Global localization of nano drone in an indoor environment

Sofie Olsson

Master's thesis
2020:E70

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Global localization of nano drone in an indoor environment

(by building a point cloud map and implementing a computer vision based localization algorithm)

Sofie Olsson
`Sofieolsson30@gmail.com`

September 5, 2020

Master's thesis work carried out at Verisure Innovation AB.

Supervisors: Alexander Israelsson, `Alexander.israelsson@verisure.com`
Karl Åström, `karl.astrom@math.lth.se`
Patrik Persson, `patrik.persson@math.lth.se`

Examiner: Niels Christian Overgaard, `niels_christian.overgaard@math.lth.se`

Abstract

For a drone to be able to navigate in an indoor environment, it needs to understand its surroundings and locate itself to be able to plan a trajectory to its final destination. This thesis aims to solve the global localization problem, i.e. estimate a drone's position and orientation in a previously mapped indoor environment by using a monocular camera and computer vision, which is an important first step towards autonomous navigation

To make a drone able to understand its surroundings, a camera is attached to it and computer vision algorithms are used to extract important information about features in the environment represented in images. Using an open source software, COLMAP the features can be recreated in a map. A feature in the map is represented by a 3D-point and a descriptor, which describe the location and the structure of the feature in the world. Many points create together a point cloud. To be able to use the point cloud as a map for navigation, the scale ambiguity problem needs to be solved. Because of similarity properties of the projection model used in COLMAP, the point cloud can have arbitrary orientation and scale. A distance in the map can then be arbitrarily big, which makes it impossible to plan a trajectory. Therefore, the point cloud is rotated to match the orientation of the gravity direction and is scaled to metric scale by using sensor data from i.a. the drone's IMU. By extracting features from images when the drone is flying, descriptors representing the features can be computed and compared with descriptors in the point cloud map. Point correspondences are then generated between the map and image. They are later used to solve the Perspective-three-point problem to derive a pose estimate of the drone in the environment.

The results presented in this thesis indicates that it is possible to use this procedure when estimating a drone's pose in an indoor environment. The procedure was only tested for two simple data sets. The details of the procedure, which were possible to evaluate closer showed a stable result, however had room for improvements.

Keywords: Global localization, Crazyflie, indoor navigation, SLAM, point cloud map, monocular camera

Acknowledgements

First of all, I would like to thank my supervisors Alexander Israelsson from Verisure Innovation AB, Patrik Persson and Kalle Åström from the department of Mathematics at LTH. The results would not have been as good without your unwavering support. Alexander was invaluable through out this work. He was generous with his time, gave quick feedback and our discussions always helped me clarify ideas and concepts. Thank you Patrik, for sharing your experience, knowledge and insights about the hardware and computer vision. Your enthusiasm and dedication to the subject really made this project so much fun. Thank you Kalle, for sharing interesting ideas, bringing important discussions to the project and for valuable insight and feedback.

Special thanks to Verisure Innovation AB for giving me the opportunity to do this master thesis and for providing with necessary equipment. I also wish to extend my gratitude to the employees of Verisure for welcoming me and making me feel like a part of the team. Special thanks to Noroz Akhagi and Johannes Book, who helped order hardware parts and to finance this project.

I would like to thank Louise Tylstedt and Simon Ågren, students at Mechanical Engineering at LTH, who also worked on their thesis at Verisure. I appreciate all valuable discussions about our projects, which helped clarifying ideas. Special thanks to Louise, who taught me how to solder and special thanks to Simon for helping me to design and 3D-print the camera/video transmitter mount.

I also would like to thank Bitcraze for developing the Crazyflie platform and thanks to Kristoffer Richardsson for giving me guidance when choosing hardware for this project.

Finally, big thanks to Emma Sjögren and Carlos Rodríguez for proof reading this report and to Lovisa Svensson for giving important encouragements and support towards the end.

Contents

Acronyms

BA Bundle adjustment. 11

BRIEF Binary Robust Independent Elementary Features. 24

CPU Central processing unit. 25

csv Comma separated values. 30

DOF Degree of freedom. 27

EKF Extended Kalman filter. 29

FAST Features from accelerated segment test. 24

GPU Graphics Processing Unit. 11

HOG Histograms of oriented gradients. 24

IMU Inertial Measurement Unit. 1, 11, 13, 14, 27, 30, 34, 35

IQR Interquartile range. 23

MVS Multi-stereo-view. 31

ORB Oriented fast and rotated brief. 11, 24

P3P Perspective three point problem. 25

PC Point cloud coordinate system. 32

PCA Principal component analysis. 23

PTAM Parallel tracking and mapping. 11

RANSAC	Random sample consensus. 23
ROS	Robot operating system. 30
SfM	Structure from Motion. 31
SIFT	Scale invariant feature transform. 12, 24
SLAM	Simultaneous localization and mapping. 10–12
ToF	Time of flight. 27
TVL	TV Lines. 28

Chapter 1 Introduction

1.1 Background

In 2019, approximately 14 700 burglaries were reported in Sweden [6]. A home alarm system is a good way to protect people's homes, valuables, properties and for people to feel safe and secure in their own homes. Verisure is a security company located in Malmö, which is specialized in home alarm systems. Their alarm system consists of different sensor units, e.g. window magnets, movement detectors and cameras, which monitor the home to protect it from intruders.

If an alarm is triggered, it needs to be verified. This can be done by security officer call-outs or by specialized staff at an alarm centre who interpret and analyse, for instance, images from camera detectors. The former is very expensive and time consuming. The latter is cheap and fast, but comes with other difficulties and problems. To verify an alarm safely, cameras are needed in all rooms to cover every corner of the home. This makes installation and maintenance costs higher and can create a sense of discomfort by the home owner, because of the decrease of privacy.

An idea to a solution to this problem is to use a camera attached to a small drone. In case of an alarm, the drone can localize the triggered alarm unit and fly autonomously to its position and stream necessary information with the camera to an alarm centre for verification purpose. The drone has a landing platform, where it can charge while the alarm is inactive. This solution would not decrease the home owner's sense of privacy, would require minimal installation and verifies alarms fast and safe. However, it also comes with new technical challenges as indoor navigation, collision avoidance and flight and landing control.

For an autonomous vehicle, navigation is very important. Usually the problem is split into three sub problems; localization, mapping and path planning. To be able to navigate in an environment, a map is needed so that the drone can interpret its surroundings and determine its location (i.e. position and orientation) before it can plan a route to its final destination.

In this master thesis, the problem of global localization will be covered, which is the determination of the drone's position and orientation in a previously mapped environment. Both a map and a localization algorithm will be created and implemented.

1.2 Related work

To solve the global localization problem another similar problem will be studied, i.e. the simultaneous localization and mapping problem (SLAM), which will be thoroughly covered in the following section. The main difference between global localization and SLAM is that in global localization the map of the environment is created in advance, while in SLAM the map is built concurrently as the drone explores the environment. The reason why SLAM is studied is that the problem is well-covered. There are many different types of solutions to the SLAM-problem, from which ideas and inspiration can be collected to find a suitable solution to the specific requirements and limits of the home alarm application introduced in this thesis.

Also external localization systems like Loco Positioning system [4] was considered. Then anchors are placed in the room and are used to triangulate the position of the drone, like a mini GPS system. However, systems like this require more installation, since several anchors need to be placed at different positions in the home. Another disadvantage is the visual aspect of having several anchors attached to the walls. It is then a design issue.

1.2.1 Simultaneous localization and mapping

Simultaneous localization and mapping is the process of concurrently building a map of sensor data connected to landmarks of the environment and using this map to obtain estimates of the vehicle's position [11]. This problem is well known and studied in the field of autonomous robots and is considered one of the main problems when trying to build autonomous robots.

Localization and mapping are two problems that are tightly coupled. To be able to know a robot's location, a map of the environment is needed. To be able to build a map of the environment, the robot needs to know where it is located to interpret its surroundings correctly. This makes the problem difficult to solve and it also makes the solution sensitive to errors. If the estimated position has a small error, the map built from the found landmarks also has the same error. The estimation errors accumulate while the robot is moving in the environment and the robot's estimated position can drift away from its true position. This also makes it very difficult for the robot to understand if it has already visited a place once. This is the so called loop closure problem [11].

There are many different solutions to the SLAM problem. The structure of the solutions depend mainly on the application and which type of sensor is used. Since in this project, a camera will be attached to the drone to verify alarms, it can at the same time be used as sensor unit. Cameras are compact, accurate and can provide rich information about the environment which other sensors can not. By analysing a sequence of images, the drone can understand its direction of motion, orientation and position relative to landmarks in the environment. This makes it possible for the drone to recognize an already visited place. Cameras can also grant information about objects in the environment, which can be useful.

1.2.2 Visual-SLAM

Visual-SLAM methods are divided into direct and feature based methods. Direct methods use the whole image as input information to estimate the motion. Processing all pixels make these methods very computationally demanding and often a GPU is required. Feature based methods on the other hand only use a sparse set of distinct features in the images, such as points and lines, which in the world correspond to corners and edges. Therefore, they require less computational power. A feature extractor is required to find and to extract features in the images. The point correspondences are used to triangulate the points' positions and the camera pose. Some important feature based SLAM solutions are presented below.

FastSLAM [23] is a particle filter based solution. A state vector is used to keep track of camera poses and landmarks. The state vector and a covariance matrix are updated for every new image. If new landmarks are detected, these are added to the state vector. FastSLAM produces a solution which is efficient and robust. It can operate over an environment with thousands of landmarks, which makes it suitable for use in a small enclosed space. If the space is unlimited or very large, then the state vector gets too big for the method to operate in real time, because of the increase of computation time. The state vector representation makes it also difficult to detect whether a place has been visited or not (loop closure problem).

In solutions like FastSLAM, tracking and mapping are conducted simultaneously when updating the state vector. This makes mapping and tracking very tightly linked. Images in the image sequence can contain redundant information if the camera has little movement of none at all. Updating the state vector takes time and in this case it does not add any new information. This motivates a method to separate the tracking and mapping steps. In the article Parallel tracking and mapping (PTAM) [18] tracking and mapping were separated and run in two different parallel threads. This made it possible to use more robust methods for localization, like presented in MonoSLAM [8]. In this article, a single monocular camera was used to perform real time tracking in an unknown environment and by using a sparse feature map of the environment. In PTAM, also the problem of redundant information could be better dealt with by ignoring redundant images and focus instead on useful keyframes. Since not all images are used, the computational time constraint for the system to run in real time loosens. The processing of the keyframe needs to be completed before next keyframe is added. This makes it possible to create more detailed maps, which makes the tracking more accurate. All or a subset of the keyframes are later used in a last step to perform the computationally expensive but highly accurate batch method, bundle adjustment (BA). BA refines the map and camera poses by globally optimize over the parameters to get globally consistent estimates.

PTAM uses FAST-corners detector to detect features. Pixels in a small patch around the feature are used to describe the point.

Another important SLAM solution is ORB-SLAM [24], which is based on the procedure of PTAM. In ORB-SLAM also FAST-corners detector is used to detect features, but instead of using a patch to describe the feature, it uses a binary 256-bit descriptor. The descriptors are extremely fast to compute and it is very easy to compare them using the Hamming norm. Therefore, newly found descriptors can be compared to already found descriptors and loop closure is possible.

The most recent SLAM-system of interest is the Visual-inertial Monocular SLAM with map reuse [25]. The authors of the article present a method to fuse camera and IMU sensory data so that incremental motion get computed with a very high precision. This makes tracking more accurate. They also present a method to solve the scale ambiguity problem for monocular cameras, which is more thoroughly described below in section 1.2.3, using accelerometer data and the incremental motion estimates.

1.2.3 The scale ambiguity problem

A map which is created by a monocular SLAM system, is affected by the scale ambiguity problem. The projection model for a monocular camera is only defined up to a similarity transform. In other words, the camera poses and the 3D-points in the map can be arbitrarily scaled, rotated and translated. The 3D-points will still be projected to the same 2D-points in the image. A consequence is therefore, that the map will not be in metric scale nor have an orientation that relates to the gravity direction. The scale and orientation of the world can only be recovered if information about the world is added to the system. This can be done by introducing IMU sensory data, which was performed in Visual-inertial Monocular SLAM [25].

It is very important to solve the scale ambiguity problem in applications where there is some form of control and path planning. Otherwise the scale of the error is unknown. The trajectory in the map will not be the same as the trajectory in the world. It is also of interest to know the direction of gravity in the map, since the accelerometer is sometimes used to understand the vehicle's movement.

It is not possible to determine the orientation around the gravity vector (north, south, east, west) without additional sensors. Therefore, orientation has to be chosen arbitrarily, but must be kept fixed in the application to not disrupt the planned trajectory.

1.3 Main idea

As mentioned in the end of section 1.1, the global localization problem is the determination of the drone's position and orientation (pose) in a previously mapped environment. Since in a home alarm system application, the drone moves in the same space and the environment will most likely not change drastically between alarm activations, the map can be created in advance. This enables the map to be built without a real time constraint. Because of the time limit of the thesis, the software to create the map is not completely made from scratch. The 3D-points in the map are generated by using an open source software called COLMAP. COLMAP can reconstruct a sparse 3D-point cloud by extracting SIFT-features from a sequence of overlapping images (keyframes) from the environment. SIFT is a feature detector and extractor, which is very commonly used because of its scale and rotation invariancy. However, computing and matching SIFT-features can not be conducted as fast as when ORB is used. COLMAP performs the bundle adjustment step, which is very important to correct accumulated errors in the map. BA is then conducted without any risk of slowing down the determination of the location of the drone. Since only the localization of the drone will be conducted in real time, only the solution to this problem is under a real time constraint. By creating a sparse feature map instead of using a direct method, and by describing the 3D-points with ORB-descriptors instead of SIFT-descriptors, the computations required to determine the drone's pose decrease. This makes it easier to create a system which could work in real time.

The map consists of a cloud of 3D-points with ORB-descriptors. Every 3D-point correspond to a real point in the environment. The map is therefore called "point cloud map". However, before it can be used as a map, the scale ambiguity problem needs to be solved. Since a monocular camera will be used, the point cloud can have arbitrary scale and orientation, which makes it difficult to use as a map for path planning. The problem will be solved by integrating IMU and pose data when creating the point cloud map, which will be thoroughly described in section 3.2.2 and section 3.2.3.

Since the drone will rest on a landing platform when the alarm is inactive, the start position will be known as the platform will have a fix location.

To be able to localize the drone in the map, feature points from the images are compared to the descriptors in the map. This gives point correspondences between the map and the image. By solving the Perspective-three-point problem, a pose estimate can be derived.

1.4 Crazyflie

The thesis is carried out in cooperation with the company Bitcraze, who develops the drone Crazyflie. The Crazyflie is a nano sized drone that weighs only 27 grams. Its small size makes it safer for indoor usage. In case of a crash, the kinetic energy will not be high enough to cause large damage to people, pets or interior.

Bitcraze also develops a flow deck which can be attached to the Crazyflie. The flow deck consists of two sensor units: a time of fight sensor and an optical flow sensor. The inbuilt system of the drone can estimate the pose more accurately when these two sensors are added.

1.5 Limitations

There are some disadvantages when using a nano sized drone like Crazyflie. The disadvantages cause some limitations of the project. The small size of the drone limits its lifting capacity. Therefore, all components need to be light-weight. The small battery has a capacity of 240 mAh which needs to give power to both the Crazyflie and the camera system attached to the Crazyflie. This shortens the time of flight to about one minute. Since the camera needs to be so small, the quality of the images is limited.

When this master thesis is conducted, there is no possibility to implement a solution of the problem on-board of the Crazyflie. Therefore, all computations are made off-board on a computer which is connected to both camera system and the Crazyflie. A consequence of this is that a time delay occurs in the system.

Because of the project's time limit, only simulations of the system will be performed and evaluated. The system will not be tested in real time nor will the result of the localization algorithm be used to control the Crazyflie.

The system to collect data was built in a way that made it hard to fly the drone and at the same time collect IMU and pose data. Therefore the data was collected while holding the drone in the hand and moving it around.

1.6 Purpose and goals

The purpose of this thesis is to establish global localization of a drone in an indoor environment by achieving the following three goals and subgoals:

1. Build a system to collect data.
    - Choose hardware components that works with the restrictions Crazyflie entails.
    - Calibrate camera and camera-IMU system.
2. Create a point cloud map of the environment in a metric scale.
    - Use open source software COLMAP to create point cloud.
    - Rotate point cloud so that the direction of gravity points in the same direction as the negative z-axis.
    - Scale point cloud to metric scale.
    - Translate point cloud in z-direction so that the origin is located at floor level.
    - Find ORB-descriptors to 3D-points in point cloud.
3. Determine the drone's position and orientation in the point cloud map by using computer vision and by solving the Perspective-three-point problem.

For system evaluation, the following questions will be answered:

- Is it possible to implement a solution using the Crazyflie with the weight, battery time, and camera quality restrictions?
- How accurate is the estimation of the scale factor of the point cloud?
- Is it possible to pair ORB-descriptors with 3D-points, which COLMAP created by detecting SIFT-features?
- How accurate is the positioning of the drone? How noisy is it?
- How fast can the implemented solution perform in simulations? Is it fast enough to be used in real time in the future?

1.7 Contribution

The content of this thesis could contribute to a new product to the Verisure alarm systems. It contains a basic idea of how an autonomous drone can be used in a home alarm application. Specifically an idea of how the navigation of the drone could be implemented is presented.

A new idea is presented of how a flow deck attached to the Crazyflie could be used to easily obtain a more accurate scale factor when scaling the point cloud to metric scale.

This master thesis also contribute to the understanding if the descriptors in a point cloud map constructed by SIFT-features could be described by ORB-features instead.

1.8 Thesis outline

In the introducing chapter, the main idea and the important related work, which inspired to the procedure in this thesis, were presented. In chapter 2 the relevant theory needed to understand the procedure of the method and implementation is presented. In chapter 3 the method of how the three steps listed in Purpose and goals (section 1.6) are achieved. The section is divided into three subsections, which resemble the three steps. In chapter 4 the results are presented followed by chapter 5 where a discussion about the result is held. Lastly, in chapter 6 the conclusion of the thesis is presented along side with a Further research section about suggested further testing and improvements of the solution.

Chapter 2
Theory

In this chapter, the relevant theory is presented. First, the camera and distortion models are explained to derive the transformation between camera and image. Followed by the explanation of rigid body transformation, where different rotation representations are presented. After transformations have been thoroughly explained, a section about important statistical tools follows, which are used both in the implementation but also to analyse the results. Lastly a section about computer vision theories and algorithms are presented and the perspective-three-point problem is explained in the end.

2.1 Projection: from camera to image

2.1.1 Camera model - pinhole

To be able to express how points are projected from a three dimensional world to an image plane, a camera and image distortion model are needed. The simplest camera model is the pinhole camera model. It does not take into account geometric distortion. Therefore, the distortion has to be compensated for separately.

