenvalues.



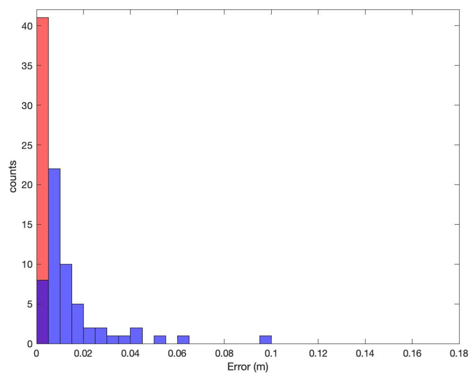
(a) Before modifications



(b) 3 vectors method



(c) Many vectors method



(d) Many vectors method with [2].

Figure 28: Distance between points in the first map which are known to be the same point. In the last subfigure, the pairs found from the many vectors method is combined with the algorithm suggested in Flood et al. [2]. The algorithm forces together the pairs found by the many vector method which makes their residual zero. Comparing these residuals with the residuals from the other methods gives a misleading comparison. Because of this, the residuals which were forced to zero are marked with red in the last subfigure.

	3-method	many-method
Number of corrected pairs found	28	33
Percentage of predicted pairs which are correct	87 %	100 %

Table 1: Number of corrected pairs found and what percentage of pairs found which were correct for the two methods.

8.2 Office Dataset Discussion

In both Figure 26 and 27 we see that there no longer is a collection of correct points with a geometrical distance error of 15 cm, such as was the case in Figure 24. Instead, most of the correct points have the same small error. This indicates that the loop closure problem is solved.

As can be seen in Figure 28, the errors in the corrected pairs are improved by both methods. However, the errors are smaller when using all the eigenvectors. Because of this, the suggested pairs from the many vectors method were used as input to the algorithm suggested by Flood et al.[2]. As can be seen in Figure 28d, the errors are further reduced even when disregarding the pairs which are forced together.

Figures 26-28 indicates that the method finds a good solution. However, it is important to remember that the main goal with this algorithm is to find correct pairs. The actual deformation is supposed to be done using [2]. Because of this, the main thing to consider when evaluating the methods should be: "How many corrected point does the method find?" and "What percentage of found points are correct?". Through human verification both of these question were answered for the two methods, see Table 1. This method of evaluation also seems to favor the many eigenvector method.

Because of verification by hand, we know that at least 89 pairs of duplicated points exists. However, the many eigenvector method only found 33 of them. Knowing that the tolerance used for how close together two points in a candidate pair needed to be combined with was 2 cm, we strangely see more than the 33 reported pairs in Figure 28c. It turns out that there are 88 pairs with a residual error less than 2 cm. Why these pairs are not reported is because they are not in the initial candidate set. Because of this there probably is an improvement to be made to how the candidate pairs are selected.

8.3 Tree Dataset

Considering that the office dataset was used continuously while developing the methods, it seems prudent to also evaluate the methods on another dataset. The tree data was only used after the methods were developed and tolerances chosen. The data was collected by filming the ground outside the Centre for Mathematical Science in Lund while walking in a circle around a tree. More about the tree dataset can be found in appendix A.2. Because the ground is more self-similar compared to the office, we expect this to be a more difficult dataset. Instead of a wall being duplicated there is a noticeable height difference between the start and end of the walk, see Figure 34.

Using the method with many eigenvectors resulted in the pairs shown in Figure 29 and resulting map shown in Figure 30. As we can see, the method identified and corrected the split. Notice that the method also found plenty of points which already were on top of each other in the original map. This seems to point at a problem with the original system. Points very close to each other which look very similar should probably be the same point.

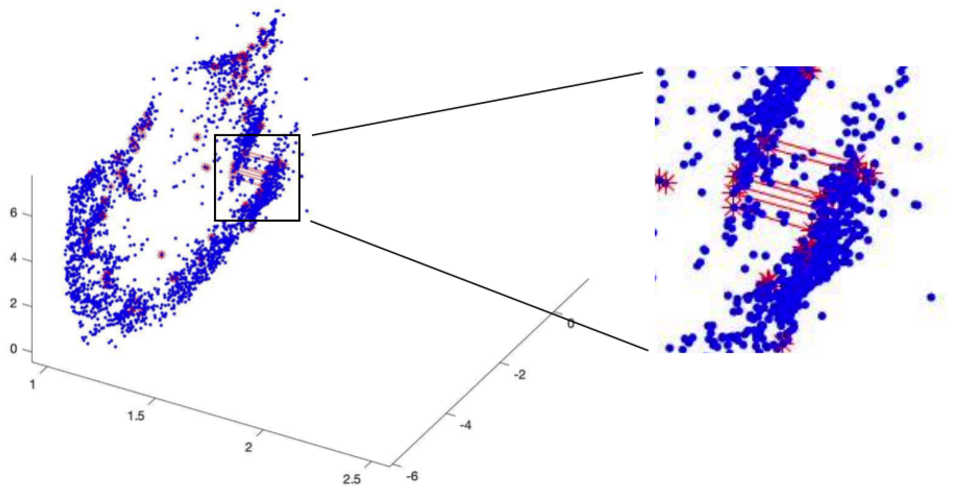


Figure 29: 3D plot showing the original map in the tree dataset. Proposed matching points have a red line between them.

