Camera based localization and autonomous navigation for a flying drone

Master's thesis in Complex Adaptive Systems

MATTIAS KJELLTOFT

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Camera based localization and autonomous navigation for a flying drone

MATTIAS KJELLTOFT

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
The Computer Vision Group
Chalmers University of Technology
Gothenburg, Sweden 2019

Camera based localization and autonomous navigation for a flying drone
MATTIAS KJELLTOFT

Supervisor: Carl Toft, Department of Electrical Engineering
Examiner: Fredrik Kahl, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
The Computer Vision Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A picture of Parrot Bebop 2, the drone used in the project.

Typeset in LaTeX
Gothenburg, Sweden 2019

Camera based localization and autonomous navigation for a flying drone
Mattias Kjelltoft
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This report presents a system built with the goal of making a flying drone able to localize itself and navigate autonomously to a waypoint in a known space. The localization is based upon computer vision and uses technology such as SIFT, KD trees, RANSAC, projection geometry and structure from motion. The various algorithms used are described briefly in the theory section of this report. Furthermore, the system architecture is described and a simple evaluation of the system is presented. The evaluation shows that the drone is able to autonomously navigate to a waypoint and thus there is a proof of concept, even though there still exist lots of potential for further development. Finally some ideas for further development of the system are presented.

Keywords: computer vision, drone, projective geometry, structure from motion, RANSAC, autonomous flight

Acknowledgements

I would to thank my supervisor and examiner for having shown a great deal of patience with me as the project have taken a long time. I would also like to thank my friends and family for nagging me just the right amount to make me finish the project.

Mattias Kjelltoft, Gothenburg, September 2019

Contents

List of Figures

List of Tables

1
Introduction

The problem of localization is central in many applications. We can find examples in areas such as the autonomous steering of cars or space exploration rovers, the tracking of the head mounted display in virtual reality equipment and even in special effects in video editing and animation for movies where tracking the cameras movement to place virtual objects stably in a scene is vital. As soon as you have some kind of automated system that needs to react to movement and positions, you need to be able to track that movement and find those positions. Add to this the problem of navigation, that is the problem of choosing a good route given a map containing a start location and a goal location, and a machine to actually try to follow this route and we have arrived at an interesting control problem.

This master thesis projects aim was to integrate a camera as the main localization sensor in a control loop for higher level navigation of a flying drone. The goal was to give the drone the ability to autonomously navigate a known indoor environment. The camera localization methods used are not novel. My work has been about understanding them and integrating them in a simple control loop.
The project was focused on the methods and models used in the analysis of the images from the drones camera and in the navigation where the drone uses the information gained in the image analysis to control its flight. This focus prompted the buying of a commercially available drone to the project rather than building a custom one, so that less time was spent on making the hardware work. The drone bought for the project had an interface for basic control available, but the software to make it fly to a specific position and angle with respect to the camera localization had to be created.
An example of a practical application that can be mentioned to justify the study of indoor localization and navigation is the problem of taking inventory of a supermarket. The research group of my examiner and supervisor was involved in a project aiming to build a demonstrator with a flying drone performing this task. That project was called “Semantic mapping and visual navigation for smart robots”.

There are several possible alternative choices of sensors for localization.
A LIDAR is a quickly spinning laser that takes a large number of distance measurements in a circle around itself, producing a point cloud. It is quite similar to a radar but with a higher frequency of the light it emits. Since it provides a point cloud image of its surroundings the sensor can be used both for localization of itself but also for identifying other objects. It is frequently used in the automotive industry’s effort to make autonomous cars, either as a ground truth to evaluate other sensors or

in some cases as the intended primary sensor for the vehicle. Some of its advantages are a high accuracy and precision, it can see in the dark and is not fooled by shadows since it provides its own light and it does not require a lot of computation to get the point cloud since the sensor is based on direct measurements. Some drawbacks are that the sensor is heavy, bulky and expensive. It is also not clear if there are long term safety risks involved with constantly bombarding everything in our environment with lasers and it would certainly be tricky to handle the interference and blinding problems if all road vehicles relied on emitting lasers [1]. The LIDAR is the only real contender to the camera sensor for both indoor navigation and object detection, but it was not used in this project due to its heavy and bulky format, its price and the fact that the author had a greater interest in image processing algorithms than in LIDAR.

Another common sensor is the GPS/GNSS. It works by listening to signals from satellites in orbit around the earth, computing the distances to the satellites from the timestamps of the signals using time of flight and then computing its own position as the intersection of the spheres around the satellites defined by those distances. The GPS has the excellent property of global coverage and with techniques such as Real Time Kinematics the accuracy can be improved to sub-centimeter with the addition of a stationary reference device. One big disadvantage of the GPS though is that it requires line of sight to the satellites in space. This makes the sensor unreliable indoors and in tunnels, but also in places such as canyons and big cities with tall buildings. The GPS was not used in this project since the drone was supposed to fly indoors.

A third sensor that is common is the Inertial Measurement Unit. The IMU measures its own acceleration and can then integrate this value to get the path traveled. The time between measurements is very short and the technique is quite good for tracking the short term movement of an object, but since it does not compare the path to any environment it is prone to drift over time. This sensor is used in this project to let the drone travel short distances with a global localization method in between those "jumps" to correct for the drift.

The fourth and last sensor that are going to be discussed here is the camera, the main sensor used for localization in this project. The camera is a sensor rich in information and we should in theory be able to extract all the same information from an image that we can get by looking with human eyes. Distinct points can be tracked between images and through that stereoscopic vision can be achieved which gives us a point cloud that can be used for localization, something that is used a lot in this project. The advances in neural networks has improved our ability to find and classify objects in images, which is not used in this project but could be useful for object detection and collision avoidance. Disadvantages of the camera as a sensor include the reliance on good light conditions and the comparatively high demand of computational power. The camera is also reliant on more complex algorithms than for example the direct distance measurements of the LIDAR and the problems are thus generally harder and less understood, but have a greater potential for more information content.

In the second chapter of this report the theoretical background of the localization

technique used is described with sections on topics such as the camera model, the solution to the perspective-3-point-problem and the RANSAC algorithm. In the third chapter the architecture of the system that was built is described, while the fourth chapter provides the results of the testing and evaluation of the system. In chapter five a short conclusion of the project is presented.

2
Theory

Assume that two images of the same object are captured from different angles. If we can recognize the projection of the same real world point in the two images, what we call a point correspondence, we can through the mathematical framework of projection geometry calculate the positions of the captured object and the cameras in relation to each other. By using this capability together with a large set of images one can construct three dimensional models of objects and spaces, a process that is usually called *Structure from Motion* (SfM).
In this project SfM has been used in order to construct point cloud models of spaces that then have been used as maps for a drone. The drone has through the use of projection geometry computed its position in relation to the map from the images captured with its camera.
In this section of this report we will provide brief descriptions of how to recognize points in images with SIFT, how to quickly find matches in a large set of points with the help of KD trees, how to effectively exclude outlying points from a set with RANSAC and how to compute positions from correspondences between points in images with by solving the perspective three point (P3P) problem.
The descriptions of the various methods and algorithms will be brief and without great detail, in part to more easily provide the reader with an overview of the methods and in part because the author of this report did not develop these methods himself but rather only used them and therefore does not own a great knowledge of the details and intricacies. References to more complete treatments will be provided for the interested reader.

Recognizing points in images with SIFT

An interest point is an object defined for an image and consist of a position in the image and an associated descriptor. The descriptor is a function of the local texture in the image around the interest point position. By constructing a descriptor that is invariant under a set of transformations of the image such as translation, rotation, scaling, lighting, etc one can find and track the same interest point through several separate images.
If we capture two images of the same object from somewhat different angles we can by matching the descriptors of the interest points in the two images find which pixels that depict the same real point, or at least pixels that depict points that look the same, and through that construct points correspondences between positions in the two images.

