Master Thesis

Vision, Graphics & Interactive Systems

Vision system for indoor UAV flight

Author:
Jonas B. Markussen

Period:
1/2/2015 - 3/6/2015

Aalborg University, Denmark

Aalborg University
Vision, Graphics & Interactive Systems
Fredrik Bajers Vej 7
9220 Aalborg, Denmark
http://www.create.aau.dk/vgis/

Title:
Vision System for Indoor UAV flight

Theme:
Computer Vision

Project Period:
Spring Semester 2015
1-2-2015 to 3-6-2015

Project Type:
Master thesis

Author:
Jonas Borup Markussen

Supervisor:
Thomas Baltzer Moeslund

Copies: 3

Page Numbers: 94

Date of Completion:
June 3rd, 2015

Abstract:

The purpose of this project is to develop a framework containing two subsystems to aid in indoor UAV flight. The two systems will be developed and tested independently. A visual odometry system to be able to estimate motion in GPS denied areas, and an obstacle avoidance system to help avoid collisions. For the visual odometry system a novel method using a downwards and upwards facing camera is proposed. Features are tracked using Shi-Tomasi and Lucas-Kanade. From the tracked features a homography is estimated for each of the cameras. These are combined into a single motion estimate. For the obstacle avoidance system, a method is proposed which divides the image into left, right and center region and classifies these independent. Features are tracked using Shi-Tomasi and Lucas-Kanade, the tracked features are combined into trajectories. The trajectories are classified using the proposed method. The visual odometry system is found to have a mean absolute error varying from 0.16cm up to 0.54cm per sample. Errors in rotation have been found to vary from 0.13° up to 0.19°. For the obstacle system 82 different tests have been carried out in 3 different scenarios. The left and right region is classified with a 78.3% detection rate. The center region is classified with a 100% detection rate.

Preface

This master thesis concludes the work of the 10th semester master programme Vision, Graphics and Interactive Systems at Aalborg University in the spring of 2015. The curriculum of the master programme is cited. [24]

During this project several external sources will be used, which will be referred to when used the first time. The reference is seen as a number in square brackets like this: [#]. The number refers to the source in the bibliography, which is found at the end of this report. In the bibliography the books are described with its author, title, pages, publishers, edition, and year. Online sources, like web pages, will be described with its author, title and URL. This bibliography is auto generated by BibTex.
The enclosed CD contains a digital PDF edition of the project, the source code of the developed systems and data sets captured for tests.
Figures and tables are numbered according to which chapter they appear in, i.e. the first figure in Chapter 7 has the number 7.1, the second figure has number 7.2 etc. Abbreviations are found in the word list on page vii. Appendices are found at the end of the report, and are referred to with A, B, C, ...

Jonas Borup Markussen

Acknowledgments

I would like to thank the people who have been a help during the development of this project. Thanks to:

Thomas Baltzer Moeslund	For supervising the project.
Sky-Watch A/S	For assisting with hardware
Chris Bahnsen	For helping with the motion estimation calculations

Wordlist

Abbreviations

UAV	Unmanned Aerial Vehicle
GPS	Global Positioning System
SFM	Structure From Motion
CV	Computer Vision
FPS	Frames per second
RANSAC	Random Sample Consensus
FOV	Field of view
IMU	Inertial measurement unit

Glossary

Monocular camera	A camera with a single lens
Visual Odometry	The problem of estimating motion from images

Contents

1 Introduction

Small UAVs have been around for some years now and are slowly entering the market industry. At the moment they are primarily used for defense, emergency management, law enforcement and agriculture. All of these areas have in common that the UAV is used outdoor. This project will aim to allow indoor use of UAVs.

1.1 The scenario

The use case this project will aim to solve is allowing the UAV to fly inside of warehouses. The idea is that an indoor flying autonomous UAV can be beneficial in reaching storage areas, which are not easily accessible for a human. This could be the area right below the ceiling where the height makes in difficult for a human to reach. It also has the benefit of being faster. A person could just order the UAV to fetch a part and it would deliver the wanted part. The outcome of this project will be a framework using computer vision to handle some of the difficulties associated with indoor flight. The problems are being handled in this project are position estimation, and obstacle avoidance.

1.2 Position estimation

Traditionally UAVs have been using GPS as a method of obtaining a position. This solution may be sufficient outdoors but when flying indoor, GPS is not available. Recently research into using monocular cameras to estimate movement of an UAV has begun. This project will be analyzing these methods and propose a system capable of detecting the motion which can be used as part of the control algorithms for an UAV in order to navigate indoors.

1.3 Obstacle avoidance

If an UAV should be allowed to fly inside of a larger warehouse it is deemed necessary that the UAV is aware of its surroundings. Indoors there are often people moving in the same space as the UAV. Therefore this project will be analyzing previous work in the field of obstacle avoidance and propose a solution capable of detecting obstacles which can be used as part of the control algorithms in order to avoid the obstacles.

2 Analysis

The analysis will be focused on researching current technologies and get an overview of how a system may be developed. Some initial tests are also introduced as part of the analysis; the purpose of these tests is to get a better understanding of how to implement the system. The final outcome of the analysis is the problem statement, which the rest of the report will be built upon.

2.1 Visual odometry

Visual odometry is the process of estimating the motion using the input from one or multiple cameras. Visual odometry is used in various applications varying from robotics to augmented reality. In this project visual odometry will be used to estimate the motion of an UAV. The term visual odometry was introduced in 2014 by Nister et. al [17]. They propose a method for both stereo cameras and monocular cameras. In their paper Harris corners are used as feature detector and features are matched through frames using SAD and a small search region. To estimate the motion between frames the 5-point algorithm, also proposed by Nister, is used [16]. Traditional odometry is the process of estimating the position from integrating the movement of wheel or other moving parts. Similar visual odometry works by finding the motion between consecutive frames in the form of rotation and translation. By integrating these values the overall motion can be found.

The problem of visual odometry is closely related to the problems associated with SFM. In the CV literature SFM is known as the problem of constructing 3D information from a set of images. In SFM the goal is to estimate camera poses to reconstruct a 3D scene, visual odometry aims only at finding the camera pose. A typical pipeline for a visual odometry system is illustrated in figure 2.1.

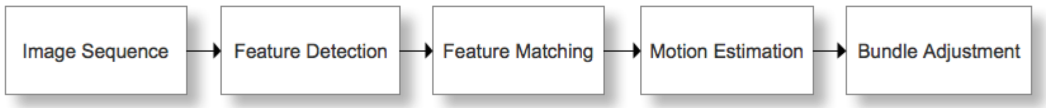

Figure 2.1: A typical visual odometry pipeline. [22]

In the following subsections the presented steps in the pipeline will be described.

Image Sequence

This step involves the gathering of images. In this step various different sensors can be used. The most common setup is a stereovision setup with two calibrated cameras. An example of this is proposed by Milella et al. [14]. Methods using a monocular setup have also proven promising results. The first method using a monocular camera was proposed by Nister[17]. Other sensors like LiDARs could be utilized in this step but no research has been found using such a sensor.

Feature Detection

The purpose of the feature detection method is to find points in a frame, which can be found again in the next frame. Various different feature detectors exist. To choose the best feature detector, the requirements of the systems need to be taken into account. Feature detectors like SIFT has the benefit of being very stable and robust to rotations and scale, but they are hard to compute. Therefore SIFT features may be of better use for an offline system where the frame rate is not critical. A method using SIFT for visual odometry is proposed by Scaramuzza et el. [21]. However with a system for an UAV as in this project, it is better to use features that are easy to compute, an example of this could be Harris corners. Harris corners is a popular method for extracting features. Compared to SIFT it is faster to compute, but is not scale invariant. However the movements between consecutive frames means that scale is not a big issue. Harris corners is used in the approach by Nister et al. [17].

Feature Matching

Feature matching is the step involved in finding corresponding points in two frames. Different approaches have been used to track features across frames. One of the most common methods used is the Lucas-Kanade method. An example of the Lucas-Kanade method used for visual odometry is described by Cambell et al. [4]. Other methods can be used such as matching points by comparing SIFT descriptors, or other feature descriptors.

Motion Estimation

Motion estimation is the most vital part of a visual odometry system. In this step the actual motion is found. The motion is expressed in a rotation matrix R and a translation vector T. The general approach is to estimate the essential matrix from a group of corresponding image points. By knowing the translation between corresponding points in image coordinates, rotation and translation can be estimated in 3D. In figure 2.2 the concept is illustrated. x_1 and x_2 are the corresponding image coordinates found from feature matching. R and T are the rotation matrix and the translation vector and $\mathbf{X}$ is the 3D point in world coordinate.

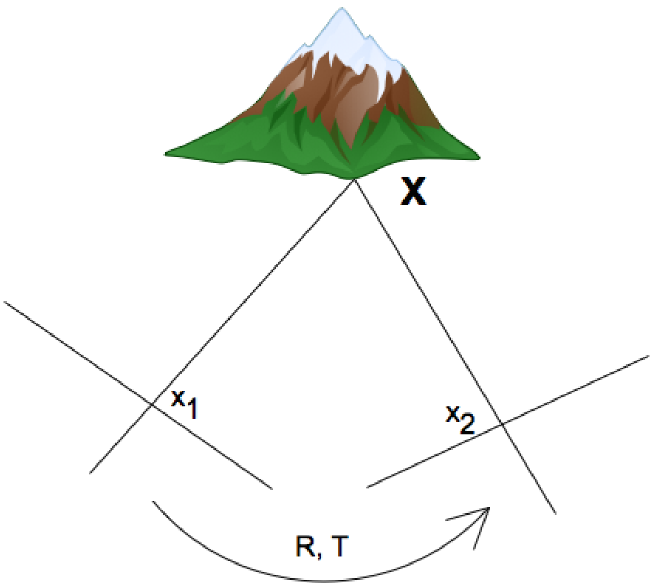

Figure 2.2: Motion estimation illustrated.

Bundle Adjustment

Bundle Adjustment is not necessary in a visual odometry setup as the motion has already been estimated. However a lot of the proposed methods in the literature apply bundle adjustment. Bundle adjustment aims to optimize the pose graph. Ideally the entire graph would be optimized but this is only possible in an offline system. To be able to run bundle adjustment online, a method called Windowed bundle adjustment can be applied [25].

2.1.1 Initial tests

To find out how to successfully implement a visual odometry system for indoor UAV flight, some initial tests are carried out. In this section the tests carried out will be described and what was learned from them. First the test implementation will be described. This implementation is not tended to be the final implementation, it is only meant to draw some initial conclusions.

Test implementation

To test how the typical visual odometry pipeline performs, a simple implementation is tested. The purpose of this test is to determine where the problems will arise in the system. The test setup is presented following the pipeline described earlier. The test system is implemented using functionality from the library OpenCV(See section 4.1). As this implementation is only used for initial tests, in-depth description of the algorithms used will not be presented.

Image Sequence

This step simply fetches the next available image from the test data set. This enables the system to run the images on the frame rate they where acquired even though the processing takes longer than available between each frame.

Feature detection

In this step good features to track are extracted. They are extracted using the Harris Corner Detector [8]. The threshold is set low to detect many features in scenes where not a lot of texture is present. To make sure the detected features are scattered across the image a minimum distance between the features is introduced. This distance is set to 20 pixels.

Feature matching

The features are tracked across 10 consecutive frames using the Lucas Kanade method [12]. In this process points that are not tracked successfully are removed. After 10 frames the matched points are passed along to the next step.

Motion estimation

This step tries the estimate the motion between the last 10 frames using the tracked points from feature detection. The implementation uses the 5-point algorithm to estimate the essential matrix E [16]. The essential matrix is the matrix that maps points from one view to another, see equation 2.1.

$$x'Ex = 0 \tag{2.1}$$

where x are points in the first view and x' are points in the second view. From the essential matrix E the rotation and translation can be estimated using SVD.

Camera mounting

The first test is to determine the best mounting of the camera. Two situations are tested these are described below.

First data set

This data set was captured using an iPhone 6. The resolution is scaled down to 640x360 to process the images faster while still maintaining the aspect ratio. The video was captured pointing the camera forward and downwards with approximately an angle of 45°. In figure 2.3 the path captured in the data set is shown.

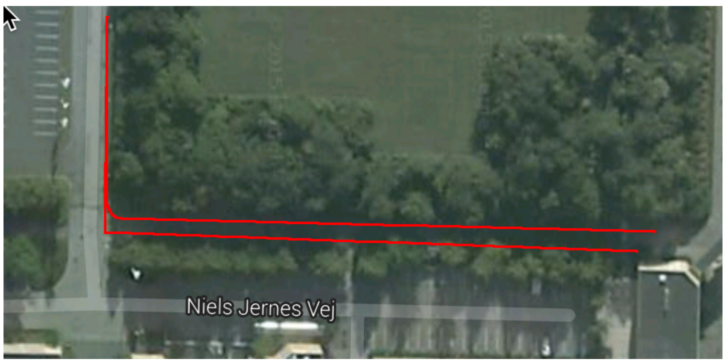

Figure 2.3: The path captured in the data set.

From testing it has been found that the rotations estimated are at times really unreliable and jumps up to 180° in a single run. To eliminate the problems caused by rotations the data set described earlier is captured without rotations. In practice it is impossible to avoid rotations when the camera is hand held. The video is captured by only moving forward, to the sides and backwards. Now the rotation can be ignored and only the translations are concatenated. As shown in figure 2.4 the extracted path does not look similar to the one described in figure 2.3. At first when the camera is moving forward the translations seem to be an almost straight line as it is supposed to be. When the camera start moving sideways or backwards the calculated translations begins to fail.

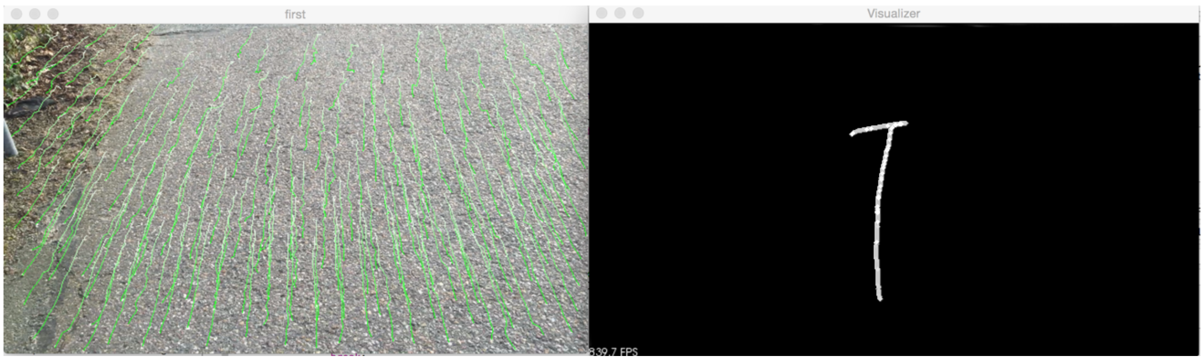

Figure 2.4: Path calculated. **Left:** Tracked Features. **Right:** Concatenated points.

When looking at the left image in figure 2.4 the tracked features are drawn. By comparing the tracked features across frames it seems that these features are tracked correctly, also when the motion estimation fails. This leads to the assumption that it is not the feature tracking part of the system that does not work but instead it is the motion estimating that fails.

Second data test

The second test performed is to test how the orientation of the camera affects the results. A new data set is collected. The path collected is the same as in the previous test but instead of pointing the camera at an angle of approximately 45° the camera is held perpendicular to the ground plane. Again the rotations calculated are found to contain large errors, therefore only the translations are used.

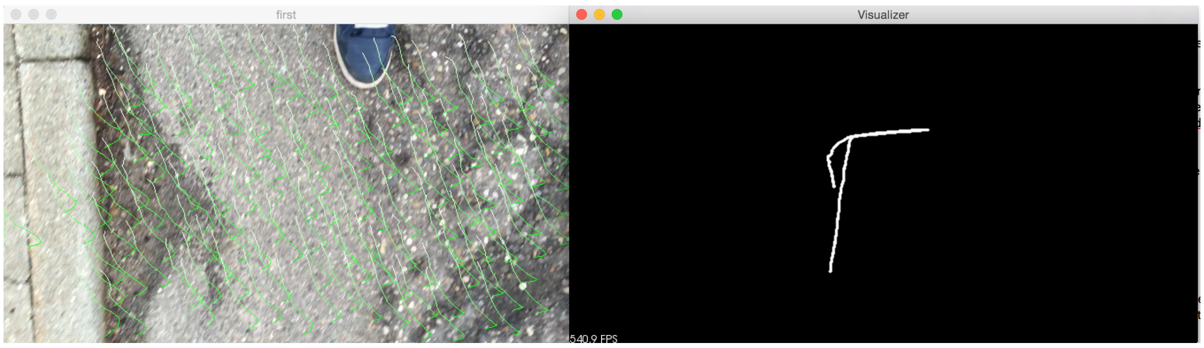

Figure 2.5: Path calculated. **Left:** Tracked Features. **Right:** Concatenated points.

In figure 2.5 the results of the second test is shown. It is clear that the results here are way better than the results from the first test. It is clear that the forward movement and side ways movement are calculated successfully but on the way back, where the camera is moving backwards, problems arise. Even though the backward movements are not correctly calculated they are still better, compared to the first test.

This leads to the assumption that it is better to mount the camera perpendicular to the ground plane.

Texture

In the two first data sets the ground plane is very well textured. This test will reveal how the system performs when the ground plane is not well textured.
As with the camera mounting test a video sequence was captured using an iPhone 6 camera. The camera was pointed perpendicular to the floor as determined in the previous test. This time the video was captured indoors with artificial lighting on a typical non-textured floor. In figure 2.6 the path captured in the video is shown.

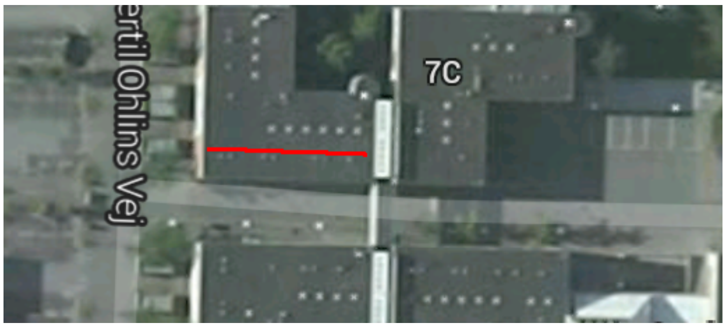

Figure 2.6: The captured path indoor

The video is fed through the test implementation to see how it performs with little to no texture present in the images. The system performs really badly and the calculated path does not at all look like a line. When looking at the tracked features it is also clear that these are not accurate. The result is shown in figure 2.7.

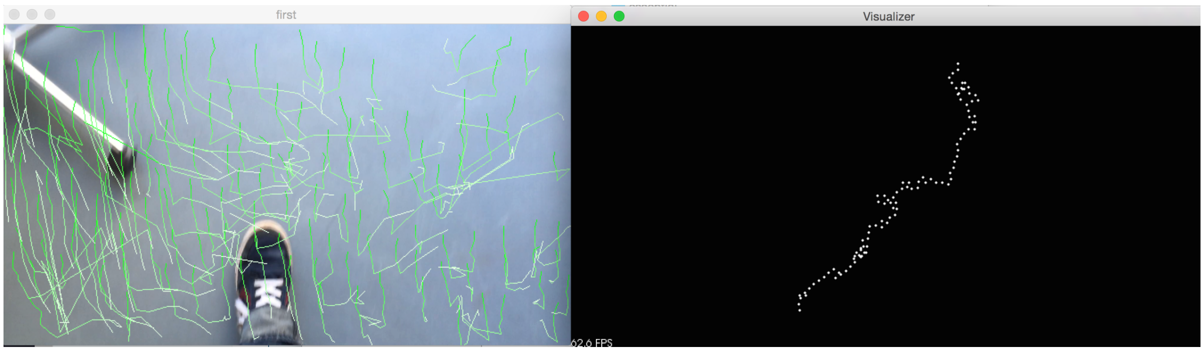

Figure 2.7: Path calculated. **Left:** Tracked Features. **Right:** Concatenated points.

This test has revealed that there is a problem when it comes to texture of the surfaces the UAV will fly over. At the moment there is no solution to this problem meaning that this will be one of the problems to tackle in the implementation of the system.

2.2 Obstacle avoidance

As mentioned in the introduction this project will also consist of a system for obstacle avoidance. Obstacle avoidance is the process of detecting obstacles before the vessel hits them. Obstacle avoidance has been heavily researched, both for ground moving vessels as well as airborne vessels. In this section previous work will be described. The methods will be described in two groups. Monocular methods and stereo based methods.

2.2.1 Monocular methods

Monocular based methods are obstacle avoidance methods based on only having a singe camera to find obstacles from. Mori et al. [15] proposed a method based on relative scale to determine obstacles. Objects are tracked using SURF features and template matching. For each feature point the region around the feature is evaluated to determine whether the scale of the region has changed. By tracking regions that grow rapidly over time potential obstacles can be determined. Figure 2.8 shows how the proposed method works, to the left the tracked features are found. To the right the regions that have expanded are shown, here it is clearly seen that the region belonging to the obstacle is found.

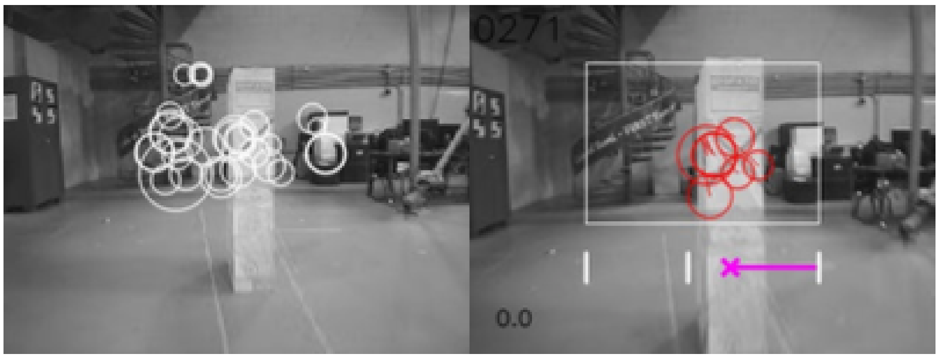

Figure 2.8: Left: Found SURF features **Right:** Regions that are expanding [15]

Another monocular based method is proposed by Zingg et al. [26]. The proposed method is based on using optical flow instead of change in relative size. This method is based on using the fact that objects closer to the camera will generate larger flow vectors than objects further away from the camera. These kinds of methods are especially great when flying in small corridors or similar scenes. In the proposed method Lucas-Kanade optical flow is applied to detect optical flow between frames. An onboard IMU is used to correct for rotations leaving only translations. Methods using optical flow have the problem that objects straight in front of the camera will not generate any flow vectors meaning that the vessel will not be able to detect potential obstacles.

2.2.2 Stereo based methods

Stereo-based methods are the methods, which are based on having more than one camera to estimate depth and use that information to detect obstacles from. A method using stereo cameras is proposed by Heng et al. [1]. In the proposed method a stereo camera is pointing in the flight direction and the disparity image is used to calculate a point cloud. This point cloud is used to generate an obstacle map and determine a free flight path. Stereovision systems have the disadvantage that the distance at where the disparity can be calculated is determined by the baseline between the cameras. If a large baseline is chosen it becomes more difficult to find corresponding points however a larger distance is supported. Another problem by using a large baseline is that the minimum supported distance becomes larger. This problem is illustrated in figure 2.9. The triangle between the two image planes shows the region in front of the stereo camera that cannot be seen by either of the cameras.

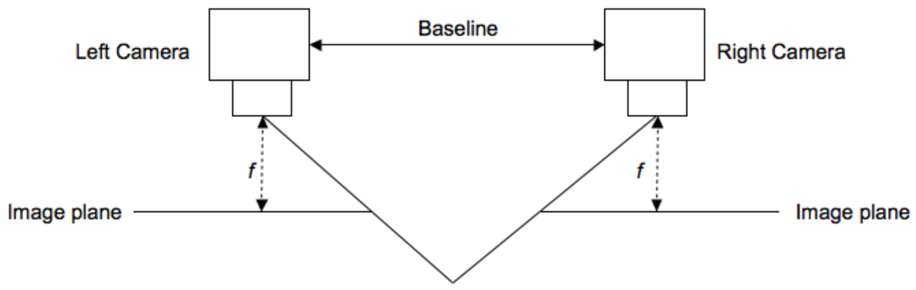

Figure 2.9: Figure showing the baseline problem.

Because many UAVs are already equipped with a single front facing camera this project will aim at developing a system using a single camera.

2.2.3 Initial tests

In this subsection a simple test implementation is used to analyze which problems arise when using a single camera to try and estimate obstacles using optical flow. Some different scenarios are analyzed in order to be able to design the final system.

Data sets

To visualize the flow some data sets are captured. Three different data sets are captured each representing a scenario where obstacles are present. The data sets are captured using a Parrot Ar.Drone 2. For an in depth description of the Ar.Drone see section 4.4.1.

The first data set captured is of an obstacle approaching on the left side of the drone. The drone is flying straight forward and a pillar is approximately 0.5m to the left of the drone as it flies by. The second data set is of an obstacle approaching on the right side of the drone. The drone is flying straight forward and the pillar is approximately 0.5m to the right side of the drone. The final data set is of an obstacle approaching straight in front of the drone. The drone is flying directly towards a pillar and is stopped just upon impact.

Optical flow

In this section optical flow will be calculated for the three different data sets to see how the optical flow behaves in the different scenarios. A simple test program is implemented which captures the optical flow and draws in on top of the image. The optical flow is captured using Shi-Tomasi(see section 3.1.3) to find corners in the image to track. The optical flow is found using Lukas-Kanade(See section 3.1.4). The flow is then drawn on the image for the last 15 frames. The flow is colored according to the sample number from blue to red. The blue is the oldest sample and red is the current sample.

Obstacle to the left

In figure 2.10 a screen shot from the test with the pillar on the left is shown.

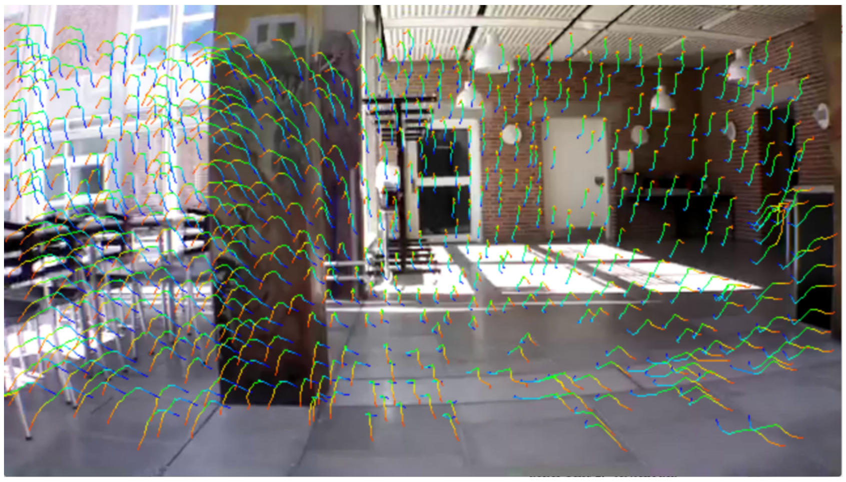

Figure 2.10: Flow with obstacle to the left.

It is seen that the flow spreads faster in the horizontal direction of the image in the region where obstacles are present. In the region of the image right to the pillar it is seen that the flow is much smaller. In the bottom of the image the flow generated by the floor is also quite large.

Obstacle to the right

In figure 2.11 a screen shot from the test with the pillar on the right is shown.

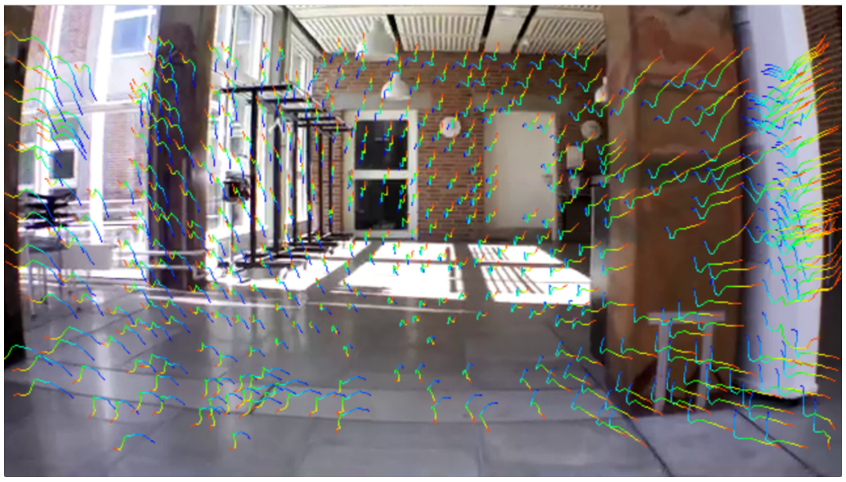

Figure 2.11: Flow with obstacle to the right.

In this scenario the flow behaves quite similar to the one where the pillar was on the left. Again it is seen that the flow is larger in the regions where obstacles is present. Also the flow generated by the floor is quite large.

Obstacle in front

In figure 2.12 a screen shot from the test with the pillar straight in front of the drone is shown.