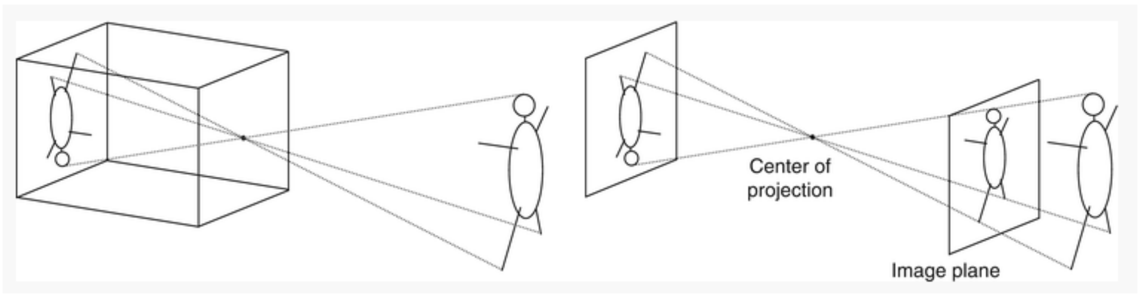


Figure 2.1: The basics of a pinhole camera model. Image from [36].

In the pinhole model, a camera is compared to a box with a small hole that lets little light through, see Figure 2.1. The rays of light that goes through the small hole hits the back of the box. If a film is placed in the back, objects can be projected onto that film and an image is created. The point, where the small hole is located is called center of projection or camera center. The projection reduces the dimensions from 3D to 2D and also information like distances and angles are lost. Because of the similarity property of triangles, the image plane can be put in front of the camera center in the model.

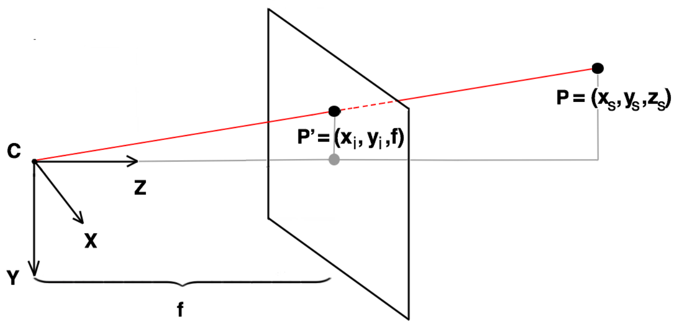


Figure 2.2: The figure shows the projection of a 3D-point, P, onto the image plane, P'. Here C is the center of projection and f is the focal length.

The origin of the coordinate system is put in the center of projection C. The focal length f, is the distance between the principal point (p_x, p_y) and center of projection. A point P is projected onto the image plane. The projected point is called P', see Figure 2.2. The similarity property of triangles gives the projection formula

$$x_i = f\frac{x_s}{z_s}, \tag{2.1}$$

$$y_i = f\frac{y_s}{z_s}. \tag{2.2}$$

Next step is to transform to pixel coordinates system. Defined as Figure 2.3 illustrates.

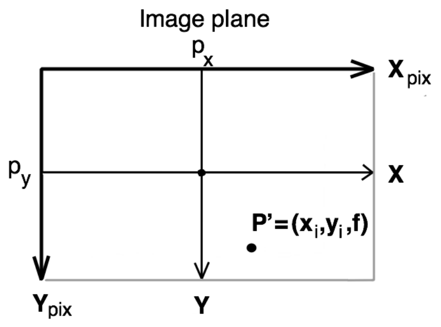


Figure 2.3: The point P' is expressed in the camera's reference frame. The x and y axes from the camera reference frame are projected onto the image plane to visualize the relation between them and the pixel coordinate system.

The pixel coordinates are,

$$x_{pix} = k_x x_i + p_x = \frac{f k_x x_s + z_s p_x}{z_s}, \tag{2.3}$$

$$y_{pix} = k_y y_i + p_y = \frac{f k_y y_s + z_s p_y}{z_s}, \tag{2.4}$$

where k_x and k_y are scale factors. By denoting $f_x = f k_x$ and $f_y = f k_y$, the transformation can be written in matrix form as

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & p_x & 0 \\ 0 & f_y & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{K} \begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix}. \tag{2.5}$$

By $x_{pix} = u/w$ and $y_{pix} = v/w$, the pixel coordinates are obtained.

2.1.2 Distortion model - fisheye

A point's pinhole projection is $P' = (a, b, 1)$. Defining

$$r^2 = a^2 + b^2, \tag{2.6}$$

$$\theta = \arctan r. \tag{2.7}$$

When fisheye distortion apply, then according to the article in [17],

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{2.8}$$

and the distorted projection coordinates are

$$x_{dist} = \frac{\theta_d}{r} a, \tag{2.9}$$

$$y_{dist} = \frac{\theta_d}{r} b, \tag{2.10}$$

which in pixel coordinates are

$$x_{pix} = \frac{f_x x_{dist} + z_s p_x}{z_s}, \tag{2.11}$$

$$y_{pix} = \frac{f_y y_{dist} + z_s p_y}{z_s}. \tag{2.12}$$

2.2 Rigid body transformation

A rigid body transformation is a geometric transformation of euclidean space, which preserves the euclidean distance between points. The transformation describes the relation between two points of reference (the origin) of two reference frames. It consists of a rotation and a translation. A point's coordinates in one frame can then easily be expressed in the other frame, which is illustrated in Figure 2.4.

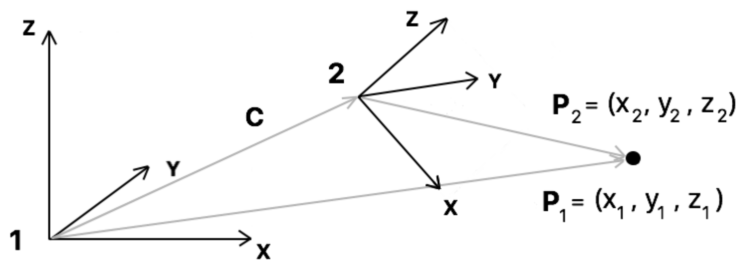


Figure 2.4: The figure shows two different coordinate systems. The point **P** can be expressed in both reference frames. To move between the two frames, a transformation is needed, describing the relative rotation and translation between them.

2.2.1 Rotations

There are several ways to describe a rotation, e.g. by a quaternion, a rotation matrix or a rotation vector. In this project, quaternions are used. In some software, other formalisms were used. Those rotations were converted to quaternions.

The advantage of using quaternions is that multiplications does not take that much computation time compare to matrix multiplication and it takes less space to store a 4 dimensional vector compare to a rotation matrix, which is defined by 9 numbers.

In the following section, the different formalisms will be described and how to convert to quaternion space.

Quaternion

Given two complex numbers $A = a + bi$ and $C = c + di$, a quaternion in quaternion space $\mathbb{H}$ can be constructed by $\mathbf{q} = A + Cj$ and by defining $k = ij$. A quaternion is a four-dimensional vector and is defined as (Cayley-Dickson construction)

$$\mathbf{q} = q_w + q_x i + q_y j + q_z k, \tag{2.13}$$

where $\{q_w, q_x, q_y, q_z\} \in \mathbb{R}$ and $\{i, j, k\}$ are three imaginary unit numbers defined so that

$$\begin{gathered} i^2 = j^2 = k^2 = ijk = -1, \\ ij = -ji = k, \qquad jk = -kj = i, \qquad ki = -ik = j. \end{gathered} \tag{2.14}$$

In this project the following operations and properties of the quaternion are used.
Multiply two quaternions

$$\mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} p_w q_w - p_x q_x - p_y q_y - p_z q_z \\ p_w q_x + p_x q_w + p_y q_z - p_z q_y \\ p_w q_y - p_x q_z + p_y q_w + p_z q_x \\ p_w q_z + p_x q_y - p_y q_x + p_z q_w \end{bmatrix}. \tag{2.15}$$

Quaternion conjugate is defined as

$$\mathbf{q}^* = q_w - q_x i - q_y j - q_z k. \tag{2.16}$$

The norm of a quaternion is defined as

$$||\mathbf{q}|| = \sqrt{\mathbf{q} \otimes \mathbf{q}^*} = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}. \tag{2.17}$$

Quaternion inverse is defined as

$$\mathbf{q}^{-1} = \frac{1}{||\mathbf{q}||}\mathbf{q}^*. \tag{2.18}$$

The unit quaternion, which has the property $||\mathbf{q}|| = 1$, is also called rotation quaternion and is used to represent the 3D rotation group.
Rotating a point x_1 by $\mathbf{q}$, where $\mathbf{q}$ is a rotation quaternion and x_1 is point extended from 3 dimensions to 4 dimensions by adding zero at the front. The rotated point x_2 is in 4 dimensions and can be transformed to 3D space by dropping the first element in the vector.

$$x_2 = \mathbf{q} \otimes x_1 \otimes \mathbf{q}^*. \tag{2.19}$$

Combining two rotations $\mathbf{q_1}$ and $\mathbf{q_2}$

$$\begin{aligned} x_2 &= \mathbf{q_1} \otimes x_1 \otimes \mathbf{q_1^*}, \\ x_3 &= \mathbf{q_2} \otimes x_2 \otimes \mathbf{q_2^*}, \\ x_3 &= (\mathbf{q_2} \otimes \mathbf{q_1}) \otimes x_1 \otimes (\mathbf{q_1^*} \otimes \mathbf{q_2^*}). \end{aligned} \tag{2.20}$$

Since multiplying two quaternions gives another quaternion (see 2.15),

$$\begin{aligned} \mathbf{q_3} &= \mathbf{q_2} \otimes \mathbf{q_1}, \\ \mathbf{q_3^*} &= (\mathbf{q_2} \otimes \mathbf{q_1})^* = \mathbf{q_1^*} \otimes \mathbf{q_2^*}, \\ x_3 &= \mathbf{q_3} \otimes x_1 \otimes \mathbf{q_3^*}. \end{aligned} \tag{2.21}$$

For lighter convention, $\otimes$ will be written $\cdot$ in sections below. It must not be mixed up with normal multiplication.

Rotation matrix

The rotation, defined by a rotation matrix R, is a rotation about the axes of a coordinate system. It is of dimension 3x3 and it can rotate a point by

$$x' = Rx. \tag{2.22}$$

The rotation matrix has the useful properties:

$$\begin{aligned} R^T &= R^{-1}, \\ \det(R) &= 1, \end{aligned} \tag{2.23}$$

which makes it easy and fast to check if the obtained rotation matrix is in fact a rotation matrix.

To convert a rotation matrix to a quaternion is a bit tedious, and the theory behind it will not be mentioned here. There are libraries in MATLAB and in python to do this, which are easy to use.

Rotation vector

A rotation vector is 3-dimensional vector where a rotation is expressed as one rotation around an axis. The vector is constructed by multiplying the angle with a unit vector,

$$\mathbf{r} = \theta\hat{\mathbf{e}} = \theta \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}. \tag{2.24}$$

A rotation vector can easily be converted to a quaternion by the following formulas

$$q_w = \cos\frac{\theta}{2}, \qquad q_x = \sin\frac{\theta}{2}e_x, \qquad q_y = \sin\frac{\theta}{2}e_y, \qquad q_z = \sin\frac{\theta}{2}e_z. \tag{2.25}$$

2.2.2 Translation

The translation vector **t** describes how the origin of the reference frame has moved compared to the other reference frame,

$$\mathbf{t} = [t_x \;\; t_y \;\; t_z]^T. \tag{2.26}$$

2.2.3 Transformation representation

The rotation and translation can be but together to describe the transformation from one coordinate system to another (see Figure 2.4). The general formula is

$$P_1 = RP_2 + \mathbf{t}. \tag{2.27}$$

where R is a rotation matrix, describing the rotation from reference frame 2 to 1 and **t** is the translation vector. The inverse transform is

$$P_2 = R^{-1}P_1 - R^{-1}\mathbf{t}. \tag{2.28}$$

When describing the relation between world and camera coordinate system, $C = -R^{-1}\mathbf{t}$ is the vector to the camera center (see figure 2.4).

When using a rotation matrix, the transformation in equation 2.27 can be assembled to a 4 by 4 transformation matrix, T, by expressing the rotation and translation as two 4 by 4 matrices, so called homogeneous coordinates

$$\begin{bmatrix} p_{1x} \\ p_{1y} \\ p_{1z} \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{2x} \\ p_{2y} \\ p_{2z} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{T} \begin{bmatrix} p_{2x} \\ p_{2y} \\ p_{2z} \\ 1 \end{bmatrix}. \tag{2.29}$$

If a quaternion is used to describe the rotation, then the transformation is

$$P_1 = \mathbf{q} \cdot P_2 \cdot \mathbf{q}^* + \mathbf{t}, \tag{2.30}$$

where the points P_1, P_2 and **t** are extended to 4 dimensions as described in equation 2.19.

2.3 Statistic tools

2.3.1 Interquartile range

The Interquartile range (IQR) is a statistic measure of variability in a data set.

Let S be an ordered set of measurements. The set is split into four equally big sets, called quartiles. The three values that separates the subsets are called first, second and third quartiles, denoted Q_1, Q_2 and Q_3. The IQR measure is defined to be

$$IQR = Q_3 - Q_1. \tag{2.31}$$

The IQR is used to detect outliers P, i.e.

$$P = \{x \in S \ : \ x < Q_1 - 1.5 \cdot IQR \text{ or } x > Q_3 + 1.5 \cdot IQR\}. \tag{2.32}$$

The IQR is computed for the data set. If there is an outlier, it is removed and the IQR is again computed with the new data set. This is repeated until there are no more detected outliers.

2.3.2 Principal component analysis

PCA is a method to project correlated data from a n-dimensional space to a space with lower dimension. By doing so, the interpretability of the data increases but at the same time minimal information is lost. The method creates new variables, which are linear combinations of the old ones. The variables are uncorrelated and that maximizes the variance in each dimension [16]. The first principal component is the dimension with the highest variance, the second principal component is the dimension with second highest variance and so on. The first principal component can be defined as the best fitting line to the data, e.i. the line that minimizes the average squared distance from a point to the line.

To compute the first principal component, the data needs to be centered. The mean of the data i subtracted from the data. Singular value decomposition, $X = USV^T$, is performed on the data X. Here U is an $m \times m$ unitary matrix, S is an $m \times n$ diagonal matrix and V is an $n \times n$ unitary matrix. The first vector in V, which corresponds to the largest singular value, is the direction of the desired “best fitting” line.

2.3.3 RANSAC

Random sample consensus (RANSAC), first presented in [10], is a iterative method to estimate the parameters in a mathematical model from a set of observed data, which contains a significant percentage of measurement errors. The method is non-deterministic. More iterations increases the probability of finding a correct model.

RANSAC uses the smallest possible subset of the data to estimate the parameters of the model. The subset is chosen randomly. The rest of the data is used to validate the model. An error tolerance is needed to determine if a data point is an inlier or an outlier. The process is repeated. The new estimated model is compared with the best model from the previous iterations and the one with the higher number of inliers is considered a better model. A threshold for the number of inliers can be used to determine if the correct model was found.

2.4 Computer Vision

Computer vision is an interdisciplinary field of how computers can gain high level understanding about the world by processing and analysing digital images. It is inspired by how the human vision system works and enables computers to do complex tasks like; image classification, scene reconstruction, object recognition, pose and motion estimation.

OpenCV [27] is an open source library with tools which are useful when implementing a computer vision application. The software is free and easy to use. Tools for i.a. feature detection, extraction and matching were used from this library.

2.4.1 Feature detection and extraction

For a computer to be able to understand which parts of an image contain the important information, algorithms that detect specific patterns are used, so called feature detectors. A feature in an image can be described visually as a point, a line or a corner. A detector identifies the interesting parts of the image and an extractor extracts the important information[19]. The interesting point's pixel coordinates in image is called keypoint. The point and an area around it is described by a descriptor, so that (ideally) the descriptor is invariant under changes in illumination, scale, rotation and translation [38]. The computer can later process the information in keypoints and descriptors to make decisions depending on what the computer "saw".

There exists different detectors and extractors. In this thesis SIFT and ORB will be mentioned and therefore only these will be described. Both descriptors are invariant to scale and rotation [37].

SIFT

SIFT stands for Scale invariant feature transform. It was first presented in [20]. SIFT is a very popular feature detector and extractor. It has been proven to be very useful in many fields of computer vision [31] and is one of the most accurate feature algorithms [37]. The SIFT-descriptor is based on histograms of oriented gradients (HOG). It is derived by computing gradients (orientation and magnitude) for each pixel in a small patch (16x16 pixels) around the keypoint. The patch is divided into four regions and a gradient direction histogram over 8 directions is computed for each region. The histograms are put together for all regions in the patch, leaving a 128 dimension vector [20]. SIFT is good for detecting blobs in an image, i.e. parts in the image where there is a difference in intensity or color between pixels [37].

ORB

ORB stands for Oriented FAST and Rotated BRIEF. It is feature detector and descriptor based on the FAST keypoint detector [30] and uses a binary descriptor based on BRIEF [7]. It was first presented in [31]. ORB has a similar matching performance and is less sensitive to image noise compared to SIFT [31]. ORB is very fast when both detecting and matching features. It is approximately ten times faster than SIFT [19], which makes it more suitable in a real time application.

The BRIEF descriptor (Binary Robust Independent Elementary Features) uses simple binary tests between pixels in the image patch where the feature is located to compute the descriptor. When used in ORB, an improved BRIEF is used, which is more stable in rotation [37]. ORB is good for detecting corners in an image [37], i.e. the intersection between two edges or a point.

2.4.2 Feature matching

Features sometimes need to be compared to know if two features correspond to the same real point. Often is matching based on the distance measures between two descriptors. If the distance is small, then the match is considered good. For ORB, the hamming distance is used, which is defined as the number of mismatching bits between two binary descriptors [5]. This means that comparing two descriptors is conducted very fast and a only a CPU is needed [7]. When matching string based descriptors like SIFT, L1 or L2 norm is used [37], which takes much more time than computing the hamming distance.

2.5 Perspective-Three-Point problem

The perspective-n-point problem is the problem of estimating the camera pose given a set of n correspondences between 3D-points in the world and 2D-points in an image. If $n = 3$, then the perspective-three-point problem (P3P) is obtained. It has been showed that three is the minimum number of required correspondences to get a finite number of pose estimates [15]. Solving P3P can generate up to four geometrically feasible solutions.

Let P be the center of projection and A, B and C are 3D-points with the 2D-point correspondences u, v and w. Further denote $p = 2\cos\alpha$, $q = 2\cos\beta$ and $r = 2\cos\gamma$. The equations for the triangles PBC, PAC and PAC form then the equation system

$$\begin{aligned} Y^2 + Z^2 - YZp - a'^2 &= 0 \\ Z^2 + X^2 - XZq - b'^2 &= 0 \\ X^2 + Y^2 - XYr - c'^2 &= 0. \end{aligned} \tag{2.33}$$

The solutions for X, Y and Z are the physical solutions to the problem. In Figure 2.5 the problem is visualized geometrically. A full geometric solution to the problem can be found at [15].

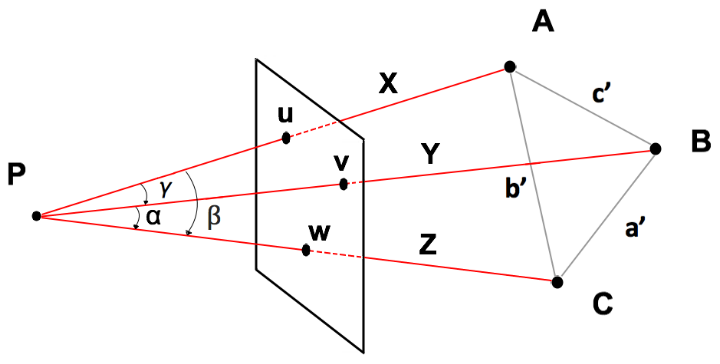


Figure 2.5: The P3P-problem.