The acronym SIFT stands for *Scale Invariant Feature Transform* and is an algorithm that handles interest points. SIFT consist of two parts; one part that finds interest points in an image and another part that assigns descriptors to the points.
The SIFT detector flags points as interest points if they are extreme points of the *difference of Gaussian* (DoG) in scale space. Different scales of the image is obtained by convolving the image with a Gaussian kernel, which more simply put can be described as smoothing the image by blurring it or as a subsampling by averaging. By repeating this process multiple times a pyramid is constructed consisting of the same image in smaller and smaller scale and ever fewer details. The DoG would then be the difference between two layers of that pyramid, corresponding to a band pass filter in level of detail. In order for a point to be flagged as interesting by the SIFT detector it needs to be extreme both in the dimensions of the image as well as in the scale dimension of the pyramid.
The SIFT descriptor is constructed by creating a histogram of all pixel gradients in the vicinity of the interest point. Usually 128 bins are used, $4 \cdot 4 = 16$ bins from dividing the pixels in a grid of 4×4 spatial boxes and the remaining 8 bins from discretizing the gradient angles into 8 bins. The contribution of the pixels are then weighted with a Gaussian kernel and finally the histogram is normalized.
The SIFT descriptor is invariant under rotation, translation and scaling of the image and robust under moderate changes in perspective and lighting. Rotational invariance is achieved by reordering the histogram bins so that the largest bin is always the first, but by keeping the internal order of the bins intact. Translational invariance is achieved directly since only the local pixels close to the interest point are considered. Scale invariance is achieved at the detector stage since interest points are searched for at all scales; the same real point would be reported as a collection of interest points, one for every scale it was detected in.
The matching of SIFT points in the program is a two stage process. Assume that one search point should be matched against a larger set of points. First the search point is pairwise matched again all the points in the set with a regular euclidean distance measure in descriptor space and the two closest matches are stored. Then the angles in descriptor space between the search point and the two neighbors are compared. If the difference between the angular distance to the closest neighbor and the angular distance to the second closest neighbor is sufficiently large the match with the closest neighbor is accepted, otherwise it is rejected. The reason for only accepting sufficiently unique matches is that one would otherwise risk making mistakes when choosing between similar looking alternative matches. If the image where the point was detected contained a repeating pattern it would be impossible to know which part of the pattern that would correspond to the correct match.
For a more complete description of SIFT see the original paper by David Lowe [2] and the article on the subject on scholarpedia [3]. Details about the implementation used in this project can be found on the VLFEAT website [4].

Fast pairwise matching with KD trees

One problem with pairwise matching a set of descriptors is that the number of pairs grows quickly (the number of pairs is $m \cdot n$ where m and n is the number of points in

the two matching sets), which leads to the task of matching all descriptors quickly starts to take a long time. One way to tackle this problem is to use a data structure called a K dimensional tree. (KD tree in short).
A KD tree is a binary tree where every node in the tree is defined by a K dimensional data point. Every node defines an implicit $(K-1)$ dimensional hyperplane dividing the space so that all points to the left of the hyperplane ends up in the left branch of the tree and all points right of the hyperplane ends up in the right branch.
The tree is created iteratively by adding points to it and creating one node with its corresponding implicit hyperplane per point. The hyperplane is created by choosing one of the K dimensions and then placing the plane in the position of the point of its node, normal to the direction of the chosen dimension. The dimension used for each node is chosen cyclically. The tree is constructed by dividing the points into branches until all points are placed as nodes in the tree.
By choosing the median point for every new split when the tree is constructed the tree remains balanced, which means that at every split half of the points end up on each side of the tree making all branches equally large. Searches are generally faster in a balanced tree.
To execute a search in a tree one starts with the search point at the root and travels upward the branches, choosing the route with the same split criteria as when the tree was constructed. The point of the leaf node at the end of the climb up the tree becomes the first candidate as closest neighbor. When a leaf node is reached we start to descend the tree again, updating the neighbor candidate along the way. The leaf node point is probably at least a good approximation of the closest neighbor, but there could potentially be a closer point just on the other side of the splitting plane. Therefore every time during the descent down the tree when the plane is closer to the search point than the stored neighbor that branch is also searched, but if the neighbor candidate is closer than the plane that is not necessary. By skipping searching branches that we know cannot contain better candidates we reduce the number of matches done, but we will still find the guaranteed best match. This kind of pruning of the search tree is usually called *branch-and-bound*.
One can easily create a faster, approximate variant of the search by canceling the search after a fixed number of matches, which in practice still often provides the best match, although not guaranteed, and in other cases often provides a good approximation. One can also find the second closest match by keeping the two best candidates in memory and search branches where the plane is closer than any of them.
For more information about the implementation used in this project see VLFEAT's website [5]. The wikipedia article on KD trees is also very informative.

From 2D to 3D with projection geometry and SfM

Homogeneous coordinates

Homogeneous coordinates is a practical tool when working with projective space. A homogeneous coordinate in 2D space is described by the vector

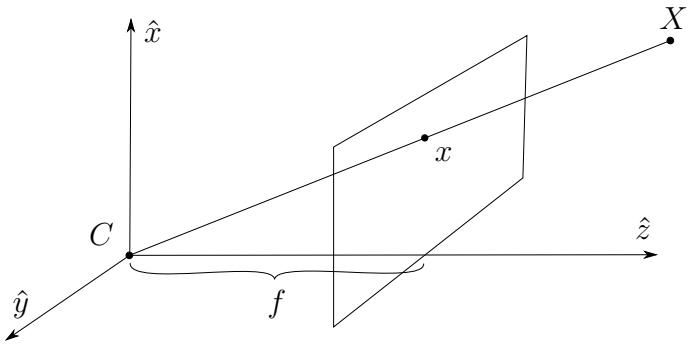

Figure 2.1: The geometry of one camera. The world point X is projected onto the image plane along a line between the point and the camera center C. The image plane is located one focal length f from the camera center in the direction of the principal axis $\hat{z}$, which defines the direction of the camera.

$$\vec{x} = \begin{pmatrix} xw \\ yw \\ w \end{pmatrix}$$

where we have added an extra parameter w at the end. Homogeneous coordinates differing only by a constant factor describes the same point and thus we can always normalize the coordinates to

$$\vec{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Lines are described naturally in homogeneous coordinates by

$$\vec{l} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

and we can easily check if a line intersects a point with the condition

$$ax + by + c = \begin{pmatrix} a & b & c \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \vec{l} \cdot \vec{x} = 0$$

From this condition we see that the line that pass through the points $\vec{x}_1$ and $\vec{x}_2$ is described by

$$\vec{x}_1 \times \vec{x}_2 = \vec{l}_{12}$$

which we will use later when we derive the fundamental matrix F.

One camera view

The pinhole camera model is a simple and commonly used model of how a camera works. The geometry of the pinhole camera can be seen in figure 2.1. In the pinhole

camera model all world points X from 3D space are mapped to 2D points on the image plane by a projection along straight lines through a camera specific point called the camera center. The image plane is the plane located one focal length f away from the camera center with its normal parallel with the cameras principal axis. The principal axis defines the direction of the camera. By using homogeneous coordinates this mapping from 3D world points to 2D image points can be easily described with a transformation matrix as

$$x = KR[I| - C]X = PX$$

where P is called the camera matrix. Note that since the vector X has length 4 but x only has length 3 the matrix P must have dimensions 3×4. The camera matrix defines both the cameras external parameters position and rotation, but also its inner parameters such as its focal length.
We begin with describing the internal parameters. For simplicity, assume that the coordinate system of the camera is aligned with the coordinate system of the world, such that the camera center is located in origo and the axes of the two systems are parallel. Then we have a mapping

$$X = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX/Z + p_x \\ fY/Z + p_y \\ 1 \end{pmatrix} = \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{pmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{pmatrix} X = K[I|0]X = x$$

where the 3×3 matrix K is called the calibration matrix of the camera. We note that f is the focal length of the camera. The parameters p_x and p_y represents the commonly used translation of the coordinate system in image coordinates such that the origin of the image coordinate system is placed in the corner of the image rather than in the center.
The extrinsic parameters, that is the rotation and position of the camera, are described by the transformation matrix

$$X_{cam} = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} X = R \begin{bmatrix} I| - C \end{bmatrix} X$$

which transforms a position from the world coordinate X to the camera coordinate X_{cam}. Here C is the vector describing the world position of the camera center and R is the rotation matrix describing the camera orientation.
If those two operations are put together to first transform from world coordinates to camera coordinates and then perform a projection from 3D to the 2D space of the image plane we end up with the expression

$$x = KR[I| - C]X = PX$$

which is the one that was provided at the start of this derivation.