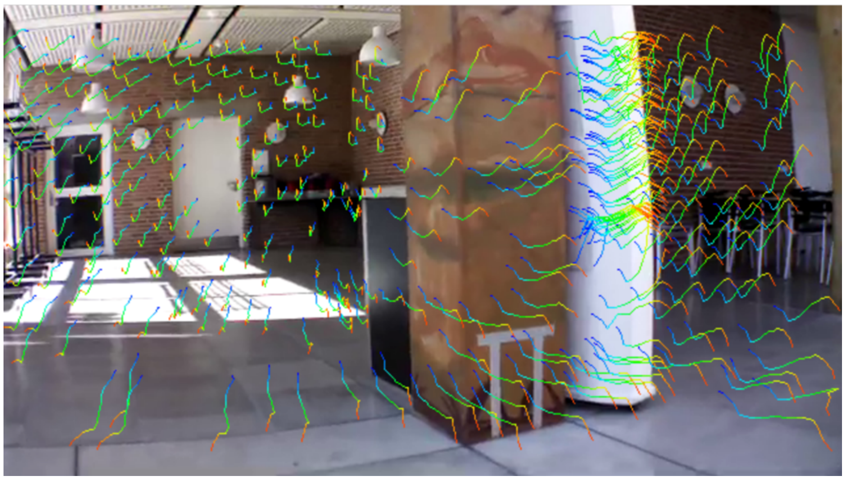

Figure 2.12: Flow with obstacle in front.

Again the result is quite similar to the two previous tests. The flow is large where obstacles are present. However the flow generated by the pillar in the center of the image does not generate larger flows than the wall to the right. This is because the pillar is in the center of the image.

Conclusion

From these tests some conclusions can be drawn, these will be described here. As both the initial tests shows and as the literature describes, obstacles in the center of the image does not generate as much flow as obstacles to the left and to the right of the center. This can be used to divide the image into sections where the optical flow can be analyzed in different ways. Because the flow behaves differently in the different sections of the image, the image can be divided into 5 different areas. These are top, bottom, left, right and center. Ultimately this can be used to determine which one of these different directions it would be safe to fly. The regions are illustrated in figure 2.13.

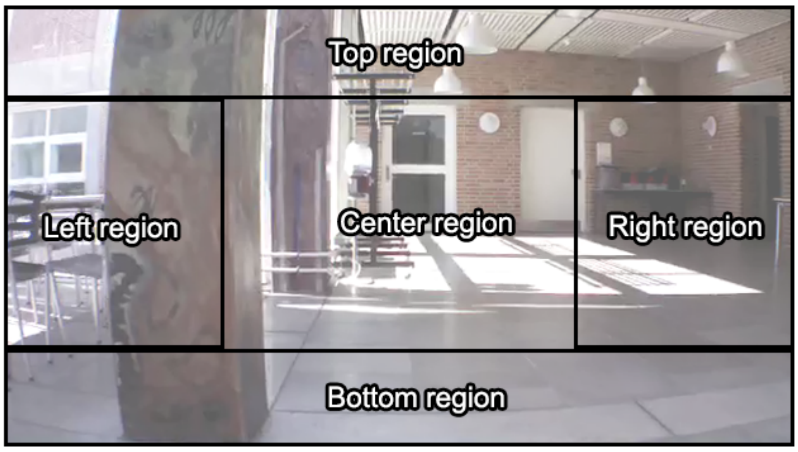

Figure 2.13: The regions the image can be divided in.

In the top region a high flow in the negative vertical direction will be obstacles. In the bottom region a high flow in the positive vertical direction will be an obstacle. In the left region a high flow in the negative horizontal direction will be obstacles. In the right region a high flow in the positive horizontal direction will be obstacles. In the center region flow expanding rapidly in all directions will be obstacles. In this project focus will be on determining whether it is safe to fly forward, left or right. This means that the top region and bottom region will be ignored.

2.3 Problem statement

Based on the description of the project and the analysis chapter, a problem description and problem statement can be formed. This problem statement will be used for concluding the project in the conclusion chapter. The problem is divided into two separate parts. These are visual odometry and obstacle avoidance.

Visual Odometry

From the initial tests a system setup can be proposed. From the initial tests it is known that the best camera position is the camera pointing downwards. The initial tests also showed that lack of texture is a big problem. To make the possibility of more texture present, it is proposed to use two cameras instead of one. These cameras will be mounted so that one points downwards and the other upwards. Along with these cameras there will be two distance sensors parallel to the image sensors. These sensors will be used to scale the motion estimate from each camera. The system will only be focusing on translation in the horizontal plane, and rotation around the vertical axis. The reason for this is that the motion vertically can be read directly from the distance sensor. The reason to only focus on the rotation around the vertical axis is that it is assumed that the UAV is capable of leveling itself. Where most visual odometry systems focus on calculating a trajectory over a longer time period, this system will be focusing on estimating the current motion at a fixed time rate. These measurement can be summed over time to estimate the full trajectory.

The output from this system would be incorporated into the control system of the UAV. Therefore the control algorithms and the characteristics of the UAV would have a large impact on the final system performance. As these parameters are unknown at this time, requirement to the system cannot be determined. Therefore the tests in this project will be focusing on what kind of performance that can be expected from the visual odometry system.

Obstacle Avoidance

From the analysis multiple methods have been analyzed which can be used for obstacle avoidance. 3D vision methods have an obvious advantage by being more precise and faster at detecting obstacles. However for 3D cameras to work at a longer distance the baseline needs to be wide which would increase the weight of the system, and with a large baseline the system would not be able to detect object that are close to the UAV. By using the 3D technology the problems of finding obstacles is already solved. Therefore this project will be focusing on using a single camera to detect obstacles. This method could easily be combined with a 3D camera to detect obstacles close to the UAV. The camera will be mounted in the front of the UAV and will be facing forwards.

The purpose of this system is to be able to detect obstacles in time to avoid an impact. Two parameters controls whether it is possible to avoid an impact. These are the velocity of the vessel and the time needed to bring the vessel to a stop. These parameters will be determined by the UAV. As this project aims at developing a generic method for different types of UAVs, general requirements cannot be determined without knowledge about the UAV. Therefore the testing of this system will be based on the parameters of the vessel available for developing this system. Furthermore the tests will generate some requirements for the vessel in order for it to be able to avoid an impact.

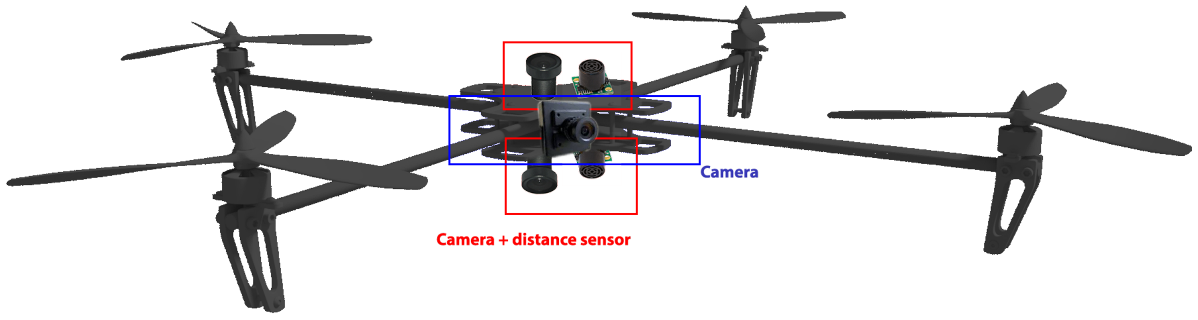

Figure 2.14: Proposed setup.

In figure 2.14 the proposed system is shown. The red boxes are the visual odometry system and the blue box is the obstacle avoidance system.

Problem statement

Now that the two problems have been narrowed down the actual problem statement is formed. As the goal is to develop a framework indoor flight with a UAV the problem statement of this project will be:

- How to develop a visual odometry capable of estimating the movement at a fixed time interval.
- How to develop a computer vision system capable of detecting and avoiding obstacles using a single camera.

3 System Composition

This chapter will be describing the methods used to solve the problems described in the problem statement. The chapter will be divided into two main sections, these are the two components described earlier. Visual odometry and obstacle avoidance.

3.1 Visual odometry

This section will be describing the methods used in the composition of the visual odometry system. The system will be described following the pipeline described in the next subsection.

3.1.1 Pipeline

To ease the description of the visual odometry system the description will follow the steps in the processing pipeline. The pipeline in shown in figure 3.1.

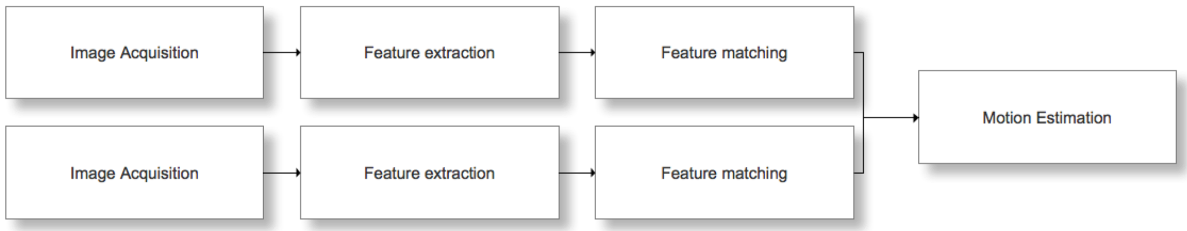

Figure 3.1: The processing pipeline of the visual odometry system.

As seen the proposed pipeline varies a bit from the typical pipeline presented in the analysis chapter figure 2.1. The proposed pipeline runs the three first steps image acquisition, feature extraction and feature matching in parallel. The reason to run these steps in parallel is because two cameras are used instead of just one. The extracted features from the two images are then combined into a single motion estimate. Also the step bundle adjustment is omitted, as it is deemed not necessary to solve the problem.

3.1.2 Image acquisition

The image acquisition step is simply grabbing the frames from the camera. Ideally the two cameras should be synchronized so that the frames are grabbed from the camera at the exact same time. However the only cameras that where available for this project, where two cheap web cams where this functionality is not available.

Distortion

When working with cameras especially cheaper models, distortion is introduced due to the lens. Two types of distortion are introduced, radial distortion and tangential distortion. Radial distortion comes from the shape of the lens, typically a wider field of view generates more distortion. Radial distortion can be seen in the image, as straight lines become bend in the image. The radial distortion is zero at the optical center and increases towards the edges of the image. The distortion is modeled using a Taylor expansion around r = 0 using three coefficients k_1, k_2, and k_3. Equation 3.1 shows how the parameters are used to correct the image. [3]

$$\begin{aligned} x_{corrected} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ x_{corrected} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{3.1}$$

In figure 3.2 a visualization of how radial distortion is generated is shown. The object has straight lines in the world, but when seen on the image plane the lines have been bend.

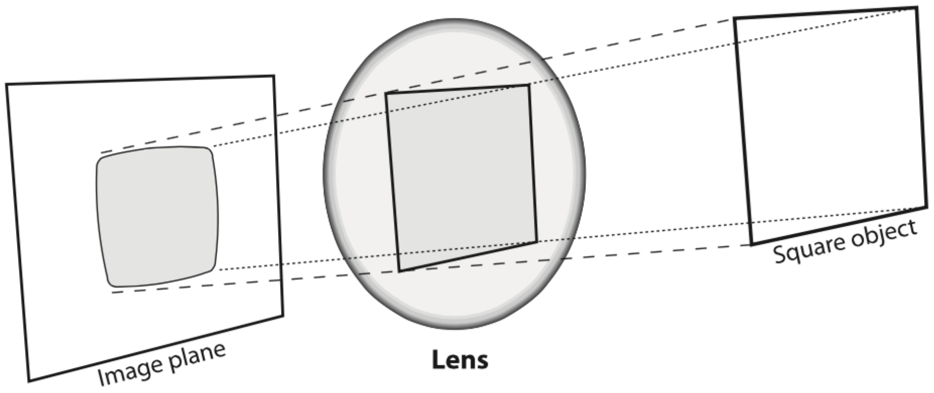

Figure 3.2: Example of how radial distortion is introduced because of the lens. [3]

Tangential distortion occurs when the image sensor and the lens is not perfectly aligned. Meaning that the lens and sensor is not parallel to each other. As with radial distortion this problem is more common when using cheap cameras. Tangential distortion can be modeled using two parameters p_1 and p_2. Equation 3.2 shows how the parameters are used to correct the image. [3]

$$\begin{aligned} x_{corrected} &= x + [2p_1 y + p_2(r^2 + 2x^2)] \\ x_{corrected} &= y + [p_1(r^2 + 2y^2) + 2p_2 x] \end{aligned} \tag{3.2}$$

In figure 3.2 a visualization of how tangential distortion is generated is shown.

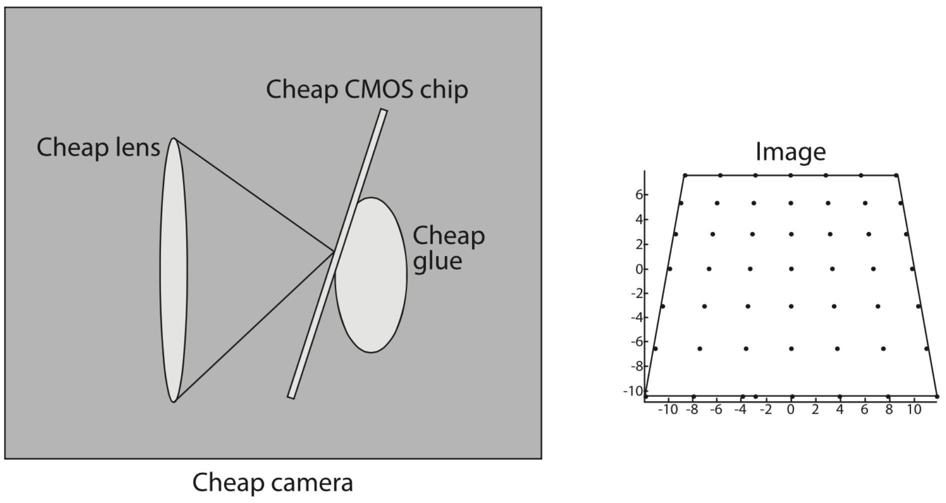

Figure 3.3: Example of how tangential distortion is introduced because the lens and sensor is not parallel. [3]

Before the captured images are sent to the next step in the pipeline they are corrected for both radial and tangential distortion.

3.1.3 Feature extraction

The features are extracted in the frames from each camera. The purpose of the feature extraction step is to find points in the frame that can be found again in the next frame. Various different methods exist for tracking of points. As this system is intended to run on board a UAV in real-time, execution time is important, as it will affect the frame rate of the system. Some of the most popular feature extractors are SIFT[11] and SURF[2]. The benefit of these feature extractors is that they are both scale invariant. A test is conducted to determine the execution time between a scale invariant feature detector and a non-scale invariant feature detector. As the feature extractors are closely related to the feature matching, the timing is measured for both the feature extraction and feature matching step. The scale variant feature detector tested is the Shi-Tomasi corner detector with Lucas-Kanade feature matching. The scale invariant feature detector tested is SURF with a nearest neighbor matching. SURF is chosen, as it is faster than SIFT. In table 3.1 the timings are shown.

	SURF		**Shi-Tomasi/Lucas-Kanade**	
	Points	Time	Points	Time
Test 1	221	65ms	400	14ms
Test 2	221	59ms	400	10ms
Test 3	208	68ms	400	10ms
Test 4	209	65ms	400	11ms
Test 5	211	67ms	400	11ms
Test 6	216	68ms	400	9ms
Test 7	198	62ms	400	10ms
Test 8	220	64ms	400	12ms
Mean	213	64.75ms	400	10.63ms

Table 3.1: Execution times SURF vs Shi-Tomasi/Lucas-Kanade.

From the table is seen that the tracking of points using the non-scale invariant method is significantly faster. As the scale will not vary much from frame to frame the Shi-Tomasi method is chosen.

Shi-Tomasi corner detector

Following the Shi-Tomasi corner detector method will be described [23]. A pixel is determined to be a corner if there is sufficient magnitude in two opposite directions. To make the corners rotation invariant the eigenvalues of the matrix A are computed see equation 3.3. I_x and I_y are the horizontal and vertical gradient images within a specified window size. These are found by convolving the image with a Sobel kernel.

$$A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \tag{3.3}$$

If both eigenvalues λ_1 and λ_2 are above a certain threshold the pixel is considered to be an edge. In figure 3.4 the decision matrix is shown. The green region represents the area where both eigenvalues are large enough to be considered a corner. The areas in red is where only one of the eigenvalues are large, this means that it is an edge.

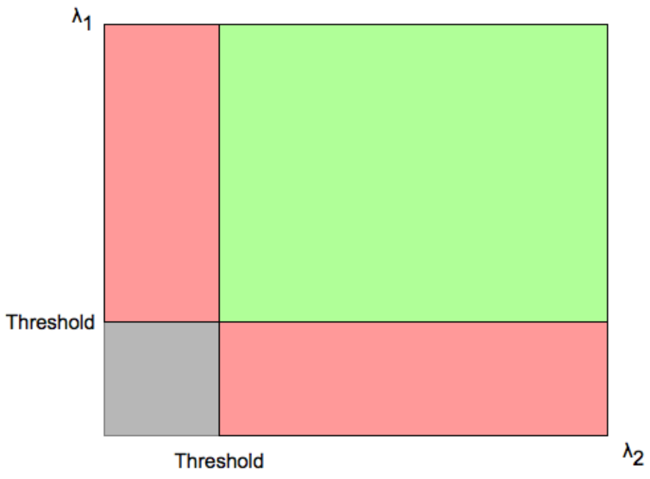

Figure 3.4: Visualization of the Shi-Tomasi method.

In figure 3.5 and 3.6 some examples of corners found using this method are shown. As seen, especially in figure 3.6, this method finds some points which are introduced due to specular highlights in the plastic. When the camera is moved these points would not be found in the next image.

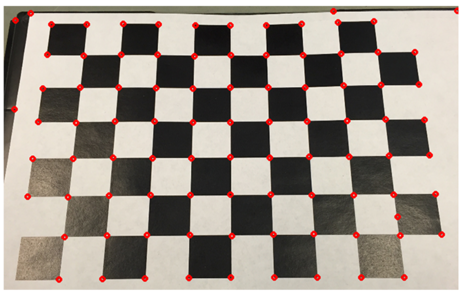

Figure 3.5: Non rotated example

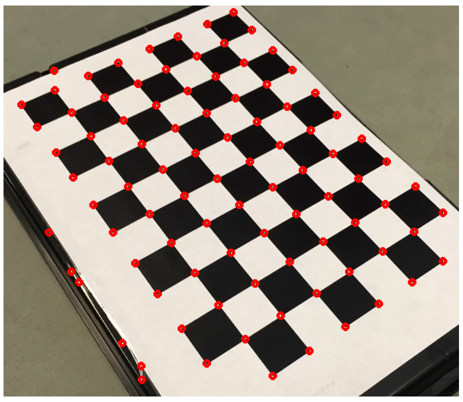

Figure 3.6: Rotated example

The algorithm also makes sure that there is sufficient spacing between the points found. This insures that all found points are not in the same region of the image. If multiple corner points are found in the same region only the one with the largest eigenvalues is used.

3.1.4 Feature matching

The purpose of the feature matching step is to find the points from the previous image in the current image. This step is very dependent on the chosen method for feature extraction. As Shi-Tomasi was chosen as the feature extractor the optical flow method proposed by Lucas-Kanade [12] is chosen as feature matcher.

Lucas-Kanade Optical Flow

The optical flow algorithm will find the a flow vector (u, v) for each point extracted in the feature extraction step of the pipeline. Each vector describes the translation from a frame at time t to the frame at time $t + 1$. In figure 3.7 these vectors are illustrated. Throughout this section the book [3] is used to describe the theory.

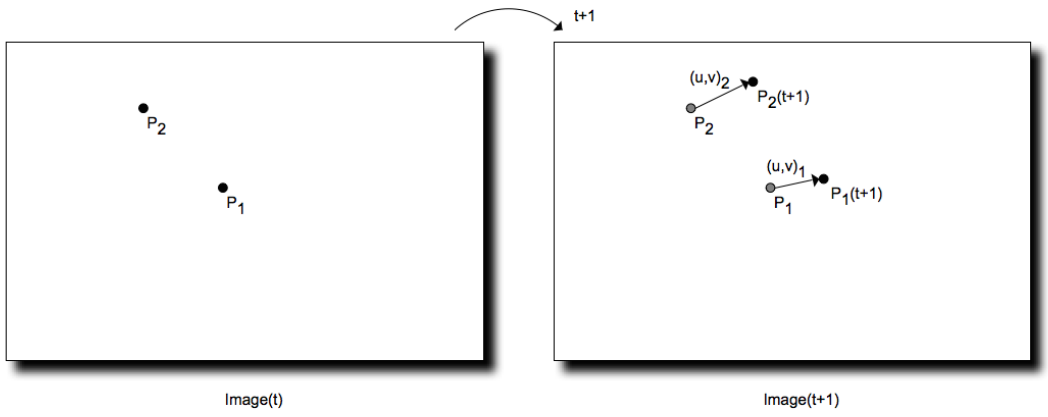

Figure 3.7: Visualization of optical flow.

The Lucas-Kanade method relies on three main assumptions:

- Pixel intensity does not change between frames.
- Neighboring pixels have similar motion.
- Movement is small between frames.

The first assumption is needed as the method operates directly on the intensity of the image, if these were to change much between each frame the algorithm would fail. The second assumption is needed as the algorithm uses a window around the point of interest. The window is necessary as (u, v) cannot be computed from just one pixel. The last assumption is needed as the algorithm uses a search window to find the vector (u, v). If the movement between frames are too large it may fall outside of the search window and the vector can not be calculated.

u and v are found by solving the equation shown in equation 3.4. I_x and I_y are the horizontal and vertical gradient images within the search window. These are found by convolving the image with the vertical and horizontal Sobel kernels. $q_1, q_2 ... q_n$ are the pixels coordinates of the pixels in the neighborhood of the point being evaluated.

$$\begin{aligned}
&I_x(q_1)v + I_y(q_1)u = -I_t(q_1)\\
&I_x(q_2)v + I_y(q_2)u = -I_t(q_2)\\
&\vdots\\
&I_x(q_n)v + I_y(q_n)u = -I_t(q_n)
\end{aligned} \tag{3.4}$$

These equations can be converted to a matrix form $Av = b$ where $v = (u, v)$

$$A = \begin{bmatrix} I_x(q_1) & I_y(q1) \\ I_x(q_1) & I_y(q1) \\ \vdots & \vdots \\ I_x(q_n) & I_y(qn) \end{bmatrix}, v = \begin{pmatrix} u \\ v \end{pmatrix}, b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix} \tag{3.5}$$

The system can be solved using the least squares method. By doing so the final solution to find the vector (u, v) can be found by solving the following equations 3.6.

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} \sum I_x(q_i)^2 & \sum I_x(q_i)I_y(q_i) \\ \sum I_y(q_i)I_x(q_i) & \sum I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_x(q_i)I_t(q_i) \\ -\sum I_y(q_i)I_t(q_i) \end{bmatrix} \tag{3.6}$$

The original Lucas-Kanade method has a problem when trying to track larger motions. If the motions are large, then a larger search window is needed to find the motion. However when a larger search window is used the coherent assumption is broken. To cope with this problem a pyramid structure is used. This pyramid consists of down sampled versions of the image. Starting from the top with the lowest resolution image the optical flow is calculated. The flow is then used as an initial guess for the position in the next level of the pyramid. The process is repeated until the lowest level of the pyramid is reached which is the full resolution image. This way a smaller search window can be used and large motions can still be found. In figure 3.8 the pyramid method is visualized.

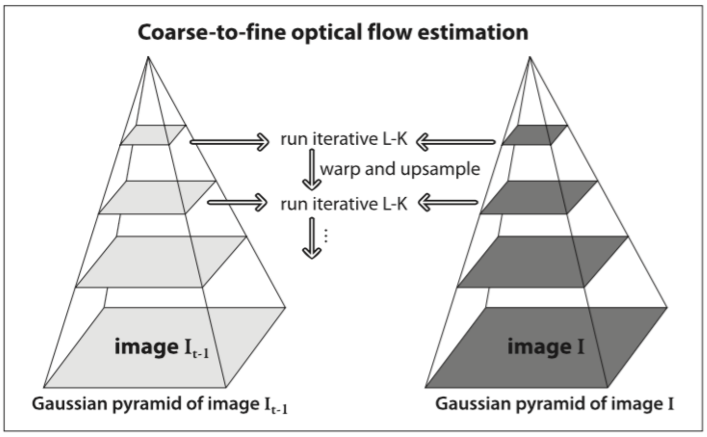
Coarse-to-fine optical flow estimation
run iterative L-K
warp and upsample
run iterative L-K
image I_{t-1}
image I
Gaussian pyramid of image I_{t-1}
Gaussian pyramid of image I

Figure 3.8: Visualization of the Lucas-Kanade pyramid approach. [3]

In figure 3.9 and 3.10 an example of the optical flow method is shown. In the first figure the corners are found as described earlier. Then the optical flow is calculated for each of these points across 10 frames. This is what is shown in the second figure where the green lines correspond to the optical flow vectors (u, v). In this example is clear to see that there is a translation in both the y direction and in the x direction. This means that the camera has been moved a bit down and to the right.

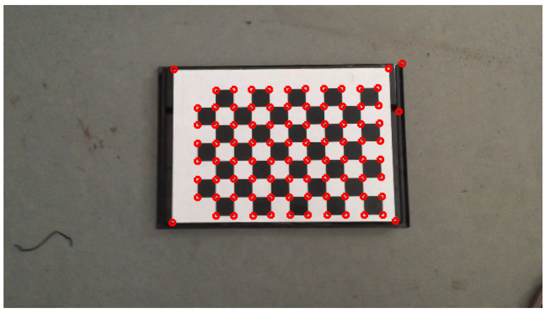

Figure 3.9: First frame where the red dots are the found corners.

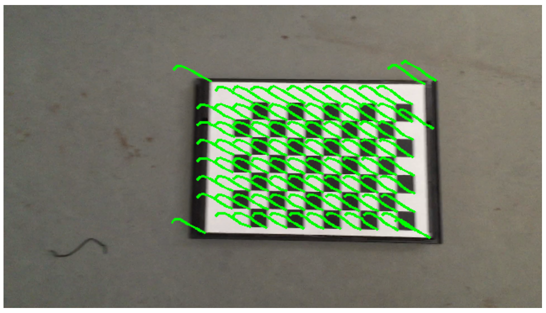

Figure 3.10: Optical flow tracked across 10 frames.

3.1.5 Motion estimation

This step is where the actual motion is estimated. The input to this step is the corresponding points between two images in time. This step will first find a motion estimate for both of the cameras. This estimate is in pixel coordinates. The next step is to combine the estimate from the upwards pointing camera and the downwards-pointing camera. Finally the last step is to map the estimate from the pixel coordinate to world coordinates.

Homography matrix

The homography matrix is used to find the motion between frames in pixels coordinates. The homography matrix relates the points from one image to another, see equation 3.7. H is the homography matrix and P_1 and P_2 are the corresponding points between two frames in homogeneous format.

$$\begin{gathered} HP_1 = P_2 \\ P_1 = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}, P_2 = \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \end{gathered} \tag{3.7}$$

This means that the homography matrix maps the points from one plane to another. This could potentially be a problem if not all the points lies on a plane. Latter it will be explained why not all the points must lie on the same plane but just most of them.

The typical way of finding the homography matrix is by using the Direct Linear Transform [5]. The equation 3.7 can be rewritten as seen in equation 3.8.

$$\begin{aligned} -h_1x_1 - h_2y_1 - h_3 + h_7x_1x_2 + h_8y_1x_2 + h_9x_2 = 0 \\ -h_4x_1 - h_5y_1 - h_6 + h_7x_1y_2 + h_8y_1y_2 + h_9y_2 = 0 \end{aligned} \tag{3.8}$$

These equations can be written in a matrix form as $Ah = 0$ with 4 corresponding points which is the minimum required points to estimate the homography matrix. For simplicity the corresponding points will be described as follows $P_1 = (x, y)^T, P_2 = (u, v)^T$.

In equation 3.9 the equations are seen in matrix form with 4 corresponding points.

$$A = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & u_1x_1 & u_1y_1 & u_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & v_1x_1 & v_1y_1 & v_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & u_2x_2 & u_2y_2 & u_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & v_2x_2 & v_2y_2 & v_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & u_3x_3 & u_3y_3 & u_3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & v_3x_3 & v_3y_3 & v_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & u_4x_4 & u_4y_4 & u_4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & v_4x_4 & v_4y_4 & v_4 \end{bmatrix}$$
$$h = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{pmatrix}^T \tag{3.9}$$

From A the vector containing the values of h can be found. h is found by using Singular value decomposition. h corresponds the last column of V where $A = U\Sigma V^T$. This leads to the final composition of H seen in 3.10.

$$A = U\Sigma V^T$$
$$H = \begin{bmatrix} V_{1,9} & V_{2,9} & V_{3,9} \\ V_{4,9} & V_{5,9} & V_{6,9} \\ V_{7,9} & V_{8,9} & V_{9,9} \end{bmatrix} \frac{1}{V_{9,9}} \tag{3.10}$$

As an example the image from the Lucas-Kanade method description is used. The homography matrix is estimated using the points gathered from the optical flow. These are the green lines, which are tracked across 10 frames. The flow is shown in figure 3.11 and the corresponding homography matrix is shown in equation 3.11.

