University of Twente

EEMCS / Electrical Engineering
Control Engineering

Self Localization of PIRATE Inside a Partially Structured Environment

Twan Mennink

MSc report

Supervisors:
prof.dr.ir. S. Stramigioli
dr. R. Carloni
ir. E.C. Dertien
dr. C.J.A. Pulles

December 2010

Report nr. 027CE2010
Control Engineering
EE-Math-CS
University of Twente
P.O.Box 217
7500 AE Enschede
The Netherlands

Summary

The PIRATE pipe inspection robot which is being developed at the control laboratory of the Twente University, is a robot who's task it is to autonomously navigate through the low pressure gas distribution net in the Netherlands, and to find leaks or other damages. The task of autonomous navigation requires that the robot is able to map the inside of the pipe network, and to localize itself in the network. In this research, an algorithm has been developed which is able to perform this task in several scenarios. To demonstrate this, a sensor vessel has been built, with the necessary sensors to accommodate this task. It is demonstrated that the sensor vessel can localize itself in a straight piece of pipe, and features like 90-degree bends and T-junctions are recognized by the algorithm.

The sensor vessel contains a laser range finder for the detection of the features, a optical encoder attached to one of the wheels which measure travel through the pipe and a acceleration sensor to detect the roll angle of the vessel. A particle filter is chosen as the technique for the self-localization, while a grid map is used to detect features. Template matching is used to match templates generated from a simulated environment, to features mapped in the real environment.

Preface

“The pleasure of finding things out.” This is a sentence by famous physicist Richard Feynmann. I find it describes well, what my motivation was to decide to study further on a master education after I finished my HBO. I’ve always found it joyfull to study a certain subject, and then to find out the deeper mathematical rules describing it. With these deeper rules, often comes more understanding about the subject, and the knowledge to predict certain events. In my years at the university, I not only gained a lot of knowledge by studying subjects, but also participated in a large learning experience called the Solar Team Twente. By experiencing this all, I now take with me a large set of tools which I can use in my further career, and hopefully I will also be able to add new tools to them.

The time has now come, to finish my study. The last assignment to complete my education was to do a research and write a master’s thesis about it. This thesis is the end result of that. During the past six months, I worked at the control laboratory on this project, and I would like to thank a few people. First of all I would like to thank my exam committee, Prof. Stefano Stramigioli, Dr. Raffaella Carloni and Ir. Edwin Dertien. Special thanks go out to my daily supervisor Edwin Dertien, for guiding me towards the end result, and for helping me with the thesis. Furthermore I would like to thank Marcel and Gerben, for their advice on building the various parts of my project. Then there are the fellow students at the lab, I would like to thank you all for the many, many fine coffee breaks.

Last, but not least, I would like to thank my parents. They have always supported me, and without them I would have never been able to get this job done, thanks a lot for that!

Twan Mennink
Enschede, December 2010

Contents

1 Introduction

1.1 Background Information: PIRATE

The PIRATE project is a research project which started at the Control Laboratory of the University of Twente in 2006. The aim of this project is to develop a fully autonomous robot for gas distribution pipe network inspection. The gas distribution network in the Netherlands needs to be inspected, to find possible gas leaks. These leaks can be caused by for instance tree roots, or during excavation work. Nowadays, many of these leaks are reported by the public, or detected using sniffing equipment (Dertien (2006)). Figure 1.1 show a picture of the first prototype of PIRATE.

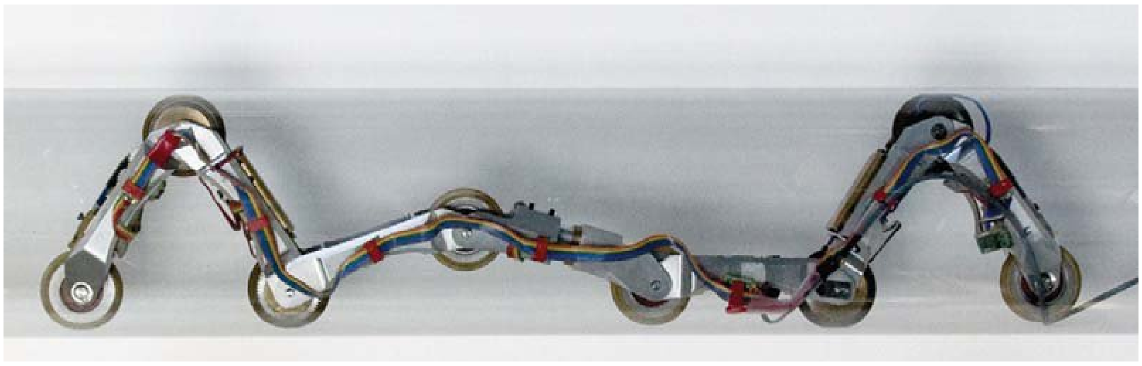


Figure 1.1: The PIRATE pipe inspection robot.

Eventually, the PIRATE robot must be able to autonomously navigate itself through the pipe network, and must be able to detect leaks and other irregularities in the pipe network. A typical scenario of a PIRATE mission would be to insert the robot into the pipe network, and pick it up a few days later on another location. The robot would have some memory card containing a map of the pipe network and the locations of problems.

1.1.1 Previous work

Many research projects have already been done on the PIRATE subject, such as Dertien (2006), who drew up the system specifications, Reemeijer (2010), who made algorithms for maneuvering around bends and T-junctions, Drost (2009), who made a optical pipe profiling system, and many more. A list of research projects done on the PIRATE subject is shown here:

- Dertien (2006) Drew up the initial system specifications.
- Dertien (2007) Electronics design of mainboard.
- Ansink (2007) Electronics design of the motorcontrollers.
- Vennegoor op Nijhuis (2007) Mechanical design of the first prototype.
- de Boer (2008) Optimization of parameters of PIRATE.
- Drost (2009) Developed an optical pipe profiling system.
- Burkink (2009) Mechanical design of a Propulsion unit for PIRATE.
- Reemeijer (2010) Developed maneuvering algorithms.

And at this moment, Doggen (2010) is working on a wireless state feedback and control system for pirate.

1.2 Problem Statement

This research is based upon the fact that the PIRATE pipe inspection robot will have to navigate its way through the gas distribution network fully autonomous. Autonomous navigation of robots is a wide topic of ongoing research, and many examples exist in literature. For PIRATE however, such a system does not yet exist. For PIRATE to be able to navigate itself autonomous, PIRATE will first have to simultaneously localize itself and map the environment. This is called a SLAM (Simultaneous Localization and Mapping) problem, and this is the subject of this research.

1.2.1 Detailed Problem Description

The PIRATE prototype currently does not yet have external sensors of any kind. Prior to this research, it was not yet known what sensors were needed to perform SLAM inside a pipe network. Therefore, the choice was made to build a separate sensor vessel, which can be used to gain knowledge about the SLAM problem. Typical questions that arise:

- What sensors are needed for pipe network self localization and mapping?
- What kind of software algorithm is suitable for this task?
- How much processing power is needed for this task?
- How can a map of the pipe network be made?

To give an answer to these questions, it is chosen to focus this research on a sub-problem of the entire localization and mapping problem. This sub-problem involves the self-localization of a sensor vessel inside a straight piece of pipe, and later expand this to detecting and mapping a feature at the end of the straight piece of pipe, such as a T-junction or a 90-degree bend. Developing the sensor vessel and algorithms to solve this problem will give a good insight into these questions and will provide a solid basis to solving the entire problem.

1.3 Goal

The goal of this research is to develop an algorithm, which is suitable to simultaneously localize PIRATE and map the inside of a pipe network, and to demonstrate this by designing and building a sensor vessel equipped with the necessary sensors to run the algorithm.

1.4 Report Outline

The report is structured as follows:

Chapter 2 gives background information about SLAM and a literature study about the topic of SLAM. The chapter is divided into three sections. The first Section describes the basic SLAM problem. The second Section describes the current state of the art in SLAM. The third Section summarizes the literature study on SLAM.

Chapter 3 describes the design of the sensor vessel, selection of the sensors and software needed for the data acquisition. Chapter 3 contains five sections. In the first Section the proposed system is described, and the system is divided into three functional parts: Hardware, Data Acquisition and SLAM Algorithms. The next three sections describe the design of these parts. The fifth section contains the conclusions of the design.

Chapter 4 describes the redesign of the custom built laser range finder, which is used on the sensor vessel. A separate chapter is devoted to this subject, because the development of this sensor was a substantial part of this research. The chapter is divided into 5 sections. These sections describe the hardware optimization, the configuration analysis, the image processing, the calibration and the verification of the sensor.

Chapter 5 contains the Design of the actual SLAM Algorithms. There are six sections in this chapter. Section 1 describes the vessel state estimation using a particle filter, Section 2 describes the observation model using ray casting, Section 3 describes the mapping of the pipe network, Section 4 and 5 describe the extension of the observation model to cope with 90-degree bends and T-junctions, Section 6 describes the template matching used to recognize features inside the pipe.

Chapter 6 Gives the simulation results obtained from testing the vessel state estimation and pipe mapping. Section 1 shows the results from performance evaluation of the particle filter and Section 2 shows the results from simulated mapping of pipe features.

Chapter 7 Shows the results obtained from the actual tests of the SLAM algorithm using the sensor vessel inside a gas pipe. There are two sections in this chapter, Section 1 describes the mapping results obtained, and Section 2 gives the conclusions.

Chapter 8, contains the conclusions of the research, and recommendations for future work. A reflection is made on the goal of this research, and conclusions are drawn discussing whether or not the goal was met.

2 SLAM: Literature Review and Background Information

This chapter introduces the main topic of this research, simultaneous localization and mapping (SLAM). Different SLAM approaches in literature are reviewed and the problem is explained.

2.1 Basic SLAM problem

The aim of SLAM is to detect the state of a robot inside its environment, and to simultaneously make a suitable map of this environment. This in principle is a difficult problem, because to make a good estimate of a robot state, the environment needs to be known, but to make a good estimate of the environment, the robot state needs to be known. Somehow, this must be translated into an iterative process in which a better estimate of the environment leads to a better estimate of the state and vice versa. Figure 2.1 shows a schematic representation of such a process. In this schematic, the robot state (x_k) interacts with the environment (world map m), and the measurements (z_k) are produced as a result of this particular state and environment. Now after each measurement, the estimated map/model of the world is updated from the measurement data and the previous world and state estimates. After that, the state estimate is updated from the same measurement data and the previous state estimate and world estimate. A stochastic approach to obtain a solution to this problem is called recursive bayesian estimation. It is described in the next subsection.

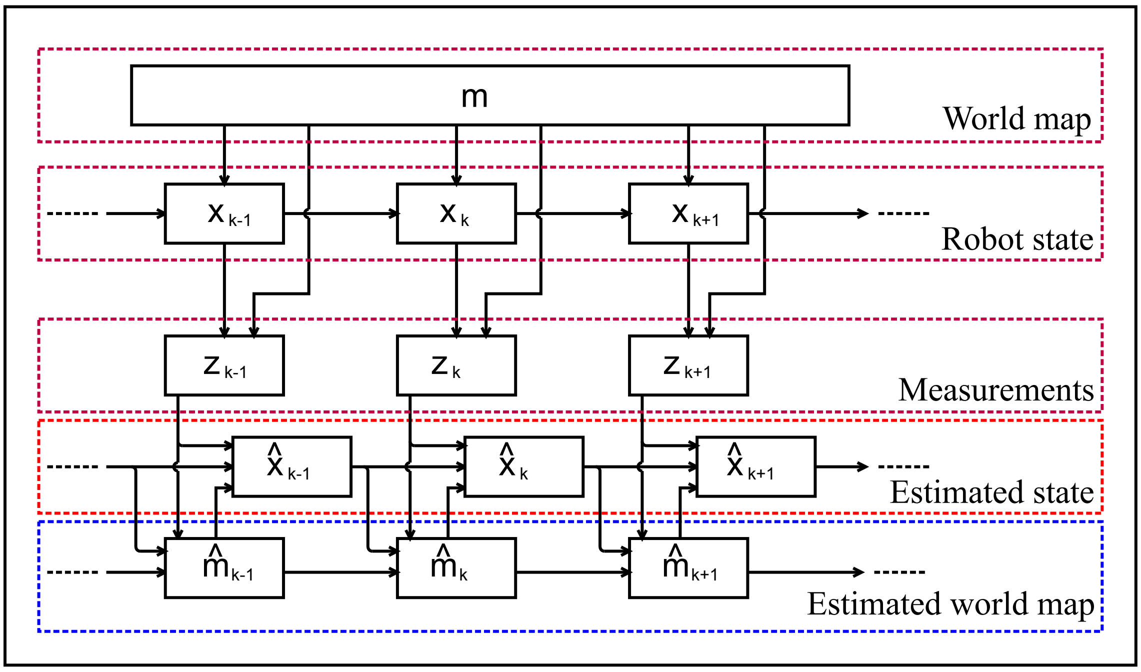


Figure 2.1: The iterative process of Simultaneous Localization and Mapping

2.1.1 Recursive Bayesian Estimation

To explain Recursive Bayesian Estimation, first the estimation of the environment m is left out. So the problem is to obtain a estimate for the state x_k, given the sequence of all measurements $Z_k = \{z_1, z_2, \ldots, z_k\}$. The state of this system evolves through time according to equation 2.1, in which f_k could be a nonlinear function and v_k is the process noise. it is assumed that f_k is known.

$$x_k = f_{k-1}(x_{k-1}, v_{k-1}) \tag{2.1}$$

Also measurements z_k are taken at each sampling interval according to Equation 2.2, in which h_k is a known possibly nonlinear function.

$$z_k = h_{k-1}(x_{k-1}, v_{k-1}) \tag{2.2}$$

To obtain an estimate of x_k, the posterior $p(x_k|Z_k)$, is required, so that for instance the minimum mean-square error (MMSE) estimate of x_k can be calculated as in Equation 2.3.

$$E[x_k|Z_k] = \int x_k \cdot p(x_k|Z_k)dx_k \tag{2.3}$$

The recursive bayesian estimation algorithm consists of a "prediction" step and an "update" step.

2.1.1.1 Prediction Step

First the prediction step will be explained. Suppose the pdf $p(x_{k-1}|Z_{k-1})$ is available. Then the prediction stage involves calculation of $p(x_k|Z_{k-1})$, which is done via the Chapman Kolmogorov equation as in Equation 2.4. $p(x_k|x_{k-1}, Z_{k-1})$ reduces to $p(x_k|x_{k-1})$, because according to Equation 2.1 the current state x_k only depends on the previous state x_{k-1} and the statistics of v_{k-1} and not on measurements Z_{k-1}.

$$\begin{aligned} p(x_k|Z_{k-1}) &= \int p(x_k|x_{k-1}, Z_{k-1})p(x_{k-1}|Z_{k-1})dx_{k-1} \\ \Rightarrow p(x_k|Z_{k-1}) &= \int p(x_k|x_{k-1})p(x_{k-1}|Z_{k-1})dx_{k-1} \end{aligned} \tag{2.4}$$

2.1.1.2 Update Step

The update step is now defined by Equation 2.5. The probability density function $p(z_k|x_k)$ is derived from the measurement equation 2.2.

$$p(x_k|Z_k) = \frac{p(z_k|x_k)p(x_k|Z_{k-1})}{p(z_k|Z_{k-1})} \tag{2.5}$$

So by this, the pdf $p(x_{k-1}|Z_{k-1})$ is evolved to $p(x_k|Z_k)$. This provides an analytical method to recursively estimate the state x_k of a dynamic system with noise.

2.1.1.3 Analytic Solution

In many cases, an analytic solution cannot be found to to the recursive bayesian estimation problem. In case the system dynamics are linear, and both process and measurement noise are additive gaussian, the problem is solved analytically with the Kalman filter. In the nonlinear case, various approximations exist, such as extended Kalman filters, which linearize the system, and particle filters, which use monte carlo integration methods. For further reading on these subjects, Ristic et al. (2004) is an excellent source of information.

2.1.2 Particle Filter

A particle filter, (Also called Sampling Importance Resampling filter), is often used in literature to solve a SLAM problem. The operation of such a filter is now explained. First the concept of importance sampling is introduced.

2.1.2.1 Importance Sampling

Monte Carlo Sampling is a technique used to approximate a density function $f(x)$ by a finite set of N samples $X = \{x_1, x_2, \ldots, x_N\}$ with associated weights $W = \{w_1, w_2, \ldots, w_N\}$. Given these samples and weights, the function $f(x)$ is approximated by equation 2.6.

$$f(x) \approx \sum_{i=1}^{N} w_i \delta(x - x_i) \tag{2.6}$$

Suppose one wants to evaluate an integral of the form:

$$F = \int g(x) f(x) dx \tag{2.7}$$

By substituting the sampling approximation for $f(x)$ in Equation 2.7, then an approximation of Equation 2.7 can be found by:

$$F \approx \int g(x) \left[\sum_{i=1}^{N} w_i \delta(x - x_i) \right] dx = \sum_{i=1}^{N} g(x_i) w_i \tag{2.8}$$

Now Suppose a sampled representation of a pdf $p(x_k|Z_k)$ is needed. Often one cannot sample directly from that pdf. In case $p(x_k|Z_k)$ cannot be sampled directly from, but $p(x_k|Z_k)$ can be evaluated for a given x_k^i, then it is still possible to generate a set of support points $\{x_k^i, i = 1 \ldots N\}$ with associated weights$\{w_k^i, i = 1 \ldots N\}$ representing $p(x_k|Z_k)$. This is done by taking a different pdf $q(x_k|Z_k)$ from which samples $\{x_k^i, i = 1 \ldots N\}$ are drawn. The weights are now proportional to:

$$w_k^i \propto \frac{p(x_k^i|Z_k^i)}{q(x_k^i|Z_k)} \tag{2.9}$$

The only prerequisite on q is that it has the same support as p. $(p > 0 : q > 0)$. This method is called importance sampling and $q(x_k|Z_k)$ is called the importance density, or in particle filtering, is also often called proposal density. Figure 2.2 shows how the importance sampling works. Samples are drawn from the importance density and all have equal weight. (sum of all weights is one). Now the weights are updated according to Equation 2.9 and are normalized. (total sum is 1).

2.1.2.2 Sequential Importance Sampling

Sequential importance sampling makes use of importance sampling to solve the recursive bayesian estimation problem. Let's say one wants to have a sampled representation of $p(X_k|Z_k)$. This can be done by importance sampling, using a proposal density $q(X_k|Z_k)$ which factors as:

$$q(X_k|Z_k) = q(x_k|X_{k-1}, Z_k) q(X_{k-1}|Z_{k-1}) \tag{2.10}$$

Suppose a set of samples drawn from $q(X_{k-1}|Z_{k-1})$ is already available, and a set of weights $w_{k-1}^i, i = 1, 2, \ldots N$ is also available. The set of weights with the samples $X_{k-1}^i, i = 1, 2, \ldots N$ is a sampled representation of $P(X_{k-1}|Z_{k-1})$. The probability density function of the trajectory of x up to time $k-1$, given all measurements z up to time $k-1$. Ristic et al. (2004) shows that if one now draws samples:

$$X_k^i \sim q(x_k|X_{k-1}, Z_k) \tag{2.11}$$

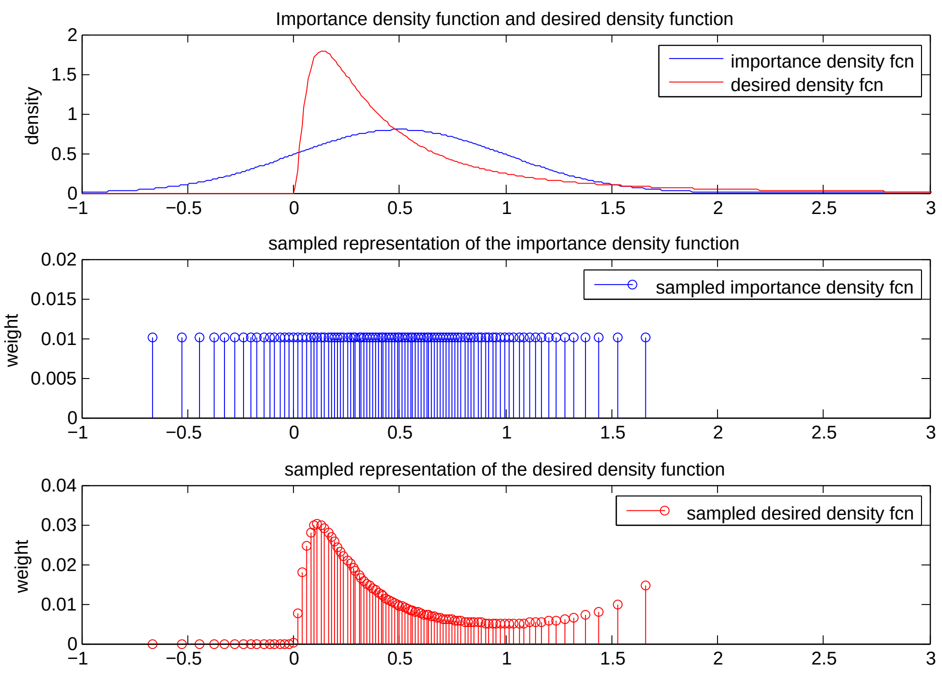


Figure 2.2: Example of importance sampling. The desired density is lognormal distributed, the importance density is normal distributed.

and the markov chain order 1 assumption is made, meaning:

$$q(x_k|X_{k-1}, Z_k) = q(x_k|x_{k-1}, z_k) \tag{2.12}$$

Then the new weights $w_k^i, i = 1, 2, \ldots N$ can be calculated:

$$w_k^i \propto \frac{p(z_k|x_k^i)p(x_k^i|x_{k-1}^i)}{q(x_k|x_{k-1}, z_k)} \tag{2.13}$$

This gives a method of taking a previous set of samples and weights, and updating them with new measurement information to a new state. The importance density $q(x_k|x_{k-1}, z_k)$ must be well chosen, because if the density function is too wide for instance, then many samples will get low weights assigned, which makes a poor approximation. Also, one must be able to draw samples from the importance density, as well as evaluate the pdf for that particular sample. So sequential importance sampling gives us a method to get a estimate of the state given all measurements, a problem with this method exists however. It is observed in literature that after only a few time steps, the weights of all particles (or samples) but one will converge to zero. This means that after a few time steps only one particle will be effective.

2.1.2.3 Sequential Importance Resampling

The Sequential Importance Resampling (SIR) filter uses sequential importance sampling to solve the recursive bayesian estimation problem. The SIR filter solves the problem of particle degeneracy by resampling after each time step. Resampling is a technique to redistribute all particles according to the existing sampled pdf where more particles are assigned to high probability areas. A high variety of resampling schemes exists in literature, for a variety of situation. A few strategies are: Residual resampling, Stratified resampling and Multinomial resampling. The paper Douc (2005) compares a number of these resampling schemes.

2.2 SLAM state of the art

The SLAM problem is a wide topic of ongoing research. The most research done on this subject concerns the 2D SLAM problem, in which a robot moves through a flat 2D environment, for example Grisetti et al. (2007a) and Schroeter and Gross (2008) work on such SLAM type problem. In this 2D world, the robot state is represented by three parameters, namely the coordinates x, y and φ. x and y represent the position on the map of the robot, and φ represents the orientation of the robot. To represent a map of the environment, an occupancy grid map is often used (Grisetti et al. (2007a), Lu and Milios (1997), Pandey et al. (2007)). This is a grid with cells, and each cell's value indicates if the cell is occupied or not. Laser range finders are particularly well suited to generate such an occupancy grid.

2.2.1 Sensors Often used for SLAM

On most 2D SLAM problems, two types of sensors are used. First, some form of odometry is present, that measures for instance wheel rotations. Secondly, a range finder in some form is often present. Holz et al. (2008) uses a IAIS 3DLS Laser range finder, which is based on the SICK 2D laser range finder. This range finder uses a straight laser line ans scans up and down. Elinas et al. (2006) use a pair of stereo cameras for mapping indoor environments. Fairfield et al. (2006) use a sonar range finder as their main sensor.

2.2.2 Filters Used in SLAM

Various types of filters are used in SLAM research, the most notable are particle filters and Extended Kalman filters. If extended Kalman filters are used, a landmark based approach is often used. Landmarks are modeled as coordinates on the map, and their position is then estimated with a certain variance, for instance Newman (1999) uses landmark based maps. Particle filters are more often used in combination with an occupancy grid map.

2.2.3 Map Consistency

A SLAM algorithm will have to construct a map consistently. Data from for instance a laser range scanner is combined with odometry data. A problem which is often encountered if all measurements are integrated in a global map, is that this will result in errors accumulating. A straight corridor can become curved, for instance. To overcome this, a map can be restored when a loop in the map is encountered. The algorithm can then correct the map to make sure both measurements of the same locations actually overlap.Lu and Milios (1997) uses such techniques. For these techniques, it is required to keep a trajectory of the robot, and the map as a function of that trajectory. If the trajectory estimate changes, then the map, which is a combination of local maps will also be updated. Another approach is to use some constraints on the map. For instance one could pose the constraint that only 90-degree angles in walls are permitted. This would not require keeping all local maps and trajectories. Rodriguez-Losada et al. (2006) uses geometric constraints to keep the map consistent.