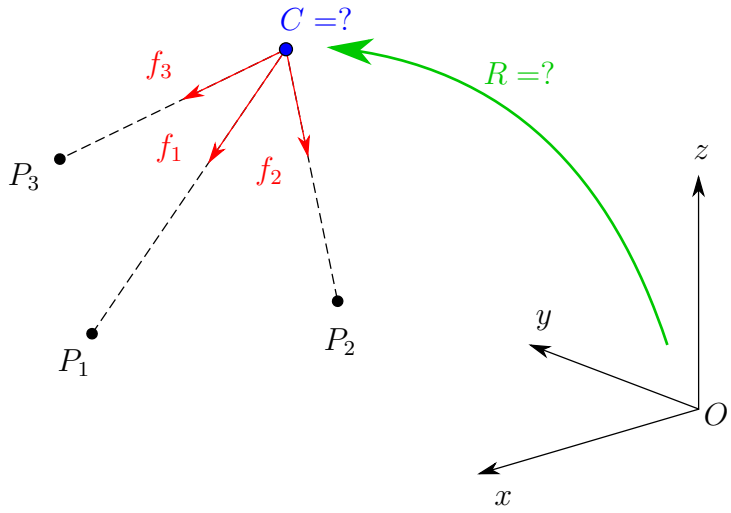

Figure 2.2: The figure illustrates the setup of the perspective three point problem. We know the positions of the three points P_i in the world frame and the three vectors f_i pointing at them in the camera frame. We want to find the position C and rotation R of the camera.

The Perspective-Three-Point problem

The perspective three point problem (P3P) aims to compute the absolute position and rotation of a camera in the world frame given three 2D-3D point correspondences. This is the basis for the camera localization used in this project. In this section the solution given in the paper by Kneip [6] will be summarized. It is his implementation of a fast and accurate solver of the P3P problem that is used in the actual code of this project.
The starting point of the problem can be seen illustrated in figure 2.2.
We know the positions of the points P_1, P_2 and P_3 in the world frame and we know the unit vectors f_1, f_2 and f_3 pointing at those three points in the camera frame. We want to find the position C of the camera center and the rotation R of the camera. In the first step of the solution we define two new frames of reference that simplify the problem for us, one frame τ aligned with the f_i vectors and one frame η aligned with the vectors P_{ij} formed between the points P_i.

$$\begin{aligned} \vec{t_x} &= \vec{f_1} \\ \vec{t_z} &= \frac{\vec{f_1} \times \vec{f_2}}{\left\| \vec{f_1} \times \vec{f_2} \right\|} \\ \vec{t_y} &= \vec{t_z} \times \vec{t_x} \end{aligned}$$

$$\vec{n_x} = \frac{\overrightarrow{P_1P_2}}{\left\| \overrightarrow{P_1P_2} \right\|}$$

$$\begin{aligned} \vec{n_z} &= \frac{\vec{n_x} \times \overrightarrow{P_1P_3}}{\left\| \vec{n_x} \times \overrightarrow{P_1P_3} \right\|} \\ \vec{n_y} &= \vec{n_z} \times \vec{n_x} \end{aligned}$$

This gives us the two new frames of reference

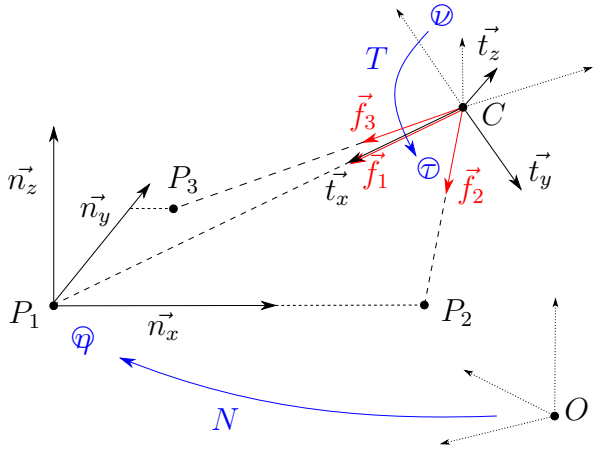

Figure 2.3: As the first step in the solution two auxiliary reference frames are constructed. The frame η is constructed from lines between the points P_i and is accompanied by the transformation N from the world frame. The frame τ is constructed from the vectors $\vec{f}_i$ and is accompanied by the transformation T from the camera frame, here noted as ν.

$$\tau = \left(C, \vec{t_x}, \vec{t_y}, \vec{t_z}\right)$$
$$\eta = (P_1, \vec{n_x}, \vec{n_y}, \vec{n_z})$$

that are illustrated in figure 2.3.
With the transformation matrix $T = [\vec{t_x}, \vec{t_y}, \vec{t_z}]^T$ vectors can be transformed into τ as

$$\vec{f_i^\tau} = T\vec{f_i}$$

while world point can be transformed into η with the transformation matrix $N = [\vec{n_x}, \vec{n_y}, \vec{n_z}]^T$ as

$$P_i^\eta = N(P_i - P_1)$$

The next step in the solution is noting the plane Π defined by the points C, P_1 and P_2 as seen in the figure 2.4, where the variables α and θ are unknown. Much of the remainder of this P3P solution will be spent finding this plane and finding the position of C inside it. First we note that the position of C in the plane Π is given by

$$C^\Pi(\alpha) = \begin{pmatrix} |CP_1|\cos\alpha \\ |CP_1|\sin\alpha \\ 0 \end{pmatrix}$$

and we get $|CP_1|$ from using the sine law as

$$\frac{|CP_1|}{\sin(\pi - \alpha - \beta)} = \frac{d_{12}}{\sin\beta}$$

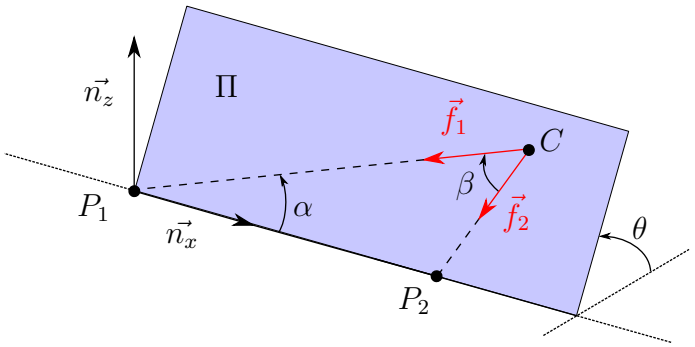

Figure 2.4: The plane Π is defined by the points in the triangle CP_1P_2. Much of this solution for the P3P problem involves finding this plane and the position of the point C within it. The two angles α and θ are unknown.

where d_{12} is known from knowing the absolute positions of P_1 and P_2 and β is known by using the dot product on the f_1 and f_2 vectors. We then continue noting that the basis vectors of τ inside the plane Π are given by

$$\begin{aligned} \vec{\tau_x}^{\Pi} &= (\ -\cos\alpha, \ -\sin\alpha, \ 0)^T \\ \vec{\tau_y}^{\Pi} &= (\ \sin\alpha, \ -\cos\alpha, \ 0)^T \\ \vec{\tau_z}^{\Pi} &= (\ 0, \ 0, \ 1)^T \end{aligned}$$

which we soon will use to define a transformation matrix.
In the next step of the solution we will tackle the rotation of the plane Π around $\vec{n_x}$, defined by the angle θ. The rotation matrix tilting the plane looks like

$$R_\theta = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

and using this matrix R_θ we can now express the position of C inside the η frame as

$$C^\eta(\alpha, \theta) = R_\theta C^{\Pi}(\alpha)$$

as well as the general transformation matrix $Q(\alpha, \theta)$ taking any vector from η to τ as

$$Q(\alpha, \theta) = \left[R_\theta \cdot \left(\vec{\tau_x}^{\Pi}(\alpha), \vec{\tau_y}^{\Pi}(\alpha), \vec{\tau_z}^{\Pi}(\alpha)\right)\right]^T$$

Gathering what we have expressed this far allows us to provide the final position and rotation of the camera as

$$C = P_1 + N^T C^\eta \tag{2.1}$$

$$R = N^T Q^T T \tag{2.2}$$

although we still need to find the values of α and θ.