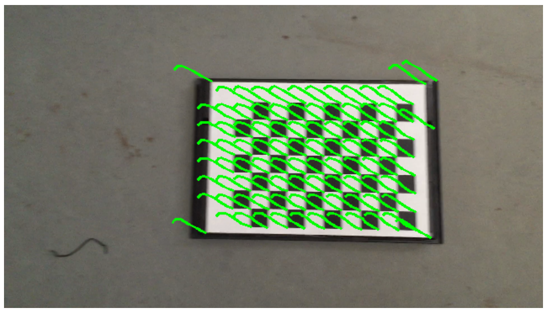

Figure 3.11: Optical flow tracked across 10 frames.

$$H = \begin{bmatrix} 0.9598 & -0.0337 & 52.8250 \\ 0.0182 & 0.9659 & 12.0100 \\ 0.000 & 0.0000 & 1 \end{bmatrix} \tag{3.11}$$

From the homography matrix a motion estimate can be extracted.

RANSAC

As mentioned earlier using the homography matrix assumes that the points are on the same plane. To make the system robust to points that are not on same plane as the majority of the points, a method called Random sample consensus (RANSAC) is utilized [7]. This method will also sort out noise generated by faulty calculated optical flows. RANSAC is a widely used method for generating models from noisy data. The typical way of estimating a model from a over determined system is by using least squares approximation. However least squares approximation is very sensitive to outliers in the dataset. Points which are not on the same plane as the majority of the points, would appear as outliers in the dataset. The RANSAC algorithm ignores these outliers in an iterative way. In figure 3.12 the difference between the result of RANSAC and least squares approximation is shown on a noisy dataset where a linear model is wanted.

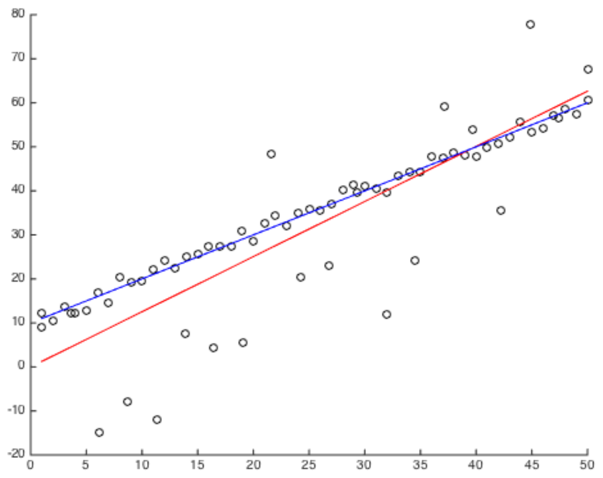

Figure 3.12: RANSAC and Least squares difference. **Red Line:** Least squares approximation. **Blue Line:** RANSAC estimation.

RANSAC works by randomly selecting a minimum of points required to generate a model. In the case of a linear model as shown in the previous example this would be 2 points. From these points a model is generated. Afterwards the distance from all points to the model is calculated. If the distance if above a certain threshold, the point is considered as an outlier. If the distance is below the threshold the point is considered as an inlier. Afterwards a ratio between inliers and outliers can be calculated. If this ratio is high enough meaning that the most points are below the threshold the model is considered to be good enough. If not then the process is started again. Different implementations exists, one method is to keep restarting this process until a fixed number of iterations is reached and then the model returning the best ratio is used. Another approach is to use least squares approximation on the points in the inlier group to refine the model.

In the case of using RANSAC to estimate the homography matrix the model is the homography matrix itself. As mentioned earlier 4 points is required to estimate the matrix. The threshold function applied to the remaining points is simply the distance from the input point multiplied with the matrix to the output point see equation 3.12.

$$\begin{gathered} E = ||P_1 - HP_2|| \\ E = \text{Distance}, P_1 = \text{Point in source}, P_2 = \text{Point in destination} \end{gathered} \tag{3.12}$$

Again the example from the optical flow section is used. This time the homography estimation is using RANSAC to find the homography matrix. In figure 3.13 the flow tracked across 10 frames is shown. In figure 3.14 the filtered flow is shown. The green lines are the ones matching the homography matrix and the red ones are those, which are, filtered away using RANSAC.

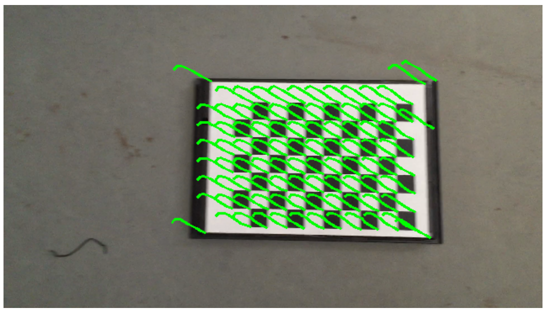

Figure 3.13: Optical flow tracked across 10 frames.

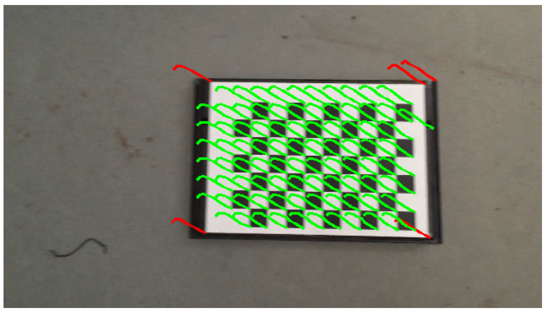

Figure 3.14: Filtered flow for homography.

Distance

To be able to map the translation from pixels to actual distances in world coordinates a distance sensor is placed along with each camera. This distance sensor returns the distance to the object in front of the sensor in meters. The sensor is placed parallel to the camera. This means that the distance should correspond to the distance from the camera to the plane where the tracked features are located.

Combination

The final step is combine the two calculated homography matrixes into a single motion estimate. The first step is to decompose the homography matrix into the following parts R which is the rotation of the plane. T which is the translation of the plane and finally S which is the skewing of the plane. The decomposition is shown in equation 3.13.

$$\begin{gathered} H = \begin{bmatrix} R & T \\ S & 1 \end{bmatrix} \\ R = \begin{bmatrix} r_1 & r_2 \\ r_3 & r_4 \end{bmatrix}, T = \begin{pmatrix} t_x \\ t_y \end{pmatrix}, S = \begin{pmatrix} s_1 & s_2 \end{pmatrix} \end{gathered} \tag{3.13}$$

As it is the translation and rotation are the parameters that are to be estimated the skewing part of the homography matrix is ignored. Translation can be generated by both actual translation of the camera, which is the translation that needs to be extracted. However translation can also be generated by the camera rotating around another axis than the optical axis, these translations needs to be removed. This is where the advantage of having two cameras comes into play. If the camera is rotated around another axis than the optical axis the translation from one of the cameras should even out the translation in the other camera. This is illustrated in figure 3.15.

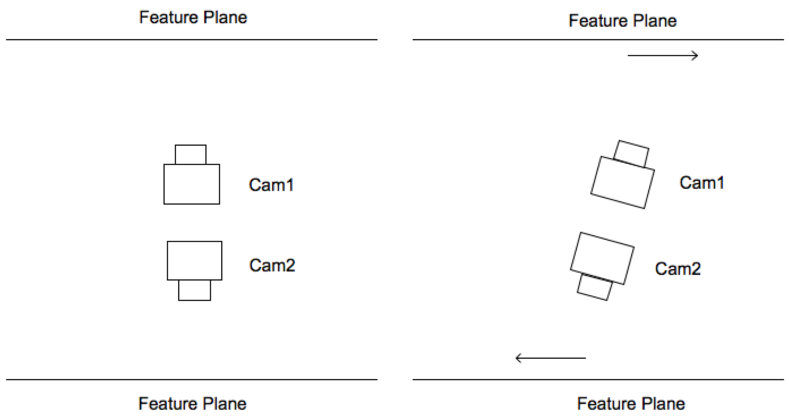

Figure 3.15

Because the translation is dependent on the distance to the plane the translations needs to be in same format before they are combined. Therefore the translation is mapped to meters. The mapping is done by multiplying the translation in pixels by the inverse focal lengths in pixels. This normalized translation can now be scaled using the distance. Equation 3.14 shown how the translation is mapped from pixels to meters.

$$T_m = T_{px} F^{-1} D$$
$$T_m = \begin{pmatrix} T_{x meters} & T_{y meters} \end{pmatrix}, T_{px} = \begin{pmatrix} T_{x pixels} & T_{y pixels} \end{pmatrix}, F = \begin{bmatrix} F_x & 0 \\ 0 & F_y \end{bmatrix}, D = Distance \tag{3.14}$$

The calculation of the focal length in pixels is shown in equation 3.15. f_x and f_y are the vertical and horizontal focal lengths. W and H are the width and height of the image sensor. w and h is the vertical and horizontal resolution of the image.

$$F_x = f_x \frac{W}{w}, F_y = f_y \frac{H}{h} \tag{3.15}$$

Now that the translations are in the same format they can be combined into a single translation estimate. The combination is simply the one translation added to the other and then divided by two.

$$T_m = \frac{T_{top} F_{top}^{-1} D_{top} + T_{Bottom} F_{Bottom}^{-1} D_{bottom}}{2} \tag{3.16}$$

In equation 3.16 the final equation to calculate the estimated translation is shown.

As the rotation will be the same in both of the two homography matrixes the rotation is simply estimated as a mean between the two extracted rotation matrixes R_{top} and R_{bottom}. The equation to estimate the combined rotation matrix can be seen in equation 3.17.

$$R = \frac{R_{top} + R_{bottom}}{2} \tag{3.17}$$

The actual angle can be extracted from the rotation matrix using equation 3.18.

$$\begin{gathered} angle = atan2(R_3, R_1) \\ R = \begin{bmatrix} R_1 & R_2 \\ R_3 & R_4 \end{bmatrix} \end{gathered} \tag{3.18}$$

Smoothing

From testing the system it has been determined that the estimates suffers from some noise. To cope with this noise a smoothing filter can be applied to the data. It is noted that the results presented in chapter 6 does not have this filter applied.

Because the cameras are not synchronized the motion is not captured in the same frames, which will lead to some undesired spikes in the output from the system. By averaging the result over a few frames the output becomes much more smooth. The function used to smooth the output is seen in equation 3.19. Where x_t and y_t are the motion estimate at time t.

$$\begin{gathered} x_t = \frac{x_t + x_{t-1} + x_{t-2} + x_{t-3}}{4} \\ y_t = \frac{y_t + y_{t-1} + y_{t-2} + y_{t-3}}{4} \end{gathered} \tag{3.19}$$

Filtering

To make the system a bit more robust to errors in the estimates the values are filtered after they are calculated. From testing the system as described up till this point, some potential error causes have been found. The first problem arises when an incorrect homography matrix is found. In such cases the three estimates are all set to zero. Whether or not the matrix is incorrect is determined by examining the result of the three estimates. If either of the three conditions shown in 3.20 are true all estimates are set zero.

$$\begin{gathered} xMotion > 6cm \\ yMotion > 6cm \\ Rotation > 6^\circ \end{gathered} \tag{3.20}$$

These thresholds have been determined by examining the results from the test data.

The second problem that arises is when rotation is present. When the platform rotates, wrong translations are calculated. To solve this problem the x motion and y motion is set to zero when rotation is larger than 1° per sample. Again this threshold has been determined by analyzing the test data.

3.2 Obstacle avoidance

This section will be describing the methods used in the composition of the obstacle avoidance system. The system will be described following the pipeline described in the next subsection.

3.2.1 Pipeline

To ease the description of the obstacle avoidance system the description will follow the pipeline composing the obstacle avoidance system. The pipeline is shown in figure 3.16.

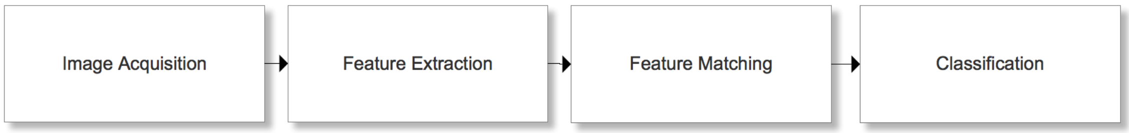

Figure 3.16: The processing pipeline of the obstacle avoidance system.

Image acquisition is responsible for data gathering, feature extraction is responsible for finding points, feature matching is the process of finding the extracted points in the next frame and finally classification will be the step responsible for determining whether or not obstacles are present in the scene.

3.2.2 Image acquisition

In this step a single image is grabbed from the camera on the UAV. Along with the image information from the IMU onboard the UAV is captured. The information from the IMU is the 3 angles yaw, pitch, roll and the horizontal velocity vx and vy.

Distortion

To be able to see as much as possible of the scene the UAV is approaching, it is beneficial to use a camera with a large FOV. When using cameras with a large FOV the effect of radial distortion becomes larger. To cope with this the image from the sensor is undistorted before being sent to the next step in the pipeline. In figure 3.17 and 3.18 a distorted image and the undistorted image from a camera with a FOV of 92°.

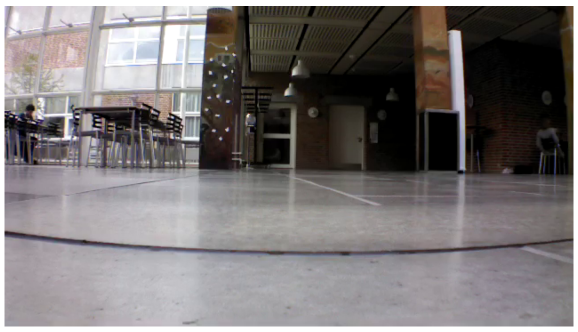

Figure 3.17: Distorted image

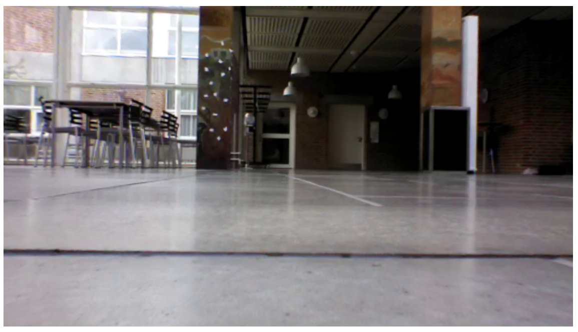

Figure 3.18: Undistorted image

In the first image the effect of the radial distortion is clearly seen as the straight lines on the floor are warped to bend lines. These lines are straight in the undistorted image. For an in depth description of distortion and how it is modeled see section 3.1.2.

Rotation

Before the image is sent to the next step of the pipeline, the image is rotated with the roll angle of the UAV. This helps to stabilize the image when the UAV is moving. In figure 3.19 and 3.20 the result of rotating the image is shown.

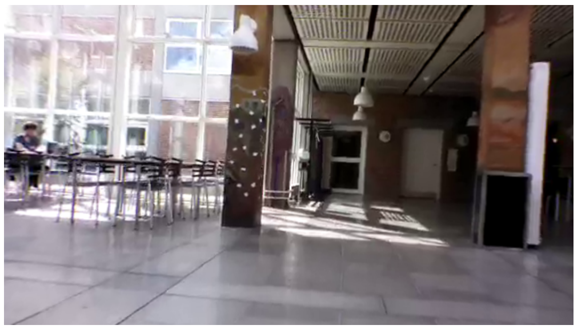

Figure 3.19: Before rotation

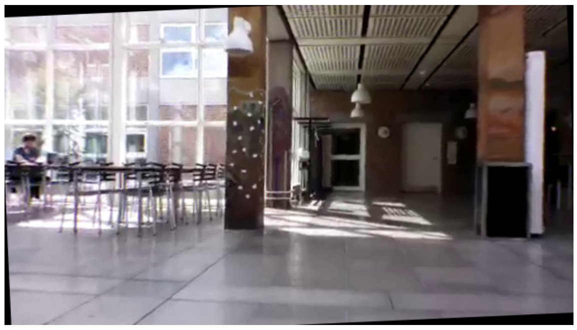

Figure 3.20: After Rotation

From the images it is seen that the rotation caused by the drone rolling is removed. However this is at the cost of cropping some of the image away and introducing the black borders in the edges of the image. This is not a problem as the edge regions of the image are not of a big interest.

3.2.3 Feature extraction + Feature matching

As the feature extraction and the feature matching steps of the pipeline have been described in detail in the visual odometry system these two steps have been combined in this section. This section will be describing the different methods that have been considered.
Two different methods have been considered for the feature extraction and feature matching steps, which extracts the optical flow. The first considered method is Farneback optical flow [6]. The advantage of using this method is that a dense optical flow map is generated; this means more information is available for the classification step. The other method considered uses Shi-Tomasi corner detector to find corner points and Lucas-Kanade optical flow to find the points in the next frame. This approach is identical to the method used in the visual odometry system. This method generates a spatial flow map, which means less information is available for the classification step. As this system is supposed to run in real time the execution time needed to process each frame is of great importance. Therefor a simple test is executed to estimate the time requirement for both methods.

The test is carried out by testing the execution time for the calculation of the optical flow between two frames using both methods. For the Shi-Tomasi/Lucas-Kanade method the flow is calculated for 1000 points. The image resolution is 640x360 pixels. The test is carried out 8 times and the results are displayed in table 3.2.

	Farneback	**Shi-Tomasi/Lucas-Kanade**
Test 1	241ms	7ms
Test 2	240ms	7ms
Test 3	239ms	8ms
Test 4	240ms	7ms
Test 5	239ms	7ms
Test 6	242ms	7ms
Test 7	238ms	7ms
Test 8	237ms	7ms
Mean	239.5ms	7.13ms

Table 3.2: Execution times for optical flow.

From the table it is clearly seen that the dense optical flow is much harder to calculate compared to the spatial optical flow. The spatial optical flow can be calculated at a frame rate of 137 fps where the dense can only be calculated at a frame rate of 4.18 fps. For this reason the spatial optical flow is used to generate the optical flow. For an in depth description of the Shi-Tomasi/Lucas-kanade method see section 3.1.3 and 3.1.4.

Trajectories

The optical flow is calculated from frame to frame. When a point is successfully tracked between two frames the same point is tracked again in the following frames and the flow is added to a vector for each frame. This results in trajectories across multiple frames, which are sent to the classification step of the pipeline. For each sample new points are added using Shi-Tomasi. The distance from these points to points already in the system is calculated and if the distance is below a threshold the points are not added. This ensures that points already belonging to a trajectory are not added again. In figure 3.21 the process of generating trajectories is shown.

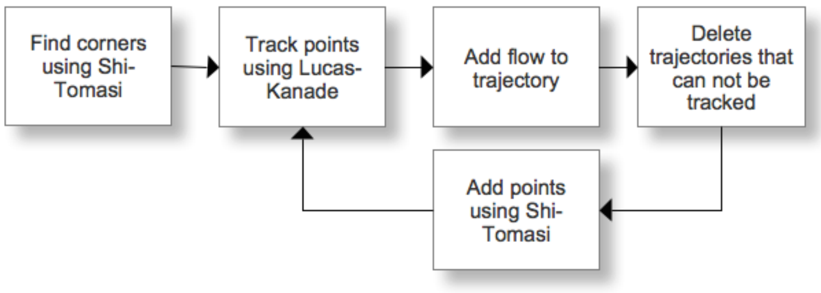
Find corners using Shi-Tomasi
Track points using Lucas-Kanade
Add flow to trajectory
Delete trajectories that can not be tracked
Add points using Shi-Tomasi

Figure 3.21: How trajectories are generated.

In figure 3.22 an example of the trajectories generated for 15 frames is shown.

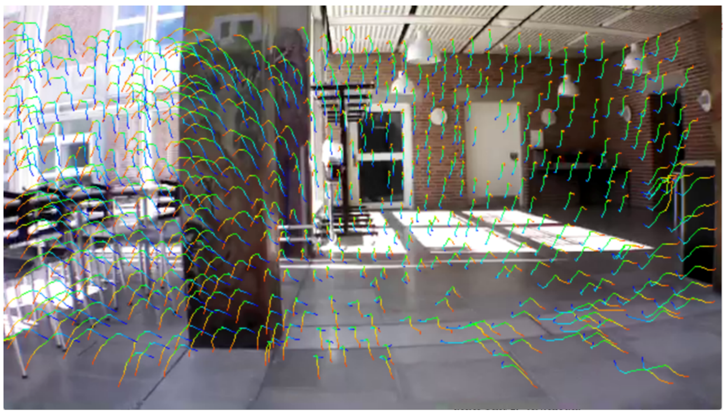

Figure 3.22: Example of trajectories generated for 15 frames.

3.2.4 Classification

From the analysis it is known that the flow behaves in different ways depending on the region of the image. Therefore the classification step is divided into two different sections. One section describes the method for estimating obstacles in the left and the right region of the image. The other section describes the method used for estimating obstacles in the center region of image. Both methods rely on the trajectories extracted from feature extraction and feature matching. For an alternative method tried for classification see appendix B.

The different sections are divided based on how much an object would fill when close to the camera. The center region is 80 pixels to both sides of the center of the image. This corresponds to an object being 20cm to the left or right of the center at a distance of 1.5 meters. This calculation is with a camera with a FOV of 92°. If another FOV is used the regions may need to be altered.

Left and right region

When the camera moves towards objects to the left or right of the camera, these objects will start to move from the center region of the image towards the outer region of the image. The closer an object is to the camera the faster the object will move towards the outer region of the image. A simple plot is generated to see how the position differs depending on the distance. In figure 3.23 the x pixel coordinate is shown for an object that is 0.5m to the right of a camera and the distance is changed from 3 to 2 meters and from 6 to 5 meters. The pixel coordinate is calculated using equation 3.21.

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = K \begin{pmatrix} 0.5 \\ 0 \\ D - \frac{1}{30} i \end{pmatrix}^T \tag{3.21}$$
$$D = \text{distance to object}, K = \text{Camera matrix}$$

The camera matrix K if the for a camera with a FOV of 92°. This scenario would correspond to the UAV flying towards the object at 1 meter per second with a frame rate of 30 fps.

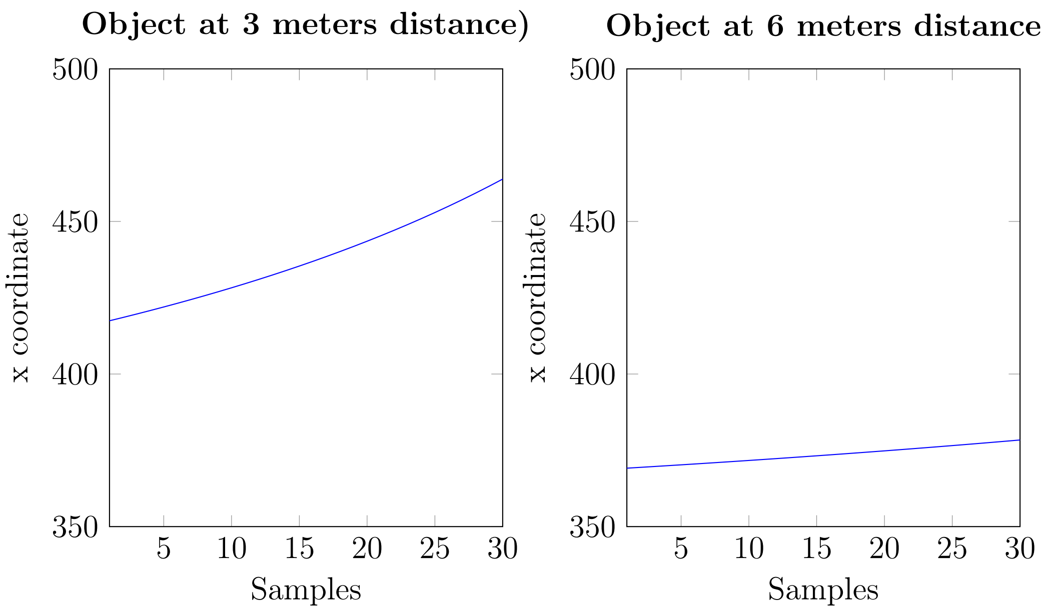

Figure 3.23: The position of the object at different distances.

From the figure it is seen that obstacles that are closer to the camera generates a steeper slope. An example sequence is extracted from the initial test data to see how this assumption fits real world data. In the example two trajectories are selected, one generated by the pillar which is close to the camera and one from the background. In figure 3.24 the two points generating the trajectories are shown, and in figure 3.25 the two curves generated are shown.

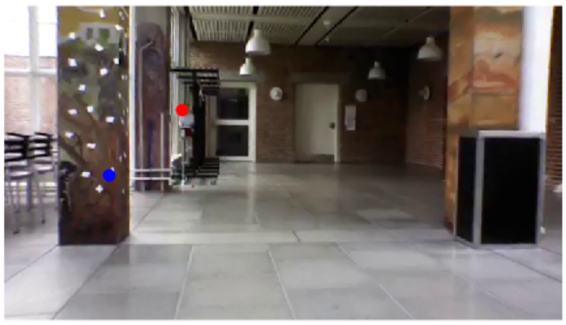

Figure 3.24: The two corner points

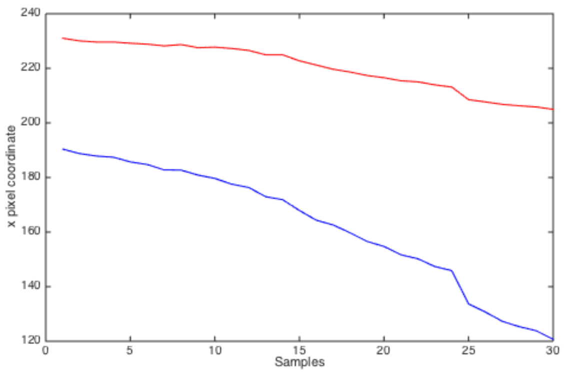

Figure 3.25: The curve of the two points.

In the figures it is clearly shown that the slope of the point on the pillar generates a much steeper slope than the one from the background.

Estimating the slope

The next step is to actually classify the slopes as obstacles close to the camera. For each trajectory the slope of the x pixel coordinate is calculated using least square linear regression. The regression line is calculated for the last 30 points of each trajectory and only if the trajectory contains at least 30 points. The formula for calculating the slope is shown in equation 3.22 where a is the slope.

$$
\begin{aligned}
y &= ax + b \\
a &= \frac{\sum_i^n (x_i y_y) - n\overline{XY}}{\sum_i^n (x_i^2) - n\overline{X}^2} \\
b &= \overline{Y} - b\overline{X}
\end{aligned}
\tag{3.22}
$$

Scaling

Because the slope of objects closer to the edge of the image generates a steeper slope than objects close to the image center the slope is scaled according to the position of the last point in the trajectory. The scale is interpolated from 3 to 1 from pixel coordinate 0 to 320 and from 1 to 3 from pixel coordinate 320 to 640. The scaling value is found by testing with the different data sets from the initial tests. In figure 3.26 the scaling of the slopes is illustrated.

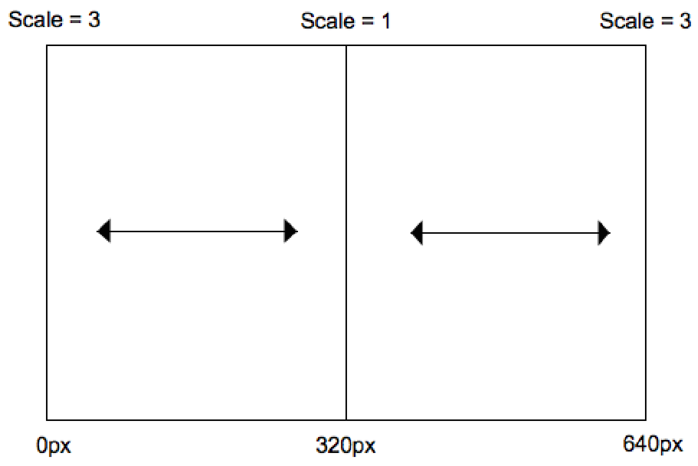

Figure 3.26: The scaling of the calculated slopes.

Determining if an obstacle is present

From tests it is known that some errors occur on some of the trajectories. These errors occur because the point position is not estimated correctly from the feature matching step. An example of this can be seen in figure 3.27. In the figure a circle is draw around the last point of each trajectory. The color corresponds to the scaled slope from blue to red. Where blue is 1 and red is 4.

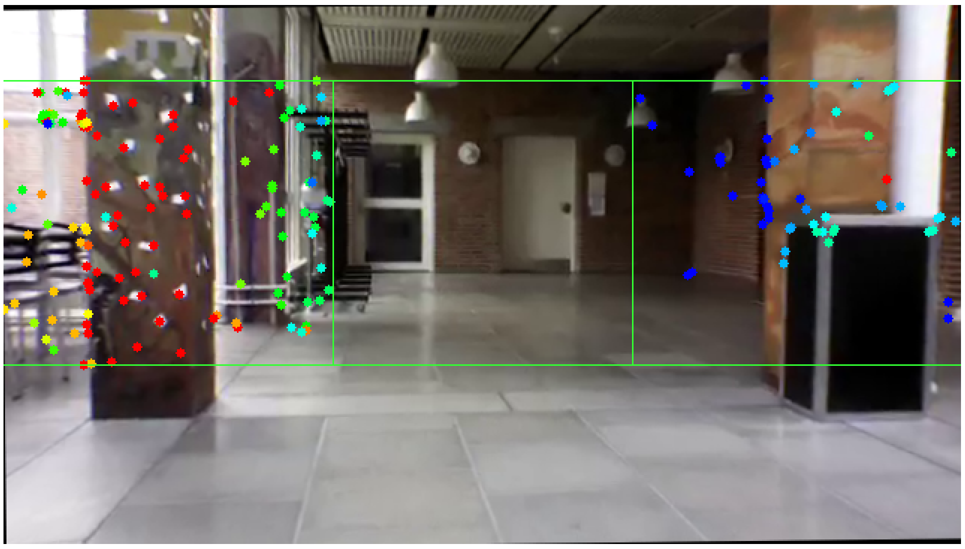

Figure 3.27: The scaled slope of the trajectories.