Chapter 3
Method

In this chapter the method and implementation of the built system is described. The project was mainly implemented in Python. The source code to the Git repository can be found at the link [26].

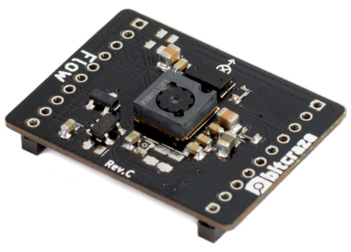
3.1 Preparation

3.1.1 Hardware

Besides the drone Crazyflie, Bitcraze also develops expansion decks, which can be attached to the Crazyflie. One of interest is the flow deck, which consists of two sensor units: a time of flight sensor (ToF) which measures the distance to the ground with high precision and an optical flow sensor that measures the velocity relative to the ground [3]. The flow deck makes the drone understand its movement a lot better than if only IMU measurements are used. Since measurements from the accelerometer need to be integrated twice to get a position estimate, the error from noise in the sensor can get very big. When using the flow deck, the velocity measurements in xy-plane only needs to be integrated once to get the position estimate and the ToF sensor in z-direction gives the absolute distance in that direction. Therefore it reduces the positional drift and is more precise.

Bitcraze's Crazyflie is designed for development projects. The code is open source and its firmware is easy to install and to use. It also has useful functionalities such as logging variables from sensors and setting parameters for control. The Crazyflie communicates through a long range and low latency radio that reaches up to 1 km. A USB-dongle (Crazyradio PA) is used to connect the Crazyflie to the computer. The Crazyflie is also equipped with a 10-DOF IMU, which consists of accelerometer, gyroscope, magnetometer and high precision pressure sensor [1].

(a) Crazyflie 2.1. Image from [1].

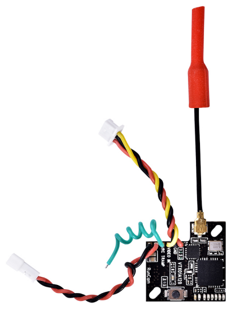

(b) Flow deck v2 expansion deck. Image from [3].

(c) Crazyradio PA dongle. Image from [2].

Figure 3.1: Hardware components from Bitcraze.

The battery is a 240 mAh LiPo battery. It has nominal voltage of 3.7 V [1]. This requires that additional components attached to the drone can operate at this voltage and does not consume too much power. An additional requirement is the weight of the components. As mentioned in the Limitation section, the Crazyflie has an upper bound of its lifting capacity. It can maximum carry 15 g [1]. Which means that the camera and video transmitter have to be very small. The company RunCam produces nano sized cameras and video transmitters which meet these requirements. The specific camera used for this project is the RunCam nano2. It has a field of view of 155-170 °and a horizontal resolution of 700 TVL [32]. The video transmitter that was used for this project is the RunCam TX200U. The video receiver, that was used in this project was the Eachine ROTG02.

(a) RunCam nano 2. Image from [32].


(b) RunCam TX200U video transmitter. Image from [33].

(c) Eachine ROTG02 video receiver. Image from [9].

Figure 3.2

The hardware components that were used in this project are specified in Table 3.1 and the final set-up can be seen in Figure 3.3.

Table 3.1: Table of hardware components used in the project.

Part	Name/model	Producer
Drone	Crazyflie 2.1	Bitcraze
Expansion deck	Flow deck v2	Bitcraze
Camera	Nano2	RunCam
Video transmitter	TX200U	RunCam
Cable		self made
Camera/video transmitter mount		self made
Video receiver	ROTG02	Eachine
Radio communicator	Crazyradio PA 2.4 GHz	Bitcraze
Processing unit	Thinkpad T470s	Lenovo

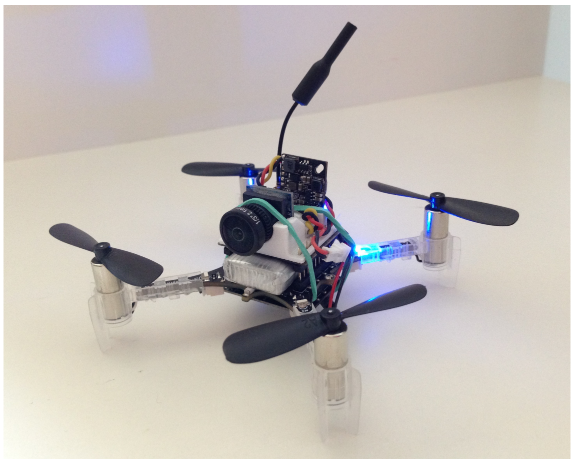


Figure 3.3: The final set-up. A 3D printed holder for camera and video transmitter was designed to keep the parts steady and in place. A rubber band was used to make sure that the camera could not move and to keep cables clear from the propellers. The flow deck is not visible in the picture, since it is attached at the bottom of the drone.

3.1.2 Collect data

Data for both calibration and the project was collected with a self built system as shown in Figure 3.4. In the figure, the blue line marks the code written specifically for this project. In Data Recorder, images from the camera were read by using the VideoCapture class [28] from OpenCV [27]. Every image was saved into a folder and marked with a timestamp in nano seconds when it was logged.

Also data from the Crazyflie was logged here. It was connected to the computer by the radio dongle. Bitcraze's own Crazyflie client has a system for logging variables. The data of interest was the one from the accelerometer and the pose estimate. The estimate of the pose was made by the extended Kalman filter (EKF) in the Crazyflie. The EKF is a recursive filter which is used to estimate states in non-linear dynamic systems where the measurements are noisy (in this case, from the accelerometer, the gyroscope and the flow deck). The EKF defines a global coordinate system, where the origin is at the start position and the orientation of the axes are the same as the local coordinate system of the drone. The pose data was expressed in the global reference frame of the drone. The known relation between the world and the global coordinate system of the Crazyflie is that the scale is the same but orientation depends on where the Crazyflie starts. In the home alarm application, the idea was to use a landing platform for when the Crazyflie is charging and the alarm is inactive. The start position and orientation of z-axis was therefore known. The orientation of x and y-axis could be arbitrarily directed as in a positive oriented reference frame. The knowledge of the relation between world and the global coordinate system of the drone was used when the scale of the point cloud shall be estimated, see section 3.2.3.

The logging of the variables were done by connecting the Crazyflie client to ROS. ROS stands for *Robot operating system*, and is an open source software which is used to create robot applications [29]. In ROS, executable files are seen as nodes. They can communicate with each other by publishing messages to topics or by subscribing to topics. The Crazyflie client is a node that publishes messages onto an IMU topic and a pose topic. The node, which consists of Data Recorder, subscribes to the mentioned topics and logs the data in two different csv files together with a timestamp in nano seconds.

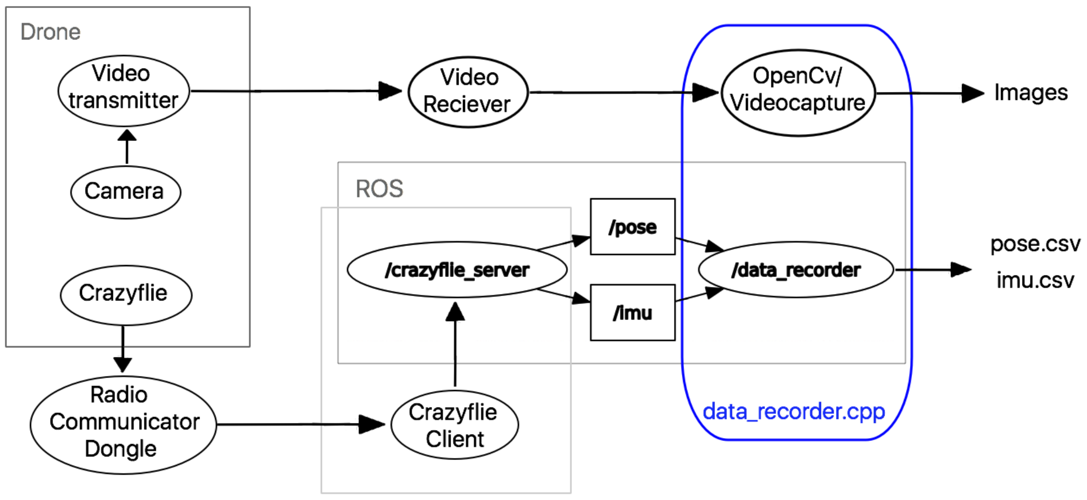


Figure 3.4: Overview of system for collecting data. The blue enclosed area marks where the data was logged. Inside the ROS-field, the circles are nodes and the squares are topics.

3.1.3 Calibration

To find the intrinsic (calibration matrix K, see equation 2.5) and distortion parameters (k_1, k_2, k_3, k_4, see equation 2.8) of the camera and the spatial and temporal relations between IMU and camera, the system needs to be calibrated. This was done by using Kalibr [22], which is a visual-inertial open source calibration system. Two toolboxes were used, *Multiple camera calibration* (based on [21]) and *IMU-Camera calibration* (based on [14] and [13]). The first returns the intrinsic and distortion parameters of the camera. The second returns the time shift between IMU and camera and the transformation matrix $T_{cam,IMU}$, from IMU to camera coordinates. The time shift between the IMU and camera occurs because the data from the two sensors travel different paths before it gets logged, see Figure 3.4.

The calibration of the camera can be shortly summarized as follows: a planar target was generated with known dimensions. In this project, the aprilgrid shown in Figure 3.5 was used as target. The world coordinates of the corners in the target was therefore known. Images of the target were taken from all angles to excite all axes for better parameter estimation. An algorithm for corner detection was used to determine the corners' pixel coordinates. Since the target has known 3D structure, the intrinsic parameters were under some constraints. The intrinsic and distortion parameters can then be estimated by using the method presented in [39].

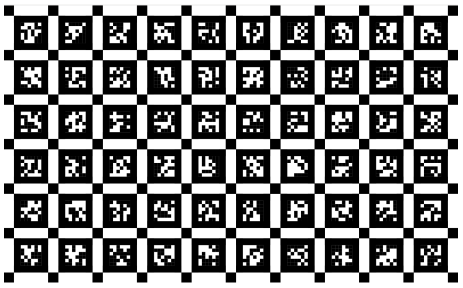


Figure 3.5: The aprilgrid that was used as target in the calibration. The dimensions of the target could be measured and were specified in advance before calibration.

3.1.4 3D reconstruction by COLMAP

COLMAP is an open source software, which has a Structure-from-motion (SfM) [34] and Multi-view-stereo (MVS) [35] pipeline with a graphical interface. COLMAP can do image-based 3D reconstructions of the scenery where the images were collected.

After data of an environment was collected (images, IMU and pose), the image set was sampled to a subset. Depending on how fast the drone moved while collecting the data, different sample rates were chosen. Since images with almost the same camera pose contains the same information, a larger sample rate value was chosen if the drone moved slow and a smaller sample rate value was chosen if the drone moved fast. However, the images must have an overlap.

The images in the subset were undistorted, by using OpenCV's fisheye library and the now known calibration matrix and distortion parameters.

The undistorted images were then used to create the point cloud by using COLMAP's reconstruction tool. The first step was to specify the camera model and its intrinsic parameters, followed by extracting features. COLMAP uses SIFT to detect and extract features. The extracted features are then matched and verified geometrically before the SfM step was performed. A sparse reconstruction was returned, which consists of camera poses and the coordinates of the reconstructed 3D-points. The output could later be used to do a dense reconstruction by MVS. That was not of interest in this project. Only images of dense reconstructions are shown in the result to make it easier to interpret the point clouds and for the interested reader.

The interesting information in the output is specified below and where it can be extracted. How the information will be used, is explained in section 3.2 further down below.

The output file **images**, see Table 3.2, contained the camera pose for all images used in the reconstruction. The poses were specified as the transformation from point cloud (PC) coordinate system to camera coordinate system. It was described by a quaternion, $\mathbf{q}_i = [Q1\ \ Q2\ \ Q3\ \ Q4]$, and a translation vector, $\mathbf{t}_i = [TX\ \ TY\ \ TZ]^T$. Here i, marks the index of the images used in the reconstruction. By using the convention in equation 2.30, the transformation between every camera pose and PC can be written

$$P_{c,i} = \mathbf{q}_i \cdot P_{PC} \cdot \mathbf{q}_i^* + \mathbf{t}_i, \tag{3.1}$$

where $P_{c,i}$ and P_{PC} are the same point but defined in camera i's respectively PC's coordinate system.

Every image has a list of detected feature points in that specific image. In the list, the points' pixel coordinates (X_{pix}, Y_{pix}) and their POINT3D_ID were specified. A point's ID was used to keep track of the corresponding 3D-point in the reconstruction. If the feature point was not in the reconstruction, the ID was -1.

Table 3.2: The text box shows the structure of how the data was stored in the images-file. The camera pose and the detected points' 2D coordinates was of interest.

```
# Image list with two lines of data per image:
#       Image number, Q1, Q2, Q3, Q4, TX, TY, TZ
#       list of detected features as (Xpix, Ypix, POINT3D_ID)
# Number of images: 2, mean observations per image: 2
  1  -0.408716  0.0432985  0.888736  -0.203038  0.653312  0.900915  3.20866
  416.13668823  415.53173828  2431  416.13668823  415.53173828 2142
  2  0.851773  0.0165051  0.503764  -0.142941  -0.737434  1.02973  3.74345
  498.82876587  299.39712524  1532  130.46496582  2.64335108  3231
```

The output file **points3D**, see Table 3.3, contains the coordinates of all 3D-points in the reconstruction, defined in PC coordinate system.

Table 3.3: The text box shows the structure of how the data was stored in the points3D-file. Here only the coordinates of the 3D-points were of interest.

```
# 3D-point list with one line of data per point:
#       POINT3D_ID, X, Y, Z
# Number of points: 3
  3231  -0.676148  0.318962  0.959069
  3235  -0.590244  0.405555  1.87921
  3239  -0.175178  0.375718  2.16493
```

3.1.5 Data sets

Two data sets were used in this project; *Pictures* and *Kitchen*. The data sets were collected in two different environments. The first data set was simpler. Only one object was in its environment; a white door with pictures and cards hanging on it. The point cloud which was constructed of the object was consequently two dimensional and was therefore easy to interpret. This data set was the first to be used to evaluate the system.

The second data set was more complex. The data was collected in a kitchen. The created point cloud was therefore three dimensional. This data set was used to evaluate if the procedure, presented in this project, could work in an environment similar to where the system is supposed to work: in a real room.

The image set of *Pictures* and *Kitchen* contain 286 and 622 images respectively. Of these, 19 and 31 images from *Pictures* and *Kitchen* respectively were used to create the reconstruction in COLMAP and later also to find ORB descriptors to the final point cloud.

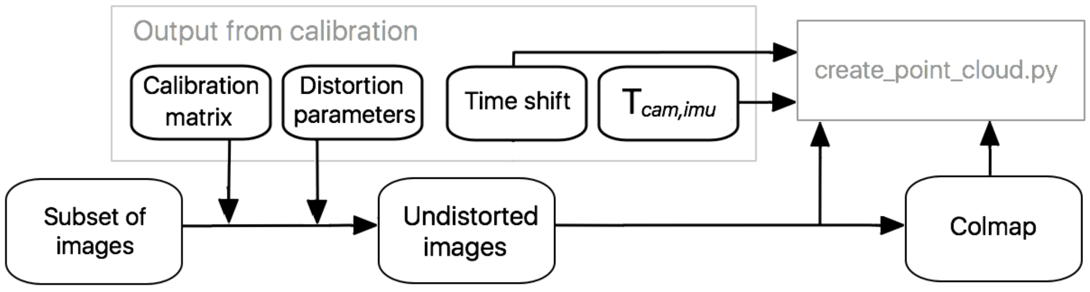


Figure 3.6: Flow chart of the steps in **Preparation**. The same subset of images used in the reconstruction will be used to create the point cloud map, as well as the output from both the calibration and COLMAP.

3.2 Create point cloud map

In the output from COLMAP, the point cloud coordinates were affected by the scale ambiguity problem, mentioned in section 1.2.3. The 3D-points and the camera pose estimates do not have the same scale, orientation or translation as the world, as illustrated in Figure 3.8 and also Figure 3.7. The orientation of the point cloud (PC) coordinate system was chosen by COLMAP to be the same as one of the camera pose estimates. Therefore, it was not of interest to use PC coordinate system to describe the map, since otherwise is the magnitude of the pose error unknown and adding a control loop would be impossible. The point cloud needs to be transformed to a reference frame, which conform to the world's reference frame, i.e. gravity direction in negative z-direction and a metric scale. As mentioned in section 3.1.2, the EKF of the Crazyflie defines a global reference frame with metric scale and the Crazyflie is equipped with IMU, which can distinguish the direction of gravity. By using this information the scale ambiguity problem can be solved.

The easiest way to describe the point cloud in world reference frame instead, was to find the transformation between PC's and world's reference frames using sensor data from the EKF and IMU. This was done in three steps. First the point cloud was rotated so that gravity direction of the point cloud was parallel to the world's gravity direction. Secondly, the point cloud was scaled to a metric scale. Lastly, the point cloud was translated, so that the origin was placed on floor level. These steps put together formed the wanted transformation, which

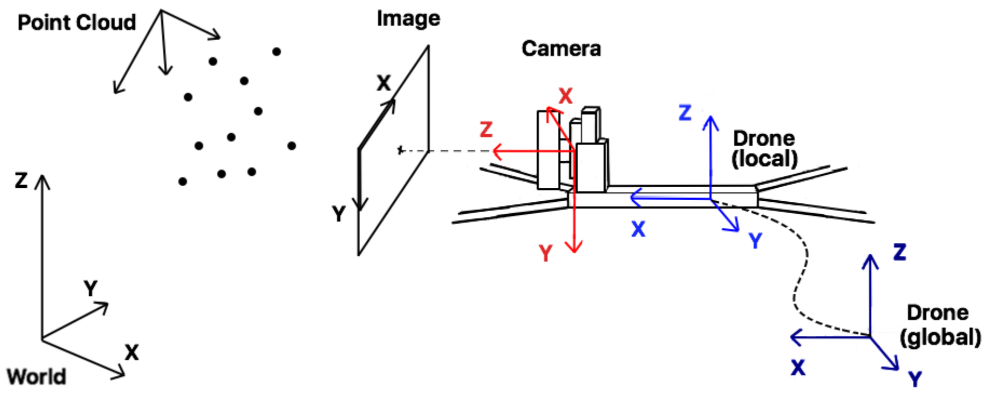


Figure 3.7: The figure shows an overview of all reference frames/coordinate systems that were used in the project.

created the point cloud map.

For the drone to be able to use this map for localization, every point also needed to have a feature descriptor, which described the the environment. This was done last in the following section.

When the point cloud map has been created, all information; 3D-point coordinates, descriptors and start pose estimate was saved in a npy-file. Which will be loaded into the localization algorithm in the initialization step. Figure 3.12, last in this section, illustrates the flow chart of how the point cloud map is created.

