University of Twente

EEMCS / Electrical Engineering
Control Engineering

Using a Time-of-Flight Camera for Autonomous Indoor Navigation

Raymond Kaspers

MSc report

Supervisors:
prof.dr.ir. S. Stramigioli
dr.ing. R. Carloni
ir. E.C. Dertien
dr.ir. F. van der Heijden

January 2011

Report nr. 001CE2011
Control Engineering
EE-Math-CS
University of Twente
P.O.Box 217
7500 AE Enschede
The Netherlands

Using a Time-of-Flight Camera for Autonomous Indoor Navigation

Raymond Kaspers

January 5, 2011

Summary

At Philips Floor Care autonomous vacuum cleaner robots are being developed. These robots autonomously find their way in unknown domestic environments. For such a robot to perform its job intelligently, it has to be able to observe its surroundings and process these observations into a map of the environment. A relatively new sensor for this task is the Time-of-Flight camera, which is capable of capturing a 3D image of its subject, multiple times a second.

In this research the potential of the Time-of-Flight camera for autonomous indoor navigation is investigated. A characterization of the camera is made and algorithms are developed that extract relevant data from the 3D images (corners, jump edges and planes). These algorithms are evaluated with respect to speed, accuracy and robustness. Finally, a proof-of-principle setup is made using several orderings of typical pieces of furniture in a test room. The recorded 3D images are then processed using the developed feature extraction algorithms. The resulting features are fed to a Simultaneous Localization and Mapping (SLAM) algorithm, which estimates a map of the detected landmarks and the camera position within this map.

Based on the results it is concluded that the selected features are robustly detected, are abundant in the test setups and can be extracted and processed by the SLAM algorithm in real time. The proof-of-principle shows that the features are accurate enough to result in stable SLAM within an average room. There are some issues identified that negatively affect accuracy, which require further research of both the camera and the algorithms. However, it can be concluded that the time-of-flight camera has good potential for autonomous indoor navigation and that the selected features can be considered good candidates for this purpose.

Preface

This thesis marks the end of my studies in Electrical Engineering at the University of Twente. This has been an important and pleasant period of my life, in which I not only learned how to be an engineer, but also made many new friends and built nice memories.

The urge to become an electrical engineer, already came early. As a little kid, I was already fascinated by electronics, always fooling around with all kinds of electronic components, taking things apart, occasionally really fixing something. Now that I have learned so much more about the subject, I am happy that this fascination is still as strong as it was then. I guess people are really born an engineer.

However, fascination is not always enough. Without the support of friends, family and teachers, it is hard to keep on the right track. In this preface, I would therefore like to thank everyone that supported me during my time at the university.

I would like to thank professor Stefano Stramigioli for his endless enthusiasm, his inspiring lectures and for talking me into this research assignment. I would like to thank Edwin Dertien, my daily supervisor, for supporting me during my research and motivating me with his always positive attitude.

Many thanks go out to the colleagues and supervisors Barry and Karel at Philips, who made the last few months of my study an enjoyable period of time. This goes also for my fellow students at Control Engineering and especially for Robert, who I gladly cooperated with during large parts of my master's studies.

I would like to thank my grand parents who have raised me to be the person I am now, who have always believed in me during every step of the way and have always motivated me to do my best. I would like to thank my mother for all her love, even during difficult times, my uncle Dirk for all the discussions about our common interest and my aunt Greet who I always could talk to about anything. Then there are the parents of my girlfriend, Jacob and Saakje, who have always been with me, both during my internship in the United States and during my graduation assignment.

My special thanks go to Marieke. I would like to thank you for believing in me... for supporting me during my downs, for celebrating with me during my ups, but especially for just being you and being able to put a smile on my face.

Raymond Kaspers

January 4, 2011

Contents

1 Introduction

Autonomous robot navigation is a popular field of research, as more and more applications are found for robots that are able to navigate and discover the environment by themselves. One of the companies that is actively conducting research in this field is Philips. The application that Philips focuses on is the autonomous vacuum cleaner robot. Such a device is able to clean the floor in a room, without intervention of a human being. The main functionality of a vacuum cleaner robot is of course moving around in a room, while picking up dust and debris on the floor. However, to clean a whole floor the robot needs a certain level of intelligence. It has to be able to observe its environment and react to it, by avoiding obstacles and do motion planning that steers the robot in directions where it has not cleaned yet. For this purpose, the robot needs to keep track of its position and the structure of the environment at all times.

Many approaches to the problem of autonomous navigation are found in literature and numerous types of sensors are available for an autonomous robot to 'see' into the world (camera's, infrared distance sensors, laser scanners, ultrasound, physical bumpers, etcetera). One of the newer technologies that is available on the market is the time-of-flight camera. This type of camera is capable of capturing not only intensity, but also distance. Hence, it is capable of making 3D images of the scene.

1.1 Problem Description

The research described in this report is done on behalf of Philips Floor Care, to investigate the potential of such a camera for the purpose of autonomous indoor navigation in domestic environments. For a sensor to be useful for autonomous navigation, it must be able to capture data that can be used to accurately determine the geometry of the environment that it observes. Usually, the raw data from the sensor is too complex to process and has to be reduced. The interesting parts of the data are called features. In this research, it is investigated which features can be extracted from the time-of-flight data that can be used to aid in the navigation and localization problem.

The chosen approach is to make a characterization of the data that the camera provides and develop several feature extraction algorithms based on ideas from literature. These algorithms are then evaluated in a proof-of-principle setup, that uses the extracted features from several realistic indoor scenes in a SLAM (Simultaneous Localization and Mapping) algorithm.

1.2 Thesis Structure

Chapter 2 starts with an analysis of the characteristics of the time of flight camera. Hereafter, an introduction and analysis of the used localization and mapping algorithm is given, which is followed by a motivation of the choice of features based on this algorithm. The chapter concludes with a detailed analysis of the developed feature extraction algorithms. In chapter 3, an overview of the developed system is given, followed by a structural overview of the implementation of the different feature extraction algorithms. For each algorithm, an analysis of the computational complexity is given. The chapter concludes with an overview of the implementation of the localization and mapping algorithm. In chapter 4 the developed models and algorithms are evaluated. The last chapter of the report covers the conclusions and recommendations.

2 Analysis

This chapter starts with a short analysis of the application, which provides a context for the research. This is followed by a characterization of the time-of-flight camera and the used localization and mapping algorithm. Based on properties of the camera and the localization algorithm, an analysis is given of the developed algorithms that process the sensor data into features that can be used for autonomous indoor navigation.

2.1 The Application

In figure 2.1, a typical vacuum cleaner robot is shown. The vacuum cleaner moves around on the floor on wheels and has several sensors that it uses to map the environment. As the movements that such a robot makes are on the floor, the assumption is made in that all movements are in the 2D horizontal plane. This makes the problem of navigation a 2D problem.

Figure 2.1: Typical vacuum cleaner robot (Philips HomeRun 2010

The algorithms that have been developed in this research are optimezed for use in a domestic environment. A domestic environment is generally divided in rooms, which are bounded by walls. The most dominant type of object in a room is usually furniture. As the walls and the furniture are the most common objects in a room, the navigation developed here is based on properties of these objects.

While a vacuum cleaner does its job, it moves both around the furniture and under the furniture. This makes that the environment is observed from both far away, very close by and from different perspectives. The actual movements of the robot also affect the measurements, which has to be taken into consideration. Note that all objects are considered not to move. In a real environment, people and pets might be walking around, which leads to error if these are used to navigate on.

In this research, all development and testing is done without an actual robot. The final application will be to integrate a time-of-flight camera in a vacuum cleaner robot and the research is therefore done with this goal in mind. The chosen algorithms are selected for speed. Furthermore, the software is written in C++ and designed to be modular, so it can be easily adapted for an embedded platform.

2.2 Frame Conventions

In the upcoming sections, algorithms and models are defined, based on Cartesian coordinates. To prevent confusion, a convention for the reference frame is first chosen. The frame is defined as in figure 2.2.

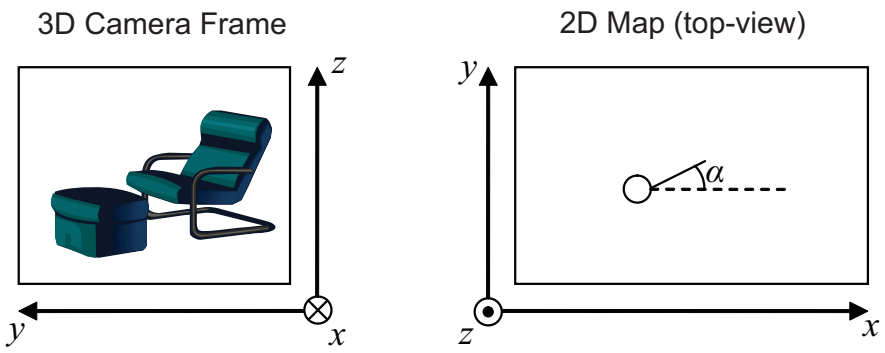


Figure 2.2: Choice of Reference Frame

On the left of the figure, the frame of the camera is displayed. This frame is defined as having the vertical direction defined as z, the horizontal direction as y and the forward direction (depth) as x. This seems a somewhat unconventional choice of coordinates, but is chosen to be able to match with the 2D top-view map as shown in the right part of the figure. An orientation of $\alpha = 0$, means that the camera is pointing in x-direction. If it is unclear, which frame is meant, coordinates will be designated with a superscript $[c]$ for the camera frame, or a superscript $[m]$ for the map frame.

The robot pose will be defined by $\vec{p}$, as in equation (2.1).

$$\vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_\alpha \end{bmatrix} \tag{2.1}$$

2.3 Analysis of the Time-of-Flight Camera

Time-of-flight cameras are sensors that are able to capture a 3D image of a scene in a single shot. They can therefore best be compared to stereo vision and structured light setups. Each of these technologies has its own strengths and weaknesses. A characterization of the time-of-flight camera is made in the upcoming paragraphs, including the strengths and weaknesses of this particular sensor.

2.3.1 The Time-Of-Flight Principle

A basic visualization of the appearance of the camera and its operation is shown in figure 2.3.

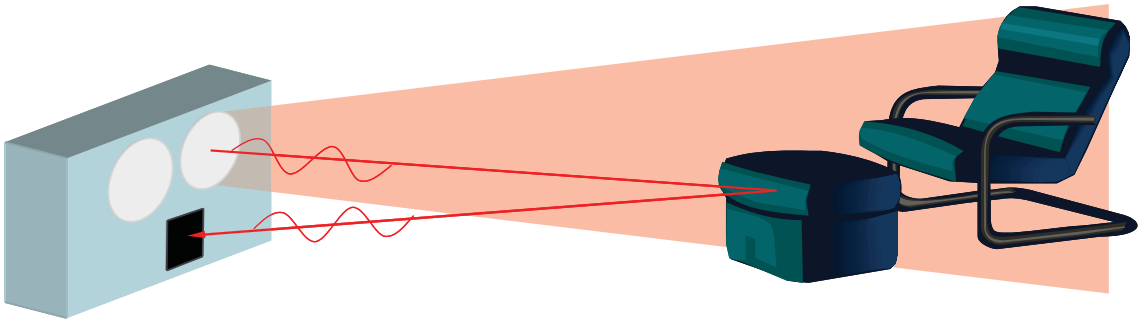


Figure 2.3: Basic setup of a time-of-flight camera in a scene

The camera has two infrared LED-light sources in the front of the device. These two light sources emit modulated light that illuminates the scene to be measured. The light reflects on the scene and is focused by a lens on a PMD (Photonic Mixing Device) sensor, which is dis-

cussed further in section 2.3.4. The PMD sensor consists of an array of pixels that individually extract amplitude, offset (due to background illumination) and distance. Note that as each pixel is able to do its own distance measurement, the system does not require further processing and is not sensitive to lack of texture like is the case with stereo vision.

2.3.2 Choice of Sensor

The camera that is used in this research is developed by IEE, a manufacturer of time-of-flight technology in Luxembourg (http://www.iee.lu/). The technology that this company uses comes from the Swissranger line, a type of camera that is used frequently in the academic world. IEE is actively bringing this technology from the academic to the industrial and consumer market by improving on the technology and its production process. Philips and IEE have decided to cooperate on further development of the time-of-flight camera and the applications for this type of sensor. This research therefore exclusively uses the 3D MLI time-of-flight camera from IEE.

2.3.3 Internal Parameters of the Used Camera

In table 2.1, the set of parameters is shown that are provided by the manufacturer.

Modulation Frequency	20 MHz
Resolution (horiz x vert)	61x56 pixels
Viewing Angles (horiz x vert)	65°x 45°
Frame rate	5 Hz
Light power	13 W

Table 2.1: Camera figures

2.3.4 Photonic Mixing Device Principles

In contrast to what the name 'time-of-flight camera' implies, the sensor does not measure the time of flight directly. Instead, it measures the phase shift between the outgoing light and the incoming light, by mixing both waves and integrating the result. Before a phase shift can be measured, the light first has to be modulated. The modulation frequency of this particular sensor is 20 MHz. The wavelength can be derived, using equation (2.2).

$$\lambda = \frac{c}{f} = \frac{3.00 \cdot 10^8}{2.00 \cdot 10^7} = 15.0\,\mathrm{m} \tag{2.2}$$

Because the sensor measures phase shift, it cannot distinguish between a delay of one period, a delay of two periods and so on. The traveling distance of the light that can be measured is equal to the wavelength. As the light has to travel back and forth, the range is limited to 7.5 m.

Physical Structure of a Pixel

A basic PMD sensor (as described in Thorsten Ringbeck (2007)) is built up out of separate pixels that each consist of two MOS structures. The substrate of the structures consists of a P-type semiconductor (having electrons as minority particles). On both sides a N-type semiconductor is added, to form two junctions on either side of the substrate. Above the substrate are two conducting transparent gates, which can be driven independently. Such a MOS structure is shown in figure 2.4.

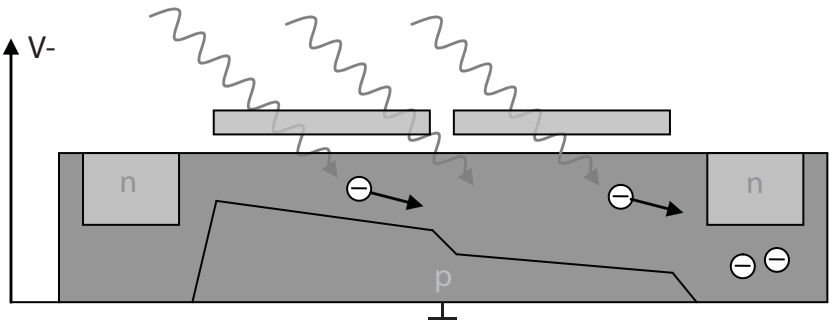


Figure 2.4: PMD MOS structure

By applying a negative bias on the gates and the substrate, the minority carriers are forced towards the bottom of the substrate, creating a depletion layer on top of the substrate. As the gates are transparent, light can reach the substrate and will generate carriers. A voltage difference (on addition to the bias) between the two gates, causes an electric field to be formed in the substrate (as shown in figure 2.4). The generated electrons drift to either the left junction or the right junction, depending on the polarity of the voltage difference. Here, they are trapped in a potential well, where they are read out by the connected circuitry.

By alternating the voltage difference between the two gates with the modulation frequency of the light, the generated electrons are distributed over the two junctions under influence of the phase shift. No phase shift causes all electrons to drift to the left, while 180°of phase shift will cause them to drift to the right. The gates of the second FET structure are modulated with a the same signal, but now delayed with a phase shift of 90°. This makes this second structure sensitive to 90°or 270°phase shift of the incoming light.

The four resulting signals are proportional to to the product of the incoming modulated light signal with the original modulation signal at four different phase shifts, integrated over time. If the four corresponding signals are defined as s_0, s_{90}, s_{180} and s_{270}, phase shift ϕ, amplitude A and offset I_0 can be extracted using the equations (2.3).

$$\phi = \arctan\left(\frac{s_0 - s_{180}}{s_{90} - s_{270}}\right) \tag{2.3}$$

$$A = \frac{s_0 + s_{90} + s_{180} + s_{270}}{4} \tag{2.4}$$

$$I_0 = \frac{\sqrt{(s_0 - s_{180})^2 + (s_{90} - s_{270})^2}}{2} \tag{2.5}$$

Multiple Measurements for Dynamic Range

A problem that occurs with determining the distance in a range of several meters is that there is a large difference in light power (proportional to the distance squared), between objects close by and objects far away. The time-of-flight camera uses four different integration times to accommodate for this. After all the measurements have been completed, the camera evaluates per pixel which integration time is best is best suited for the particular signal strength. Note that every measurement takes a finite amount of time. Pixels that have been measured using different integration times, will therefore differ in the time instant that they were captured. This has to be taken into account if the camera is moved.

2.3.5 Basic Camera Model

A camera sensor consists of a matrix of pixels. As pixels are not infinitely small points but surfaces, the light that is captured by one pixel, also comes from a surface with a size that depends

on the distance. Instead of using a complex model that takes these surfaces into account, the sensor is first approximated with a pinhole model. In this model, each pixel is modeled as an infinitely small point, that captures the distance to an infinitely small point in the scene.

The pinhole model describes the camera as consisting of a sensor and a scene with a plate in between that contains an infinitely small hole. This hole can be thought of as the ideal lens, because it covers the sensor from light, that does not come from straight forward. Such a camera is therefore in focus for every distance.

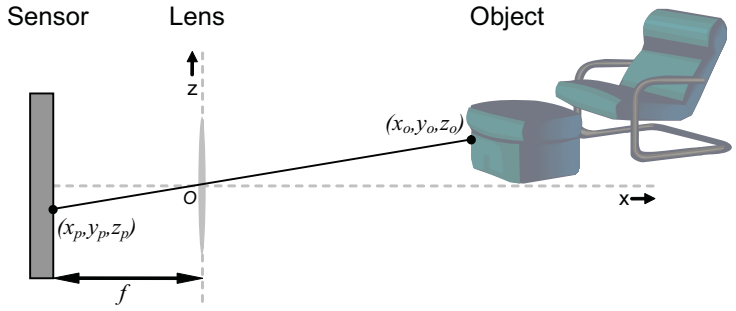


Figure 2.5: Graphical Representation of the Pinhole Camera Model

The mathematical equation that covers the pinhole model is given in equation 2.6, where (y_p, z_p) is the location of the pixel on the sensor, (x_o, y_o, z_o) is the measured point in space on the object and f is the focal distance of the camera.

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = -\frac{f}{x_o} \begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix} \tag{2.6}$$

One of the implications of this model is that the y-coordinates are monotonically increasing with pixel indices from right to left and z-coordinates are monotonically increasing with pixel positions from bottom to top. This geometric structure is an important property of the data, which is exploited in the processing algorithms covered later on.

2.3.6 Disturbances

The ideal camera model that is sketched above is not sufficient to work with as it neglects too much of the disturbances that are introduced in reality. In the upcoming paragraphs, these different noise sources are covered.

Lens Distortion

One of the causes of error in the camera is caused by distortion that is introduced by the lens. Because of the curvature of the lens, the virtual ray that can be drawn from the pixel through the lens gets bent as it propagates through the lens (as shown in figure 2.6. The bending effect is called radial distortion and depends on the angle between the ray and the normal of the lens.

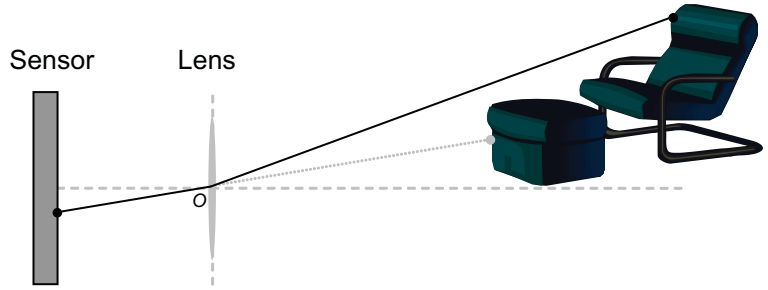


Figure 2.6: Radial distortion in lens

For the supplied sensor, the lens parameters are not available, but it has been accurately calibrated. For each pixel, a unit vector is specified in the sensor software that marks the direction from the origin of the lens to the scene. A graphical outline of the distribution of these unity vectors is given in figure 2.7. It represents what the camera would return if a plane at a distance of 1 m is observed. This figure is determined in experiment (A.1).

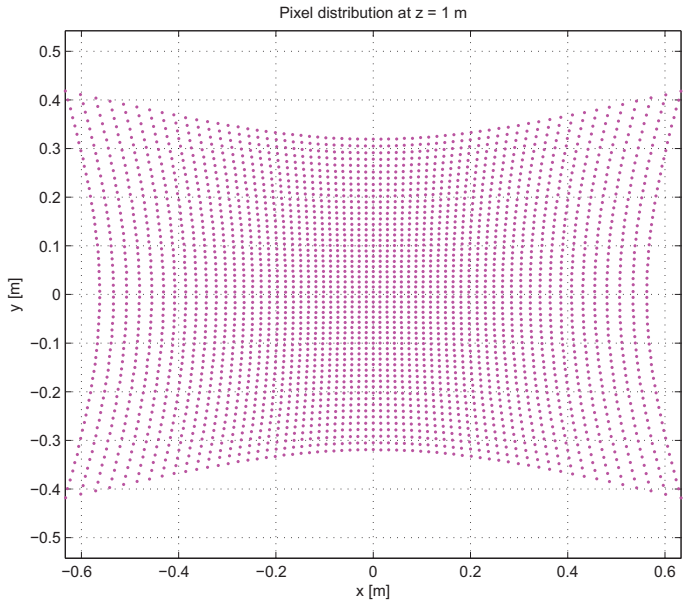


Figure 2.7: Pixel distribution at unity distance

Range Noise

The range measurements suffer from additive noise that can be approximated by a normal distribution, with zero mean and a standard deviation of σ_r. This noise is dependent on the number of photons generated by active light (N_{active}), the background illumination ($N_{background}$) and the electronic shot noise (N_{pseudo}). An approximation of the standard deviation in formulated by IEE (2009) in equation (2.7).

$$\sigma_r = \frac{L}{\sqrt{8}} \frac{\sqrt{N_{active} + N_{background} + N_{pseudo}}}{2 \cdot c_{mod} \cdot c_{demod} \cdot N_{active}} \tag{2.7}$$

In this equation, L is the maximum range (7.5 m for this camera). Paramters c_{mod} and c_{demod} are the modulation and demodulation coefficients respectively. The relation between number of photons and measured signal is given by equation (2.8).

$$I = c_{mod} \cdot c_{demod} \cdot N \tag{2.8}$$

This makes that one can write equation (2.7) in terms of measured signal, as equation (2.9).

$$\sigma_r = \frac{L}{2\sqrt{8}} \frac{\sqrt{I_{active} + I_{background} + I_{pseudo}}}{I_A \sqrt{c_{mod} \cdot c_{demod}}} \tag{2.9}$$

The modulation and demodulations constants and electrical noise figure I_{pseudo} are usually known for a specific camera. By measuring the amplitude I_{active} and the background illumination $I_{background}$, which both are available from a PMD sensor, the total standard deviation can be calculated.

Unfortunately, the camera that is used in this research, does not provide the background illumination or total intensity on the interface. This means that it is not possible to achieve an accurate estimate of the range noise based on background illumination. There is however some statistical data available for various levels of background illumination (as shown in figure 2.8). This data can be used to estimate the noise level based on amplitude only.