In the figure it is clearly seen that the trajectories generated by the pillar to the left is draw in a red color. The points lying on other surfaces have a smaller slope and are therefore not drawn in red. However there are some points, which does not lie on the pillar that is still marked as red. Because these errors occur the region cannot be marked as containing an obstacle if just one trajectory is above the slope threshold.

To find groups of trajectories the trajectories are clustered using euclidean distance. The trajectories are clustered if the distance between them is below 20 pixels and if the difference between their slopes is below 0.5. If any clusters are found with a size larger than 8 points and a scaled slope above 3.5 the region is considered as not safe. These values have been found from tests using the data sets from the initial tests. In figure 3.28 the final output from the left and right section obstacle detection is shown.

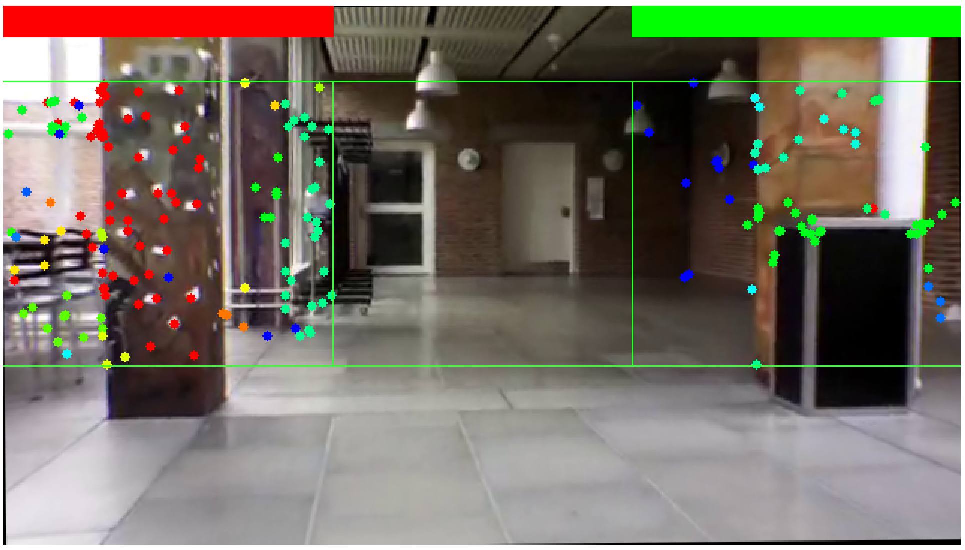

Figure 3.28: The final output from the left and right section.

In the figure the last point of the trajectories from the clusters are draw from blue to red where the color represents the scaled slope from 1 to 4. The two colored rectangles in the top of the image shows if the system considers the region to contain obstacles or not. It can be seen that the pillar to the left of the camera has been detected and therefore the rectangle in the left section is colored red. The distance to the pillar to the right is larger and has therefore not been considered as an obstacle. This is shown as a green rectangle in the right section.

Center region

When the camera moves towards an object in the center of the camera frame, the object will become larger. The closer the object is to the camera the more it will grow between frames. This means that object closer to the camera will grow faster than object further away. A simple plot is generated to see how the scale differs depending on the distance to an object. In figure 3.29 4 points on an object is drawn, the object is in the exact middle of the frame and have a size of 20x20cm. To the left 30 frames is draw of the object approaching from a distance of 2 meters and to the right the same object at a distance of 4 meters.

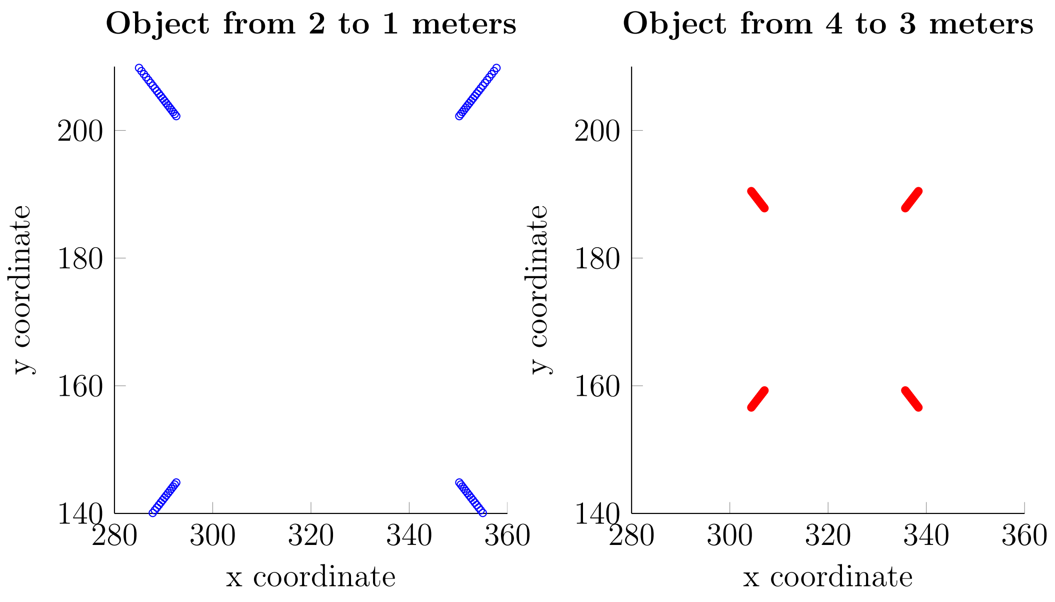

Figure 3.29: The scaling of an object based according to distance.

From the figure it is clearly seen that the object grows faster i.e. the scaling between the frames is larger as the object comes closer to the camera.

Extracting the scale

To have a parameter to describe the scale of all the points in the center region of the image, the homography matrix between each frame is extracted. An alternative approach have been tested to extract the scale, for a description of this method see appendix A. To see a detailed explanation of the homography matrix and how it is extracted see section 3.1.5. From the homography a scaling factor can be extracted. In equation 3.23 the equation used for extracting the scale from the homography matrix is shown.

$$S = \sqrt{H_{1,1}^2 + H_{1,2}^2}$$
$$S = \text{Scale}, H = \begin{bmatrix} H_{1,1} & H_{1,2} & H_{1,3} \\ H_{2,1} & H_{2,2} & H_{2,3} \\ H_{3,1} & H_{3,2} & H_{3,3} \end{bmatrix} \tag{3.23}$$

The scale extracted is the scale in the horizontal direction, as the objects should scale uniformly the scale in the horizontal direction is the same as the scale in the vertical direction. In figure 3.30 the scale of the object from before is plotted from 2 to 1 meters and from 4 to 3 meters.

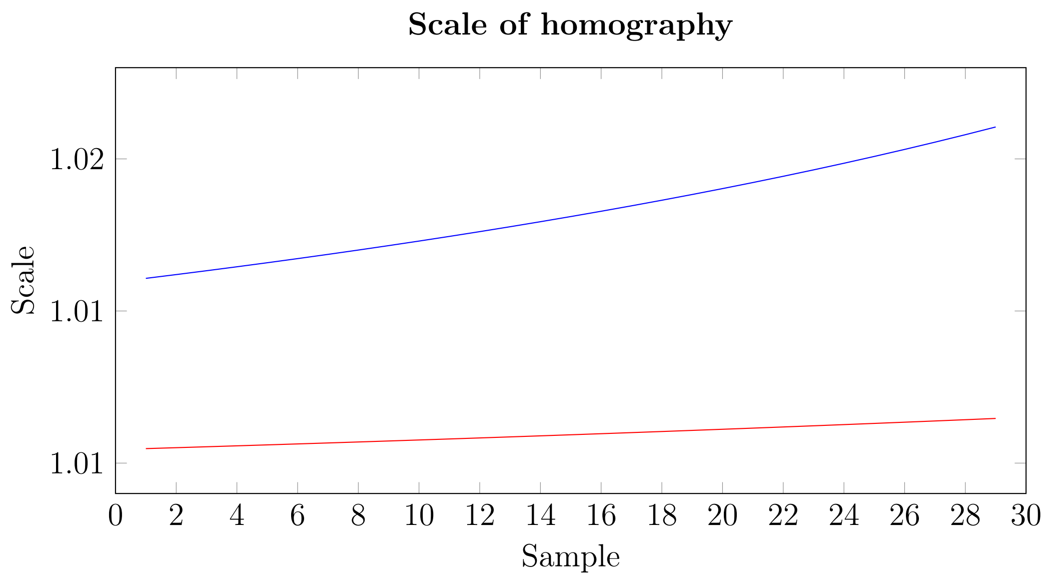

Figure 3.30: The scaling extracted from homography. **Blue:** From 2 to 1 meters **Red:** From 4 to 3 meters

From the figure it is clearly seen that the scale increases as the object comes closer to the camera. The scale of the object closer to the camera increases faster than the one from the object further away. Also it is noted that the increase in scale follows a second order polynomial. Two example sequences are extracted from the test data to see how well these assumptions fit real world data. Two sequences are selected; one where the camera moves straight towards an obstacle, and one where no obstacles is present. In figure 3.31 the scenario without obstacles is seen, the frame corresponds to sample 30 in figure 3.32.

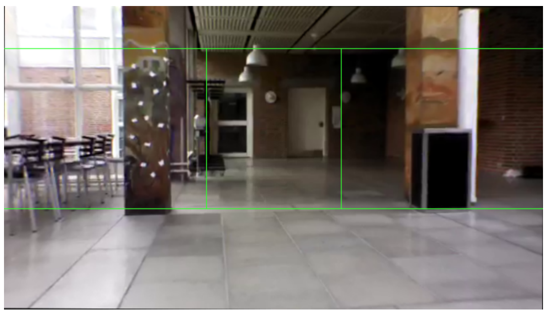

Figure 3.31: The scenario with no obstacles in center region

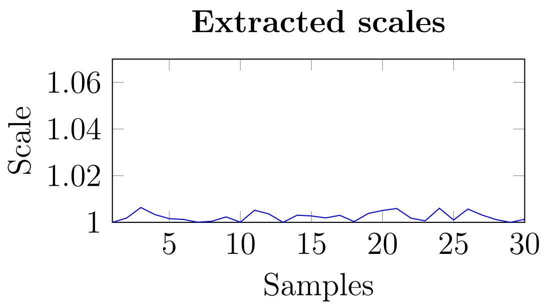

Figure 3.32: The scale extracted from the center region

In the plot it is seen that the scale between frames lays around 1, which fits with the assumptions that the scale only increases when an obstacle is approaching. From the plot it is seen that some noise is generated so that some samples have a higher scale than they should. This is probably because of points not tracked entirely correct from the feature matching step. Based on this result it is deemed possible to determine that no obstacle is present.

The same test is carried out with an obstacle present in the center region of the image. In figure 3.33 the scenario with an obstacles is seen, the frame corresponds to sample 30 in figure 3.34.

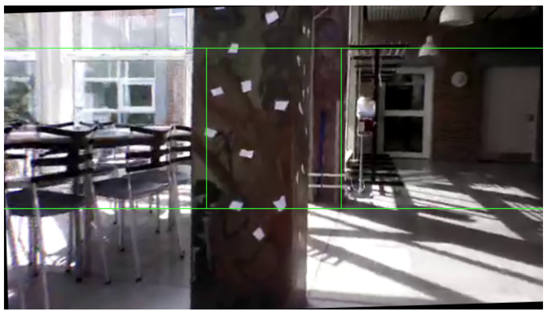

Figure 3.33: The scenario with an obstacles in center region

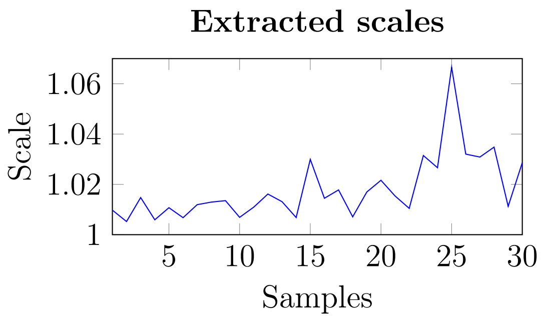

Figure 3.34: The scale extracted from the center region

In the plot is seen that the scale is higher and increases towards the last samples when an obstacle is present in the center region. As with the scales from the previous test some noise is present in the scales. It seems that the noise is larger when an obstacle is present compared with result where no obstacle was present. This may be due to the fact that not the entire center region belongs to the obstacle, which means that all points will not lie on the same plane. By applying RANSAC as described in section 3.1.5 the homography estimation should be more robust to this problem. Based on this result it is deemed possible to determine that an obstacle is present

Filtering

As previously mentioned, the extracted scales contain some noise. Before the scales are further processed a filtering process is applied. The purpose of the filter is to remove some of the noise and make the scales fit the desired result better. In equation 3.24 the filter applied is shown.

$$\begin{gathered} F_i = \alpha F_{i-1} + (1-\alpha)S_i \\ F = \text{Filtered values}, S = \text{Unfiltered values}, \alpha = \text{Filter constant} \end{gathered} \tag{3.24}$$

The filter is a simple low pass filter controlled by the filter constant α. A high α value will result in values not affected by large spikes in the samples. If a low α value is chosen the effect of spikes will be more prominent and the filter will be faster to react to changes in the samples. In this scenario a slowly reacting filter is wanted which culls large spikes in the data. Therefore a α value of 0.9 is chosen. In figure 3.35 the affect of the filter is shown.

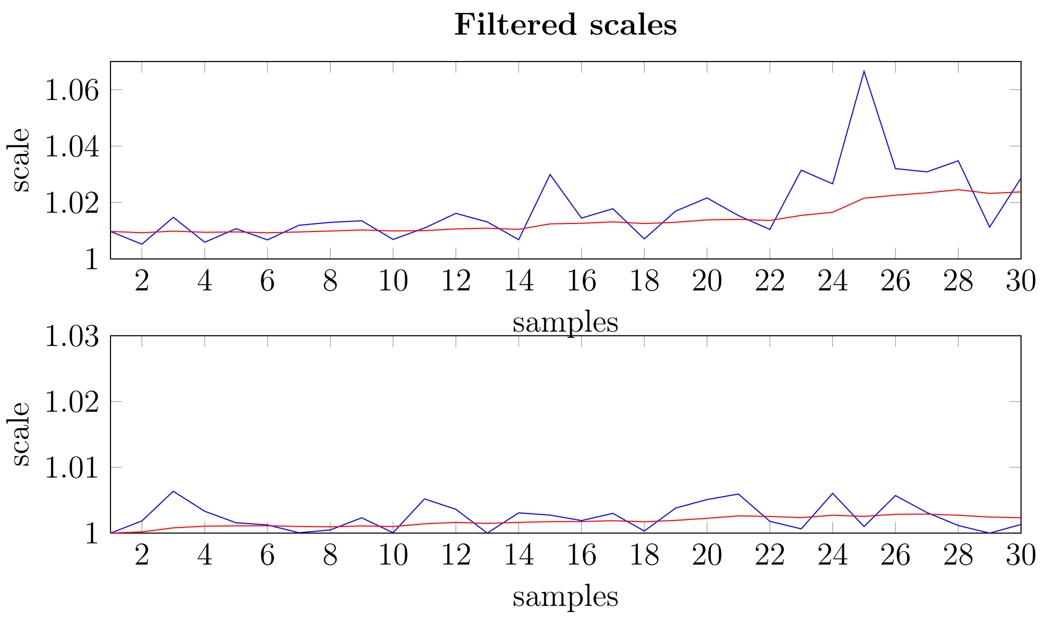

Figure 3.35: The scaling extracted from homography. **Blue:** Extracted scales **Red:** Filtered scales

In the plots is seen that the filtered values resembles the scales, which were calculated for a similar scenario.

Slope estimation

As previously noted, the scales increases following a second order polynomial. Therefore a second order least squares polynomial fitting is applied to the data in order to parameterize the data. In equation 3.25 the calculation of the second order least squares polynomial fit is shown.

$$
\begin{aligned}
y &= ax^2 + bx + c \\
a &= \frac{Sx2y\,Sxx - Sxy\,Sxx2}{Sxx\,Sx2x2 - Sxx2^2} \\
b &= \frac{Sxy\,Sx2x2 - Sx2y\,Sxx2}{Sxx\,Sx2x2 - Sxx2^2} \\
c &= \overline{Y} - b\,\overline{X} - a\sum(x^2)\,\frac{1}{n} \\
Sxx &= \sum(x^2) - \frac{1}{n}\sum(x)^2 \\
Sxy &= \sum(x\,y) - \frac{1}{n}\sum(x)\sum(y) \\
Sxx2 &= \sum(x^3) - \frac{1}{n}\sum(x^2)\sum(x) \\
Sx2y &= \sum(x^2\,y) - \frac{1}{n}\sum(x^2)\sum(y) \\
Sx2x2 &= \sum(x^4) - \frac{1}{n}\sum(x^2)^2
\end{aligned}
\tag{3.25}
$$

From the equation the 3 parameters describing the slope of the data can be found. In figure 3.36 the samples, the filtered values and the least squares solution is shown.

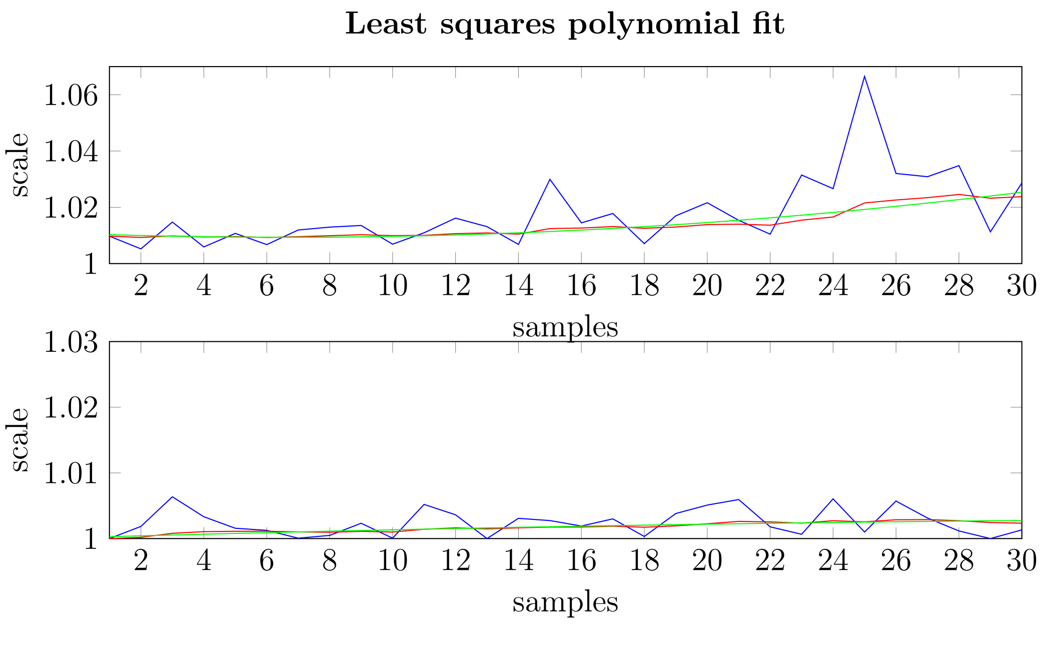

Figure 3.36: The scaling extracted from homography. **Blue:** Extracted scales. **Red:** Filtered scales. **Green:** The least squares solution.

Determining if an obstacle is present

The final step is to determine if an obstacle is present or not. The decision is based on the polynomial coefficients found using least squares. From the previous tests it is known that a scale that increases towards the last samples means that an obstacle is present. Therefore the slope of the tangent line at the last sample is used to determine is an obstacle is present. The tangent line is found by differentiating the second order polynomial at sample 30. The calculation of the slope is shown in equation 3.26.

$$slope = 2 \times a \times 30 + b \tag{3.26}$$

If the slope is above a threshold of 0.0013 the center region is marked as containing an obstacle. This threshold value has been found from testing the method against different data sets both with and without obstacles. As with the left and right region obstacle avoidance system, a rectangle is draw in the top of the image. If no obstacles are present the rectangle will be green. If an obstacle is detected the rectangle will turn red. In figure 3.37 an example is shown. In the center the trajectories used to calculate scale is shown and in the top the status rectangle is shown.

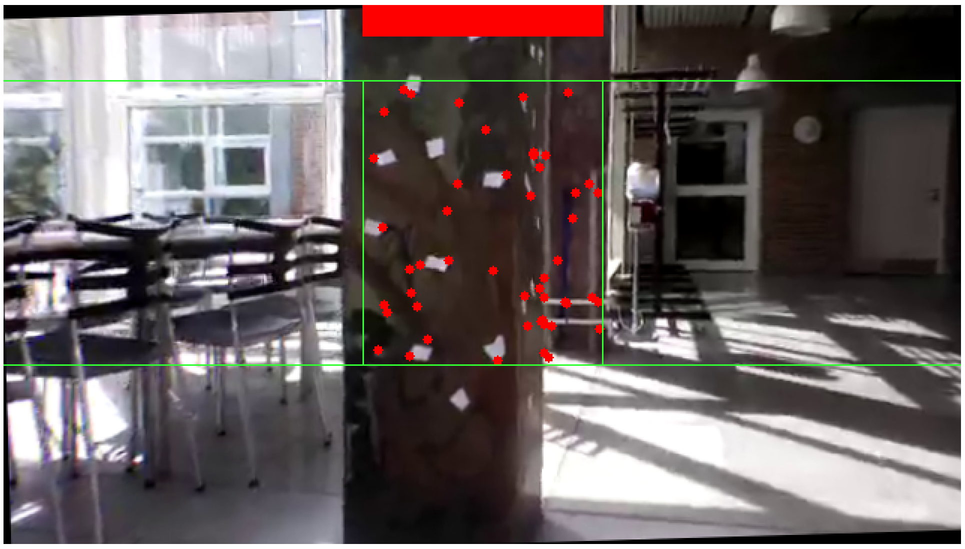

Figure 3.37: The final output from the center region classification.

Combination

Now that a classification method have been developed for both the left, right and center region these can be combined. The bar in the top of the image determines whether or not it is safe to fly in the three directions. In figure 3.38 an example of the final output from the obstacle avoidance system is shown. An obstacle is present to the left and is detected.

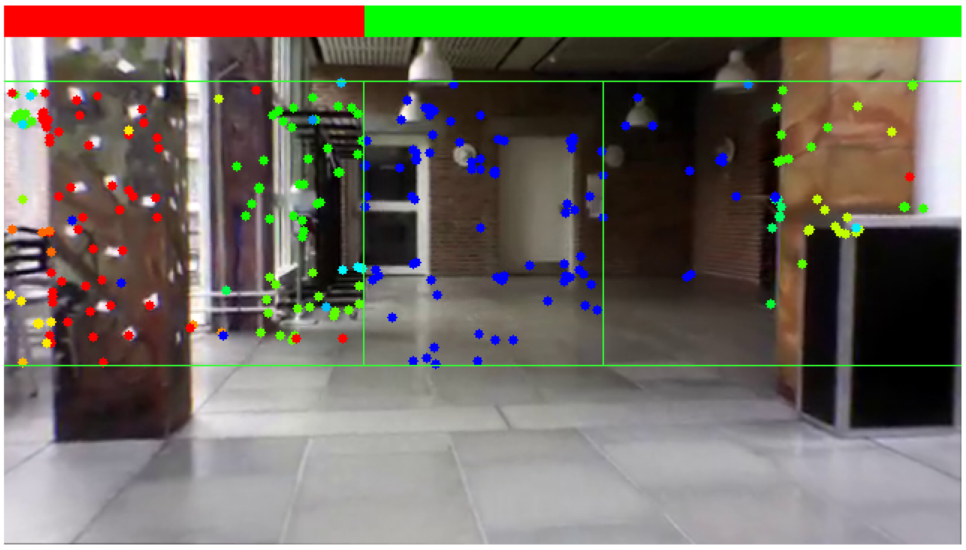

Figure 3.38: The final output from the obstacle avoidance system.

4 System Implementation

In the following chapter the implementation of the systems will be described. First of some general implementation regarding both of the systems are described. Then the details for each of the two systems are described.

4.1 OpenCV

Both the visual odometry system and the obstacle avoidance system is implemented using OpenCV. OpenCV is an open source library containing functions used for computer vision computations [18]. The library is implemented in C++ therefore the implementation of the systems will also be done in C++. As C++ programs are compiled it allows for fast execution of code, which is preferable when working with real time systems. The library contains functionality to compute many of the required operations as described in the system composition chapter of this report. Following the functions used from the OpenCV library will be described.

Undistortion

The OpenCV function *undistort* takes a distorted image and undistorts it. It takes the distortion coefficients as a vector with the 3 radial distortion coefficients and the 2 tangential distortion parameters. Also the camera matrix is needed as the image is shifted so that the optical center matches the image center.

Shi-Tomasi

The OpenCV function *goodFeaturesToTrack* is used to find points to track using the Shi-Tomasi method. The method is supplied with 3 parameters. These are the maximum points to find in the image, the threshold for when a point is considered as a corner and the minimum distance between the points.

Lucas-Kanade

The OpenCV function *calcOpticalFlowPyrLK* is used to estimate the optical flow for the points found using Shi-Tomasi. The function takes a vector of points and returns another vector with the corresponding points. The function parameters are windows size, number of levels to use for the pyramid.

Homography

The OpenCV function *findHomography* is used to find the homography matrix. The function already implements RANSAC, which means that the found matrix has already removed points that are outliers. The function takes two vectors of corresponding points and returns the found homography matrix along with a vector of describing whether or not the points are inliers or outliers to the model.

4.2 Calibration

Both the visual odometry and the obstacle avoidance systems apply undistortion of the images. To undistort the images the distortion parameters are needed. As well as the distortion parameters the camera matrix is needed in the visual odometry system. From the camera matrix the focal length expressed in pixels can be extracted which is used to map from pixel coordinates to world coordinates. The OpenCV library contains functionality to extract these parameters from a set of images. The process of getting these parameters is by holding a known texture in front of the camera in different configurations. The texture used is a chessboard, which is used because the corners in this texture are easy to find by using Harris corners. 15 different images are used for each camera with different translation and rotation. In figure 4.1 4 examples of calibration images are shown.

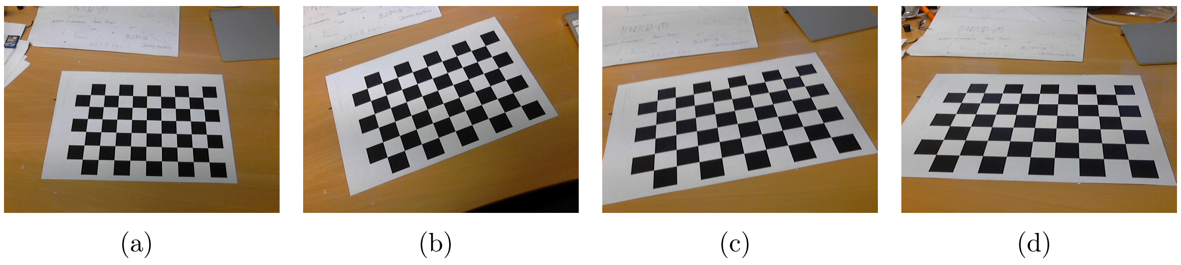

Figure 4.1: Examples of calibration images

4.3 Visual odometry

In this section the implementation specific to the visual odometry system will be described.

4.3.1 Hardware

First the hardware used to compose the visual odometry system will be described. This is the cameras and the distance sensors.

Cameras

The cameras used in this system are Logitech C310 webcams [10]. These cameras are cheap and small which also means that the quality of the images leaves a lot to be desired. The reason these cameras are chosen is simply that these where the cameras available during the project period. In table 4.1 the specifications of the cameras are shown

Specification	*Value*
Max resolution	1280x960
Max FPS	30fps @ 640x480
Field of View	60°
Connection Type	USB 2.0

Table 4.1: Data sheet of the Logitech C310. [10]

The camera cannot both run at full resolution and full frame rate at the same time. For this reason it has been chosen to use the lower resolution of $640x480$ as the higher frame of $30fps$ is prioritized.

Distance sensors

As mentioned earlier the distances from the cameras to the plane they are looking at are needed. In this project it has been chosen to use an ultra sonic range finder. The reason to choose this sensor is simply that this sensor was available during the project period. The sensor used is the Maxbotix MB1320 XL-MaxSonar [13]. In table 4.2 the specifications of the ultra sonic range finder are shown.

Specification	*Value*
Resolution	1cm
Update rate	10Hz
Minumum range	20cm
Maximum range	765cm

Table 4.2: MB1320 specifications. [13]

The sensors can output the distance using 3 different methods, these are: Analog voltage, RS232 Serial and Analog Envelope. For simplicity it has been chosen to use the analog voltage. The voltage is read using a micro controller that is sending the values to a computer using UART. In this project it was chosen to use an Arduino UNO as the micro controller. The Arduino read the analog value of both sensors and maps the value to cm. Afterwards the value is send to the computer using UART. The update rate is at 10Hz as this is the update rate of the sensor. In figure 4.2 the wiring of the sensors and the Arduino is shown.

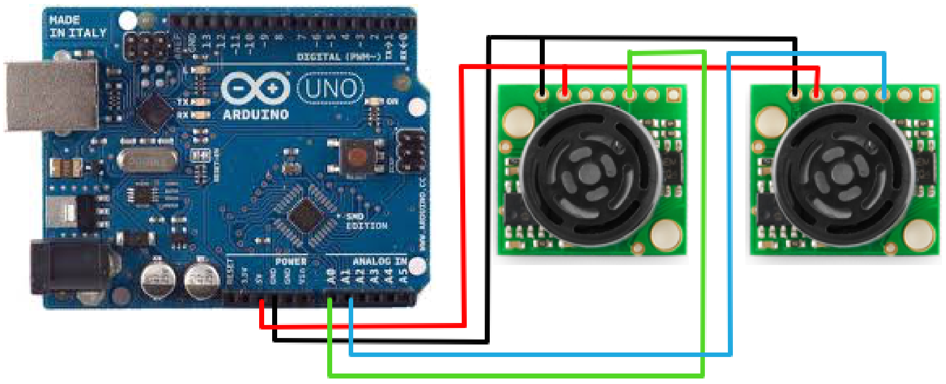

Figure 4.2: The connection of the sensors to the Arduino.