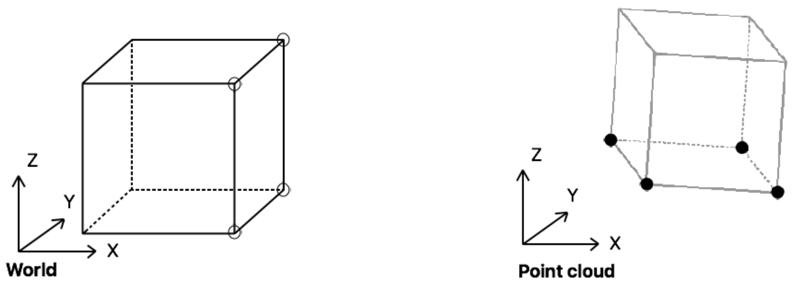


Figure 3.8: The left figure shows a scenery, consisting of a square, and the orientation of the world's coordinate system. The right figure shows the same scenery, but in PC local coordinate system. Both orientation and scale of the scene is different. The visible feature points are encircled and the corresponding points in the cloud can be seen as black dots.

3.2.1 Find closest data

The collected data was not logged with the same frequency. The camera has a frame rate of 25 fps, while the Crazyflie client collected 100 samples of data from IMU and pose every second. Not every image was later used to do the reconstruction, see Figure 3.9. Of the image stream, every 15th image of the *Pictures* data set and every 20th image of the *Kitchen* data set was used in the reconstruction. Every image in the subset was therefore matched with the temporally closest data point in both IMU and pose data sets by looking at the timestamps. The known time shift between the camera and IMU was taken into consideration. This was for later use in the sections 3.2.2 and 3.2.3 further down below.

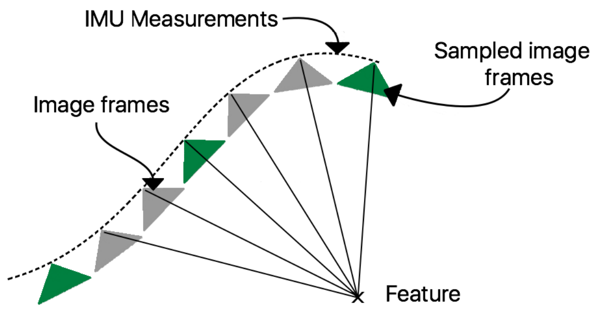


Figure 3.9: IMU and pose data were logged with a higher frequency than the images. The subset of the images used in the reconstruction and to create the point cloud map were matched to the IMU and pose data which were temporally closest for later use.

3.2.2 Rotate

As described in the introduction to the section 3.2, a transformation that described the relation between PC and world coordinate system was desired. The first step to achieve the goal was to find a rotation quaternion, $\mathbf{q}_{rot}$, which described the rotation between the two reference frames, i.e.

$$P_{rot} = \mathbf{q}_{rot} \cdot P_{PC} \cdot \mathbf{q}^*_{rot}. \tag{3.2}$$

To determine $\mathbf{q}_{rot}$, the matched image frames and accelerometer data was used. Since the drone moved in almost constant speed, the biggest contribution to the accelerometer was the gravity, which pointed in a positive z-direction of the world. In Figure 3.10, the vector is called a_{imu}. The vector was normalized to unit vector to make the following computations simpler.

The transformation between camera and IMU, $T_{cam,IMU}$, was known from calibration. According to equation 2.29, the upper left corner of $T_{cam,IMU}$, was the rotation matrix, which can be converted into a quaternion, $\mathbf{q}_{imu}$.

Also the transformation between the camera and PC was known from COLMAP's camera pose estimation (see equation 3.1). It was therefore possible to express the vector a_{imu} in PC's coordinate system instead. Since the orientation of a_{imu} only was of interest, the translation vector of all transformations can be removed. By indexing all images in the subset and denoting the index set I, the equations

$$a_{c,i} = \mathbf{q}_{imu} \cdot a_{imu,i} \cdot \mathbf{q}^*_{imu}, \tag{3.3}$$
$$a_{PC,i} = \mathbf{q}^*_i \cdot a_{c,i} \cdot \mathbf{q}_i, \tag{3.4}$$

were derived for every $i \in I$. By putting equation 3.3 and 3.4 together, the following equation is obtained

$$a_{PC,i} = (\mathbf{q}_i^* \cdot \mathbf{q}_{imu}) \cdot a_{imu,i} \cdot (\mathbf{q}_{imu}^* \cdot \mathbf{q}_i), \tag{3.5}$$

which described the transformation between IMU and PC reference frames for the camera pose used when taking the image i.

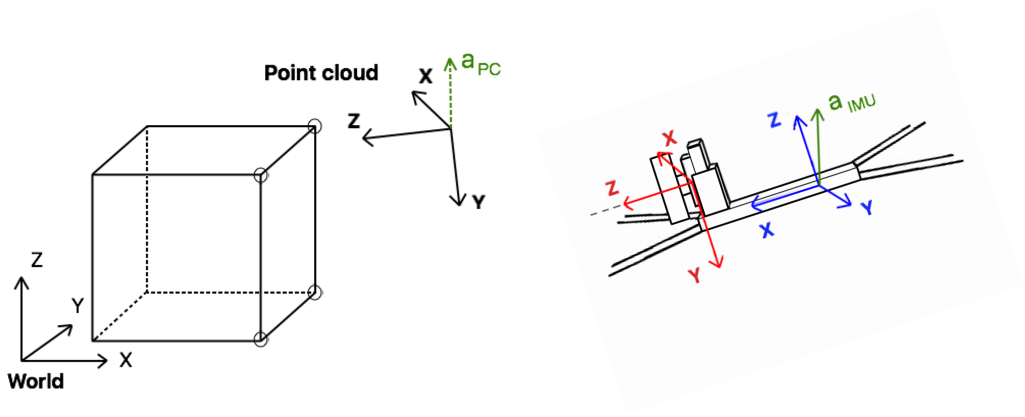


Figure 3.10: The figure shows the different global coordinate systems. With the vectors $a_{PC,i}$, where $i \in I$, it was possible find the quaternion $\mathbf{q}_{rot}$, which described the wanted rotation.

This was done for all $i \in I$, which leaved $n = |I|$ number of a_{PC} vector estimates. The average of the vector was computed as

$$\bar{a}_{PC} = \frac{1}{n}\sum_{i=1}^{n} a_{PC,i}. \tag{3.6}$$

The desired rotation was the one which rotated z-axis to a_{PC}. To do this, the rotation vector r was found by the cross product. Since a_{PC} was normalized and $z = [0\ 0\ 1]$, both vectors are unit vectors Therefore, r was also a unit vector.

$$r = \bar{a}_{PC} \times z = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}. \tag{3.7}$$

The angle of rotation θ, defined as Figure 3.11 illustrates, is

$$\cos\theta = z \cdot \bar{a}_{PC}, \tag{3.8}$$

$$\theta = \arccos(z \cdot \bar{a}_{PC}). \tag{3.9}$$

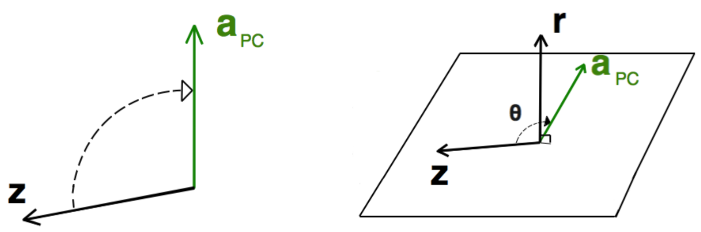


Figure 3.11: The rotation vector r was perpendicular to the plane, spanned by z and a_{PC}. The angle between z and a_{PC}, θ, was then the rotation angle.

When both the rotation vector and rotation angle were known, the formula from 2.25 was used to obtain $\mathbf{q}_{rot}$, i.e.

$$\mathbf{q}_{rot} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}e_x i + \sin\frac{\theta}{2}e_y j + \sin\frac{\theta}{2}e_z k. \tag{3.10}$$

Every point in the point cloud was rotated using the formula in equation 3.2.

3.2.3 Scale

The next step was to scale the point cloud to metric scale, so that the scale of the point cloud map was the same as the world. This was done by using the pose data from the Crazyflie's extended Kalman filter, which was expressed in the drone's global coordinate system. As mentioned in section 3.1.2, the scale of the global coordinate system and the world's coordinate system was the same. Since the flow deck was used, the pose estimates got very accurate.

From the camera pose estimates, the camera center in PC coordinates was computed, $C = -\mathbf{q}^* \cdot \mathbf{t} \cdot \mathbf{q}$, for every image frame. Also from the matched pose data, the position of the camera center in world coordinates was known for every image frame.

By computing the euclidean distance,

$$d = \sqrt{(x_j - x_k)^2 + (y_j - y_k)^2 + (z_j - z_k)^2} \qquad \forall j, k \in I \text{ and } j \neq k, \tag{3.11}$$

between two camera positions in both coordinate systems, the scale factor,

$$s_i = \frac{d_{PC,i}}{d_{w,i}}, \tag{3.12}$$

could be computed. The scale factor was computed for every combination of two camera positions, giving $n(n-1)/2$ scale factor estimates. Remember that n denoted the number of images in the subset.

Because of noise in both camera pose estimates and pose data, the computed scale can vary. Outliers were removed by IQR, leaving N measurements. The average scale factor was then computed

$$\bar{s} = \frac{1}{N}\sum_{i=1}^{N} s_i. \tag{3.13}$$

Every point in the rotated point cloud was scaled by

$$P_s = \frac{1}{\bar{s}} P_{rot}. \tag{3.14}$$

3.2.4 Translation

Lastly, the point cloud was translated so that the origin of the coordinate systems coincided. This was done by looking at the position of the first camera in both world and PC coordinates. The positional difference in x, y and z was computed,

$$\begin{bmatrix}\bar{x}\\ \bar{y}\\ \bar{z}\end{bmatrix} = \begin{bmatrix}x_{PC}\\ y_{PC}\\ z_{PC}\end{bmatrix} - \begin{bmatrix}x_w\\ y_w\\ z_w\end{bmatrix}. \tag{3.15}$$

Every point in the point cloud was corrected according to these differences. Since the flow deck's ToF sensor measured the absolute distance to the floor, the origin should be placed at floor level.

In the equation 3.15, x_{PC}, y_{PC} and z_{PC} are the camera center in PC coordinate system and x_w, y_w and z_w are the camera center in world coordinate system. The transformation for translating the point cloud was then obtained as

$$P_w = P_s - \begin{bmatrix}\bar{x}\\ \bar{y}\\ \bar{z}\end{bmatrix}. \tag{3.16}$$

If the transformations were put together for rotating 3.2, scaling 3.14 and translating 3.16 the points in the cloud, the wanted transformation was derived

$$P_w = \frac{1}{\bar{s}}(\mathbf{q}_{rot} \cdot P_{PC} \cdot \mathbf{q}^*_{rot}) - \begin{bmatrix}\bar{x}\\ \bar{y}\\ \bar{z}\end{bmatrix}. \tag{3.17}$$

In section 1.3, it was mentioned that the start position of the drone was known because of the fixed location of the landing platform. When collecting the data for building the point cloud map, it was assumed that the drone started from the platform. Therefore could the first pose estimate from COLMAP be used to give a rough start pose estimate. Since the point cloud has been translated, the estimated start position should correspond to the start position in the world. However, the start orientation could be arbitrarily.

3.2.5 Find ORB-descriptors

COLMAP used SIFT to detect and extract features from the images. Since SIFT is based on histograms of gradients, it has to compute a gradient for each pixel in a patch. This takes time and SIFT was therefore less suitable to use in a real time application. Instead it was a lot faster to use binary descriptors, like ORB. A binary descriptor can encode information of a patch in a binary string using only comparison of intensity images. Comparing two ORB-descriptors can also be done very fast when using the Hamming distance. Every point in the cloud should therefore be described by an ORB-descriptor rather than a SIFT-descriptor.

To goal was to pair as many 3D-points as possible to an ORB-descriptor. This was done by using the same image subset that was used to create the point cloud in COLMAP. By using OpenCV, ORB-features were detected and computed for every image in the subset. The positions of all ORB-keypoints in an image were known. As described in the previous section 3.1.4, all keypoints that were detected by SIFT in an image were specified in a list with a POINT3D_ID that connected keypoints to their corresponding 3D-point in the cloud. By pairing ORB-keypoints with SIFT-keypoints, 3D-points got connected to ORB-descriptors. The keypoints were paired by taking the ORB-keypoint which was closest to a

SIFT-keypoint, by computing the euclidean distance. A threshold was used to remove pairs with the euclidean distance bigger than one pixel.

This was repeated for every image in the image subset. If more than one ORB-descriptor was matched to a 3D-point, all descriptors were saved in a list. The 3D-points that did not get an assigned ORB-descriptor, were removed from the cloud.

To make the following result and discussion further down below easier to formulate and understand, some shorter and describing words were instead used for the following three sets: the set of visible 3D-points in an image was called SIFT-set, since the points were created using SIFT-features, the set of detected ORB-keypoints in an image was called ORB-set, and the set of visible 3D-points in an image with an assigned ORB-descriptor was called SIFT→ORB-set.

When using the function in OpenCV, which detected and computeed ORB-features from the images, a parameter had to be specified, saying how many features the function should try to find and extract. In this project the parameter was set to 3000. The limited quality of the images made it difficult for the function to find that many features. However, the parameter was chosen to be big to make the ORB-set as large as possible. The idea was; if the ORB-set was large, then the intersection between the SIFT-set and ORB-set would be as large as possible, which was the SIFT→ORB-set. A large SIFT→ORB-set implies that there were many 3D-points in the point cloud map. A map with more 3D-points was more describing and representative of the environment than a map with less points, meaning that the localization could get more accurate.

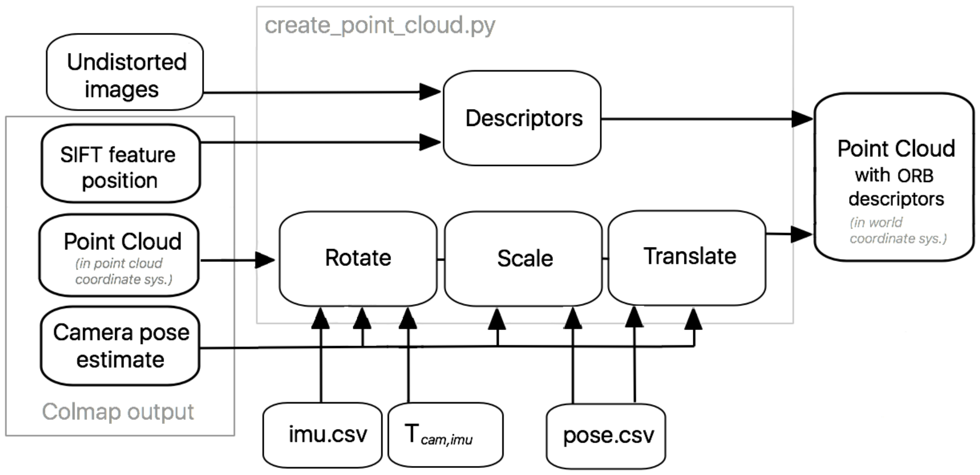


Figure 3.12: The flow chart shows the different steps in creating the point cloud map.

3.3 Localization algorithm

The localization algorithm used the information from the images from the camera, which was attached to the drone to estimate its pose. When the drone was moving in a previously mapped environment, it used the detected features in the images and compared them to the information in the map to understand where in the map it was localized. The goal when creating the map was to create a map that corresponds to the world's scale and orientation. By knowing where it was localized in the map, it also knew where it was localized in the world.

Before the drone could start moving around, the system was initialized by loading the map, start position estimate, calibration matrix and distortion parameters of the camera. The first image that was read into the system was used to validate and/or determine the start pose estimate. After that, the initialization step was done and the drone could start moving around.

Below are the main steps of the algorithm listed. Followed by a flow chart showing the different paths the algorithm could take when trying to find a pose estimate. The different paths and steps are explained more thoroughly further down in this section.

The main steps of the localization algorithm are:

- Detect and compute ORB-features in the image from the camera.
- Match ORB-descriptors from the image to ORB-descriptors in point cloud map to get correspondences between 3D-points and 2D-points.
- Solve the P3P-problem using RANSAC to obtain a pose estimate.

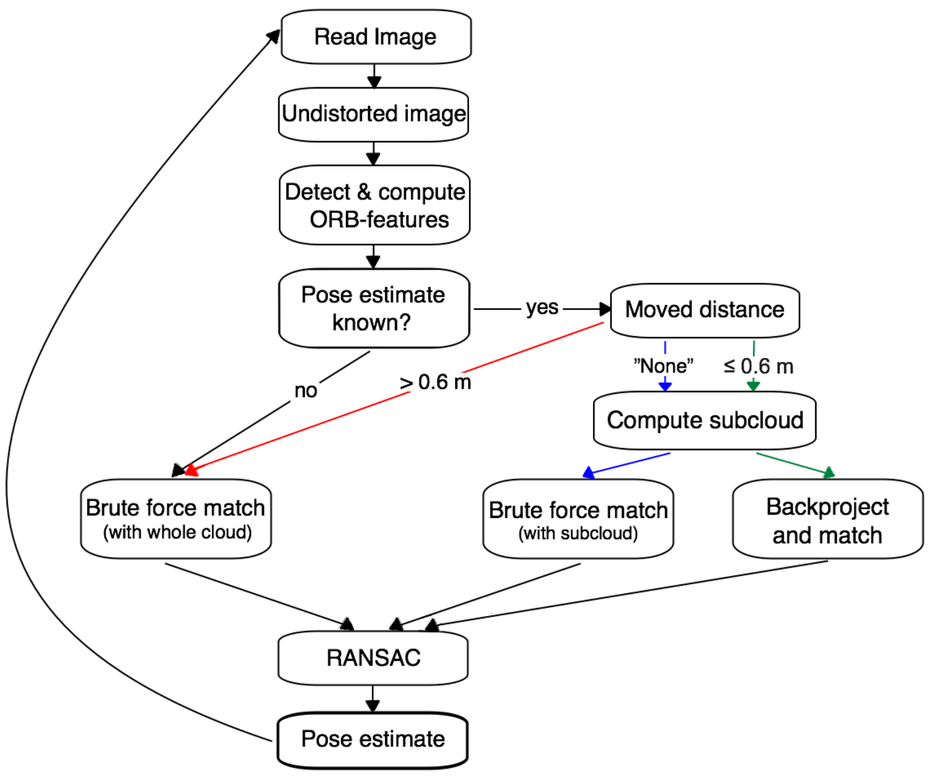


Figure 3.13: The flow chart shows the different steps in the localization algorithm.

First, an image was read from the image stream from the camera. The image was undistorted by using the intrinsic and distortion parameters known from calibration. ORB-features were detected and computed in the image. The parameter, which decided how many features the function should try to find was set to 3000, i.e. same as when creating the point cloud map. The idea was to create a big overlap between the descriptors in the map and the descriptors detected in the image, meaning that the number of possible correct matches should increase. This could then also increase the accuracy of the pose estimate.

Next step was to match descriptors. In the algorithm developed in this project, there were different paths to match descriptors and to get point correspondences, see Figure 3.13. The different paths depend on the current state and what was known about the previous pose.

If the drone already had a pose estimate, then the moved distance was checked, e.i. the euclidean distance between the current pose and the previous pose. If it was the first time computing an estimate, then moved distance was “None”. The start pose estimate was not so accurate, therefore Brute Force match was used to match descriptors.

If the moved distance was ≤ 0.6 m, then the algorithm was assumed to be working. The visible points were back projected to the image plane and a matching descriptor was searched for in a small radius around it.