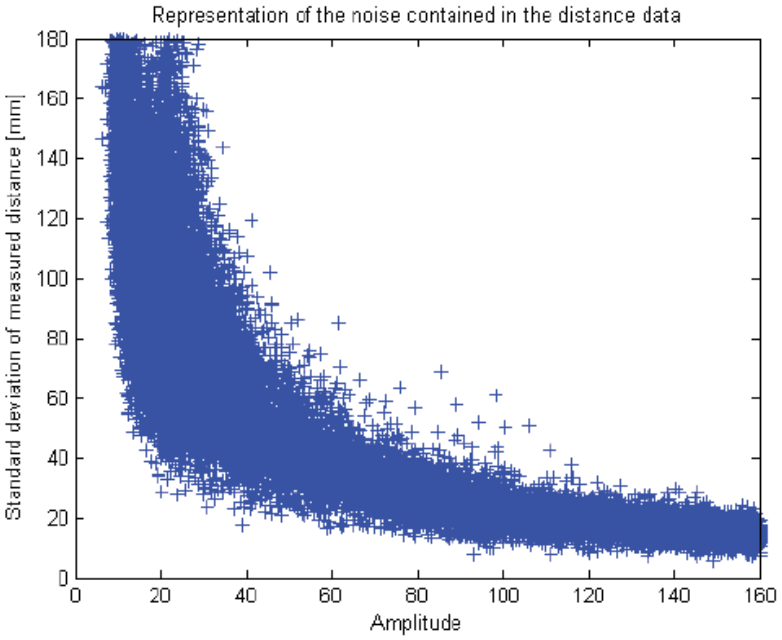


Figure 2.8: Statistical data on range noise for different levels of background illumination

Pixel Averaging Effect and Jump Edges

The ideal pinhole model assumes that the pixels are infinitesimally tiny and captures therefore the distance of an infinitesimally tiny surface. In reality, the pixels have a certain dimension and will therefore capture a part of a surface in the scene all together. By looking at the PMD principle (as described in section 2.3.4, it is easy to recognize that an averaging will take place of the surface distance. Because the surface is not bound to have a homogeneous reflectance, it becomes a weighed average and therefore unpredictable.

The effect is greatest where there are large jumps in the measured distance, for instance when one pixel observes both the edge of a near object and the background as is shown in 2.9. In literature, this is called a jump edge.

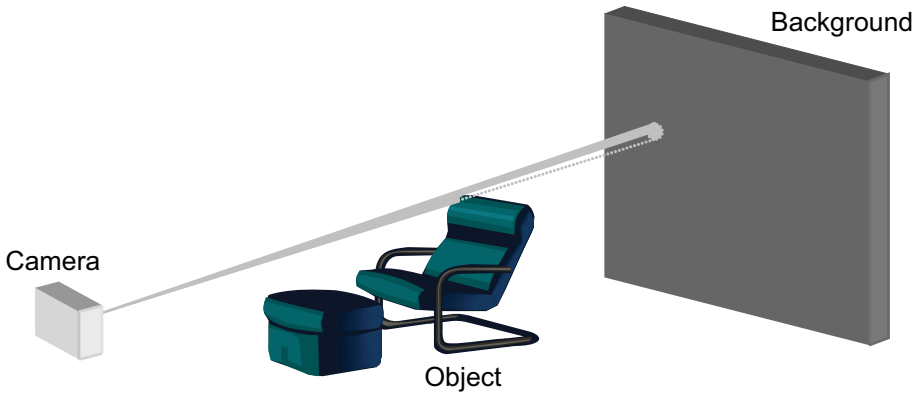


Figure 2.9: Example of a Jump Edge

Much of the distortion caused by this averaging effect can be accounted for, by detecting the pixels that are eligible for being on a jump edge and handling this information during further processing. A special filter has been developed by Fraunhofer, which is described in 3.2.3.

Aliasing

Aliasing occurs when the camera measures beyond its range of a phaseshift of 180°. For example, it cannot differentiate between a phase shift of 10°and 190°, and therefore might conclude that an object is nearer than it is in reality. In the newest time-of-flight cameras, this problem is circumvented by doing a multi spectral measurement. The range where aliasing occurs is different for each frequency. The difference in range measured for different frequencies can be used to reliably estimate the actual range. However, the camera that is used in this research is not able to detect aliasing.

Disturbances Induced by Measurement Principle

The measurement principle introduces other types of non-linear disturbances. These are mostly caused by reflection of the light, both in the scene and in the lens. Reflections on the scene cause surfaces to be illuminated, not only by the light source (direct light), but also by the light that is reflected off another surface (indirect light). This causes the measured distance to be larger, than it would if a surface was only illuminated directly.

The second type of disturbance is scattering of light in the lens of the camera. Because a lens operates with several boundaries with a difference in refraction index, strong light sources can induce large reflections in the lens itself. This causes parts of the to be illuminated by light that is meant for another part of the chip. Especially when objects are very near, while others are very distant, the strong light from near objects affects the measured distance to distant objects strongly. This type of noise is neglected in this thesis, by avoiding this situation in measurements.

If the incoming light gets so strong that it saturates the PMD (even with the shortest integration time), the distance cannot be accurately determined anymore. Pixels that are saturated, are marked by the camera as erroneous. These pixels can therefore be filtered out, before further processing of the data. Note that this, together with scattering in the lens, effectively blinds the camera if a very bright (often near) object is observed.

Motion Blur

Using the camera during movement (as will be the case if it is mounted on a vacuum cleaner) results in motion blur. Motion blur is a kind of disturbance that is caused by moving the camera while capturing. Because the sensor integrates the signal for a certain time period, a moving

sensor will cause each pixel to observe a trail of surfaces. Like the name implies, it generates a blurring effect. This means that detail in the depth information will be attenuated. The effect of motion blur increases with speed and is particularly heavy during rotation

In total, the motion blur of a pixel depends on the linear velocity and angular velocity of the robot, the angular velocity vector of the robot, the distance of the measured surface, the unit vector of the pixel and the integration time. If all these parameters are known or can be estimated, the trustworthyness of a pixel with respect to motion blur can be calculated.

A factor that worsens the effect of motion blur is the multiple integration time measurement of the PMD sensor (as described in section 2.3.4). Because the four sub-measurements all take a finite amount of time, the time of capture is different for each of these sub-measurements. Therefore, the observed trail of surface is also different. If exact timing information is available, a correction may be applied to the different sub-meausrements, based on a prediction of the motion.

In this research, motion blur is not taken into account. Further testing is therefore required to determine the effects of motion blur at high velocities.

2.4 Simultaneous Localization and Mapping

SLAM is the abbreviation of Simultaneous Localization and Mapping. It is a name for the group of algorithms that estimate the location of a robot with respect to the environment and the environment with respect to the robot at the same time.

The robot pose is usually determined by dead-reckoning. Because dead-reckoning uses relative displacements, the measurement error on these displacements makes the uncertainty in robot pose increase without bound. By acquiring absolute measurements with respect to the environment, this uncertainty can be bounded. However, the environment is initially unknown and is often observed relative to the already uncertain robot position. While the environment is observed and mapped during robot movement, it therefore suffers from the same uncertainty as robot pose does. Hence, the more accurate the robot pose becomes, the more accurate the structure of the environment becomes and vice verse. A stable SLAM algorithm makes the uncertainty in the environment mapping and the robot localization converge to the minimum achievable uncertainty.

SLAM algorithms are often based on Bayesian probability theory. Such algorithms ideally calculate the estimates, by using all of the knowledge that can be acquired by the system to come to the probability distribution that best fits the measurements. Often, the solution to the SLAM problem is more complex than is computationally or comprehensively achievable, and an approximation is therefore made to the Bayesian estimator.

2.4.1 Flavors of SLAM

There are many flavors of SLAM algorithms, using different kinds of input or handling and storing the information in different ways. The most popular SLAM algorithms are based on Extended Kalman Filters and Particle Filters. Both of these filters are discussed in depth in Sebastian Thrun (2001) and F. van der Heijden (2008). The corresponding flavors are:

- EKF SLAM
- FAST SLAM

EKF SLAM is an algorithm that represents the environment as a set of landmarks. The position of these landmarks is stored together with the robot pose in an expectancy vector and covariance matrix. These represent the estimates, uncertainties and mutual dependencies of both the robot position and the landmarks. After each movement and measurement, the expectancy and covariance of the robot position and the landmark positions are updated, to reflect the newly acquired knowledge. As more landmarks are observed, the expectancy vector

and covariance matrix grow accordingly. This is one of the downsides of EKF SLAM, as its computational complexity is N^2 where N is the number of landmarks.

FAST SLAM is an algorithm that is similar to EKF SLAM, but uses a particle filter to remove the dependencies between the landmarks. This means that the large extended Kalman filter, which is the heart of EKF SLAM, can be split up low dimensional extended Kalman filters for each landmark. This can make a FastSLAM algorithm much faster as the number of landmarks increase. The lowest achievable computational complexity is $O(M \cdot \log(N))$, where M is the amount of particles in the filter and N the amount of landmarks. The downside of FastSLAM is that it has the tendency to forget its past due to necessary resampling of the particle and become overly confident.

A different approach is provided by SLAM algorithms that instead of working with landmarks, work with entire 3D images. These algorithms use registration methods for aligning different measurements. The most popular registration algorithms are variants on Iterative Closest Point (ICP), as described in Censi (2008), S. Rusinkiewicz (2001) and D. Chetverikov (2002). These algorithms match points in two 3D point clouds by minimizing a certain metric that defines the distance between points. ICP is a computationally expensive procedure if many points are used, but has proven useful for registration of images where an accurately defined subset of points can be identified. Good results have been accomplished with higher resolution time-of-flight cameras using this method (S. May (2008)).

For the proof-of-principle in this research, the FastSLAM 1.0 algorithm is used as described in Sebastian Thrun (2001). The choice for this algorithm is motivated by the amount of landmarks that is generated during measurements. Furthermore, FastSLAM has interesting properties for recognizing landmarks. This will be discussed in the upcoming section.

2.4.2 FastSLAM

As was mentioned above, the FastSLAM algorithm is based on a particle filter. In EKF SLAM, the landmark and robot pose are estimated in one large Extended Kalman Filter. As all the landmarks are observed by the robot, and the robot position is uncertain, all the landmarks are linked through the uncertainty of the robot position. If the robot pose would be known exactly, all the landmarks become unlinked and can be estimated by their own low-dimensional Extended Kalman Filter. By Rao-Blackwellizing the EKF estimator, using a particle filter that represents the robot pose by a number of 'assumed to be true' robot poses, this can be accomplished. This means in effect that each particle in the particle filter maintains its own map of individual unlinked landmarks, with corresponding extended Kalman filters.

Algorithm Overview

The FastSLAM algorithm starts by applying a pose update to each particle in the particle filter, which represents the predicted motion of the robot and its uncertainty. This is implemented by adding a random generated vector to the particle pose. The motion model for this is based on the odometry or control input that is provided by the robot, which is covered in section 2.4.2. After this update, the particle distribution resembles the uncertainty of the pose of the robot.

The second step is to match all the measured landmarks with the known landmarks. Because each particle maintains its own map of landmarks, it is possible for each particle to make its own matches independently. The matching is done using a maximum likelihood estimator. If a landmark does not match, a new landmark is added to the particle map. If a landmark is matched, the weight of the particle is adjusted using the measured likelihood. This makes that particles that provide a better fit to the measurements, get a higher weight. For all the landmarks that are matched, the corresponding Kalman filter is updated using the measurement. Note that the a priori state estimate for the filters can be skipped, as landmarks are assumed not to move.

The last step is resampling the particles. This redistributes the particles according to the weights, to make the density of the particles match the probability density function that the particle filter is used to estimate. After resampling, all the information in the particle cloud is in the distribution of the particles and the weights can be reset to 1.

Particle Filter Pose Update

As seen earlier, the pose of the robot at sample time n is described by (2.10).

$$\vec{p}(n) = \begin{bmatrix} p_x(n) \\ p_y(n) \\ p_\alpha(n) \end{bmatrix} \tag{2.10}$$

When the robot moves, the pose vector changes. The design of the robot determines how this pose vector advances based on a specific control signal. This research assumes a two wheel robot with linear and angular velocity control, which yields the motion model described in equation (2.11). Parameters v and ω are the linear and angular velocities of the robot respectively, Δt is the time between updates and n the sample time.

$$\vec{p}(n) = \vec{p}(n-1) + \begin{bmatrix} -\frac{v(n)}{\omega(n)}\sin\big(p_\alpha(n-1)\big) + \frac{v(n)}{\omega(n)}\sin\big(p_\alpha(n-1)+\omega(n)\Delta t\big) \\ \frac{v(n)}{\omega(n)}\cos\big(p_\alpha(n-1)\big) - \frac{v(n)}{\omega(n)}\cos\big(p_\alpha(n-1)+\omega(n)\Delta t\big) \\ \omega(n)\Delta t + \gamma(n)\Delta t \end{bmatrix} \tag{2.11}$$

The linear and angular velocities are considered to be known with Gaussian uncertainty from the odometry or control signals of the vehicle. For the state update, each particle is now updated using the pose update vector, where v and ω are sampled from the corresponding Gaussian distributions. As the noise is only represented on both velocities, this model is restricted to a circular motion. Real motions are however not restricted to circular motions. Therefore, a third noise parameter γ is added to solve the degeneracy. It represents a change of orientation after the pose vector has obtained its new value and is sampled from a zero-mean Gaussian distribution.

Detection of old and new Landmarks

The detection of new landmarks is done on a per particle basis. For each particle i, the measurements are compared to the landmarks already detected, by means of a maximum likelihood estimator. Such an estimator is defined by equation (2.12), where l is a landmark on the map and $\vec{z}$ the measurement vector.

$$\widehat{l}(\vec{z}) = \underset{l}{\operatorname{argmax}}\, p(\vec{z}|l) \tag{2.12}$$

Each landmark is represented by an extended Kalman filter. Its uncertainty is represented by the estimated position μ_l and a corresponding covariance matrix $\mathbf{C}_l$. From this mean and covariance matrix, an estimated measurement vector and covariance matrix can be calculated $\vec{z}_{est,l}$ and $\mathbf{C}_{est,l}$. The measurement uncertainty is represented by $\mathbf{C}_z$, the total covariance becomes $\mathbf{C}_{m,l} = \mathbf{C}_{est,l} + \mathbf{C}_z$. This allows us to compute the probability density function $p(\vec{z}|l)$ for each measurement k, and estimate the best match using equation (2.13).

$$\widehat{l}_k = \underset{l}{\operatorname{argmax}} \frac{1}{\sqrt{|2\pi\mathbf{C}_m|}} e^{-\frac{1}{2}(\vec{z}-\vec{z}_{est,l})^T\mathbf{C}_{m,l}-1(\vec{z}-\vec{z}_{est,l})} \tag{2.13}$$

Apart from the likelihood of matching a landmark, there is also a situation where a landmark is actually observed for the first time and should not be matched. This can be modeled by putting

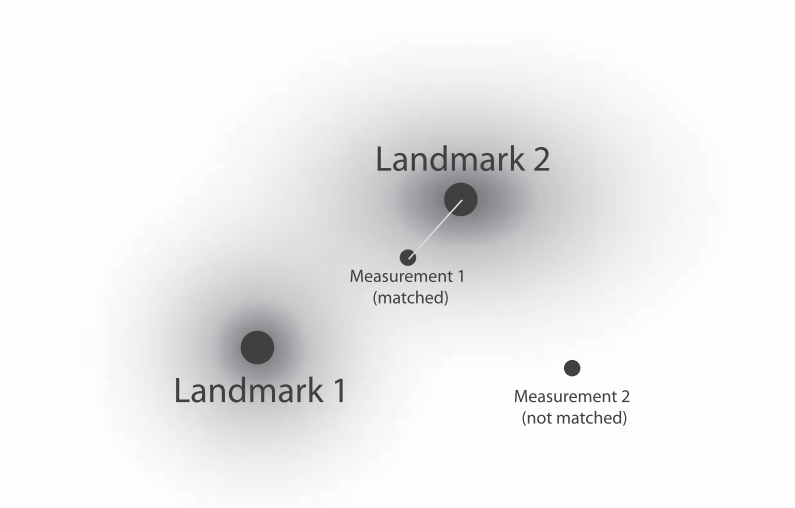


Figure 2.10: Two landmarks and two measurements

a lower threshold on the probability density function. If the probability density function $p(\vec{z}|l)$ is below a certain threshold for all l, a new landmark is created. In figure 2.10, an example is shown of two landmarks and two measurements. Measurement 1 falls within the likelihood thresholds, and is most likely to match with landmark 2. Measurement 2 is below the threshold and is not matched. This measurement will generate a new landmark.

Measurement Update

For landmarks that have not been sighted before, a new Kalman filter is created. This Kalman filter is initialized with the mean and covariance as computed from the mean and covariance of the measurement vector. For all the matched landmarks, a measurement update is done. As each landmark is represented by a Kalman filter, this is accomplished by doing a measurement update of the corresponding Kalman filter.

Particle Weighing

To represent the knowledge introduced by measurements, particles are weighed according to their likelihood to be the true position, based on these measurements. Because each measurement is matched using a maximum likelihood estimator, the weight w_n for measurement k can be extracted from the estimator as the likelihood using equation (2.14).

$$w_k = \max_l \frac{1}{\sqrt{|2\pi \mathbf{C}_m|}} e^{-\frac{1}{2}(\vec{z}-\vec{z}_{est,l})^T \mathbf{C}_{m,l}-1(\vec{z}-\vec{z}_{est,l})} \tag{2.14}$$

Note that this only provide information for measurements that are actually matched. Measurements that are not matched and create new landmarks have no attributed likelihood. The problem now arises that if one particle matches a landmark, while the other does not, the weights of both particles are not based on the same amount of landmarks. The solution that is used here for this problem is to set the weight of a non-matched measurement to the threshold used in the matching of the landmarks. This assures that a particle that has fewer matches gets a higher weight than a particle that has made more matches. It is important to prevent particles that create new landmarks often (which is a sign of an unlikely position estimate) to get precedence over particles that have more matches.

Multiple measurements are usually done during one iteration of the particle filter. The total weight w_i for a particle i, can be calculated as the product of the weights of all the measurement for particle i (w_k, i).

$$w_i = \prod w_k, i \tag{2.15}$$

Resampling

Particle filters require resampling, to keep the particle densities high enough in the areas that are most likely. The resampling process that is used here, is based on the random stratified resampling algorithm, as implemented by Tim Bailey. The algorithm makes use of N (the number of particles) equally sized bins, with a width $\Delta\rho$ that is equal to the cumulative weights of all the particles, divided by N.

$$\Delta\rho = \frac{\sum w_i}{N} \tag{2.16}$$

The bins are marked by the average value of the bin, which we will call ρ (where ρ_1 is the average value of the first bin). These bin positions are randomized by adding a sample from a uniform distribution, with a width equal to the bin width, effectively randomizing the bin widths between 0 and $\Delta\rho$. This is shown in equation (2.17), where ϵ_j is sampled from a uniform distribution in the domain $[-\frac{1}{2}\Delta\rho, -\frac{1}{2}\Delta\rho]$.

$$\rho_j = \Delta\rho \cdot (j - \frac{1}{2}) + \epsilon_j \tag{2.17}$$

Each particle is given a cumulative weight, which is the sum of all particles weights from the first particle up to the current particle i (equation (2.18)).

$$w_{cumulative,i} = \sum_{j=1}^{i} w_j \tag{2.18}$$

The resampling process starts with the first particle and the first bin. If the bin border is lower than the cumulative weight of the particle, the bin is 'filled' with a copy of the particle and the current bin number is increased. Otherwise, the current particle number is increased. The process continues until all bins have been filled with a particle. This makes that particles with larger weights, are attributed to a larger amount bins. The contents of all the bins now become the new particle set.

2.5 Features in the Time-of-Flight Data

In this section, the extraction of features from the data is discussed. Because a landmark based SLAM is used, features need to be extracted that can represent stable landmarks. It is important for features to possess certain qualities that make them useful for navigation purposes. Such qualities are:

- Easily and repeatably detectable
- Robust to changes in observer position
- Accurately positionable
- Distinguishable from other features
- Plentiful in the scene

The first two qualities make sure that features are detected often, and are likely to be detected in consequent frames. This makes it easy to relate features between different frames. Features

that are hard to detect, clutter the feature set and therefore use computation power, while not actually contributing to solving the navigation problem. An accurate position needs to be attributed to be able to navigate properly using this feature. Distinguishability is a quality that makes sure that features can be linked to features that were seen before. Finally, it is important for features to be plentiful in a scene. The more features there are, the more accurate can be navigated and the lower the risk that a scene does not contain any features, which lets the uncertainty of the position of the vehicle grow.

2.5.1 Choice of Features

If one looks at literature, several kinds of features are extracted from 3D data for navigation. A popular feature is the plane. Planes are robust features, because they are often described by many pixels in a 3D image. This makes planes easily detectable under various observer positions and distances. A plane has the property that it very accurately constrains the distance and orientation to the plane in the direction of its normal. However, for rotations around its normal and translations perpendicular to the normal it does not provide any information, as the plane does not constrain these transformations. The plane is therefore considered powerful, and robust in detection, but leaves some degrees of freedom.

An interesting way of looking at 3D data is proposed in Thirion (1996). This paper describes the definition of extreme points and lines in 3D data, which are called geometric invariants. Examples of these are lines of maximum curvature in a surface and intersections of three surfaces. These features are used for CAT and MRI scans to do a registration of multiple images. The strength of these features is that they are scale invariant, accurately defined in space and robust to observer positions. The detection of these features as presented in this paper is nonetheless mathematically complex and therefore not feasible for real time use. It is however possible to use features inspired on the this theory, as discussed in the upcoming sections.

Another type of feature that is considered very powerful for high resolution cameras is the SIFT or SURF key point as described in Lowe (1999) and Bay et al. (2006). These keypoints are commonly used for matching 2D camera images, because they can be robustly detected from different observer positions and can be extracted in real time. These features become extremely powerful for SLAM algorithms when a 3D position can be attributed to it, as is possible with a Time-of-Flight camera (J.J Wang (2009)). This however does require a higher resolution than the camera used in this research provides. A way to circumvent this problem is to use a separate 2D camera for key point extraction and map the 3D data of the time-of-flight camera in the coordinate space of the 2D camera. Combining data from two different cameras requires an accurate calibration and is a computationally intensive task. This method is therefore not explored in this research.

Based on the ideas discussed above and by looking at which kind of features can be found in a domestic environment in general, a selection of three types of features is made: vertical planes, vertical corners and jump edges. In figure 2.11, examples of these features are shown.

Figure 2.11: Three types of features from left to right: vertical planes, vertical corners and jump edges

In domestic environments, there are often many flat surfaces. Straight walls are good examples, but also furniture like chairs, cabinets and sofas often consist of one or more flat surfaces. If points on these surfaces are detected, a plane can be fitted through them. As navigation for autonomous vacuum cleaners is done in the 2D horizontal plane, horizontally oriented planes are of little interest and this research therefore focuses on vertically oriented planes.

A corner appears where two surfaces meet. If the boundary of the surfaces is upright, the corner is considered to be a vertical corner. Vertical corners are robust features, as they can be detected at multiple altitudes and are invariant to change in observer orientation and distance. A lot of objects in domestic environments have vertical corners.