4.3.2 Flow

To summarize the composition and the implementation of the visual odometry system the final system is described in a flow chart. This chart is shown in figure 4.3. The final implementation can be found on the enclosed CD.

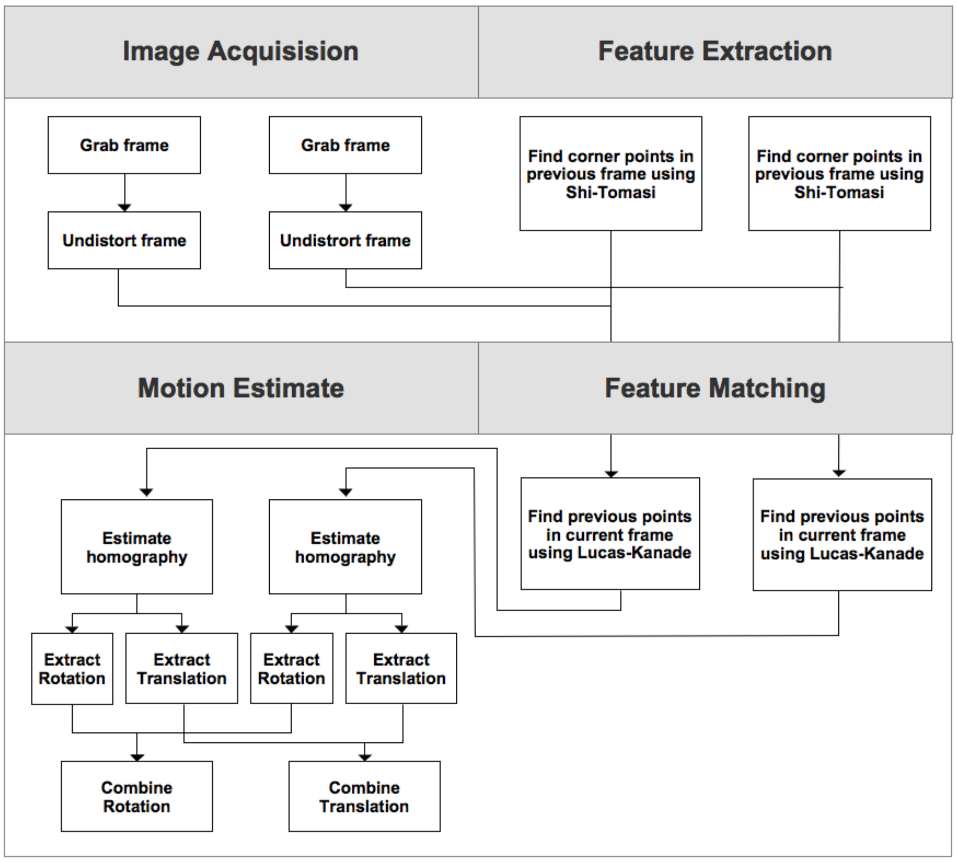

Figure 4.3: The flow of the final visual odometry system

4.4 Obstacle avoidance

In this section the implementation specific to the obstacle avoidance system will be described.

4.4.1 Hardware

First the hardware used to implement the obstacle avoidance system is described. The hardware used is a Parrot Ar.Drone 2 [20]. The Ar.Drone is a small quadcopter equipped with a front facing camera. The quadcopter is controlled via Wi-Fi. In figure 4.4 the quadcopter is seen.

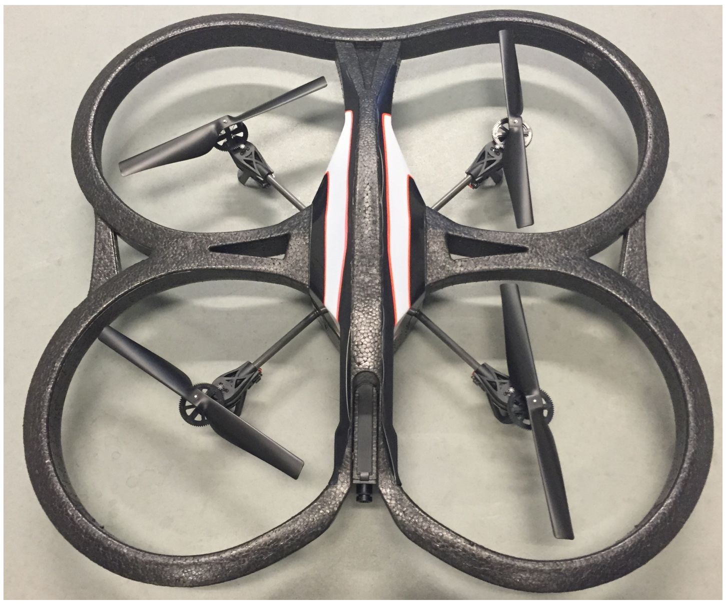

Figure 4.4: The quadcopter used to gather data.

The description of the Ar.Drone is divided into two main sections, camera and IMU.

Camera

The camera used is the one on board the Ar.Drone. The camera is a small camera and the video is compressed using H264 compression. The reason why the feed is compressed is to be able to send it fast through the wireless connection with a low latency. The compression introduces some artifacts and may affect the precision of the detected corners. In table 4.3 the specifications of the camera are listed.

Specification	*Value*
Max resolution	640x360
Max FPS	30fps
Field of View	92°
Connection Type	Wifi

Table 4.3: Data sheet of the Ar.Drone camera. [20]

IMU

On board the Ar.Drone is an IMU. The control algorithms on board the drone use the IMU in order to stabilize it during flight. The roll, pitch and yaw angle from the IMU is continuously sent through the Wi-Fi link to the computer. No details about the precision of these angles are available, however it seems that the precision is below 5°. As well as angles the velocity in all 3 directions is sent through the wireless link. The velocity is calculated as an unspecified fusion between the IMU and a downwards-pointing camera. Again the precisions of these velocities are not specified and are dependent on the amount of visual features on the floor below the drone.

Drone control

The drone is controlled through a Wi-Fi connection. The way the drone is moved is by sending an angle, which the drone will hold. If a negative pitch angle is set the drone will move forward, a positive angle and the drone will move backwards. To move the drone left or right the roll angle is altered. If all angles are set to zero, the drone will go into a hovering mode where the drones tries to hold its position. The software developed to control the drone and receive video is based on the public available HeliSimple software [9]. This software is altered to copy the image information into the OpenCV format.

4.4.2 Flow

To summarize the composition and the implementation of the obstacle avoidance system the final system is described in a flow chart. This chart is shown in figure 4.5. The final implementation can be found on the enclosed CD.

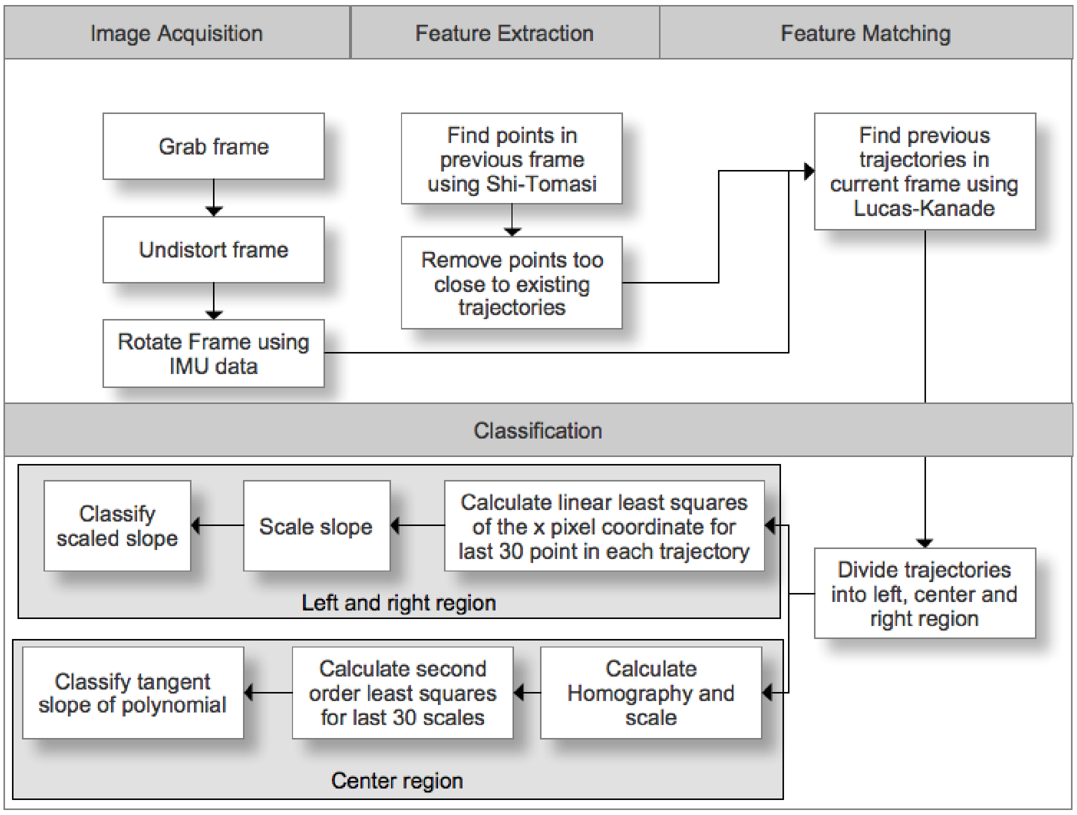
Image Acquisition
Feature Extraction
Feature Matching
Grab frame
Undistort frame
Rotate Frame using IMU data
Find points in previous frame using Shi-Tomasi
Remove points too close to existing trajectories
Find previous trajectories in current frame using Lucas-Kanade
Classification
Classify scaled slope
Scale slope
Calculate linear least squares of the x pixel coordinate for last 30 point in each trajectory
Left and right region
Divide trajectories into left, center and right region
Classify tangent slope of polynomial
Calculate second order least squares for last 30 scales
Calculate Homography and scale
Center region

Figure 4.5: The flow of the final obstacle avoidance system

5 Tests

This chapter will be describing the test setups for both of the proposed systems. The results from the tests will be described in chapter 6.

5.1 Visual odometry

In this section the tests of the visual odometry system will be described. First the test setup including the hardware will be described. Following the test steps will be described. As no requirements to this system can be derived at this point, the tests will be designed to reveal the precision that can be achieved using the proposed method.

5.1.1 The test setup

In this subsection the setup used to conduct the tests are described.

Mounting

To test the system the cameras and distance sensors needs to be mounted as described earlier. It has been chosen to mount the cameras and the distance sensors on a piece of hard plastic, which can be moved around simulating a UAV flying. The most realistic results would have been gathered by mounting the system on an actual UAV, however the time allotted to this project means that there is not enough time to get such a system up and running. In figure 5.1 the mounting of the sensors on the plastic plate is shown.

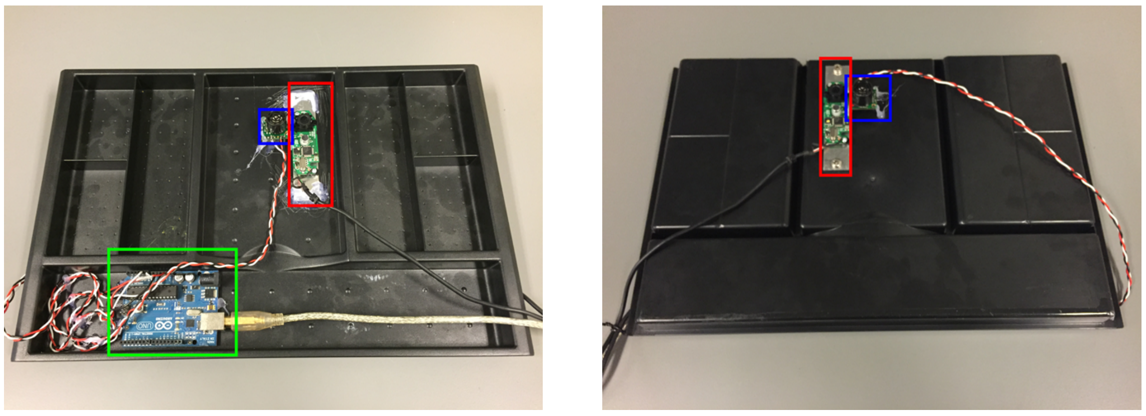
(a) (b)

Figure 5.1: The mounting of the camera and distance sensors. **Red boxes:** The cameras. **Blue boxes:** The distance sensors. **Green box:** The Arduino used to process the signal from the distance sensors.

To make sure that the cameras are mounted exactly on top of each other, the cameras are mounted on a piece of aluminum. These two pieces are identical and with holes drilled out at the same place. When the cameras were mounted on the plate, a screw was screwed through these holes and through the plastic plate. This results in the cameras being mounted exactly on top of each other. The distance sensors are simply glued to the plate next to the image sensor. In figure 5.2 the alignment of the cameras are illustrated.

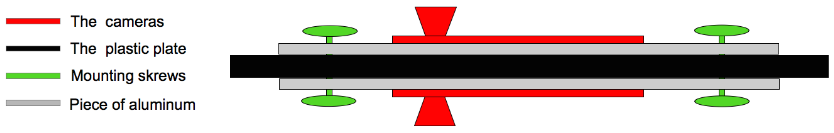

Figure 5.2: The alignment of the cameras.

Ground truth data

To be able to compare the results gathered from the system, some ground truth data is needed. The ground truth data is needed to compare the output from the system with the actual true data. The university has a laboratory named AVA Lab (Audio Visual Arena Laboratory) available where it is possible to track an object in 3D. This room is used to capture the position of the plate as well as the orientation of the plate. In figure 5.3 the room is shown.

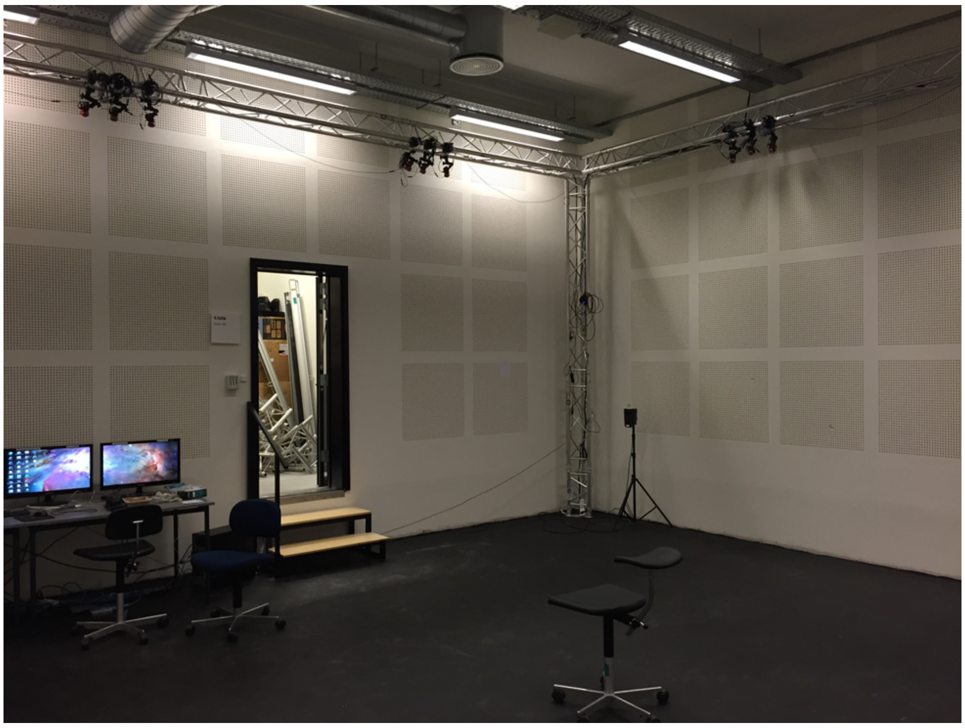

Figure 5.3: The room used to capture the position of the plate.

The system works by having 18 infrared cameras mounted on an aluminum grid along the ceiling of the room. These cameras emit light in the infrared spectrum that is not visible to the human eye. An object can be tracked by placing markers, coated in a special reflective material, on the object. These markers are easily found by the cameras. A computer is gathering the images from the cameras and by knowing the cameras precise position in the room; triangulation can be used to calculate the position of the markers in the room. The processing of the images and the triangulation process is handled by the software application Motive, which is supplied by the manufacturer of the system OptiTrack [19].

The room is approximately 7.5x7.5x7.5 meters and the area where the object can be tracked is approximately 3x2 meters. This area is a bit small to test the system drift over a longer distance, but it will be enough to test the precision over a smaller period of time. The precision of the system is reported to be ±1mm by the Motive application.

Mounting of the markers

As mentioned reflective markers are needed on the object in order to be able to track it. The markers are mounted on a plastic structure that is then glued to the plastic plate. In order to get the position of the camera centers instead of the markers, the position is offset using the Motive application. This means that the position returned is actually the position of the camera sensor instead of the markers, this is shown in figure 5.5. In figure 5.4 the reflective markers mounted on the plastic plate is shown.

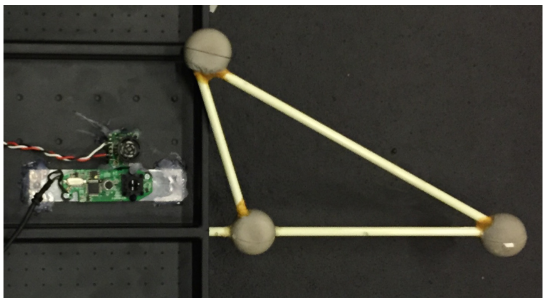
Figure 5.4: The reflective markers mounted on the plastic plate.

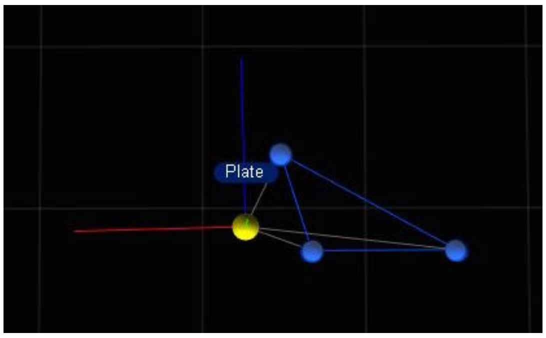

Figure 5.5: The offset of the markers to the camera. **Blue sphere:** The detected reflective markers. **Yellow sphere:** The center position of the cameras.

Texture

The initial test showed that a lack of texture on the plane that is tracked leads to errors. The floor in the room is painted black which means that not corners would be found, to cope with this problem small pieces of white tape is taped to the floor in a random pattern. The white tape on the black floor should result in some nice sharp corners for the algorithm to find. The ceiling of the room is white and features are only present in the areas where ventilation pipes are running across the room. To add some extra features the ceiling two pieces of black cloth measuring 7.4x1.4 meters is stretched across the room. On these two pieces of cloth small pieces of white tape is applied in a random pattern. As with the floor this should result in nice sharp corners. Figure 5.6 shows the added texture in the room.

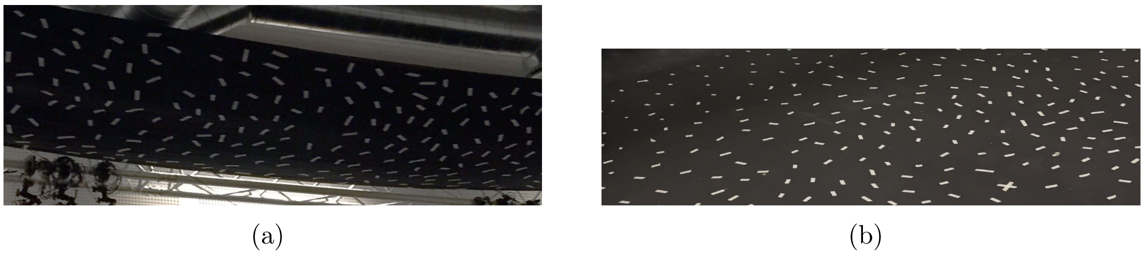
(a) (b)

Figure 5.6: The texture added to the laboratory. **a:** The ceiling. **b:** The floor.

Data gathering

To be able to post process and tweak parameters in the system it has been chosen to not conduct the tests in real time. Instead data sets containing the necessary information are gathered. The data sets contains the following: Top image, bottom image, a text file with the heights from the two sensors and a text file containing the ground truth positions and rotations. The ground truth data is sent from the processing computer to a laptop using Ethernet. This laptop is gathering the images from the two cameras, and the distance from the sensors. Due to some technical issues the sample rate is only 15 fps instead of 30 fps.

5.1.2 Test scenarios

Different test scenarios are setup which aims to determine the precision of the system. The reason to have different tests is to determine how different movements affect the system. As the purpose of the project is not to validate that the system works but instead to determine how well the proposed method works, the outcome of the tests will be results described as precision.

Following the different test scenarios will be described and the purpose of the scenarios.

Hoover test

This test is conducted to determine how the system performs when no movement is present. This would be the equivalent of the UAV hovering at a fixed position. Following the test steps are described.

1. A person holds the platform as far away from the person as possible.
2. The recording of data is started.
3. The platform is held as steady as possible.
4. The recording is stopped after 60 seconds.

Translation test

This test is conducted to show how the system performs when the system is moving without rotations. In this test the platform is only moving sideways, forwards and backwards. The reason to not include rotations in this data set is to eliminate errors occurring due to rotations.

1. A person holds the platform as far away from the person as possible.
2. The recording of data is started.
3. The person starts moving the platform following the specified path.
4. When the person reaches the end of the path, the recording is stopped.

The path the platform should follow is shown in figure 5.7. As it is a person moving the platform it is impossible to eliminate rotations. However a UAV would not be able to eliminate rotations either, which makes the test quite realistic.

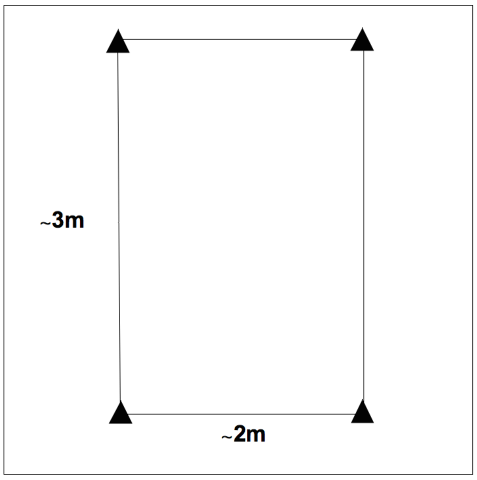

Figure 5.7: The path the test platform is following.

Translation + rotation test

This test is conducted to show how the system performs when the system is moving. In this test the platform is moving sideways, forwards, backwards and rotating around the optical axis. This test is conducted to simulate actual UAV flight.

1. A person holds the platform as far away from the person as possible.
2. The recording of data is started.
3. The person starts moving the platform following the specified path.
4. At the specified points the platform is rotated.
5. When the person reaches the end of the path, the recording is stopped.

The path the platform should follow is shown in figure 5.8. As it is a person moving the platform it is impossible to eliminate rotations at other points than the designated. However a UAV would not be able to eliminate rotations either, which makes the test quite realistic.

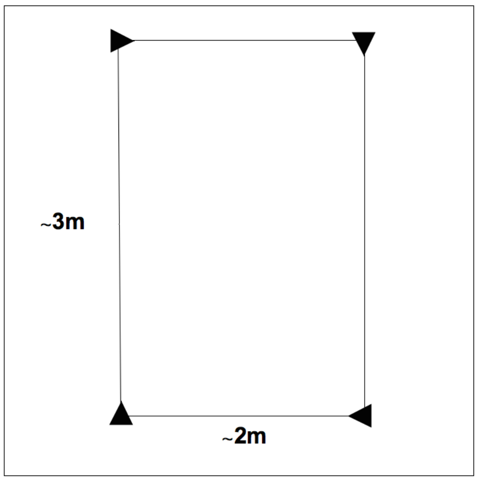

Figure 5.8: The path the test platform is following.

Rotation

This final test is conducted to examine the effect of only rotating the platform, leaving out the translation. This would be the equivalent of the UAV hovering at a fixed position while yawing.

1. A person holds the platform as far away from the person as possible.
2. The recording of data is started.
3. The platform is continuously rotated to -90° up to 90°.
4. The recording is stopped after 60 seconds.

On the enclosed CD the data set from the translation test can be found.

5.2 Obstacle avoidance

In this section the tests of the obstacle avoidance system will be described.

Test execution

The purpose of the obstacle avoidance system is to be able to detect obstacles in order to be able to avoid them. As described in the problem statement, the minimum distance to an object before it must be detected varies depending on the platform. To validate the proposed method, the system will be tested using the Ar.Drone. From testing it has been found that the Ar.Drone needs less than 0.7 meter to stop independent on the flying speed. This value was found by flying at speeds from 0.6m/s to 2m/s. The distance needed to stop was determined by examining the data from the IMU.

Three different tests are executed, two of the tests are used to test how the system performs with an obstacle present, and one to determine how it performs with no obstacle present. The reason to have different tests with an obstacle is because of the two different classification methods. These classification methods are evaluated individually as both methods works independently of each other.

Left and right region

The purpose of the left and right region test is to verify that obstacles to either sides of the drone can be detected. The test is conducted using the following steps:

1. The drone is placed at least 6 meters from the obstacle and approximately 0.5 meters to the left or the right of the obstacle.
2. The drone takes off.
3. The drone is rotated so that it faces the obstacle.
4. The recording is started.
5. The drone flies towards the obstacle at $\approx$1m/s and passes by.
6. The drone is stopped.
7. The recording is stopped.

Center region

The purpose of the center region test is to verify that obstacles straight in front of the drone can be detected. Straight in front of the drone is defined as within the horizontal field of view of 24.6°. This corresponds to the 160px wide center region as described in the system composition subsection 3.2.4. The test is conducted using the following steps:

1. The drone is placed at least 6 meters from the obstacle.
2. The drone takes off.
3. The drone is rotated so that it faces the obstacle.

4. The recording is started.
5. The drone flies towards the obstacle at ≈1m/s.
6. The drone is stopped just before hitting the obstacle.
7. The recording is stopped.

No obstacles

The purpose of the test with no obstacle present is to verify that the system does not generate false positives. The test is conducted using the following steps:

1. The drone is placed at the beginning of the specified path.
2. The drone is rotated so that is faces the end of the specified path.
3. The recording is started.
4. The drone flies towards the end of the specified path at ≈1m/s.
5. The drone is stopped when reaching the end of the path.
6. The recording is stopped.

5.2.1 The test setup

The test is divided into three scenarios. First the two tests with obstacles are carried out in a controlled environment. The purpose of this test is to see how the system performs when a single obstacle is present in a otherwise uncluttered scenario. In this scenario the obstacles is generated with as well defined texture as possible. In the second scenario the same two tests with an obstacle is carried out. The second scenario is used to test the system in a cluttered environment similar to the one where the initial test data is gathered. This scenario is deemed more similar to a real world scenario. In this scenario the texture on the obstacle is less prominent. The last scenario is in a large open space with no obstacles, in this scenario the test with no obstacles is carried out. All three scenarios are described in detail in subsection 5.2.2.

Texture

In the two scenarios with an obstacle present, texture is added to make it easier for the algorithm to track the obstacle. In the uncluttered environment the obstacle is generated from a piece of black cloth. On this cloth a lot of small pieces of white tape are taped. The white tape on the black cloth will generate corners for the feature extraction step to find. In the cluttered environment the obstacle is a pillar. Larger pieces of white paper are taped to the obstacles, which will generate more corners for the feature detection step to detect. In the uncluttered scenario more pieces are added and with a smaller distance compared to the cluttered scenario. This will help to determine if the amount of texture have an impact on the ability to detect obstacles. In figure 5.9 and 5.10 the texture from the two scenarios are shown.

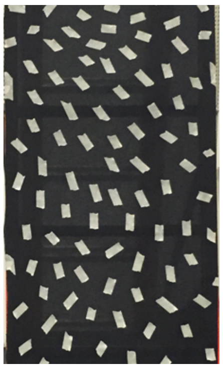
Figure 5.9: Uncluttered

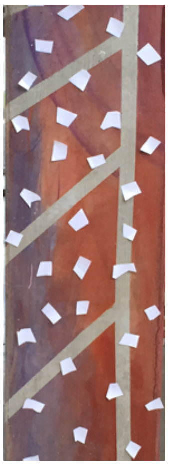
Figure 5.10: Cluttered

Data gathering

To be able to post process the data it has been chosen not to conduct the tests in real time. Instead different data sets are collected in order to verify the performance of the system. Each data set contains the necessary information from the drone to simulate a real time flight. Each data set contains all the images for the flight as well as IMU data. For each image received the image is written to the hard drive. Along with the images the IMU data roll, pitch and yaw angles, as well as velocity information is written to a text file. The file contains a single line with the information for each image received. Data is sampled at ≈25Hz.

5.2.2 Test scenarios

As mentioned earlier, test data is gathered from three different scenarios. These will be described in the following subsection.