If the moved distance was > 0.6 m, then the pose estimate was most likely wrong, since the drone would not move so fast. This solution was rejected. By using Brute force match and try to match descriptors with all descriptors in the whole cloud, a new pose estimate was derived. In the same way a start pose was estimated, if it was not already known.

When the descriptors had been matched, then the P3P-problem was solved by using RANSAC to get a pose estimate.

Some of the steps are more thoroughly described below.

3.3.1 Compute subcloud

When the pose estimate was known, the transformation between image plane and world was also known. All 3D-points in the point cloud map could be projected into the image reference frame. By checking the x and y pixel coordinates, it was easy to determine if the corresponding 2D-point could be visible in the image. The size of the image was (640,480) pixels. Then the point was not visible if the point’s pixel coordinates were outside the intervals

$$\begin{aligned} x_{pix} &< 0 - \epsilon, \\ x_{pix} &> 640 + \epsilon, \\ y_{pix} &< 0 - \epsilon, \\ y_{pix} &> 480 + \epsilon. \end{aligned} \tag{3.18}$$

Since the drone probably had moved a little and the previous estimation was not 100 % accurate, a tolerance $\epsilon = 10$ pixels was added to the limits. By computing the visible subcloud, the number of 3D-points and descriptors used in the matching step decreased. Which could reduce computation time and risk of false matches.

3.3.2 Brute force match

When the pose estimate was not known, for example if the start pose was not known or if the previous estimated pose was rejected, then the ORB-descriptors in the image need to be compared to all descriptors in the point cloud map. The matching was conducted in a brute force way: all ORB-descriptors in the image were matched to a descriptor in the point cloud map. A match was evaluated by the hamming difference of the descriptors. As mentioned previously, a 3D-point might have been described by several ORB-descriptors. The set of matches could contain a specific 3D-point more than once. By going though all matches, duplicates could be removed. Then the match with the lowest distance was chosen. If the hamming distance was bigger than 80, then the match was considered not to be good enough and was removed. From the matches, correspondences between 3D-points and 2D-points were obtained.

3.3.3 Back project and match

When the pose estimate was known, then the algorithm was assumed to be working. The time difference between every image in the stream was small and the drone moved in a velocity so that the moved distance between every image was small. If the camera only had moved a small distance, then the 3D-points that were matched to a 2D-point in the previous image were very likely to be found very close to the previous position in the next image. By computing the new subcloud and projecting the 3D-points in the subcloud back to pixel coordinates, a descriptor was searched for in a radius of 25 pixels around the projected point. It was very likely to get a correct match. Matches that had a hamming distance bigger than 80 were removed. From the matches, correspondences between 3D-points and 2D-points were obtained.

3.3.4 Pose estimation

When the point correspondences were known, the P3P-problem, described in section 2.5, could be solved to find a pose estimate. The set of matches contained outliers. By using the procedure of RANSAC, see section 2.3.3, the pose estimate got less sensitive to the false data. The three point correspondences, which were needed to solve P3P-problem, were chosen randomly from the set of matches. The geometric system of equations in 2.33, were solved to find the pose of the camera. The rest of the matches were used to validate the model by back projecting the 3D-points to pixel coordinates and computing the euclidean distance to its corresponding 2D-point. If the distance was less than eight pixels, then the match was considered an inlier. The procedure was repeated 1000 times and the camera pose with the most inliers were chosen as the correct pose.

Chapter 4
Results

The results are presented in four steps. First the results from calibration are presented. Second, the result from when creating the point cloud map. Third, the result from the localization algorithm and last, the results from two noise tests. The two data sets that where used are presented separately.

4.1 Calibration

For the calibration 841 images were used, of which 39 were used to estimate all parameters and the rest were used to validate the intrinsic and distortion parameters.

The reprojection error was: $[-0.000025 \pm 0.457717,\ 0.000010 \pm 0.497190]$ in pixels.

Table 4.1: Camera calibration parameters; focal length and principal point in pixels.

f_x	f_y	p_x	p_y
330.990	337.430	317.778	231.0133

Table 4.2: Distortion parameters.

k_1	k_2	k_3	k_4
−0.0592	−0.00581	−0.00442	0.00312

The transformation matrix $T_{cam,IMU}$ extracted from calibration was:

$$T_{cam,IMU} = \begin{bmatrix} -0.0112 & -0.999 & -0.0132 & -0.00855 \\ 0.0304 & 0.0129 & -0.999 & 0.0134 \\ 0.999 & -0.0116 & 0.0302 & -0.0127 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}. \tag{4.1}$$

The rotation matrix can be converted to a rotation vector to make it clearer how the rotation was defined:

$$r_{cam,IMU} = \theta\hat{\mathbf{e}} = 2.076[0.564 \ \ -0.579 \ 0.589]. \tag{4.2}$$

The rotation angle θ is in radians.

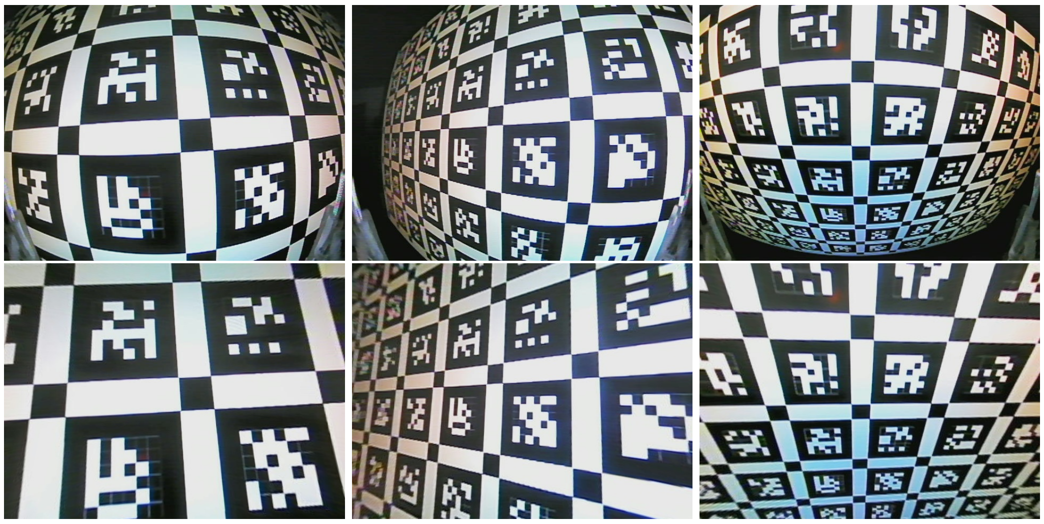


Figure 4.1: The intrinsic and distortion parameters were used to undistort images. Here is the result of a test of how well the camera and distortion models performed. The top row are images taken with the RunCam nano2 camera and the bottom row are the same images but undistorted by using the models.

4.2 Create point cloud map

The results of creating the point cloud map for the two data sets are presented separately below. Since the first data set is flat and the second data set is of a 3D environment, different methods were used to evaluate the results of the steps in creating the point cloud map.

Sometimes results will be shown just for demonstration reasons. In both sections, a dense reconstruction is shown, in the first section a scatter plot of the point cloud as it looked from the beginning is shown, in the second section images showing how features are found in an image and how points get assigned a descriptor is shown.

4.2.1 Pictures data set

COLMAP can make a dense reconstruction of the point cloud. The dense reconstruction was never used in this project, but since it is built from the point cloud that was used, it gets easier to interpret the point cloud visually by looking at the dense reconstruction, see Figure 4.2.

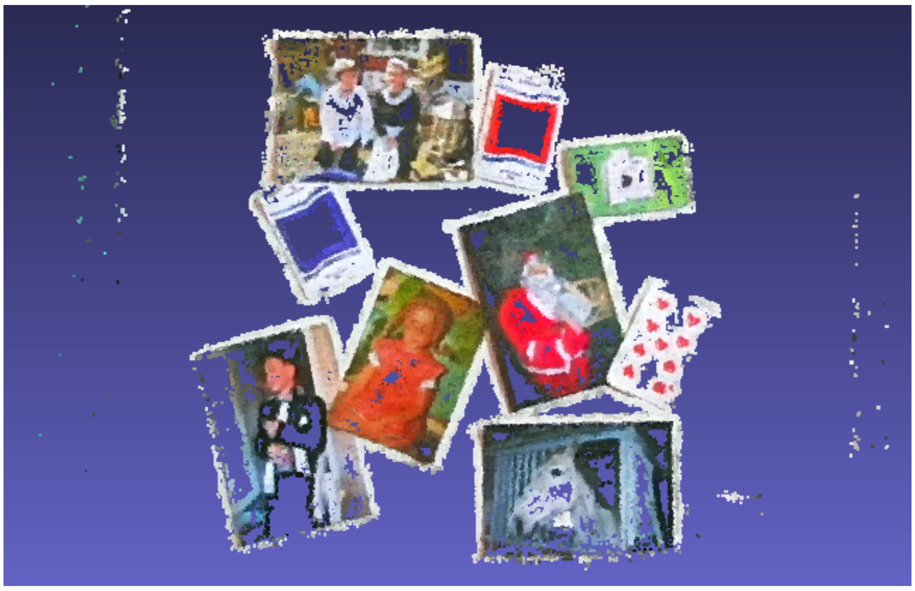


Figure 4.2: Dense reconstruction by COLMAP of the *Pictures* data set. Visualized in Meshlab.

The point cloud was built by using a sequence of 19 images, depicting different parts of the images hanging on the wall. COLMAP used SIFT to detect and extract features from the images. The features from the different images were matched and verified geometrically before the scene was reconstructed by following the procedure of Structure from Motion.

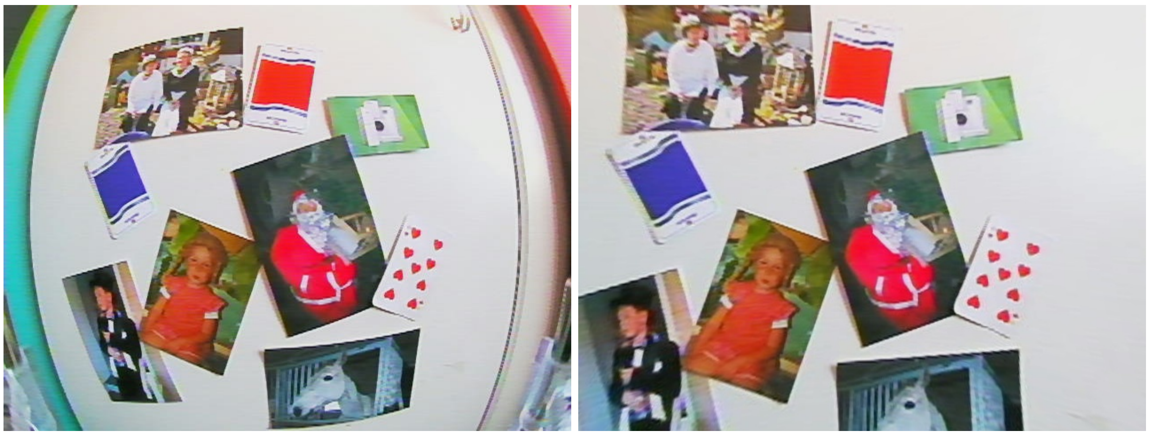


Figure 4.3: An example image from the data set *Pictures*. The left image is the original and the right is after the image has been undistorted.

The total result of rotating, scaling and moving the point cloud can be seen in Figure 4.4 and Figure 4.5.

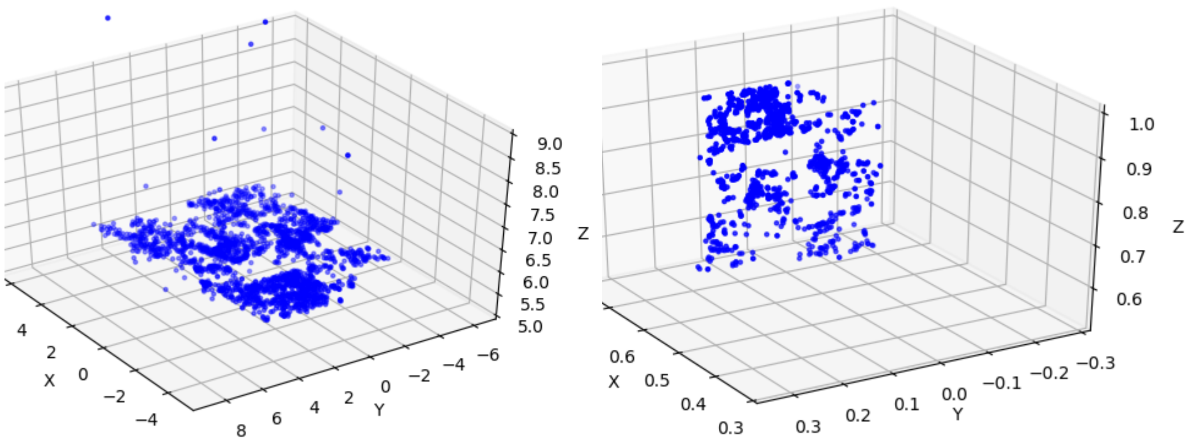


Figure 4.4: The left figure shows the output point cloud from COLMAP. The right figure shows the point cloud map, where the point cloud has been rotated, scaled and translated. Also some points have been removed, since not all points get an assigned descriptor. Both coordinate systems have the same orientation to make the visual change more noticeable. By visually comparing the point cloud to the dense reconstruction and the example image, it is easier to interpret the point cloud. The main structures of the images can be seen, which makes it possible to check if orientation, scale and height are reasonable.

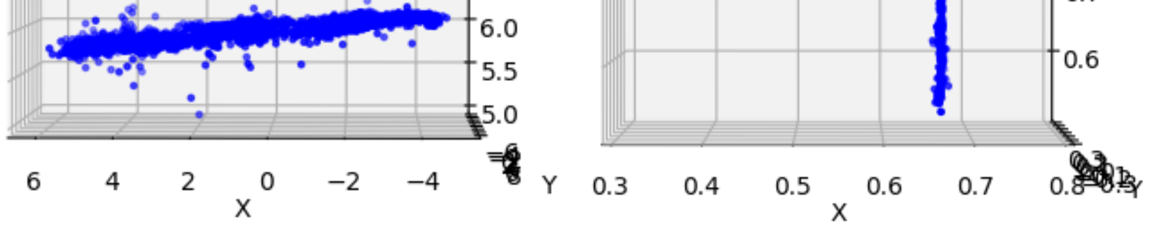


Figure 4.5: The figures shows the point cloud from the side. The left figure shows the output point cloud from COLMAP and the right figure shows the point cloud map. By comparing the two figures, it appears that the point cloud in the left figure is not completely flat. It also contains more noise than the right figure.

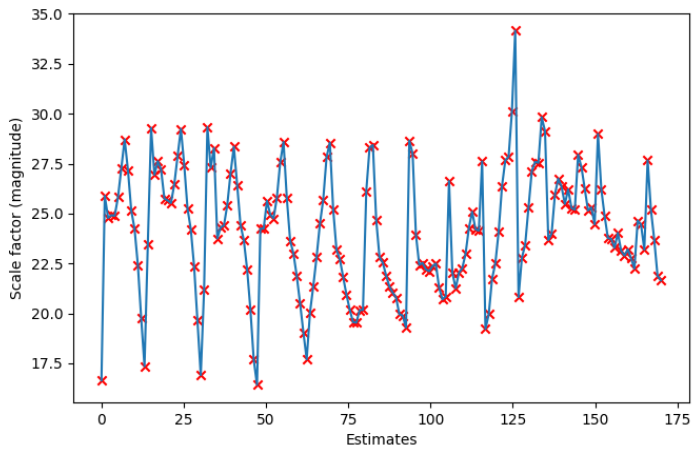


Figure 4.6: The figure shows the data of the scale estimates before outliers are removed. It contains 170 data points, which are distributed on the x-axis and the magnitude of each estimate in showed on the y-axis. By IQR, 7 data points are removed. The final scale factor estimate is 24.37.

By identifying the largest respectively smallest x and z value of the points' positions, the difference between them was computed. The distance between the points were measured with a ruler in the real world. Also the distance to the floor z_0 was measured.

Table 4.3: The table shows the difference between specific points in the world respectively in the point cloud map.

	Point cloud map	**World**
$x_{max} - x_{min}$	0.382 m	0.37 m
$z_{max} - z_{min}$	0.436 m	0.41 m
$z_{min} - z_0$	0.532 m	0.58 m

The sizes of the sets; SIFT, ORB, and SIFT→ORB were computed for every image. The average size of the sets are presented in the table below. To give a reminder: the SIFT-set is the set of visible 3D-points in an image, the ORB-set is the set of detected ORB-features in an image, and the SIFT→ORB-set is the set of 3D-points with assigned ORB-descriptor.

Table 4.4: Table shows the average sizes of the three sets of all images used when creating the point cloud map.

Set	**Average size**
SIFT	652.79 points/image
ORB	2448.05 points/image
SIFT→ORB	138.95 points/image

The point cloud contained 2267 points from the start. After removing the points with no descriptor, the point cloud contained 1021 points.

4.2.2 Kitchen data set

The point cloud of a 3D environment is more difficult to present in a 2D-image without it becoming messy and difficult to interpret. Therefore only the scatter plots from the point cloud map is shown. To demonstrate the result of the steps in creating the point cloud map, the camera positions are used instead. Since the position data from COLMAP's pose estimation and the position data from the EKF is less dense and clearly structured, the result from the three steps are easily visualized. The point cloud is transformed in the same manner.

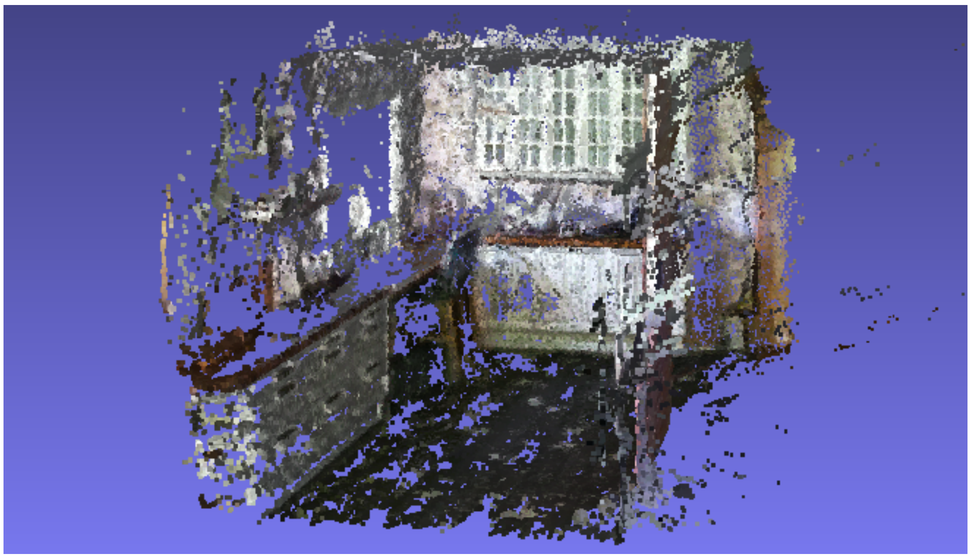


Figure 4.7: Dense reconstruction by COLMAP of the *Kitchen* data set. Visualized in Meshlab.