If only one surface of a corner is observed, where the other is obscured, a jump edge is observed. As mentioned in section 2.3.6, a jump edge is defined as a sudden jump in depth, which occurs when the border of a near object is observed together with the background. Vertical jump edges have very similar properties to vertical corners. Both the vertical corners and the vertical jump edges are accurately can be attributed an accurate position in the horizontal plane.

2.6 Vertical Plane Extraction

To get an accurate view of how a plane is represented in a 3D image, a set of statements is put up for the points that define an ideal plane.

- The normal at the points on a plane is equal
- Plane points are constrained by the equation $ax+by+cz+d=0$ (in Cartesian coordinates)

These two statements hold true for all planes, and is at the base of many plane fitting algorithms.

2.6.1 Methods of Plane Fitting found in Literature

There are several ways of detecting vertical planes in 3D data. The method that is encountered most in literature is RANSAC (J.J Wang (2009), Mufti (2010)). RANSAC randomly selects a set of points and tries to do a linear least squares fit of a plane through these points. It then iteratively removes outliers from the used points, until a sufficient set of points is left that make up a plane, or the remaining set is too small and is discarded. By repeating this step several times, it becomes more likely that all planes will be found in the scene. The advantage of this method is that many implementations of this algorithm are readily available and it has proven itself in many applications. The disadvantages of this algorithm are that it does not take the geometric structure of the data in account and therefore is computationally intensive. Limiting the number of tries can be used to bind to computation time, but decreases the chance of finding all planes in the scene.

Another popular algorithm that is used often (especially for lines, the 2D dual of the plane) is Split and Merge (Viet Nguyen (2006)). It is designed to operate with structured data. This algorithm starts of with considering the whole scene as one big plane and dividing the plane into four subregions. These subregions are evaluated independently and are, if needed, subdivided themselves or merged with neighboring regions. After the algorithm is done, the whole scene is subdivided into planes of different sizes. By making use of the structure of the data, it can be executed significantly faster than RANSAC. It however, does divide the whole image into planes iteratively, even the regions that do not contain planes. This causes a considerable overhead.

The algorithm that is proposed in this research is a plane extraction method that is based on region growing. Region growing starts with a certain pixel and iteratively compares this pixel with its neighbors. The advantage of this algorithm is that it only has to compare each pixel with its neighbors only once, which makes it very fast in computation. Nonetheless, it is prone to erroneous estimation of surfaces that are curved slightly.

2.6.2 Plane Extraction by Region Growing

The region growing method consists of the following steps:

- Calculate normal for each pixel
- Start region growing from each pixel if pixel is not evaluated yet
- Select regions of sufficient weight
- Derive plane equation for each region
- Determine quality of the plane
- Convert planes to 2D representation

The normal of a pixel is defined as the normal of the triangle made up by the pixel together with the right and bottom neighboring pixel as shown in figure 2.12.

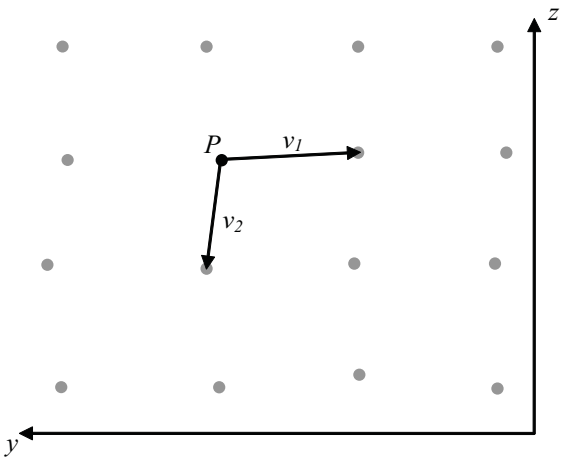


Figure 2.12: Vectors used for calculation of normal

The geometry of the data guarantees that the vectors between a pixel and its neighbors are not parallel and not equal to zero. It is therefore sufficient to calculate the normal by means of the cross product between the two vectors.

$$\mathbf{n} = \frac{\vec{v}_1 \times \vec{v}_2}{\|\vec{v}_1\| \, \|\vec{v}_2\|} \tag{2.19}$$

As soon as a region is started, the pixel is checked if it has been evaluated before. If this is not the case, it is compared to all unevaluated 4-neighbors. This comparison is based on the difference in normal with the neighboring pixel and the difference with the average normal of the region. A thresholding operation is performed for both these differences. The first threshold T_n prevents pixels that do not fit in the direct neighborhood to be incorporated in the plane and therefore suppressing outliers. The second threshold T_avg prevents regions that have a curvature from being recognized as a plane. Both thresholds are based on the inner product as defined in equation (2.20), with $\vec{n}_p$ the normal of the pixel to be evaluated, $\vec{n}_n$ the normal of the neighbor and $\vec{n}_{avg}$ the average normal.

$$\vec{n}_p \cdot \vec{n}_n \overset{?}{<} T_n \qquad \vec{n}_p \cdot \vec{n}_{avg} \overset{?}{<} T_{avg} \tag{2.20}$$

The entire scene has now been evaluated and is divided into regions. The regions are selected by another thresholding operation on the number of plane pixels (N) that the region consists of, also defined as the weight of the plane (see equation (2.21)). If the number of pixels is higher than threshold T_w, the plane is added to the selection.

$$N \overset{?}{>} T_w \tag{2.21}$$

For each selected plane, the coefficients of the plane equation are then computed by a least squares estimator. As the camera is looking forward, most of the error during the estimation process occurs in the x-direction (defined as forward). The system is approximated as only having error in this direction. This, together with the fact that due to the geometric structure of the data points, the plane is guaranteed to be overdefined by the measured points, makes it possible to define a linear least squares estimator by minimizing equation (2.22). The error is defined as $(x - \hat{x})$, with $x = c_1 y + c_2 z + c_3$ as the estimated x-coordinate and $\hat{x}$) the measured x-coordinate.

$$E = \sum \left(c_1 y_i + c_2 z_i + c_3 - \hat{x} \right)^2 \tag{2.22}$$

The minimum can be found at the point where the gradient of this equation equals zero as specified in equation (2.23).

$$\nabla E = 2 \sum \left(c_1 y_i + c_2 z_i + c_3 - x_i \right) \begin{bmatrix} y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.23}$$

Writing this equation in matrix form yields equation (2.24).

$$\begin{bmatrix} \sum {y_i}^2 & \sum z_i y_i & \sum y_i \\ \sum y_i z_i & \sum {z_i}^2 & \sum z_i \\ \sum y_i & \sum z_i & \sum 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} \sum y_i x_i \\ \sum z_i x_i \\ \sum x_i \end{bmatrix} \tag{2.24}$$

By using a method for solving the set of linear equations, for example Gauss-Jordan elimination, the coefficients can be derived. The computed coefficients can be transformed into appropriate coefficients in the plane equation (2.25), as shown. The choice of $a = 1$ in this equation is arbitrary.

$$ax + by + cz + d = 0 \quad \text{with} \quad a = 1,\ b = -c_1,\ c = -c_2,\ d = -c_3 \tag{2.25}$$

Using this equation, the quality of the planes can be determined by using the variance of the distance of the N points that the plane consists of, with respect to the fitted plane. By applying a thresholding operation on this variance, it is made certain that the quality of a plane is sufficient. This is shown in equation (2.26).

$$\frac{1}{N} \sum_{i=1}^{N} \left(\frac{a x_i + b y_i + c z_i + d}{\sqrt{a^2 + b^2 + c^2}} \right)^2 \overset{?}{<} T_q \tag{2.26}$$

2.6.3 Measurement Vector and Landmark Representation for SLAM

Before the extracted planes can be used in the SLAM algorithm, a suitable 2D representation of the vertical plane landmark has to be defined. Such a representation consists of both a measurement vector and a state vector. The state vector consists of the variables that the Extended Kalman Filter (as described in section 2.4.2) will use to define the plane landmark on the map. The measurement vector consists of the variables that are used to represent the measurement of the plane.

In a 2D top-view, a vertical plane ideally becomes a line of infinitesimal width. However, as real planes are not ideally vertical, such a line representation gains a width. The weighted average of this line can be approximated by a 2D linear least squares approximation that estimates the line through the projection of the points on the horizontal plane. Note that this is different from the least squares plane fit, as the z coordinates of the points are not used.

The least squares estimate is done using equations (2.27).

$$S_x = \sum x_i \qquad S_y = \sum y_i \tag{2.27a}$$
$$S_{xx} = \sum {x_i}^2 \quad S_{yy} = \sum {y_i}^2 \quad S_{xy} = \sum x_i y_i$$

$$a_l = \frac{NS_{xy} - S_y S_x}{NS_{yy} - S_y^2} \tag{2.27b}$$

$$b_l = \frac{1}{N} S_x - a_l \frac{1}{N} S_y \tag{2.27c}$$

$$s_\epsilon^2 = \frac{1}{N(N-2)} \left(NS_{xx} - S_x^2 - a_l^2 (NS_{yy} - S_y^2) \right) \tag{2.27d}$$

$$\sigma_a^2 = \frac{N s_\epsilon^2}{NS_{yy} - S_y^2} \tag{2.27e}$$

$$\sigma_b^2 = s_a^2 \frac{1}{N} S_{yy} \tag{2.27f}$$

(2.27g)

The acquired coefficients represent a line, by the relation in equation (2.28). The noise on these coefficients is characterized by the standard deviations σ_a and σ_b.

$$x = a_l y + b_l \tag{2.28}$$

Before the line is estimated using this method, first the means of the x and y-coordinates ($\overline{x}_l$ and $\overline{y}_l$) are subtracted from the coordinates. This makes that the line estimate is centered around the average of the plane points, which ensures that the noise on coefficients a_l and b_l is independent. This independence is important for estimation of the noise, which is discussed later on. Note that this also implies that the sums S_x and S_y are zero, which in turn implies that $b_l = 0$.

One of the line representations that is suitable for SLAM is the Hessian normal representation, which is defined as in equation (2.29).

$$x \cos\phi + y \sin\phi = r \tag{2.29}$$

This form contains two parameters: the shortest distance to the origin r and the direction of the normal expressed as an angle ϕ, which yields a state vector as shown in (2.30).

$$\vec{\mu} = \begin{bmatrix} \mu_r \\ \mu_\phi \end{bmatrix} \tag{2.30}$$

The measurement vector (2.31) is defined with the same parameters as the state vector: the distance to the origin and the angle in the (expressed in the map frame $[m]$). Note that this choice causes the measurement Jacobians in the Kalman Filters to become equal to the identity matrix. This choice yields a performance increase during the matching procedure (described in section 2.4.2), as the estimated measurement covariance is equal to the estimated state covariance.

$$\vec{z} = \begin{bmatrix} r^{[m]} \\ \phi^{[m]} \end{bmatrix} \tag{2.31}$$

The two parameters of the measurement vector can be computed from the line estimation. The angle $\phi^{[c]}$ (defined in the camera frame) is related to the a_l coefficient, as defined in equation (2.32). Note that due to the choice of frame, a positive a_l means a negative angle, which explains the minus sign.

$$\phi^{[c]} = -\text{atan}(a_l) \tag{2.32}$$

The distance can be computed from a_l, b_l, $\overline{x}_l$ and $\overline{y}_l$, using equation (2.33). Note that b_l is kept in the equation, even though it is equal to zero. This is done to be able to use the equation later on for the estimation of the covariance matrix of the measurement vector.

$$r^{[c]} = (\overline{x}_l + b_l)\cos(\phi^{[c]}) + \overline{y}_l \sin(\phi^{[c]}) \tag{2.33}$$

As all measurements are taken relative to the camera and not the world frame, they have to be transformed to fit this measurement vector. Because the pose is defined by the particle in the SLAM algorithm, this step is fairly trivial. First the measured angle $\phi^{[c]}$ is transformed using the orientation of the pose p_α , by simple addition.

$$\phi^{[m]} = \phi^{[c]} + p_\alpha \tag{2.34}$$

The measured distance needs to be computed relative to the origin, instead of relative to the position of the pose. This can be done by adding the distance component of the pose to the origin in the direction of the line normal. This normal can be derived from the angle $\phi^{[m]}$.

$$\vec{n} = \begin{bmatrix} \cos(\phi^{[m]}) \\ \sin(\phi^{[m]}) \end{bmatrix} \tag{2.35}$$

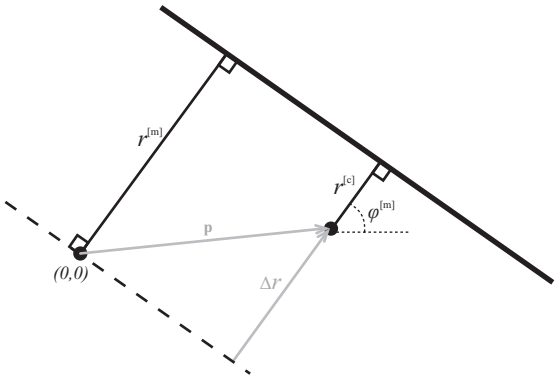


Figure 2.13: Computing the measurement vector from measurements

The dot product of the vector to the position of the pose and this normal equals the distance difference that needs to be added for transforming the distance to the world frame (see figure 2.13). The resulting distance is computed in (2.36).

$$r^{[m]} = r^{[c]} + \Delta r = r^{[c]} + \vec{n} \cdot \begin{bmatrix} p_x \\ p_y \end{bmatrix} = r^{[c]} + p_x \cos(\phi^{[c]} + p_\alpha) + p_y \sin(\phi^{[c]} + p_\alpha) \tag{2.36}$$

Errors in Plane Extraction

The plane extraction algorithm introduces an uncertainty on both the angle and the distance. For a successful implementation of SLAM, it is important that this uncertainty is accurately estimated. The method that is proposed here, is based on the sample variance of the different

pixels. This is justified when the number of pixels that make up a plane is large enough, which is assumed to be the case.

The estimated variance is computed in the linear least squares estimation as σ_a^2 and σ_b^2 (see equation (2.27)). This yields the line covariance matrix in equation (2.37).

$$\mathbf{C}_l = \begin{bmatrix} \sigma_a^2 & 0 \\ 0 & \sigma_b^2 \end{bmatrix} \tag{2.37}$$

As the variances on both coefficients are expected to be small, this covariance matrix can be transformed to the covariance on $r^{[c]}$ and $\phi^{[c]}$, by using the linearization of equations (2.32) and (2.33), with respect to coefficients a_l and b_l. This linearization yields the Jacobian in equation (2.38).

$$\mathbf{J}_{l,c} = \begin{bmatrix} -\frac{1}{a_l^2+1}\left((\overline{x}_l + b_l)\cos\phi^{[c]} - \overline{y}_l \sin\phi^{[c]}\right) & \cos\phi^{[c]} \\ -\frac{1}{a_l^2+1} & 0 \end{bmatrix} \tag{2.38}$$

The noise on $r^{[c]}$ and $\phi^{[c]}$ is now defined by the covariance matrix in equation (2.39).

$$\mathbf{C}_c = \mathbf{J}_{l,c}\mathbf{C}_l\mathbf{J}_{l,c}^T \tag{2.39}$$

This covariance matrix is defined in the camera frame. To translate this matrix to the world frame, a second linearization is made of equations (2.34) and (2.36) with respect to $r^{[c]}$ and $\phi^{[c]}$. This yields the Jacobian in equation (2.40).

$$\mathbf{J}_{c,z} = \begin{bmatrix} 1 & -p_x \sin\phi^m + p_y \cos\phi^m \\ 0 & 1 \end{bmatrix} \tag{2.40}$$

By using the chain rule, the total transformation becomes as described in equation (2.41).

$$\mathbf{C}_z = \mathbf{J}_{c,z}\mathbf{C}_c\mathbf{J}_{c,z}^T = \left(\mathbf{J}_{c,z}\mathbf{J}_{l,c}\right)\mathbf{C}_l\left(\mathbf{J}_{l,c}\mathbf{J}_{c,z}\right)^T \tag{2.41}$$

Note that despite the identity measurement Jacobian in the Kalman Filter, the Extended Kalman Filter does not become a linear Kalman Filter, as transforming the covariance matrix of the line coefficients to the measurement covariance matrix still involves a linearization. This linearization is just not done in the Kalman Filter itself.

From experiment A.4, it has become apparent that the noise model described above is not sufficient to cover all the noise in the plane estimation. This is due to disturbances in the depth estimation of the camera that change with observer positions. To accomodate for this, additional noise is added on $r^{[c]}$ and $\phi^{[c]}$. This replaces covariance matrix $\mathbf{C}_c$ with a new covariance matrix $\mathbf{C}_c^*$, which is defined in equation (2.42)

$$\mathbf{C}_c^* = \mathbf{C}_c + \begin{bmatrix} \sigma_{r,add}^2 & 0 \\ 0 & \sigma_{\phi,add}^2 \end{bmatrix} \tag{2.42}$$

The corresponding measurement covariance is defined in equation (2.43).

$$\mathbf{C}_z = \mathbf{J}_{c,z}\mathbf{C}_c^*\mathbf{J}_{c,z}^T \tag{2.43}$$

2.7 Vertical Corner Extraction

In this section, the extraction of vertical corners is analyzed. Just as with the planes, statements can be defined that are valid for the points that make up a an ideal vertical corner. Vertical corner points:

- Mark a discontinuity in the horizontal component of the surface normal
- Share a common projection on the horizontal plane
- Can be attributed a direction, based on normals on both sides

2.7.1 Method of Detection

Based on the properties above, the first step is the detection of a normal discontinuity at a certain point. This requires evaluation of the region around this point. Because we restrict ourselves to vertical corners, only the normal in horizontal direction is of interest.

For determining the normals from the pixel data, we look at the neighboring pixels. If it is assumed that the neighboring pixels on the left and the right share the same z-coordinate, determining the normals becomes a 1D operation along a pixel line. Note that this is an approximation, because the lens introduces radial distortion as shown in section 2.3.6.

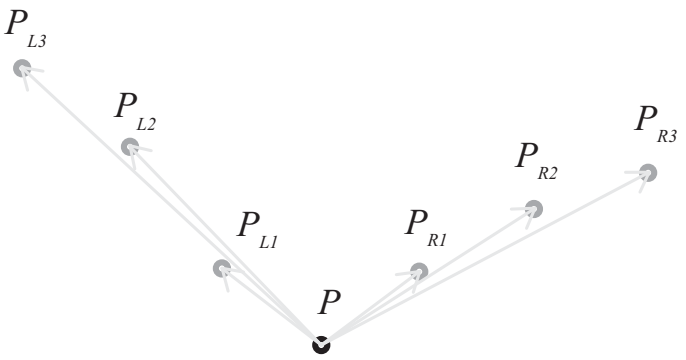


Figure 2.14: Top-view of a pixel row around the pixel of interest

A top-view of such a pixel line is shown in figure 2.14. The center pixel P is the pixel that is suspected to be a corner. By determining the normalized difference vectors of this pixel with left and right neighbors and comparing these vectors, the amount of discontinuity can be determined. By repeating this procedure with second neighbors, third neighbors and so on, the robustness of this presumed corner pixel is tested on different scales. The two different vectors are given in equation (2.44), where n is the scale number.

$$\vec{v}_{diff,Ln} = \frac{\vec{P}_{Ln} - \vec{P}}{\left\|\vec{P}_{Ln}\right\| \left\|\vec{P}\right\|} \qquad \vec{v}_{diff,Rn} = \frac{\vec{P}_{Rn} - \vec{P}}{\left\|\vec{P}_{Rn}\right\| \left\|\vec{P}\right\|} \tag{2.44}$$

For comparing the two difference vectors, the dot product of these two vectors is computed. This yields a value between -1 and 1, with higher values meaning a higher likelihood of the pixel being a corner. By using a threshold T_p on this dot product, a choice can be made if the pixel represents a corner at this scale. The thresholding equation is shown in (2.45), where n can be replaced by the scale.

$$\vec{v}_{diff,Ln} \cdot \vec{v}_{diff,Rn} \overset{?}{>} T_d \tag{2.45}$$

If the threshold is exceeded at enough scales, the pixel can be considered robust and eligible to be part of a vertical corner. To determine the direction of the corner, the measured difference vectors at different scales are normalized and averaged on each side separately (left and right of

the center pixel). This yields a vector representing the average direction on both sides (equation (2.46)).

$$\vec{v}_{dir,l} = \sum_{n=1}^{N} \frac{\vec{v}_{diff,Ln}}{\left\|\vec{v}_{diff,Ln}\right\|} \qquad \vec{v}_{dir,r} = \sum_{n=1}^{N} \frac{\vec{v}_{diff,Rn}}{\left\|\vec{v}_{diff,Rn}\right\|} \tag{2.46}$$

By normalizing and summing the resulting left and right vectors, a vector is acquired that points in the direction of the corner (equation (2.47)). Using the four quadrant arc tangent, this vector can be converted into an angle (equation (2.48)).

$$\vec{v}_{dir} = \frac{\vec{v}_{dir,l}}{\left\|\vec{v}_{dir,l}\right\|} + \frac{\vec{v}_{dir,r}}{\left\|\vec{v}_{dir,r}\right\|} \tag{2.47}$$

$$c_\alpha = \arctan\left(\vec{v}_{dir}\right) \tag{2.48}$$

A robust vertical corner consists of several pixels that mark the same corner. In contrast to the ideal vertical corner, these points do not have the exact same projection on the horizontal plane. Some kind of decision process is needed, that determines which corner pixels belong to which corner feature. The decision process that is proposed here, has the properties of the region growing algorithm. Each newly detected corner pixel is compared to the already known corner features. If it matches, it is added to the feature, if not, it starts its own feature.

This comparing process is based on a simple classification using two thresholds. One threshold T_d is put on the difference in distance of the projection of the pixel and the projection of the feature on the horizontal plane. Another threshold T_α is put on the difference in direction of the pixel and the feature. If both differences fall within limits, the pixel is added to the feature. The position and angle of the feature is computed as the average values of all the pixels that are participating in the feature. Note that this is a simple classifier. If more extensive data is known about corner geometries, a likelihood estimator might enhance performance. Also note that the first match is chosen, instead of the best match. This has proven to be sufficient and speeds up the algorithm.

The average position together with the average angle makes up the corner feature vector (defined in the camera frame [c]), as shown in equation (2.49).

$$\vec{c}^{[c]} = \begin{bmatrix} x^{[c]} \\ y^{[c]} \\ \alpha^{[c]} \end{bmatrix} \tag{2.49}$$

2.7.2 Measurement Vector and Landmark Representation for SLAM

The corner can be represented as a position in Cartesian coordinates. This yields the state vector in equation (2.50). Note that in the SLAM implementation defined here, the orientation of the corner is not yet taken into account. Adding this orientation, provides additional information for matching features.

$$\vec{\mu} = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \tag{2.50}$$

As with the plane features, the corner features are measured relative to the camera position. Because the robot pose is "known" from the particle filter, one can transform the corner feature

to the world frame by doing a rotation and a translation. The rotation can be specified by a rotation matrix with the robot orientation as a parameter (see equation (2.51)).

$$\mathbf{R} = \begin{bmatrix} \cos p_\alpha & -\sin p_\alpha \\ \sin p_\alpha & \cos p_\alpha \end{bmatrix} \tag{2.51}$$

After the rotation is completed, the translation is done by adding the robot position. This yields the measurement vector in equation (2.52), where $x^{[m]}$ and $y^{[m]}$ are the measured corner coordinates in the map frame.

$$\vec{z} = \begin{bmatrix} x^{[m]} \\ y^{[m]} \end{bmatrix} = \mathbf{R} \begin{bmatrix} x^{[c]} \\ y^{[c]} \end{bmatrix} + \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{2.52}$$

2.7.3 Estimation of the Error in Vertical Corner Features

The error in position of the vertical corner feature depends on many factors. Some of these factors are due to measurement errors, others are due to the geometry of the observed corner. The first kind of errors are relatively easy to deal with, as they can be computed from camera models. The second kind is unpredictable as information about the geometry is reduced by the sampling of the camera and the corner feature extraction algorithm.