In order to solve for the values of α and θ we need two conditions. We find those two conditions by transforming P_3 into τ and then impose that the vector $\vec{f_3^\tau}$ must point at this point.

$$P_3^\eta = N(P_3 - P_1) = (p_1, p_2, 0)^T$$

$$P_3^\tau = Q(\alpha, \theta) \cdot (P_3^\eta - C^\eta(\alpha, \theta))$$

We now ensure that f_3^τ points at P_3^τ by imposing

$$\begin{cases} \phi_1 = \frac{f_{3,x}^\tau}{f_{3,z}^\tau} = \frac{P_{3,x}^\tau}{P_{3,z}^\tau} \\ \phi_2 = \frac{f_{3,y}^\tau}{f_{3,z}^\tau} = \frac{P_{3,y}^\tau}{P_{3,z}^\tau} \end{cases} \tag{2.3}$$

that is, the fractions ϕ_i of the components of the vectors must be equal. Doing some arithmetics starting from equation 2.3 and inserting all the relevant expressions from previously in the solution we arrive first at an expression for α as

$$\cot\alpha = \frac{\frac{\phi_1}{\phi_2} p_1 + p_2 \cos\theta - d_{12} \cot\beta}{\frac{\phi_1}{\phi_2} p_2 \cos\theta - p_1 + d_{12}} \tag{2.4}$$

and then by instead eliminating α we end up with the fourth order polynomial

$$a_4 \cos^4\theta + a_3 \cos^3\theta + a_2 \cos^2\theta + a_1 \cos\theta + a_0 = 0 \tag{2.5}$$

where the coefficients a_i are only expressed in known values $\{p_1, p_2, \phi_1, \phi_2, \beta, d_{12}\}$ and looks like

$$\begin{aligned}
a_4 &= -\phi_2^2 p_2^4 - \phi_1^2 p_2^4 - p_2^4 \\
a_3 &= 2p_2^3 d_{12} b + 2\phi_2^2 p_2^3 d_{12} b - \phi_1 \phi_2 p_2^3 d_{12} b \\
a_2 &= -\phi_2^2 p_1^2 p_2^2 - \phi_2^2 p_2^2 d_{12}^2 b^2 - \phi_2^2 p_2^2 d_{12}^2 + \phi_2^2 p_2^4 \\
&\quad +\phi_1^2 p_2^4 + 2p_1 p_2^2 d_{12} + 2\phi_1 \phi_2 p_1 p_2^2 d_{12} b \\
&\quad -\phi_1^2 p_1^2 p_2^2 + 2\phi_2^2 p_1 p_2^2 d_{12} - p_2^2 d_{12}^2 b^2 - 2p_1^2 p_2^2 \\
a_1 &= 2p_1^2 p_2 d_{12} b + 2\phi_1 \phi_2 p_2^3 d_{12} \\
&\quad -2\phi_2^2 p_2^3 d_{12} b - 2p_1 p_2 d_{12}^2 b \\
a_0 &= -2\phi_1 \phi_2 p_1 p_2^2 d_{12} b + \phi_2^2 p_2^2 d_{12}^2 + 2p_1^3 d_{12} \\
&\quad -p_1^2 d_{12}^2 + \phi_2^2 p_1^2 p_2^2 - p_1^4 - 2\phi_2^2 p_1 p_2^2 d_{12} \\
&\quad +\phi_1^2 p_1^2 p_2^2 + \phi_2^2 p_2^2 d_{12}^2 b^2
\end{aligned}$$

Up to four real solution for $\cos\theta$ are then obtained by applying Ferrari's closed form solution for finding the roots of a fourth order polynomial to equation 2.5. Each root then leads to exactly one value of α by substitution into equation 2.4 and we end up with four value pairs (α, θ) that we then in turn insert into the equations 2.1 and 2.2 to receive four solutions for the absolute position and rotation of the camera. Which one of those four solutions that is correct can then be verified by backprojection of a fourth world point using a fourth point correspondence.

Structure from Motion

In the section about the P3P problem we learned how we can find the pose of a camera in relation to a point cloud. In this section we will take a look at how the point cloud is constructed from a series of images with a process called *structure from motion* (SfM) [7]. Assume that we have a series of images of the same scene taken from different angles.
First identify SIFT points in all images and assign descriptors. Then match descriptors to identify point correspondences. After that we compute the fundamental matrix (and thus the relative geometry) for all image pairs using the Hartley-Zisserman eight-point algorithm, which is described in the appendix, and RANSAC.
The problem now is to estimate the parameters for the positions, rotations and focal lengths of the cameras and the positions of the SIFT points such that the reprojection error for the whole set of images is minimized. The total reprojection error is defined in the appendix. The minimization problem is solved with a non-linear least square solver such as the Levenberg-Marquardt algorithm.
In practice the estimation process is iterative. First one image pair is used as an initialization. Then the remaining cameras are added into the model one by one. The next camera is added by taking a fundamental matrix between it and a camera already in the model and orienting the coordinate system so that the two descriptions of the camera already in the model match. That orientation provide the initialization of the parameters for the new camera which would then be further refined by solving the non-linear least square problem of the reprojection error.
When all the cameras have been added the result should be a reconstructed 3D model of all the SIFT points found in the images and the poses of all the cameras. It is this point cloud 3D model that we will use as a map for our drones navigation.

Exclusion of outliers with RANSAC

RANSAC stand for RANdom SAmple Consensus and is an algorithm for excluding outliers when model parameters are calculated from noisy and partly corrupt data. In the algorithm a hypothesis model is computed from a small set of randomly sampled points from the data set while the remaining points then "vote" for or against the hypothesis model through a threshold in deviation from the models prediction. This sampling and voting process of constructing and testing a hypothesis is repeated many times, then the hypothesis with the most votes win.
RANSAC is dependent on the assumption that there is an underlying model which can explain the data and the assumption that the outlying points are relatively few compared to the points agreeing with the underlying model. If those assumptions are true only the non-outlying points should consequently agree and vote for one and the same model. The outliers should be randomly distributed enough never to simultaneously vote for the same incorrect model.
The algorithm consist of the following steps:

1. Randomly sample the least amount of data points necessary for computing the model.
2. Compute a hypothesis model from the sampled points.

3. Compare all points to the hypothesis model. All points deviating more than a certain threshold from the models prediction counts as outliers while the ones closer to the prediction counts as inliers. The set of inliers is called the consensus set.
4. Use all points in the consensus set to recompute and refine the hypothesis model with more data. Update the consensus set again.
5. If this hypothesis model has the largest consensus set so far, store the model.
6. Restart from the random sampling and iterate. Always store the best model so far.
7. When an appropriate number of iterations has been done the model is accepted or rejected depending on if its consensus set is large enough.

In our case RANSAC is used for computing the best camera model from our point correspondences. We have a map of a space consisting of a point cloud with descriptors for the points. Assume now that we have an image of this space and that we wish to derive the pose of the camera that captured the image. First we use SIFT to find points in the image and compare their descriptors with the point cloud, which gives us a set of point correspondences. We now wish to derive the best camera model P from this data set and it is here that RANSAC enters the picture.
In order to compute P three points are needed, so that is the amount of points we sample randomly in the algorithms first step. Then we compute P by solving the P3P problem in the algorithms second step. In the third step of the algorithm the outlying points are separated by using P to project the 3D points of the point cloud down upon the image plane and control their distance from their pairs that were detected in the image. The remaining steps in RANSAC handles the tracking of the best-so-far model.