Uncluttered scenario

The first scenario is with an artificial obstacle created with features as optimal as possible for the feature detection step to detect. The background is chosen to contain as little features as possible. The tests are carried out in the cantina area at Aalborg University Rendsburggade 14. 10 data sets are collected with the obstacle in the center, 5 data sets are collected with the obstacle to the left and 5 data sets are collected with obstacle to the right. In total 10 sets are collected to tests the center region classification and 10 data sets to test the left and right region classification.

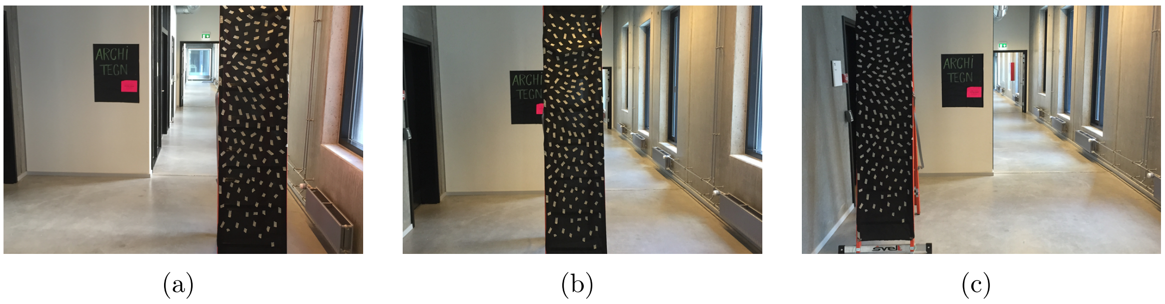

Figure 5.11: The three different test setups in a uncluttered environment.

In figure 5.11 the three different setups are shown.

Cluttered scenario

The purpose of gathering data sets from a cluttered environment is to see how the system performs in a more realistic scenario with a more realistic obstacle. The cluttered area is the same as where the initial test data is gathered. This is in the cantina area of Aalborg University Frederik Bajers Vej 7. Data are collected from 4 different pillars not including the one used for gathering the initial data sets. The reason to gather data from different pillars is to test the system with varying backgrounds.

For each pillar 4 data sets are captured where the drone approaches the pillar straight on. The purpose of these tests is to test the classification of the center region of the image. For two of the pillars 4 data sets are collected where the drone is flying past the pillar to the right. For the other two pillars 4 data sets are collected where the drone flies past the pillar to the left. This means that in total 16 data sets are collected to test the center region of the image, and 16 data sets are collected to test the left and right region of the image.

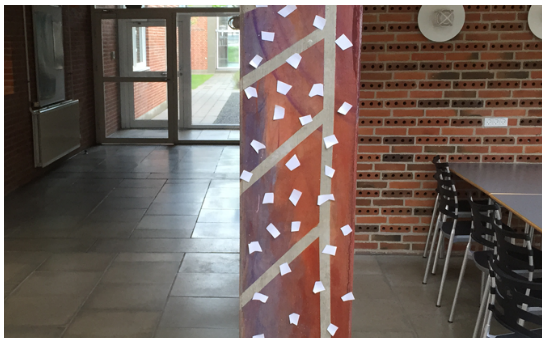

Figure 5.12: Pillar 1

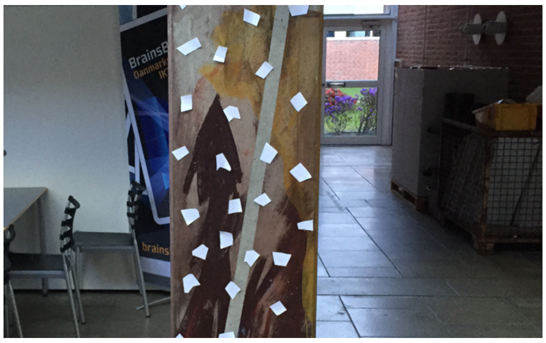

Figure 5.14: Pillar 2

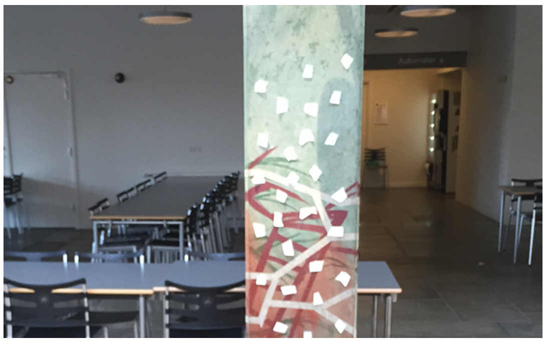

Figure 5.13: Pillar 3

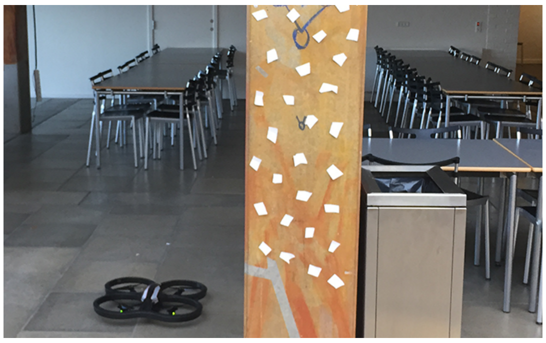

Figure 5.15: Pillar 4

Figure 5.12, 5.14, 5.13 and 5.15 shows the 4 different pillars where data sets are collected.

Open space scenario

The purpose of gathering data sets from the open space scenario is to verify that the system does not generate false positives when flying in an area where no obstacles are present. The test is carried out in the hall area of Novi at Niels Jernes Vej 14, Aalborg University. The "No obstacles" test is carried out with 3 different paths. The reason to have multiple paths is to vary the background in where features are tracked. For each of the three different paths 10 data sets are collected. Figure 5.16 shows the 3 different paths in the open area.

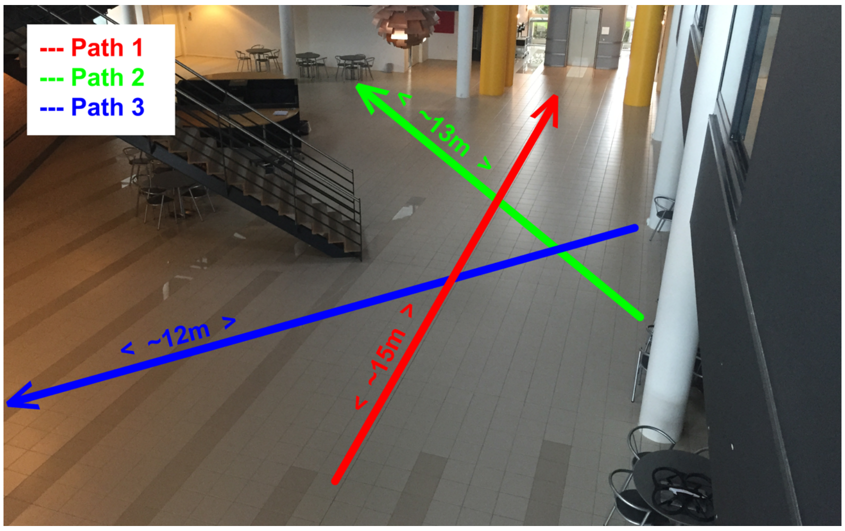

Figure 5.16: The open scenario with the three paths shown.

Summary

In table 5.1 a summary of the different test is shown. A total of 82 different data sets are collected from the three different scenarios.

	Uncluttered Scenario	**Cluttered Scenario**	**Open Scenario**	**Total**
Left Region	5	8	0	13
Right Region	5	8	0	13
Center Region	10	16	0	26
No obstacle	0	0	30	30

Table 5.1: Summary of the different test carried out in the different scenarios.

On the enclosed CD an example of each of the tests from each of the scenarios can be found.

6 Results

In this chapter the results from the tests described in chapter 5 will be presented. The chapter is divided into two sections, one for visual odometry and one for obstacle avoidance.

6.1 Visual odometry

In this section the results from the tests of the visual odometry system, described in chapter 5, will be presented. The test will be divided into three subsections; the first subsection will be focusing on the precision of the system per sample. The other subsection will be focusing on the results from concatenating the results into a trajectory. Finally the third subsection will be discussing the reasons believed to cause errors.

6.1.1 Per sample results

In this subsection focus will be on the per sample precision of the system. For each of the tests two plots are displayed. The first plot shows the estimated motion along with the ground truth for the x-axis, the y-axis and for the rotation. The second plot shows the absolute error per sample and the mean absolute error. The calculation of the absolute error and the mean absolute error is shown in equation 6.1.

$$\begin{gathered} E_i = |X_i - G_i| \\ \overline{E} = \frac{\sum_{i=1}^{n} |X_i - G_i|}{n} \\ E = \text{Error}, \overline{E} = \text{Mean error}, X = \text{Estimate}, G = \text{Ground truth} \end{gathered} \tag{6.1}$$

The reason to look at the absolute error and the mean absolute error is because it tells something about what the predicted error would be of the system and which constrains it would put on the UAV. All the results are shown without the smoothing filter, the reason to not filter the results is to show the raw per sample data. The filtering of the data would typically be implemented as a step of the control algorithms and fused with information from other sensors.

Following the results from the four tests will be presented.

Hoover test

In figure 6.1 the results from the hoover test are shown. The left column shows the estimate(Red line) along with the ground truth(Blue line). The right column shows the absolute error(Red line) along with the mean absolute error(Blue line). The rows correspond to the x-axis, the y-axis and rotation respectively.

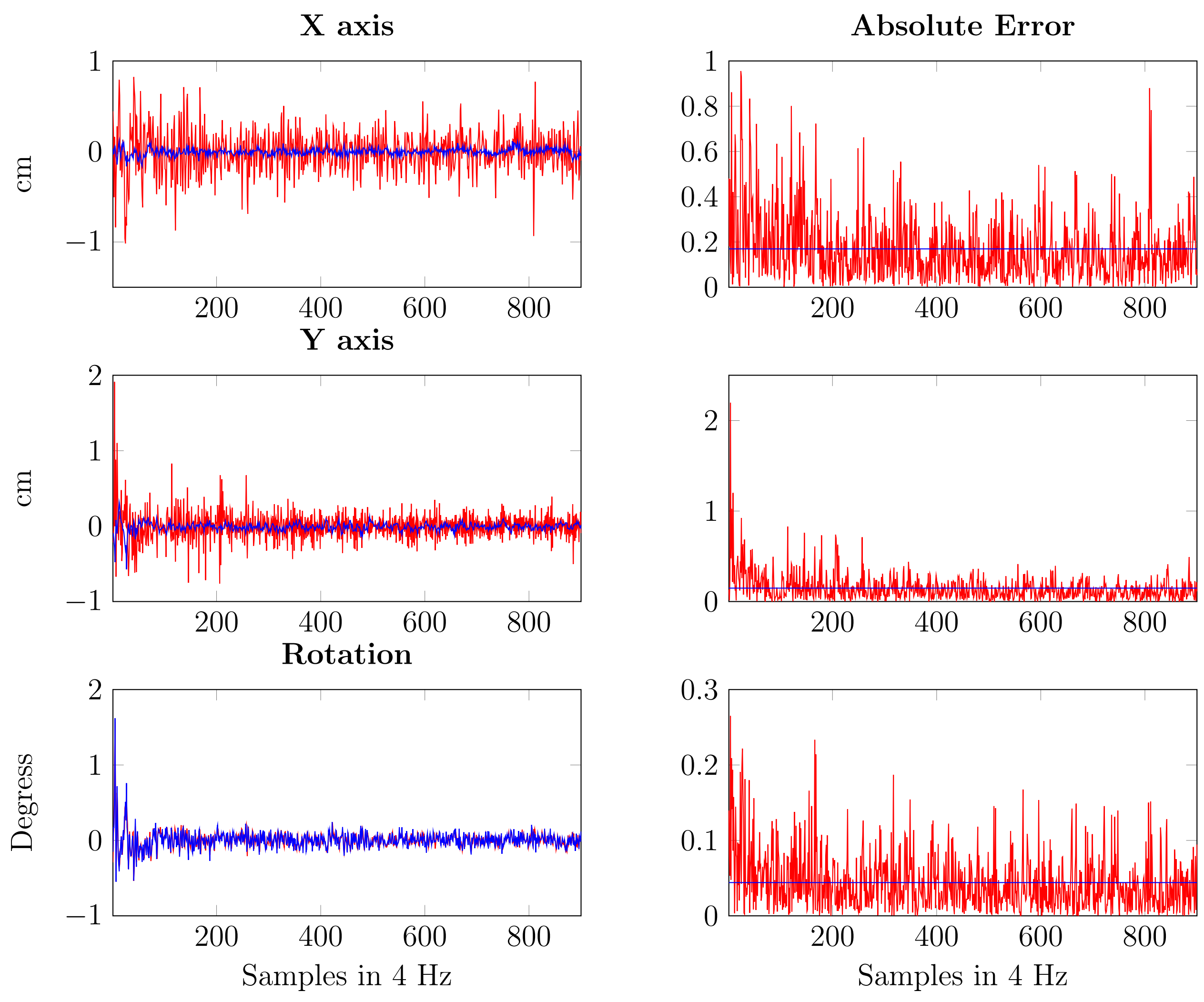

Figure 6.1: Hoover test results.

This plot shows that there is some error present in the system. An interesting tendency to notice is that the error is more or less scattered equally on both sides of the ground truth.
In table 6.1 the minimum, maximum and mean absolute error is show for each of the three estimates.

	Min	**Max**	**Mean**
X Axis	0.0010cm	0.9557 cm	0.1696cm
Y Axis	0.0003cm	2.1974cm	0.1482cm
Rotation Axis	0.0000°	0.2650°	0.0441°

Table 6.1: Error values for hoover test.

Translation test

In figure 6.2 the results from the translation test is shown. The left column shows the estimate(Red line) along with the ground truth(Blue line). The right column shows the absolute error(Red line) along with the mean absolute error(Blue line). The rows correspond to the x-axis, the y-axis and rotation respectively.

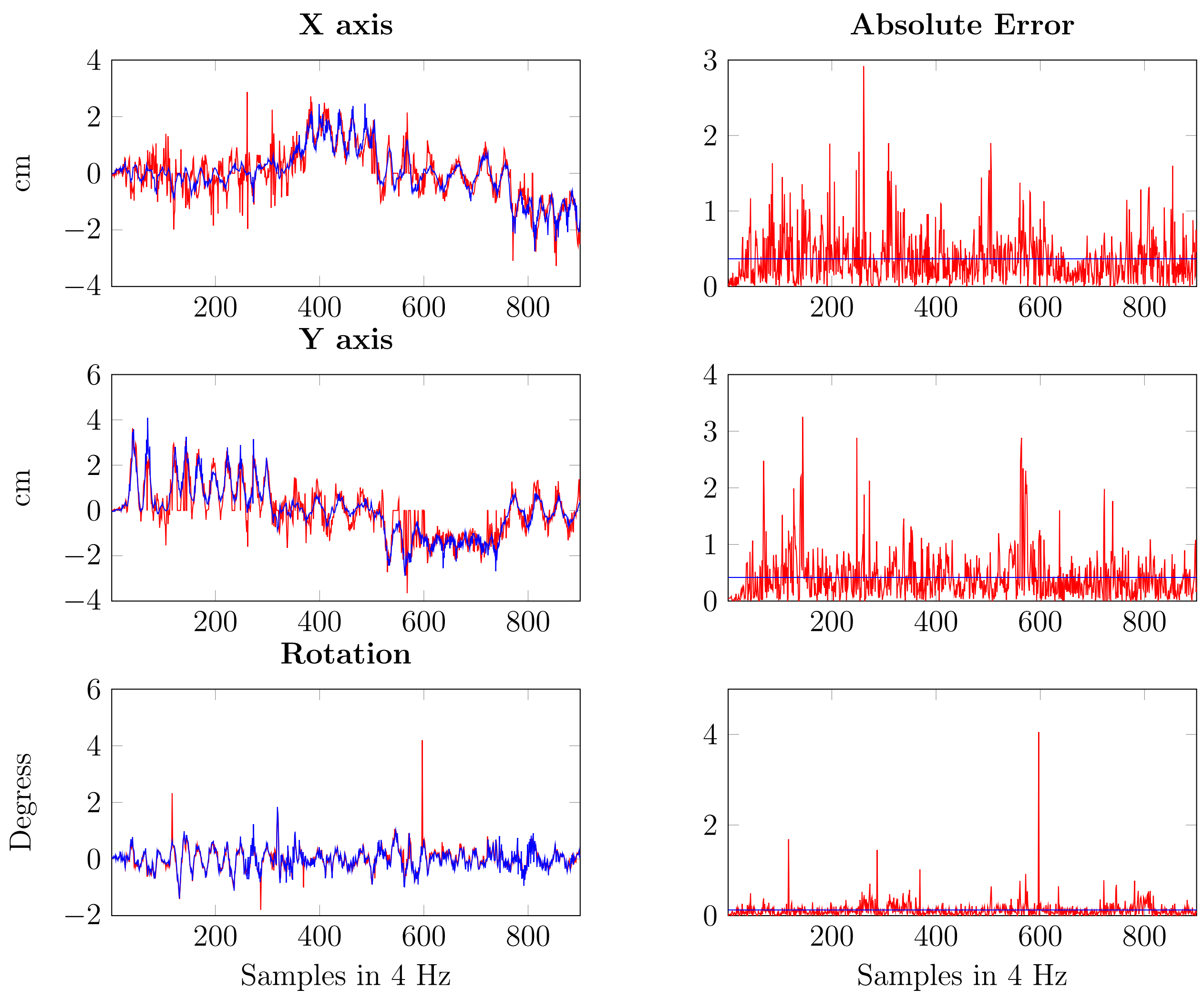

Figure 6.2: Translation test results.

This plot clearly shows that there is some error present in the system. By looking at the plots it seems that the errors are larger in periods where there is large movement in the opposite direction. An example of this can be seen from sample 50 to 150 where the error seems large in the x direction while there is movement in the y direction. Another place where errors are present is around sample 580 where the rotation is just above the threshold that set the motion to zero. As seen in the previous test there is a tendency towards that the error is more or less scattered equally on both sides of the ground truth. In table 6.2 the minimum, maximum and mean absolute error is show for each of the three estimates.

	Min	**Max**	**Mean**
X Axis	0.0003cm	2.9189 cm	0.3671cm
Y Axis	0.0000cm	3.2529cm	0.4170cm
Rotation Axis	0.0002°	4.0516°	0.1247°

Table 6.2: Error values for translation test.

Translation+Rotation test

In figure 6.3 the results from the translation+rotation test is shown. The left column shows the estimate(Red line) along with the ground truth(Blue line). The right column shows the absolute error(Red line) along with the mean absolute error(Blue line). The rows correspond to the x axis, the y axis and rotation respectively.

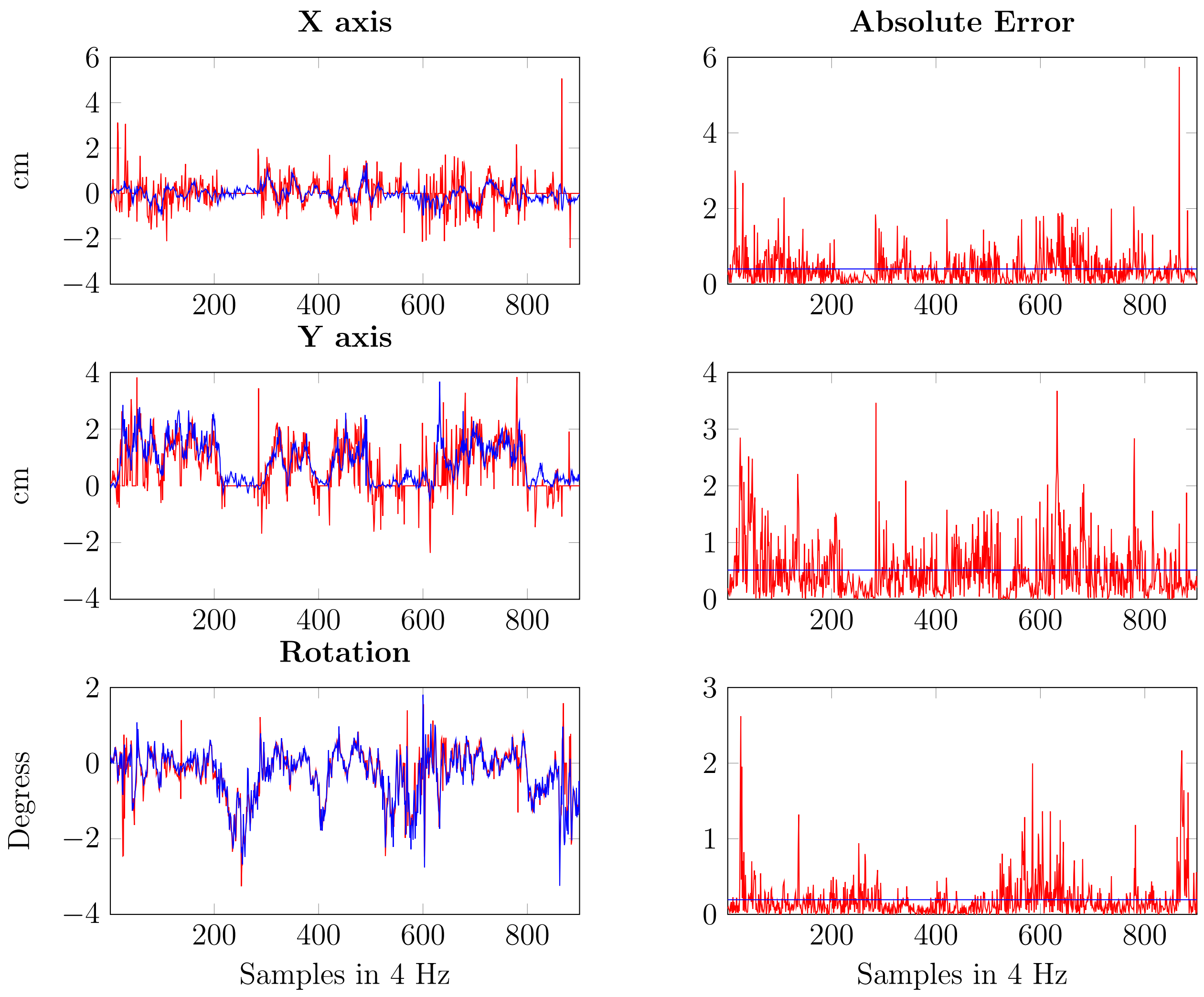

Figure 6.3: Translation+rotation test results.

This plot shows that there is some error present in the system. By examining the results it seems that there is a tendency towards larger errors in the translation estimate when

the platform is rotating. An example of this can be seen from sample 580 to 650 where the error estimate in the x-axis is much larger probably due to the rotation. As seen in the previous test there is a tendency towards that the error is more or less scattered equally on both sides of the ground truth except for the segments where the platform is rotating.
In table 6.3 the minimum, maximum and mean absolute error is show for each of the three estimates.

	Min	**Max**	**Mean**
X Axis	0.0001cm	5.7430 cm	0.4023cm
Y Axis	0.0012cm	3.6747cm	0.5139cm
Rotation Axis	0.0002°	2.6207°	0.1940°

Table 6.3: Error values for translation+rotation test.

Rotation

In figure 6.4 the results from the rotation test is shown. The left column shows the estimate(Red line) along with the ground truth(Blue line). The right column shows the absolute error(Red line) along with the mean absolute error(Blue line). The rows correspond to the x-axis, the y-axis and rotation respectively.

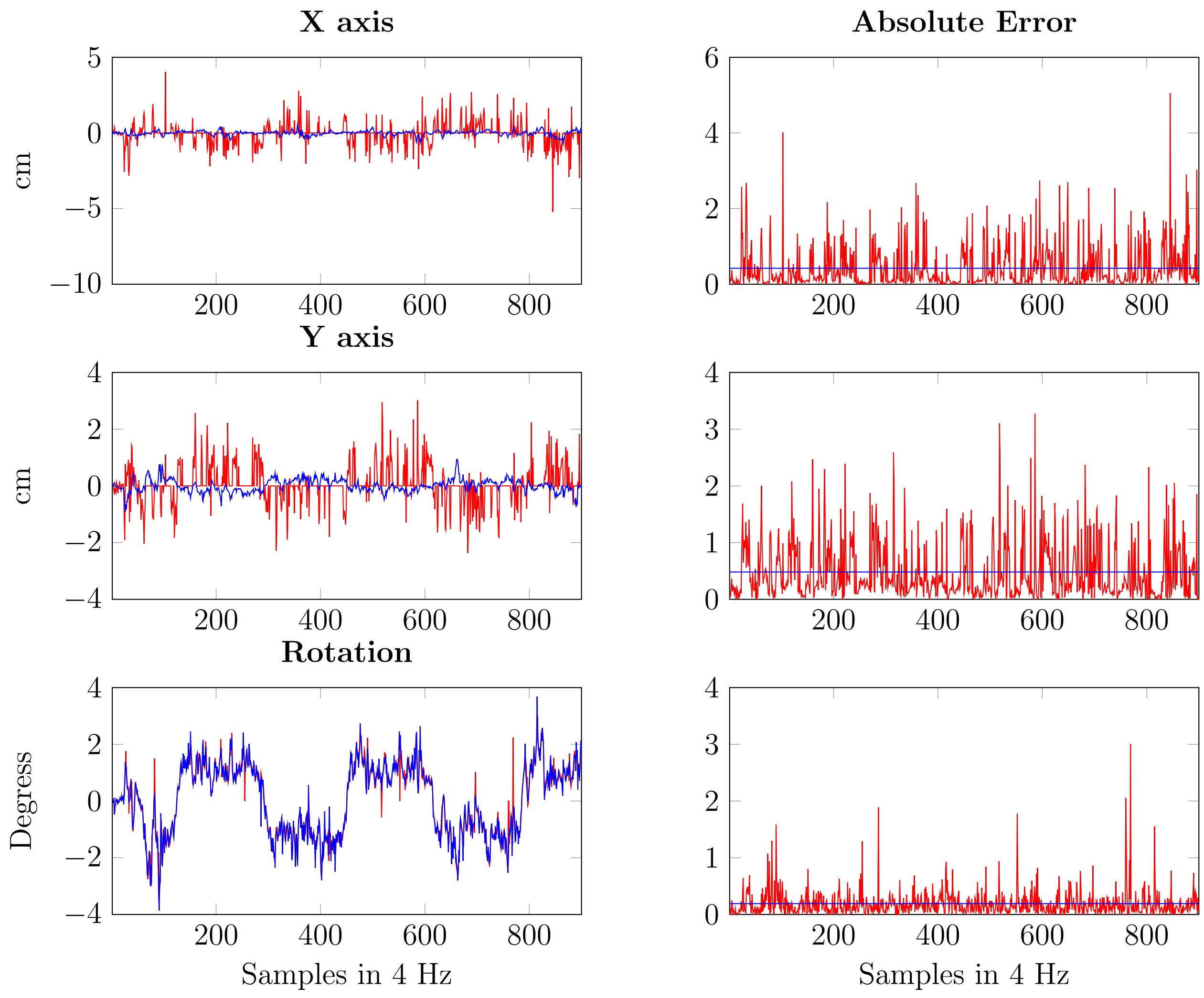

Figure 6.4: Rotation test results.

This plot shows that there is some error present in the system. The results from this test match quite well the ones from the translation+rotation test. It is clearly seen that there are higher errors present in the translation estimate. This matches well with the assumption that the errors are caused by the platform rotating.
In table 6.4 the minimum, maximum and mean absolute error is show for each of the three estimates.

	Min	**Max**	**Mean**
X Axis	0.0005cm	5.0555 cm	0.4221cm
Y Axis	0.0019cm	3.2692cm	0.4817cm
Rotation Axis	0.0007°	3.0082°	0.1913°

Table 6.4: Error values for rotation test.

6.1.2 Trajectory results

This subsection will be focusing on the results of calculating a trajectory from the estimates. Only two of the test scenarios are used here, these are the translation test and the translation+rotation test. The reason to only use these two scenarios is because there is no movement in the other tests. For each of the tests, a plot is generated for each of the three estimates. Also a scatter plot in 2D showing the trajectory is plotted. The trajectory is simply found summing the results and rotating using the summed rotation. The calculation of the trajectory is shown in equation 6.2

$$\Delta_i = \Delta_{i-1} + \begin{bmatrix} cos(\sum_1^i \theta) & -sin(\sum_1^i \theta) \\ sin(\sum_1^i \theta) & cos(\sum_1^i \theta) \end{bmatrix} T_i \tag{6.2}$$

$$\Delta = \text{The trajectory}, \theta = \text{The estimated rotation}, T = \text{The estimated translation}$$

Translation

In figure 6.5 the calculated trajectory for the translation test is shown. The left column shows the individual estimates along with the ground truth. The right column shows the 2D plot of the trajectory. The red lines are the estimate and the blue lines are the ground truth.