- The depth estimation error
- Quantization errors by the pixel distribution
- Irrregularities in geometry
- Verticalness of the corner

Where the depth estimation errors and quantization errors can be estimated using the information from the camera, there is no way of determining the error introduced by geometry. As there are many error sources that cannot be measured, the choice is made to compute the covariance matrix using the sample variance. The advantage of this method is that geometrical differences are part of this variance estimate. The drawback is that the smaller the amount of points, the less accurate it becomes. Another drawback is that it assumes that the corner is perfectly vertical, which comes in to play when a slanted corner is observed at different altitudes. This effect is shown in figure 2.15.

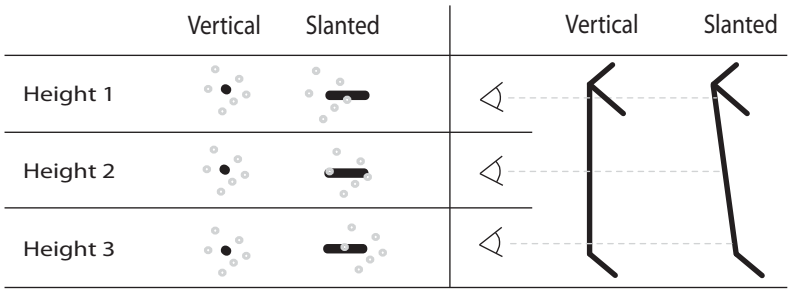


Figure 2.15: Vertical and slightly slanted corner observed at different heights

As the observed geometry can have any shape, there is not an optimal solutions for every encountered corner. It might be possible to do a guess of the slanting error, by using a fitted 3D

line through the corner points. From this line representation, a unit vector can be created parallel to the line. This is however not yet applied in this research.

The sample variance can be calculated using equation (2.53). In this equation $\vec{z}_i$ is one of the region points and $\vec{\bar{z}}$ is the sample mean of the measurement vectors.

$$\mathbf{C}_{sample} = \frac{1}{N}\sum(\vec{z}_i - \vec{\bar{z}})^T(\vec{z}_i - \vec{\bar{z}}) \tag{2.53}$$

As each pixel is considered independent, the error of a measurement consisting of N pixels is given by equation (2.54).

$$\mathbf{C}_c = \frac{1}{N}\mathbf{C}_{sample} \tag{2.54}$$

The final step is rotating this covariance matrix defined in the camera frame to the map frame, using matrix **R** (defined in equation (2.51)). This is done using equation (2.55).

$$\mathbf{C}_z = \mathbf{R}\mathbf{C}_c\mathbf{R}^T \tag{2.55}$$

Because the measurement vector is in the same coordinate system as the state vector and consists of the same parameters, the measurement Jacobian is again equal to the identity matrix. Note that as **R** represents a pure rotation, the Extended Kalman Filter becomes a linear Kalman Filter.

In experiment A.5, it is shown that this method of error estimation generally underestimates the error. Three likely causes have been pointed out:

- Averaging effect, as pixels measure a surface instead of a point
- Quantization noise
- Errors due to reflections in the scene

A crude approximation to the error is made by an additional error with a standard deviation that scales linearly with distance. The variance of this additional noise is defined by equation (2.56). In this equation, $|\vec{z}|$ is the distance to the feature and γ is the standard deviation of the noise per unit distance.

$$\sigma^2_{add} = \gamma^2 \|\vec{z}\|^2 \tag{2.56}$$

The corrected covariance matrix $\mathbf{C}^*_c$ is shown in equation (2.57), where $\mathbf{C}_c$ is the original covariance matrix. In the experiment, $\gamma = 0.02$ [m/m] is determined to be appropriate. This is about equal to the space between the points, scaled with distance (see figure 2.7).

$$\mathbf{C}^*_c = \mathbf{C}_c + \begin{bmatrix} \sigma^2_{add} & 0 \\ 0 & \sigma^2_{add} \end{bmatrix} \tag{2.57}$$

The corrected measusrement covariance is defined in equation (2.58).

$$\mathbf{C}^*_z = \mathbf{R}\mathbf{C}^*_c\mathbf{R}^T \tag{2.58}$$

2.8 Vertical Jump Edge Extraction

Analogous to the corners and the planes, also for the jump edges a set of statements is defined:

- Jump edge points mark a discontinuity in depth in the horizontal direction
- Jump edge points share a common projection on the horizontal plane
- One side of the jump edge is foreground, the other side background

- Can be attributed a direction, based on the foreground normal
- The line between the foreground and background points of an ideal jump edge is parallel to the line of sight

2.8.1 Method of Detection

Detecting a vertical jump edge is, just as with the vertical corner, an operation that only uses pixels on the same row. One of the possibilities for detecting a jump edge, is by using the property of the line between two neighboring jump edge pixels being parallel to the line of sight. At the Fraunhofer Institute of Technology (S. May (2008)), a special filter has been developed that exploits this characteristic by putting a threshold on the amount of parallelity. A graphical representation is shown in figure 2.16.

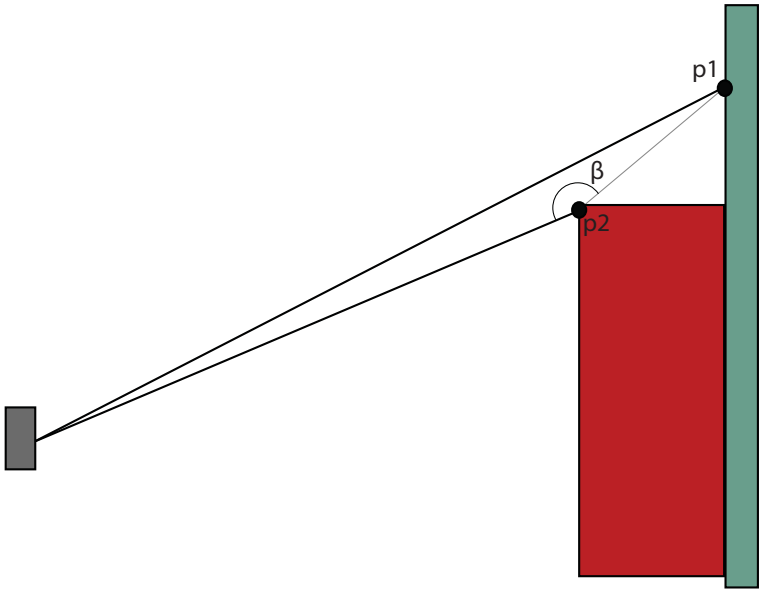


Figure 2.16: Graphical representation jump edge detection

The two pixels p_1 and p_2 are compared. For both pixels, the lines of sight have been constructed and the thin line that connects the two pixels is a linear approximate of the surface. Angle β is now a measure for the 'jump-edge-ness' of the pixels. If this angle is greater than a certain threshold, the pixels cannot be trusted and are marked as jump edge pixels.

If the origin of the camera is defined to be at location $o = (0,0)$, both viewing lines can be represented by the vectors:

$$\mathbf{p}_1 = \begin{pmatrix} p_{1,x} \\ p_{1,y} \end{pmatrix} \text{ and } \mathbf{p}_2 = \begin{pmatrix} p_{2,x} \\ p_{2,y} \end{pmatrix} \tag{2.59}$$

Angle β can now be defined as (2.60)

$$\beta = \pi - \arccos \left| \frac{\mathbf{p}_2 \cdot (\mathbf{p}_1 - \mathbf{p}_2)}{\|\mathbf{p}_2\| \cdot \|\mathbf{p}_1 - \mathbf{p}_2\|} \right| \tag{2.60}$$

As jump edges consist of several neighboring pixels, which differ in position, a choice has to be made which pixel is the most stable one to represent the jump edge. If jump edges are observed under different angles, the background continuously changes, while the foreground remains the same. The best pixel to represent the actual jump edge is the foreground pixel. Based on this foreground, a direction can be attributed, which is chosen equal to the normalized difference vector of the jump edge pixel and its neighbor.

After the foreground jump edge pixels have been identified, they are clustered in the same way as with the vertical corner features. As the jump edges features are very similar to corners, the landmark representations for both the measurement vector and the state vector are kept the same. This holds also true for the noise vector. This is described in section 2.7.

3 Design and Implementation

To be able to test and develop the different feature extraction algorithms, a processing pipeline has been created that facilitates in receiving the data from the camera, processing the data and providing feedback to the user. This chapter provides a global overview of this pipeline, followed by a description of the implemented feature extraction algorithms.

3.1 System Overview

The whole processing system can be divided in three separate blocks:

- Data Acquisition
- Processing Algorithms
- Visualization and Control

The structure of the system is shown figure 3.1. As can be seen, all the blocks are separated by a network connection, which makes it possible to run each block on different systems. This is particularly useful for moving parts of the design to a robot and others to a separate system, depending on the needs.

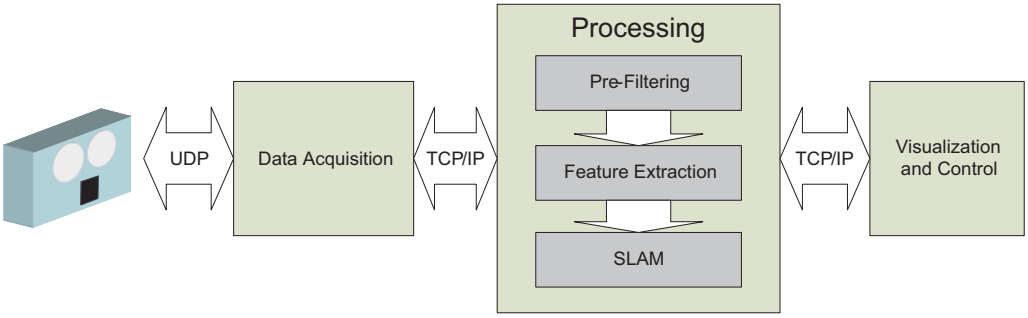


Figure 3.1: Block diagram of system design

The data acquisition block is responsible for communicating with the camera. In essence, it retrieves the captured frames from the device, reformats them to a processable data format and outputs the frames to the TCP/IP server. The next step is the processing. This block can be divided into three subsystems, one that polishes the data to make it ready for feature extraction (pre-filtering), one that actually applies the feature extraction algorithms and the SLAM implementation. The last block Visualization and Control makes the data available to the user in a way that gives insight in the performance of the algorithms. It also allows the user to have basic control of the processing process.

3.1.1 Data Acquisition

The data acquisition has been designed as a stand-alone C++ (Qt) application with the following key-functionalities:

- Communication with the time-of-flight camera
- Preview the 3D data for evaluation
- Capture single frames of data
- Capture a continuous stream of data
- Provide limited editing capabilities to correct dataset
- Save the data to a file
- Make the data available on a TCP server

The camera provides its data using the UDP protocol over an Ethernet connection. The protocol for requesting the data from the camera is a proprietary protocol and therefore not docu-

mented further here. These are however available in IEE (2010a) and IEE (2010d). A library has been provided by the manufacturer that exports an API that can be used on Windows. However, the choice has been made to implement the protocol directly, to make it compatible with other operation systems for development on an embedded platform.

As the data comes in, it can be previewed using an OpenGL interface. The interface provides a first quality check before the data is actually captured. It contains rotating, translating and zooming functionality to get a good impression of what is looked at. Before display, a rough filtering step is done to clean up the data, by removing pixels that have a large difference in depth with respect to its neighbors (the threshold is set at 15 cm). This makes the image more clear at the expense of loosing some detail. Note that this filtering step is only applied to the displayed data and does not influence the dataset.

The snapshot function can be used to capture a single frame. This frame is added to the dataset already in memory. By moving the camera and taking snapshots, this function has been used extensively to capture static datasets that suggest the notion of motion. Apart from taking snapshots, it is also possible to capture the continuous stream of frames that comes out of the camera at a rate of 5 Hz. Like with the snapshots, the captured frames are added to the dataset and stored in memory.

After the capturing is done, each frame can be viewed and inspected more closely in the OpenGL viewer. If a frame is not useful or erroneous for some reason, it can be removed using a delete function. This provides a versatile way of capturing data. Once satisfied with the dataset, it can be saved to disk as a CSV file or as raw data. It is also possible to load a dataset stored earlier for reevaluation and editing.

Finally, for real-time processing purposes, the application can act as a TCP/IP server that provides the formatted data for further processing.

3.1.2 Processing Algorithms

The processing algorithms are combined in a single C++ application. It provides the following functionalities:

- Read raw data from a file or a TCP/IP server
- Pre-process the data to make it ready for feature extraction
- Extract features from the data
- Process the features in the SLAM algorithm
- Build a visualization of the extracted features and the SLAM map
- Write visualization data to a TCP socket of the visualization application

The data is read from a file or the TCP/IP server and brought into the processing algorithm. The frame rate is fixed at 5 Hz, which is controlled in hardware by the camera. The processing module processes the frames as they come in.

When a frame comes in, it is first lead through a couple of pre-filtering algorithms. These filters remove pixels that are unusable or erroneous from the data and do some additional preprocessing steps like smoothing. All of these steps are discussed in detail in section 3.2. After the pre-processing, the data is fed to several feature extraction algorithms. These algorithms extract 2D features which resemble points of interest that can be used for Simultaneous Localization and Mapping (SLAM). The process of feature extraction is discussed extensively in chapter 2.

All the data that is returned by the feature extraction algorithms is combined into a 2D map and a camera amplitude image. Both of these are sent the visualization and control application through a TCP/IP connection, if available. Otherwise no output is given. As the processing is separated in a stand-alone module, it can easily be adapted for use in another software system.

3.1.3 Visualization and Control

The visualization and control block is also covered by a stand-alone application. This application acts as a server, which the processing block can connect to. As soon as the connection is made, this application can send commands to the processing unit and the processing unit can send data to be visualized.

The visualization consists of three images:

- The amplitude image of the camera with the feature pixels marked with a color (green for planes, red for corners and blue for jump edges)
- The local map (top-view) of the extracted features in the camera image
- The global map (top-view) of the SLAM algorithm

The commands that can be sent from the processing algorithm are:

- Process a file
- Stream live data from the camera
- Process one frame at a time or process free running

3.2 Pre-filtering

Before features can be extracted, the data first needs to be prepared. One of the first processing steps is to correct the data for the structural noise as determined in experiment A.2. Also, the raw 3D pixel grid that the camera provides contains a lot of pixels that do not provide useful information for one or more of the feature extraction methods.

These include:

- Erroneous pixels (marked by the camera)
- Pixels that are an alias (distance larger than 7.5 m)
- Jump edges

3.2.1 Measured Distance Correction

In the experiment in section A.2, it was determined that the measured distance differs from the real distance by a multiplication factor that is different for each per pixel. To correct this, the data from the experiment was converted into a factor for each pixel. The correction algorithm applies this factor to each pixel. The complexity is $O(N)$, with N the number of pixels.

3.2.2 Pixel Error Filter

The camera is capable of identifying pixels with an amplitude that is either too low or too high for an accurate measurement. This information is sent out by the camera as a per-pixel error field. The pixel error filter is therefore a very simple filter, that checks each pixel that comes out of the camera. If it is marked as being erroneous and translates the camera flag to an application defined flag. The complexity is $O(N)$, with N the number of pixels.

3.2.3 Jump-Edge Filter

As jump edges are regions where the depth is uncertain, they provide no value for the feature algorithms, other than the jump edge extractor. Therefore, the jump edge detection is part of the pre-filtering so that the jump edges can be filtered out before further processing. The detection is implemented as described in the analysis in section. A flow diagram of this is shown in figure 3.2.

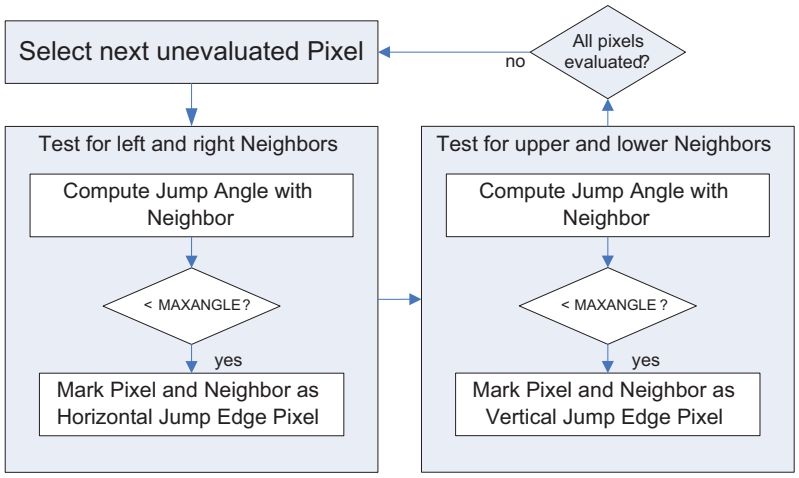


Figure 3.2: Flow Diagram of Jump Edge Detection

Each pixel is compared with its 4-neighbors (with the boundary pixels being exceptions as they do not have four neighbors). This makes the algorithm of the order $O(N)$ with N the number of pixels. Note that a difference is made between jump edges in horizontal direction and in vertical direction, as the horizontal jump edge pixels are used in the jump edge extraction algorithm.

3.2.4 Low-Pass Filter

For plane detection, the noise that is found in the whole image disturbs the process of matching the normals of neighboring pixels. A lowpass filter is used to remove this noise. The lowpass filter is implemented as a non-causal 2D convolution filter using a Gaussian kernel (3x3) for all coordinates (x, y, z) separately on the pixel grid in y and z direction. The Gaussian filter is made with a standard deviation of 1 pixel in both y and z directions.

In this process, the non-linearity of the pixel distribution is neglected and all the pixels are treated as being equally spaced in both directions. This is justified by the fact that the filter has the tendency to make noisy planes more 'plane like', as long as the y-coordinates are monotonically increasing in the y-direction of the 3D point array and the z-coordinates are monotonically increasing in the z-direction of the 3D point array. This is guaranteed by the pinhole model as described in section 2.3.5. For a successful plane detection, the low-pass filter is applied twice. The complexity of the algorithm is $O(N)$, with N the number of pixels.

3.2.5 Aliasing Filter

The camera does not contain special hardware to detect aliasing. This means that an alternate way of detecting aliasing needs to be applied. A very simple aliasing filter can however be constructed based on amplitude. As the amplitude of the incoming light attenuates proportional to the squared distance to the object that is measured, very low amplitude measurements are likely to be coming from an alias. By putting a threshold on the amplitude of pixel i, aliases can be filtered out (equation (3.1)). Note that because low amplitude also corresponds to high noise, such a filter is also beneficial to filter out noisy regions.

$$I_{A,i} \overset{?}{<} A_{thres} \tag{3.1}$$

This algorithm is executed for each pixel and therefore has a complexity of $O(N)$.

3.3 Feature Extraction

In this section, the implementation of the different feature extraction algorithms is shown. These implementations are based on the analysis of the different methods in chapter 2. For each algorithm, the complexity is calculated in the term of pixels.

3.3.1 Plane Extraction

In figure 3.3, the flow diagram of the plane extraction algorithm is shown. This algorithm is an implementation of the region growing algorithm as described in section 2.6.

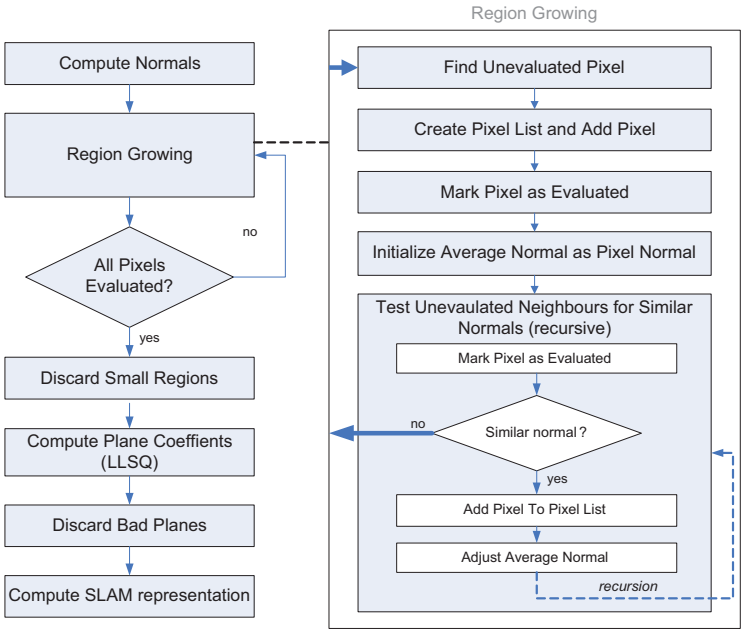


Figure 3.3: Flowdiagram of Plane Extraction

The algorithm starts with initializing a point array (of equal size as the pixel grid) using the low-pass filtered 3D data and determining the normals of all valid (non-erroneous) points. Once the normals have been determined, region growing is started. This is a recursive operation as described in section 3.3.

The first step in the region growing is finding the first valid and unevaluated pixel. This is done using a naive implementation, which checks all pixels one after the other. If such a pixel is found, region growing is started by creating a list which will hold all the points for this region. This is implemented as a linked list, where each point holds a reference to the next point in the list. The first points is added to the list, marked as evaluated and an average normal is set for the region. Now, the normals of the four unevaluated neighboring points are compared to the normal of the current point and the average normal of the region. If a neighbor matches, it is added to the list, marked again as evaluated and the plane average is recomputed. Now, the neighbor recursively starts testing its neighbors using the same procedure. In this way, the region is grown, until it can grow no further. This procedure is repeated, until all pixels belong to a region.

For each region that has enough weight (contains at least M pixels, with M the minimum plane weight), plane coefficients are determined using linear least squares. After having determined the plane coefficients, the test is done to discard the bad planes, which are not vertical enough, or have a large standard deviation in the distance of the points to the plane. This is implemented as an iteration over all pixels. Computation of the SLAM representation is left and computed by linear least squares as described in the corresponding analysis in section 2.6.

Algorithm Complexity

Initialization of the point array and calculating the normals is done on a per pixel basis. This means that in the worst case (all pixels valid), this step is executed approximately N times. Note that this is approximately as each normal computation requires two pixels in a row and two pixels in a column, hence the last pixels in the rows and columns are not tested. The complexity can be considered $O(N)$.

Finding a valid and unevaluated pixel requires checking all the pixels individually and has a complexity of $O(N)$. In the worst case, region growing is started for each pixel. This means that in this case N point lists are created. In practice, this will be less, because there are many invalid pixels and there are often many regions to find in the scene that have a considerable weight.

Testing of all the neighbors of a pixel if they are evaluated already is eventually done for each pixel and has a complexity of $O(N)$. Actually comparing the normals is only done for the unevaluated neighbors. Hence, if all pixels are valid each pixel will compare normals with one of its neighbors, which results in an approximate complexity of $O(N)$ (again due to that each comparison requires two pixels). In the worst case (if the whole scene is one big plane), recomputing the normal and adding a point to the list has a complexity of $O(N)$.