3

System architecture

The main goal of this project has been to create a drone capable of localizing itself and navigate autonomously using computer vision, but in order to reach this goal lots of work has been spent developing practical support structures such as a user interface. The program *bebopAutoFly* is the result of this development effort and is the main result of this project.

Overall description of the system

In this section we will provide an overview of the system and how it allows the drone to localize itself and navigate autonomously in a space.
The system consist of a drone and a base station computer communicating with each other wirelessly over wifi. The drone contains a set of sensors and an onboard computer and is capable of controlling its own flight to the extent that it can remain stable in the air without external input and it can execute simple tasks such as "move one meter at a certain angle upwards and forwards" or "capture an image". The base station computer takes care of more high level localization and navigation which is not as time critical. It demands new images from the drone which it uses to localize the drone in a point cloud map of the space and then it computes flight vectors to take the drone to a destination in the map. Those flight vectors are uploaded to the drone which follows them blindly. The flight vectors specify small jumps and the drone is localized between them, closing a feedback loop that ensures some level of robustness to the navigation.
Before the system is used for autonomous flight a point cloud map of the space to navigate needs to be created. This is done by photographing the space from as many angles as possible (preferably at least a few hundred images for lecture hall sized room) and then compute the geometry of the space with structure from motion. The result is a data set consisting of 3D points with corresponding SIFT-descriptors describing all interest points that could be found in the image data and we call this data set a point cloud map. Example of such maps can be seen in figure 3.1 that shows the room used for testing during the development of the system and figure 4.3 that shows the room used for the evaluation of the autonomous navigation.
Before the map is used it needs to be calibrated in order to ensure that the scale of the map and the real space is matched. This is done by flying the drone a specified distance in the space and localize the drone in the map before and after the movement. We then get a vector in the map with a known length in the real space and thus the scale of the map is known.

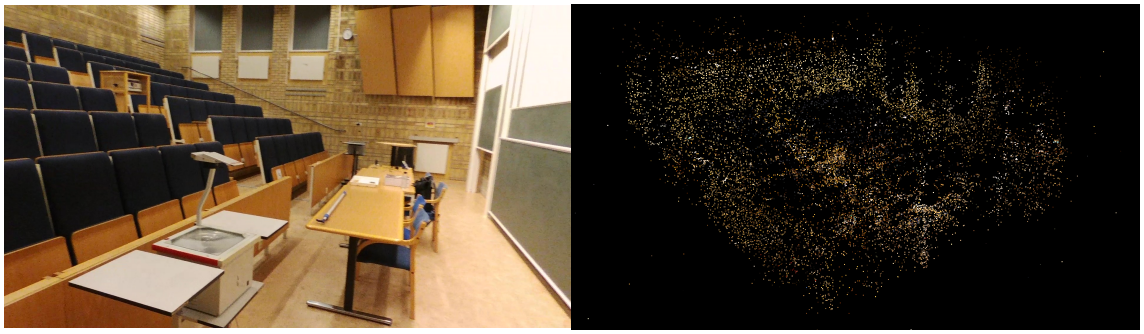

Figure 3.1: The left figure shows the room that was used during development and testing of the system. The right figure shows the point cloud representation of the same room.

When we are to localize the drone we use an image from its camera and compare the SIFT points we find in it with all the 3D points in the point cloud map to get point correspondences. We then use the correspondences to compute the position of the drone by solving the P3P problem.

The drone

The drone model used in the project was Parrot Bebop 2, see figure 3.2. It was chosen because the manufacturer provides an API for sending commands to the drone and control it from another computer. The API includes commands such as direct control of the flight speed, displacement vectors relative to drones current pose, take off, land, capture image, stream video, etc. This interface and the fact that the drone came preassembled and was ready to use out of the box allowed the project to be focused on localization and navigation rather than making the hardware work.

The drone communicates via wifi. When it is turned on it acts as a wifi hotspot and to control it one must connect another wifi unit to this network. The drone was originally designed to be controlled by a mobile phone through the company's app, but any computer can control the drone through the company's API.

The drone weighs 500 g and is capable of roughly 20 minutes of flight time. It is equipped with a camera with fisheye lens, GPS, sonar, barometer, accelerometer, gyro, magnetometer and an optical-flow-sensor. These sensors are not accessible directly via the interface but their readings are instead presented as fused information; the altitude presented for instance is based on sonar or air pressure depending on which sensor the drone considers most reliable for the moment [8].

The drone can stream video, but the stream is too unstable, the image quality too low and anomalies too common for it to be useful for visual navigation. Instead still images were used, where image quality and reliability was promoted at the cost of refresh rate.

Figure 3.2: The figure shows Parrot Bebop 2, the drone used in the project. In the nose of the drone the fisheye lens camera is visible. Among the sensors on the drones belly we can see an optical flow camera used as a speed sensor and a sonar used as an altitude sensor. Several other sensors are also used in the sensor fusion to estimate those and other states of the drone.

The anatomy of the software

The control application is designed as a set of modules with different responsibilities. An overview of the system architecture can be seen in figure 3.3.

- DroneController – responsible for the interface with the drone, sends and receives commands, messages and data.
- Locator – responsible for the computer vision algorithms used for localizing the drone in relation to the point cloud map.
- Navigator – responsible for controlling the drones autonomous flight and turns waypoints into flight vectors.
- UserInterface – responsible for the applications user interface which both writes important information on the display and reads the keyboard input. This module can be regarded as the programs main thread and controls the other modules.
- Visualizer – responsible for visualizing the point cloud data and the drones position in it. Implemented as a separate application.

Since we are dealing with a realtime system with a flying, potentially dangerous object it is very important that no process in the application blocks another so that it is always possible to abort a dangerous situation. The application is therefore designed so that new threads are spawned for most new tasks. The application is also designed so that the operator always has the ability to assume manual control of the drone. Any pressed button will in any moment immediately set the drone to manual flight which means that without further input it will stop and hover in place.

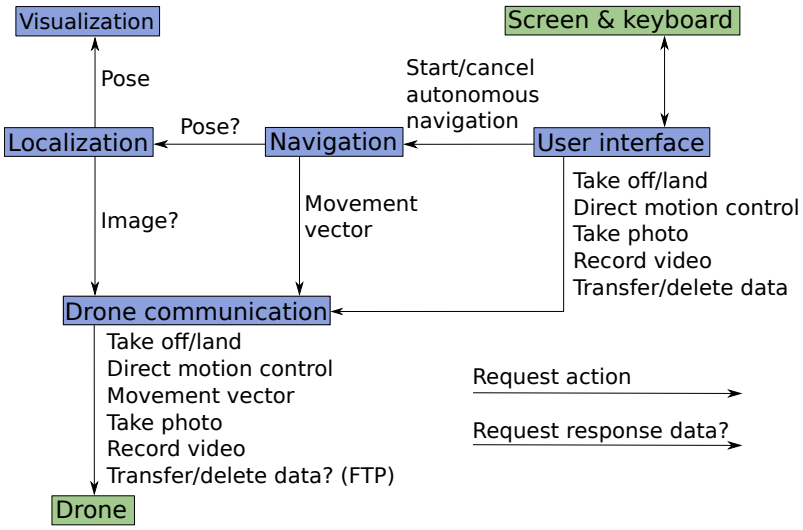

Figure 3.3: Overview of the system architecture.

The drone interface

The module *DroneController* is the control applications interface with the drone and handles all messages, commands and data sent between the drone and the computer. This module is the only one communicating directly with the drone, making it easier to swap the drone for a different model in the future without needing to make changes to other modules. The module is mainly a wrapper around ParrotSDK (sometimes also referenced as ARDroneSDK3) which is a C API provided by the Parrot company for communication and control of their various drone products. Images and video is downloaded from the drone to the computer via FTP. ParrotSDK can handle the FTP link, but I found it easier to use a third party software instead [9].
The module *DroneController* initialize the communication with the drone and then mostly passes messages and provides easy access to a set of chosen commands from ParrotSDK. There are instances where the module does more than passing messages and instead also has some logic of its own. One such example is FTP downloads, where the logic consist of waiting for an image to be reported as captured before the the module tries to download it.