2.2.4 Problem of High Dimensionality

In SLAM, the state of the system is the combined state of the robot and the state of the map. Either a grid map or a landmark based map results in a state vector of very high dimensionality. A map can often contain thousands or millions of landmarks, or grid cells. This results in joint probability density functions of very high dimension. In case a particle filter is to be used, it would need a unpractical amount of particles/samples to approximate this high-dimensional joint probability function. One way to overcome this, is to use Rao-Blackwellization. Elinas et al. (2006) uses this, as well as Grisetti et al. (2007b). The objective of SLAM is to estimate the robot trajectory $x_{1:k}$ and map m, given the sequence all odometry measurements $u_{1:k}$ and range finder observations $r_{1:k}$. This is:

$$p(x_{1:k}, m|u_{1:k}, r_{1:k}) \tag{2.14}$$

The key aspect of Rao-Blackwellization is to factorize this into:

$$p(x_{1:k}, m|u_{1:k}, r_{1:k}) = p(x_{1:k}|r_{1:k}, u_{1:k})p(m|x_{1:k}, r_{1:k}) \tag{2.15}$$

$p(x_{1:k}|r_{1:k}, u_{1:k})$ is a localization problem, which is solved by a particle filter, and $p(m|x_{1:k}, r_{1:k})$ takes the particles of the state trajectories as inputs. The Rao Blackwellized particle filter for SLAM does need to keep a map with every particle though, but this then also has the advantage that multiple hypothesis are kept, so a loop-closing is automatically performed. Only particles which close a loop correctly "survive", and during a resampling step they will be copied.

2.3 Conclusions on SLAM in Literature

It has been found that many examples exist in literature that solve the 2D, 3DOF SLAM problem satisfactory. Two filtering techniques are most used, Particle filters and Extended Kalman filters. Also, two map representation techniques are most used, occupancy grid mapping and landmark based maps. The main differences between Extende Kalman Filters (EKF) and Particle Filters (PF) are displayed in Table 2.1. In Table 2.2 the differences between Occupancy Grids and Landmark based maps are displayed. To use a landmark based map, often a camera or a camera stereo pair is used to detect SIFT keypoints which are translated into landmarks. The method is less suited if sparse sensor data, from for instance a sonar, or very noisy data is available.

PF	EKF
handles complex systems	Low computational load
good noise immunity	high demand on noise parameters
handles non-linear systems	linearizes system
Keeps multiple hypothesis	Can lose "track"
High computational costs	linear & gaussian: KF=optimal

Table 2.1: Properties of Particle Filters versus Extended Kalman Filters

Occupancy Grid	Landmarks
Fixed spatial resolution	landmarks can be anywhere
Memory fixed	Memory depends on number of landmarks
no feature recognition required	Must recognize landmark features
difficult to match subsets	Easy to match subsets

Table 2.2: Properties of Occupancy grids versus Landmark based maps

3 Design of the Sensor Vessel

To perform SLAM inside a small pipe network, a sensor vessel had to be designed and built. This chapter describes the selection of the sensors used on the vessel, and the design of the vessel itself.

3.1 System description

The task of the sensor vessel is to move through a straight section of pipe, and to map the location of features like bends and T-junctions. This is the first step towards a fully autonomous PIRATE pipe inspection robot. It is likely that in the future a new version of the PIRATE prototype will be built. It has to be taken into account that the sensors, hardware and software used are suitable for implementation on a stand-alone low-power platform. The focus of this research is to develop algorithms for performing SLAM in a pipe network, therefore it is not required that the processing is done embedded. It is chosen to divide the entire sensing system to perform SLAM into three parts. Hardware, Data-acquisition and SLAM algorithm. Figure 3.1 shows the overview of this system. The function of each part is described in Table 3.1. In the next sections, the design of these three parts is described.

Hardware
Data acquisition
SLAM Algorithms
Figure 3.1: Overview of the system design

Component:	Function
Hardware	To move through a straight section of pipe, and generate sensor data.
Data-acquisition	To read the sensor data, to present the data to the SLAM Algorithms
SLAM Algorithms	To localize the vessel inside the pipe, to map the pipe features.

Table 3.1: Function of the different components of the sensing system

3.2 Hardware Design

The hardware of the sensing system consists of a mechanical setup which holds all sensors used, and is able to move through the pipe, and the sensors itself with interfacing electronics. First the selection of the different sensors used is described.

3.2.1 Sensor Selection

As pointed out in Subsection 2.2.1, sensors used for SLAM are often a combination of a visual type sensor such as a laser range finder and odometry such as wheel encoders. For this project, it is chosen to use three types of sensors. A Laser range finder, a rotary wheel encoder and a Inertial Measurement Unit. When moving inside a straight pipe, these three sensors combined can be used to obtain a state estimate.

3.2.1.1 Laser Range Finder

A laser range finder is a sensor which emits a laser pattern, and using a optical sensor such as a camera, detects the location of this pattern in 3D. In most 2D SLAM applications, the laser range finder is pointed at the floor ahead of the robot, to detect obstacles which are placed on that floor. Inside a pipe, however, there is no floor. The system needs to scan the entire pipe-wall for obstacles or features. Drost (2009) has built such a system at our laboratory. It is a laser range scanner which projects a circle on the pipe wall. This system still has some drawbacks, most notably are its physical size and the speed of the data processing. Therefore, this sensor is redesigned. A detailed description of the redesign can be found in Chapter 4. If the sensor is placed inside a perfectly round, straight section of pipe, then the sensor provides enough information to detect the pose of the sensor with respect to the pipe axis. However, it cannot detect rotations around the pipe axis, and also cannot detect translational motion along the pipe axis.

3.2.1.2 Inertial Measurement Unit

To overcome the problem of not being able to detect rotations around the pipe center axis, a inertial measurement unit is added to the sensor array. The sensor used is a Xsens MTi. (www.xsens.nl). This sensor has the capability estimating its entire orientation, but this relies on the earth-magnetic field. Inside the gas network, which may be metal pipes, this will not work properly. Also, the PIRATE Mainboard contains acceleration sensors. It is therefore chosen to only use the acceleration data of the Xsens unit. This makes the sensing system more compatible with future PIRATE designs. Using the acceleration data, now a distinction can be made between two orientations which are rotated around the pipe axis. Figure 3.2 shows a photo of this sensor.

Figure 3.2: Xsens MTi Inertial Measurement Unit (IMU)

3.2.1.3 Rotary Wheel Encoder

To be able to also measure translational motion along the pipe axis, a optical encoder is fitted to one of the wheels of the sensor vessel. Now three sensors are available which make it possible to sense all vessel states inside a straight pipe. Figure 3.3 shows the location of the wheel encoder.

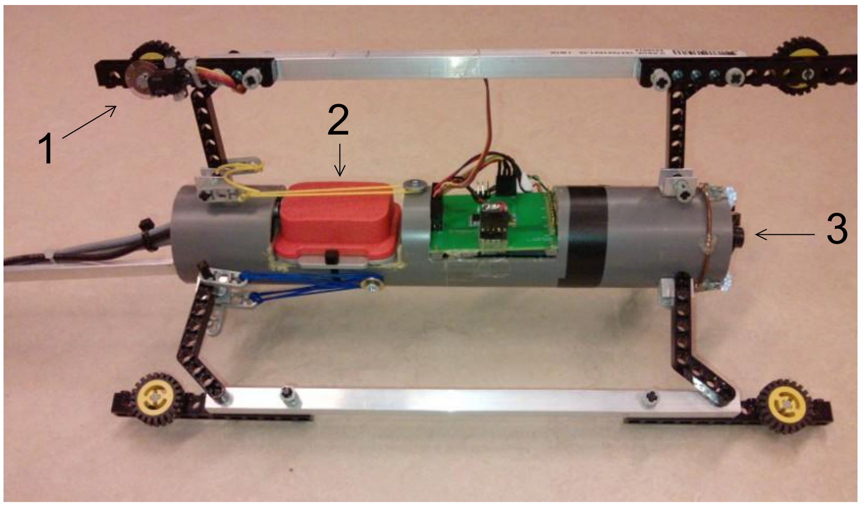


Figure 3.3: Sensor vessel outside pipe with 1: Wheel encoder, 2: IMU, 3: Laser range finder

3.2.2 Mechanical Design of the Sensor Vessel

A prototype style sensor vessel has been built for this project. As said, the sensor vessel contains a Inertial Measurement Unit (IMU), one wheel is equipped with an optical encoder and a custom built laser range finder built upon the work of Drost (2009). A few pictures of the built vessel can be seen in Figure 3.3 and Figure 3.4.

The chassis of the sensor vessel of made to hold all the sensors. A very basic design is chosen, the chassis just consists of a piece of PVC pipe. The suspension has been made such, that the vessel approximately centers itself inside a pipe. The minimum diameter of pipe in which this vessel fits is $120[mm]$.

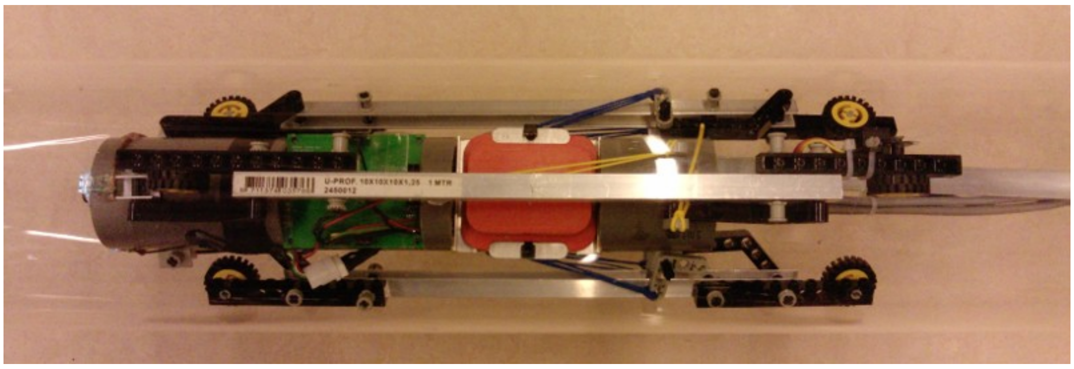


Figure 3.4: Photographs of the sensor vessel inside the pipe,where the suspension is fully impressed.

3.2.3 Interface electronics

The interfaces of the sensor vessel to the outside world are shown schematically in Figure 3.5. A personal computer connects with two serial ports and one USB port to the vessel. The custom made serial communication protocol with the microcontroller is described in Appendix B.1.

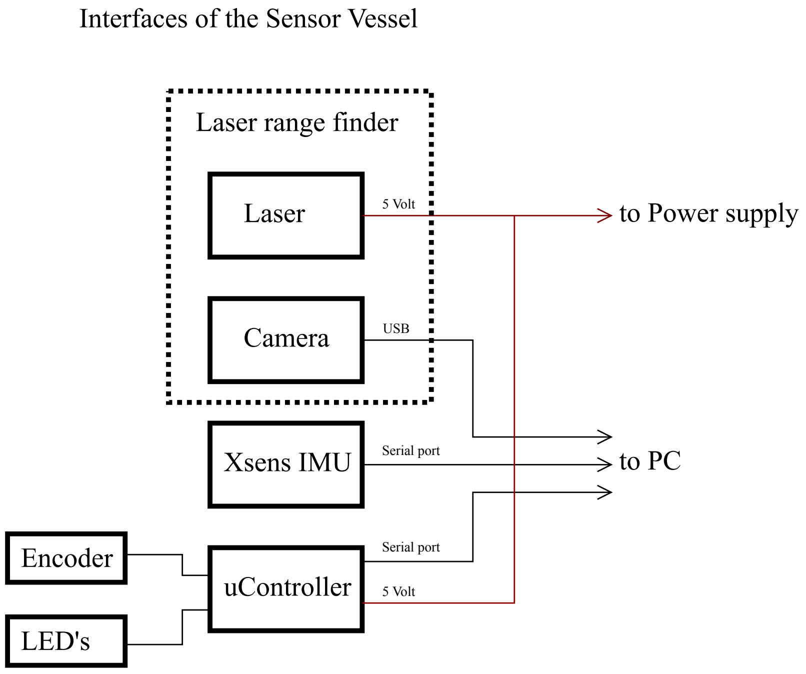


Figure 3.5: The interfacing between the sensor vessel and PC

3.3 Data Acquisition Software

The data acquisition software is the piece of software which reads out all sensors on the sensor vessel, and presents this data in a structured manner to the SLAM algorithm. It is chosen the divide the data acquisition software into a server and client which communicate via a TCP/IP network connection. The function of the server is to communicate with the various sensors, and to give this data in a simple manner to the client. The function of the client is to receive the data from the server and to present it to the SLAM algorithm in a struct. Figure 3.6 shows the connection in this data acquisition software.

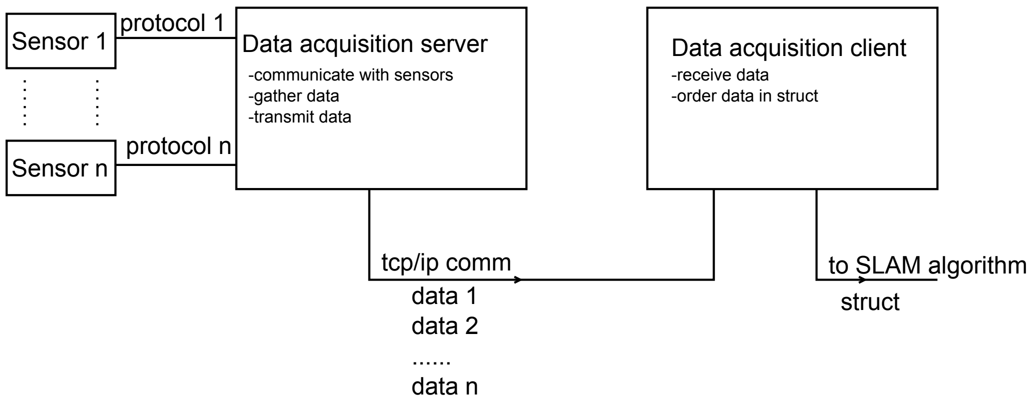


Figure 3.6: Schematic representation of the function of the data acquisition software

3.3.1 Basic Operation of the Data Acquisition Server

The data acquisition server makes use of different threads to do its tasks. A total of four processes run simultaneous. The function of the threads are:

- Image processing: Read frames from camera and extract the laser curve in polar coordinates.
- Xsens: Continuously receive the Xsens data and place in a fifo buffer. Analyse buffer content for valid data frames.
- Microcontroller: Wait for data from encoder readout.
- UDP Server: The network thread, which waits for messages from the client.

So the threads continuously gather data from the sensors, and when a client asks for data, the server gives the most up-to-date sensor values. A detailed description of the threads used and the different classes can be found in Appendix B.2. Figure 3.7 shows the basic communication between the server and the client.

3.3.2 Data Acquisition Client

The client which calls the server periodically for sensor data is implemented in Matlab. Matlab has some built in functions to handle network UDP traffic, so these functions are used. The communication protocol between the client and the server is very basic. The client sends a one-byte value, ranging from 1 to 5, which indicates the type of data requested. The server then responds with a message with the raw data. No frame headers or such are used. The data-acquisition client also uses calibration data to calculate 3D information from the polar coordinate laser range finder data. This is done by a mapping table and details about this can be found in Chapter 4. The source code of the data acquisition client can be found in the files listed in Table G.2. The program is started by running getNetwork.m, listed in Table G.2. The program will store all measured data in a struct array “Sample”, containing the entries shown in Table 3.2:

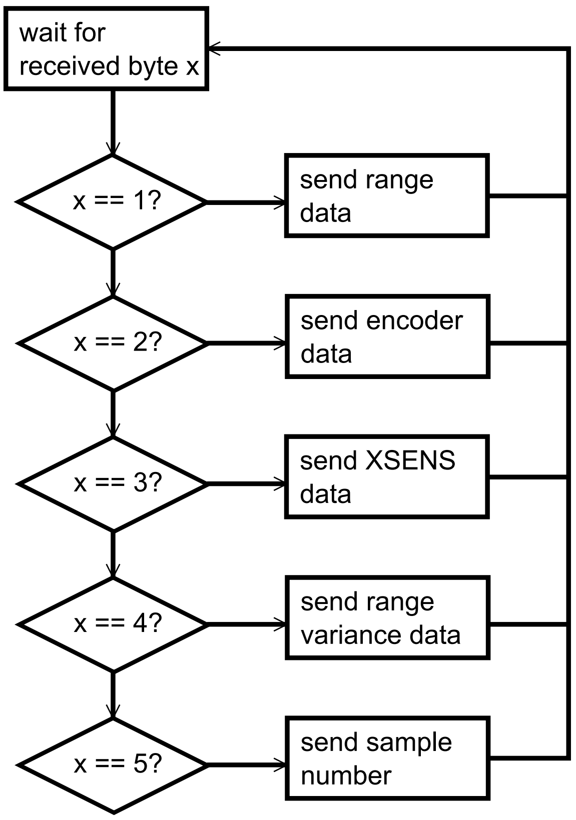


Figure 3.7: Basic mode of operation of the UDP server. A thread from the main program continuously listens for data and replies.

struct entry:	description:
x	calculated world x-coordinates, laser range finder $[mm]$
y	calculated world y-coordinates, laser range finder $[mm]$
z	calculated world z-coordinates, laser range finder $[mm]$
R	vector with the acceleration data from Xsens $[m/s^2]$
std	standard deviation of laser range finder datapoints $[pixels]$
s	encoder counter value, calculated in mm of travel $[mm]$

Table 3.2: Description of the struct array Sample, containing pre-processed measurement data

3.4 SLAM Algorithm

The slam algorithms used in the sensing system is a combination of vessel state estimation using a particle filter, and a pipe map algorithm using a variation to the occupancy grid. The design and implementation of these algorithms are described in Chapter 5

3.5 Conclusions

In this chapter, the overall design of the sensor vessel with the used software is described. Now a sensor vessel is available with data acquisition software which presents data for the SLAM algorithms in a structured manner. In Chapter 4 the details of the redesign of the Laser range finder is described, and in Chapter 5 the design of the actual SLAM algorithm is described.

4 Redesign of the Laser Range Finder

This chapter describes the redesign and implementation of the hardware and software of the laser range finder. First the upgrade of the hardware is described. After that an analysis is made of the optimal setup for the sensor. The sensing system also consists of a software processing part. These parts are described in Section 4.3 Furthermore the Calibration of the sensor is described in Section 4.4. Finally the correct working of the sensor is verified in Section 4.5.

4.1 Hardware Optimization of the Optical Sensor

The optical system to measure pipe profiles Drost (2009), is chosen as a basis for this project's laser range finder. This system however, has to be redesigned somewhat to become suitable for the new sensor platform. The existing system by Drost (2009) is shown in Figure fig:opticalsetup. The size of the sensor system needs to be made smaller, and it is preferred to eliminate the moving parts of the system. Also, the processing algorithms need to be less time consuming.

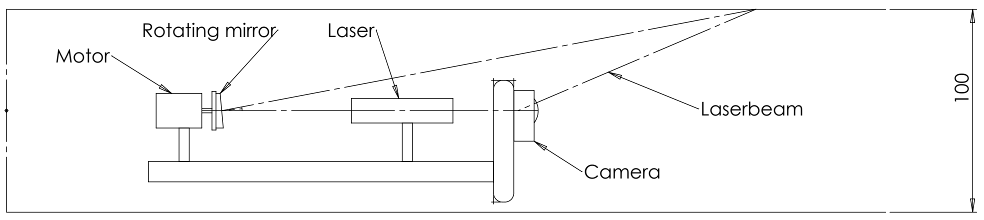


Figure 4.1: The optical setup as designed by Drost (2009)

4.1.1 Elimination of Sensor Moving Parts

The moving parts of the optical sensor system consist of the fast rotating, titled, mirror which forms the basis of the cone projection. Fortunately this part can be replaced by a passive element, called a Diffractive Optical Element (DOE). Such an element is based on diffraction of the laser beam to create a circle shaped diffraction pattern. The setup built in this project is using a standard Diffractive Optical Element manufactured by Holo-eye (http://www.holoeye.com). This element projects a circle on a flat surface, with a radius that linearly depends on the distance to the surface. Figure 4.2 shows what these Diffractive Optical Elements look like.

Figure 4.2: Sample photograph of Diffractive Optical Elements

4.1.2 Replacing Camera and Alterations to the Lens

For the Sensor Vessel, the old camera was not suitable because its size is not suitable for placement inside the vessel chassis. A camera with a different physical geometry is chosen to overcome these problems. In Figure 4.3, the new camera is shown which has a cylindrical shape, and the old camera has a more flat shape. The cylindrical shape is better suited for this application because now the laser beam which is coming from behind the camera is not blocked by the camera body. After the new camera was tested, it was concluded that the angle of view was not large enough. This was solved by adding a fish-eye lens to the camera.

(a) Old camera

(b) New cylindrical shaped camera

Figure 4.3: Picture of the old and the new camera

4.2 Optical Sensor Configuration Analysis

The optical pipe profiling system uses triangulation methods with structured light to obtains 3D information about the environment. Figure 4.4 shows a possible configuration of this system. In this section it is analyzed how different parameters of the setup influence overall performance of the system. This is done by taking a finite number of coordinate points which lie on the laser cone surface, and calculating what the camera pixel coordinates for these coordinate points will be. From this data, it can be assesed what the theoretical resolution and range of the system will be. A number of configurations will be simulated this way, to find the most feasible configuration.

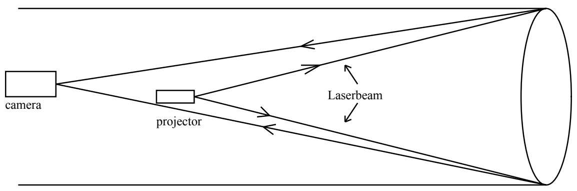


Figure 4.4: A simple representation of a possible configuration of the pipe profiling system

4.2.1 Mathematical framework

This subsection describes the mathematical framework used for the analysis. First the camera is described and after that, the Projected laser cone is described.

4.2.1.1 Camera Model

For this analysis, the camera is modelled as an ideal pinhole camera. Such a camera can be fully described by a calibration matrix K, a rotation matrix R and a translation vector $\vec{T}$. R and $\vec{T}$ describe the orientation and position of the camera with respect to the fixed world coordinate system. The calibration matrix K describes the conversion between 3D points in camera coordinates to 2D points on the camera projection plane. So if we now have a point $\vec{P}_w$ in world coordinates, and we want to know where the corresponding point in camera coordinates, $\vec{P}_c$ is, we get:

$$\vec{P}_c = R\left(\vec{P}_w - \vec{T}\right) \tag{4.1}$$

If it is assumed that the pixel size of the camera is exactly square everywhere, and that the image plane is exactly centered around the pinhole of the camera, then the calibration matrix K becomes:

$$K = \begin{bmatrix} f\cdot s & 0 & 0 \\ 0 & f\cdot s & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

In which f is the focal distance, or the perpendicular distance between the image plane and the pinhole, and s is a scaling factor which represents the number of pixels per metre on the image plane. From this it is easy to see that the horizontal (u) and vertical (v) pixel coordinates of the image will become:

$$u = \frac{x_c}{z_c}\cdot f\cdot s \tag{4.3}$$

$$v = \frac{y_c}{z_c}\cdot f\cdot s \tag{4.4}$$

This concludes the mathematical camera model.

4.2.1.2 Laser Cone model

A cone which is centered around a coordinate axis can easily be described by using cylindrical coordinates. If the cone for instance is aligned with the z axis, then using cylindrical coordinates (r,θ,z) this becomes:

$$r = k\cdot z \tag{4.5}$$

In order to calculate a finite number of points along the cone's surface, a parametric equation is made which describes a spiral along the cone's surface. α is defined as the opening angle of the cone.

$$r = \tan\frac{\alpha}{2}\cdot z \tag{4.6}$$

$$\theta = K\cdot z \tag{4.7}$$

This can be converted into carthesian coordinates to get:

$$x(z) = \tan\frac{\alpha}{2}\cdot z\cdot\cos(K\cdot z) \tag{4.8}$$

$$y(z) = \tan\frac{\alpha}{2}\cdot z\cdot\sin(K\cdot z) \tag{4.9}$$

The parametric cone equation can be translated and rotated to get any desired cone in 3D space. Figure 4.5 shows this representation, in which vector $\vec{V}$ describes the origin of the cone and vector $\vec{A}$ describes the direction of cone. The original cone equation points towards the z axis, so to transform this by means of a rotation matrix R_c to a cone pointing in the direction of A, at least the third column of this matrix should be the unit vector $\frac{A}{|A|}$. The cone is invariant to rotations around the axis A, so the only restriction on the first and second columns of R_c are

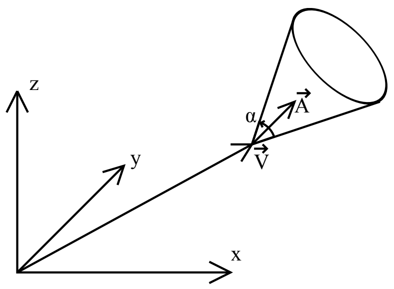


Figure 4.5: Cone description in 3D space.

that they are perpendicular to both themselves and A. If we take a random vector $\vec{B}$, this can be achieved as follows:

$$\vec{r}_3 = \hat{\vec{A}} \tag{4.10}$$

$$\vec{r}_2 = \frac{\vec{B} \times \vec{r}_3}{\left|\vec{B} \times \vec{r}_3\right|} \tag{4.11}$$

$$\vec{r}_1 = \vec{r}_2 \times \vec{r}_3 \tag{4.12}$$

$$R_c = \begin{bmatrix} \vec{r}_1 & \vec{r}_2 & \vec{r}_3 \end{bmatrix} \tag{4.13}$$

Using this matrix and the vector $\vec{V}$, the coordinates of a point $p_{\vec{cone}}$ on the cone surface can be transformed into world coordinates by:

$$\vec{p}_{world} = R_c \cdot \vec{p}_{cone} + \vec{v} \tag{4.14}$$

In Appendix A.1, a Matlab script is given which calculates points along a desired cone surface, at a certain interval (distance between points). The output result of this script is shown in Figure 4.6.