Each region is tested for its weight, which has a maximum complexity of $O(N)$ (if each pixel is a region). For each region with enough weight, the plane coefficients are computed. This requires summations over at most all the pixels ($O(N)$). In the worst case, the scene is divided in N/M regions, where M is the minimum amount of points that define a plane. The maximum complexity for the actual computation of the coefficients in this step is therefore $O(N/M)$.

Discarding the bad planes consists of checking the plane verticality ($O(N/M)$) and comparing every pixel in the valid regions to the estimated plane equations normal. If every pixel is part of a region with enough weight and each region is vertical, this has complexity ($O(N)$). As all the required summations required in the least squares estimate are known from the plane estimation procedure, the maximum complexity of estimating the line coefficients has complexity in $O(N/M)$.

This makes that the overall complexity of the algorithm is linear in N.

3.3.2 Corner Extraction

In figure 3.4, the flow diagram of the corner extraction algorithm is shown. This algorithm is an implementation of algorithm as described in section 2.7.

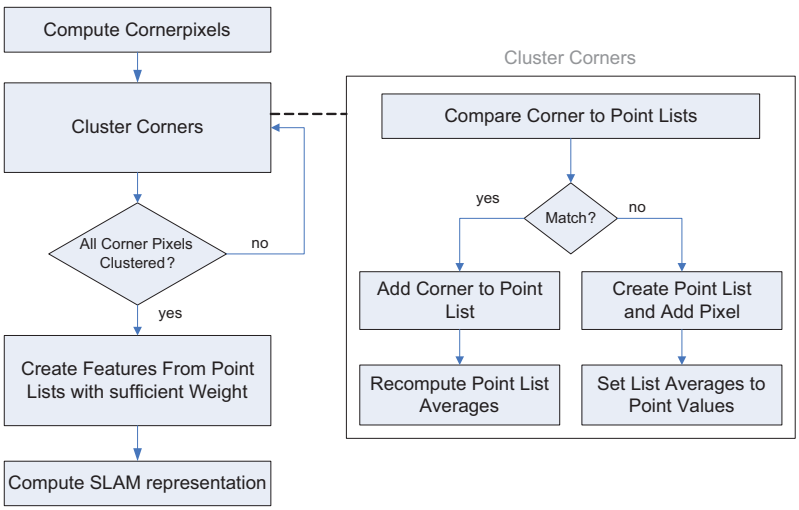


Figure 3.4: Flow Diagram of Corner Extraction Algorithm

The first step computes for each pixel if it is eligible to be part of a corner feature or not. This step is started by initializing two counters, one for the current scale i and one for the successfully detected scales *count*. In a loop, the dot product of the normalized difference vectors of the pixel with the left and the right neighbors is computed for each scale. If the pixel is a valid corner pixel at the current scale, the valid scale count is increased and the average angle of the different scales is updated. As soon as the current scale exceeds the maximum scale, the loop is ended. The valid scale count is tested. If the corner was detected at enough scales, the point is added to the list of corner pixels.

The second step in the flow diagram is the clustering of the corner pixels. An empty set of corner point lists (linked list implementation) is created to cluster the corners in. For each corner pixel, it is checked if it matches one of the already created lists. If it does not match, a new list is created and the averages of the angle and position are set to the angle and position of the point. Otherwise, the corner is added to an existing list and the averages are updated accordingly. After all the points have been clustered, corner features are created from the lists that are of sufficient weight. Finally a SLAM representation of the feature is computed and the feature is ready to be used in the SLAM framework.

Algorithm Complexity

Determining which pixels are corner pixels is a task that is done approximately for each pixel. The loop variables are initialized once ($O(N)$). For each pixel, a loop is executed S times $O(N \cdot S)$, where S is the maximum scale used to detect the corners. The worst case means that each pixel is a valid corner pixel at each scale. In this case, every pixel is marked ($O(N)$).

Clustering of the corners is done by comparing each corner pixel to the already created available corner point lists. The worst case would mean that, each pixel would create its own list, which is taken into account for comparing the other pixels. The number of corner point lists would grow linear with the amount of evaluated pixels. This would come down to $N(N+1)/2$ operations for the worst case. The complexity can therefore become rather large. Luckily, in regular scenes, the amount of corners is not very large and the complexity becomes much less.

An approximation of the actual complexity would be $O(N \cdot M)$, where M is the average amount of corners per scene. Note that the implementation chosen here is very naive, as it also tries to match corner pixels that are far apart in the image. If the amount of corner pixels in a scene becomes large, a tree based matching algorithm might improve performance. Calculating the SLAM representation is done for every corner in the scene $O(M)$.

This makes that the overall complexity of the algorithm is approximately linear in N, under the assumption that the average number of corners does not increase if more detail is provided by a higher resolution.

3.3.3 Jump Edge Extraction

The jump edges are already identified during the pre-filtering step. This information can be used to further process the jump edges, using the algorithm shown in the flow diagram of figure 3.5.

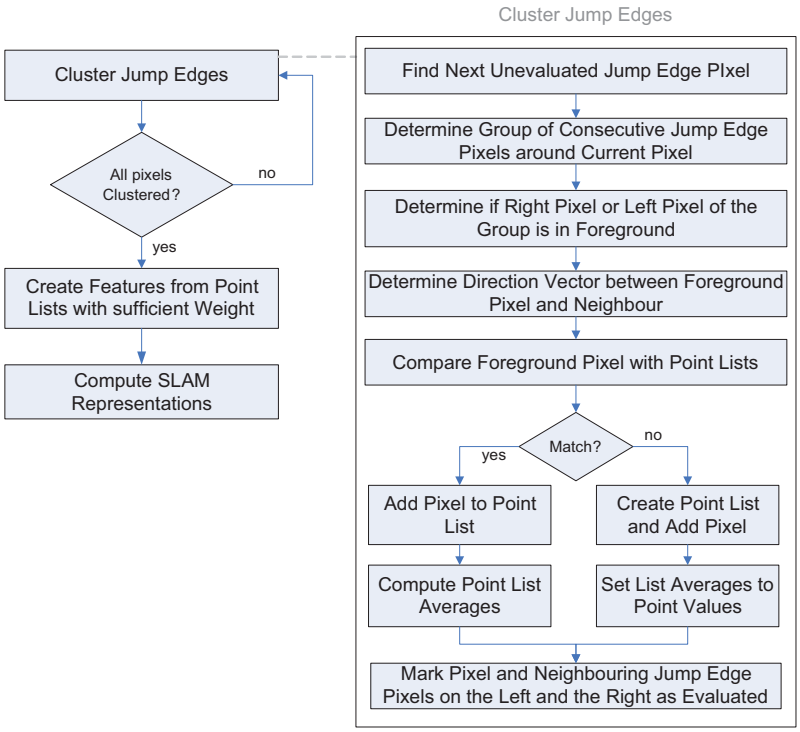


Figure 3.5: Flow Diagram of Jump Edge Extraction

The first step is to cluster all the jump edge pixels. This is done by iterating over all horizontal pixel lines from left to right. As soon as a horizontal jump edge pixel is encountered, a group is started. If the right neighbor is also a jump edge pixel, it is added to the group. This is continued until the beginning and ending pixel of the jump edge (at this pixel line) is found.

Next, it is determined whether the right pixel is in the foreground or the left pixel is in the foreground, simply by comparing the radial distance of both pixels. This determines the direction of the jump edge. If the left pixel is in the foreground, a difference vector is determined with its left neighbor. The other way around, if the right pixel is in the foreground, a difference vector is determined with its right neighbor.

Analogous to the corner clustering, the clustering of the foreground jump edge pixels is done by creating point lists. The jump edge pixel is compared to all existing jump edge point lists (linked list implementation). If it matches (is within thresholds), it is added to the points list and the averages of the position and the angle are recomputed. Otherwise, it creates a new lists and the average point is set to the pixel values. Finally, all pixels in the evaluated group are marked as evaluated.

For all point lists with enough pixels, a feature is created and the appropriate SLAM approximation is calculated using the mean value and the sample variance of the pixels.

Algorithm Complexity

Clustering of the jump edge pixels, involves evaluating all the pixels to test if they are marked as horizontal jump edges ($O(N)$). If all pixels were marked as jump edges, the complexity of determining the group and marking the pixels as evaluated is also $O(N)$. The rest of the clustering is only executed if a group of jump edges pixels is encountered. As detecting a jump group requires at least two consecutive jump edge pixels and one clear jump edge pixel in a row, the worst case number of pixels to be clustered is $N/3$.

Just like the corner clustering, the complexity can be approximated by $O(N \cdot M)$, where M is the average number of jump edges detected in a frame. This approximation is valid for the case

that all jump edge lists are created immediately and every pixel is compared to each of these lists. Creating the SLAM representation of each valid feature takes therefore at most $O(M)$.

This makes that the overall complexity of the algorithm is approximately linear in N, under the assumption that the average number of jump edges does not increase if more detail is provided by a higher resolution.

4 Results and Evaluation

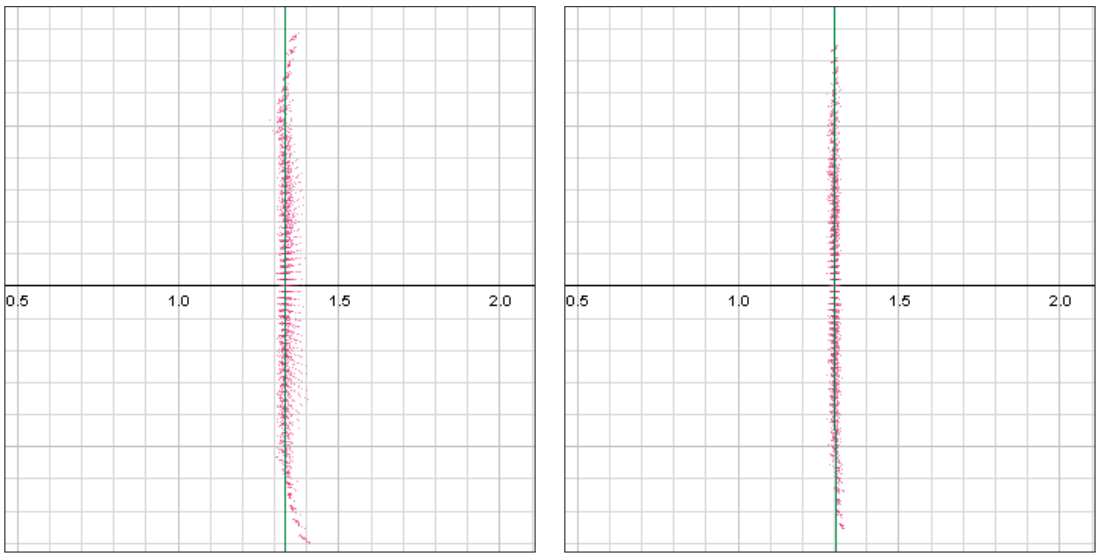
In this chapter, the different algorithms are evaluated. The chapter starts with a short discussion of the most apparent disturbances in the time-of-flight camera measurements that are encountered. This is followed by a discussion of the results that have been acquired using the feature extraction algorithms. The chapter concludes with the results of the proof-of-principle setup, where two datasets are recorded of two different domestic environments. The extracted features from this dataset are fed to the SLAM algorithm, which estimates the trajectory of the camera through the environment.

4.1 Evaluation of the Time-of-Flight Data

During this research, two disturbances in the time-of-flight measurements were most apparent. These are the calibration inaccuracy and the error caused by reflections in the scene.

4.1.1 Calibration Error

In experiment A.2, the calibration errors have been determined. It is shown that the measured distance of each pixel is slightly off by a factor. This factor is different for each pixel. A rough estimation of these factors has been made, using measurements of a straight wall at different distances. Applying a correction using these factors, results in an improvement as shown in the top view of the estimated plane in figure 4.1. This figure is made by pointing the camera at a straight wall at a distance of 130 cm.



(a) Before correction **(b)** After correction

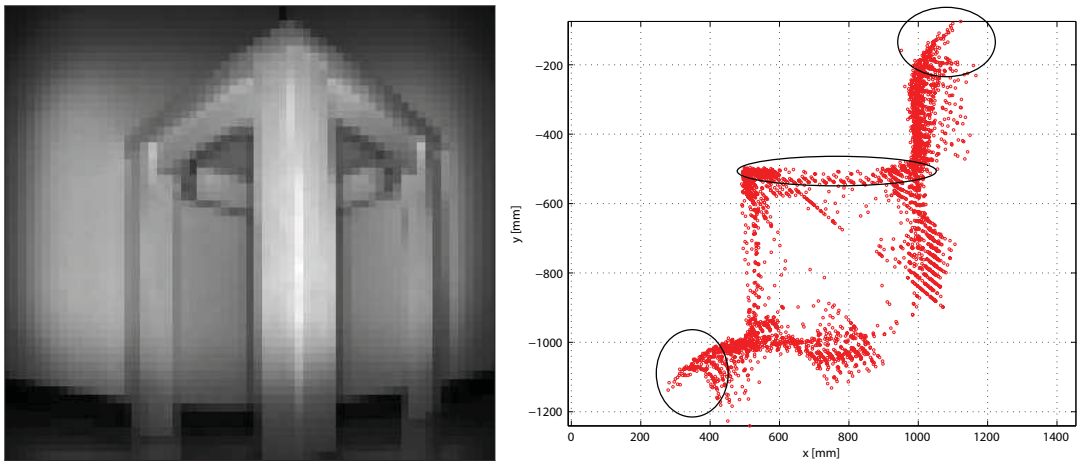
Figure 4.1: Top view of straight wall measured at 130 cm

Note that this calibration is based on measurements in a range of 1 to 2 meters. Better calibration might result in better performance, but this requires an accurately defined calibration setup.

4.1.2 Errors due to Reflections in the Scene

When light is reflected from one surface to another surface, an error is introduced. The light from the indirect path has a larger phase shift than the direct light coming from the camera. In experiment A.3, the effect of reflections in the scene has been investigated.

In figure 4.2, the influence of the reflections can be observed. It shows that the straight surfaces, marked by the ellipses, tend to curve outwards in the measurement of the time-of-flight camera. This effect is particularly severe on the sides of the point grid. If the amplitude image is observed, it is shown that the surfaces that are observed by the pixels on the sides of the camera, are less well illuminated by the direct light of the camera. This causes these surfaces to be much more prone to reflections.



(a) Amplitude image **(b)** Top-view of measured points

Figure 4.2: Measurement of a table in the corner

Note that this disturbance is highly unpredictable and causes problems in correctly applying SLAM. Features that are observed by the pixels on the side of the sensor, suffer from large inaccuracies. If multiple observations of a feature are made with these pixels and are fed to a SLAM algorithm, problems arise. The landmark estimate will be erroneous, as the Kalman Filters only converge to the correct value for zero-mean noise and therefore become overly confident.

It is therefore important that this type of disturbance is minimized as much as possible. This can be done by either sufficiently illuminating the surface or discarding the pixels on the sides of the sensor. Because the viewing angle and resolution are already not very large, the first option is preferred.

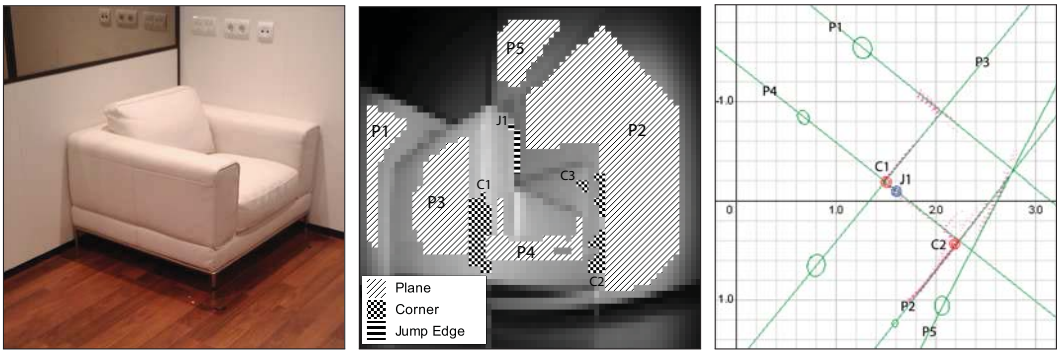
4.2 Evaluation of the Feature Extraction Algorithms

In this section, the feature extraction are evaluated. First the features are shown in a typical scene, followed by an evaluation of the position of the features in different recorded sets.

4.2.1 Features in a Typical Scene

In figure 4.3, a typical scene is shown with a chair in the corner. The left part of the image is a photograph of the chair. The middle image is the amplitude image of the camera, with the feature pixels marked. In the right image the corresponding 2D top-view is shown. This shows a map of the extracted features and there covariances as 1σ-ellipses.

The camera is placed in the origin and is pointing to the right (x-direction).



(a) Photograph **(b)** Extracted Features **(c)** Top-view map

Figure 4.3: Feature extraction from a 3D image of a chair

The major planes in the scene are detected, which are the two walls (marked as P1 and P2) and the two sides of the chair (P3 and P4). Furthermore, corners are extracted where the surfaces make an angle (C1, C2 and C3). Note that corner C3 is not visible in the map as its weight is too low to generate a landmark. One jump edge is visible (J1). Note that the corner that the chair makes with the wall on the left of the image is too near to the border of the 3D image to be detected.

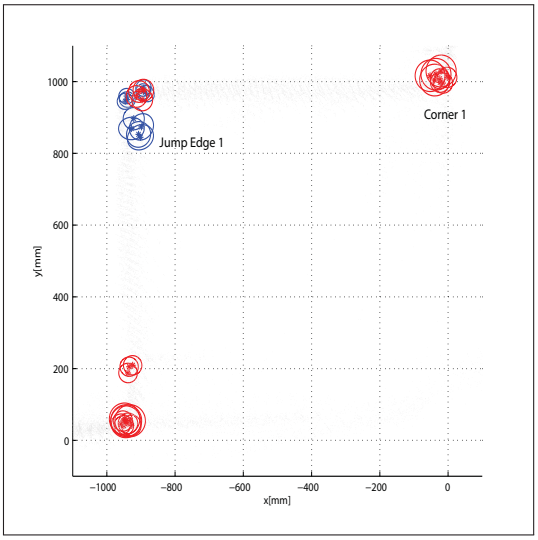
If the 2D top-view map is observed closely, it can be seen that there seems to be an odd plane in the corner of the scene (P5). Because there is a large mirror on the wall, the wall is illuminated by both direct light and light reflected by the mirror. Note that this is a very severe case of reflection disturbance as the mirror almost completely reflects all incident light. This shows however, that highly reflective surfaces in the scene can cause problems in correct depth estimation.

4.2.2 Evaluation of the Feature Extraction Methods

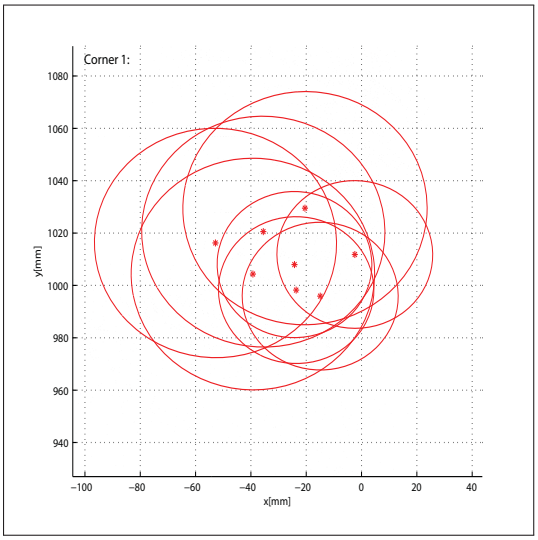
In experiment A.5, the corner and jump edge features have been evaluated for accuracy and noise. It is shown that for the measured distance, these features have on average an accuracy of about 5 cm at a distance of 2.5 m. This accuracy is expected to scale linear with distance, as the density of the measured points scales also scales linear with distance. For most features detected in this experiment, a fairly accurate covariance matrix is generated. For highly irregular corners however, the covariance is underestimated. Additional research, might provide a better noise model. For this proof-of-principle, it is however considered sufficient.

An example of the estimated covariances is given in figure 4.4. A dataset of a chair, observed from three different positions and different orientations is shown. The image consists of an amplitude image, a top-view map of the extracted features and two zoomed views of a corner and a jump edge. For each observation of a feature, the position and estimated covariance ellipse (1σ) is shown.

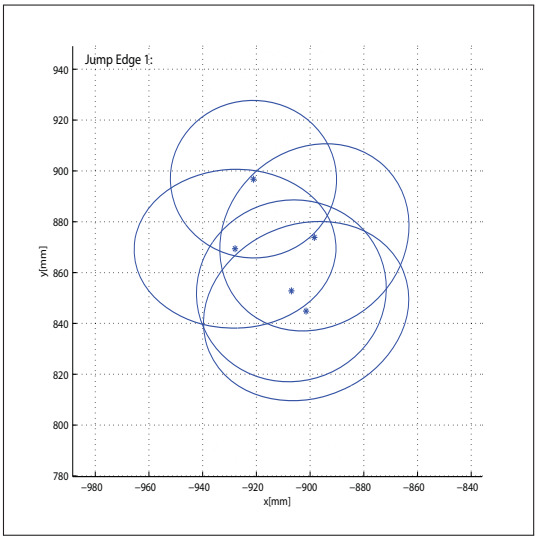
(a) Amplitude image



(b) Top-view map



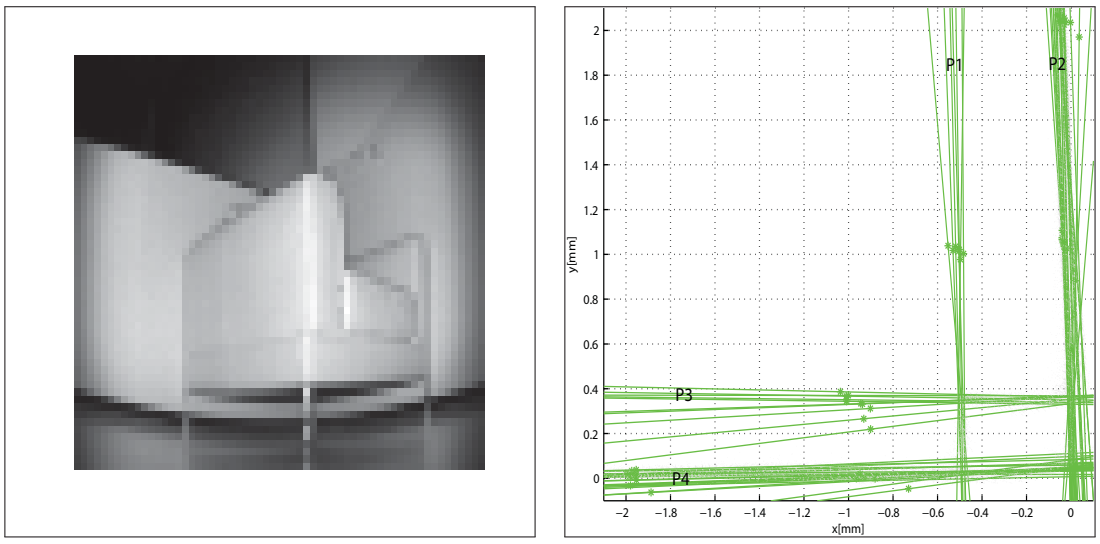
(c) Zoom of corner



(d) Zoom of jump edge

Figure 4.4: Results for the corner and jump edge extraction on a chair

The accuracy of planes is severely distorted if a plane is fitted through the border pixels of the sensor. In experiment A.3 it is determined, that this distortion is most likely caused by reflections in the scene. Providing a homogeneous illumination across the whole range of the sensor, can solve this problem. An example of the plane estimates for a chair are shown in figure 4.5. It is clearly seen that the planes generate several outliers. These correspond to observations by the border pixels.