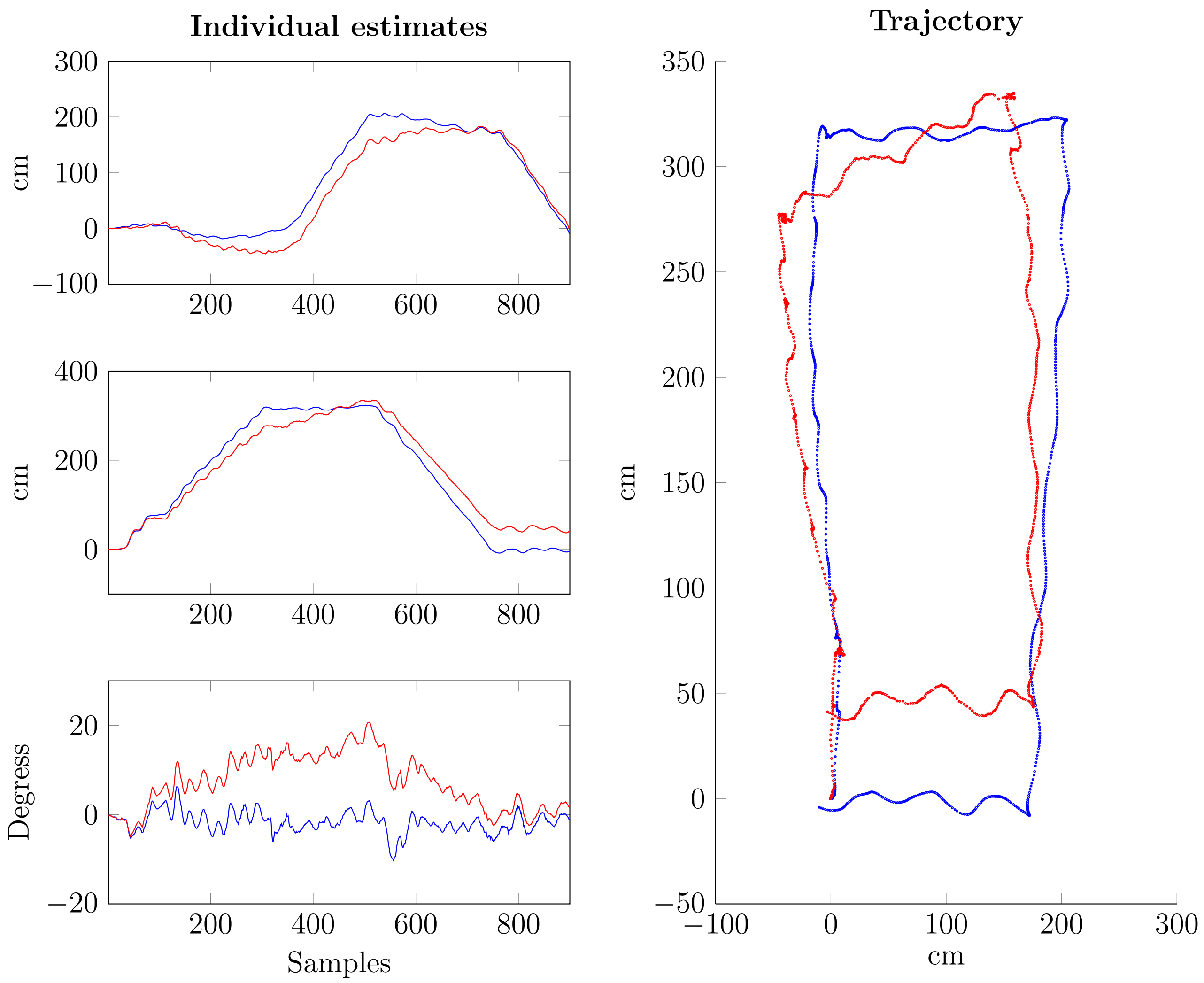

Figure 6.5: Translation test trajectory result. **Top:** X axis. **Middle:** Y axis. **Bottom:** Rotation.

By examining the plots it seems that the translation estimate is fairly accurate. However it is clearly seen that the rotation drifts quite a lot. An interesting thing to notice is that the rotation drifts in a positive direction when the platform moved forward and in a negative direction when the platform moves backwards. This is seen as the rotation trajectory approaches the ground truth near the last samples.

Translation+Rotation

In figure 6.6 the calculated trajectory for the translation+rotation test is shown. The left column shows the individual estimates along with the ground truth. The right column shows the 2D plot of the trajectory. The red lines are the estimate and the blue lines are the ground truth.

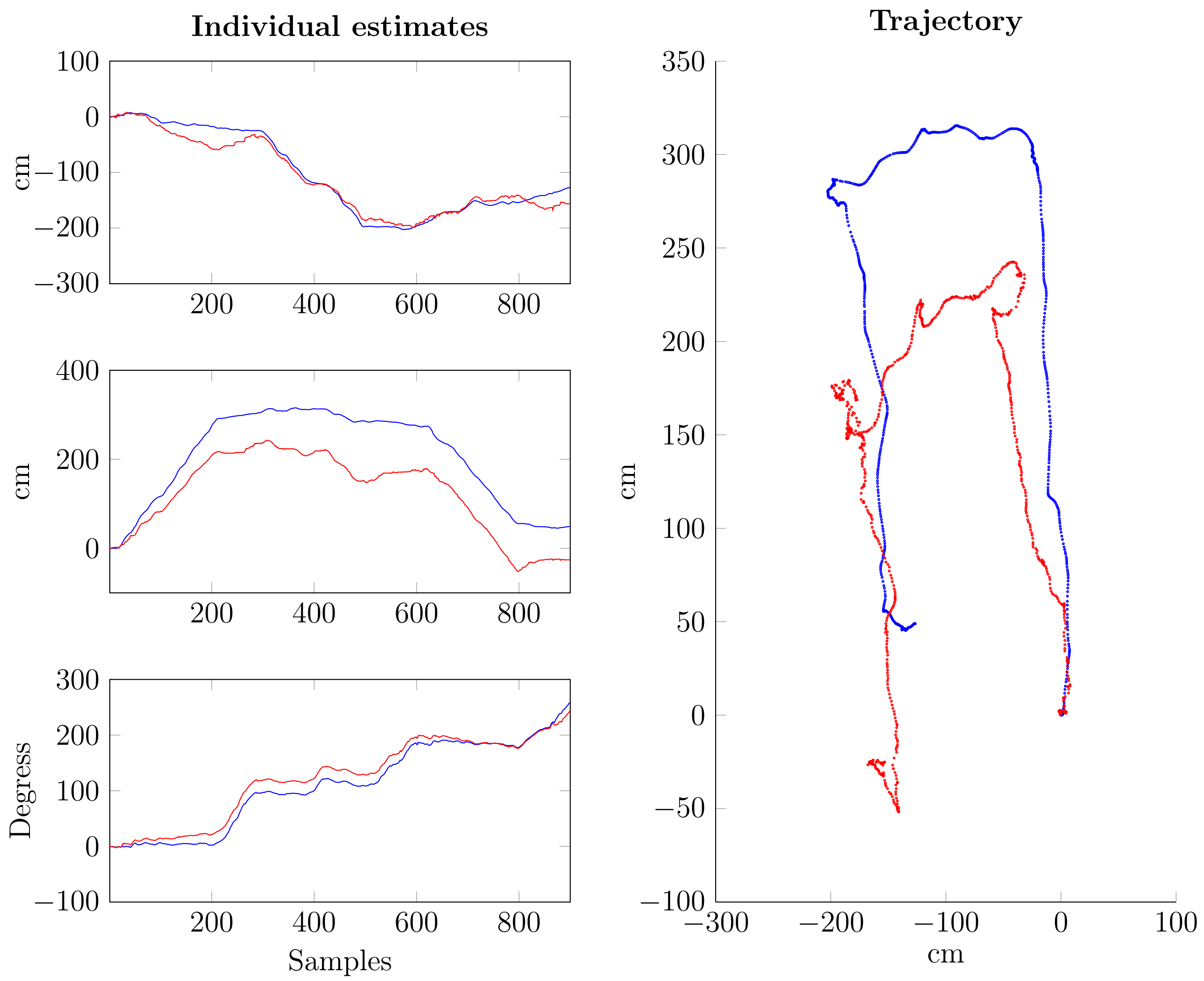

Figure 6.6: Translation+rotation test trajectory result. **Top:** X axis. **Middle:** Y axis. **Bottom:** Rotation.

By examining the plot it seems that the translation estimate is fairly accurate except in the areas where the platform rotation. As with the previous test it is clearly seen that the rotation drifts.

6.1.3 Error causes

In this subsection the causes believed to generate the errors will be described.

Rotation filter

As described in the system composition subsection 3.1.5, a filter is applied to the output estimates. This filter sets the motion equal to zero when the rotation is above a certain threshold. When examining the motion output is clearly seen when this filter kicks in. It is believed that the results are better when this filter is applied but it could be more intelligent. Sometimes the filter removed too much and other times too little. This is especially visible in the translation+rotation test trajectory. In the x-axis trajectory too little motion is removed and in the y direction too much is removed. The reason that this filter is needed at all is probably due to some of the reasons described following. If these problems where fixed it is believed that this filter is not needed at all.

Non synchronized cameras and low frame rate

The problem of non-synchronized cameras is believed to cause two problems and the effects of these problems are increased due to the low frame rate of 15fps. The two problems is the noise in the estimate, the second is actual errors in the estimate.

Equal noise

When looking at the plots that are comparing the estimate and the ground truth for all the tests, it is clearly seen that estimate fluctuates on both sides of the ground truth. This is probably because the two cameras are not in sync. This means that the world can have moved more in one of the cameras than the other. This missing motion is then caught up in the other camera in the next frame, which results in these fluctuations. If the system is used to calculate a trajectory this problem is not a big problem as it is leveled out when summing up the estimates. However is the motion per sample is used then this could cause some problems.

Errors in estimate

Another problem that can arise when having non-synchronized cameras is that potentially an extreme point can be caught. When the platform is moving it is impossible to eliminate shaking of the platform. If one of the cameras grabs a frame at an extreme point and the other one does not then the motion estimate will be wrong. The problem is illustrated in figure 6.7. The blue line is the motion in one direction. The red and green lines are the sampling points from the two cameras.

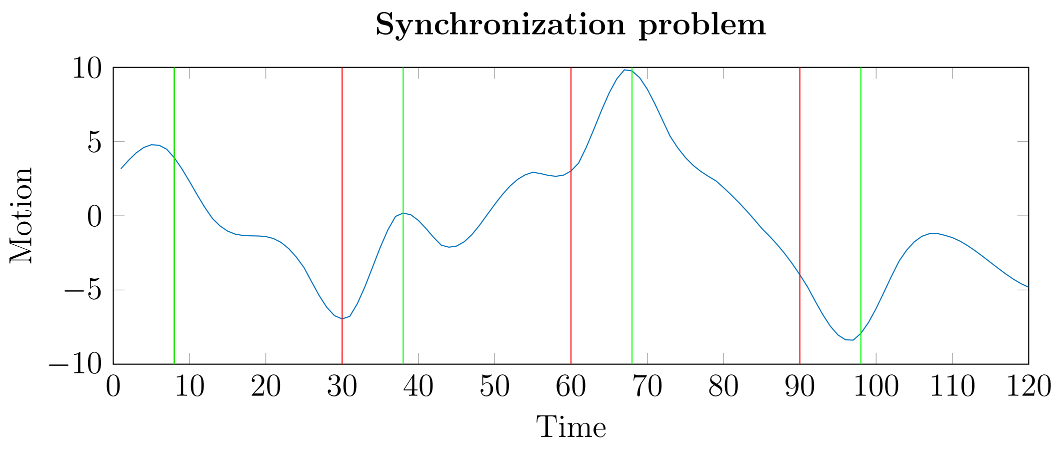

Figure 6.7: Figure showing cause of error due to non synchronized cameras.

To get a better understanding of how large errors are to be expected, some calculations are carried out to determine the impact of the different parameters. The two main parameters that determine the error are the rotation and the distance to an object. The translation in a frame can be calculated from the formula show in 6.3.

$$
\begin{gathered}
T = D \times tan(\theta) \\
T = translation, D = distance, \theta = angle
\end{gathered}
\tag{6.3}
$$

From the ground truth data the maximum rotation around the two other axis than the optical is determined to be 10° per second. By dividing this value with the frame rate the maximum rotation per sample can be determined. In figure 6.8 the impact of the two parameters is shown.

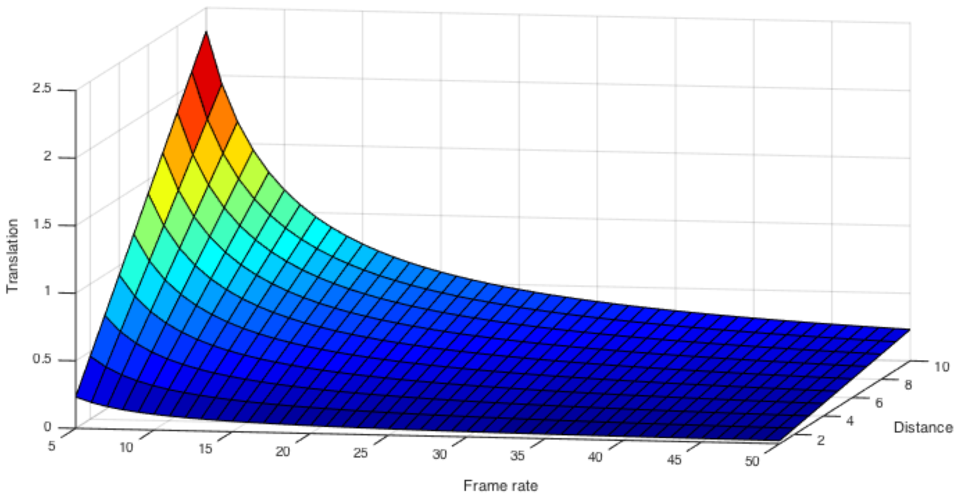

Figure 6.8: Maximum translation with varying frame rate and distance.

In the test data acquired the distance from the camera to the ceiling is 3 meters. At a frame rate of 15 frames per second the translation error can be up to 0.234cm. By increasing the frame rate to 60Hz this error can be reduced to 0.056cm.

Distance sensors

The precision of the distance sensors can also have an impact on the results. It has been found that the distance sensor at times jumps to a wrong height for a period of time. To investigate the precision of the distance sensor further the ground truth data is compared with that of the distance sensor pointing downwards. The height from the ground truth data should be equal to the distance from the downwards-pointing distance sensor. In figure 6.9 the height from the ground truth data(Blue line), and the distance returned by the distance sensor(Red line) is shown. To the right the absolute errror(Red line) and the mean absolute error(Blue line) is shown.

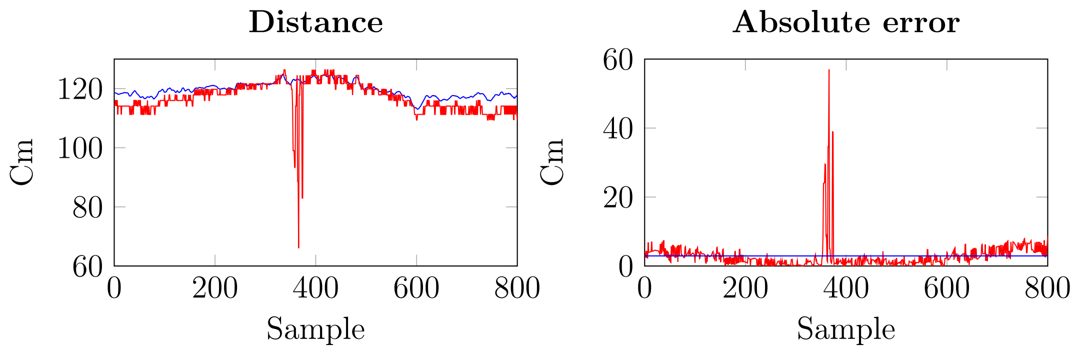

Figure 6.9: Left: is the output from the distance sensor shown along with the height from the ground truth data. **Right:** The absolute error.

From the figure it is clear that the distance does not math the ground truth data. In one of the samples the error in the distance is 59cm. This can lead to large errors in the motion estimate. For a translation of 1 meter at a height of 1.2m, as in the above example, the translation would be calculated as a translation of 0.51m instead if the error in the height were 59cm.

Motion Blur and Low Light

Another cause of errors is simply the quality of the images. To not have the top pointing camera look directly up into the lamps in the ceiling, these where turned off while gathering the data, and some smaller lamps where placed around in the room. This means that there are not as much light present as preferable. A higher quality camera would be able to take pictures in the same lighting conditions but with less noise. An example of a noisy image due to low light in shown in figure 6.11.

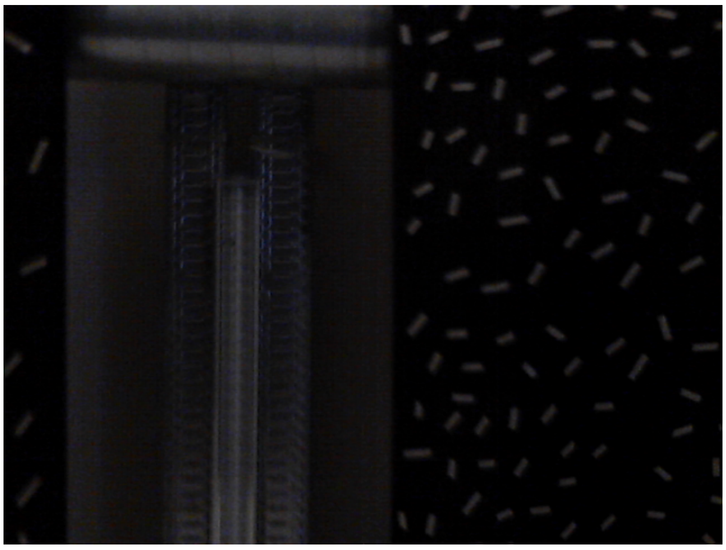

Figure 6.10: Example frame showing the low light problem.

Also due to the quality of the cameras motion blur is introduced when the motion is quick. Especially when the platform rotates, a lot of motion blur is seen in the pictures.

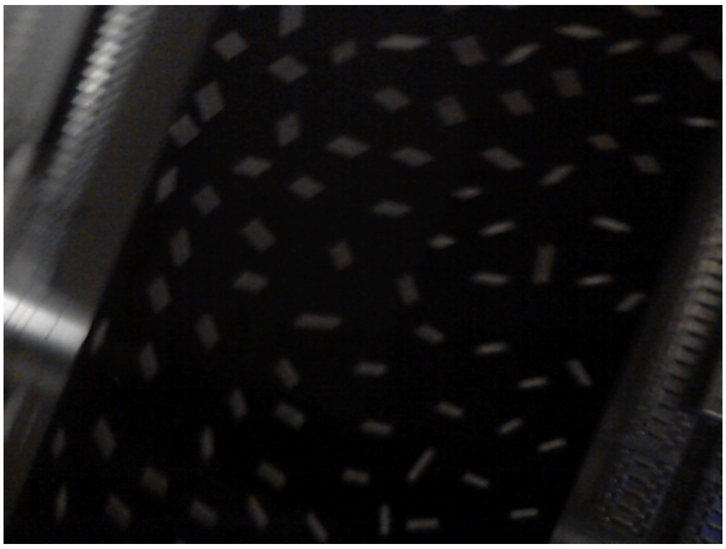

Figure 6.11: Example frame showing the motion blur problem.

As a consequence of the image quality the point quality threshold used for the Shi-Tomasi method have also been set very low. If higher quality images where used the, threshold could be higher. This would leave better point to estimate a more precise homography from. Also blur in the images can lead to errors in the precision of the points.

The work of Ferit Üzer [27] have been dedicated to analyze the effect of blur on feature detectors. In the thesis the effect of motion blur on Harris Corners was analyzed, due to the similarities between Harris corners and Shi-Tomasi corners the effect is assumed to be the same. In the work it was determined that the precision of the detected corners decreased as the amount of blur increases. From this work it was also determined that an effect of motion blur can be a single corner splitting into multiple.

6.2 Obstacle avoidance

In this section the results from the obstacle avoidance tests described in section 5.2 will be presented. First the methods used to describe the results are presented, following the actual results will be presented. Finally some error causes will be discussed.

Result classes

To describe the results from the system the results are grouped in four different classes. These are false positives(FP), true positives(TP) and false negative(FN). These groups are used to group the results into classes than can describe results across different test sets.

False positive

False positives are used to describe a scenario where no obstacles are present, but the system classifies the scenario as containing an obstacle.

True positive

True positives are used to describe a scenario where an obstacle is present and where the system classifies the scenario as containing an obstacle.

False negative

False negatives are used to describe a scenario where an obstacle is present but the system classifies the scenario as not containing an obstacle.

True negative

True negatives are used to describe a scenario where no obstacle is present and the system classifies the scenario as not containing an obstacle.

For each test the occurrence of each of these classes will be logged.

Distance calculation

For each of the tests the distance at where the obstacle is detected is presented. As no distance sensor is available on the test platform the distance is calculated from the camera matrix and the known size of the obstacle. In equation 6.4 the distance calculation is seen.

$$D = \frac{W}{w \times F_x^{-1}} \tag{6.4}$$

Where D is the distance to the obstacle, W is the width of the obstacle in meters, w is the width of the obstacle in pixels and F_x is the horizontal focal length in pixels.
This method can lead to some small errors as the pixel width is manually measured in the frame. However, when the obstacle is close to the camera single pixels does not alter the distance a lot. If this method were used to calculate distances further away from the camera this problem would be much larger. Another thing to notice is the calculated distance corresponds to the distance to the object at the time the frame is grabbed. The drone may actually be closer to the obstacle at this moment due to latency in the overall system.

6.2.1 Uncluttered scenario

In this subsection the results from the tests in a uncluttered environment will be described. First the results from the left and right region tests will be described, then the results from the center region tests will be described

Left and right region

Following the results from the tests of the right and left region will be presented. The tests are divided into two tables. One for the tests where the obstacle is to the left and one for the tests where the obstacle is to the right. The distance is the distance at where the object is first detected.

	FP	**TP**	**FN**	**Distance**
Test1	0	1	0	2.05m
Test2	0	0	1	N/A
Test3	0	0	1	N/A
Test4	0	1	0	2.94m
Test5	0	1	0	2.97m
Mean	0	0.6	0.4	2.65m

Table 6.5: Left region results from uncluttered scenario

	FP	**TP**	**FN**	**Distance**
Test1	0	1	0	3.00m
Test2	0	1	0	2.42m
Test3	0	1	0	2.91m
Test4	0	1	0	2.71m
Test5	0	1	0	2.21m
Mean	0	1	0	2.65m

Table 6.6: Right region results from uncluttered scenario

From the two tables the results from the 10 different tests are shown. In all the cases where the obstacle is detected, it have been detected at a distance higher than the 0.7 meters which have been determined to be the minimum distance needed to stop the drone. The test fails 2 times where the obstacle is not detected. The reason why these tests have failed is discussed later in this chapter section 6.2.4. It is noted that all tests where the system fails is where the obstacle is to the left of the drone. However it is believed that this is just a coincidence and that the left and right region generally performs equally

Center region

Following the results from the tests of the center region will be presented. The distance is the distance at where the object is first detected.

	FP	TP	FN	At distance
Test 1	0	1	0	1.74m
Test 2	0	1	0	1.87m
Test 3	0	1	0	1.55m
Test 4	0	1	0	1.64m
Test 5	0	1	0	1.61m
Test 6	0	1	0	0.93m
Test 7	0	1	0	0.96m
Test 8	0	1	0	1.88m
Test 9	0	1	0	1.84m
Test 10	0	1	0	2.09m
Mean	0	1	0	1.61m

Table 6.7: Center region results from uncluttered scenario

In table 6.7 the results, dedicated to testing the center region, are shown. In all the tests the obstacle have been found with a distance higher than the threshold of 0.7m, which is determined to be the minimum. This means that all of the tests have passed.

6.2.2 Cluttered scenario

In this subsection the results from the tests in a cluttered environment will be described. First the results from the left and right region tests will be described then the results from the center region tests will be described

Left and right region

Following the results from the tests of the right and left region, described in section 5.2, will be presented. Each table presents the results for at single pillar. The distance is the distance at where the object is first detected.

	FP	TP	FN	Distance
Test1	0	1	0	2.69m
Test2	1	1	0	2.43m
Test3	0	1	0	3.33m
Test4	0	1	0	4.17m

Table 6.8: Right region results from pillar 1

	FP	TP	FN	Distance
Test1	0	1	0	2.49m
Test2	0	1	0	4.41m
Test3	0	0	1	N/A
Test4	0	0	1	N/A

Table 6.10: Left region results from pillar 2

	FP	TP	FN	Distance
Test1	0	1	0	2.01m
Test2	0	1	0	3.34m
Test3	0	1	0	3.29m
Test4	0	1	0	3.54m

Table 6.9: Left region results from pillar 3

	FP	TP	FN	Distance
Test1	0	1	0	1.53m
Test2	0	0	1	N/A
Test3	0	1	0	1.48m
Test4	0	0	1	N/A

Table 6.11: Right region results from pillar 4

In the four tables the results from the 16 different tests are shown. In all the cases where the obstacle is detected, it has been detected at a distance higher than the 0.7 meters, which have been determined to be the minimum distance. The test fails 4 times where the obstacle is not detected. The reason why these tests have failed is discussed later in subsection 6.2.4. In table 6.12 the mean results from the tests of the left and right region are shown.

	Mean FP	Mean TP	Mean FN	Mean Distance
Pillar 1	0.25	1	0	3.16m
Pillar 2	0	0.5	0.5	3.45m
Pillar 3	0	1	0	3.05m
Pillar 4	0	0.5	0.5	1.51m
All	0.06	0.75	0.25	2.79m

Table 6.12: Mean results for all 4 pillars left and right region.

Center region

Following the results from the tests of the center region will be presented. Each table presents the results for at single pillar. The distance is the distance at where the object is first detected.

	FP	TP	FN	Distance
Test1	0	1	0	1.62m
Test2	0	1	0	2.14m
Test3	0	1	0	1.43m
Test4	0	1	0	2.78m

Table 6.13: Center region results from pillar 1

	FP	TP	FN	Distance
Test1	0	1	0	1.43m
Test2	0	1	0	1.93m
Test3	0	1	0	1.39m
Test4	0	1	0	1.38m

Table 6.15: Center region results from pillar 2

	FP	TP	FN	Distance
Test1	0	1	0	1.53m
Test2	0	1	0	2.29m
Test3	0	1	0	1.42m
Test4	0	1	0	1.47m

Table 6.14: Center region results from pillar 3

	FP	TP	FN	Distance
Test1	0	1	0	1.19m
Test2	0	1	0	1.28m
Test3	0	1	0	1.02m
Test4	0	1	0	1.18m

Table 6.16: Center region results from pillar 4

In the four tables the results from the 16 tests, dedicated to test the center region, are shown. In all the tests the obstacle is detected and with a distance higher than the 0.7 meters, which was determined to be, the minimum distance needed. In table 6.17 the mean results from the tests are shown.

	Mean FP	Mean TP	Mean FN	Mean Distance
Pillar 1	0	1	0	1.99m
Pillar 2	0	1	0	1.53m
Pillar 3	0	1	0	1.68m
Pillar 4	0	1	0	1.17m
All	0	1	0	1.59m

Table 6.17: Mean results for all 4 pillars center region

6.2.3 Open space scenario

In this subsection the results from the open area scenario will be presented. Table 6.18 shows the results from the 30 tests. Each test is classified as true negative or false positive. In case of a false positive the region, where the false positive is registered, is also shown.

	Path 1			Path 2			Path 3		
	FP	TN	Region	FP	TN	Region	FP	TN	Region
Test 1	0	1	N/A	1	0	Center+Right	1	0	Center
Test 2	0	1	N/A	0	1	N/A	0	1	N/A
Test 3	0	1	N/A	0	1	N/A	0	1	N/A
Test 4	0	1	N/A	0	1	N/A	1	0	Right
Test 5	0	1	N/A	1	0	Right	1	0	Right
Test 6	0	1	N/A	0	1	N/A	0	1	N/A
Test 7	0	1	N/A	0	1	N/A	0	1	N/A
Test 8	0	1	N/A	0	1	N/A	0	1	N/A
Test 9	1	0	Right	1	0	Right	0	1	N/A
Test 10	0	1	N/A	0	1	N/A	1	0	N/A
Mean	0.1	0.9		0.3	0.7		0.4	0.6	

Table 6.18: Results from the 30 test in the open space scenario.

From the table it is seen that the system generates some false positives. The overall false positive rate is 26.7%, however from the test it is seen that typically the regions that generate false positives are actual obstacles. This means that the system is over sensitive to obstacles as obstacles further away are considered as obstacles. The false positive rate is only 6.67% for the center region. In figure 6.12 two examples of this problem is shown.

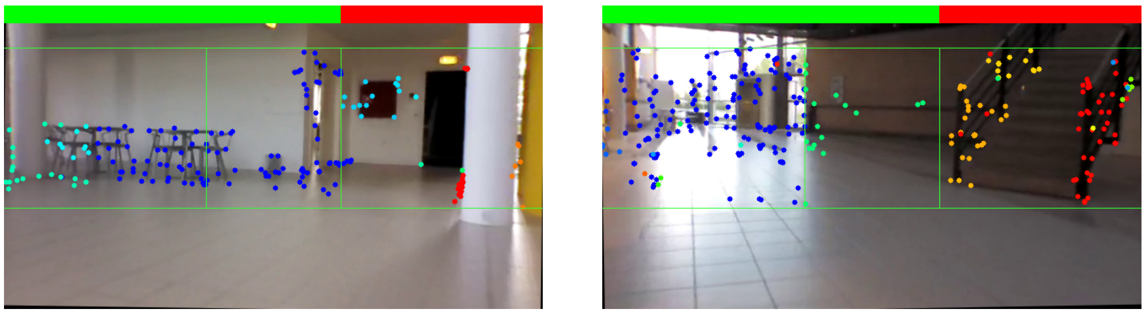
(a) (b)

Figure 6.12: Examples of false positives on actual obstacles.

6.2.4 Error causes

In this subsection the reason behind the errors will be discussed. First the reason for the false positives will be described, then for the false negatives.

False positives

False positives are both observed in the left and right region classifier as well as the center region. First an explanation of the left right region false positives will be described. Secondly an explanation of false positive errors in the center region will be described.