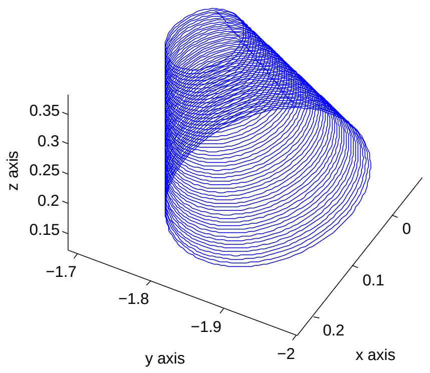


Figure 4.6: Cone output of matlab script

4.2.2 Numerical Simulation of Camera Images

In this subsection, a numerical simulation of the camera images produces by different configuration is described. This simulation is made by first calculating a large amount of points on the surface of the laser cone (Appendix A.1), and then the ideal pinhole camera image of these points is calculated. A total of six configurations will be simulated here. Figure 4.7 shows the different configurations which are simulated here.

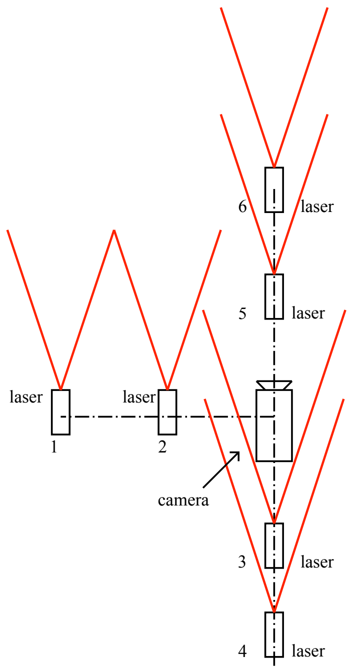


Figure 4.7: Different simulated configurations of the optical sensor

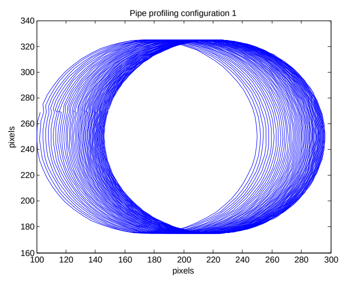
The opening angle of the cone is set such that a circle with diameter of 60mm will be projected at 100mm distance. (this specific opening angle is taken from the Diffractive optical element specifications). The positions of the projector with respect to the camera pinhole are as in Table 4.1.

Config nr:	x pos	z pos
1	-5 [cm]	0 [cm]
2	-2.5 [cm]	0 [cm]
3	0 [cm]	-2.5 [cm]
4	0 [cm]	-5 [cm]
5	0 [cm]	2.5 [cm]
6	0 [cm]	5 [cm]

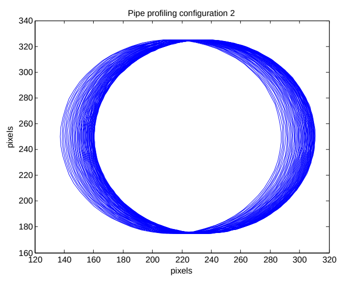
Table 4.1: Position of projector with respect to camera pinhole

4.2.2.1 Results of the Numerical Simulation

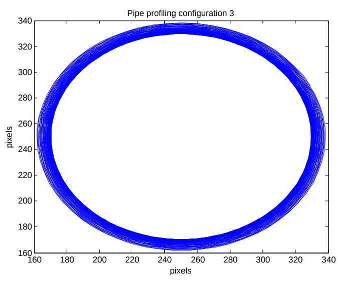
The results of the numerical simulations are shown in figure 4.8. As can be seen, in case the cone projector is not on the optical axis, as in configurations 1 and 2, the mapping from points on the cone to camera coordinates is not one-to-one. This means that for some camera coordinates, there are multiple world coordinates associated with that point. It can easily be shown that this only happens when the camera pinhole is located outside the cone. If the camera pinhole is behind the cone, then a one-to one mapping is also possible, but only if the camera is inside the virtual 'mirror' cone. (The invisible section of cone behind the origin of the projected cone). The simulations show that there is a bigger difference between largest and smallest diameter if the projector is at a distance away from the camera pinhole. Configurations 4 and 6 are better in this property than configurations 3 and 5. Of course if the projector would be located at the camera pinhole, then there would be no difference at all between small and large



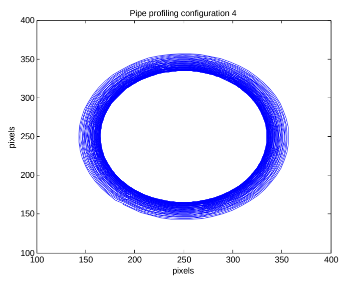
(a) Configuration 1



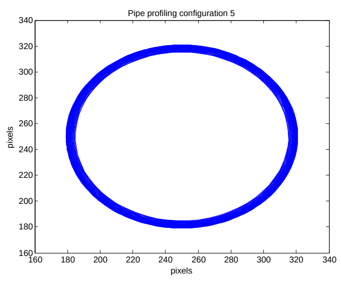
(b) Configuration 2



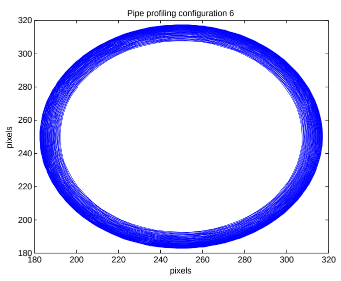
(c) Configuration 3



(d) Configuration 4



(e) Configuration 5



(f) Configuration 6

Figure 4.8: Numerical simulation output of configurations 1,2,3,4,5 and 6.

diameters. If the projector is placed too far in front of the camera, almost difference will be present. If the projector is placed too far back of the camera, then the desired diameters to measure will be out of the field of view of the camera. The influence of varying this parameter is shown in Figure 4.9.

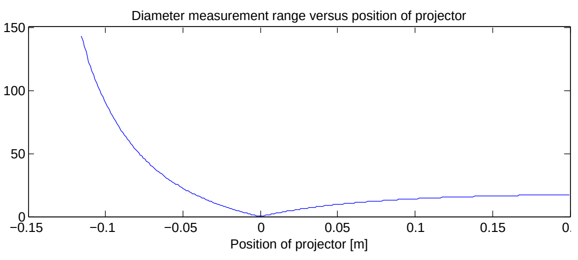


Figure 4.9: Influence of varying the projector position along the optical axis of the pinhole camera. If projector origin is at the pinhole, no triangulation is possible.

4.2.3 Conclusions of the Laser Range Finder Analysis

In this subsection a quantitative analysis has been done of the pipe profiling measurement system. It is argued that for a one-to-one mapping of world coordinates lying on the cone, to camera pixel coordinates, the camera pinhole should be inside the cone, or inside the virtual mirror opposite of the cone (negative z-coordinates in equation 4.9). For equal dynamic range in all directions, it is desirable to place the projector onto the optical axis of the camera, and also have the projector point into the same direction as the camera. The best position for optimal dynamic range would be to place the projector behind the camera. This concludes the analysis of the optical measurement system.

4.3 Processing of the Image Data

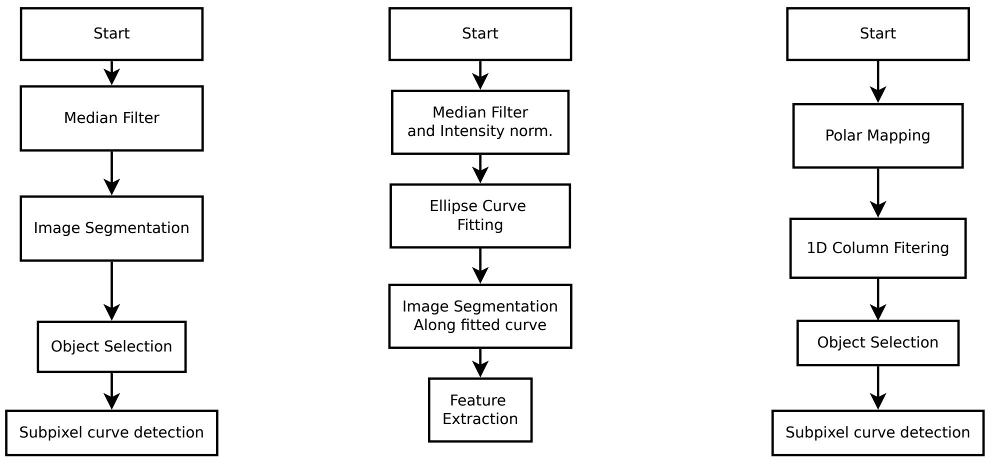
This section will discuss the procedure to obtain the desired 3D information from the structured light optical sensor system. The processing method used by the system built by Drost (2009) is looked at, and also alternative methods are compared.

4.3.1 Common Procedure for 3D Information Retrieval

In common, the retrieval of 3D information from structured light projection images taken by a camera, is involving some sort of laser curve extraction algorithm, a formula for calculation of 3D coordinates from image coordinates and some form of calibration data. These procedures will now be discussed.

4.3.1.1 Laser Curve Extraction Methods

The first step in measuring, is to extract the pixels from the camera image that belong to the laser curve. The method used by Drost (2009) consist of five steps, which are shown briefly in Figure 4.10(a). Furthermore, Duran et al. (2003) use a approach which is a bit different, by first fitting an ellipse to the data. Their method is shown in Figure 4.10(b). This is because Duran wants to find defects on the pipe wall, and Drost needs to map the entire pipe wall. Drost's method would be suitable for this project, only the speed of the algorithm is not fast enough. Therefore it was chosen to try a different approach. The next subsection will describe this new approach.



(a) Drost's method to extract the laser curve.

(b) Duran's method to extract the laser curve.

(c) New approach to extract the laser curve.

Figure 4.10: Different image processing approaches for extracting laser curves.

4.3.1.2 New Approach to Laser Curve Extraction

This subsection describes the new approach of laser curve extraction, which makes use of the fact that all image data is radially oriented around some center point which is the camera image of the center axis of the cone. The assumption is made here that the camera image of center of the cone does not move much as a function of distance. (This is true only if the projector is aligned exactly with the optical axis of the camera). The new approach will make a polar coordinate mapping of the image around the center point. Part of the calibration data will thus be obtaining this center point (x_0, y_0). This mapping is shown in equations 4.16.

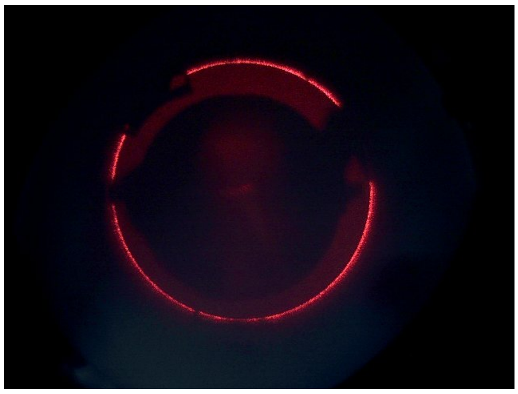
$$r = \sqrt{(x-x_0)^2 + (y-y_0)^2} \tag{4.15}$$

$$\phi = atan \frac{y-y_0}{x-x_0} \tag{4.16}$$

As an example, the resulting transform of the image in Figure 4.11(a) looks like the image in Figure 4.11(b). The advantage of having this representation, is that because of approximately radial orientation of the original image, the laser can be found in the vertical direction on every column. This makes finding the laser curve a one-dimensional search problem on every column. It is assumed that the laser curve will have a Gaussian radial intensity, so the search algorithm will need to find the peak of a Gaussian shape in each column. Finding this Gaussian peak will be done by first smoothing the data with a vertical blur on each column and then apply a 1D Laplacian of Gaussian function to the image. The blur and second-derivative of Gaussian convolution kernels can be combined into one single kernel. The Gaussian blur function is shown in Equation 4.17, the second derivative function is shown in Equation 4.18. The realization of these kernels can be shown in Figure 4.12. When these are applied to the image of Figure 4.11(b), the resulting image is shown in Figure 4.22. The minimum value of each column now represents the location of the laser curve. Using a weighted average around the pixel with minimum intensity will give sub-pixel accuracy to this measurement.

4.3.1.3 Comparison of new Laser Curve Extraction with Existing Algorithm

To compare the proposed laser curve extraction algorithm with the existing one by Drost (2009), both are tested on the same computer. Performance is measured by using the Matlab Profiler tool, the time duration of every step is recorded. The most significant steps are listed in Table 4.2. The new approach is faster by almost a factor of 10. After inspection of the segmentation algorithm, it is found that a lot of nested loops are present in this algorithm. These probably cause the long calculation time of this part. A remark must be made however, that the relative accuracy of the algorithms are not easy to obtain because the algorithms are both written to different hardware used with different calibration procedures.


(a) Example image to describe the laser curve extraction.

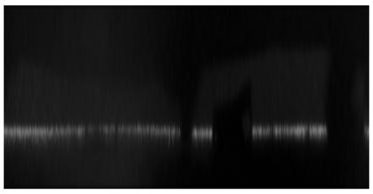

(b) Polar coordinate transform of the example image.

Figure 4.11: Example of transforming image with polar coordinate mapping.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\cdot\sigma^2}} \tag{4.17}$$

$$gx = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\cdot\sigma^2}} \cdot \left(\frac{(x-\mu)^2}{\sigma^4} + \frac{1}{\sigma^2}\right) \tag{4.18}$$

.

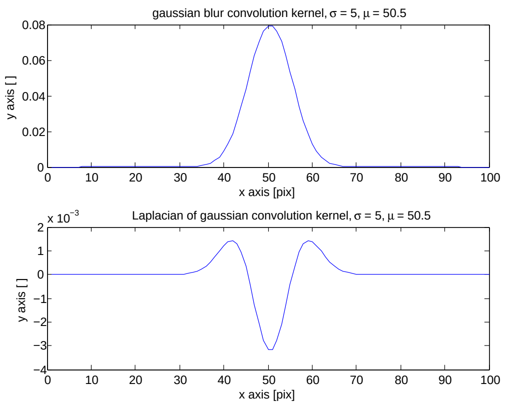
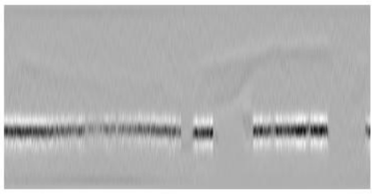
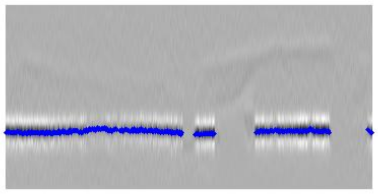


Figure 4.12: Convolution kernels for detection of laser Edge.



(a) Preprocessed and filtered image



(b) Filtered image, with extracted laser curve

Figure 4.13: This Figure shows the image from Figure 4.11(b), but now it is filtered with the kernel from equation (4.17) and then filtered with the kernel from equation (4.18). In (b), the detected laser curve is also added.

(a) Time duration Drost algorithm

Program step	Time used
Median filter	0.929[s]
Segment Image	6.149[s]
Select objects	1.16 [s]
total time	8.241 [s]

(b) Time duration new approach

Program step	Time used
Polar transform	0.460 [s]
Filter image	0.411 [s]
Select objects	0.03 [s]
total time	0.874 [s]

Table 4.2: (a):Time duration of the most significant steps in the laser curve extraction algorithm by Drost. (b)Time duration of the most significant steps in the approach used in this research. (Hardware, Pentium 4-2600 Mhz, 1.5 Gb RAM)

4.3.1.4 Retrieval of 3D Information from Extracted Laser Curve

This subsection describes the methods used and in particular the method used for this research to retrieve the 3D information from the extracted laser curve. Basically, there are two methods to retrieve 3D information. In literature, the 3D information is often retrieved by using a mathematical model of the camera and the projector, and a list of calibrated model parameters. Zhang (1999), Reilink (2008), Drost (2009) and Duran et al. (2003) all use this method. Perwass and Sommer (2006) use a bit different version of a camera model, but still rely on the use of a model to retrieve 3D information. Another way of retrieving 3D information from camera pixel locations is by making a lookup table between the pixels and the points in 3D space. Interpolation can then be used to obtain 3D information for the entire domain. Such an approach is used by for instance Wu1 and Tsai1 (2009). In their paper, they make a lookup table for world coordinates lying on a plane perpendicular to the camera optical axis at a certain distance. They are using analytical methods to adapt this table for a camera at different orientations and/or distances. This means that the camera orientation needs to be known.

4.3.1.5 Retrieval of 3D data using lookup Table

For this research, the lookup table method is chosen, because this method has low complexity, and requires little calculation. This is beneficial if implementation onto an embedded target system is required. For this method to be applied, a number of points on the cone surface are mapped to unique camera pixel coordinates. This mapping is recorded using the calibration procedure described in the next Section. The cone projected by the laser is actually a 2D surface. Therefore, the mapping needed is also from $\Re^2$ to $\Re^2$. How this mapping is made exactly is shown in Subsection 4.4.2.6.

4.4 Calibration of the Optical Setup

This section describes the calibration method of the optical sensor. Conventionally, a camera is calibrated by estimating the intrinsic and extrinsic camera parameters (for instance, Zhang (1999)). On top of this, the nonlinear lens distortions must also be estimated (See Reilink (2008) and also Zhang (1999)). The result of such a calibration will be that captured images captured can be transformed in such a way, that it seems as if the picture was taken by an ideal pinhole camera with known intrinsic and extrinsic parameters.

4.4.1 Disadvantages of Conventional Calibration

For the conventional camera calibration method, it is required to have a fairly good model of the camera and lens behaviour. This setup however, contains a large number of components with poor specifications available. It was observed by rotating the camera lens for instance that the fish-eye lens is not exactly located in the center of the camera. Also the alignment between the laser projector and camera are not easily measured. It is estimated that using a conventional calibration method will give poor estimations because a lot of unmodeled behavior.

4.4.2 New Calibration Method

For the 3D acquisition procedure used described in section 4.3.1.4, it is not necessary to have estimates of the camera parameters. The only data required is a mapping function which maps camera pixels to points on the surface of the projected cone. This is determined by using known points on the surface of the cone and finding the image location of these known points. Using interpolation, a complete map of the locations inbetween these points can be made.

4.4.2.1 Calibration Points on the Projected Cone

For the calibration, first a number of predefined points on the surface of the cone are defined. On the cone surface a number of conic sections are taken, perpendicular to the cone axis. These sections are circles, which have a diameter varying from 70mm until 135 mm with steps of 5mm. On each circle, 18 points are selected, equally spaced every 20 degrees. For example, on the first ring, the first point will be at coordinate x = 70, y=0 z = $\frac{70}{tan(\alpha)}$. (alpha is the opening angle of the cone, as in Equation 4.7). In Figure 4.14 the points used for Calibration are displayed in a 3D frame.

4.4.2.2 Referencing the Calibration Points

Now a method of recording these points needs to be found. This is done by placing a flat surface exactly perpendicular to the projector axis, at the predefined distances from the cone origin. This will result in a circle of required diameter to be projected onto the flat surface. Now a

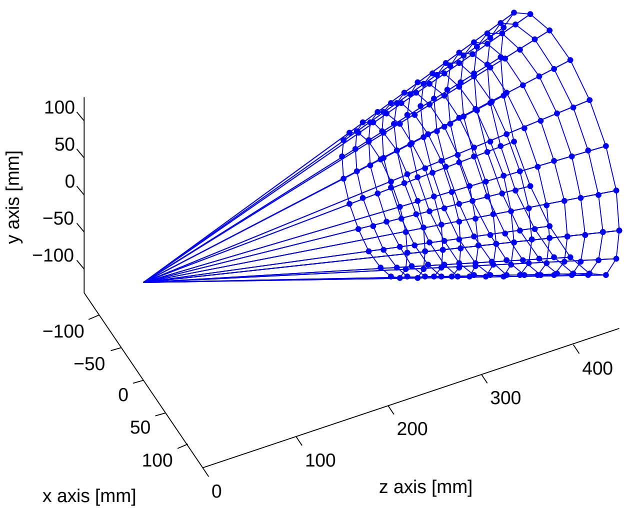


Figure 4.14: The points on the cone used for calibration

reference point on the circle needs to be defined, for the 18 points onto the circle to have a common starting point. Figure 4.15 shows what the projected circle looks like. Three small sections of the circle are not lit, this is because the construction of the sensor vessel still needs an optimization. These sections can now be exploited, to have a common starting point for the 18 points on each circle. The location of these 18 points are drawn as blue circles in Figure 4.15. The arrow with the word “reference” points towards the end of the laser gap, which is the location of the first point.

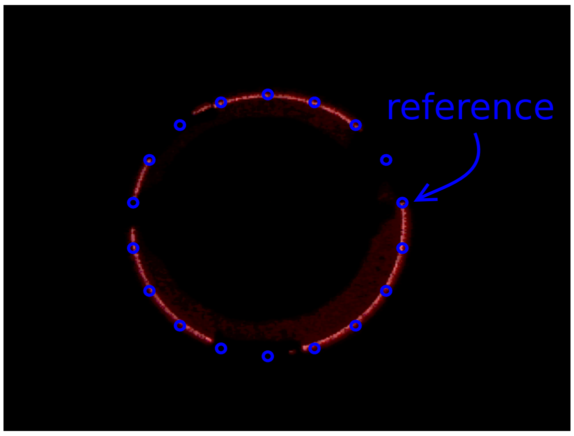
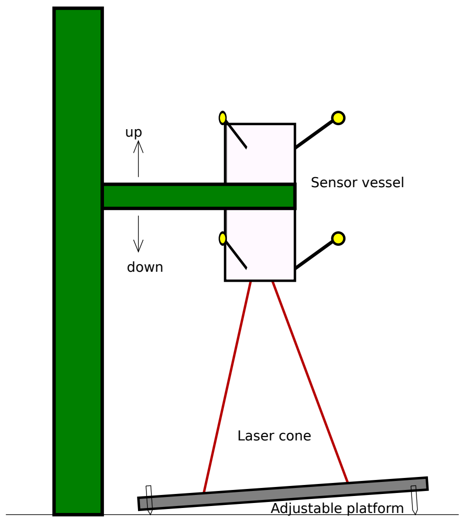


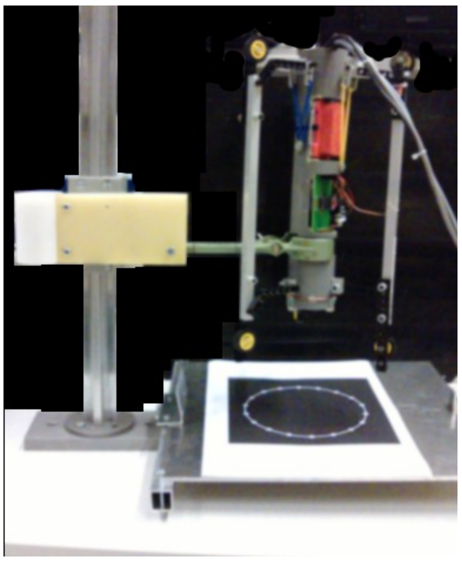
Figure 4.15: Laser cone projected on a flat surface perpendicular to cone, with reference marked, and locations of the 18 points along the circle also marked.

4.4.2.3 Projector Calibration Hardware

To record a set of points on the perpendicular plane, first the laser projector axis needs to be aligned exactly perpendicular to the projection plane. This is done by building a piece of calibration hardware which holds the sensor vessel, contains an adjustable projection plane, and the distance between the projector and this plane is variable. Figure 4.16 shows this setup.



(a) Schematic display of the calibration hardware.



(b) Photo of the calibration hardware, the sensor vessel is held by a clamp.

Figure 4.16: The calibration hardware, the height of the projector is adjustable, the ground plane's orientation is also adjustable

4.4.2.4 Aligning the Projector with Projection Plane

As told, the projector axis needs to be perpendicular to the projection plane. The hardware is built such, that the plane can be adjusted by adjusting the length of the three bolts supporting the plane. The procedure to make the plane perpendicular to the projector axis is to use a cylindrical piece of alignment tube which has a line printed on the inside. When the laser beam is exactly aligned with the line on the inside wall of the tube, then the plane is perpendicular to the projector axis. The alignment tube is shown in Figure 4.17. The alignment could also be done by printing a circle on a piece of paper, but using this vertical tool makes the sensitivity for misalignment much larger. After the projection plane is properly aligned, the height adjustment knob of the setup can be used to adjust the radius of the projected circle on the plane.