(a) Amplitude image **(b)** Top-view map

Figure 4.5: Plane estimates in the dataset of a chair

The measured average execution time per frame is 21 ms on a 2.0 GHz Intel CPU. As frames come in at a rate of 5 Hz, the system is loaded to approximately 10 percent by the algorithms. As the algorithms are highly parrallelizable, a different architecture might improve upon this execution time.

4.3 Performance of Features in a SLAM environment

The performance of the SLAM environment has been evaluated by running two datasets from two measurements in the Philips testing room through the processing algorithms. Both datasets are based on the same ground truth, defined by a set of markers taped on the ground (see figure 4.6). The camera has been moved with a maximum linear velocity of about 15 cm/s and a maximum angular velocity of about 0.5 rad/s.

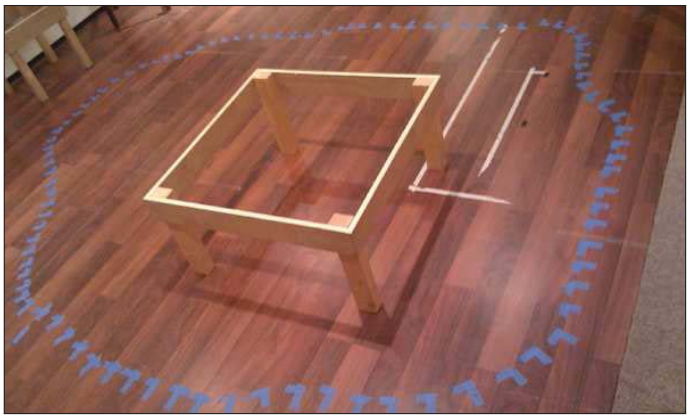


Figure 4.6: Ground truth of both datasets

Each of the datasets is based on a different setup of the furniture in the room. These setups are shown in figure 4.7.

The SLAM algorithm is provided with a bold motion prediction, based on the robot motion model described in section 2.4.2. The parameters of this prediction is set to an average forward velocity of 0.015 m per frame and an average angular velocity of 0 m per frame. Note that the linear velocity is chosen higher than the actual average velocity, for better performance during

(a) Scene 1 **(b)** Scene 2

Figure 4.7: Photographs of the two different scenes

loop closing. The Gaussian noise added to the linear velocity has a standard deviation of 0.03 m per frame. The noise on the angular velocity has a standard deviation of 0.03 radians per frame. The noise on the final angle is also 0.03 radians per frame. These parameter values enable the algorithm to follow the movements of the camera and still have enough density left in the particle cloud. In a real situation these parameters would of course be replaced by odometry data, which provides additional accuracy. The rest of the estimation is based solely on the features extracted from the datasets. As the recorded data forms a loop, the SLAM algorithm can run infinitely on the dataset.

In figures 4.8 and 4.9, a map is shown of the SLAM algorithm (corresponding to the two scenes in figure 4.7). The brown dots represent the ground truth, while the blue dots correspond to the estimated path. The landmarks are shown in green (planes), blue (jump edges) and red (corners). The ellipses that drawn around the landmarks, represent their covariances. Note that the landmark map that is shown is of the current best particle in the particle cloud.

The algorithm has proven to be stable for the two tested scenes, without the odometry that a robot normally provides. The average deviation from the path is in the order of several centimeters (about 5 to 10 cm), and is expected to get smaller as more accurate odometry is provided. Note that the reflection effect is again seen in the upper left corner of the map, where multiple plane features are created at different angles. There are however enough corner features jump edge features left, to keep the algorithm stable.

With a particle count of 1000, the complete processing algorithm is able to run in real time on a 2 GHz processor. However, if odometry is provided, the amount of particles necessary to represent the robot pose uncertainty is expected to be less. Moreover, as the FastSLAM algorithm linearly scales in complexity with the amount of particles, the processing power needed is decreased. Note also that FastSLAM is based on a particle and therefore consists of N parallelizable individual estimators, with N the number of particles. A change of architecture might therefore speed up the FastSLAM algorithm.

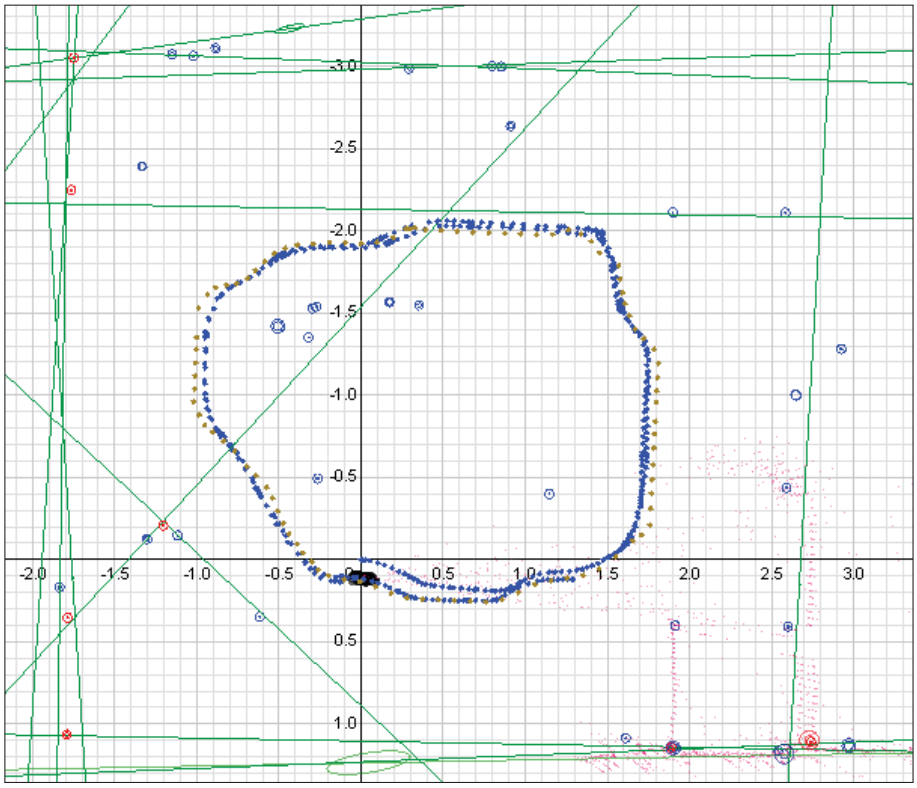


Figure 4.8: SLAM map of scene 1

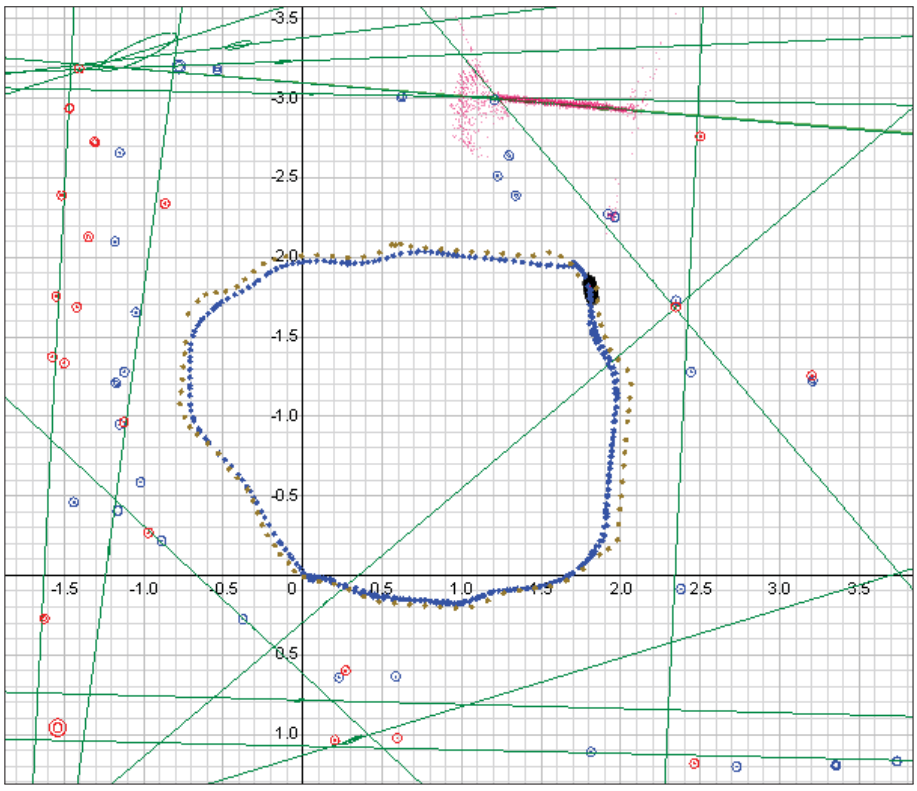


Figure 4.9: SLAM map of scene 2

5 Conclusions and Recommendations

5.1 Conclusions

The goal of this research is to investigate the potential of a time-of-flight camera for autonomous indoor navigation. This has been done by developing a proof-of-principle setup for this purpose. The setup is capable of extracting valuable data (features) from the time-of-flight images and use these features in a SLAM algorithm that estimates the structure of the environment together with the position of the camera within this environment.

A SLAM algorithm requires landmarks that are robust, easy to detect and that are accurately positionable, to be able to give a reliable map and position estimate. A total of three different types of features have been selected that fulfill these requirements: vertical planes, vertical corners and vertical jump edges. For each of these features, a separate algorithm has been developed that is able to extract these features from the 3D images and process the features into a representation that can be passed to an arbitrary landmark based SLAM algorithm.

The different feature extraction have been tested in two different setups with pieces of furniture in a rectangular room. In these setups, the furniture has been placed to resemble a domestic environment. It has been shown that the feature extraction algorithms are highly responsive in this kind of setup. The selected features therefore seem good candidate features for use in a domestic environment. The corner features and jump edge features provide on average an accuracy of 5 cm at a distance of 2.5 m. This accuracy is expected to scale linear with distance. The plane extraction algorithm suffers from disturbances in the camera data due to reflections of light in the scene. The accuracy of this algorithm can therefore not be accurately determined.

To perform the actual localization and mapping, an implementation of the FastSLAM 1.0 algorithm has been made. Test results using two different setups of a domestic environment show an average localization error between 5 to 10 cm without further sensors, provided that feature information is constantly available. If additional sensors are added, like odometry sensors or a gyroscope, the accuracy of the algorithm is likely to increase further and become stable in situations where no feature information is available for a short time.

For the development of all the algorithms, a software pipeline has been created. This pipeline consists of three modules: a module that retrieves data from the camera, a module that processes this data into a map and position estimate and a module that gives feedback to the user. All of these modules are connected by a TCP/IP connection, which lets each of these modules be executed on a different system. This is convenient for development on an embedded platform.

The complete system is capable of running in soft real-time on a 2 GHz processor. About 10 percent of the total CPU load of the processing is used for feature extraction. The rest is mostly used by the SLAM algorithm. As additional sensors are added, the CPU load of the SLAM algorithm is expected to be greatly reduced. Many of the modules in the processing software can be run in parallel and most of the algorithms can be split up in simple operations that are applied to all pixels. This makes that the algorithms could benefit from a multicore CPU or a programmable logic device.

5.2 Recommendations

5.2.1 Recommendations for Further Research and Testing

The proof-of-principle setup has been tested in two basic living room setups. Further research and testing needs to be done, to make the algorithms robust in a broader range of scenes. It is

advisable to first mount the camera on an actual robot for this purpose. Difficult scenarios, in which the algorithms have not been addressed are:

- Cleaning under furniture
- Coping with very near objects that blind the camera
- Handling dynamic objects
- Fast motion of the camera with respect to the environment
- Environments with small objects

Cleaning under the furniture makes that large parts of the scene are obscured and that the light from the camera is confined to a tight space. This makes that light is likely to be heavily reflected, which causes inaccuracies in the data. If surfaces are observed from nearby (which also applies when cleaning under furniture), the measurements further decrease in accuracy. This is because very bright light causes the sensor to saturate and generates large reflections inside the lens. Further testing is therefore needed, to determine the performance of the sensor in these situations.

Dynamic objects are objects in the room that do not stay in place during the mapping of the environment. Examples of dynamic objects are people and pets walking around. If landmarks are generated based on features on such objects, localization and mapping becomes erroneous. Further testing is required to characterize the errors that dynamic objects introduce in the 3D data and to develop algorithms that address these errors.

Fast motion of the camera with respect to the environment introduces motion blur. In the current tests, the linear and angular velocities of the camera have been limited to approximately 15 cm/s and 0.5 rad/s. The SLAM algorithm seems to perform well at these velocities, but exact data is not available. Combining a motion estimate of the camera movement, with the 3D image and timing information from the camera, can yield to an accurate per pixel motion blur estimate. Such an estimate could then be used determine the error in the extracted features.

The objects that were used in the testing are rather large. In an environment with small objects, many landmarks are expected to be generated that are not very useful. Features on these objects are detected as soon as the camera gets near these objects, but if the camera is further away, these features do not aid in localization anymore as they cannot be observed. Further testing therefore needs to be done in different environments with different objects, to determine the overall performance of the extraction algorithms.

5.2.2 Recommendations for Improvement of the SLAM and Feature Extraction Algorithms

The SLAM algorithm itself is implemented fairly naive. The most costly procedure in the algorithm SLAM is the matching of the landmarks. This can be optimized, by determining which landmarks need to be matched (for example using tree like structures). FastSLAM has also some drawbacks. Loop closing is for example something that FastSLAM does not do well. Implementing HybridSLAM (Brooks and Bailey (2008)) might increase the stability of the algorithm.

The robustness of the localization could be increased by adding more information about the features to the feature vectors. For the vertical corners and jump edges, only the position is used right now in the SLAM algorithm. Adding for example the direction or the average height of detection makes these features more distinguishable from other corners and jump edges. Note that corners can also become jump edges if observed from a different position. This is currently not taken into account, but might be used to extract additional information about the scene.

Another possibility is to add information of the amplitude image to the feature vector. This might be especially useful in distinguishing between planes. Vertical planes features are currently described by infinitely long lines in the 2D map. This makes them prone to false match-

ing, if planes are extracted of which the representations more or less resemble the representations of other already detected planes.

5.2.3 Recommendations for Improvement of the Time-of-Flight camera

A couple of suggestions can be given for the camera that would improve the performance of the algorithms:

- During the research, some disturbances in the camera have been identified that negatively affect the results. The most important one is the lack of illumination on the surface that is measured by the border pixels. These pixels are very sensitive to indirect reflections and therefore cause large errors in plane estimation.
- Making the illumination power adjustable by software, would make it possible to do depth estimation in situations where the camera currently becomes blinded. This is especially convenient for moving in narrow spaces (under furniture for example).
- The PMD sensor provides information on the background illumination. This is however not available on the interface, which makes it impossible to do an accurate noise estimation for each pixel.
- A higher resolution could open the door to using 3D SIFT or SURF keypoints, which has shown good results in other researches.
- A wider field of view in horizontal direction, would make it easier to track features while rotating.

A Experiments

A.1 Determining Pixel Distribution

In this experiment, the pixel distribution of the sensor is estimated. Radial distortion in the lens changes the direction of the incoming light rays. The camera compensates this by specifying an adjusted unit vector for each pixel. These unit vectors do not change with the images and determine the relation between radial distance and x, y, and z coordinates.

The unit vectors van be found by dividing each measured coordinate (p_x, p_y and p_z), by the measured radial distance (equation (A.1)).

$$\vec{u}_i = \frac{\vec{p}_i}{r_i} \tag{A.1}$$

A scene is picked that is measured at a distance of approximately 2 m, which provides a resolution of about 0.5 mm in depth resolution. To make the pixel distribution visible, a projection of these vectors is made on the plane defined by $z = 1$. This would equal the measurement of a flat vertical wall at a distance of 1 m. The points p_proj of this projection can be calculated from the unit vectors using equation (A.2):

$$p_{\vec{proj},i} = \frac{\vec{u}_i}{u_{i,z}} \tag{A.2}$$

A.1.1 Result

The result of this test is shown in figure A.1.

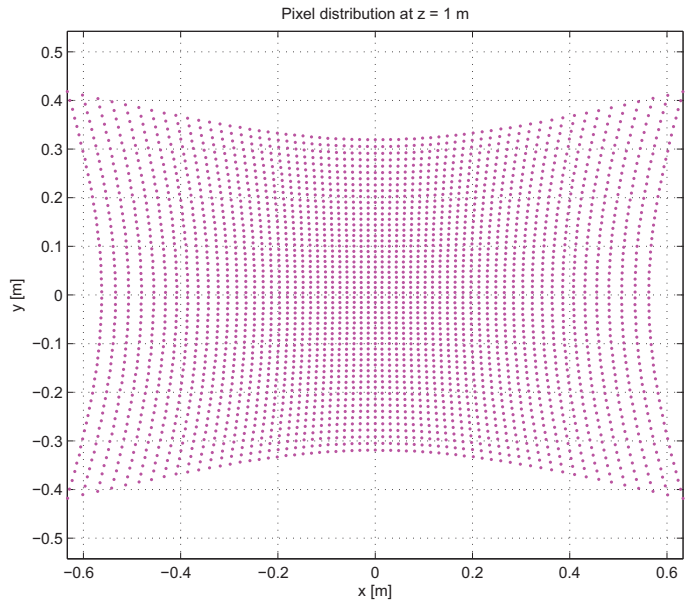


Figure A.1: Pixel distribution at unity distance

A.2 Estimation of Calibration Errors

From observations of the data, it has become apparent that the measured distance has a structural deviation from the actual distance. This deviation is different for each pixel. In figure A.2, the measured distance errors of a measured wall at 110 cm from the camera are shown

as a function of the pixel indices in y-direction and z-direction. The distance error for border pixels as much as 20 cm. In this experiment this deviation is determined and a correction is introduced based on the estimated deviation.

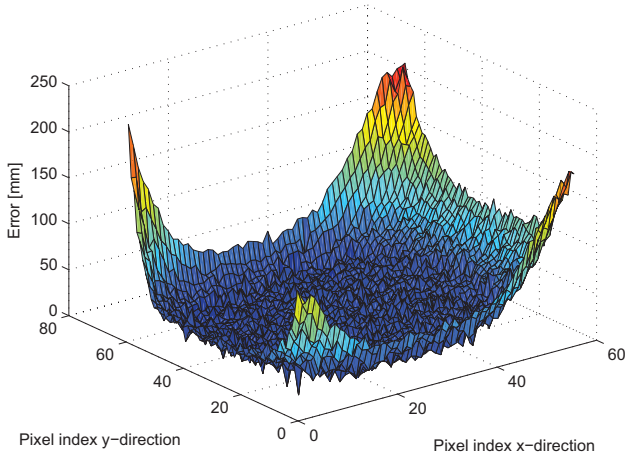


Figure A.2: Plot of the distance error on the pixel grid

To determine the relation between measured distance and actual distance, the camera is placed in front of a straight wall, which is considered a perfect plane. A figure of the setup is shown in A.3 wall and the camera are aligned in vertical direction, by using a using a spirit level. The camera is now tilted until both give the same spirit level read out. The rotation of the camera around the z-axis is determined using the line fit algorithm of the plane extractor. The camera is rotated until an average angle of 0 is measured.

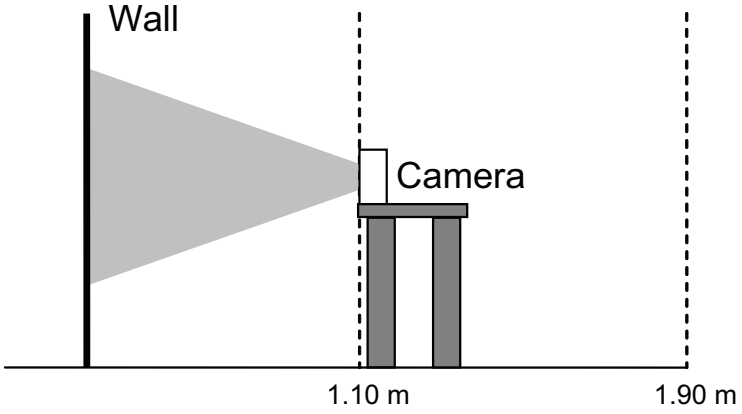


Figure A.3: Measurement setup for determining calibration errors

The camera is placed at four different distances from the wall: 110, 130, 140, 160 and 190 cm. This distance is measured from the base of the camera, using a measuring tape. For each of these distances, the difference between the measured pixel and the actual pixel is calculated. Finally, a linear approximation is made to these errors that can be used to correct the data.

A.2.1 Results

The resulting distance error at different wall distance, is shown for 5 arbitrary points in figure A.4. The graph reveals, that de difference error increases with distance and the effect seems to be approximately linear and decreasing to 0 as the distance becomes smaller.