Localization

The module *Locator* encapsulates the localization algorithms. It was originally developed to use image localization along with IMU integration since the two localization methods in theory complements each other well. IMU integration provides fast movement updates and is good for small adjustments, but the estimated position is prone to drift and accumulate errors over time. Image localization is on the other hand a relatively slow method for estimating the position, but it is globally stable. By using IMU integration to continually estimate the drones position but correct it regularly with image localization the combined position estimate should be both rapidly updated and globally stable.
However, in practice it proved to be hard to calibrate the IMU integration and fuse

it with the point cloud map of the image localization. One problem was that the drone supplier Parrot does not provide direct access to the IMU readings in their drone interface. With such access it is possible that it would have been easier. The attempts to fuse the IMU data with the image localization was abandoned and instead image localization was used alone in the *Locator*-module. If one considers the localization module and the navigation module together as a whole however image localization and IMU integration is still used in roughly the way described here, but the IMU integration is instead delegated to the built in methods in the drone.
The image localization is based on solving the P3P problem and estimates the drones camera pose with respect to a point cloud map of the space it flies in constructed prior to the flight. When a new position estimation is engaged the following steps are executed:

1. An image is captured.
2. The image is converted to the grayscale format PGM.
3. SIFT points are extracted from the image.
4. The SIFT points are matched against the point cloud points. A KD tree data structure is used to speed up the matching.
5. The camera pose is computed with RANSAC and a P3P solver.
6. The resulting pose is either accepted or rejected based on the size of the consensus set from the RANSAC algorithm.

When a position has been accepted it can be used by the other modules, for instance for navigation and visualization. The most important parameters to adjust for the localization are the various RANSAC parameters, such as the deviation threshold that governs whether a point is in the consensus set and how large the consensus set needs to be for a camera model to be accepted.
The module uses ImageMagick for the grayscale conversion to PGM, VLFEAT for SIFT and the KD tree. RANSAC and the projective geometry algorithms where provided by Chalmers and based on Kneips P3P-solver implementation [6].

Navigation

The *Navigator* module handles the control applications various modes of autonomous navigation and contains methods for saving waypoints, flying to previously saved waypoints and executing a calibration dance.
The autonomous navigation modes utilizes the drones *moveBy* command from the ParrotSDK drone interface, where one instead of setting a velocity sets a vector relative to a body fix frame of reference to the drone; one could for instance demand that the drone flies two meters to the right. The drone is probably using IMU integration internally to achieve such a movement, one evidence of this is that it works well indoors, but it is not specified by the supplier how this feature is actually achieved.
The navigation module contains two versions of the calibration dance, *calibrationDance* and *calibrationDance2*. The first version was developed when the IMU integration was still used for localizing the drone and thus the movements are designed to measure both the scale and the rotation of the point cloud map with respect to

reality. The calibration was executed by measuring two vectors in the map where the size and direction of their real space counterparts was known. A vector is measured by first capturing an image, then flying one meter in a known direction and then capturing another image. The two captured images are then localized in the map and their positions provide a vector. In the first version of the calibration dance the drone first flies one meter straight up and then one meter to the right. The transformation matrix between the point cloud maps frame of reference and the real world frame of reference is stored for later use in future localization and navigation. The second version of the calibration dance is intended to be used with the current localization system where only image localization is used. Therefore the only property of the point cloud map needed to be measured is the scale. This allows for a simplified calibration dance and so in this version the drone only measures one vector by flying one meter to the right.
The control applications main mode of autonomous navigation is *moveToWaypoint*, where the drone tries to return to a previously stored position. In this mode the drone flies toward a destination in several jumps with a maximum one meter length and stops in between jumps to perform an image localization and compute the flight vector of the next jump. When the drone is close enough to the destination (at the moment it should be closer than half a meter) it stops, adjusts its yaw angle to match the destination pose and finally reports that it has reached its destination.
This mode of navigation consisting of small jumps with image localization in between them is more or less identical to the originally intended combination of IMU integration and image localization, but in this case the IMU integration is delegated to the drones built-in functions instead of being a part of the off board control application.

4
Testing and evaluation

The autonomous navigation capability of the drone was evaluated by testing how well it managed to navigate to a waypoint. The method of evaluation used was unfortunately very rough and simple and while it might not provide the most accurate quantitative picture of the drones capabilities it does serve as a proof of concept for the autonomous localization and navigation. The test was conducted by first letting the drone localize itself and store the pose and then release the drone from some distance away and see how well it finds its way back to the stored pose. The final deviation between the end pose and the goal pose was then measured. This was repeated several times with the drone released from different starting positions each time.
The external measurement method used to determine the drones position was based on a camera mounted at a fix distance and a fix angle from the goal position. When the drone reported that it had reached its destination an image was captured and the deviation between the drones end pose and the goal pose could be measured in two translational degrees of freedom by superimposing those respective images, see figure 4.1. One does not get the total distance between the end and goal positions since a measurement of the depth is missing, but the two distances that is measured still says at least something about how well the drone finds its way back to the waypoint. The evaluation setup can be seen in figure 4.2 and the room used for the evaluation can be seen from another angle in figure 4.3. A list with a summary of the different parameter settings used during the evaluation can be seen in table 4.2.
When extracting the drones position from an image the first unit one gets is pixels, so in order to get a useful measurement in a real world distance unit like meters one needs to know the scale of the pixels. Figure 4.4 describes the geometry of the setup and shows two similar triangles constructed between the goal position, the end position and the camera both in the image plane and in reality. By fixing the cameras focal length and measuring the inner parameters of the camera matrix we get the focal length of the camera in pixels [10]. If we then places the camera so that its focus is placed at the goal position we can use the similar triangles from figure 4.4 as

$$\frac{X}{L} = \frac{x}{f} \implies X = \frac{Lx}{f}$$

and thus receives the end positions deviation from the goal position in meters.
The test was repeated 21 times. The results are displayed in table 4.1 and figure 4.5. The measurements agrees with what could be roughly eye-balled when the measurements where conducted.

	x	y	total
mean (m)	0.69	1.11	0.98
standard deviation (m)	0.37	0.19	0.53

Table 4.1: Statistical measures of the deviation between the end position and goal position when the drones autonomous navigation was evaluated.

Parameter	Value
SIFT number of octaves	max
SIFT levels per octaves	3
SIFT first octave	0 (start at full resolution)
SIFT descriptor matching threshold*	0.65
KD-tree number of trees	12
KD-tree max number of comparisons	50
RANSAC consensus set size acceptance threshold	20
RANSAC number of iterations	100
RANSAC tolerance**	0.002
Calibration dance flight distance	1 m
Waypoint arrival distance threshold	0.5 m
Size of the point cloud map	6000 points
Number of images used to construct the map	67
Distance between external evaluation camera and waypoint	4.5 m
Number of evaluation measurements	21

Table 4.2: A summary of the parameter settings used during the evaluation. (*) The fraction between two angles in descriptor space, see the section on SIFT matching. (**)Reprojection error from the p3p solution in normalized coordinates.

We note that there is a systematic error present in the measurements, the drone always seem to end up above and to the right of the goal position. This might be explained by the fact the the drone was released from the right more often than from the left. Since the drone stops when it detects that it is within a certain distance threshold from the destination a systematic error should arise if there was a bias in release direction. The fact that the systematic error is larger than the stopping threshold and that all points, even those from when the drone was released left of the goal position, lies to the right of the goal acts as evidence against this explanation though.

So what other error sources do we have? First we note that for the drone to accept a localization result the consensus set size of the RANSAC algorithm must be at least 20 which is a high enough number to suggest that RANSAC did converge well during the evaluation flights.

The next possible error source is the point cloud map and its calibration. Here it would be interesting to know how accurate the calibration dance is in measuring the scale of the map and then, if there were to be an error in the scale, how that impacts the navigation.