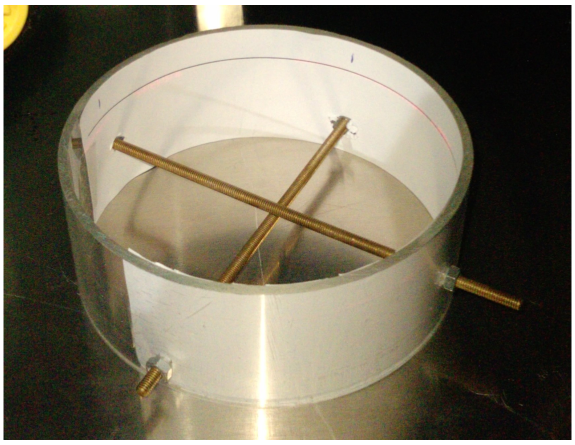


Figure 4.17: Cylinder shape used to assist in aligning the Projection plane with the projector.

4.4.2.5 Calibrating Patterns

Now that the projector is properly aligned with the projection plane, a set of calibration patterns is used to construct a mapping between points on the cone surface and camera image pixels. For every distance to be calibrated, a circle with 18 points on its perimeter is printed. This image is laid onto the projection plane, and the height of the projector is adjusted until the laser exactly lays on the circle printed onto the paper. Now the exact location of the projector with respect to the plane is known. This image can now be captured by the camera, and the pixel location of the 18 points along the circle can be measured. Figure 4.18 shows the pattern and captured image of the circle with 70mm radius. The captured image is saved to hard disk and a Matlab script is used to accurately select the 18 points and calculate the location.

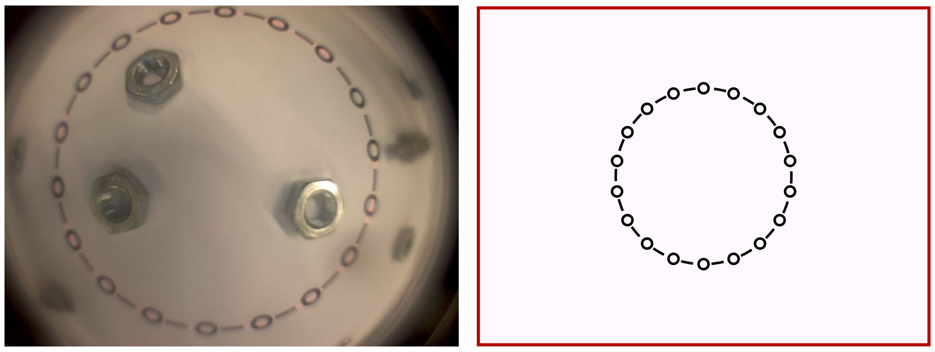

(a) Calibration image captured by camera (b) Image used for calibration

Figure 4.18: The calibration image, for the 70 mm radius circle. (a) Gives the photo of the calibration pattern shown in (b)

.

4.4.2.6 a Mapping of the Cone Surface

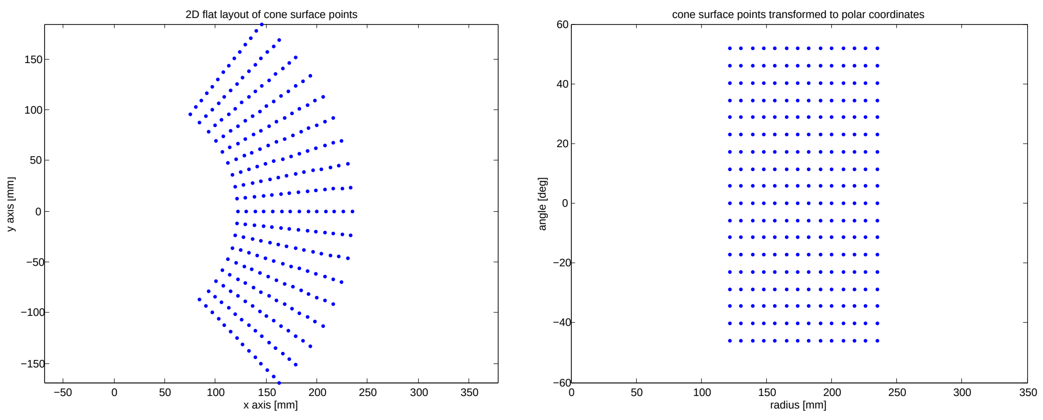
After making all measurements, a set of pixel coordinates becomes available, with corresponding location on the cone's surface. This is used to create a mapping function which maps the 2D locations in the image to 3D locations on the cone. When looking to the points on the cone, it becomes clear that only two degrees of freedom are present there. The cone surface can be cut open and laid out onto a flat plane. This will then look like Figure 4.19 (a). After transformation into polar coordinates, this looks like Figure 4.19 (b).

4.4.2.7 a Mapping of the Detected Image Coordinates

By using the calibration patterns (Figure 4.18), a set of point coordinates are collected. From acquisition chapter it follows that the result of the calibration procedure should be a mapping function from 2D polar coordinates to 3D information. Therefore the captured coordinates will first be translated so that they are centered, and then transformed into polar coordinates. The centering will be done by taking the mean of all captured coordinates. in Figure 4.20(a), these captured points are shown. Figure 4.20(b) shows the polar coordinates of the captured points, after first translating the points to make them centered around origin. The sinusoidal distortions along the vertical axis in the polar coordinates mapping is mainly due to the fact that the center of all captured points is not the center of each individual circle, due to misalignment between the camera and the projector.

4.4.2.8 Interpolation of a Mapping between camera pixels and World Coordinates

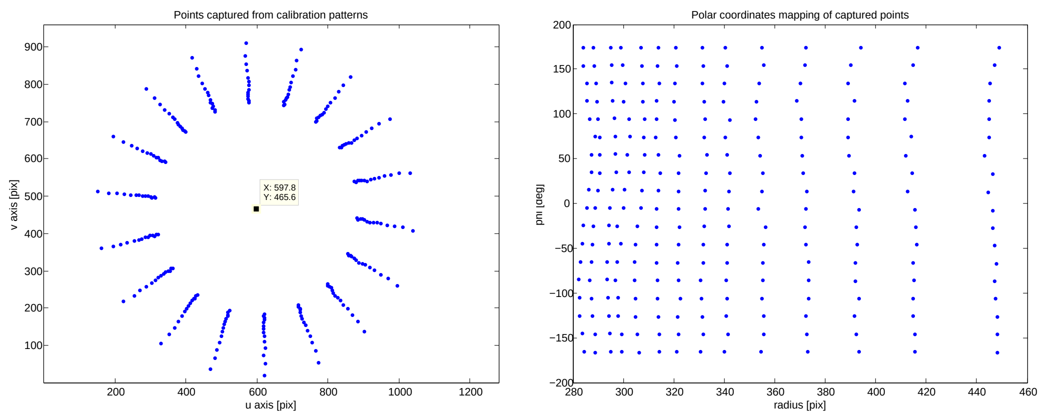
From the previous subsections 4.4.2.6 and 4.4.2.7, two approximately rectangular shaped domains are available in which each point in one domain has a corresponding point in the other domain. These domains are shown in Figure 4.19(b) and in Figure 4.20(b). It is now easy to create a mapping function using interpolation to find the transform of points in between the



(a) Layout of cone points on a flat surface

(b) flat surface cone points transformed into polar coordinates

Figure 4.19: Layout of the points on the cone surface (see Figure 4.14 on a 2D plane (a)), and then transformed to polar coordinates (b)



(a) Calibration points captured with camera (with centre coordinates)

(b) Calibration points polar transformed

Figure 4.20: Pixel coordinates of the captured points on the cone surface (see Figure 4.14) (a), and then transformed to polar coordinates (b)

calibrated points. Matlab has a built in function called "TriScatteredInterp", which can be used to create this interpolated mapping. A function called "makeMapping()" is made, and is used to create the mapping from a set of measured calibration points. Appendix A.4 shows the code for this function. The makes use of the script Create_coords.m, found in Appendix A.3, to read in the measured calibrated coordinates.

4.5 Verification of the Calibration Procedure

The new calibration procedure needs to be verified. The way to verify the calibration is to take a set of known world coordinates and apply the measurement procedure with the calibration data. The first test is a flat plane. The plane is put perpendicular to the projector, using the calibration hardware (Figure 4.16). The projected circle is measured using a sliding gauge, and was found to have a diameter of 104 mm. Figure 4.21 shows the image captured of this situation.

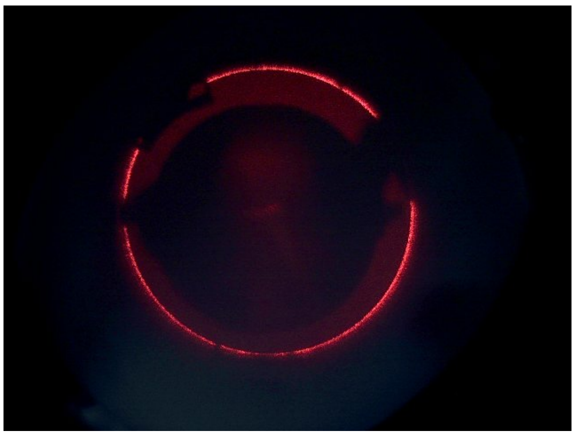


Figure 4.21: Image used to verify the calibration procedure.

4.5.1 Processing the image

The image must be processed. First it is converted to grayscale, then a polar coordinate transform is made, a gaussian blur in the vertical direction is then performed, and then a 1D vertical blob-detection convolution filter is used. The resulting image is shown in Figure 4.22. As can be seen, the line is detected very clear. Now the peak of the laser curve is found by taking the pixel with the minimum value in each (valid) row, and then taking the weighted average of that pixel index and the 10 surrounding pixels.

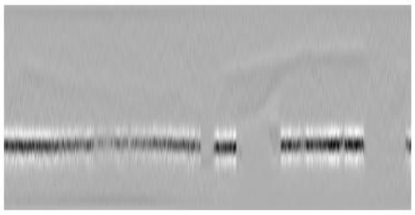


Figure 4.22: Processed image of the laser curve from Figure 4.21

4.5.2 Retrieving 3D information

After this, the 3D information can be found by using the interpolation functions made with the function of Appendix A.4, and then using some simple geometry. The end result is shown in Figure 4.23. The script used for the derivation of this data is found in Appendix A.5. When the data is analyzed, the mean and standard deviation of the circle radius are written in Table 4.3. This is considered accurate enough, so hereby the calibration procedure is verified.

Name:	Value:
mean radius	52.0248 [mm]
standard deviation radius	0.3355 [mm]
number of samples	246

Table 4.3: Measured data about the radius of the circle

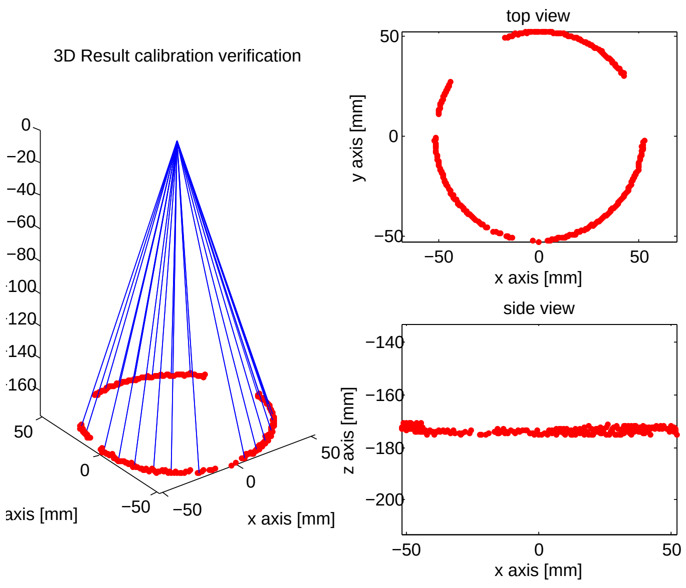


Figure 4.23: The detected 3D points, using a plane perpendicular to the projector, circle diameter is 104 mm, projector distance is 173.33 mm

4.6 Conclusions on the Redesign of the Laser Range Finder

A laser range finder system has now been developed which, for calibration requires no knowledge about the physical parameters of the camera or lenses used. The method of calibration constructs a large table of coordinate pairs, matched between world and camera coordinates. The table can be interpolated for increased resolution, which is also done. A elaborate manual is made for calibration of the sensor, which can be found in Appendix E.

5 Design and Implementation of SLAM the Algorithm

This chapter describes the filtering techniques used for the estimation of the state of the sensor vessel and mapping of the local environment. In Chapter 2, it is argued that the problem of SLAM is a highly coupled problem between localization and mapping. This is especially true in SLAM problems where a robot moves on a 2D Plane, where large drifts will occur in odometry data, which will result in curvatures in the map of nothing is done to compensate for this. In this research however, the freedom of movement of the robot is restricted by the pipe walls, therefore less drift from odometry is to occur. For this reason, the SLAM problem is decoupled into the localization problem and the mapping with a known pose problem. First the localization problem is discussed.

5.1 Sensor Vessel State Estimation

To effectively estimate the state of the sensor vessel, a Sequential Importance Resampling filter, also called Particle Filter is used. This section only discusses estimation of the vessel state in a straight piece of pipe. Later this will be expanded to detecting features like bends and T-junctions, by making use of an occupancy grid map. First, the Algorithm used is explained briefly.

```
1 Draw samples $x_k^i$ from importance density $q(x_k|x_{k-1}^i, z_k)$
2 Update weights using $\frac{p(z_k|x_k^i)p(x_k^i|x_{k-1}^i)}{q(x_k|x_{k-1}^i, z_k)}$
3 Resample if $N_{eff} < \frac{N}{2}$
```

To make a suitable importance density function, first a reasonable good dynamic model for the state and measurements is needed. They will be described in Subsections 5.1.1,5.1.2 and 5.1.3. These models must be chosen such, that it becomes feasible to create an importance density. If they differ too much from the real state equations however, then the performance of the particle filter may become degraded.

5.1.1 Definition of the Vessel State Vector

First, the vessel state vector is defined. The vessel is modeled as a single rigid body with six degrees of freedom. The degrees of freedom of the vessel are composed of a 3 dimensional position $\vec{P}$ and a pitch roll and yaw orientation $\vec{\Theta}$. This gives a state vector:

$$\vec{x}_k = \begin{bmatrix} \vec{P}_k \\ \vec{\Theta}_k \end{bmatrix} \tag{5.1}$$

The laser range scanner axis is aligned with the x-axis of the vessel coordinate system, and the origin of the laser range finder is at the origin of the vessel coordinate system. Figure 5.1 shows this configuration. The forward movement of the sensor vessel is towards the positive x-axis.

5.1.2 Sensor Vessel Dynamic State Equation

The sensor vessel is made to fit in pipes with different radii. A schematic of the vessel mechanical layout can be seen in Figure 5.2. The wheels are spring suspended so that the vessel approximately centers itself inside the pipe. vessel can be moved forwards and backwards through the pipe by applying external force. Therefore, the direction of travel will be approximately along the center axis of the pipe. The vessel has no motors, an not much is known about the vessel dynamic behavior. Furthermore, the sample frequency of the measurement system is in the order of $1[Hz]$, which is much lower than the dynamics of the vessel. The vessel dynamics are therefore modeled as a noisy process, with two additive noise sources. The first noise source is a small source of random movements in all directions and orientation, because of unmodeled

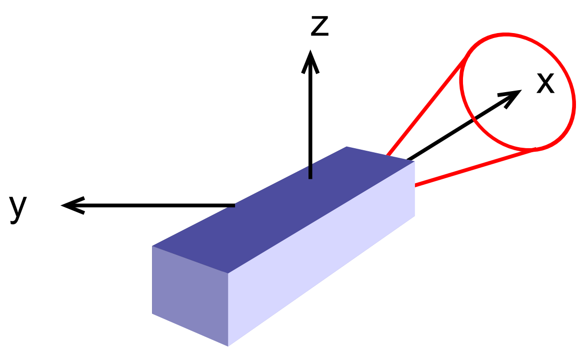


Figure 5.1: Default orientation of the vessel with zero Rotation

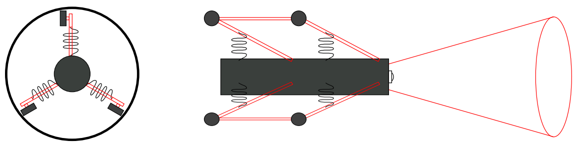


Figure 5.2: Schematic front and side view of the sensor vessel. The wheels are spring suspended and one of the wheels contains an optical rotation encoder

dynamics called $\vec{v}^n_{k-1}$. The second noise source is the movement due to external pushing or pulling the vehicle, which is a large noise source called $\vec{v}^o_{k-1}$. The second noise source has only a component tangential to the pipe center axis, and is zero in any other direction. Equation 5.2 shows the dynamic model of the state.

$$\begin{aligned}\vec{x}_k &= \vec{x}_{k-1} + \vec{v}^n_{k-1} + \vec{v}^o_{k-1} \\ \vec{v}^n_{k-1} &\sim N(0,Q) \\ \vec{v}^o_{k-1} &= \begin{bmatrix} R \\ 0^3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} v^e_{k-1} \\ v^e_{k-1} &\sim N(0,m^2)\end{aligned} \tag{5.2}$$

in which:

- $\vec{v}^n_{k-1}$: normal distributed random variable with zero mean and covariance matrix accounting for small random movements and rotations between time steps.
- Q: 6x6 matrix with covariances of $\vec{v}^n_{k-1}$.
- $\vec{v}^o_{k-1}$ Vector directing towards pipe axis with magnitude v^e_{k-1}
- 0^3: 3x3 matrix with all elements zero.
- R: Rotation matrix representing the orientation of the pipe.
- v^e_{k-1}: Normal distributed random variable to account for movements along pipe axis. -m..m represent maximum vessel distance traveled through the pipe in between time steps.

The variance of the position random variables in v^n_{k-1} are assumed the same, and are captured in parameter σ^2_p. The variance of the orientation random variables in v^n_{k-1} all have variance σ^2_o

The sum of $\vec{v}^n_{k-1}$ and $\vec{v}^o_{k-1}$ can be combined in a single vector with covariance matrix Q_{k-1}:

$$Q_{k-1} = \begin{bmatrix} \sigma_p^2 + r_{11}^2 m^2 & r_{11}r_{22}m^2 & r_{11}r_{33}m^2 & 0 & 0 & 0 \\ r_{11}r_{22}m^2 & \sigma_p^2 + r_{11}^2 m^2 & r_{22}r_{33}m^2 & 0 & 0 & 0 \\ r_{11}r_{33}m^2 & r_{22}r_{33}m^2 & \sigma_p^2 + r_{11}^2 m^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_o^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_o^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_o^2 \end{bmatrix} \tag{5.3}$$

5.1.3 Measurement Equation

The measurement equation that is needed to calculate a importance density, will now be made. It is chosen to omit the laser range finder data and to only use roll-angle from acceleration measurement and encoder data. The measurement vector z_k is split into a laser range finder part r_k, and a encoder + roll angle part called u_k. The roll angle from the acceleration sensor is calculated by:

$$\theta_r = atan(\frac{acc_y}{acc_z}) \tag{5.4}$$

Which is shown schematically in Figure 5.3.

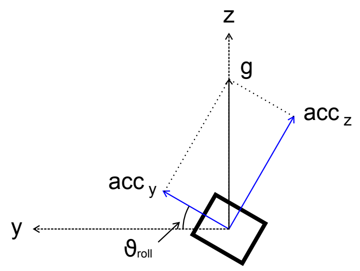


Figure 5.3: Roll angle, from acceleration data.

The measurement model for the roll angle with noise w_k^{roll} becomes:

$$u_k^{roll} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \vec{x}_k + w_k^{roll} \tag{5.5}$$

It is assumed that the measurement from the optical encoder will be approximately equal to the distance traveled by the vessel in the direction tangential to the direction of the pipe center axis. The encoder measurement model with noise w_k^{enc} is:

$$u_k^{enc} = \begin{bmatrix} r_{11} & r_{21} & r_{31} & 0 & 0 & 0 \end{bmatrix} \cdot \vec{x}_k + w_k^e nc \tag{5.6}$$

where $< r_{11}, r_{21}, r_{31} >$ is a unit vector in the direction of the pipe centre axis. Now the entire measurement model becomes:

$$u_k = H_k x_k + w_k \tag{5.7}$$

With w_k normal distributed with covariance matrix R_k according to:

$$w_k \sim N(0, R_k) \tag{5.8}$$

and:

$$H_k = \begin{bmatrix} r_{11} & r_{21} & r_{31} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.9}$$

5.1.4 Choice of the Importance Density

In the first step of the SIR algorithm, a new set of samples must be drawn from the importance density $q(x_k^i|x_{k-1}^i, z_k)$, which was introduced in Subsection 2.1.2.1. The optimal choice for q, is shown by Doucet et al. (2000) to be:

$$q(x_k|x_{k-1}^i, z_k)_{opt} = p(x_k|x_{k-1}^i, z_k) \tag{5.10}$$

Unfortunately it is not possible to evaluate this pdf analytical in this case, due to the complex nature of the laser range finder. It is also either not possible, or very hard, to sample from this density function. Therefore a sub-optimal alternative has to be found. The most common (Ristic et al. (2004)) sub-optimal importance density is:

$$q(x_k|x_{k-1}^i, z_k)_{subopt} = p(x_k|x_{k-1}^i) \tag{5.11}$$

From the state dynamics, it follows that using the sub-optimal choice of Equation 5.11 is not suitable, because encoder measurements are not taken into account. This would lead to a lot of uncertainty about the position of the vessel along the pipe axis, thus will also lead to a large number of particles with low weights. Ristic et al. (2004) gives a solution for Equation 5.10 in case the state dynamics are nonlinear but the measurement equation is linear, and the process and measurement noises are normal distributed. The system equations without the laser range finder data fits to this requirement, Hence, the importance density is chosen to be:

$$q(x_k|x_{k-1}^i, u_k) = p(x_k|x_{k-1}^i, u_k) \tag{5.12}$$

In Ristic et al. (2004), page 46, calculation of the density function of Equation 5.12 has been solved and the pdf becomes:

$$p(x_k|x_{k-1}^i, u_k) = N(x_k; a_k, \Sigma_k) \tag{5.13}$$

where:

$$a_k = x_{k-1}^i + \Sigma_k H_k^T R_k^{-1}(z_k - b_k) \tag{5.14}$$

$$\Sigma_k = Q_{k-1} - Q_{k-1} H_k^T S_k^- 1 H_k Q_{k-1} \tag{5.15}$$

$$S_k = H_k Q_{k-1} H_k^T + R_k \tag{5.16}$$

$$b_k = H_k x_{k-1}^i \tag{5.17}$$

in which R_k is the measurement noise covariance matrix and Q_{k-1}, the process noise covariance matrix. Now the weight update equation is needed. The weight update equation is:

$$w_k^i \propto w_{k-1}^i \frac{p(z_k|x_k^i)p(x_k^i|x_{k-1}^i)}{q(x_k^i|x_{k-1}^i, u_k)} \tag{5.18}$$

The measurement vector z_k can be split up into a laser range finder part r_k and a encoder measurement part u_k, of which it can be argued that they are conditionally independent random variables. Using that fact, the weight update equation with the chosen importance density can be expanded:

$$w_k^i \propto w_{k-1}^i \frac{p(z_k|x_k^i)p(x_k^i|x_{k-1}^i)}{q(x_k^i|x_{k-1}^i, u_k)} \tag{5.19}$$

$$\ldots = w_{k-1}^i \frac{p(u_k, r_k|x_k^i)p(x_k^i|x_{k-1}^i)}{q(x_k^i|x_{k-1}^i, u_k)} \tag{5.20}$$

$$\ldots = w_{k-1}^i \frac{p(u_k|x_k^i)p(r_k|x_k^i)p(x_k^i|x_{k-1}^i)}{q(x_k^i|x_{k-1}^i, u_k)} \tag{5.21}$$

Combining this with the fact that:

$$q(x_k^i|x_{k-1}^i, u_k) = \frac{p(u_k|x_k^i)p(x_k^i|x_{k-1}^i)}{p(u_k|x_{k-1}^i)} \tag{5.22}$$

It follows that:

$$w_k^i \propto w_{k-1}^i p(r_k|x_k^i)p(u_k|x_{k-1}^i) \tag{5.23}$$

The solution for $p(u_k|x_{k-1})$ can be found in From Ristic et al. (2004) (page 46) , it is normal distributed with pdf:

$$p(u_k|x_{k-1}) = N(u_k, b_k, S_k) \tag{5.24}$$

with S_k and b_k already listed in Equations 5.16 and 5.17. Now a suitable importance density is found, which can be evaluated and also samples can be drawn from it, both are necessary to implement the SIR algorithm. This leaves only the calculation of $p(r_k|x_k)$ for which a fairly elaborate observation model for the laser range finder is needed. The implementation of this observation model, based on “ray casting”, is described in the next section. First the resampling algorithm is described shortly.