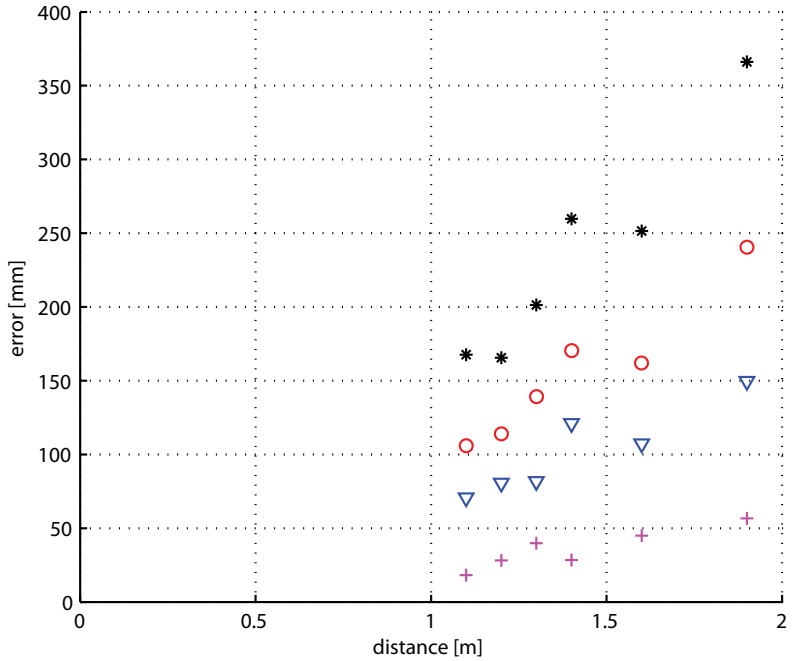


Figure A.4: Distance error measurements at different distances for four arbitrary pixels

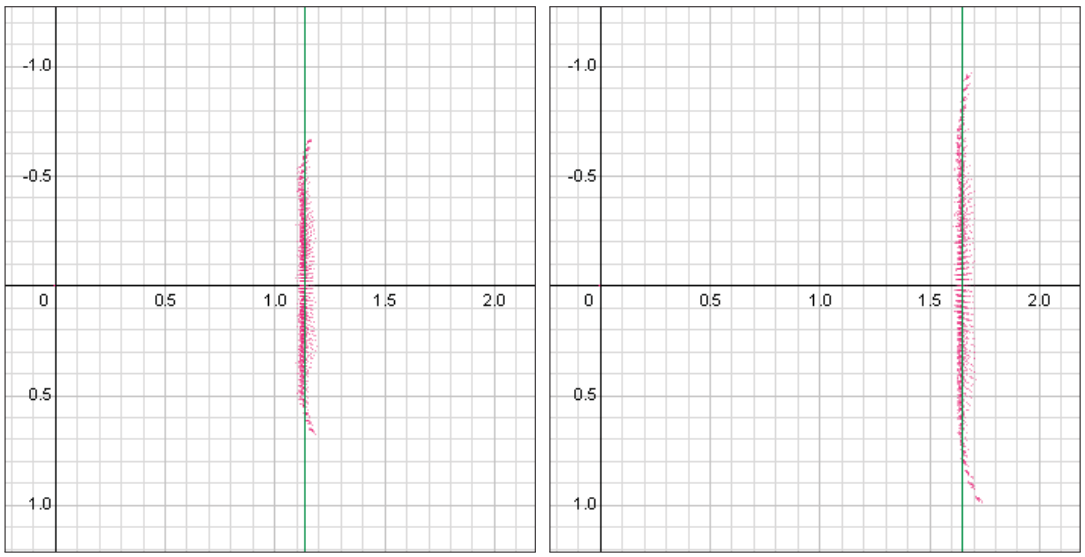
A rough estimate of the distance correction is made, by fitting a line through the points and the origin for each pixel separately, using a least squares estimator. Such a line is defined by its slope. The distance correction can be computed using this slope a by equation (A.3).

$$r_{corr} = r - ar = (1-a)r \tag{A.3}$$

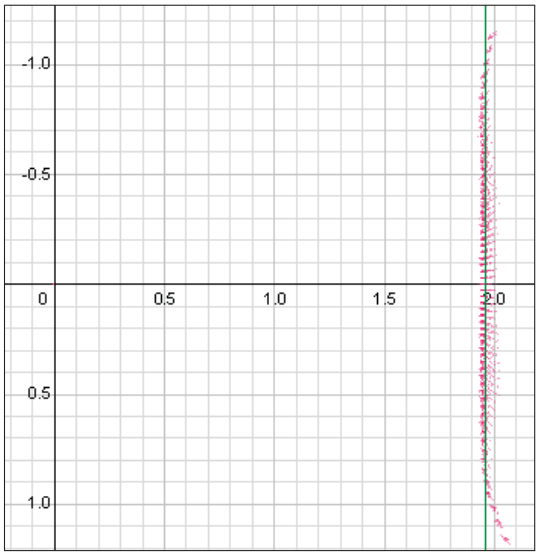
As the x, y and z coordinates are defined as unit vectors scaled by the distance, this can be generalized to these coordinates (equation (A.4)).

$$x_{corr} = (1-a)x \qquad y_{corr} = (1-a)y \qquad z_{corr} = (1-a)z \tag{A.4}$$

In figures A.5 and A.6, the results before and after correction are shown.

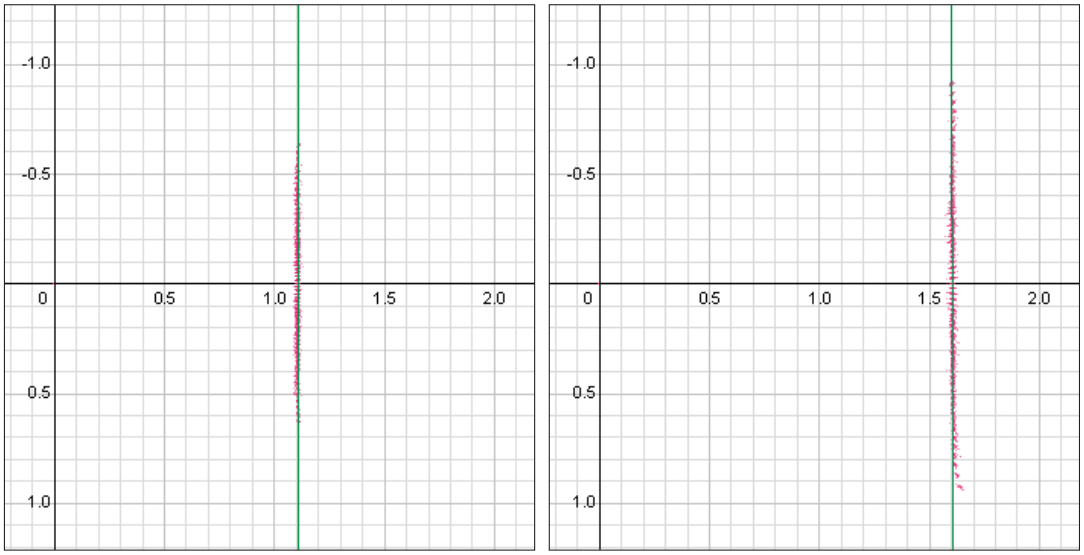


(a) 110 cm **(b)** 160 cm

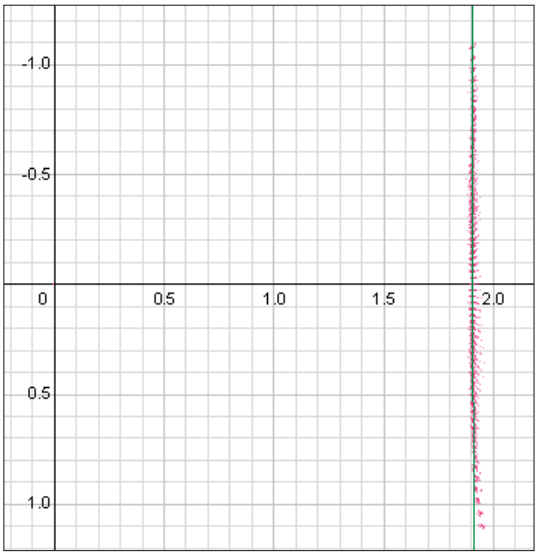


(c) 190 cm

Figure A.5: Walls at 110, 160 and 190 cm, before correction



(a) 110 cm **(b)** 160 cm



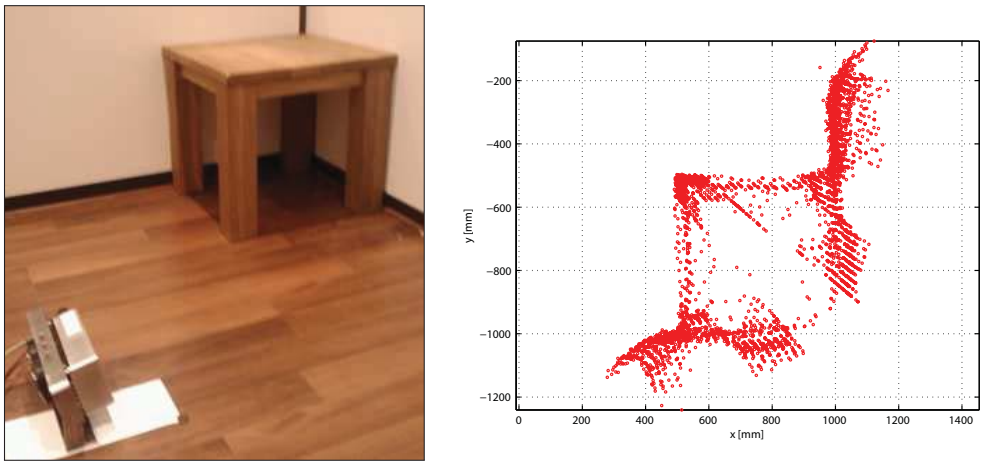
(c) 190 cm

Figure A.6: Walls at 110, 160 and 190 cm, after correction

Note that this is just an estimate calibration in a limited range of the camera. A better calibration should be done in a larger wall room or a special testing facility, so that the estimated coefficients can be validated for larger distances. For this research, this rough calibration is considered sufficient.

A.3 Determining the Influence of Reflections

A severe kind of distortion is observed in the measured distance. An example of this is shown in a measurement of a table in a corner at a distance of 1m to both walls of the corner. A photograph of this setup is shown in figure A.7a. The corresponding top view of all the pixels is shown in figure A.7b.



(a) Photograph **(b)** Top-view of measured points

Figure A.7: Table in corner

It can be clearly seen that sides of the table seem dented, so do the walls near the corner. The most heavy distortion is however at the end of the walls. The curved side of the table and the walls can be explained by due to the fact that light is reflected off the wall onto the table or onto the other wall.

To explain the curvature of the walls at the ends, a look is taken at the amplitude image of the scene. This amplitude image is shown in figure A.8. It can be seen that the sides of the amplitude image have a greatly decreased amplitude, with respect to the center pixels. As the pixels do not receive a lot of direct light, they are much more sensitive to light coming from reflections on other surfaces.

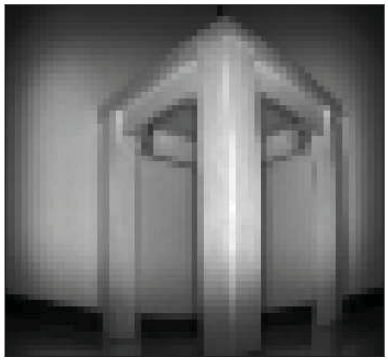


Figure A.8: Amplitude image of table in corner

This is a distortion that is unpredictable and causes significant errors, especially on surfaces. Plane estimates are therefore highly influenced by this effect, as is shown in figure A.9, which is a top-view of two estimated planes (green). The planes should have an angle of 90°, but the angle is bigger due to reflections.

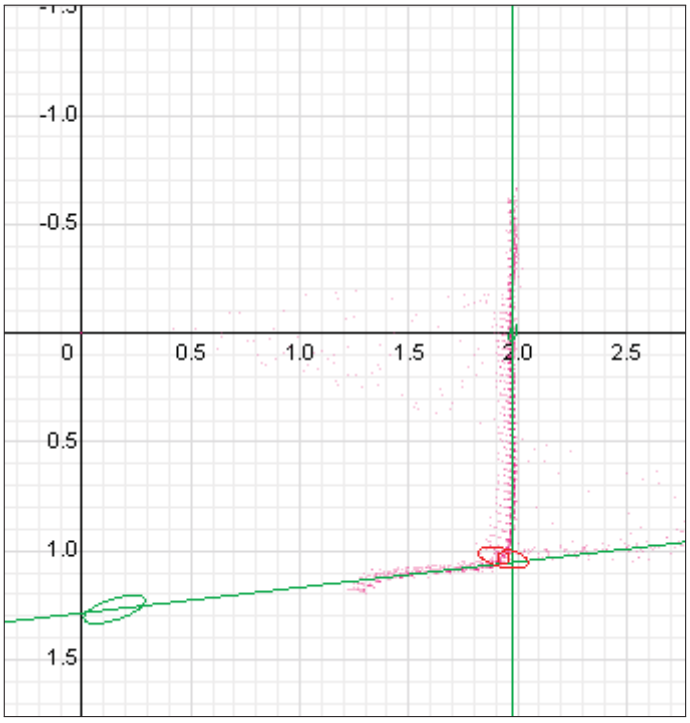


Figure A.9: Top-view of corner of 90 degrees

This influence is most apparent on the sides, due to the lack of proper illumination. As SLAM creates many new landmarks based on features that slide in from the side if the images, this effect is extra severe. It is therefore important to have a very homogeneous illumination to minimize this effect as much as possible.

A.4 Evaluation of the Plane Extraction Algorithm

In this experiment, the accuracy of the plane extraction algorithm is evaluated. This is done by extracting planes from several 3D images of typical objects in a domestic environment.

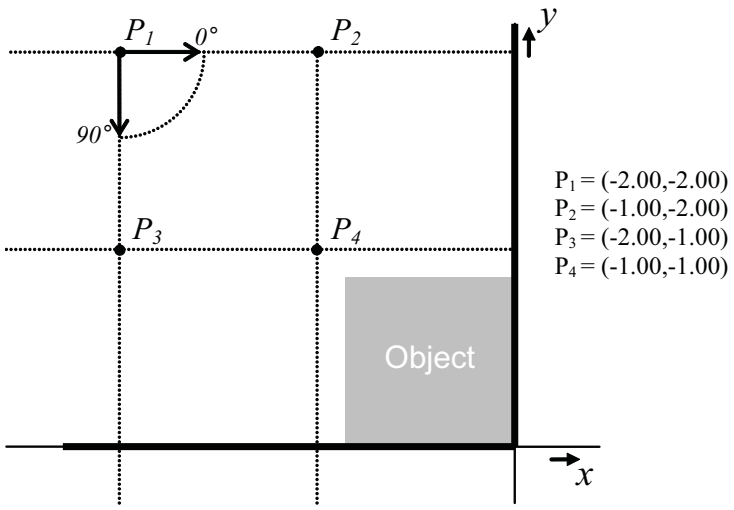


Figure A.10: Measurement setup for evaluation of the estimated errors in feature extraction

A measurement setup is constructed in an empty corner of a room, as shown in figure A.10. In this corner, four measurement positions are defined (P_1, P_2, P_3 and P_4). The camera is placed at one or more of positions, depending on the object. At each of the measurement positions,

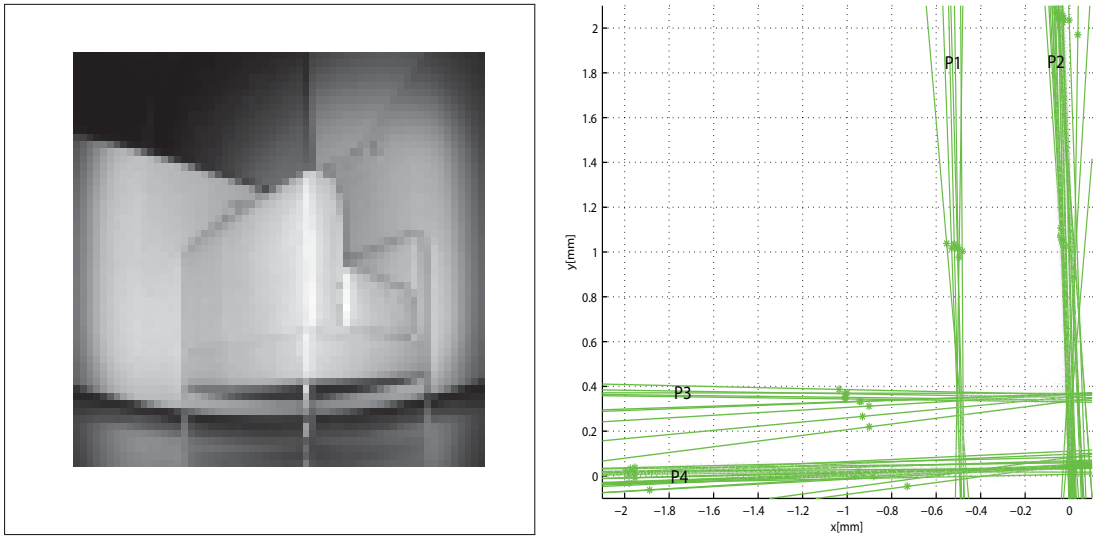
the camera is given ten different orientations from 0°to 90°in steps of 10°. Two different objects are placed in the corner: a chair and a box.

The extracted plane features in each image are matched to the plane features in the other images. Using these matches, a set of plane feature lists is made. Each list contains the observations of the same plane in the scene from all the 3D images. For each plane feature list, the covariance is computed as the average of the estimated covariance of each observation i. This is described in equation (A.5)), where $\mathbf{C}_{m,i}^{[m]}$ is the covariance of observation i in the map frame.

$$\mathbf{C_{avg}} = \sum \mathbf{C}_{m,i}^{[m]} \tag{A.5}$$

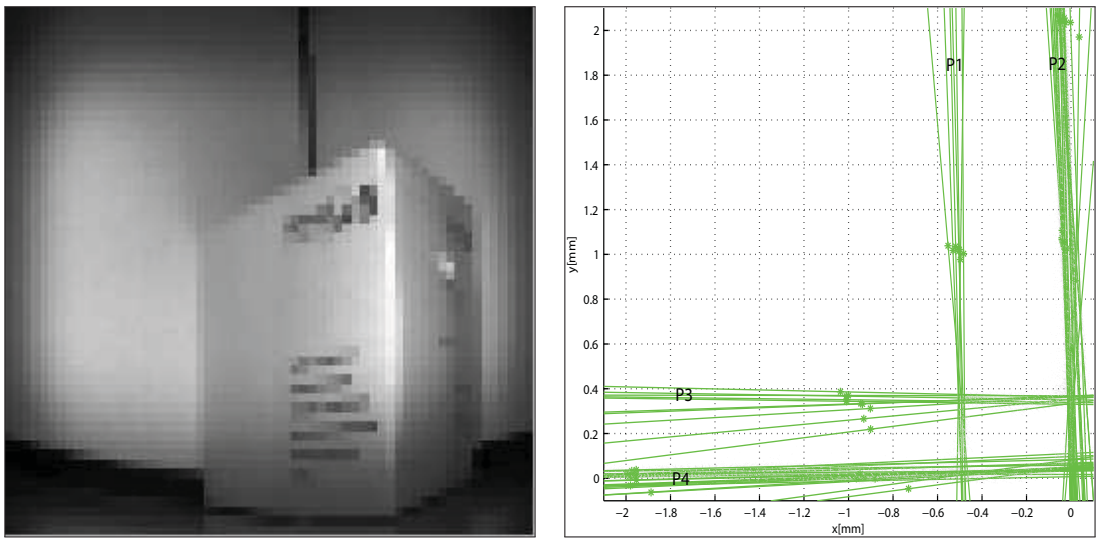
A.4.1 Results

In figure (A.11), the plane estimates of the dataset of the chair are shown. It shows that especially the estimates of plane P1 and P3 contain several outliers. These outliers are caused by erronous depth estimation at the sides of the point grid (as shown in (A.3)). In figure (A.12), the same is shown for the box, where the same effect is observed.



(a) Amplitude image **(b)** Top-view map

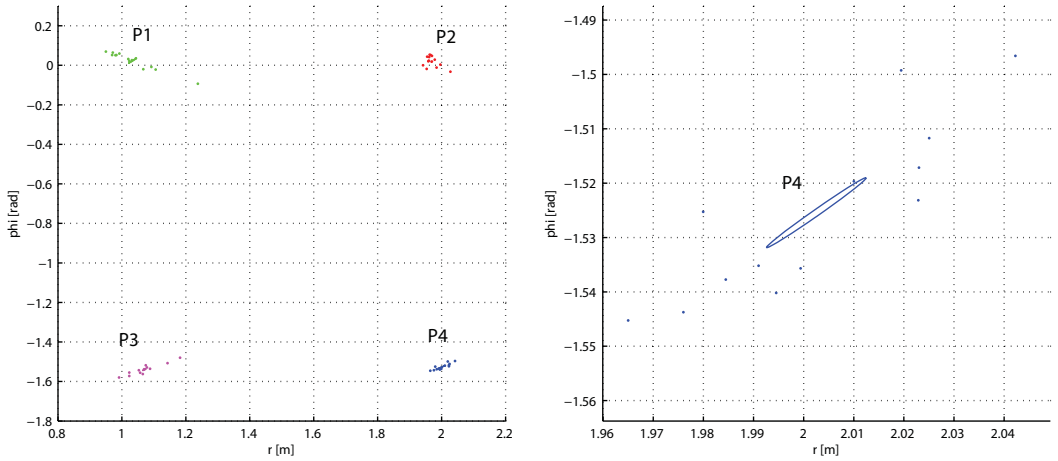
Figure A.11: Plane estimates in the dataset of a chair



(a) Amplitude image **(b)** Top-view map

Figure A.12: Plane estimates in the dataset of a box

In figure (A.13), the estimated distances of the observations are shown against the estimated orientations of the plane normals. The right part of the figure shows a zoomed view of plane (P4), including the estimated covariance ellipse (1σ). As can be seen from the graph, the covariance is severely underestimated. It is apparent that the sample variance in one plane estimate does not resemble the variance of several observations from different positions.



(a) Estimates for all planes **(b)** Zoomed view of plane

Figure A.13: Estimated distance with respect to estimated angle (chair)

Reflections in the scene and other disturbances in the depth estimation are assumed to be the major cause of this. To compensate for this underestimation, an additional variance $\sigma^2_{r,add}$ is added to the estimated distance to the plane and an additional variance $\sigma^2_{\phi,add}$ is added to the estimated orientation of the plane. This yields a new covariance matrix $\mathbf{C}^*_c$, as defined in equation (A.6).

$$\mathbf{C}^*_c = \mathbf{C}_c + \begin{bmatrix} \sigma^2_{r,add} & 0 \\ 0 & \sigma^2_{\phi,add} \end{bmatrix} \tag{A.6}$$

By trial and error, $\sigma_{r,add} = 0.5$ [m] and $\sigma_{r,add} = 0.4$ [rad] have been determined as good approximates to the real covariance. The results are shown in figures A.14 and A.14, for the chair and the box respectively. It shows that the covariance estimate has improved, but there are still many outliers. This choice has however been made, to keep the SLAM algorithm stable. The algorithm will generate multiple landmarks for planes that generate many outliers, which makes the localization and mapping less sensitive to plane estimates that have converged to an erroneous value.

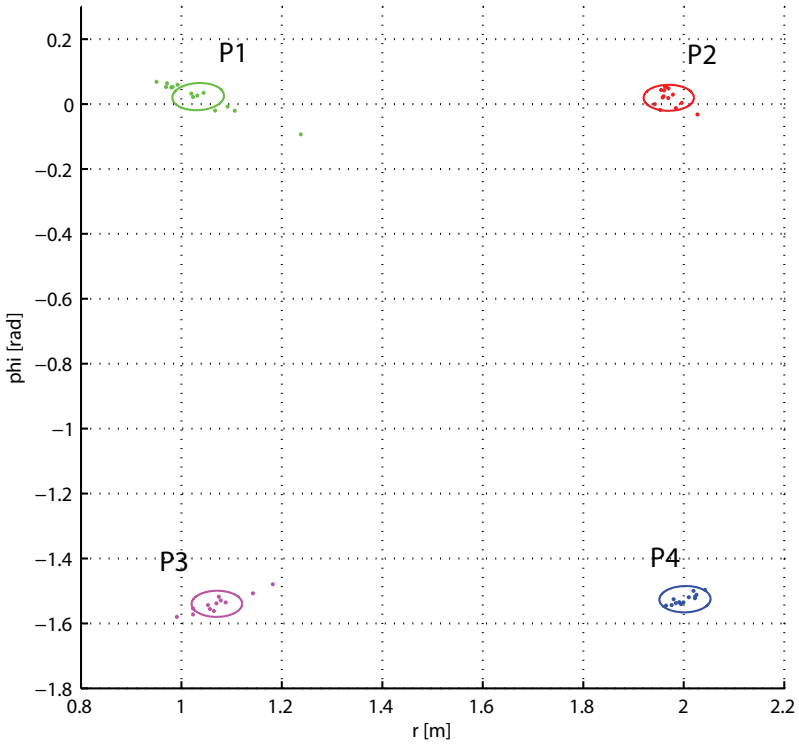


Figure A.14: Estimated distance with respect to estimated angle, with corrected noise model (chair)

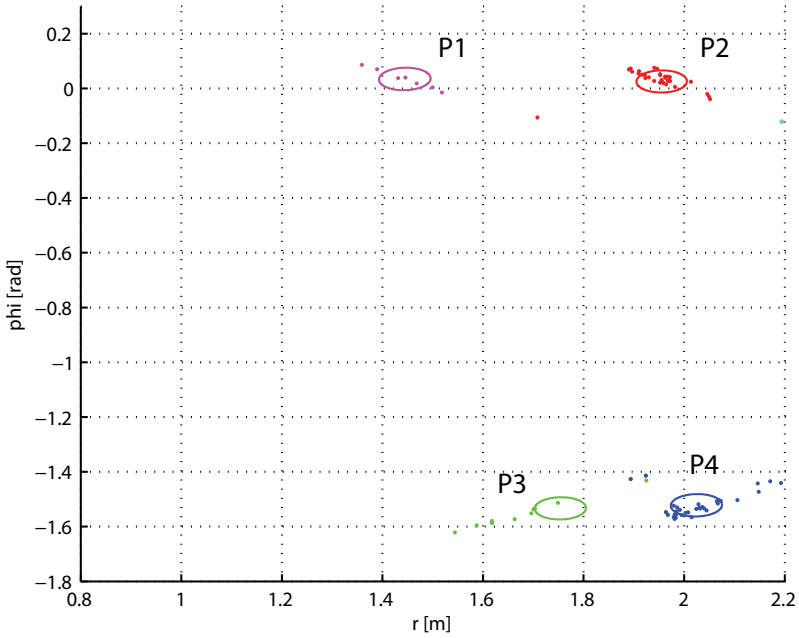


Figure A.15: Estimated distance with respect to estimated angle, with corrected noise model (box)

A.5 Evaluation of the Corner and Jump Edge Extraction Algorithms

The corner and jump edge extraction algorithms estimate the errors in the measurement vector of the extracted features. In this experiment, these estimated errors are evaluated. This is done by extracting features from several 3D images of typical objects from a domestic environment.