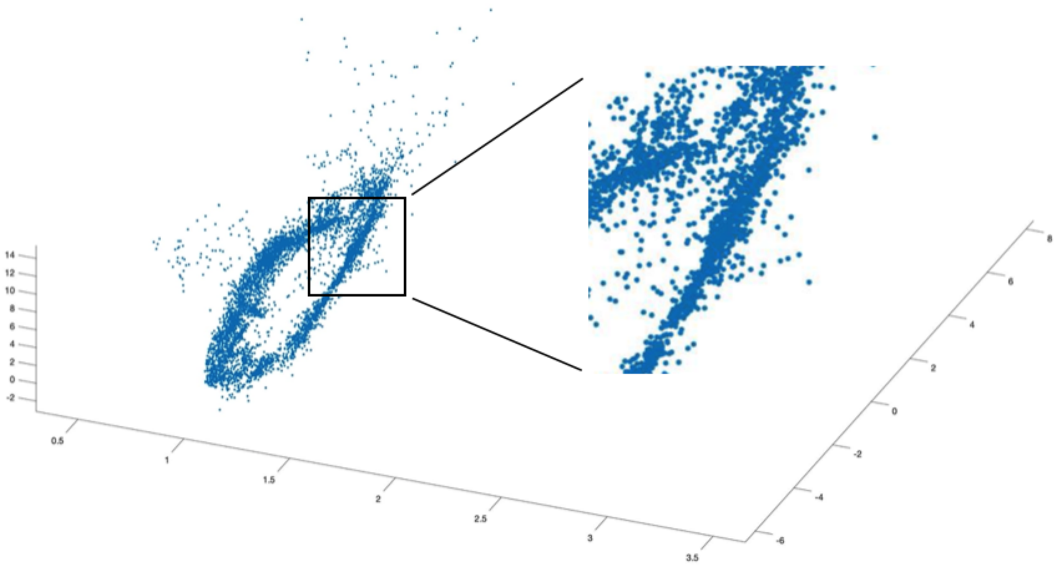


Figure 30: The resulting 3D plot after deforming the map with the method with many directions which resulted in the pairs shown in Figure 29.

8.4 Future Work

There are some areas which I would have liked to spend more time on. The main extension of this work would be to find matching points when combining multiple maps. As an example, imaging two 3D points from the first map of the office set and one from the second map are all three actually the same point. There should be a way to allow small changes along the smallest eigenvectors for the two maps while also allowing an arbitrary similarity transform to combine the two maps. Solving this problem together with generalizing the solution to an arbitrary number of maps is something I would have liked to spend more time on as a next step.

In Section 7.2.1, there exists a problem with quantifying what the meaning of a small change is. Similarly, there is also a problem with selecting what tolerance to use to decide when a candidate pair has converged. The problem with selecting a fixed tolerance is that scaling down all distances, which does not change the reprojection errors, will result in all points being inside of this tolerance. On the other hand, if the tolerance is scaled based on the average distance between the points and the origin, then we can also get all the pairs to converge by sending a few points far away from the rest in order to increase the tolerance. It would be interesting to study the possibility of preventing rescaling of the map by using IMU data.

The solution used in the three eigenvector method is to limit how far from the previous solution the new solution can be. No real robust solution for this is used in the many eigenvector method. However, most of these fake solutions have a much larger reprojection error. This makes them unlikely to be selected because we are computing the minimal increase of reprojection error.

For my algorithm to be usable in a larger system, it would need to be able to automatically decide if it found a good solution or not, similar to what is done in [2]. If the current method was used on a map with no loop closure errors or other duplicated points, it would still say that some pairs should be forced together, which of course is bad. The solution might be to run this algorithm to get pairs and then run the merging algorithm together with the merging verifier presented in [2]. This is something I would have liked to study if I had more time.

Another thing I would have liked to study further is the speed of the algorithm. The problem we are trying to solve occurs because we are trying to save time by not doing a full bundle optimization. It therefore is important that we actually save time with this method. I strongly believe this method should be much faster, but it would be nice to have it quantified.

A Datasets

A.1 Office

The main dataset used in this thesis is referred as the office dataset. It contains two maps and their specifications can be seen in Table 2. The first one contains all four walls but one of the walls is duplicated, see Figure 31. The second one contains two walls, one of which corresponds to the duplicated wall in the first map, see Figure 32. Usually, scale is unknown when performing SfM. In this case however, IMU data was used when computing the maps. What this means is that the scale is determined by using a reference such as earth's gravity.

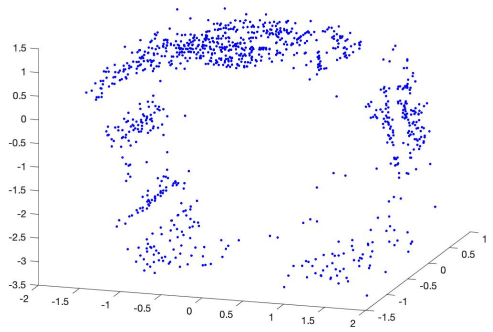


Figure 31: The first map in the office dataset. It contains 999 points spread over four walls, one of which is duplicated.