5.1.5 Resampling Algorithm

The resampling algorithm used, is a low computational load algoritm called Residual Systematic Resampling, which was developed by by Bolić et al. (2004). No regularization of particles is performed. According to the new weights, it is calculated how often a particle should be copied after resampling. So particles with large weights will be copied often and particles with small weight will be omitted.

5.2 Observation Model for the Laser Range finder

To calculate the posterior $p(r_k|x_k)$, an observation model is needed. This model will be used to calculate the probability of a measurement given the state and map. The observation model must somehow simulate sensor output given a certain state and map. In Literature many different approaches exist for such a a "beam model". Laet et al. (2008) presents a rigorous analysis on probabilities for different beams and accounts for unmodeled objects in the beam. Their beam model is called RBBM, short for Rigorously Bayesian Beam Model. Another approach is done by Plagemann et al. (2007), where they present a Gaussian Beam Process.

5.2.1 Different Approaches to a Observation model

A measurement from a laser range finder will consist of many beams r, with a certain length, and a certain direction with respect to the sensor. So this measurement can be written down as in Equation 5.26.

$$r_k = r_k^1, r_k^2, .., r_k^n \tag{5.25}$$

$$\tag{5.26}$$

One approach is to treat each ray as an indepentend process. A few families of models consider each ray as an independent measurement, with its own pdf, they are:

- Ray cast or Beam based models: For every measurement a one-dimensional distribution is calculated.
- End point models: The probability of a measurement is a function of the shortest distance between the end-point of the ray and an object. (e.g. pipe wall)

The probability density function $p(r_k|x_k)$ if each ray is independent becomes:

$$p(r_k|x_k) = \prod_{i=1}^{N} p(r_k^i|x_k) \tag{5.27}$$

This will lead to extremely peaked distributions, as a laser range scanner typically produces a large amount of rays per time-step. This very small peak will in turn cause many weights of the particles in the particle filter to become very low.

Other algorithms use more elaborate ways to overcome this extreme peakedness. One way is to take into account the fact that often the estimate of the position of the sensor is worse than the laser range finder precision. If the position uncertainty is somehow taken into account, the probability distribution will be less peaked. Pfaff et al. (2007) makes use of this fact by estimating $p(z|x)$ based on the local environment $U(x)$ around x. Equation 5.28 shows this.

$$p(z|x) = \int p(z|\tilde{x})p(\tilde{x})_{\tilde{x}\in U(x)} \tag{5.28}$$

Other ways is to make to reduce the extreme peakedness is to subsample the rays of the laser range scanner, or to introduce a lower bound to $p(r_k^i|x_k)$.

5.2.2 Design of the Observation Model

The quality of the custom built laser range finder, which is used in this project is not comparable to commercial grade laser range finders. All of its properties or not very well known yet. Because of this, it is not considered useful to put a lot of effort in finding the mathematically most correct model of the range finder. A simple model is designed to take into account basic laser range finder properties. The model is based on a technique called “ray casting”. It is almost the same as the ray-tracing technique used in rendering computer generated graphics.

5.2.2.1 Ray Casting

Ray casting is a method which looks along each laser range finder ray to see at what position along this ray an object is found (or no object is found). The laser range finder used in this project outputs an array of <x,y,z> coordinates which give the location of a detected laser point, with respect to the projector. This gives us for each measured point a ray running from the projector centre to the measured point. Figure 5.4 shows one such ray.

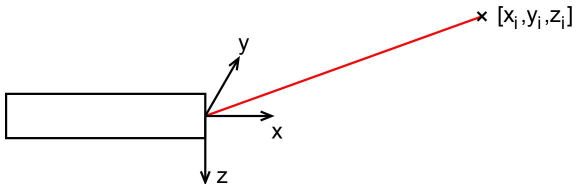


Figure 5.4: One single ray from the laser range finder

The ray can be parameterized as:

$$\vec{r} = \vec{T} + t\vec{D} \tag{5.29}$$

in which $\vec{T}$ is the begin point of the ray, and $\vec{D}$ is the unit direction vector of the ray, in sensor vessel based coordinates. If now also the pipe orientation and radius is taken into account, then this measured ray can be placed inside the pipe according the the orientation and position of the vessel. The pipe orientation and vessel orientation values can be transformed into a rotation matrix. These are called R_w for the pipe orientation matrix, and R_v for the vessel orientation. matrix. in Section 5.1.1 the vector $\vec{P}_k$ defines the position of the vessel particle. Let the vector $\vec{S}$ define the starting point of the pipe. With this information, the ray can be transformed to pipe-based coordinates by doing:

$$\vec{T}_{pipe} = R_w^{-1}((R_v\vec{T} + \vec{P}_k) - \vec{S}) \tag{5.30}$$

$$\vec{D}_{pipe} = R_w^{-1}((R_v\vec{D} + \vec{P}_k) - \vec{S}) \tag{5.31}$$

So now the parameterization for the ray, in pipe coordinate system is:

$$\vec{r}_{pipe} = \vec{T}_{pipe} + t\vec{D}_{pipe} \tag{5.32}$$

Now an equation is needed which calculates the value of t, to have the ray touch the wall of the pipe. For a pipe with infinite length, the following must be true if the ray hits the pipe wall:

$$\vec{r}^{T}_{pipe} M \vec{r}_{pipe} = rad^2 \tag{5.33}$$

where rad is the pipe radius, and M is the matrix:

$$M = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.34}$$

If the elements of $\vec{T}_{pipe}$ are called T_x, T_y, T_z and the elements in $\vec{D}_{pipe}$ are called D_x, D_y, D_z, This leads to a second order polynomial in t:

$$(T_y + tD_y)^2 + (T_z + tD_z)^2 = rad^2 \tag{5.35}$$

This can easily be solved and will lead to two solutions. If the vessel is located inside the pipe, then one solution of t will be $t > 0$ and the other will be $t < 0$. The positive solution is then the solution we need. Furthermore, for the ray to be valid, it must be in the pipe segment so:

$$0 < T_x + tD_x < l \tag{5.36}$$

in which l is the length of the pipe segment. t is the calculated length of the ray. This can be easily compared to the actual measured length of the ray.

5.2.2.2 Stochastic model

For every measured ray, a length is calculated given the vessel state particle x_k. So now a stochastic model is needed to calculate the pdf of $p(r_k|x_k)$. if there are N laser point measurements:

$$p(r_k|x_k) = \Pi_{i=1}^{N} p(r_k^i|x_k) \tag{5.37}$$

There are three types of events that can cause a certain laser ray length. First there is the case that some measurements from the laser range finder are completely random across the range of the range finder. also, sometimes measurements clip to the maximum or minimum range of sensor. Furthermore, the probability that an object is actually measured at the correct distance is assumed to be normal distributed around that object distance. So this gives three eventualities A, B, and C which lead to a certain measurement. It is chosen to divide the probabilities as:

- P(A): 0.8: Normal object measurement, ray length expected r_{obj}.
- P(B): 0.1: Purely random measurement, ray length uniform between r_{min}, r_{max}.
- P(C): 0.1: min or max clipped range data r_{min}, r_{max}

The probability density functions of measurement r_k^i given an event are modeled as:

$$p(r_k^i|x_k, A) = N(r_k^i; r_{obj}, \sigma_A) \tag{5.38}$$

$$p(r_k^i|x_k, B) = U(r_k^i; r_{min}, r_{max}) \tag{5.39}$$

$$p(r_k^i|x_k, C) = 0.5 \cdot (\delta(r_k^i - r_{min}) + \delta(r_k^i - r_{max})) \tag{5.40}$$

So the probability density function of a certain measurement given the state, is

$$p(r_k^i|x_k) = p(r_k^i, A|x_k) + p(r_k^i, B|x_k) + p(r_k^i, C|x_k) \tag{5.41}$$

$$\Rightarrow p(r_k^i|x_k) = P(A) \cdot p(r_k^i|x_k, A) + P(B) \cdot p(r_k^i|x_k, B) + P(C) \cdot p(r_k^i|x_k, C) \tag{5.42}$$

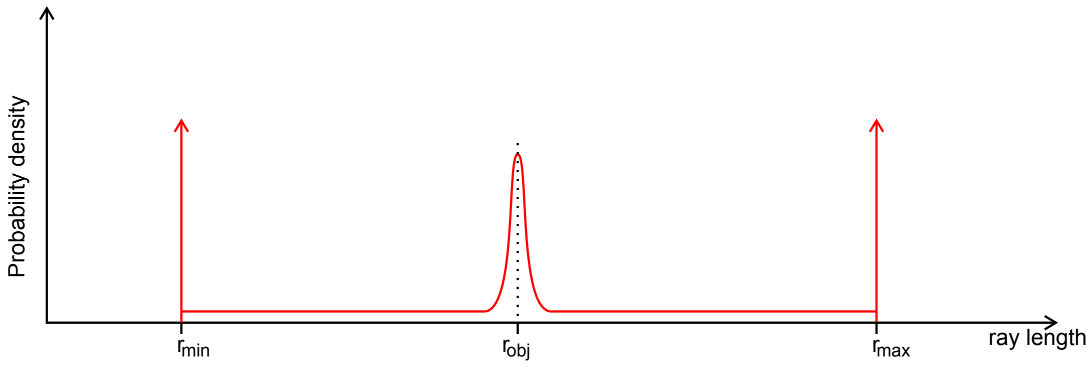


Figure 5.5: pdf of $p(r_k^i|x_k)$

This will still produce very peaked distributions for $p(r_k|x_k)$, but at least the outliers do not make the probability approach zero very fast. Figure 5.5 shows the pdf for a single ray.

Now an observation model is available which can calculate $p(r_k|x_k)$, if the vessel is inside a straight piece of pipe.

5.3 World Mapping, Estimation of Features

The vessel state estimation is capable of detecting the vessel pose and position inside a straight piece of pipe. The environment of the vessel also contains features like bends and T-junctions. in the next subsections, two approaches to handle these features will be discussed. Subsection 5.3.1 discusses mapping using a parameterized environment, and Subsection 5.3.2 discusses a method using a grid.

5.3.1 Mapping with Parameterized Environment

Many approaches in literature add a world model to the particles of the particle filter. This means that the dimensionality of state vector is increased, This results in an exponential increase in particles needed for the SIR algorithm. A way overcome this high dimensionality is given with the Rao Blackwellized Particle Filter SLAM algorithms such as used by Schroeter and Gross (2008), where first the robot trajectory is estimated using a particle filter to estimate $p(x_{1:k}|r_{1:k},u_{0:k})$, and then this is used to estimate $p(x_{1:k},m|r_{1:k},u_{1:k})$ which is factorized into:

$$p(x_{1:k},m|r_{1:k},u_{1:k}) = p(m|x_{1:k},r_{1:k})p(x_{1:k}|r_{1:k},u_{0:k}) \tag{5.43}$$

One approach to represent the vessel environment m, would be by extending the observation model for different features, and then using a parameterization of the environment and a Rao Blackwellized particle filter, the factorization used in 5.43, to estimate these parameters. The parameterization is:

$$\vec{w} = \begin{bmatrix} \vec{S} \\ \vec{O} \\ l \\ r \end{bmatrix} \tag{5.44}$$

Where $\vec{S}$ gives the starting point of the straight section of pipe, $\vec{O}$ the roll,pitch and yaw angle of the pipe, l gives the length of the pipe and r gives the radius of the pipe. This can be expanded by assuming the vessel is in a straight section of pipe, and this section will alway end in a feature of some kind, for instance a bend or T-junction. Such a feature can then be described by a integer number for the type of feature, and an angle describing the roll angle around the pipe center axis. This roll angle is already present, because in the roll-pitch yaw description of the straight section of pipe, the roll angle has no meaning. This can now be given meaning by describing the roll of angle of the feature. So the state vector for the environment will now become:

$$\vec{w} = \begin{bmatrix} \vec{S} \\ \vec{O} \\ l \\ r \\ T \end{bmatrix} \tag{5.45}$$

in which T is the integer number describing the type of the feature. The particle filter for vessel state estimation must now use the extended ray-casting algorithms on every particle of the particle filter. Especially the ray-cast algorithm of a 90-degree bend (Section 5.4)is very time consuming, making the particle filter very slow. It was experienced that a single time step can take over 15 seconds to analyse with the particle filter. Also, using this method, only one timestep of measurements is taken into account for detection of features. This makes it very hard to succesfully detect features, with the noisy data from the range finder.

5.3.2 Mapping with a Grid Based Method

Now a grid based method of mapping will be discussed. An occupancy grid is a datagrid in which each cell represents the probability that this cell is occupied in the map. (An object is present). This probability is updated every time new information from measurements becomes available. The elegance in this method is that estimates of the map are updated only locally where new information has become available. Some variants of grid based methods use a grid for each particle of the particle filter. This done because often odometry of a vessel is subject to drift, meaning that the orientation estimate drifts are the vehicle moves. In this project, there will not be much drift from odometry, which is one dimensional. Therefore it is chosen to use one single global map of the pipe interior.

5.3.2.1 Grid Type

A grid has to be made of the local pipe network, consisting of a section of pipe with a feature attached. A full 3D grid of a pipe will not give much extra information, it is chosen to make a 2D grid which is projected on the wall of the pipe. The ray casting algorithm will calculate the ray length which hits the wall of a straight section of pipe. If a feature is present, then the measured ray will deviate from this value. This can be used to alter grid cell values, making them more positive if a calculated ray is longer than the measured ray, and making them more negative if the calculated ray is shorter than the measured ray. The equation used to update each cell C is basically a low pass filter:

$$C_k = K \cdot C_{k-1} + (1-K) \cdot (|\vec{r}_c| - |\vec{r}_m|) \tag{5.46}$$

where K is a "memory" constant, with which the update can be made more or less significant, $|\vec{r}_c|$ is the calculated ray length, and $|\vec{r}_m|$ is the measured ray length. The cell value will be gradually updated as more measurements of that coordinate become available. the update speed depends on the value of K, if $K = 1$, then no updates will be made and if $K = 0$ the new measurement will fully replace the cell content. Figure 5.6 shows an example of such a map. The color of each cell gives represent the cell value. The red spot in the map indicates a dent in the pipe wall..

5.3.2.2 Detecting signatures

Using this method, each feature will give a certain signature to the map. These signatures can be stored in memory, and during the state estimation, the map can be updated and signatures can be correlated to the map. The map wraps around the pipe, so in 2D, the map is periodic in one direction. If a feature has an other roll angle than the signature, this will automatically be resolved by making use of the periodicity of the map. The correlation will find the the "phase"

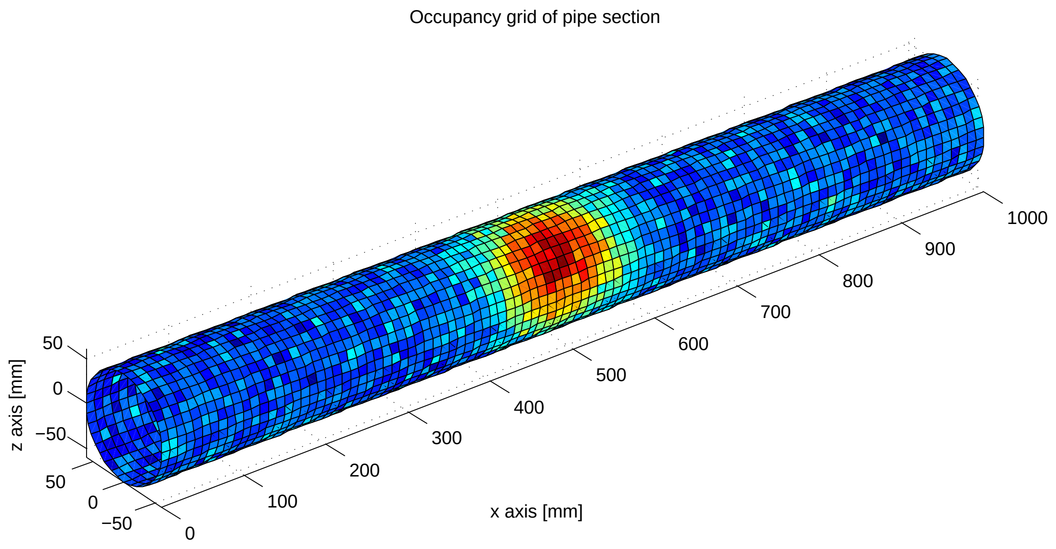


Figure 5.6: Example of a pipe occupancy grid map, red spot indicates dent (Note that the grid color in the picture was not simulated, it is just an example of what it could look like)

information, which will tell the orientation of the feature. Figure 5.7 shows a flat version of the pipe wall occupancy grid, wrapped around several times to show the periodicity.

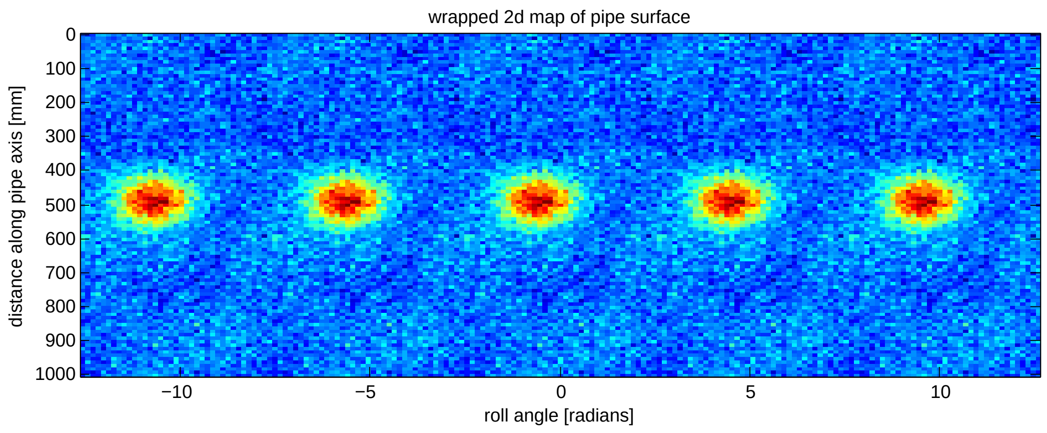


Figure 5.7: 2D occupancy grid of the pipe surface, rolled out and repeated several times

5.3.2.3 Obtaining Pipe Feature Templates

To obtain templates of different features, two methods are a possibility. The first method is measuring features using the sensor vessel, and storing the feature part of the map. This is not considered ideal, because the sensor measurements are noisy, which add to the uncertainty of the map signature. The second method is simulation. The ray cast algorithm can be used to simulate rays originating from a vessel moving exactly along the pipe axis. If the observation model then incorporates bends and T-junctions and other features, then the rays falling into those features can also be calculated, and compared to the simulated rays in the straight section of pipe. This way the same map can be generated but without the measurement noise. Section 5.4 will show the extension of the observation model with extra features, and Section 6.2 will show the simulation results of different signatures.

5.4 Observation Model Extension for 90-Degree Bends

To extend the observation model for 90-degree bends, the ray-casting algorithm must be able to calculate ray lengths inside bends. This is done by modeling the bend as a quarter torus, as seen in Figure 5.8. The ray casting algorithm now needs a equation to find a ray-torus intersection. Because a ray can intersect a torus at four points, this becomes a fourth order polynomial.

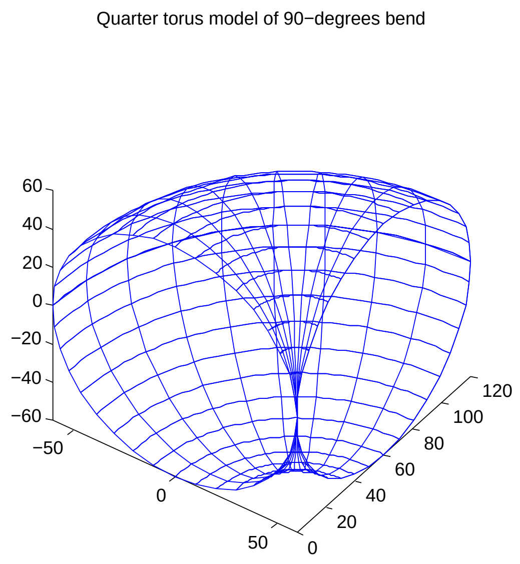


Figure 5.8: 90 degrees Bend model

A torus can be described using an inner radius R_i and an outer radius R_o. The inner radius is the radius of the pipe segment, and the outer radius is the radius of the circle describing the centerline of the pipe. Using this, the equation describing a torus lying on the x-y plane is:

$$(x^2 + y^2 + z^2 - R_i^2 - R_O^2)^2 + 4R_O^2(z^2 - R_i^2) = 0 \tag{5.47}$$

If we add the equation for the ray:

$$\vec{r} = \vec{P} + t\vec{D} \tag{5.48}$$

then this can be solved for t. It leads to a fourth order polynomial of the form:

$$at^4 + bt^3 + ct^2 + dt + e = 0 \tag{5.49}$$

in which:

$$a = (\vec{D}\cdot\vec{D})^2 \tag{5.50}$$

$$b = 4(\vec{D}\cdot\vec{D})(\vec{P}\cdot\vec{D}) \tag{5.51}$$

$$c = 4(\vec{P}\cdot\vec{D})^2 + 2(\vec{D}\cdot\vec{D})((\vec{P}\cdot\vec{P}) - R_i^2 - R_o^2) + 4R_o^2 D_z^2 \tag{5.52}$$

$$d = 4(\vec{P}\cdot\vec{D})((\vec{P}\cdot\vec{P}) - R_i^2 - R_o^2) + 8R_o^2 P_z D_z \tag{5.53}$$

$$e = ((\vec{P}\cdot\vec{P}) - R_i^2 - R_o^2)^2 + 4R_o^2 P_z^2 - 4R_o^2 R_i^2 \tag{5.54}$$

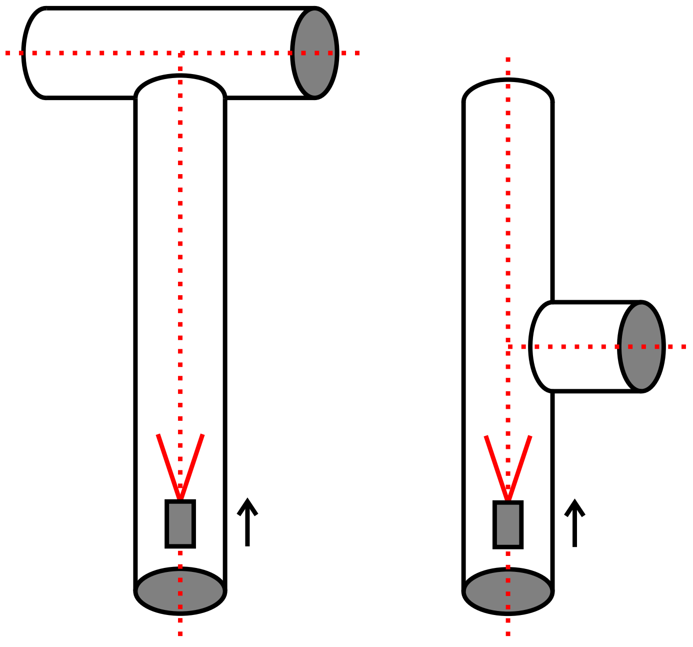
So now the polynomial roots have to be found, this can easily be done in Matlab with the command “roots”, but unfortunately it is a slow function. After the roots are found, it has to be checked which of the four roots is the correct one. Generally, if the ray does not intersect the torus, all roots will be complex. So complex roots can be discarded. In this situation, a ray comes from inside a straight section of pipe, and enters the quarter torus. Using this assumption, the largest value for t should be the value which we seek. The entire algorithm in short is as follows:

- Convert ray to torus coordinate system
- Calculate polynomial entries a, b, c, d, e.
- Calculate roots if polynomial.
- Discard complex roots.
- Find largest real root.

So after using the straigh pipe raycasting algorithm, rays should be selected which hit the ray at a x-coordinate outside of the pipe. These rays should then be ray-casted again with the bend-ray casting algorithm presented here.

5.5 Observation Model Extension for T-junctions

Two types of T-junctions can be encountered. These two types are shown in Figure 5.9. A T-junction can be encountered from the “center” pipe as in Figure 5.9(a), or from one of the side branches as in Figure 5.9(b).



(a) T junction type 1 (b) T junction type 2

Figure 5.9: Two types of T-junctions can be encountered, (a) and (b) show these two types.

5.5.1 Ray Cast in Type 1 T-junction

If type 1 T-junction is encountered, as in Figure 5.9(a), then the ray-cast algorithm must work as in a normal section of straight pipe, until a ray endpoint comes within the radius of the perpendicular pipe section of the T-junction. If this happens, the ray cast must use the perpendicular pipe for its calculation of ray length. The algorithm in short is:

- Calculte ray length where $y^2 + z^2 = rad^2$ (rad = pipe radius)
- If $(x - l)^2 + z^2 < rad^2$ or $x > l$: calculate ray in second pipe. (l = length of first pipe until T-junction)

All coordinates are transformed into pipe-coordinate system.