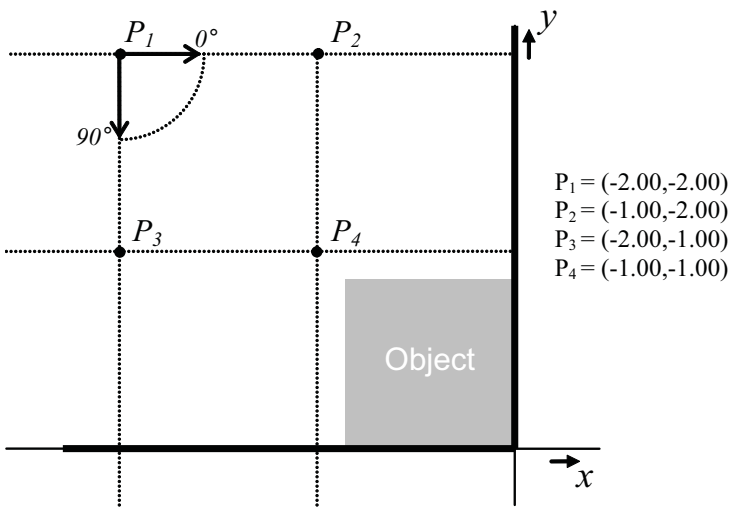


Figure A.16: Measurement setup for evaluation of the estimated errors in feature extraction

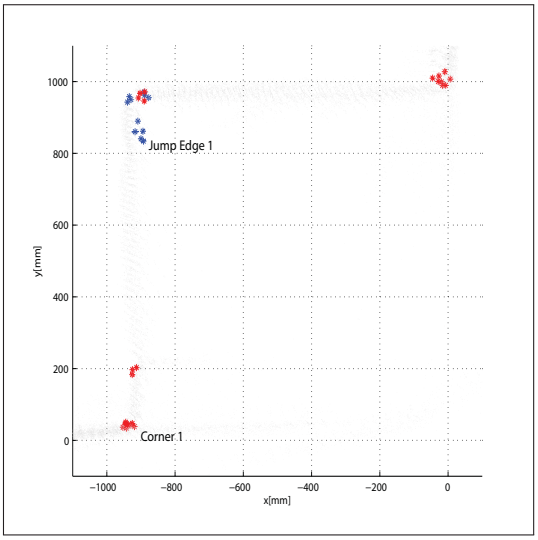
A measurement setup is constructed in an empty corner of a room, as shown in figure A.16. In this corner, four measurement positions are defined (P_1, P_2, P_3 and P_4). The camera is placed at one or more of positions, depending on the object. At each of the measurement positions, the camera is given ten different orientations from 0°to 90°in steps of 10°. Three different objects are placed in the corner: a chair, a box and a pair of pillows.

The results from the feature extraction algorithms are plotted in a top-view map, including 1σ covariance ellipses. From these results, the features are evaluated for their accuracy and error.

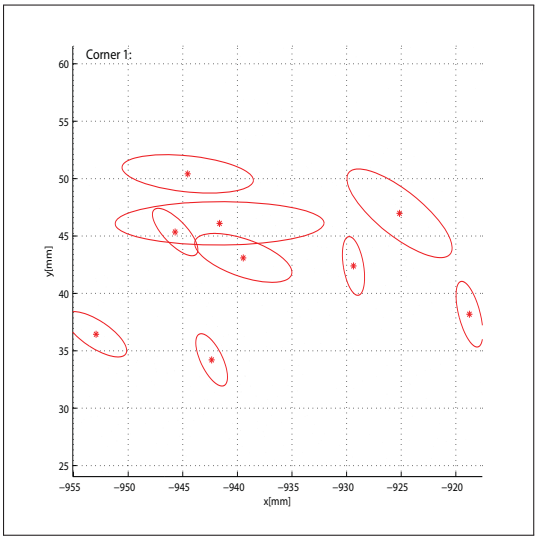
A.5.1 Results

In A.17, the estimates are shown of the features extraction from three datasets of the chair. In the top of the figure A.17, the amplitude image and top-view map is shown. The bottom part of the figure shows the estimated covariances. It becomes clear that that the error is severely underestimated. This indicates that the measurements of the features depend highly on observer position.

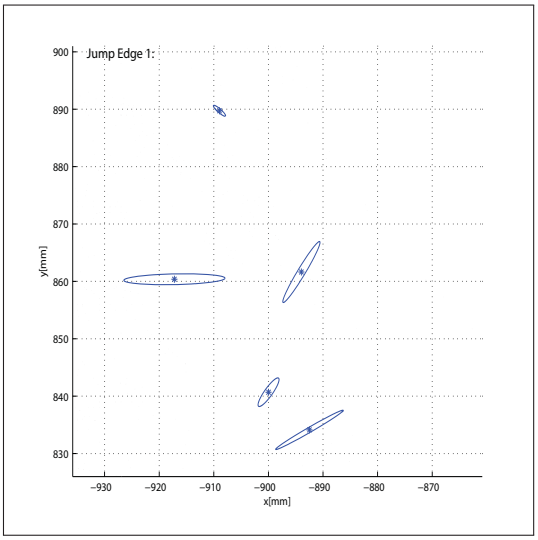
(a) Amplitude Image



(b) Top-view map



(c) Zoomed view of corner



(d) Zoomed view of jump edge

Figure A.17: Initial results for chair

A few causes can be pointed out for this dependability:

- Averaging effect, as pixels measure a surface instead of a point
- Quantization noise
- Errors due to reflections in the scene

As pixels measure a surface instead of a point, an observed corner or jump edge point is always an average of the region around the point. This averaging is proportional to the distance between the observed points, which is in turn proportional to the distance of the points to the camera. This means that this deviation in measured range to a corner, scales linearly with distance. A second cause is the quantization noise, due to the sampling of the surface by the pixels. Sometimes a corner or jumpedge is observed by the center of a pixel, and sometimes on the border of two pixels. This causes the measurement to shift in the direction of the neighboring points, depending on observer position. Note that this effect also scales with the distance between observed points. Reflections in the scene are the third cause and give the same disturbance as with the planes. Corners and jump edges however, are less sensitive to this type of disturbance, as the pixels that are marked as jump edge pixels and corner pixels do not change.

It only affects the measured range for each pixel. From experiment A.3, it is clear that this effect is largest for pixels on the border of the sensor.

Because the first two effects scale linearly with distance, a crude approximation to the error is made by an additional error that scales linearly with distance. The variance of this additional noise is defined by equation (A.7). In this equation, $|\vec{z}|$ is the distance to the feature and γ is the standard deviation of the noise per unit distance.

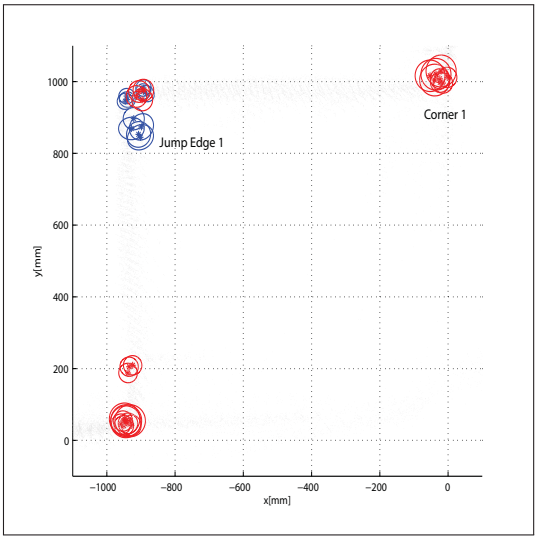
$$\sigma^2_{add} = \gamma^2 \|\vec{z}\|^2 \tag{A.7}$$

The corrected covariance matrix $\mathbf{C}^*_c$ is shown in equation (A.8), where $\mathbf{C}_c$ is the original noise. By trial and error, $\gamma = 0.02$ [m/m] is chosen. This is about equal to the space between the points, scaled with distance.

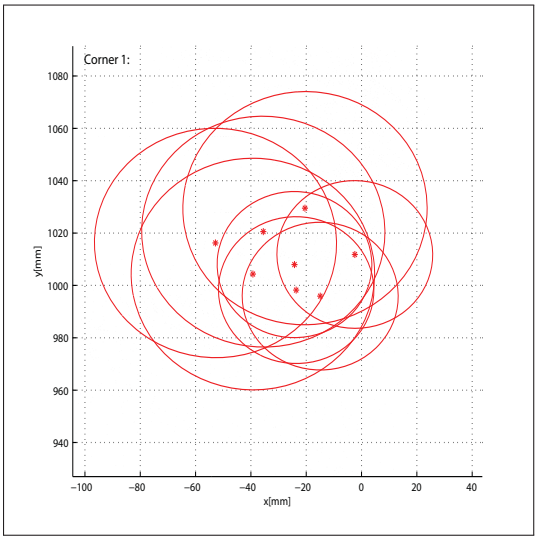
$$\mathbf{C}^*_c = \mathbf{C}_c + \begin{bmatrix} \sigma^2_{add} & 0 \\ 0 & \sigma^2_{add} \end{bmatrix} \tag{A.8}$$

In figures (A.18), (A.19) and (A.20), the results are shown for the new error estimates. For the chair and the box, the estimate of the error nicely corresponds to the actual covariance (a little overestimated). For the irregular shape of the pillows, the covariance is still underestimated. As the covariance is on average set to a workable value, this is neglected for now. Future research might provide a better estimate for the covariance. This requires extensive testing in various domestic environments. If we look at the accuracy in general, it is seen that for the features detected in this experiment, the positioning error is approximately 5 cm at a distance of 2.8 m.

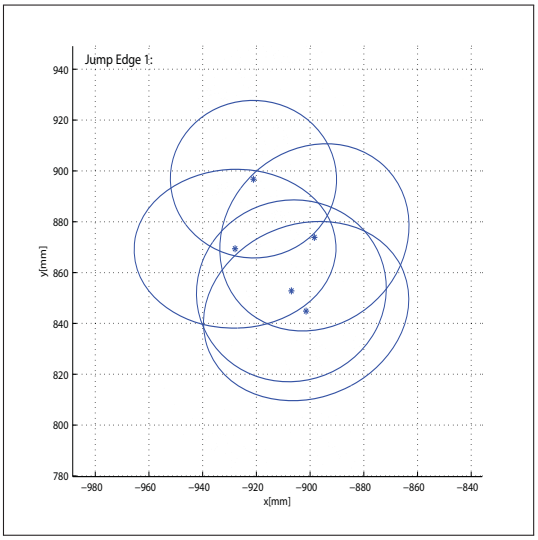
(a) Amplitude image



(b) Top-view map



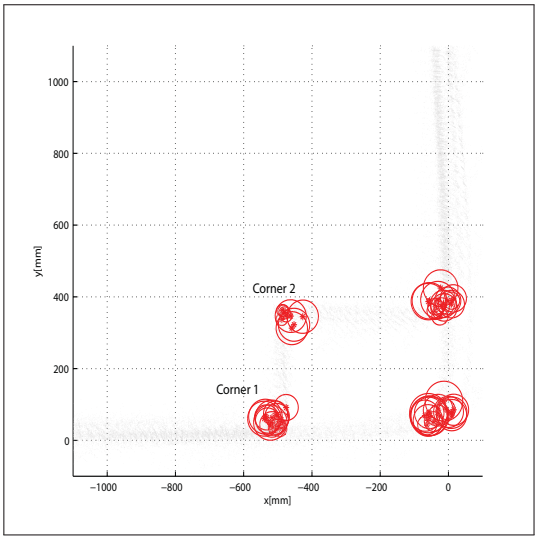
(c) Zoomed view of corner



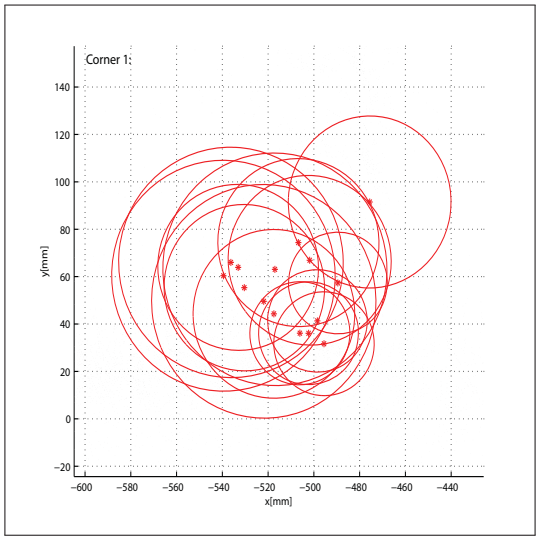
(d) Zoomed view of jump edge

Figure A.18: Results for the chair, using new error estimate

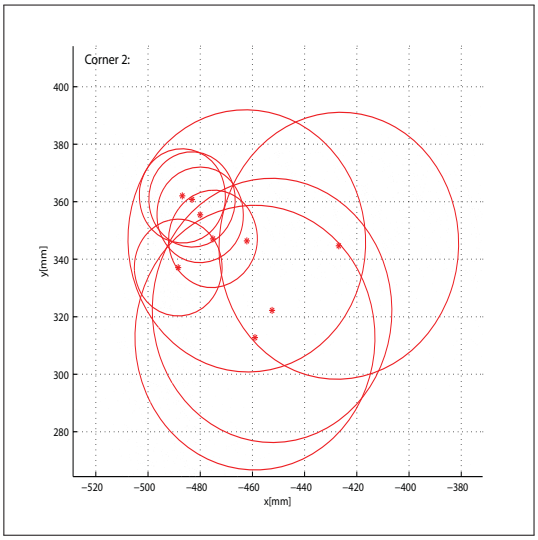
(a) Amplitude image



(b) Top-view map

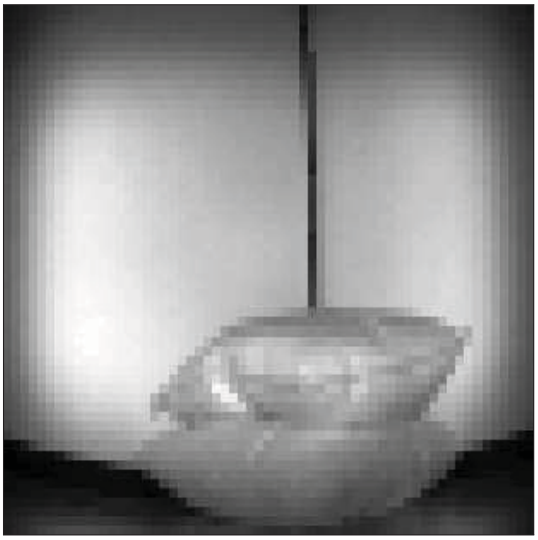


(c) Zoomed view of corner

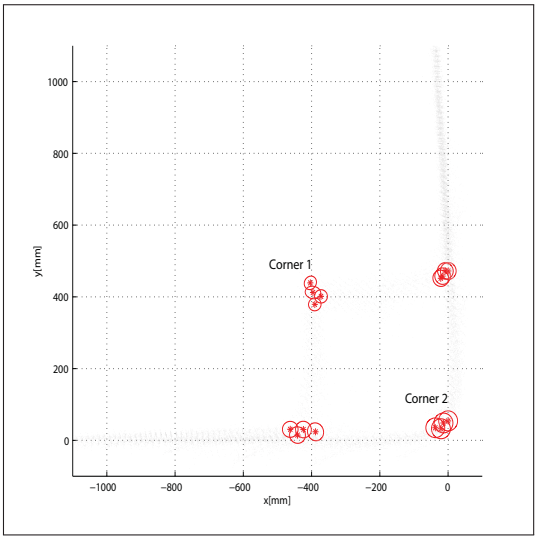


(d) Zoomed view of corner

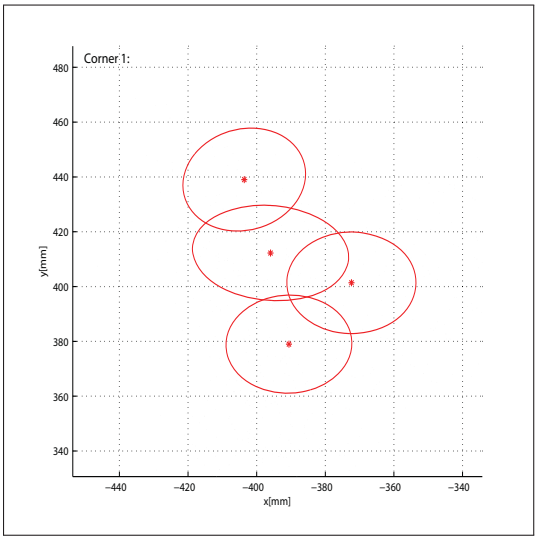
Figure A.19: Results for a box, using new error estimate



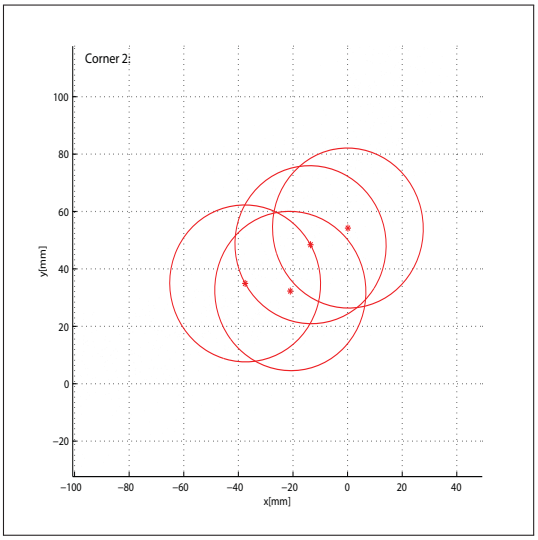
(a) Amplitude image



(b) Top-view map



(c) Zoomed view of corner



(d) Zoomed view of corner

Figure A.20: Results for a pair of pillows, using new error estimate

Bibliography

A. Prusak, O. Melnychuk, H. R. I. S. R. K. (2008), Pose Estimation and Map Building with a Time-Of-Flight-camera for Robot Navigation, *Int. J. Intell. Syst. Technol. Appl.*, **5**, 3/4, pp. 355–364, ISSN 1740-8865, doi:http://dx.doi.org/10.1504/IJISTA.2008.021298.

Bay, H., T. Tuytelaars and L. V. Gool (2006), Surf: Speeded up robust features, in *In ECCV*, pp. 404–417.

Brooks, A. and T. Bailey (2008), HybridSLAM: Combining FastSLAM and EKF-SLAM for Reliable Mapping, in *Workshop on the Algorithmic Fundamentals of Robotics, 2008*.

Censi, A. (2008), An ICP variant using a point-to-line metric, in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 19 –25, ISSN 1050-4729.

D. Chetverikov, D. Svirko, D. S. P. K. (2002), The Trimmed Iterative Closest Point Algorithm, in *In International Conference on Pattern Recognition*, pp. 545–548.

David Droeschel, Stefan May, D. H. P. P. S. B. (2009), Robust Ego-Motion Estimation with ToF Cameras.

Guðmundsson, S. . and K. Lyngby (2006), Robot Vision Applications using the CSEM Swissranger Camera, Technical report, University of Denmark.

Hansen, D., M. Hansen, M. Kirschmeyer, R. Larsen and D. Silvestre (2008), Cluster tracking with Time-of-Flight cameras, in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pp. 1–6.

F. van der Heijden, R.P.W. Duin, D. d. R. D. T. (2008), *Classification, Parameter Estimation and State Estimation - An Engineering Approach Using MATLAB*, ISBN 978-0-470-09013-8.

IEE (2009), MLI Measurements Principles.

IEE (2010a), IEE UTP V3 Protocol.

IEE (2010b), Philips - Introduction to 3DMLI XYZCam Embedded Software and VSCL.

IEE (2010c), Philips AE FST 20100401.

IEE (2010d), XYZCam SW Interface Specifications.

J.J Wang, G. Hu, S. H. G. D. (2009), 3D Landmarks Extraction from a Range Imager Data for SLAM, in *Australian Conference on Robotics and Automation (ACRA), December 2-4, 2009, Sydney, Australia*.

Low, K.-L. (2004), Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration, Technical report, Department of Computer Science, University of North Carolina at Chapel Hill.

Lowe, D. (1999), Object Recognition from Local Scale-Invariant Features, pp. 1150–1157.

Mufti, F. (2010), 3D Image Analysis and Smart Automotive Applications, Canberra, Australia.

S. May, D. Droeschel, D. H. C. W. S. F. (2008), 3D Pose Estimation and Mapping with Time-of-Flight Cameras, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Workshop on 3D Mapping*, Nice, France.

S. Rusinkiewicz, M. L. (2001), Efficient Variants of the ICP Algorithm, in *INTERNATIONAL CONFERENCE ON 3-D DIGITAL IMAGING AND MODELING*.

Sebastian Thrun, Wolfram Burgard, D. F. (2001), *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*.

Thirion, J.-P. (1996), New feature points based on geometric invariants for 3D image registration, *International Journal of Computer Vision*, **18**, pp. 121–137, ISSN 0920-5691, 10.1007/BF00054999.

Thorsten Ringbeck, B. H. (2007), A 3D Time of Flight Camera for Object Detection.

Viet Nguyen, Ahad Harati, A. M. N. T. B. S. (2006), Orthogonal SLAM: a step toward lightweight indoor autonomous navigation, in *In: Proceedings of the IEEE/RSJ Intenational Conference on Intelligent Robots and Systems, IROS*.