A new flight vector for the drone is calculated purely within the map, drawing a line between current pose and waypoint, and then rotated to match the body-fix coordinate system of the drone. The scale is then applied and jump and arrival distances are computed in the drones frame of reference. Therefore an error in the alignment of the camera with the drones body would cause an error in the angle of the flight vector, while an error in the scale of the map would cause an error in the length of the flight vector in the drones frame. Since the camera is rigidly mounted on the drone the angles should not be a problem. The scale however is likely to be somewhat off, so we should expect an error in the arrival distance threshold.

If the scale of the map is too small the drone should be stopping early before it arrives at the waypoint. If the scale is too large the drone might overshoot the waypoint and might also have trouble getting acceptably close to the waypoint to be considered arrived. The map scale being too small could account for the systematic error in the mean end position being larger than 0.5 meters.

The calibration dance is reliant on the accuracy of two measurements; the localization in the point cloud map and the drones flight vector command. Neither of those were evaluated, but the point cloud localization is probably a lot more accurate than the flight.

If I were to do this project again I would focus more on testing and evaluation. One important issue would be to get a good ground truth measurement. The external camera used in this project is a first step towards that but there are much better systems available, for instance the Qualisys tracking cameras [11]. With such a system providing accurate measurements of the drones absolute position at every moment the evaluation would have been almost trivial.

There is however some things that could have been done even with the tools that was available in this project. One could eliminate the inaccurate drone flight part of the calibration dance to get a better measurement of the scale by simply placing the drone still on the ground and then moving it by hand to the next spot, measuring the distance by hand. Thus the quite bad accuracy of the flight gets replaced by

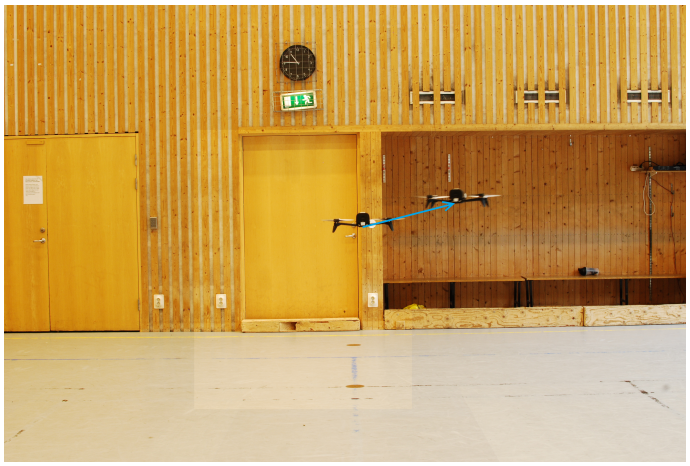

Figure 4.1: The error in two degrees of freedom was measured by superimposing images of the pose the drone tried to replicate and the end pose from the attempts.

Figure 4.2: The setup used for the evaluation of the system. The drone is placed so that the goal position is located in focus in front of the camera.

the quite good accuracy of a measuring tape. This would help improve the flight distance and arrival threshold, provided that the inaccuracy in the drones flight only lies in a high variance and not also in a bias in the mean.

Figure 4.3: The figure to the left shows the room used for the evaluation of the drones navigation. The figure to the right shows the point cloud representation of the same room, with the drones pose marked by a cone.

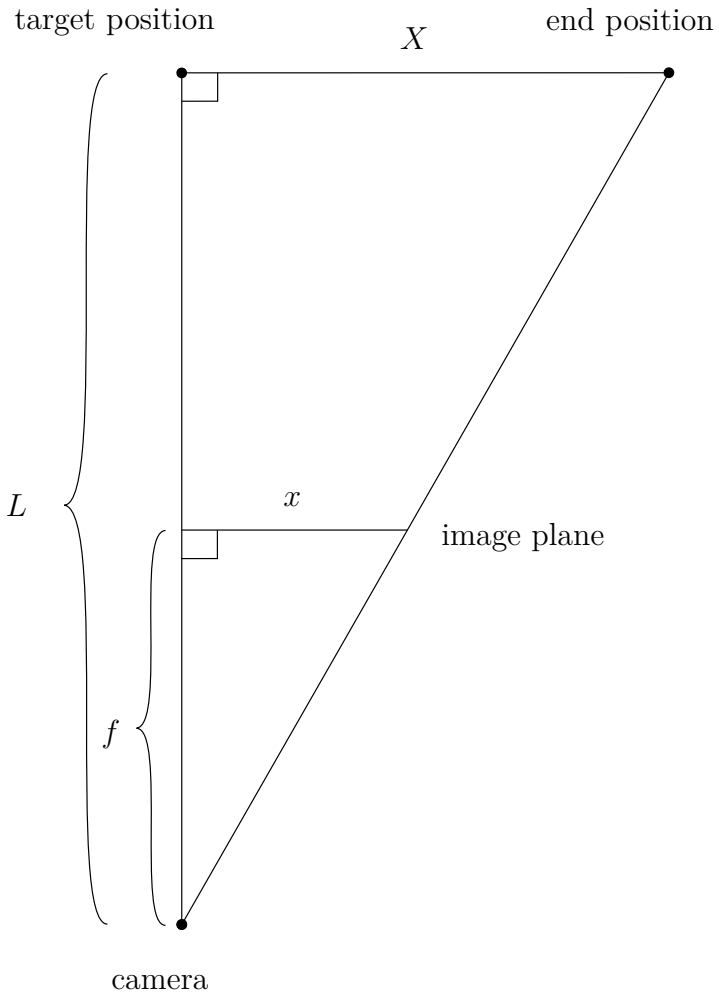

Figure 4.4: A sketch of the geometry of the problem to translate from pixels in the image to meters in real space. Note the two similar triangles, one with focal length and deviation in the image as sides and the other with length between camera and goal position and end position deviation in real space as sides.

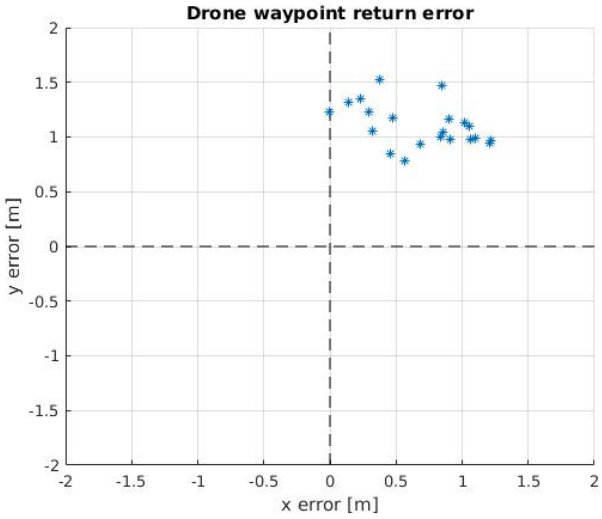

Figure 4.5: The results of the evaluation of the drones autonomous navigation. The points shows the deviation between goal position and end position. Note the systematic error.

5
Conclusion

This report describes how a system consisting of a quadcopter drone equipped with a camera can localize itself in a previously mapped environment and navigate autonomously to a destination.
The evaluation of the drones autonomous navigation shows that the system works, the control application managed to steer the drone from a somewhat arbitrary position to a destination position in the same space. More work could be done to enhance the navigation, make the localization more robust and make the deviation of the end position from the destination smaller, but the concept was proven to work well and the project should thus be regarded as a success.
There are many possible interesting ways to further develop the system. The most obvious next step would be to add a list of waypoints, so that the drone can follow a route. This should be relatively simple to do with the existing code base. When the possibility to follow a route exist comes the the problem of planning such a route through the map. Here many interesting problems arise such as being able to detect empty volume in the point cloud where the drone could fly and then plan an efficient route between two arbitrary points. A lot of work has already been done in this field and I recommend reading more about SLAM (Simultaneous Localization And Mapping).
Another useful ability to develop would be to detect obstacles in front of the drone from the camera images. Such obstacles would need to be handled, either by stopping in order to avoid collision or by planning a new route around the obstacle.

Bibliography

[1] H. Shin, D. Kim, Y. Kwon & Y. Kim, "Illusion and Dazzle: Adversarial Optical Channel Exploits against Lidars for Automotive Applications," in ches, 2017, pp. 445-467.