5.5.2 Ray Cast in Type 2 T-junction

For a type 2 T-junction, Figure 5.9(b), almost the same algorithm can be used:

- Calculte ray length where $y^2 + z^2 = rad^2$ (rad = pipe radius)

- If $(x-l)^2 + z^2 < rad^2$ and $y < 0$: calculate ray in second pipe. (l = length of first pipe until T-junction)

5.6 Template Matching

To detect features in the pipe, a template matching algorithm is needed. This algorithm will have to match different templates to a measured occupancy grid map. Different measures exist for such a algorithm, such as correlation, or sum of absolute differences (SAD). One problem is that often a map does not contain the full template, because the vessel did not move far enough to map all information about the feature, or if the vessel is not exactly centered in the pipe, also some parts will not be mapped. Therefore, a partial template must also give a (partial) measure.

5.6.1 Requirements on Template Matching Algorithm

The template matching algorithm must comply to the following requirements:

- Noise: The algorithm must be made robust to noise.
- Partial map: Provisions must be made to handle a partial template match.
- Comparable: Measures of correspondence must be comparable between different templates. A sort of normalization is required on the measure.

5.6.2 Matching Measure

The measure chosen to find the correspondence, is the sum of absolute differences measure. This measure is as follows: Given a $n \times m$ map matrix M, and a $u \times v$ template matrix T, the measure becomes:

$$SAD(i,j) = \sum_{k=0}^{n-1} \sum_{l=0}^{m-1} |M(i+k, j+l) - T(i,j)| \tag{5.55}$$

A good correspondense between map and template means that the SAD measure has a low value.

5.6.3 Noise Immunity

The SAD measure is fairly susceptible to noise. Also, hills and valleys in a measured map could have different heights than the template. It is chosen to make a thresholded image of the map as well as the template. Also, the map is low-pass filtered to remove high-frequency noise. The new thresholded map $\tilde{M}$ is as follows:(lpf is the filter kernel)

$$\tilde{M} = \begin{cases} 2, & \text{if } M * lpf > 50 \\ 1, & \text{if } 50 \geq M * lpf > 10 \\ 0, & \text{if } -10 \leq M * lpf \leq 10 \\ -1, & \text{if } -50 \leq M * lpf < -10 \\ -2, & \text{if } M * lpf < -50 \end{cases} \tag{5.56}$$

The new thresholded template $\tilde{T}$ is:

$$\tilde{T} = \begin{cases} 2, & \text{if } T > 50 \\ 1, & \text{if } 50 \geq T > 10 \\ 0, & \text{if } -10 \leq T \leq 10 \\ -1, & \text{if } -50 \leq T < -10 \\ -2, & \text{if } T < -50 \end{cases} \tag{5.57}$$

Using the thresholded map and template makes the SAD measure more noise robust.

5.6.4 Partial Map Measure

Sometimes, a map is not yet measured everywhere. The entries of the map matrix in the unmapped sections will contain zeros. These zeros will result in a large error added to the SAD measure. This causes bad matches for features which are not fully mapped (which is often the case). One way to overcome this, is to keep a matrix in which each cell is 1 if the map M contains a measurement at that cell, or 0 if M is not yet charted at that location. This matrix is called B, and has the same size as M. The SAD measure now has to be adapted, to give a different weight to unmapped parts. If the unmapped parts are not used in the SAD measure, then a unmapped part which has the size of the template, will always give a perfect match. This match will then always be better than measured parts of the map that actually contain the feature of the template. It is chosen to give the difference found at unmapped cells a weight factor of 0.5. (Mapped cells are 1.0). So using the matrix B, containing 0's and 1's, the SAD now becomes:

$$SAD(i,j) = \sum_{k=0}^{n-1}\sum_{l=0}^{m-1} |\tilde{M}(i+k,j+l) - \tilde{T}(i,j)\cdot\left(1-\frac{1}{2}B(i+k,j+k)\right)| \tag{5.58}$$

Note that use is made of the fact that M contains 0's at the uncharted cells.

This makes the SAD measure handle partial mapped features.

5.6.5 Comparing of SAD Template Measures

A scaling of each SAD Measure is needed for every template. To demonstrate this, an example is given. Say a template for a bend is matched to a map and gives a SAD measure of 1000. Now say a template for a T-junction is also matched, and the SAD measure gives a value of 1500. One cannot determine yet which feature now fits better. To overcome this, a benchmark value is used. This value is the SAD measure of a template with a MAP containing all 0's. Every SAD measure is now divided by this benchmark value. So that a template matched with all 0 will result in a measure of value 1. A SAD measure of the template with itself will be zero. The new SAD, $S\tilde{A}D$, with benchmark value Z (as in zero) now becomes:

$$Z = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} |\tilde{T}(i,j)| \tag{5.59}$$

$$S\tilde{A}D(i,j) = SAD(i,j)\cdot Z^{-1} \tag{5.60}$$

There is one problem with this however, because of the partial map handling, a template match value with a unmapped section of a map, will result in $SAD\cdot Z^{-1} = 0.5$. So the best SAD measure on an entire map will always be smaller than 0.5. It is also preferred to have small values for a bad fit and high values for a good fit. Because the worst fit is always 0.5. Using the following transformation will give a new measure, called match:

$$\text{match}(i,j) = 1 - 2\cdot SAD(i,j)\cdot Z^{-1} \tag{5.61}$$

So now the entire measure function becomes:

$$\text{match}(i,j) = 1 - 2\cdot\frac{\sum_{k=0}^{n-1}\sum_{l=0}^{m-1} |\tilde{M}(i+k,j+l) - \tilde{T}(i,j)\cdot\left(1-\frac{1}{2}B(i+k,j+k)\right)|}{\sum_{k=0}^{n-1}\sum_{l=0}^{m-1} |\tilde{T}(k,l)|} \tag{5.62}$$

The entire algorithm is implemented in a Matlab mex file, which can be found in the Appendix D.

6 Simulation Results

In this chapter all simulation results of the state estimation and environment mapping are presented.

6.1 Simulation of the Vessel State Estimation

This section describes the simulation results obtained for the state estimation Particle Filter. Measurement results of the laser range finder are simulated using the Ray casting algorithm from Section 5.2.2.1.

6.1.1 State Estimation in Straight Section of Pipe

In this subsection, a simulation is presented which moves the vessel through a straight section of pipe with no features or disturbances. The performance of the particle filter is measured. First a known state is generated, and then the difference between the estimated state and the known state is calculated. The error measure is:

$$E = |\vec{x}_{est} - \vec{x}_{real}| \tag{6.1}$$

6.1.1.1 Effect of Changing Number of Particles

For the first test, the known state is centered in the pipe and the orientation is <0,0,0>. The vessel moves 20mm per timestep into the pipe. The simulation is performed several times with different number of particles to see the effect on performance if the number of particles is varied. Figure 6.1 shows the test result.

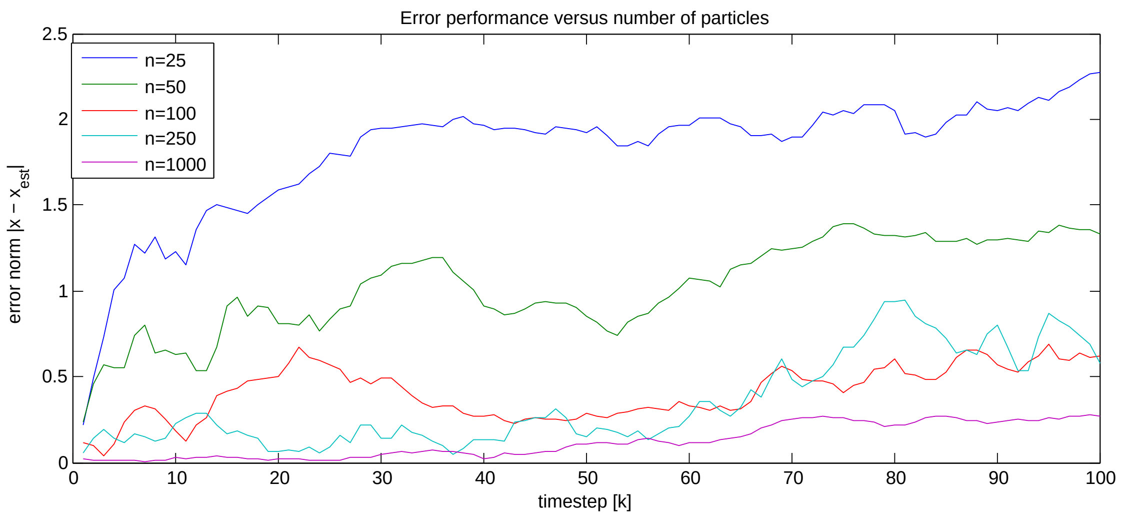


Figure 6.1: Error norm versus number of particles used

The parameters of the importance density were set according to the values in Table 6.1. Furthermore, the standard deviation of the gaussian in the observation model is set to 15 mm. ($p(r_k^i|x_k, A)$ from Equation 5.42). It can be seen that increasing the number of particles has effect until about 100 particles. After that, the increased number of particles does only make slightly better estimate.

6.1.1.2 Effect of Changing Importance Density Variance

Now the effect is observed of changing the variance of the importance density. If the variance is enlarged, particles will be more spread out, which will be beneficial if a lot of process noise is

Parameter:	Value:	Description:
σ_p	0.05	Standard deviation of position.
σ_o	0.005	Standard deviation of orientation.

Table 6.1: Parameters used for the simulation

present. In case of a large offset of the sensor vessel, increasing the variance of the importance density will make the particle filter adjust to the large offset faster. The price to pay is a less certain state estimate because of more particles with low weight. (The importance density is wider than the target density) The following test is performed with $N = 100$ Particles. Figure 6.2 shows the result of varying the standard deviation of the position and orientation in the importance density. As can be seen, the best estimate is gotten when the standard deviation is

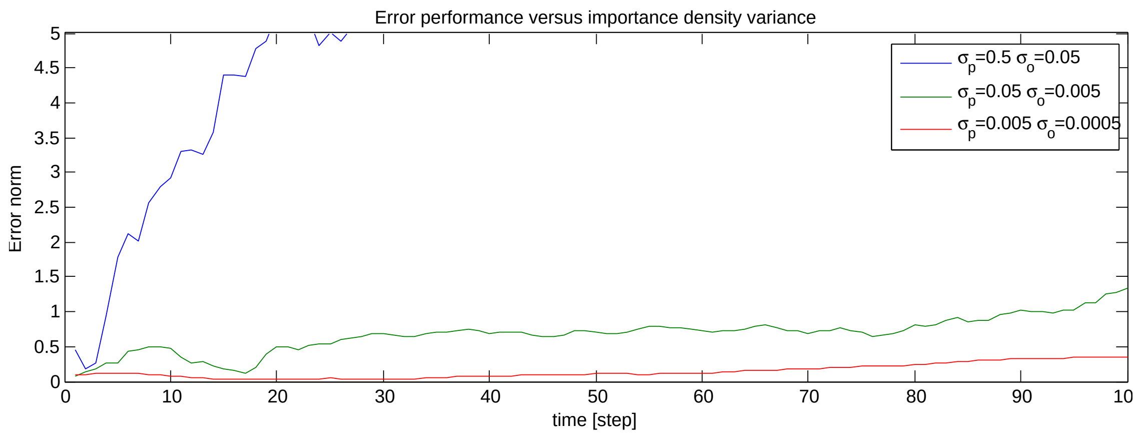


Figure 6.2: Error norm versus different standard deviations of the importance density

smallest. This is logical because the real state does not move, so no process noise is present. Choosing too small values for the standout deviations will cause the filter to be unable to track the process noise.

6.1.1.3 Tracking a disturbance

The last simulation will be to see tracking performance after a disturbance. At timestep $t = 30$, the position of the vessel is disturbed by $< 0, 10, 0 >$ $[mm]$. The orientation is disturbed by $< 0, 5, 0 >$ $[deg]$. The result is shown in Figure 6.3

As can be seen, the disturbance is tracked by the particle filter, but not very fast.

6.1.2 Conclusions of State Estimation Simulation

A number of variables can be changed which affect the performance of the particle filter. When testing the filter on the real measurement data, these parameters must be tuned to get best particle filter performance. Important parameters are:

- σ_o: Standard deviation of the orientation in importance density.
- σ_p: Standard deviation of the position in importance density.
- Standard deviation of the observation model.
- Number of particles.

Looking at the results, it seems that a number of 100 particles is suitable for this filter.

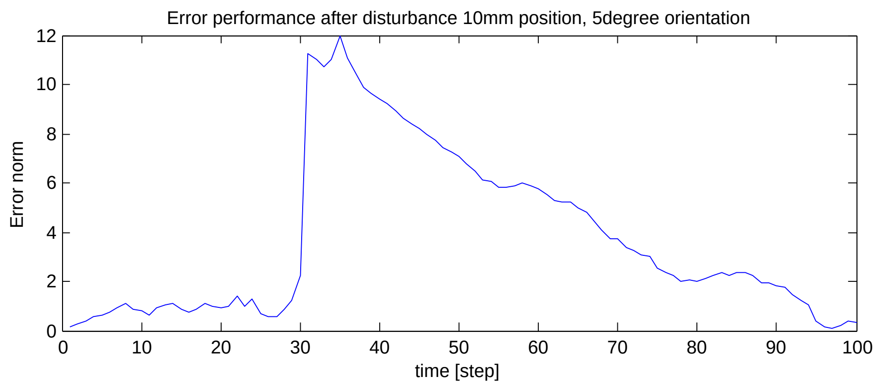


Figure 6.3: Error norm after a disturbance of the real state is induced at t=30. $\sigma_p = 0.5, \sigma_o = 0.005$

6.2 Environment Mapping

This section describes the simulated mapping of a occupancy grid. The vessel is moved in a simulated environment, exactly aligned with the pipe axis. The goal of this is to obtain templates for different pipe features like bends and T-junctions. First, the ray cast algorithm for the feature is tested and then the map is generated.

6.2.1 Template for 90-degrees Bend

In this subsection, the template for the 90-degree bend is obtained, using the observation model for the 90-degree bend. The ray cast algorithm from Section 5.4 is used. The first test is to see if the ray-cast algorithm works. For this, a vessel is moved through a pipe with a 90 degree bend at the end. All calculated ray end-points are stored, and displayed. Figure 6.4 shows the result of this. This result confirms the correct workings of the ray cast algorithm.

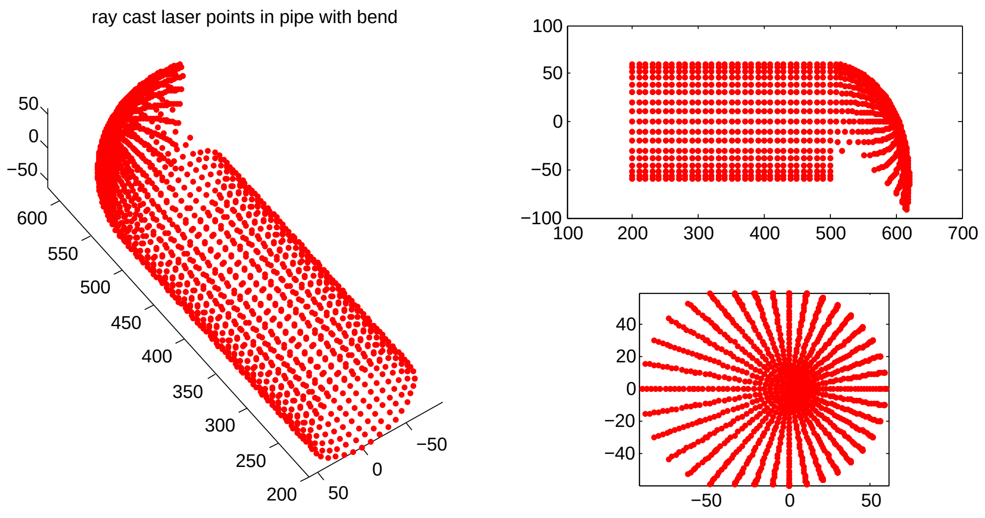


Figure 6.4: Test of the ray Cast Algorithm, 90 degree bend.

6.2.1.1 Mapping the Bend without Disturbance

Now a occupancy grid will be created of using the algorithm described in section 5.3. First a map of the bend will be made, with the vessel perfectly aligned with the pipe center axis. After that, the vessel position will be disturbed to see the effect on the map. Figure 6.5 shows the result of the mapping. The blue spot clearly indicates the opening in the pipe, and the red zone indicates the end-wall of the bend.

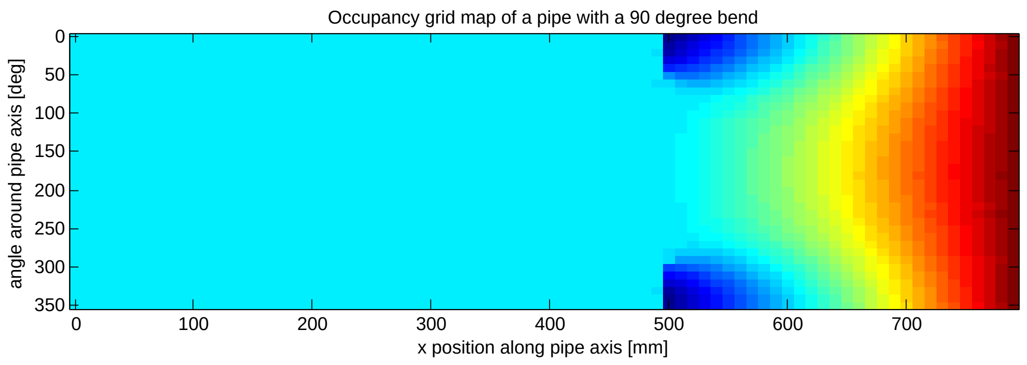


Figure 6.5: Simulated occupancy grid map, of a section of pipe with a 90-degree bend at the end. The vessel position is centered in the pipe with perfect alignment of the orientation.

6.2.1.2 Position Disturbance influence on Grid Mapping

Now the influence of position disturbance on the vessel position is considered. First the vessel position is shifted 10mm along the y-axis. Figure 6.6 shows the result. As can be seen, the characteristic features of the bend are still recognizable, but the image is a bit distorted.

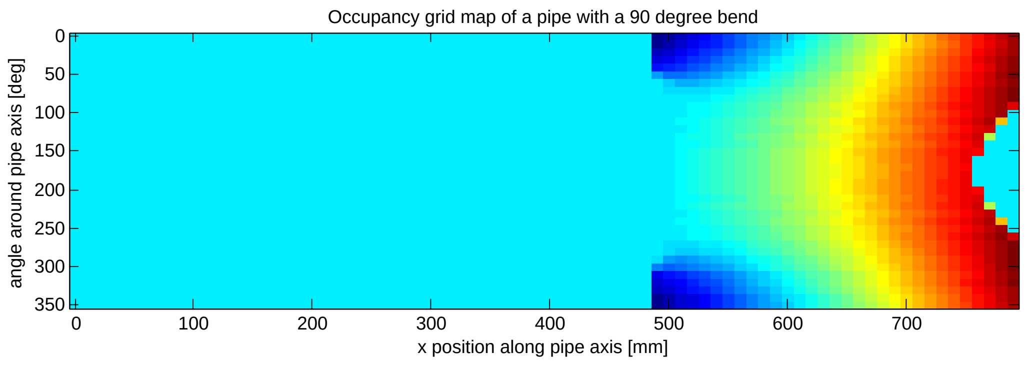


Figure 6.6: Simulated occupancy grid map, of a section of pipe with a 90-degree bend at the end. The vessel is position placed 10mm of-axis of the pipe axis, with orientation parallel to the pipe axis.

6.2.1.3 Orientation Disturbance influence on Grid Mapping

Now the influence of orientation disturbance on the vessel position is considered. First the vessel position is centered inside the pipe. The orientation is disturbed to $< 0, 10, 0 >$ $[deg]$. Figure 6.7 shows the result. As can be seen, the characteristic features of the bend are again recognizable, while the orientation angle is substantially disturbed.

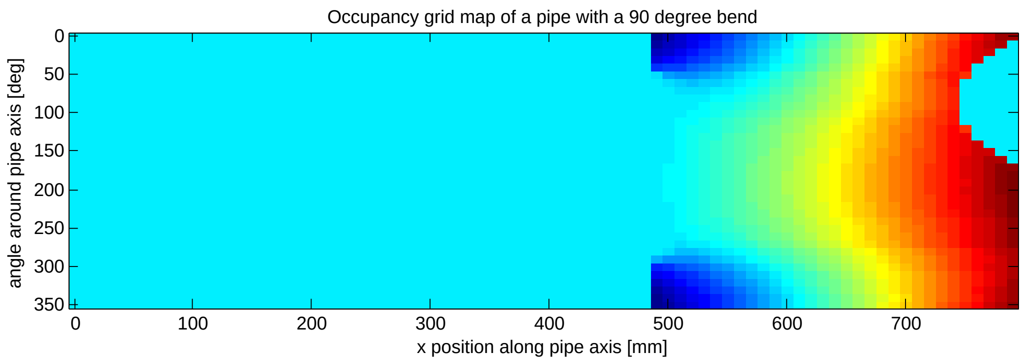


Figure 6.7: Simulated occupancy grid map, of a section of pipe with a 90-degree bend at the end. The vessel orientation is disturbed 10 degrees.

6.2.1.4 Conclusions of the Mapping of a Bend

It seems that the chosen method of mapping is fairly immune to disturbances of the vessel orientation and position. To test this, a subsection of Figure 6.5 is taken as a "bend template", and this template is matched with all three images, using a "sum of absolute differences" method. The template is shown in Figure 6.8. result of matching the three maps with the template image is shown in Table 6.2.

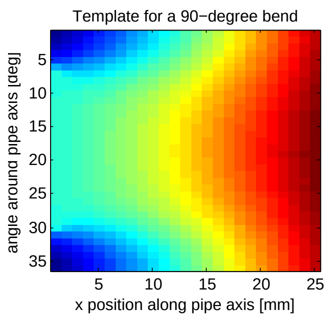


Figure 6.8: template of a 90-degree bend.

Map:	Match factor:	start of bend:	angle of Bend:
Figure 6.5	0.9084	500.25 [mm]	0.00 [deg]
Figure 6.6	0.9003	500.18 [mm]	-0.01 [deg]
Figure 6.7	10.8576	500.10 [mm]	0.04 [deg]
Bend rotated 45 [deg]	0.9451	500.33 [mm]	49.76 [deg]

Table 6.2: Correlation of the map with

The detection of the bend on these three simulated maps goes well. If the bend is rotated, however, the resolution of the grid map causes a error of $4.76[deg]$. The resolution can be increased, but this will give more computational costs. Also, more measurement points are needed to get a reliable grid.

6.2.2 Template for T-junction Type 1

In this subsection, the template for the T-junction is obtained, using the observation model for the T-junction type 1. (Figure 5.9(a)). First the ray cast model is tested by driving the vessel through a pipe with a T-junction type 1. Figure 6.9 shows the result of this simulation.

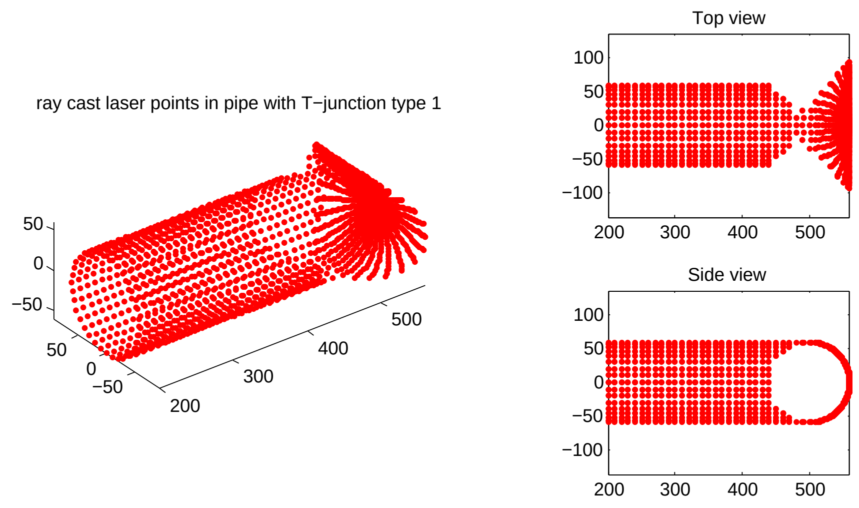


Figure 6.9: Test of the ray Cast Algorithm, T-junction type 1.

6.2.2.1 Mapping the T-junction type 1 without Disturbance

Now a occupancy grid is Simulated by driving the vessel through the pipe exactly aligned with the pipe axis. At 500mm a T-junction will start. Figure 6.10 shows the resulting map of the T-junction. Clearly, now two blue spots are visible (one in the middle and one wrapped around above and below).