Left and right region

The system is built upon the assumption that the UAV will only fly forwards. If the UAV flies to the left, right or yaws the translation of the tracked points will not only be due to obstacles coming closer. For some unknown reason the drone is having problems flying entirely straight. This is what causes the test to fail. In the cluttered environment pillar 1 test 2, an example of a false positive is seen. For some reason the drone starts to drift to the left. This causes a rapid translation in the tracked points. The classifier mistakes this translation as an obstacle.

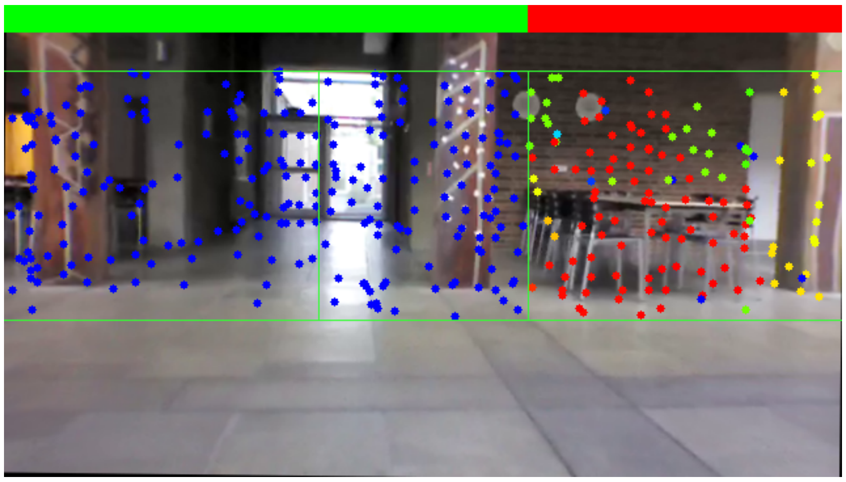

Figure 6.13: Example of a false positive.

In figure 6.13 a screen-shot is shown where a false positive is detected. It is clearly seen that a lot of points, even though they are not on the same plane, is classified with a steep slope.

Center region

In two of the tests a false positive have been registered in the center region. To find out the reason for this, the scales leading up to the misclassification are investigated. In figure 6.14 the scene where the misclassification happens is shown. In figure 6.15 the 30 scales leading up the misclassification are shown.

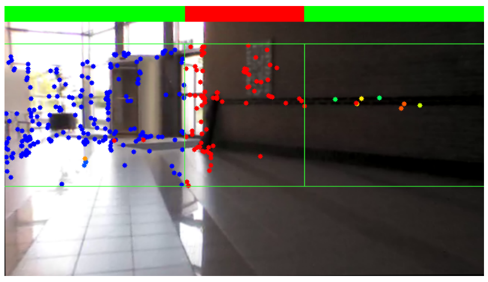
Figure 6.14: The misclassified center region.

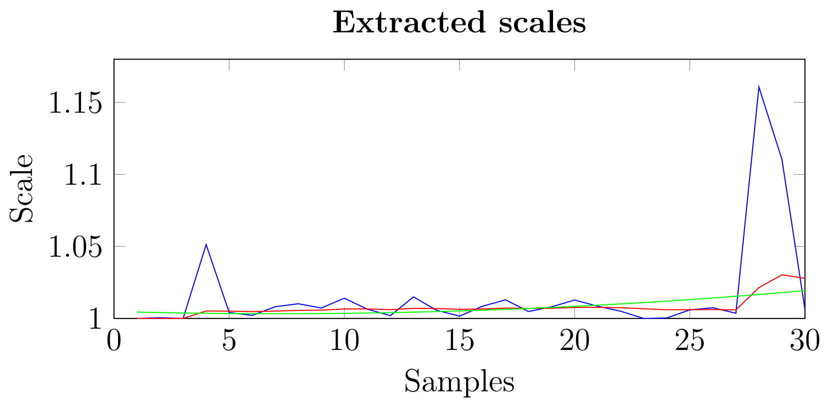

Figure 6.15: The misclassified center region.

From the scales it is clearly seen that errors occurs in the estimates in the scale in sample 28 and 29. By implementing a threshold that the scales cannot be above these sorts of errors could potentially be removed.

False negatives

False negatives are only seen in the left and right region classifier. The problem arises when too few corners are tracked when the drone approaches the obstacle. This will lead to the clustering of the points failing as the distance between the points is above the threshold specified. Normally new points would be added, however these points needs to be tracked for a minimum of 30 frames before they can be used to calculate a slope. In figure 6.16 an example of this problem is shown from pillar 4 test 2.

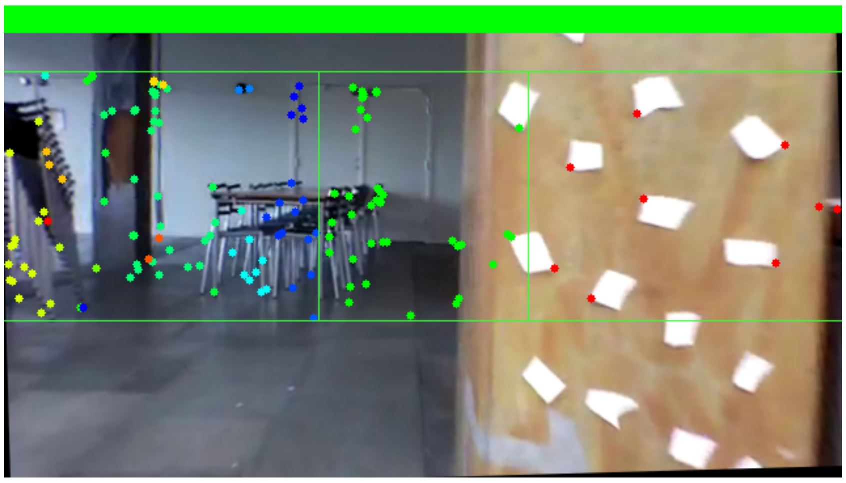
Figure 6.16: Example of a false negative.

In the figure it is seen that the single trajectories on the obstacle is drawn in red, which means that they have a steep slope, but because of the few amount of points they are not clustered correctly leading to a false classification.

7 Conclusion

In this project two systems, to aid indoor UAV flight, have been developed and tested. A visual odometry system has been developed which is capable of estimating horizontal motion as well as rotations around the vertical axis. The system has been tested by implementing the proposed method on a platform, which can simulate UAV flight. The output from the system has been compared with ground truth data in order to analyze the precision of the system. The mean absolute error have been found to be 0.1589cm per sample when the platform if hovering. When motion is introduced the error increases to 0.371cm per sample. The largest error in the estimate is seen when rotations are introduced. For the test with both rotation and translation the error was found to be 0.546cm per sample. With only rotation and no movement the error was found to be 0.452cm per sample. For the rotation the mean absolute error is found to be 0.044° for the hoover test. For the tests with rotation the error varies from 0.125° to 0.194°. The purpose of the visual odometry system was to reveal what kind of precision could be achieved using two cheap cameras. The results section have revealed that motions can definitely be estimated using the proposed method, whether or not the accuracy is high enough will be determined by the platform and the specific use case, in which the system is implemented on. Furthermore the project has revealed which parameters will affect the precision of the system. This is important as it will be help determining which hardware to use for such a system.

As well as a visual odometry system, an obstacle avoidance system has been developed. In order to detect obstacles in front of the camera, as well as to the sides of the camera, two different classifiers have been developed. One for the left and right region and one for the center region of the image. The system has been implemented on an Ar.Drone 2 in order to test the performance of the system. A total of 82 data sets have been collected from three different scenarios. 26 sets containing an obstacle in the left or right region, 26 sets containing an obstacle in the center region and 30 sets with no obstacle. For the left and right region classifier the total true positive rate reaches 78.3% and the obstacles are detected at a mean distance of 2.70m. The center region classifier reaches a rate of 100% true positives, and the obstacles are detected at a mean distance of 1.6m. 30 tests are carried out to determine the rate of false positives. For the left and right region classifier the total false positive rate is 20%. For the center region the false positive rate is 6.67%. With more time and more data sets it is believed that better results could be reached by investigating the thresholds in the two classifiers. The purpose of the obstacle avoidance system was to develop a system that would make is possible for an UAV to detect obstacles and avoid them. The tests have shown that it is possible to detect obstacles in time for an Ar.Drone to avoid them. Whether or not this method would work on any other platform will be dependent on the UAVs ability to stop. The result section have revealed at which distances the method is able to detect obstacles from, using the Ar.Drone as a platform. If another platform is used the distance may be different, and the parameters controlling the algorithm would also have to be altered.

8 Discussion

This chapter will be discussing the implemented methods and the results from the tests. The discussion chapter is divided into three main sections, which are visual odometry, obstacle avoidance, and finally a perspectivation is presented where the possible future use of the technology is discussed.

8.1 Visual Odometry

In this section a discussion of the visual odometry system will be presented. The discussion will be about considerations for further work, and about which considerations are important if the system is to be implemented on an UAV.

Trajectory without rotation

The tests showed that the rotation estimation resulted in errors that lead to large errors in the estimated trajectories. It is believed that the trajectories can be estimated with a much lower error rate if the rotation is known from another sensor. This could be from a magnetometer, which is capable of detecting the magnetic north direction. To see how well the system can generate trajectories with a more precise estimation of rotations, a test is carried out where the rotation is substituted with the one from the ground truth data. In figure 8.1 the trajectory generated from this test is shown. The test data is from the test with no rotations and only translation as described in section 5.1.2.

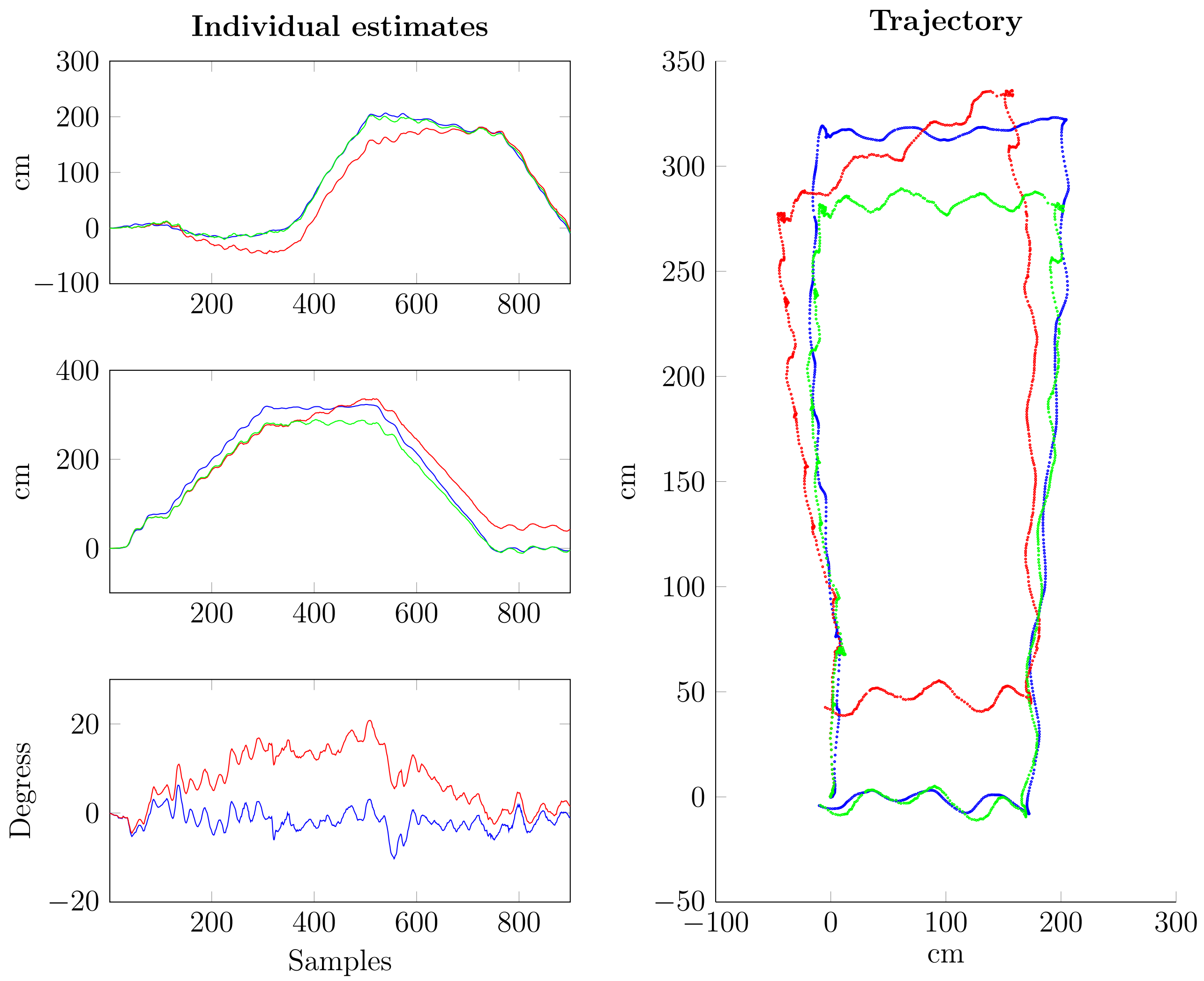

Figure 8.1: Translation test trajectory result. **Blue:** Ground truth **Red:** Trajectory calculated with estimated rotation **Green:** Trajectory calculated with ground truth rotation

From the figure it is seen that the trajectory is estimated with a higher precision when omitting the rotation estimate. It is also seen that the x-axis is almost perfect throughout the trajectory where the y-axis is a bit too low. A reason for this could be that the focal length in pixels is not estimated correctly leading to a scale error in the y-axis. This leads to the assumption that the trajectory can be used to determine the position of the UAV within smaller periods of time. However it is still seen that some error is present when calculating the trajectory. To be able to estimate the UAVs position precisely indoor, it would have to be coupled with another system. A solution could be to put some known markers on the floor around the room the UAV will navigate. If the position of these markers are known then the trajectory could be corrected each time the UAV detects one of these markers. The visual odometry system could then be used to determine the position between these markers. Such a system would allow the UAV to navigate indoors only using cameras.

Fusion with other sensors

It would be interesting to see how the system would perform if the data was fused with another sensor. As this system is meant to be implemented on an UAV it would be natural to fuse the data with that of the IMU on board the UAV. By doing so some of the spikes that leads to errors in the trajectory could possibly be removed.

Further testing

To get a better understanding of how the system would perform, further testing is needed. A test setup on an actual UAV would be preferred. Furthermore some of the probable error causes should be fixed. This means two synchronized cameras, a higher frame rate and a more precise distance sensor. As the platform was manually moved around in the laboratory the height difference in the data sets are not varying much. It would be interesting to investigate how the system would perform when variations in height occurs.

Implementation in real system

From the tests of the method, important considerations have been revealed. The planes the UAV flies over and under must contain extractable features for the motion estimation to function properly. The frame rate of the cameras are essential for the precision of the per sample results. A well-lit area is important as low light scenes are harder to track compared to well lit scenes. This is due to the lack of contrast that will make it difficult to find corner points. It has also been found that it will be beneficial to use cameras with a trigger so that both cameras can be synchronized, if the motion is to be fused with other sensor data, it is also essential that the other sensor can be synchronized with the cameras.

8.2 Obstacle avoidance

In this section a discussion of the obstacle avoidance system will be presented.

Alternative left and right region classifier

From the results it is seen that the left and right region classifier generates all the occurrences of false negatives, and a large part of the false positives. This is mainly due to the fact that the UAV used to test the system cannot fly entirely straight. Fewer errors are seen in the center region as this classifier method works independently of translations in the frame and only used the scale between the points. It would be interesting to try and adapt this translation independent center method, into an alternative classifier for the left and right region. By doing so it is believed that the error rates could be drastically improved.

Stabilized camera

As mentioned translations caused by movement of the UAV causes errors. It would be interesting to gather some test data from an UAV with a camera stabilizer. Larger UAVs are commonly equipped with a camera gimbal. A camera gimbal is essentially 3 motors connected to an IMU. When the UAV rotates, the motors will counter act these rotations, which results in a very steady image. In the test implementation a rotation of the image in one of the axes is implemented. A gimbal would stabilize the image in all 3 axes.

Alternate center classification threshold

From the test of the center region classifier it is seen that not a single obstacle have been missed. This leads to a sensitivity of 100%. As a few false positives have been observed the precision is lower than the sensitivity. This could be an indication of a too low threshold in the center classifier. This problem is illustrated in figure 8.2.

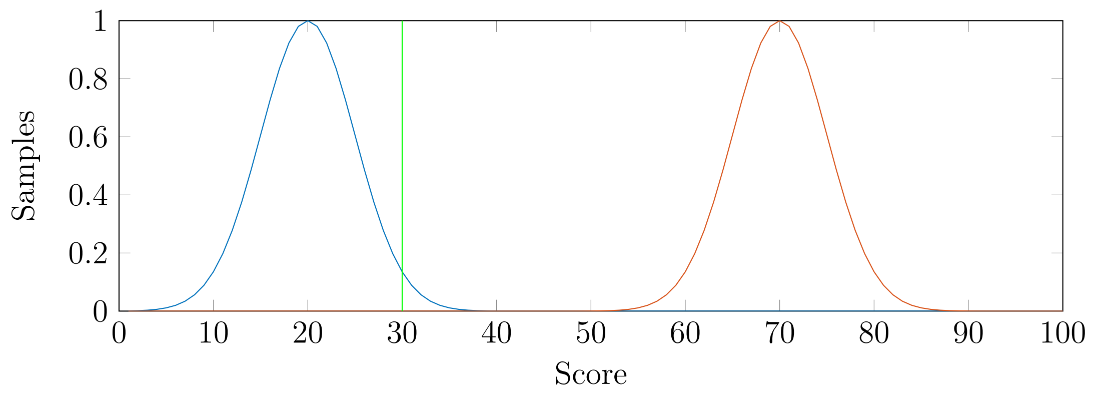

Figure 8.2: Classifier threshold. **Blue:** No obstacle. **Red:** Obstacle. **Green:** Threshold.

The above plot is generated from fictional data to illustrate the problem. It would be interesting to analyze the actual distribution of scores for all the gathered data. This way a more informed decision about the threshold could be taken. At this point the threshold have been set by trial and error from a few initial data sets.

Implementation real system

From the tests of the proposed method, some important requirements to the platform has been revealed. The tests showed that the left and right region classifier has some problems when the platform is not able to fly straight without rotations and translations. This means that either the platforms need to be able to fly straight or the camera needs to be mounted on a gimbal for stabilization. It is noted that this problem only affects the left and right region and not the ability to detect obstacles in front of the camera. The lowest distance at which an obstacle has been detected is 0.95m. This leads to the requirement that the UAV has to be able to stop in less than 0.95m. It is noted that the distance at which the method is able to detect obstacles may vary if another camera, with another frame rate is chosen.

8.3 Perspectivation

In this project two systems to help UAVs fly indoor have been developed. These systems alone are not enough to allow indoor UAV flight all together. However these systems can be a vital part of how the UAVs can be moved from their current environment outdoors, into indoor environments. By using cameras and computer vision it is believed that the UAVs will be able to fly autonomous in more or less all areas where human currently can remote control them. By equipping the UAVs with cameras they have the ability to see as we humans can, which means they will be able to perform the tasks that we humans can based on our vision. It is the believe of the author that computer vision in general will be the key factor in moving the UAVs from a remote controlled camera, which is essentially what UAVs are today, to the next step where UAVs can automatically make decisions. The two developed systems are not only usable indoors, but also outdoors. A lot of larger companies are experimenting with UAVs for delivering packets. For this to become a reality the UAVs must be able to find their way to a defined address. It is impossible for the UAV to know what is between its current location and destination. Therefore the UAVs must be able to make decisions about obstacles and route in real-time based on what they see. When UAVs fly in the middle of larger buildings it can be difficult to get a good GPS signal. Therefore the visual odometry system can help an UAV in periods where the GPS signal is lost. It is believed by the author that UAVs will be able to carry out a lot of the tasks that humans today are involved in, and in the near future it is believed that it will be a common sight to see an autonomous UAV without a person in the other end.

Bibliography

[1] *Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing*, 2011.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[3] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning Opencv, 1st Edition.* O'Reilly Media, Inc., first edition, 2008.

[4] Jason Campbell, Rahul Sukthankar, Illah Nourbakhsh, and Aroon Pahwa. A robust visual odometry and precipice detection system using consumergrade monocular vision. In *in Proceedings of the 2005 IEEE International Conference on Robotics and Automation ICRA 2005*, pages 3421–3427, 2005.

[5] Elan Dubrofsky. Homography Estimation. Master's thesis, THE UNIVERSITY OF BRITISH COLUMBIA,, Vancouver, 2007.

[6] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, pages 363–370, Berlin, Heidelberg, 2003. Springer-Verlag.

[7] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[8] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[9] Tomáš Krajník, Vojtěch Vonásek, Daniel Fišer, and Jan Faigl. AR-Drone as a Platform for Robotic Research and Education. In *Research and Education in Robotics: EUROBOT 2011*, Heidelberg, 2011. Springer.

[10] Logitech. Logitech homepage `http://www.logitech.com/en-gb/product/hd-webcam-c310`.

[11] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[12] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[13] Maxbotix. Maxbotix homepage `http://www.maxbotix.com/Ultrasonic_Sensors/MB1320.htm`.

[14] Annalisa Milella and Roland Siegwart. Stereo-based ego-motion estimation using pixel tracking and iterative closest point. 2006.

[15] Tomoyuki Mori and Sebastian Scherer. First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In *ICRA*, pages 1750–1757. IEEE, 2013.

[16] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):756–777, June 2004.

[17] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. pages 652–659, 2004.

[18] OpenCV. Opencv homepage `http://opencv.org`.

[19] OptiTrack. Motive product homepage `http://www.optitrack.com/products/motive/`.

[20] Parrot. Ardrone product homepage `http://ardrone2.parrot.com/`.

[21] D Scaramuzza and R Siegwart. Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles. *IEEE Transactions on Robotics*, October 2008. Special Issue on Visual SLAM.

[22] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robot. Automat. Mag.*, 18(4):80–92, 2011.

[23] Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 – 600, 1994.

[24] SICT. Study regulation for vgis 1.-4. semester `http://www.sict.aau.dk/digitalAssets/36/36173_vision_graphics_interactive_systems_godkendt.pdf`. Downloaded 2013-09-20.

[25] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ICCV '99, pages 298–372, London, UK, UK, 2000. Springer-Verlag.

[26] S Zingg, D Scaramuzza, S Weiss, and R Siegwart. Mav navigation through indoor corridors using optical flow. In *Proc. of The IEEE International Conference on Robotics and Automation (ICRA)*, May 2010.

[27] Ferit Üzer. Camera motion blur and its effect on feature detectors. Master's thesis, Middle east techincal university, 2010.

A Alternative center obstacle classification

The purpose of this appendix is to describe an alternative method that was initially used to detect scale in the center region of the obstacle avoidance classifier. The idea behind this method was as in the final method to extract the scaling factor between frames. In the final method the homography matrix is used to determine scale and is extracted from all tracked points in the center region. The initial approach was to cluster trajectories based on euclidean distance and estimate the scale between each group of two points. The first three steps of the pipeline, image acquisition, feature extraction and feature detection are the same as described in the system composition section 3.2. Therefore only the classification step of this method will be described.

For each point its nearest neighbor is found that is not already grouped with another points. For each of these groups of two points the scale between them over the last 30 frames is calculated. In figure A.1 an example of the grouped point are shown. The grouped points are connected by a line. The points and the line connecting them is colored depending on their scale. Red corresponds to a high scale and blue corresponds to a low score.

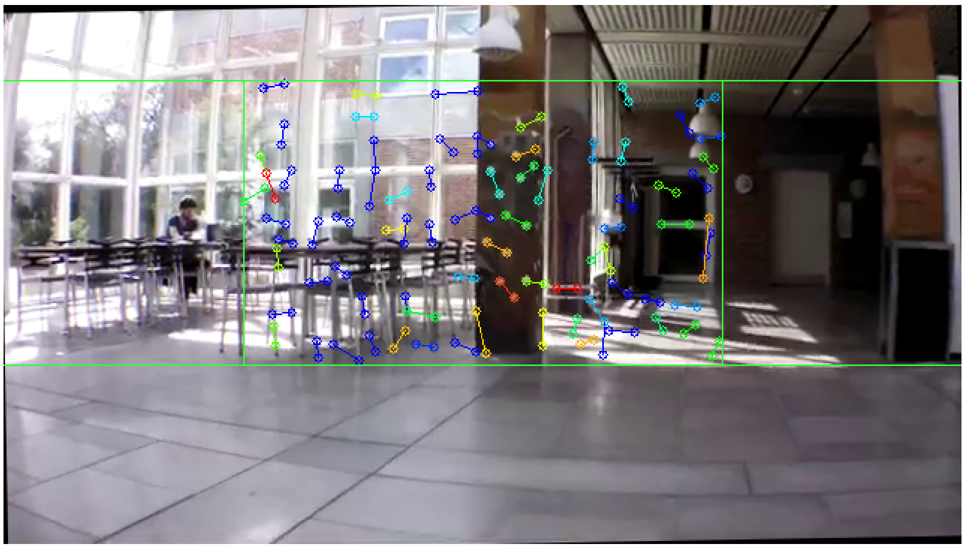

Figure A.1: Example clustered points.

For the method to work groups lying on the obstacle should be more red than groups lying on the background. From the figure it is seen that there is no coherence between the color of the groups and the surface they lie on. It is believed that the scale between the frames is so small that the noise from the feature detection is so large that it is impossible to classify the scales. For this reason the other method using the homography is used. As this approach uses more points to estimate the scale, the system becomes more resistant to noise.

B Obstacle detection from extrinsic parameters

The purpose of this appendix is to describe an alternative method to detect obstacles but ultimately was discarded as the results was deemed unusable. The idea behind this method is to take advantage of the fact that the Ar.Drone has a IMU available where motion and rotations can be extracted from. The first three steps of the pipeline, image acquisition, feature extraction and feature detection are the same as described in the system composition section 3.2. Therefore only the classification step of this method will be described.

Classification

The classification step is responsible for determining whether or not obstacles are present in the scene. Different from the chosen method this alternative method will be classifying the entire image as one instead of dividing the frame into sections. Ultimately this method would be able to classify all the tracked trajectories as obstacles or non obstacles.

The method is based on the idea of extracting extrinsic parameters from the IMU of the drone. As the velocity and the rotations are sent from the drone these can be used to calculate the extrinsic matrix. The velocity can be mapped to a distance using the sample time. First each point is mapped to world coordinated by multiplying with the inverse camera matrix, this is shown in equation B.1.

$$\begin{gathered} p_w = K^-1p \\ p_w = \text{Point in world}, K = \text{Camera matrix} \end{gathered} \tag{B.1}$$

The point is normalized by dividing with the z component of the vector. In the next step the coordinate is multiplied with the extrinsic parameter matrix. This is seen in equation B.2.

$$\begin{gathered} p_r = KEp_w \\ p_r = \text{Point after rotation and translation}, K = \text{Camera Matrix}, E = \text{Extrinsic paramters} \end{gathered} \tag{B.2}$$

The Extrinsic parameters are found using equation B.3.

$$E = \begin{bmatrix} R & T \end{bmatrix}$$
$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) \\ 0 & -sin(\alpha) & cos(\alpha) \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & -sin(\beta) \\ 0 & 1 & 0 \\ sin(\beta) & 0 & cos(\beta) \end{bmatrix} \begin{bmatrix} cos(\gamma) & sin(\gamma) & 0 \\ -sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}, T = \begin{pmatrix} T_x & T_y & T_z \end{pmatrix}^T$$
$$\alpha = \text{Pitch angle}, \beta = \text{Yaw angle}, \gamma = \text{Roll angle} \tag{B.3}$$

To validate how far away an object is the point p_r is compared with the one tracked using Lucas-Kanade. In figure B.1 an example of the calculated points for motion of 10 samples is shown.

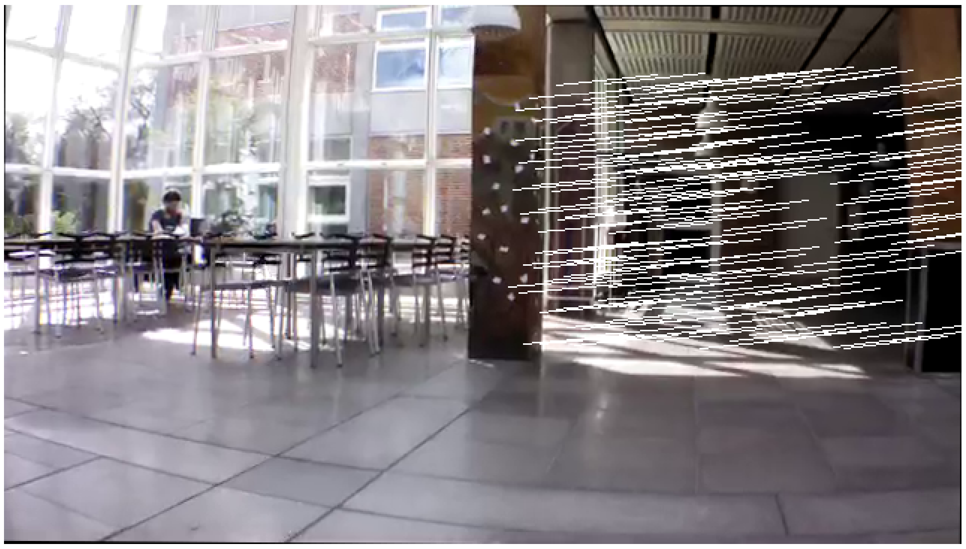

Figure B.1: Example of the calculated points.

After some initial testing with this method it was discarded as the information from the IMU proved to be too imprecise to estimate the extrinsic parameters.