[2] D. G. Lowe, "Object recognition from local scale-invariant features," in iccv, 1999, pp. 1150-1157.

[3] T. Lindeberg, "Scale Invariant Feature Transform," 2012. [Online]. Available: `http://www.scholarpedia.org/article/Scale_Invariant_Feature_Transform`, Accessed on: Aug 10, 2019.

[4] A. Vedaldi, "Scale Invariant Feature Transform (SIFT)," 2007. [Online]. Available: `http://www.vlfeat.org/api/sift.html`, Accessed on: Aug 10, 2019.

[5] A. Vedaldi & D. Novotny, "KD-trees and forests," 2007. [Online]. Available: `http://www.vlfeat.org/api/kdtree.html`, Accessed on: Aug 10, 2019.

[6] L. Kneip, D. Scaramuzza & R. Siegwart, "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation," in cvpr, 2011, pp. 2969-2976.

[7] N. Snavely, S. M. Setiz & R. Szeliski, "Photo tourism: exploring photo collections in 3D," in ACM transactions on graphics (TOG), 2006, pp. 835-846.

[8] Parrot SA, "Introducing Parrot Bebop 2: Your flying companion!," 2015. [Online]. Available: `http://blog.parrot.com/2015/11/19/introducing-parrot-bebop-2-flying-companion/`, Accessed on: Aug 10, 2019.

[9] otom, "FTP Client Class," 2012. [Online]. Available: `https://www.codeproject.com/Articles/8667/FTP-Client-Class`, Accessed on: Aug 10, 2019.

[10] J. Y. Bouguet, "Camera Calibration Toolbox for Matlab," 2015. [Online]. Available: `http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/parameters.html`, Accessed on: Aug 10, 2019.

[11] Qualisys AB, "Applications in Engineering and Cybernetics," 2019. [Online]. Available: `https://www.qualisys.com/applications/engineering/cybernetics/`, Accessed on: Aug 10, 2019.

A

Structure from Motion details

The geometry of two camera views and the fundamental matrix

In the section about one camera view we explored how to project a point X from the three dimensional world down on a two dimensional image plane with the help of the camera matrix P. In this section we will use this projection in order to calculate how two cameras and one object seen by both cameras are related to each other through the fundamental matrix F. The geometry of the problem is illustrated in figure A.1.
We begin by reminding ourselves how a psuedoinverse looks like.

$$A^+ = (A^T A)^{-1} A^T \,,\; A^+ A = I$$

$$A^+ = A^T (A A^T)^{-1} \,,\; A A^+ = I$$

Assume first that we have a world point $\vec{X}$ and a first camera with matrix P. Then $\vec{X}$ is mapped onto the image plane as

$$\vec{x} = P\vec{X}$$

and we can also recreate $\vec{X}$ by casting a ray from the camera center through $\vec{x}$ as

$$\vec{X} = P^+ P\vec{X} = P^+\vec{x}$$

We can find the image point of $\vec{X}$ as seen from the other camera P' in the same manner as

$$\vec{x'} = P'\vec{X} = P'P^+\vec{x}$$

where we then use the from $\vec{x}$ reprojected point $\vec{X}$ in order to relate the two image points.
For the next operation we remind ourselves that in homogeneous coordinates points and lines are the same and that the line between two points is received by taking the cross product of the two points. The camera centers of the two cameras are seen in each others image planes as the points $\vec{e}$ and $\vec{e'}$ and are called the epipolar points. The center of the camera P' thus becomes the point $\vec{e}$ in the view of camera P.
First we define the line $\vec{l}$ as the line between $\vec{e}$ and $\vec{x}$. This line is called the epipolar line and describes all the possible positions of $\vec{X}$ if we only regard the information

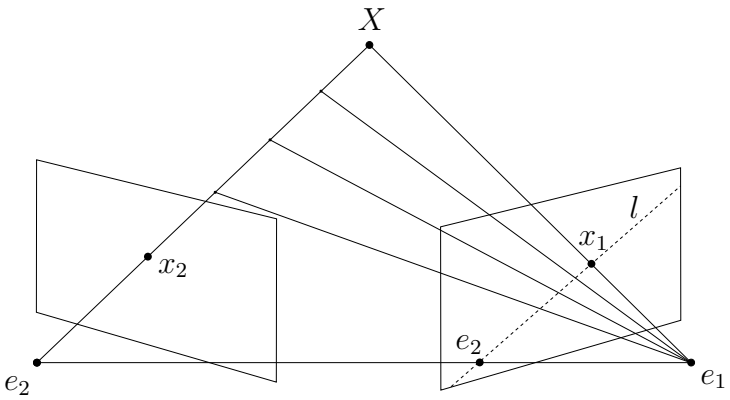

Figure A.1: The geometry of two cameras that sees the same world point X. One camera cannot alone determine the distance to a point that it sees. The line defining all the possible positions of a point in sight for camera e_2 is seen as the epipolar line l in the view of the other camera. Together the two cameras are able to determine the position of X.

from the other camera. After having constructed $\vec{l}$ we do the assumption that $\vec{x}$ and $\vec{x'}$ describes the same point X which allows us to do a substitution. Then we rewrite the left hand side of the cross product to matrix form as $\vec{e}\times = [\vec{e}]_\times$ and combines the expression to a matrix F.

$$\vec{l} = \vec{e} \times \vec{x} = \vec{e} \times PP'^{+}\vec{x'} = [\vec{e}]_\times PP'^{+}\vec{x'} = F\vec{x'}$$

The position of the image point $\vec{x'}$ now constrains the point $\vec{x}$ to the line $\vec{l}$ in order to describe the same world point X. This is described by the equation

$$\vec{x} \cdot \vec{l} = \vec{x}^T F \vec{x'} = 0$$

We can now find the fundamental matrix F numerically through a series of corresponding points $\vec{x} \leftrightarrow \vec{x'}$ by setting up a system of linear equations and solving it in a least square sense.

$$\begin{pmatrix} x_1x'_1 & y_1x'_1 & x'_1 & x_1y'_1 & y_1y'_1 & y'_1 & x_1 & y_1 & 1 \\ x_2x'_2 & y_2x'_2 & x'_2 & x_2y'_2 & y_2y'_2 & y'_2 & x_2 & y_2 & 1 \\ & & & & \vdots & & & & \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = 0$$

If we also knew the internal parameters of the cameras, their calibrations, we could now deduce the positions of the cameras. If we knew the the calibrations K we could compensate for them. Assume thus for simplicity that we did this and set

$$K = I$$

which then gives us the camera matrices

$$P' = [I|0]$$

$$P = [R|\vec{t}]$$

where the first camera P' is located in the origin and rotated to parallel with the global coordinate axes and the other camera P is offset with some vector $\vec{t}$ and rotation R from the first camera.
The pseudoinverse of P' is then

$$P'^{+} = \begin{bmatrix} I \\ 0 \end{bmatrix}$$

which together with P extracts the rotation matrix R as

$$PP'^{+} = R$$

We can now identify translation and rotation in the fundamental matrix as

$$F = [\vec{e}]_{\times} PP'^{+} = [\vec{t}]_{\times} R$$

where we extract the matrices R and $[\vec{t}]_{\times}$ from F using the SVD matrix decomposition.
We have now shown how to go all the way from point correspondences in two images, through the fundamental matrix and by the help of camera calibration all the way to an expression of translation and rotation of the cameras. This is used to build the point cloud maps that later are used to localize the drone.

The total reprojection error

Each camera consist of seven parameters in total; three rotation parameters, three position parameters and one focal length. Let Θ_i denote the total parameter collection for camera i. Let p_j denote the three position parameters that make up world point j and let q_{ij} denote the 2D projections of p_j through camera Θ_i. Let also $P(\Theta, p)$ denote the mapping of a point p to its 2D projection by a camera with parameters Θ.
Then the total reprojection error can then be stated as

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} ||q_{ij} - P(\Theta_i, p_j)||$$

where $w_{ij} = \{0, 1\}$ is an indicator variable denoting if point p_j is visible in camera Θ_i and n and m is the number of cameras and points respectively.