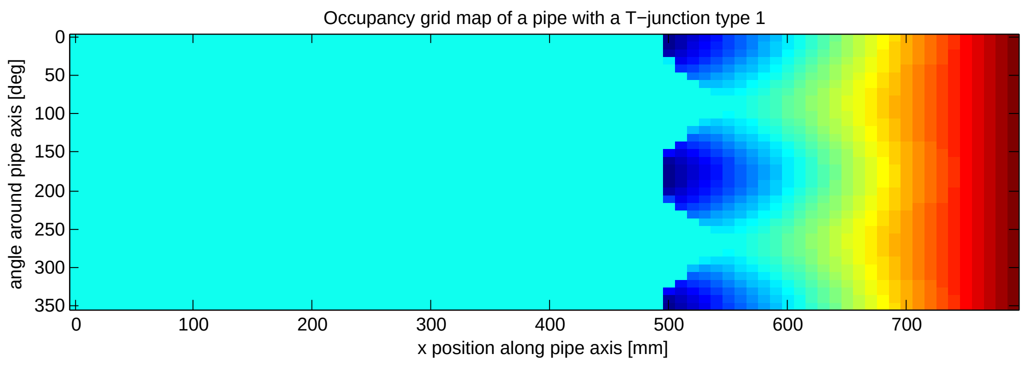


Figure 6.10: Simulated occupancy grid map, of a section of pipe with a T-junction type 1 at the end. The vessel position is centered in the pipe with perfect alignment of the orientation.

6.2.2.2 Position Disturbance Influence on Grid Map

Now a disturbance of the vessel position of 10mm from pipe axis is studied. The vessel is oriented parallel to the pipe axis. Figure 6.11 shows the result of this simulation.

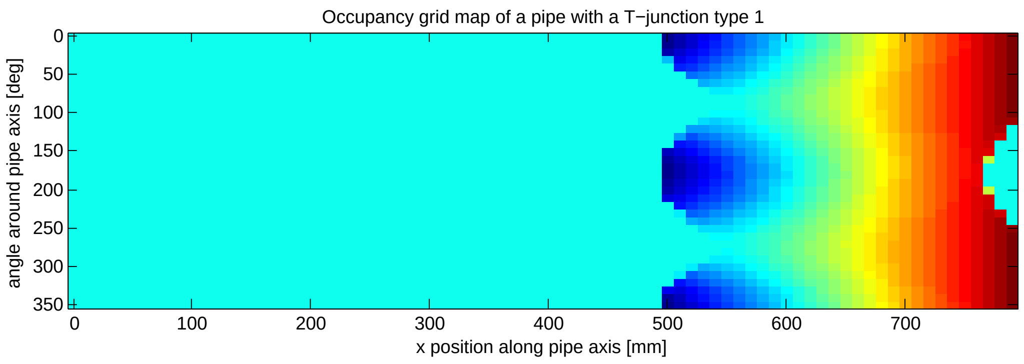


Figure 6.11: Simulated occupancy grid map, of a section of pipe with a T-junction type 1 at the end. The vessel is position placed 10mm of-axis of the pipe axis, with orientation parallel to the pipe axis.

6.2.2.3 Orientation Disturbance Influence of Grid Map

Now the orientation of the vessel is disturbed by $10[deg]$, just as with the simulation of the 90-degree bend. note that $10[deg]$ is a very large deviation considering the vessel physical properties.

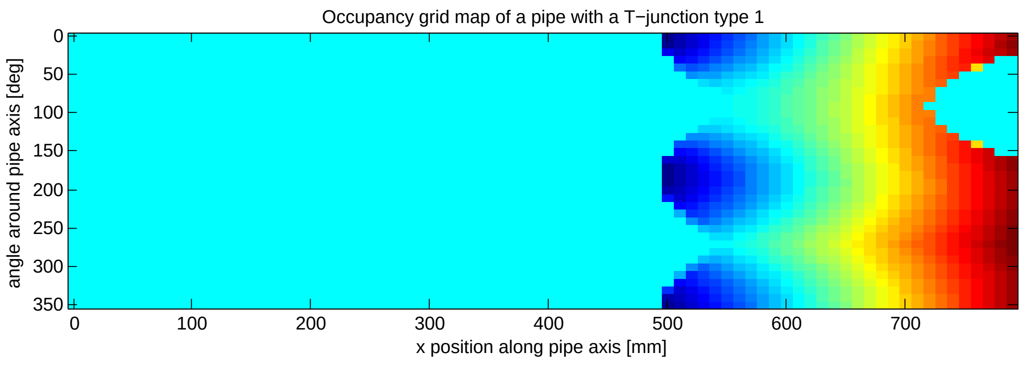


Figure 6.12: Simulated occupancy grid map, of a section of pipe with aT-junction type 1 at the end. The vessel orientation is disturbed 10 degrees from the pipe axis.

6.2.2.4 Conclusions of the Mapping of a T-junction type 1

The t-junction type 1, is now mapped. The map, just like the 90-degree bend map, seems to be fairly immune for disturbances. Figure 6.13 shows the template which has been taken from the undisturbed map.

Now the template will be matched with all three generated maps, and a additional map with the T-junction rotated around the pipe axis (roll axis) for $45[deg]$. The results are shown in table. The matching is again done by finding the minimal value of the “sum of absolute difference”.

As can be seen in Table 6.3, about the same results are obtained as with the 90-degree bend. The value of the template match is a bit different, but still good. The angle detected sometimes goes to 180 [deg], this is because one cannot make a distinction between the two orientation of 0 [deg] and 180 [deg] for the T-junction type 1. When the T-junction is rotated 45 [deg], the error because of the limited map resolution becomes 4.64 [deg].

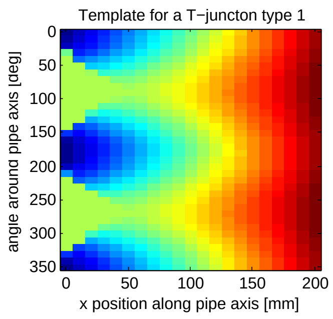


Figure 6.13: template of a T-junction type 1.

Map:	Minimal (s.o.a.d):	xpos T-junction:	angle of T-junction:
Figure 6.10	0.8494	500.42 [mm]	179.96 [deg]
Figure 6.11	0.7933	500.00 [mm]	179.96 [deg]
Figure 6.12	0.7955	500.10 [mm]	179.90 [deg]
T-junction rotated 45 [deg]	0.8787	500.41 [mm]	40.36 [deg]

Table 6.3: Correlation of the map with the T-junction type 1 template

6.2.3 Template for T-junction type 2

Now the T-junction type two will be simulated. First, the ray-casting algorithm is tested. Figure 6.14 shows the result. As can be seen, less laser points are now deviated from the straight pipe model.

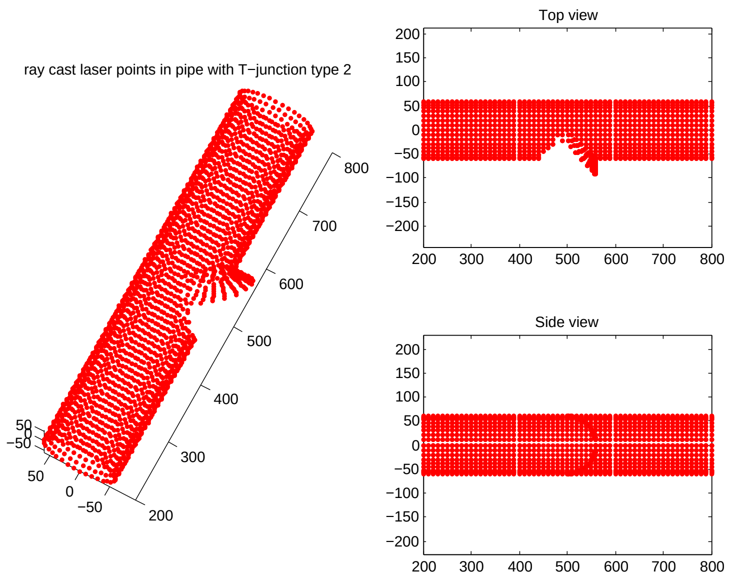


Figure 6.14: Test of the ray Cast Algorithm, T-junction type 2.

6.2.3.1 Mapping the T-junction type 2 without Disturbance

Now a occupancy grid is Simulated by driving the vessel through the pipe exactly aligned with the pipe axis. At 500mm a T-junction will start. Figure 6.15 shows the resulting map of the T-junction. Clearly, now two blue spots are visible (one in the middle and one wrapped around above and below).

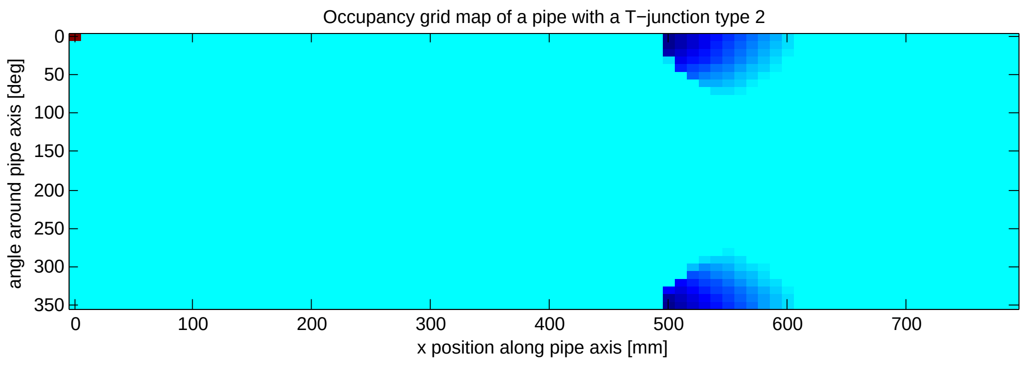


Figure 6.15: Simulated occupancy grid map, of a section of pipe with a T-junction type 2 beginning at 500 mm. The vessel position is centered in the pipe with perfect alignment of the orientation.

6.2.3.2 Position Disturbance Influence on Grid Map

Now a disturbance of the vessel position of 10mm from pipe axis is studied. The vessel is oriented parallel to the pipe axis. Figure 6.16 shows the result of this simulation.

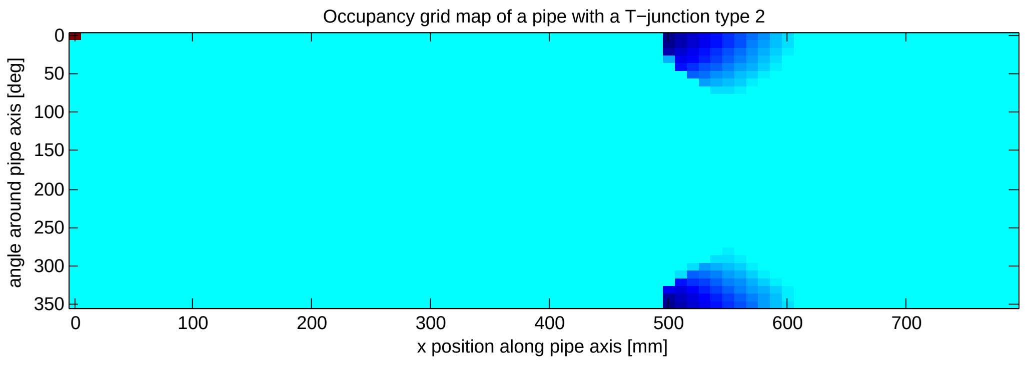


Figure 6.16: Simulated occupancy grid map, of a section of pipe with a T-junction type 2 starting at 500mm. The vessel is position placed 10mm of-axis of the pipe axis, with orientation parallel to the pipe axis.

6.2.3.3 Orientation Disturbance Influence of Grid Map

Now the orientation of the vessel is disturbed by $10[deg]$, just as with the simulation of the 90-degree bend.

6.2.3.4 Conclusions of the Mapping of a T-junction type 2

The t-junction type 2, is now mapped. The map, just like the 90-degree bend map, and the T-junction type 1,seems to be fairly immune for disturbances. Figure 6.18 shows the template which has been taken from the undisturbed map.

Now the template will be matched with all three generated maps, and a additional map with the T-junction rotated around the pipe axis (roll axis) for $45[deg]$. The results are shown in table.

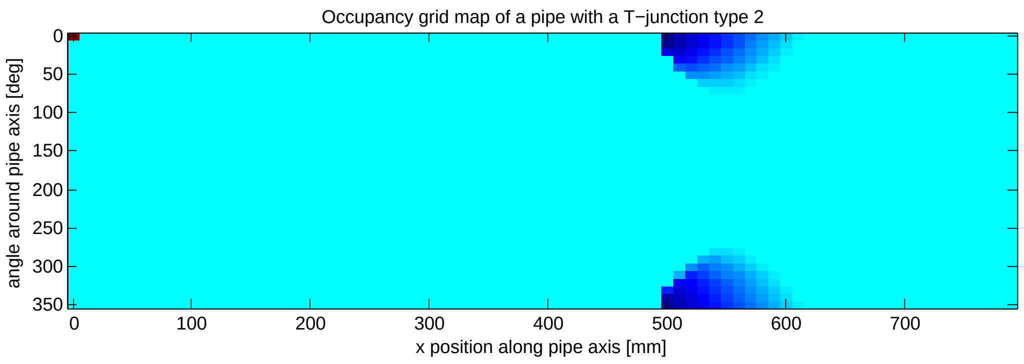


Figure 6.17: Simulated occupancy grid map, of a section of pipe with a T-junction type 2 placed at 500 [mm]. The vessel orientation is disturbed 10 [deg] from the pipe axis.

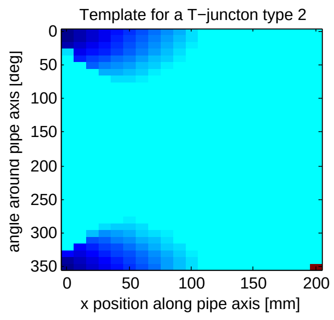


Figure 6.18: template of a T-junction type 2.

The matching is again done by finding the minimal value of the “sum of absolute difference”. The result is shown in Table 6.4.

Map:	match:	xpos T-junction:	Detected angle of Bend:
Figure 6.15	0.7055	500.64 [mm]	0.00 [deg]
Figure 6.16	0.6564	500.10 [mm]	-0.30 [deg]
Figure 6.17	0.7055	500.37 [mm]	0.48 [deg]
T-junction rotated 45 [deg]	0.8037	500.65 [mm]	48.99 [deg]

Table 6.4: Correlation of the map with the T-junction template

Again, the angular resolution is +/- 5 [deg], so the 45 degree bend is detected as a 48 degree bend.

6.2.4 Cross Matching Features

As seen in the previous section, templates can be generated from certain features in a pipe. Now it has to be tested how a template fits to all three features. If for instance a bend would make a better fit to a T-junction then the T-junction template, this would pose a big problem. Therefore, all templates are matched to all three features to see the sum of absolute difference values.

against Template⇓	Feature pipe + Bend	Feature pipe + T-junction 1	Feature pipe + T-junction 2
Bend	0.9084	0.2208	0
T-junction 1	0.0831	0.8494	0
T-junction 2	0	0	0.7055

Table 6.5: Cross-matching the templates with different pipe features

The results of cross-matching features with templates is shown in Table 6.5. As can be seen, the template algorithm makes a good distinction possible between features. In Chapter 7, this will be verified with real measurement data.

7 Experimental Results

This chapter describes the results of experiments performed with the sensor vessel inside a small pipe environment.

7.1 Environment Mapping

This section describe the experimental results obtained from testing the mapping of the pipe wall. First a map is made of a section of pipe which contains a 90-degree bend, then a T-junction type 1 and finally a T-junction type 2. The physical pipe setups are shown in Figure 7.1

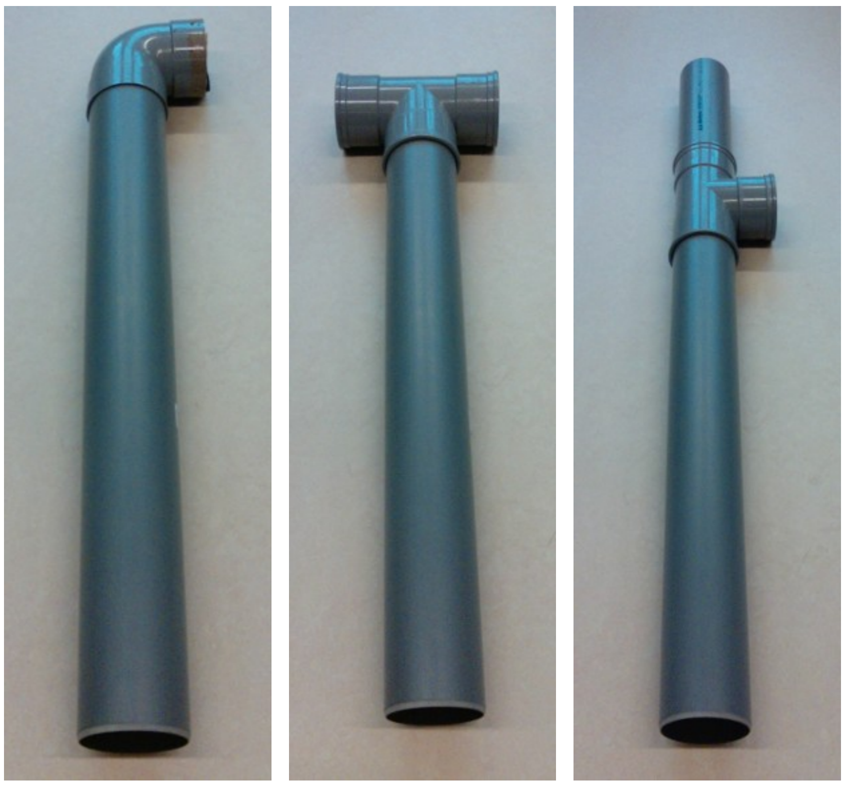

(a) 90-degree Bend (b) T-junction type 1 (c) T-junction type 2

Figure 7.1: Pictures of the three pipe configurations tested

As can be seen in Figure 7.2, the laser curve has some interruptions due to mechanical imperfections in the design of the sensor vessel. This will result in an incomplete map of the environment. To overcome this problem, the acceleration sensor data is used, to estimate the roll angle of the sensor vessel. The vessel is moved back and forth a few times into the pipe, and the vessel is rotated each time. This will result in overlapping the missing sections of sensordata.

7.1.1 Occupancy grid of a 90-degree Bend

The vessel is moved through the pipe back and forth several times. Each time the vessel is rotated a bit, to compensate for the missing parts of the laser curve. This makes sure that every angular section of the map is covered. Figure 7.3 shows the resulting occupancy grid map. Figure 7.4 shows a simulated Map with the template of a 90-degree bend inserted at the position where the best fitting feature is found. Although the measured map is missing information at some spots, a clear resemblance can be seen between both maps.

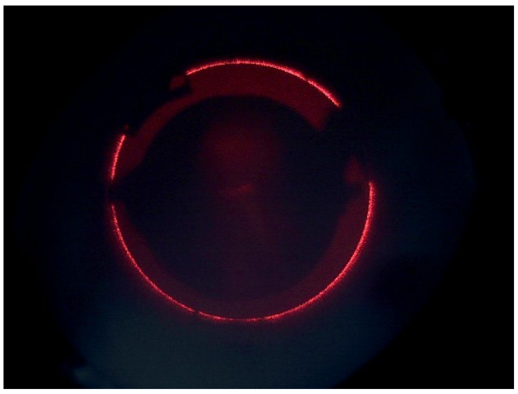


Figure 7.2: Shape of the laser curve

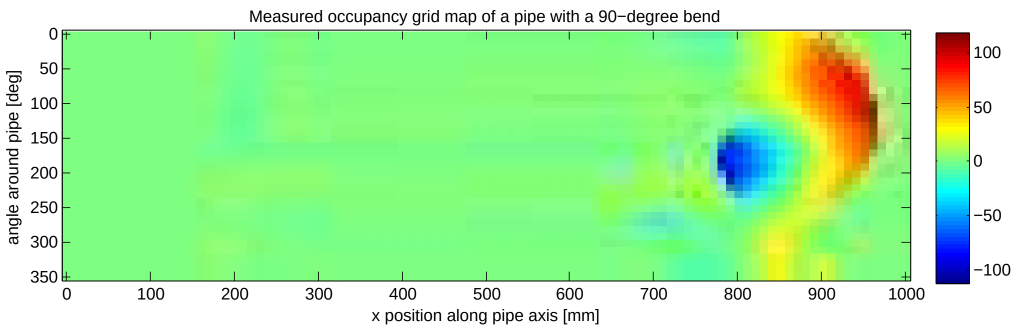


Figure 7.3: Occupancy grid measured by moving vessel several times back and forth in the pipe, with a 90-degree bend at the end.

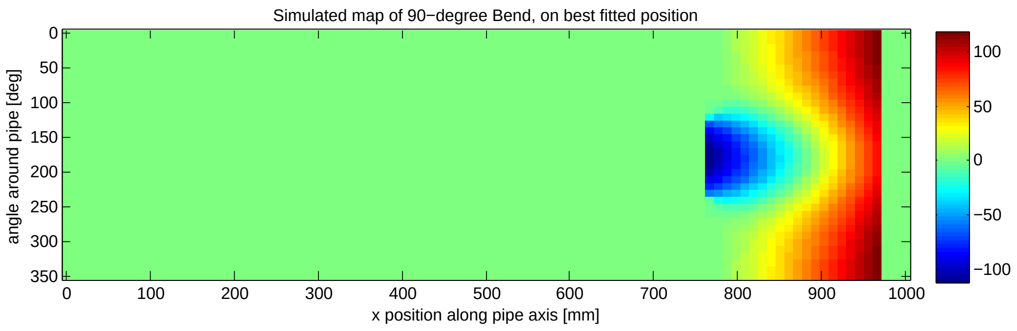


Figure 7.4: Simulated map of the best fitted template position of the 90-degree bend. (Fitted position: $x = 770.31$ [mm],$\varphi = 180.17$ [deg])

Figure 7.5 shows the fit factor between the templates and the map, as more and more samples of data are taken into account. As can be seen, the correct template, that of the 90-degree bend, matches much better than the other two. Around time step 100, the T-junction type two matches better. This is because this template is actually a subset of the Bend template, which fits better to the incomplete data. (The red parts of the map were not yet measured). As more data comes available, the fit for a T-junction type 2 again worsens, while the bend template gets better.

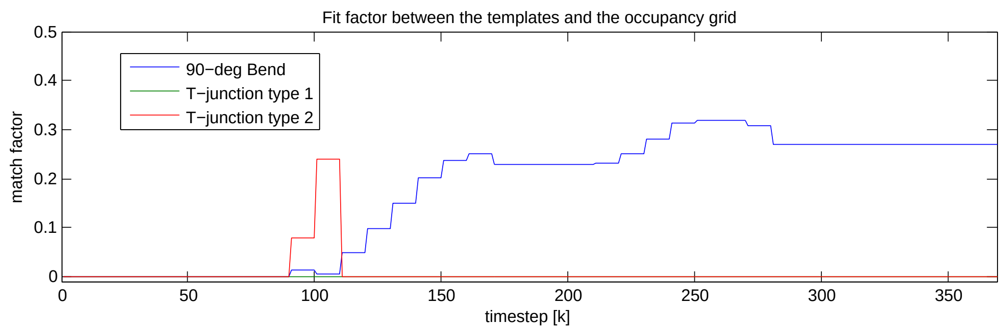


Figure 7.5: Dimensionless fit factor between templates of a 90-degree bend, T-junction type 1 and T-junction type 2. The correct template matches increasingly as more data comes available.

7.1.2 Occupancy grid of a T-junction type 1

With the same procedure as with the 90-degree Bend, a pipe with a T-junction type 1 is measured. Figure 7.6 shows the measures occupancy grid. Again the clear resemblance is seen with the simulated occupancy grid map, of Figure 7.7. The fit during measurements is shown in Figure 7.8 again finds the highest fit for the correct template. Also the template of T-junction type 2 becomes larger before the positive (red) parts of the map are measured. This is due to the partial overlap between the T-junction type 2 template and the other templates.

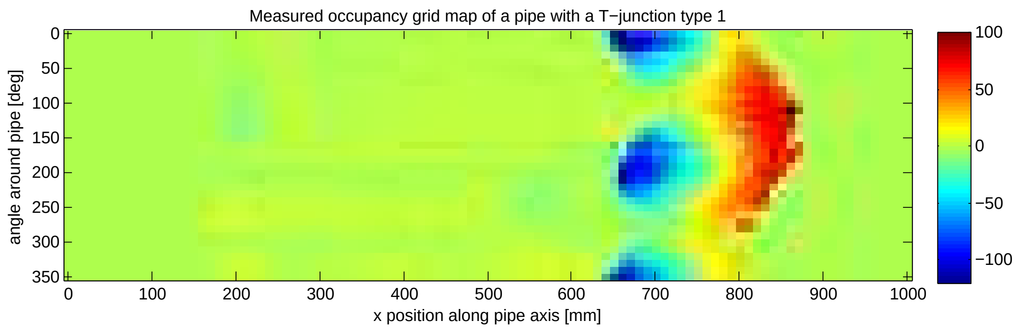


Figure 7.6: Occupancy grid measured by moving vessel several times back and forth in the pipe, with a T-junction type 1 at the end.

7.1.3 Occupancy grid of a T-junction type 2

The last feature to be measured is the T-junction type 2. First the occupancy grid is measured. The result is shown in Figure 7.9. The simulated map of a T-junction type 2 is shown in Figure 7.10, and the fitting progress of the templates as more measurements become available are shown in Figure 7.11.