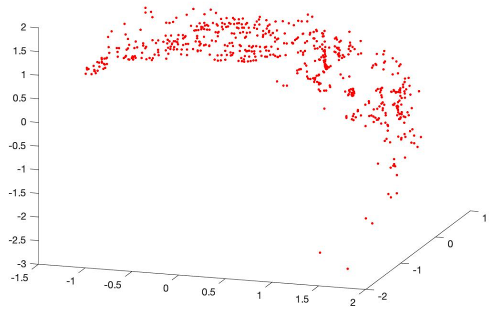


Figure 32: The second map in the office dataset. It contains 603 points spread over two walls.

Map	3D points	Cameras	2D features
1	999	104	9459
2	603	57	5986

Table 2: Description over the two maps in the office dataset.

A.2 Tree

The second dataset used in this thesis is referred as the tree dataset, see Figure 33 and Table 3. It only contains one map which was collected by walking one turn around a tree. During the walk around the tree an error was accumulated which caused the ground at the end and beginning of the loop to be at different heights, see Figure 34.

Map	3D points	Cameras	2D features
1	6013	379	68230

Table 3: Description over the two maps in the tree dataset.

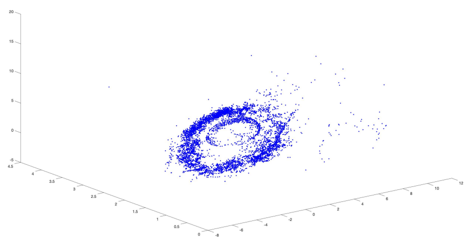


Figure 33: The tree dataset.

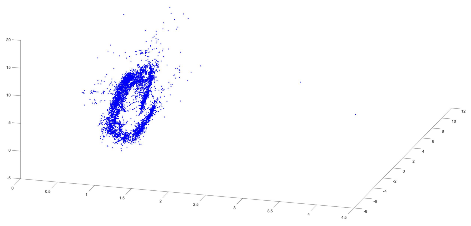


Figure 34: The turn around the tree resulted in a split which is shown in this figure. The split can be seen in the rightmost part of the ring of blue dots.

References

[1] Gabrielle Flood et al. “Efficient Merging of Maps and Detection of Changes”. In: *Scandinavian Conference on Image Analysis*. Springer. 2019, pp. 348–360.

[2] Gabrielle Flood et al. “Generic Merging of Structure from Motion Maps with a Low Memory Footprint”. In: *25:th International Conference on Pattern Recognition (ICPR) 2020*. 2021.

[3] Martin A Fischler and Robert C Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[4] Viktor Larsson. *Computational Methods for Computer Vision: Minimal Solvers and Convex Relaxations*. Lund University, 2018.

[5] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[6] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International conference on computer vision*. Ieee. 2011, pp. 2564–2571.

[7] Christopher G Harris, Mike Stephens, et al. “A combined corner and edge detector.” In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[8] Public Domain. *A diagram of a pinhole camera*. Feb. 2021. URL: `https://en.wikipedia.org/wiki/Pinhole_camera_model#/media/File:Pinhole-camera.svg`.

[9] Onur Ozyesil et al. “A survey of structure from motion”. In: *arXiv preprint arXiv:1701.08493* (2017).

[10] Christian Forster et al. “IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation”. In: Georgia Institute of Technology. 2015.

[11] George Vogiatzis and Carlos Hernández. “Video-based, real-time multi-view stereo”. In: *Image and Vision Computing* 29.7 (2011), pp. 434–441.

[12] Michael Kaess et al. “iSAM2: Incremental smoothing and mapping using the Bayes tree”. In: *The International Journal of Robotics Research* 31.2 (2012), pp. 216–235.

[13] Carl Olsson. *Computer Vision: Lecture 6" from "Computer Vision 2019*. Jan. 2021. URL: `http://www.maths.lth.se/matematiklth/personal/calle/datorseende19/notes/forelas6.pdf`.

[14] Olof Enqvist, Fredrik Kahl, and Carl Olsson. “Non-sequential structure from motion”. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE. 2011, pp. 264–271.

[15] Anders Eriksson et al. “Rotation averaging and strong duality”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 127–135.

[16] Niels Chr. Overgaard. *Föreläsning 8: Struktur och rörelse*. Jan. 2021. URL: `http://www2.maths.lth.se/matematiklth/vision/datorseende/f08.pdf`.

[17] Berthold KP Horn, Hugh M Hilden, and Shahriar Negahdaripour. “Closed-form solution of absolute orientation using orthonormal matrices”. In: *JOSA A* 5.7 (1988), pp. 1127–1135.

[18] Andrew William Moore. “Efficient memory-based learning for robot control”. In: (1990).

Master's Theses in Mathematical Sciences 2021:E3
ISSN 1404-6342

LUTFMA-3436-2021

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/