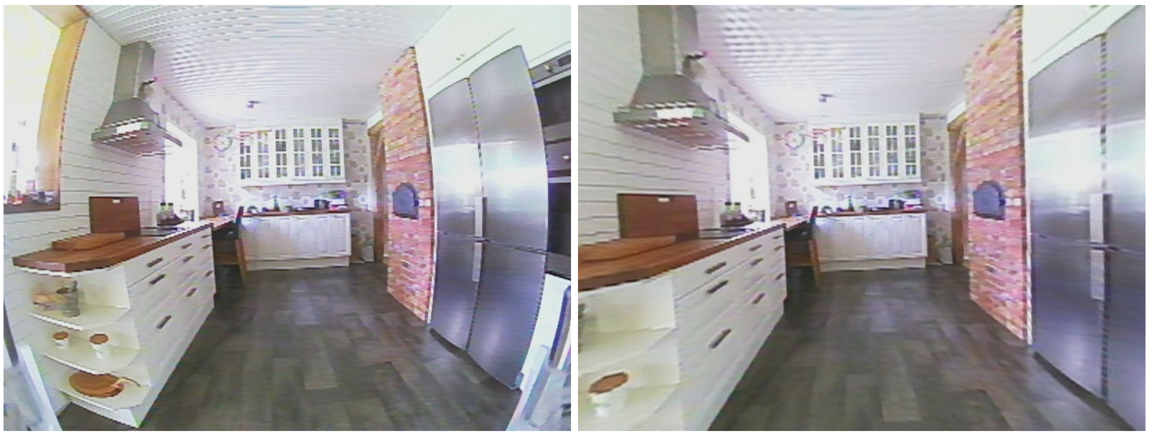


Figure 4.8: An example image of the *Kitchen* data set. The left image shows the original and the right shows the same image after it has been undistorted.

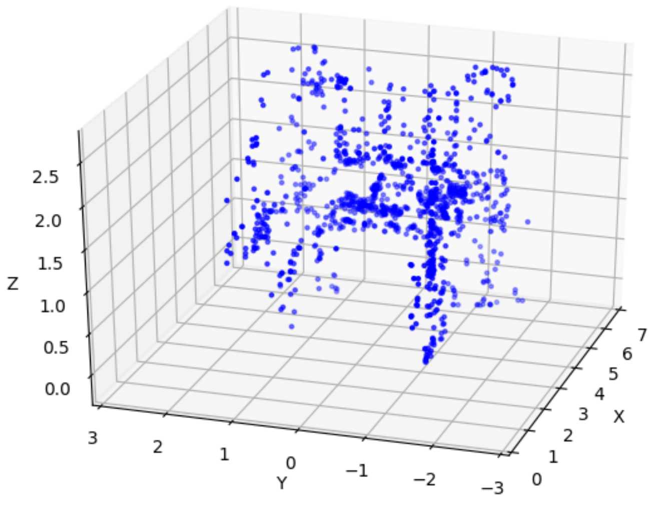
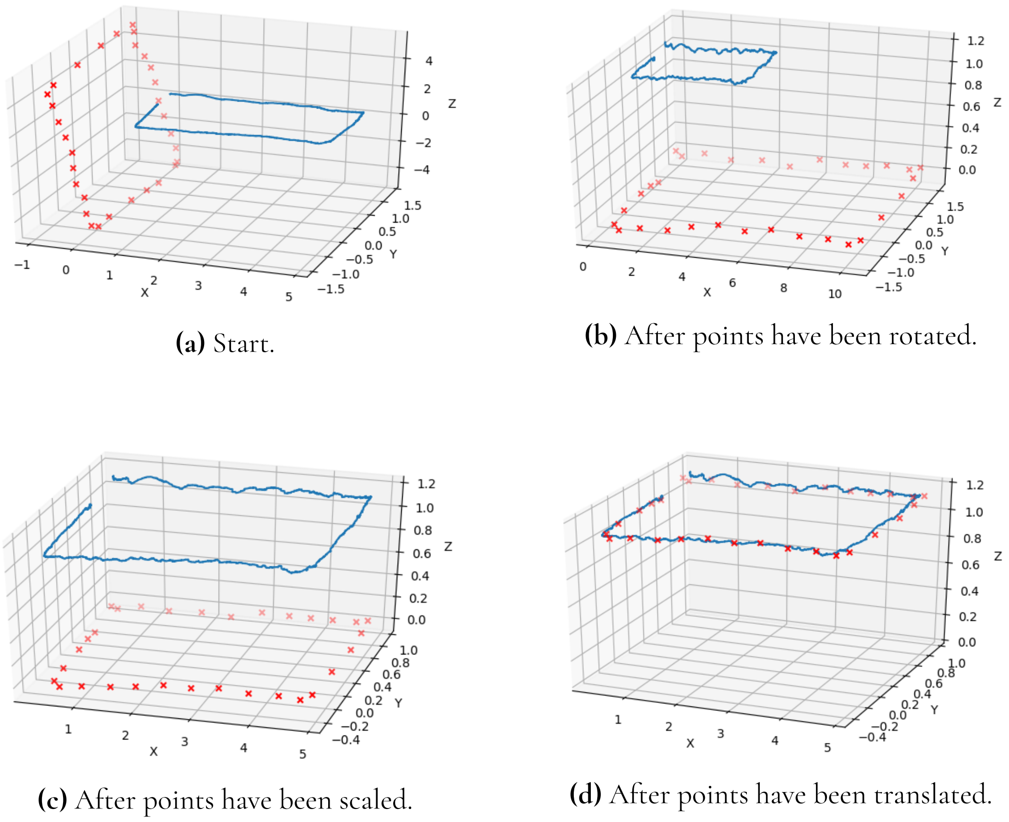


Figure 4.9: The figure shows the point cloud map. The point cloud from COLMAP has been rotated, scaled and translated. Points have been removed, which were not assigned a descriptor. The orientation of the coordinate system is approximately the same as in the dense reconstruction. By comparing the cloud to the dense reconstruction, it is easier to interpret the cloud.



(a) Start.

(b) After points have been rotated.

(c) After points have been scaled.

(d) After points have been translated.

Figure 4.10: The red crosses mark the camera pose estimates from COLMAP. The blue line is the position data from the drone.

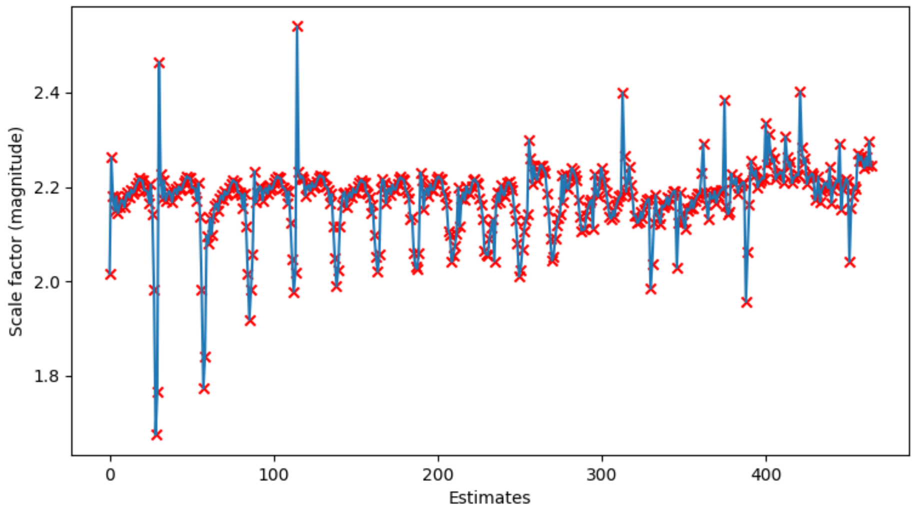


Figure 4.11: The figure shows the data of the scale estimates before outliers are removed. Of 464 data points, which are distributed on the x-axis and the magnitude of each estimate in showed on the y-axis, 145 data points are removed by IQR. The final scale factor estimate is 2.196.

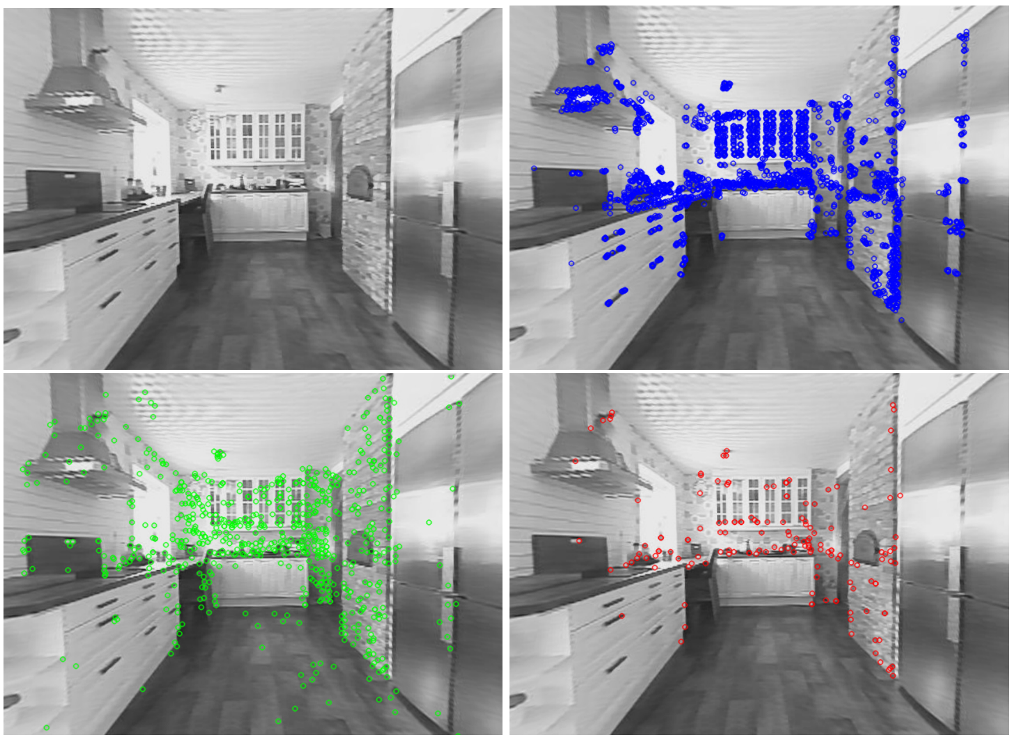


Figure 4.12: The figure shows an example image from the *Kitchen* data set. The blue markers shows all detected ORB-features in the image. The green markers shows the positions of the projected 3D-points from the cloud, which were detected using SIFT. The red markers show the projected points which have an assigned ORB-descriptor.

The sizes of the sets; SIFT, ORB, and SIFT→ORB were computed for every image. The average size of the sets are presented in the table below. To give a reminder: the SIFT-set is the set of visible 3D-points in an image, the ORB-set is the set of detected ORB-features in an image, and the SIFT→ORB-set is the set of 3D-points with assigned ORB-descriptor.

Table 4.5: Table shows the average sizes of the three sets of all images used when creating the point cloud map.

Set	Average size
SIFT	500.8 points/image
ORB	2848.64 points/image
SIFT→ORB	105.0 points/image

The point cloud contained 3216 points from the start. After removing the points with no descriptor, the point cloud contained 1276 points.

4.3 Localization algorithm

The results of the localization algorithm are presented in two parts. First the result of the algorithm's performance in the simpler environment, *Pictures* data set, is presented. In the second part, the localization algorithm was tested in the more complex environment, *Kitchen* data set. In both cases, the same data sets were used for creating the point cloud map as for testing the algorithm. Since only a small subset of the image set is used in the reconstruction, the intersection of descriptors between the two sets is small. Therefore this was not considered as a problem. The pose estimate from COLMAP could then also be used to evaluate the algorithm.

The scale of all plots is in meters.

4.3.1 Pictures data set

Below are plots of the trajectory estimated by the localization algorithm in the *Pictures* environment.

The average moved distance between two position estimates was 0.0165 m and the average percentage of inliers in the matching step was 86.8%. The largest moved distance was 0.0612 m.

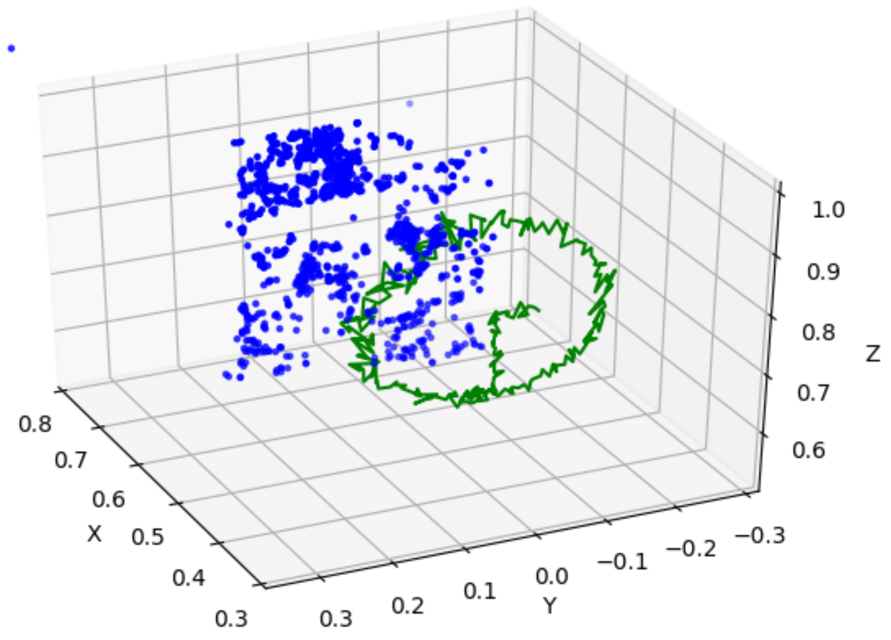


Figure 4.13: The green jagged line is the trajectory composed by the position estimates, computed by the localization algorithm of the camera in the point cloud map of the *Pictures* data set.

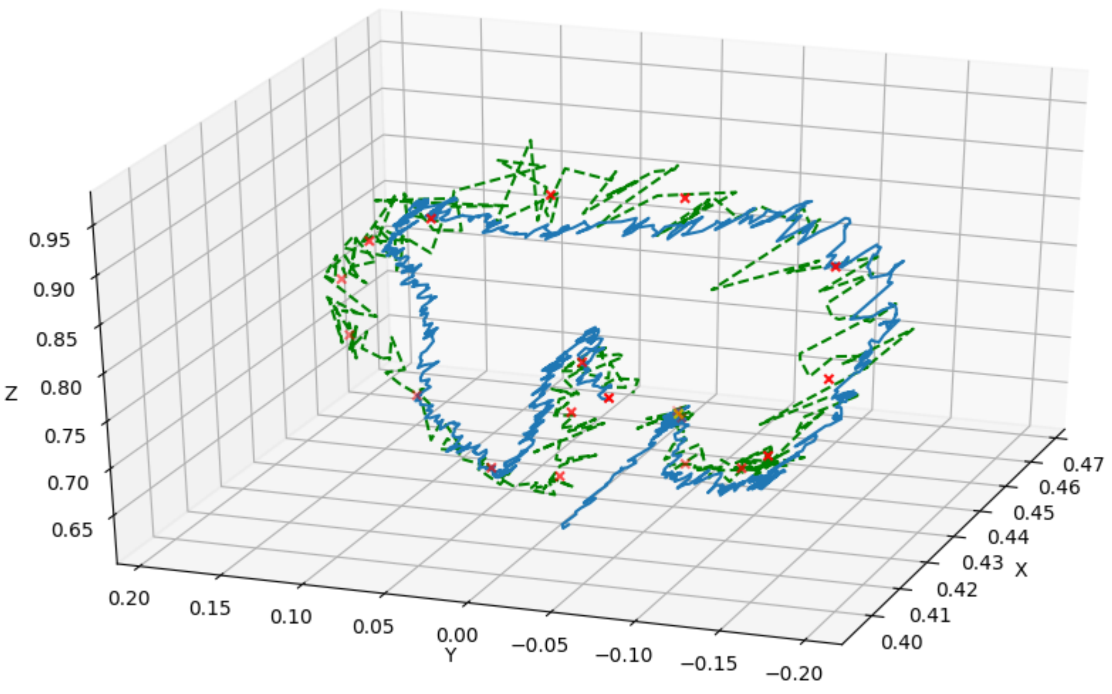


Figure 4.14: The figure shows the different position data. The red markers are position estimates from COLMAP. The blue line is position data from the drone's EKF. The green dashed line is the position estimate from the localization algorithm.

The algorithm was timed to get an idea if it was fast enough to run in real time. The average run time of the different steps can be found in the table below.

Table 4.6: Time distribution of the different steps in the algorithm. One loop includes all steps but the initialization step.

Function	Run time
Initialization	0.0473 s
Read, undistort image, features	0.0173 s
Compute subcloud	0.00519 s
Match	0.106 s
Solve P3P with RANSAC	0.00247 s
One loop	0.131 s

4.3.2 Kitchen data set

Below are plots of the trajectory estimated by the localization algorithm in the *Kitchen* environment.

The average moved distance between two position estimates was 0.0933 m and the average percentage of inliers in the matching step was 59.0%. The largest moved distance was 0.507 m and corresponding inlier matches was then 16.0%.

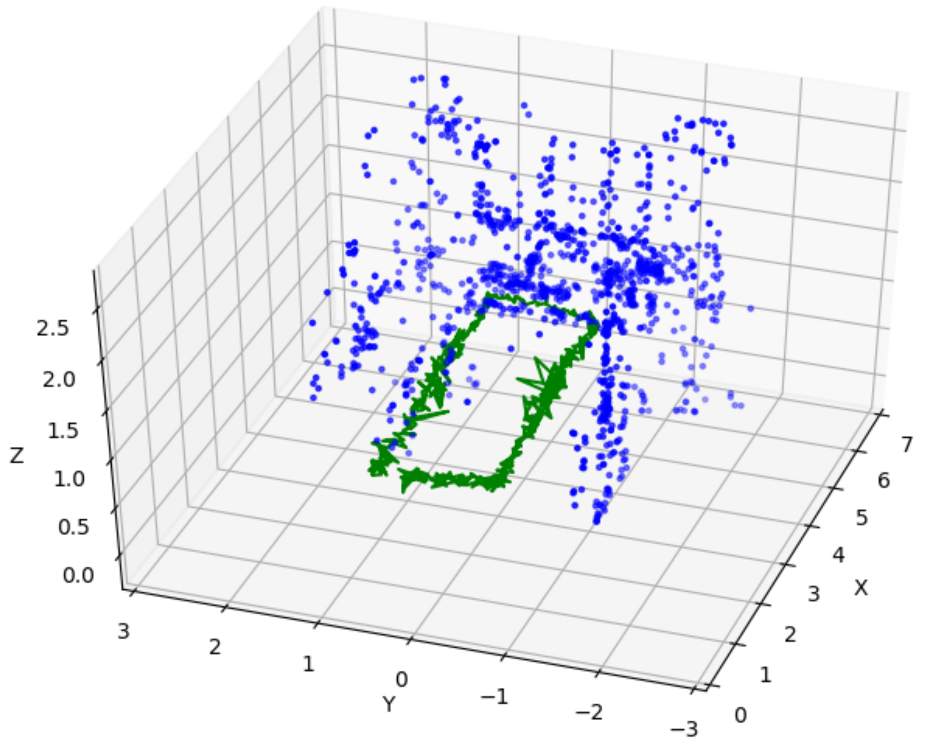


Figure 4.15: The green jagged line is the trajectory composed by the position estimates, computed by the localization algorithm of the camera in the point cloud map of the *Kitchen* data set.

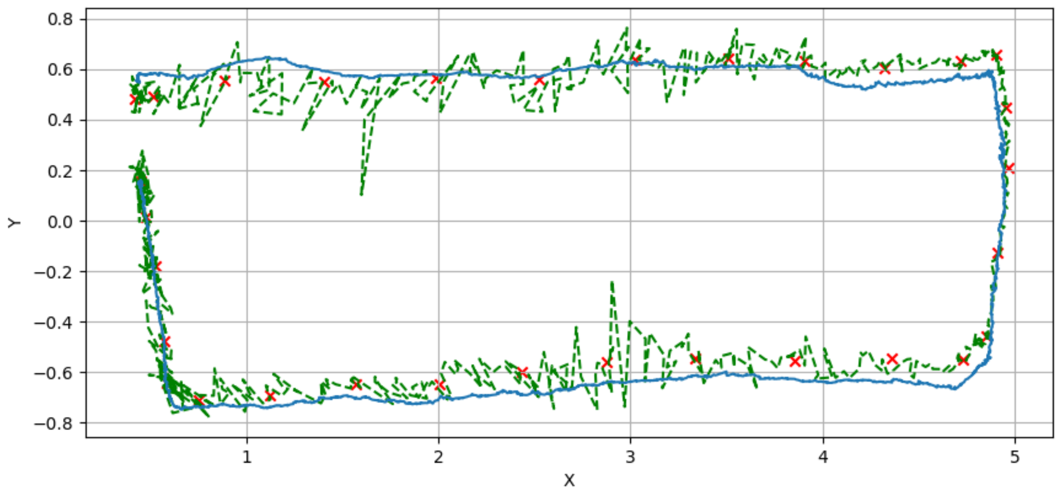


Figure 4.16: The figure shows the different position data. The red markers are position estimates from COLMAP. The blue line is position data from the drone's EKF. The green dashed line is the position estimate from the localization algorithm.

The algorithm was timed to get an idea if it was fast enough to run in real time. The average run time of the different steps can be found in the table below.

Table 4.7: Time distribution of the different steps in the algorithm. One loop includes all steps but the initialization step.

Function	Run time
Initialization	0.0511 s
Read, undistort image, features	0.0253 s
Compute subcloud	0.00879 s
Match	0.251 s
Solve P3P with RANSAC	0.00815 s
One loop	0.293 s

4.4 Noise test

Two noise tests were done to get an estimate of how noisy the algorithm's positioning was. The tests were carried out in the same environment as the *Kitchen* data set. The map, created with data from *Kitchen* data set, was used.

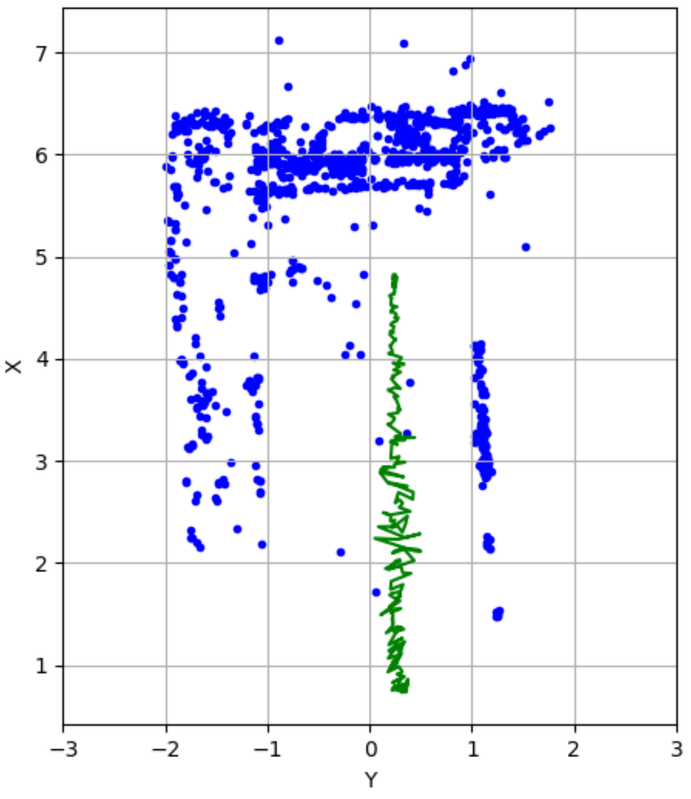


Figure 4.17: Result of the positioning in the first noise test. For easier visual interpretation, the result is shown from above. The green line is the position estimate of the camera.

In the first test, the idea was to move the drone in as straight trajectory as possible in x-direction. The drone was placed on a chair, which was pushed forward in the kitchen. The distance to the floor, 0.525 m, stayed constant while the position in y-direction might have varied a little. The movement was tried to be done with a constant velocity. The true position of the drone would then form a straight trajectory. There is no way of knowing the exact trajectory in the map. Therefore, a line is fitted to the position estimates and is used as ground truth of the trajectory in the test. The line was fitted to the 3D-position estimates by using Principal component analysis, see section 2.3.2. The vector, defining the line was denoted $\vec{v}$.

The average euclidean distance from all position estimates to the line were computed. This was done by computing the vector $\vec{u}_i$, which is defined as the vector from a point on the line to the position estimate i, and projecting this vector onto the vector $\vec{v}$. Then the obtained projected vector was denoted $\vec{w}_i$,

$$w_i = \frac{\vec{u}_i \cdot \vec{v}}{||\vec{v}||^2}\vec{v} \qquad \forall i = 1...n. \tag{4.3}$$

The norm $d_i = ||w_i - u_i||$ is the desired closest distance between the point i and the line. The average of the distance was computed

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i = 0.0615\ m. \tag{4.4}$$

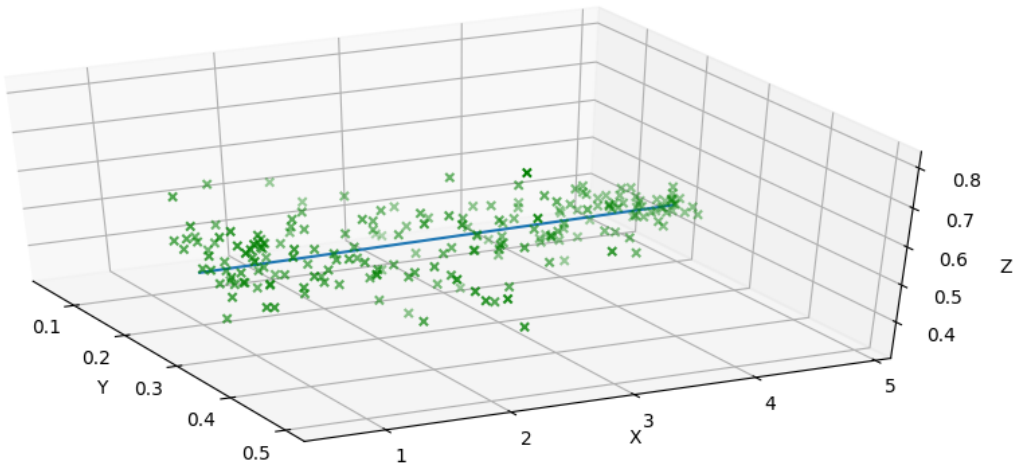


Figure 4.18: A scatter plot of the position data in the first noise test. The blue line is fitted to the 3D-points using the first principal component.

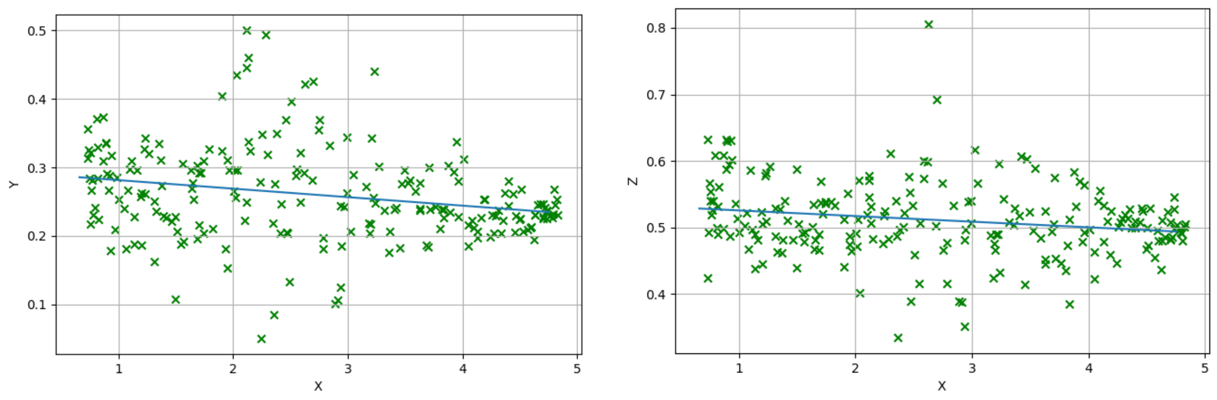


Figure 4.19: The result of the positioning in the first noise test. To the left is the result in xy-plane and to the right is the same result but in xz-plane.

In the second test, 27 images taken with the same camera pose were used. Only very small differences in light between the images could be seen. Those differences depended on the quality of the camera. The position data was analysed by computing the standard deviation in the three dimensions separately

$$\sigma_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{x} - x_i)^2} = 0.0463\ m, \tag{4.5}$$

$$\sigma_y = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{y} - y_i)^2} = 0.0554\ m, \tag{4.6}$$

$$\sigma_z = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{z} - z_i)^2} = 0.0497\ m. \tag{4.7}$$

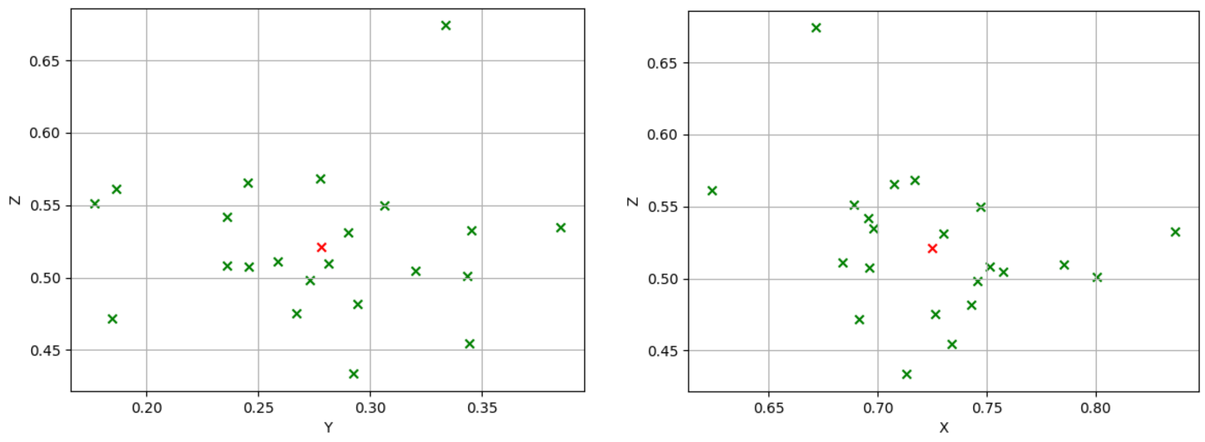


Figure 4.20: Scatter plot of the result of the second noise test. The red marker in the center is the mean of the data points. The two plots show the same result but from two different angles.

Chapter 5

Discussion

The discussion has a similar disposition as chapter 3 and 4. The methods and results from creating the point cloud map and from the localization algorithm are discussed in separate sections. All substeps from when creating the point cloud map are thoroughly discussed in separate subsections. When discussing the results from these sections, all results from both data sets are used to evaluate the methods. Calibration, Crazyflie and flow deck are discussed in separated sections, while the rest of the interesting results and methods are covered in the introduction 5.1. There are also sections to discuss and evaluate the noise tests and the run time of the localization algorithm. In the end of the chapter, general problems about the procedure are discussed.

5.1 Introduction

The result of the procedure to solve the global localization problem was overall good.

It was possible to implement a solution on the Crazyflie with the hardware in Table 3.1 and the restrictions that Crazyflie entails, see section 1.5. Since data had to be collected while holding the drone in the hand, the battery was not used as if the Crazyflie would fly. Therefore, it was not evaluated if the battery time restriction caused a problem.

It was possible to implement a solution with the limited camera quality. However, the quality did affect the result. The localization algorithm performed better when the drone moved closer to objects. The quality of images are better when the objects are closer, which means that the features are more clear. The system was only tested in daylight. The image quality was not tested nor evaluated in other lighting.

5.2 Crazyflie and flow deck

Through out the process of implementing the solution, other problems and limits related to the EKF and the flow deck were discovered. One big problem was that the EKF sometimes got very erroneous pose estimates. This seemed to happen when the drone moved fast in z-direction, e.g when the drone was picked up from the ground. The position estimates in x- and y-direction could obtain errors with magnitude up to one meter. This problem was avoided by starting collecting the data when the drone was already up in the air.

Big errors in the pose data also arised when the drone rotated around the z-axis. The EKF did not estimate the rotation perfectly and therefore did the estimate drift away more and more from the true position in x- and y-direction every time the drone rotated. Therefore did the drone not rotate when collecting data for creating the map. The descriptors representing the points in the map are therefore very one sided, since the point of view are very similar between images. This can cause the algorithm to only work well when the drone moves in a similar trajectory as when it collected the data. When the drone follows a different trajectory, it might not recognize the features in the world anymore.

Using the flow deck also entailed a problem. The ToF sensor measures the absolute distance in negative z-direction in the local reference frame of the drone. If the drone would fly over an object when collecting data for creating the point cloud map, then the pose data contains big errors and the scale get falsely estimated. This can of course be easily avoided by not moving over objects. However, there might be environments where this is impossible.

5.3 Calibration

The result of calibration was visually evaluated in Figure 4.1. The bottom three images show that the lines and corners have been straightened. The reprojection error was small in both x- and y-direction. This signifies that the camera and distortion model were well estimated.

The transformation matrix between the camera's and IMU's reference frame, was verified by measuring the distance between the camera center and IMU center in all directions. For this, a ruler was used. The precision of the measurements was not that high. However, high enough to be able to state that the translation between the camera's and IMU's reference frames are well estimated. Also the rotation seems to be correct, which can be stated by comparing the rotation vector of the rotation, see equation 4.2, with the orientation of the two reference frames, see Figure 3.7 how the coordinate systems are defined.

5.4 Create point cloud map

It was difficult to evaluate if the point cloud map corresponded well to the environment it should represent. The two data sets that were used were different in their complexity. In contrary to the *Kitchen* data set, the points in *Pictures* data set lied in a plane and had a clear structure, which made it easier to identify which points in the cloud corresponded to a specific image/card on the wall. This was mainly due to the white background, which did not cause any background blur. The *Pictures* data set was also smaller than the *Kitchen* data set. Therefore, could the data set be better and more accurate be evaluated.

It would be easier to evaluate if the map is correct if the system would include a feedback control loop to the drone. If the scale and/or rotation of the cloud are falsely estimated it would be easily detected, since the drone would not fly in the same direction and trajectory as desired in the map.

5.4.1 Rotate

Of all methods in creating the point cloud map, the rotate-method was most difficult to evaluate if it worked perfectly. Since *Pictures* data set was flat, it was possible to visually evaluate if the point cloud was completely vertical after rotating it. In Figure 4.5 the point cloud before and after it had been rotated, scaled and translated can be seen. It is clear that the point cloud is vertical.

In the *Kitchen* data set, the estimated camera poses from COLMAP were used to visually evaluate the rotate-method. As can be seen in Figure 4.10, the pose estimate from the EKF (blue line) and the camera pose estimates (red markers) seem to follow each other very well. Also the difference between the first and last plot is significant.

The idea of how to estimate the desired rotation (and scale) is based on the known transformations from local drone to camera to PC coordinate systems. The transformation between local drone and camera has already been evaluated in the section 5.3 above. The transformation between camera and PC coordinate system comes from the estimated camera poses from COLMAP. If the estimated orientation (or position when estimating the scale) made by COLMAP is a little off, then also the error propagates to the estimated gravity direction. However, COLMAP optimizes both camera poses and reconstructed points in the BA step, which should give reliable estimates.

An important assumption in the procedure of estimating the rotation, was that the drone moved with constant velocity. This is of course impossible, since the drone has to accelerate when starting, landing and turning. The contribution of the gravity is a lot bigger than any contribution of acceleration in the movement. It was never closer investigated if the data from the accelerometer contained influences of the movement or other noise sources.

To get a more reliable gravity direction estimate, one of the following procedures are suggested: implement pre-integration theory to minimize IMU errors (which was done in [12]), add more measurements (use more images in the reconstruction), identify noisy/false measurements and remove them. The latter is discussed below.

To identify false measurements, the euclidean norm could be computed of the acceleration measurements. If the norm of a measurement deviates too much from 9.81 the measurement could be ignored. The tolerance should not be bigger than the known sensor noise (known from sensor producer). It is not possible to assume that the contribution of the gravity is in positive z-direction of the local reference frame of the drone. This is because the drone might be tilted when collecting the data. Hence, no ground truth of the gravity direction exists in the local reference frame of the drone.

5.4.2 Scale

The scale factor in *Pictures* data set was better estimated than in *Kitchen* data set. In Table 4.3 the difference between the largest and smallest x-value respectively z-value of the points were computed. The differences were compared to the real world, which was measured using a ruler. As can be seen in the table, the difference between world and point cloud map is not that big. The error is not constant. This can depend on inaccuracy in measurements when using a ruler or that the points in the cloud were falsely matched to the real points in the world.

In *Kitchen* data set, it was more difficult to identify which real point in the world a point in the cloud corresponds to. The environment was both bigger and had one more dimension, which made it difficult to measure and identify the points in the real world. Therefore, no ground truth exists when evaluating the scaling. However, the Figure 4.9 indicates that the scaling is not as accurate as in the other data set. The distance between floor and ceiling is roughly estimated to be 2.9 m in the point cloud map. The same distance in the world is 2.4 m. By looking at Figure 4.10, it is clear that the pose data from both EKF and COLMAP follow each other very well, meaning that the scale estimate seem to be very accurate. The two observations are contradicting each other. Either the pose estimate from the EKF contains errors, or the reconstruction made by COLMAP contains erroneous points close to the floor and ceiling. The latter could be because both the floor and ceiling have a patterned structure. Detected features in these areas can then be confusing for COLMAP and 3D-points are added to the point cloud which should not be there.

Another idea why the scale factor was better estimated in *Pictures*, is that the drone moved in zy-plane instead of xy-plane. Because of the ToF sensor, the pose estimates get more accurate.

5.4.3 Translation

The result from translating the point cloud was also better for the *Pictures* data set compared to the *Kitchen* data set. In Table 4.3 the distance to the floor in the map and in the world are compared for the *Pictures* data set. The difference is 0.048 m. A similar measure was not generated in the *Kitchen* data set because of the same reason stated in section 5.4.2. However, there are points located below the $z = 0$ limit, which means that the floor level of the map is not the same as the world. In the translation-method, the pose data from ToF sensor is used to translate the point cloud in z-direction, therefore should no points be located under this limit. Since the scale of the point cloud does not seem to be correct, then also the translation will be false.

5.4.4 Find ORB-descriptors

It was possible to pair ORB-descriptors with 3D-points, which COLMAP created by detecting SIFT-features. By looking at the tables 4.4 and 4.5 it is clear that the SIFT→ORB-set, which is intersection between the SIFT- and ORB-set, is not that big in comparison to the other two. In *Pictures* and *Kitchen* approximately 5.6% respectively 3.7% of the found ORB-descriptors in an image were used to create the map. For both data sets, approximately 20% of the visual 3D-points in an image got an assigned descriptor.

In Figure 4.12, the three sets are visualized in an example image from the image subset used to create the point cloud map in *Kitchen*. The SIFT-, ORB-, SIFT→ORB-set are marked with green, blue and red markers respectively.

It was mentioned in section 3.2.5, that it was desired to find as many ORB-features as possible. The idea was that the intersecting set would increase in size and thereby increase the number of points in the map and also increase the accuracy of the pose estimate. However, in Figure 4.12 the detected ORB-features are located very close to each other, while the SIFT-features are more widely spread, meaning that the positional intersection between the two sets will most likely not increase by extracting as many ORB-features as possible. Since ORB is designed to detect corners while SIFT is designed to detect blobs, it is obvious that the

the two extractors will have a limited overlap of features. Therefore, it might not be a good idea after all to extract that many ORB-features. The environment and the camera quality limits the amount of good corner features in an image. In both data sets the extractor tried to find 3000 ORB-features. However, in neither data sets there was an image with that many features. If the extractor is forced to find that many, the quality of the features will decrease. It might be the case that a 3D-point get a “bad” descriptor assigned to it when a descriptor, describing the feature better, could have be used instead. Therefore, it could be a good idea to decrease the number of extracted ORB-features. This could also have an impact on the run time, which will be further discussed in section 5.5.1.

5.5 Localization algorithm

The overall result from the localization algorithm was good. The estimated positions of the drone formed a trajectory, which looked very similar to how the drone moved in the environment when collecting the data. In Figure 4.13 the estimated trajectory from the *Pictures* data set is shown. Not only the shape of the trajectory looks correct but also how the drone has moved in relation to the map. The same result was for the *Kitchen* data set which can be seen in Figure 4.15. The estimated trajectories also follows the pose data from the EKF and the pose estimates made by COLMAP very well, see the Figures 4.14 and 4.16. However, the estimated trajectories in both data sets are not completely smooth. The position estimates in *Kitchen* vary a lot more than in the other data set. The average percentage of inlier matches was a lot higher for *Pictures* (86.8%) than for *Kitchen* (59.0%). It could be that the features in *Pictures* are a lot more diverse than in *Kitchen*. In the latter, there are many repeating corners in the structure of the images. For example, by looking at the example image in Figure 4.8, the similar corners from the tiles at the left wall or the small window corners of the cabinet in the front of the camera could be confusing when trying to match ORB-descriptors to the 3D-points. Also the camera quality makes a difference here. The resolution is better for images closer to the object than for images far away. That could also explain why the estimated trajectory (green jagged line) in Figure 4.16 is very smooth in the far right in the plot, i.e. when the camera is located very close to the cabinet.

The orientation of the pose estimates could not be separately evaluated. This was because it is harder to estimate a ground truth. Therefore, there are no measure of how accurate the estimate was. However, since the pose was used to back project point when i.a. computing the subcloud, it is known that the estimate was good enough for the algorithm to keep generating ok estimates. The algorithm never got “lost”, i.e. the moved distance was never larger than 0.6 m.

It is possible that the accuracy of the algorithm could be increased if it is more closely investigated how tight the constraints can be, which are caused by tolerances, thresholds and parameters in the four main steps in the algorithm; compute subcloud, Brute force match, back project and match and pose estimation. The referred variables are for example; the tolerance $\epsilon = 10$ pixels when computing the subcloud, the threshold of 80 hamming distance for removing bad matches, the search radius of 25 pixels when trying to find a match when back projecting points or the allowed reprojection error of eight pixels in the RANSAC step when estimating the pose. The specific values were all set to something that was considered reasonable. None of the values were optimized in any way. By finding better optimized values, then the accuracy might increase and the changes might also affect the run time. The latter will be further discussed in section 5.5.1.

Also the moved distance threshold (0.6 m) in the algorithm was not optimized. The threshold is very high. Since the camera has a frame rate of 25 fps, an image is taken every 0.04 seconds. If the drone would move 0.6 m in that time, then the velocity would be about 15 m/s, which is of course unreasonable. The largest moved distance in *Kitchen* data set was 0.507 m. This measure is therefore very clear to be erroneous, which can also be stated by looking at the biggest spikes in the trajectory in Figure 4.16. Therefore should the limit have been lower. Since the moved distance never was over 0.6 m, the Brute force match method was not thoroughly tested.

5.5.1 Run time

To evaluate the total run time of the algorithm and to answer the question stated in section 1.6 mainly the *Kitchen* data set is used, since the environment is more representative for the application. However, both data sets are used to determine where in the algorithm the bottleneck is located and what mainly causes it. For this the run time tables (4.6 and 4.7) will be used. The discussion below will just touch upon the subject of improvements to speed up the algorithm. The subject will instead be covered in section 6.2.

By comparing the tables 4.6 and 4.7 it can be seen that the run time for initialization, read and undistort image, extract features and compute subcloud are approximately the same for both data sets. The small difference depends most likely on the different map sizes and how many features were extracted. Matching descriptors took significantly the most time to compute. In the tables both matching methods are put together, i.e. Brute force match and back project and match. However, since the algorithm never gets “lost”, mainly the run time is computed for back project and match. It took longer time to compute matches in *Kitchen* than in *Pictures*. Since the camera in *Kitchen* is always pointing in the direction of the length of the room, almost the whole point cloud is also contained in the subcloud. Therefore, many more points and descriptors are used when determining matches. In back project and match, the distance between a back projected point and all ORB-descriptors is computed. If many points are back projected then this operation is conducted many more times than if the subcloud would be smaller. Contrary in *Pictures* the camera moved so that about a third of the point cloud was visible. The subcloud got significantly smaller than the whole cloud, which resulted in that the match step was computed tremendously faster. This result confirms that it is worth computing the subcloud before moving on to the match step. The last step in the algorithm, estimating the pose, was very fast for both data sets. However, there is a relative big difference between then two data sets. This probably again depends on the size of the subcloud. Iterating over a bigger data structure takes more time than over a smaller.

The average total run time for one loop in *Kitchen* was 0.293 seconds. If the drone would move in what is considered normal walking pace 1.4 m/s, the drone would move 0.41 m between two poses. This is obviously not fast enough. The drone would not be able to follow a trajectory accurately and crashing into object would be inevitable. In a home alarm application, it is important that the drone can get to a desired position fast, both considered the battery time limit of the Crazyflie but also so that the alarm gets verified fast to stop a possible break in. Therefore, it is not wrong to desire a higher velocity than 1.4 m/s. In other words, it is of great importance to speed up the algorithm before it can work properly in a home alarm application.

5.6 Noise test

The results from both noise tests were good considering the parameters have not been optimized in the algorithm. The Figure 4.17 shows the result in comparison to the map. The trajectory is similar to the trajectory moved in the world. The average distance to the estimated trajectory, i.e. the line fitted to the points see Figure 4.18, was 0.0615 m. The estimated error is relatively small.

In Figure 4.19 it is clear that the line is not completely constant in z-dimension. The difference in start and end position of the line in z-direction was 0.0266 m. The small tilt do not have to mean that the point cloud map has wrong orientation, but it can depend on the position estimates are more accurate in the end of the trajectory compared to the start. In the end of the trajectory, the velocity decreased even though the velocity was tried to be kept constant. More estimates are therefore very close to each other, which makes the line better estimated to these points. The spread of the z-coordinate of the line was (0.498 m - 0.525 m), which is very close to the distance to the floor in the world.

In the second noise test, the x, y and z were separated to see if there was any difference in variation in the three dimensions. By looking at the results stated in the equations 4.5, 4.6 and 4.7, there was a slight difference between the standard deviations in the three dimensions. Since the difference was not significant, there is not that much to discuss.

Since the start poses of the both tests were not known, compared to in the result of the *Pictures* and *Kitchen* data sets, the algorithm had to estimate the start pose by using the Brute force match method. The method succeeded in estimating the start pose, which is clear in the Figures from the noise tests. There is no position estimate in the beginning which is more erroneous than the other estimates.

Note that in both tests, the position estimates are used to both estimate the ground truth and to evaluate the test.

5.7 General problems with main idea

There are some more general problems with the procedure presented in this master thesis of solving the global localization problem, which not yet have been mentioned.

A big problem with the procedure, is that the map is built for a specific scenery. Objects like chairs and decorations will be moved in the every day life. If a point in the map correspond to a movable object, then either the map will contain redundant points if the object is removed or the map will contain errors if the position of the object is slightly changed. The former case does not create a problem, however the redundant points could be removed and thereby speed up the run time. The latter case can cause bigger problems. Especially if many objects are moved. Then either the pose get falsely estimated or no solution is found, since many point correspondences are geometrically wrong. This problem is called the scene change detection and map updating problem.

Chapter 6
Conclusion

6.1 Conclusion

The purpose of this thesis was to establish global localization, i.e. estimate a drone's position and orientation in a previously mapped indoor environment by using a monocular camera and computer vision. Three main goals were stated to fulfill this purpose, i.e. build a system to collect data, create a point cloud map and determine the drone's position and orientation using this map. All three goals were achieved for two data sets, which imply that the procedure definitely works to some extent. It was difficult to evaluate all different details of the procedure. However, the parts which could be closer evaluated indicated a stable result with room for improvements.

Since the solution presented in this thesis is developed and adapted to a home alarm application, a conclusion can be drawn that the global localization problem is not an obstacle when developing a home alarm application using a drone to verify alarms. However, the procedure needs to be further tested and there are still many problems to solve before a working system can be installed in a home.

6.2 Further research

Since only two data sets were used to test and evaluate the procedure, the result is biased. To really know how well it performs or to identify weaknesses and other problems, the procedure needs to be further tested. It is suggested to test the procedure using new data sets. Both from the same environment used in the thesis and from new environments. The former is to investigate how sensitive it is to changes in the scenery, different lighting and if it can handle different view points of the scene. The latter is to investigate how broadly it can be used and to identify not yet known weaknesses. Another interesting test, would be to use the Loco positioning system[4], mentioned in section 1.2, to define a ground truth for better evaluation of both the point cloud map and the localization algorithm.

It is also suggested to add a control loop to the system, the procedure could then be more thoroughly tested in real time. This step is also a vital to implement to obtain a working home alarm system application.

As mentioned in the conclusion, the procedure has room for improvements. The most important to mention are; to make the algorithm more accurate, stable and faster. To increase the stability, i.e. to prevent the position estimate to jump uncontrollably, it is suggested to add a motion model and to fuse IMU data into the system. Then pose estimates which are unreasonable can be rejected. The drone's trajectory can then also be smoother and more accurate than the trajectories in the result presented in this thesis. Then maybe the simple structure of how the localization algorithm was divided by the threshold of 0.6 m can be removed. Another suggestion to increase the accuracy of the pose estimate, is to tune the parameters, which was discussed in section 3.3.

In the discussion in section 5.5.1, it was stated that the run time bottleneck of the algorithm was the matching step, since many comparisons are made between two big sets of descriptors. One idea to speed up the algorithm is therefore to optimize the number of descriptors in both the map and when extracting features in the algorithm. It was stated in section 5.4.4 that it was very excessive to try to extract 3000 features. Therefore, it could be investigated how small this number could be without jeopardising an accurate pose estimate.

Another idea is to implement a method which keep track of how often a descriptor in the point cloud map is used in an inlier match. Descriptors, which are seldomly or never used, could be removed from the map. This would reduce redundant information in the map.

Before the application can be installed in a home, some other problems stated in the section 1.1 have to be solved, i.e. path planning, collision avoidance, flight and landing control and also the scene change detection and map updating problem, mentioned in section 5.7.

References

[1] Bitcraze. *CrazyFlie 2.1*. https://www.bitcraze.io/products/crazyflie-2-1/. [Online; accessed 2020-07-21].

[2] Bitcraze. *Crazyradio PA*. https://www.bitcraze.io/products/crazyradio-pa/. [Online; accessed 2020-07-21].

[3] Bitcraze. *Flow deck v2*. https://www.bitcraze.io/products/old-products/flow-deck/. [Online; accessed 2020-07-21].

[4] Bitcraze. *Loco*. https://www.bitcraze.io/products/loco-positioning-system/. [Online; accessed 2020-08-10].

[5] Abraham Bookstein, Vladimir Kulyukin, and Timo Raita. "Generalized Hamming Distance". In: *Information Retrieval* 5 (Oct. 2002). DOI: 10.1023/A:1020499411651.

[6] Brå. *Bostadsinbrott*. https://www.bra.se/statistik/statistik-utifran-brottstyper/bostadsinbrott.html. [Online; accessed 2020-07-29].

[7] Michael Calonder et al. "BRIEF: Binary Robust Independent Elementary Features". In: *Eur. Conf. Comput. Vis.* 6314 (Sept. 2010), pp. 778–792. DOI: 10.1007/978-3-642-15561-1_56.

[8] Andrew Davison et al. "MonoSLAM: real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29 (July 2007), pp. 1052–67. DOI: 10.1109/TPAMI.2007.1049.

[9] Eachine. *ROTG02*. https://www.eachine.com/Eachine-ROTG02-UVC-OTG-5_8G-150CH-Dual-Antenna-Audio-FPV-Receiver-for-Android-Tablet-Smartphone-p-1063.html. [Online; accessed 2020-07-21].

[10] 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 (June 1981), pp. 381–395. DOI: ACM0001-0782/81/0600-0381.

[11] John Folkesson. "Simultaneous Localization and Mapping with Robots". In: *KTH* (Jan. 2005).

[12] Christian Forster et al. “On-Manifold Preintegration for Real-Time Visual-Inertial Odometry”. In: *IEEE Transactions on Robotics* PP (Aug. 2016). DOI: `10.1109/TRO.2016.2597321`.

[13] P. Furgale, T. D. Barfoot, and G. Sibley. “Continuous-time batch estimation using temporal basis functions”. In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 2088–2095.

[14] Paul Furgale, Joern Rehder, and Roland Siegwart. “Unified temporal and spatial calibration for multi-sensor systems”. In: *Proceedings of the ... IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems* (Nov. 2013), pp. 1280–1286. DOI: `10.1109/IROS.2013.6696514`.

[15] Xiao-Shan Gao et al. “Complete Solution Classification for the Perspective-Three-Point Problem”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 25 (Sept. 2003), pp. 930–943. DOI: `10.1109/TPAMI.2003.1217599`.

[16] Ian T. Jolliffe and Jorge Cadima. “Principal component analysis: a review and recent developments”. In: *Philos Trans A Math Phys Eng Sci* 374(2065) (Apr. 2016). DOI: `10.1098/rsta.2015.0202`.

[17] Juho Kannala and Sami Brandt. “A Generic Camera Model and Calibration Method for Conventional, Wide-Angle, and Fish-Eye Lenses”. In: *IEEE transactions on pattern analysis and machine intelligence* 28 (Sept. 2006), pp. 1335–40. DOI: `10.1109/TPAMI.2006.153`.

[18] Georg Klein and David Murray. “Parallel Tracking and Mapping for Small AR Workspaces”. In: *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality* (Dec. 2007), pp. 225–234. DOI: `10.1109/ISMAR.2007.4538852`.

[19] Shimiao Li. “A review of feature detection and match algorithms for localization and mapping”. In: *IOP Conference Series: Materials Science and Engineering* 231 (Sept. 2017), p. 012003. DOI: `10.1088/1757-899x/231/1/012003`. URL: `https://doi.org/10.1088%2F1757-899x%2F231%2F1%2F012003`.

[20] David Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–. DOI: `10.1023/B:VISI.0000029664.99615.94`.

[21] Jérôme Maye, Paul Furgale, and Roland Siegwart. “Self-supervised Calibration for Robotic Systems”. In: *IEEE Intelligent Vehicles Symposium, Proceedings* (June 2013). DOI: `10.1109/IVS.2013.6629513`.

[22] Jérôme Maye et al. *Kalibr*. `https://github.com/ethz-asl/kalibr/wiki`. [Online; accessed 2020-07-23].

[23] Michael Montemerlo et al. “FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges”. In: *Proc. IJCAI Int. Joint Conf. Artif. Intell.* (June 2003).

[24] Raul Mur-Artal, J. Montiel, and Juan Tardos. “ORB-SLAM: a versatile and accurate monocular SLAM system”. In: *IEEE Transactions on Robotics* 31 (Oct. 2015), pp. 1147–1163. DOI: `10.1109/TRO.2015.2463671`.

[25] Raul Mur-Artal and Juan Tardos. “Visual-Inertial Monocular SLAM with Map Reuse”. In: *IEEE Robotics and Automation Letters* PP (Oct. 2016). DOI: 10.1109/LRA.2017.2653359.

[26] Sofie Olsson. *GlobaLoco*. https://github.com/sofiol/GlobaLoco.

[27] *OpenCV*. https://opencv.org/about/. [Online; accessed 2020-07-31].

[28] OpenCV. *VideoCapture*. https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html. [Online; accessed 2020-07-23].

[29] Open Robotics. *ROS*. http://wiki.ros.org/Documentation. [Online; accessed 2020-07-21].

[30] Edward Rosten and Tom Drummond. “Machine Learning for High-Speed Corner Detection”. In: *Comput Conf Comput Vis* 3951 (July 2006). DOI: 10.1007/11744023_34.

[31] Ethan Rublee et al. “ORB: an efficient alternative to SIFT or SURF”. In: *Proceedings of the IEEE International Conference on Computer Vision* (Nov. 2011), pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[32] RunCam. *nano2*. https://shop.runcam.com/runcam-nano-2/. [Online; accessed 2020-07-21].

[33] RunCam. *TX200U*. https://shop.runcam.com/runcam-tx200u/. [Online; accessed 2020-07-21].

[34] Johannes Lutz Schönberger and Jan-Michael Frahm. “Structure-from-Motion Revisited”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[35] Johannes Lutz Schönberger et al. “Pixelwise View Selection for Unstructured Multi-View Stereo”. In: *European Conference on Computer Vision (ECCV)*. 2016.

[36] Peter Sturm. “Pinhole Camera Model”. In: *Computer Vision: A Reference Guide*. Boston, MA: Springer US, 2014, pp. 610–613. ISBN: 978-0-387-31439-6. DOI: 10.1007/978-0-387-31439-6_472. URL: https://doi.org/10.1007/978-0-387-31439-6_472.

[37] Shaharyar Ahmed Khan Tareen and Zahra Saleem. “A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK”. In: Mar. 2018. DOI: 10.1109/ICOMET.2018.8346440.

[38] Deepanshu Tyagi. *Introduction To Feature Detection And Matching*. https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d. [Online; accessed 2020-08-21].

[39] Zhengyou Zhang. “A Flexible New Technique for Camera Calibration”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22 (Dec. 2000), pp. 1330–1334. DOI: 10.1109/34.888718.

Master's Theses in Mathematical Sciences 2020:E70
ISSN 1404-6342

LUTFMA-3428-2020

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

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