7.2 Conclusions of the Experimental results

The experiments done in this chapter give a clear picture of the performance of the occupancy grid mapping with the particle filter and template matching combined. Although the match value of each template with a measured map is significantly less than the simulated results, still a clear match can be made. Typically the correct template match is around 0.4 or 0.5, while the match values of the incorrect templates goes to zero after enough data is gathered. (a match

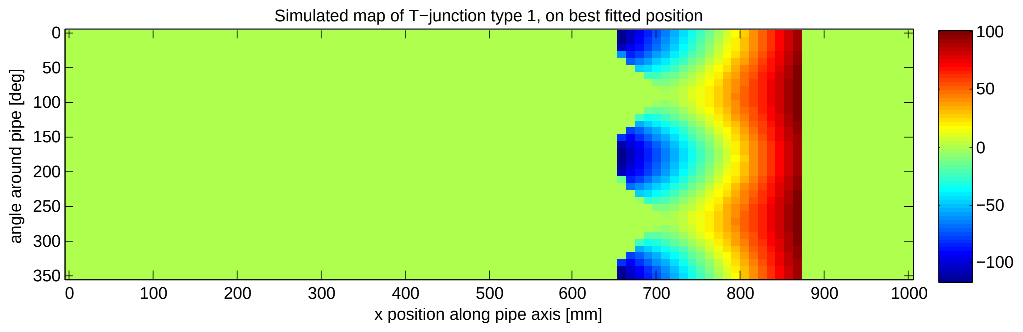


Figure 7.7: Simulated map of the best fitted template position of the T-junction type 1. (Fitted position: $x = 659.35$ [mm],$\varphi = 0.16$ [deg])

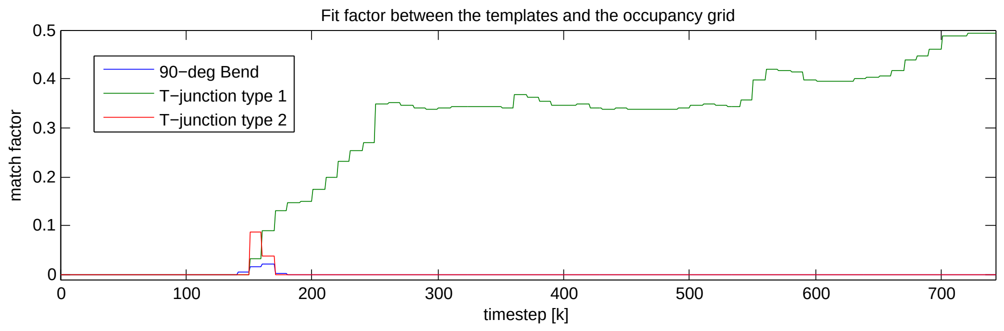


Figure 7.8: Dimensionless fit factor between templates of a 90-degree bend, T-junction type 1 and T-junction type 2. The correct template matches increasingly as more data comes available.

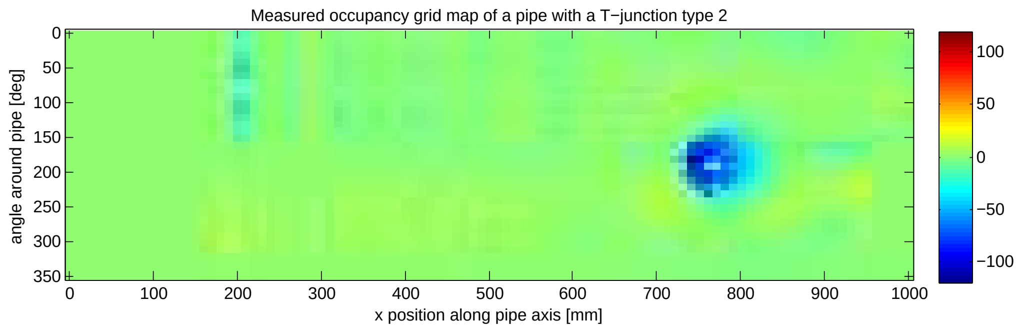


Figure 7.9: Occupancy grid measured by moving vessel several times back and forth in the pipe, with a T-junction type 2 at the end.

of zero means that the best match found in the map is a unmapped portion of the grid. See Section 5.6.4).

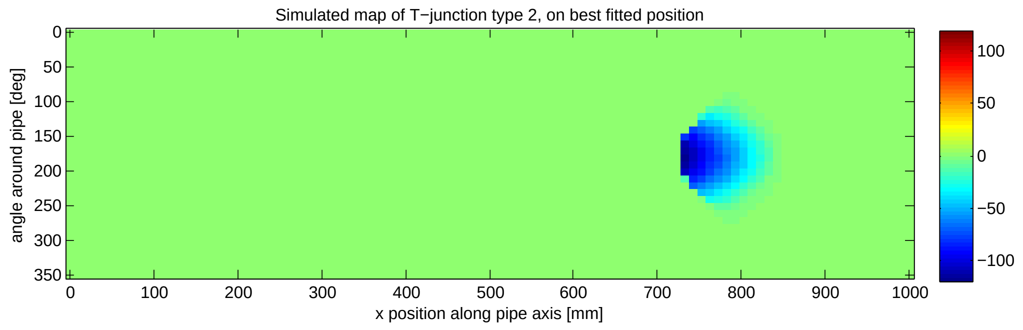


Figure 7.10: Simulated map of the best fitted template position of the T-junction type 2.(Fitted position: $x = 739.08$ [mm],$\varphi = 179.34$ [deg])

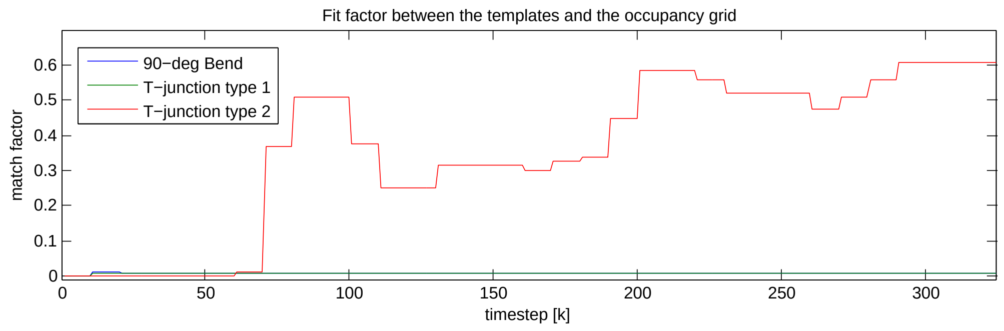


Figure 7.11: Dimensionless fit factor between templates of a 90-degree bend, T-junction type 1 and T-junction type 2. The correct template matches increasingly as more data comes available.

8 Conclusions and Recommendations

In this chapter the results of the research are discussed, and recommendations for future work are stated.

8.1 Discussion and Conclusions

This section describes the discussion over the research results. First the different components of the system are discussed.

8.1.1 SLAM

The SLAM algorithm developed in this research, is split up into two separate parts. A part which maps the inside of the pipe, and a part which performs self localization. Most SLAM algorithms in literature combine these two and use information from the map to get a better state estimate, and use information from the state estimate to get a better map. Normally, these two must be combined because otherwise large odometry drifts will occur. In this research, use is made of the fact that the vessel is inside a straight piece of pipe, this restricts the drift of the vessel position. Using this information, the state estimation works satisfactory for this task. For future research, Subsection 8.2.1 lists some adaptations to the particle filter to incorporate the map into the state estimation.

8.1.2 Mapping Performance

As can be seen in Chapter 7, the mapping in a real-world application suffers from noise. The template matching algorithm however is still capable of clearly determining the type of feature encountered. The resolution for the map used is 10[mm] translational by 10[deg] angular. This is a fairly low resolution, but this is done because the laser range finder contains a lot of noise.

8.1.3 Laser Range Finder

The laser range finder has been redesigned. The purpose of this redesign was to decrease the physical size of the sensor, to improve the speed of the laser curve extraction and to remove any moving parts. The physical size is indeed smaller, but not yet small enough to be fitted on PIRATE. The speed of the laser extraction algorithm has been increased by approximately a factor 10, from a little more than 8 seconds per frame to about 0.8 seconds per frame. This is important because this was a bottleneck for the SLAM speed. Also, the moving parts are replaced by diffractive optics, which is much smaller in size and uses no energy. The diffractive optics do introduce a finite resolution of the laser curve, but until now this has not been a problem. The entire resolution of the laser range scanner did suffer however, mainly because the new camera is of a low quality standard, with a high level of noise. Also a custom lens is added to the camera which decreases resolution. So concluding, new algorithms for the laser range finder have been introduced which increase speed, but the resolution has been decreased. This resolution influences also the SLAM performance.

8.1.4 Embedded

Most of the algorithms (almost all algorithms) currently are run on a desktop computer. In the future, all algorithms must run on an embedded target, and the power usage (CPU power) must be kept at a minimum. To see if the algorithms are suitable for embedded targets, the software will be listed into its components ,and the suitability for embedded systems will be listed in Table 8.1.

Software Component:	Part:	important for embedded:
Image processing	polar transform	floats ,trigonometry fcn's.
Image processing	gaussian filters	many nested loops o
Image processing	curve extraction	is a light algorithm.
Image processing	calc. 3D information	Table lookup
Sensor readout	Acceleration	Already on PIRATE
Sensor readout	Wheel rotation	Already on PIRATE
Particle Filter	Draw samples joint pdf	Need random generator
Particle Filter	Observation model	Parallel?, divisions, exp's
Occupancy grid	Adding measurements	is a light algorithm.
Occupancy grid	Template matching	many nested loops, float divisions

Table 8.1: Software components, with some properties important for implementation on an embedded target.

Probably the most difficult to implement on an embedded target is the particle filter. Exponential functions cost many CPU cycles, and every time step in evaluating $p(r_k|x_k)$, at least one exp, per laser ray(+/- 300), times the number of particles (100) is needed. That makes a total of at least 30.000 calculations of an exponential function per time step. Perhaps using the logarithm of the $p(r_k|x_k)$ gives a solution to this problem.

8.1.5 Conclusions

The goal of this research was to develop an algorithm which is able to perform SLAM inside a pipe network, and to demonstrate this by designing and building a sensor vessel and implementing the SLAM algorithms on it. It is concluded from the experimental results, that it is in fact demonstrated that it is possible to map the inside of a pipe, using only three sensors. The accuracy is not yet very high, but it is estimated that there is room for improvement by using a better visual sensor. Using a circular laser shape has proven to be highly efficient inside a pipe, with only one curve the entire pipe wall is mapped. If a better sensor is made or bought, then it will be easy to improve the resolution of the occupancy grid. Using a particle filter has proven to be quite suitable for this project. Especially the nonlinearities in the observation model (laser range finder) are handled well. PIRATE contains many states and measurements, a particle filter is an ideal candidate for fusing all this data into a single state estimate. Finally, it is concluded that although some parts need improvements, this research has delivered a good starting point towards a fully autonomous PIRATE robot.

8.2 Recommendations for Future Work

In this section, the recommendations for future work are given.

8.2.1 Adding map to Particle Filter

It is recommended that the occupancy grid map is to be used by the particle filter for state estimation. Most SLAM algorithms in literature work this way. A consequence is that every particle in the particle filter needs its own map. The advantage is that multiple hypothesis of possible maps are kept, and only the best fitting maps survive in the end. The ray cast algorithm should be adapted to use map data, many examples for the 2D case can be found in literature. (for instance, Grisetti et al. (2007a)).

8.2.2 Laser Range Finder

The size and quality of data of the laser range finder are still sub-optimal. The algorithms used are fast, but the camera hardware is noisy. Setting a too high exposure value will results in dots appearing in the image. Also it was observed that the camera becomes drastically more noisy as it is used for a long period of time. Therefore, a better camera is needed. It is recommended to make another iteration in the design of the laser range finder, to miniaturize the hardware (perhaps using beam-splitters) and to use a better lens configuration.

8.2.3 Increase Map Resolution

The resolution of the occupancy grid is fairly low at this time. If a better laser range finder is made, which preferrably fits on PIRATE, then it is recommended to increase the resolution of the map.

8.2.4 PIRATE State Estimation

At this time, Doggen (2010) is working on a wireless measurement system for pirate. This system read all sensors and setpoints of PIRATE realtime. It is recommended to use all these measurements of PIRATE in the particle filter. If a improved laser range finder is now added, it should be possible to get a accurate estimate of the position of PIRATE inside a pipe. This information can be fed to the algorithms developed by Reemeijer (2010) to maneuver PIRATE through bends and T-junctions, this comes very close to autonomous navigation.

A Various Scripts for Analysis Laser Range Finder

A.1 Matlab script for getting points on a given cone surface

```
%Script to calculate points on the surface of a cone
clear all;
A = randn(3,1);            %create a random lookat vector for the cone
V = randn(3,1);            %create random origin position of the cone
dmin = 0.050;              %begin diameter of the cone;
dmax = 0.130;              %end diameter of the cone;
alpha = atan(30/100);      %opening angle of the cone;
dot_dist = 0.005;          %distance between calculated points on
                                                  %cone surface

%calculate a rotation matrix from the lookat vector A
B = randn(3,1);
r_3 = A/norm(A);
r_2 = cross(B,r_3)/norm(cross(B,r_3));
r_1 = cross(r_2,r_3);
R = [r_1 r_2 r_3];
z1 = dmin/tan(alpha);
z2 = dmax/tan(alpha);
Pcone = [];      %create empty matrix for cone coordinates
for(z = z1:dot_dist:z2)
    radius = z*tan(alpha);
    theta=0:(dot_dist/radius):2*pi;
    %Append cone coordinates of this z-coordinate to the array
    Pcone = [Pcone [cos(theta)*radius;sin(theta)*radius;(z+theta*0)]];
end
%calculate the world coordinates
Pworld = R*Pcone + meshgrid(V,1:length(Pcone))';
%display result
plot3(Pworld(1,:),Pworld(2,:),Pworld(3,:));
axis equal;
```

A.2 Matlab Script to Capture Calibration Points

The following code belongs to the matlab file:

“FilesUsed/Calibration/Graphinput.m”

```
clear all;
im = imread('135mmCAL.bmp');
[xc,yc] = meshgrid(1:1280,1:960);

imshow(im);
for(i=1:18)

[x,y] = ginput(2);
x = round(x);
y = round(y);
```

```
W = 1−sum(im2double(im(y(1):y(2),x(1):x(2),2:3)),3);
W = W−min(W(:));
W = W/max(W(:));
W(find(W<0.6))=0;
W = W./sum(W(:));

mx(i) = sum(sum(W.*xc(y(1):y(2),x(1):x(2),:)));
my(i) = sum(sum(W.*yc(y(1):y(2),x(1):x(2),:)));
hold on;
plot(mx,my,'*');
hold off;
end;
%Array C contains the Coordinates. This should be
%saved to a .mat file with correct name.
C = [mx;my;120*ones(1,18)];
%Naming convention for saving variable
% C: 135mm.mat, 090mm.mat or XXXmm.mat
```

A.3 Create Coordinates: Matlab script for reading Calibration Data

The following matlab script called Create_coords.m, is used to read in all the calibration data files and combine those measurements into two arrays. Array “C” contains the camera coordinates of the calibration points, “w” contains the corresponding world coordinates of the calibration points.

```
%This script takes a set of .mat files and creates all world and camera
%coordinates out of it.

clear all;
th = (0:20:340)*pi/180;
u =[];
v=[];
z =[];
x=[];
y=[];
s = ['070mm.mat';'075mm.mat';'080mm.mat';'085mm.mat';'090mm.mat';'095mm.
    mat';'100mm.mat';'105mm.mat';'110mm.mat';'115mm.mat';'120mm.mat';'125
    mm.mat';'130mm.mat';'135mm.mat'];
diameters = [70;75;80;85;90;95;100;105;110;115;120;125;130;135];

for(i=1:size(s,1))

    C = load(s(i,:));
    %obtain the 18 camera coordinates
    u=[u C.C(1,:)];
    v=[v C.C(2,:)];
    r = diameters(i)/2;
    %create world coordinates belonging to the cone
    z = [z r/30*100*ones(1,18)];
    x = [x r*cos(th)];
    y = [y r*sin(th)];
```

```
end;

%make two arrays ,C is camera coordinates, W = world coordinates
C = [u;v];
W = [x;y;z];
```

A.4 Function for Pixel to World Coordinate Mapping:makeMapping

The following function is ran to create the matlab interpolation mapping functions used to translate between polar camera coordinates and polar Cone surface coordinates.

```
function [FR, FPhi] = makeMapping()
Create_coords;
mid = mean(C,2);
rw = sqrt(sum(W(:,:).^2));
tmp1 = atan2(W(2,:),W(1,:));

tmp2 = sqrt(sum(W(1:2,:).^2));
aw = tmp1.*tmp2./rw;
ac = atan2(C(2,:)-mid(2),C(1,:)-mid(1));
rc = sqrt( (C(2,:)-mid(2)).^2+(C(1,:)-mid(1)).^2);

AW = (reshape(aw,18 ,14)');
RW = (reshape(rw,18,14)');
AC = (reshape(ac,18,14)');
RC = (reshape(rc,18,14)');
AC(:,10) = AC(:,10) -2*pi;
AW = [AW(:,11:end) AW(:,1:10)];
AC = [AC(:,11:end) AC(:,1:10)];
RW = [RW(:,11:end) RW(:,1:10)];
RC = [RC(:,11:end) RC(:,1:10)];
K1 = (tmp1(11:18:end)+2*pi).*tmp2(11:18:end)./rw(11:18:end);
K2 = (tmp1(10:18:end)-2*pi).*tmp2(10:18:end)./rw(10:18:end);
AW(:,19)=K1;
AW = [K2' AW];
AC(:,19)=AC(:,1)-2*pi;
AC = [(AC(:,18)+2*pi) AC];
RC = [RC(:,18) RC RC(:,1)];
RW = [RW(:,18) RW RW(:,1)];

%t = 0:251;
%ac2 = atan2(sin(-t*2*pi*20/360+ac(1)),cos(-t*2*pi*20/360+ac(1)));
%ac = ac2;

%for(j=1:14)
%    rc((j*18-17):(j*18))=mean(rc((j*18-17):(j*18)));
%end



FR=TriScatteredInterp(RC(:),AC(:),RW(:),'natural');
FPhi=TriScatteredInterp(RC(:),AC(:),AW(:),'natural');
```

```
%FR = TriScatteredInterp([RC' angC'],RW','natural');
%FPhi = TriScatteredInterp([RC' angC'],angW','natural');
```

A.5 Script to Verify the Calibration Procedure

The script in this Section is called “TransformImage.m”. It is used for verification of the calibration procedure. This verification has been done by projecting a circle on a plane perpendicular to the projector and measuring the diameter of the circle first. The diameter is found to be 104 mm using a sliding gauge.

```
clear all;
Create_coords;
mid = mean(C,2);
im = im2double(imread('FlatPlateR104.bmp'));

red = im(:,:,1)+im(:,:,2)+im(:,:,3);
red = red(1:2:end,1:2:end);
%red(find(red<0.4))=0;
%red = red - 0.4;
%red = red/max(red(:));
[xm,ym] = meshgrid(1:2:1280,1:2:960);
[p,r] = meshgrid(0:360,200:380);      %make polar plot coordinates
%convert image to polar coordinates;
Bpolar=interp2(xm,ym,red,r.*cos(p*pi/180)+mid(1),r.*sin(p*pi/180)+mid(2))
    ;

f = makeDiffKernel(100,5)';       %Make a second derivative of gaussian
    kernel.
   %                    This is for blob detection.
f2 = fspecial('gaussian',[50 1],3);
Bpolar = imfilter(Bpolar,f2);
Blobdet=imfilter(Bpolar,f,'replicate');

[a,b]=min(Blobdet);
imagesc(0:360,200:380,Blobdet);
xlabel('Angle␣[deg]');
ylabel('Radius␣[pix]');
title('Polar␣transformed␣and␣filtered␣Image␣of␣laser␣curve');
%t  =[174:183 206:244 319:355];
valid = [0:109 112:173 186:205 246:318 356:360];
i1 = max(1,b-10);
i2 = min(181,b+10);
for(i=1:361)
     rad(i) = sum(Bpolar(i1(i):i2(i),i)'.*((i1(i):i2(i))+199))./sum(Bpolar
         (i1(i):i2(i),i));
end
%hold on;
%plot(valid,rad(valid+1),'.');
%hold off;
```

```
%plot(b)
%r = ones(1,length(t))*160;
%imshow(red);

%hold on;
%plot(mid(1)/2 + r.*cos(t*pi/180),mid(2)/2 + r.*sin(t*pi/180) ,'.');
%hold off;
idx = find(a<-0.005);
radius = rad(idx);

ph = (idx-1)*pi/180;
ph(find(ph>pi))=ph(find(ph>pi))-2*pi;
[fR,fA] = makeMapping();

rcalc = fR(radius,ph);
acalc = fA(radius,ph);
idx = find(isnan(rcalc)==0);
rcalc = rcalc(idx);
acalc = acalc(idx);
z = rcalc/sqrt(1+900/10000);
r = z*30/100;
ang = acalc.*rcalc./r;
x = r.*cos(ang);
y = r.*sin(ang);

x2 = zeros(1,length(x)*2);
y2 = x2;
z2 = x2;
x2(1:2:end)=x;
y2(1:2:end)=y;
z2(1:2:end)=z;

subplot(2,2,[1 3]);
plot3(x,y,-z,'.','color','red');
hold on;
plot3(x2(1:11:end),y2(1:11:end),-z2(1:11:end));
axis equal;
title('3D␣Result␣calibration␣verification');
xlabel('x␣axis␣[mm]');
ylabel('y␣axis␣[mm]');
zlabel('z␣axis␣[mm]');
hold off;
subplot(2,2,2);
plot(x,y,'.','color','red');
axis equal
title('top␣view');
xlabel('x␣axis␣[mm]');
ylabel('y␣axis␣[mm]');
subplot(2,2,4);
```

```
plot(x,-z,'.','color','red');
axis equal
title('side␣view');
xlabel('x␣axis␣[mm]');
ylabel('z␣axis␣[mm]');
```

B Interfacing Electronics and Software

This chapter describes the interfaces of the sensor vessel in more detail.

B.1 Microcontroller

This section describes the microcontroller hardware which is used to read the optical encoder and to power the LED lights at the front side of the vessel. The hardware PCB is the same as the motorcontroller PCB's used on PIRATE. this hardware contains two PWM outputs and two encoder inputs. One PWM output is used to control the LED's brightness, and one encoder input is used to read out tehe optical encoder placed on one of the wheels.

B.1.1 Microcontroller firmware

The already existing firmware of the PIRATE motorcontroller has been altered, to accomodate this project. Normally communications would run throught I2C, but now the microcontroller UART is used. Also, the encoder counter was only 8-bit, this is changed into a 32-bit signed integer value.

B.1.2 Communication protocol

The communication protocol with the microcontroller is very easy. A 8-bit value must be sent to the controller, and it immediately returns a 32-bit value. The value sent represents the wanted brightness of the LED's, the value returned is the encoder counter value.

B.2 Data Acquisition Software

The data acquisition software gathers all the sensor data. The software is written in c++. Table B.1 lists the details of the main tools used.

tool type:	tool used:	version:	addon:
operating system	Ubuntu	10.04	-
C++ IDE	eclipse	galileo	CDT
C++ compiler	gcc	Ubuntu 4.4.3-4ubuntu5	-
Library	OpenCV	libcv4	Custom alteration

Table B.1: Software tools used

The data acquisition software is all inside a single executable file called "CaptureCamCV". The program consists of a main and four classes. The classes are described in the next subsections.

B.2.1 Class: ContinuousSerialport

This class is used to interface with the serial port connected to the Xsens module. The mode of operation of the Xsens IMU is that it sends measurement data frames periodically, at a rate of 100 frames per second. The xsens is setup such, that it sends acceleration data only, consisting of three 32-bits floating point numbers. Including header data, this amounts to 17 bytes of data per frame. Reading out a serial port in linux can be done by two methods. The read function can wait for a specified number of bytes to arrive, or the read function can just read the bytes that are currently available in the buffer and move on. For this class, the variant is used where the read function does not wait. The basic mode of operation is now described.

B.2.1.1 Basic Mode of Operation of ContinuousSerialPort Class

This class relies upon the fact that its read function will be called periodically by a thread, and the read function puts the data available in a large FIFO buffer. The thread checks the number of bytes inside the FIFO buffer, and if this is more than 400 bytes, the thread will call the "AnalyseData" function of the class ContinuousSerialport, which then reads all bytes inside the FIFO buffer and analyzes the data to extract Xsens dataframes. The data from frames is placed in a 3-float array. If multiple frames are found inside the FIFO buffer, the newer data overwrites the older data. The "AnalyseData" function works as a state machine, of which the current state is saved in class a member variable. This has the benefit, that the function works even if only half a dataframe is present in the fifo buffer. The next time "AnalyseData" is called, it will continue were it was left. Figure B.1 show the flowchart of the thread calling an object of ContinuousSerialport.

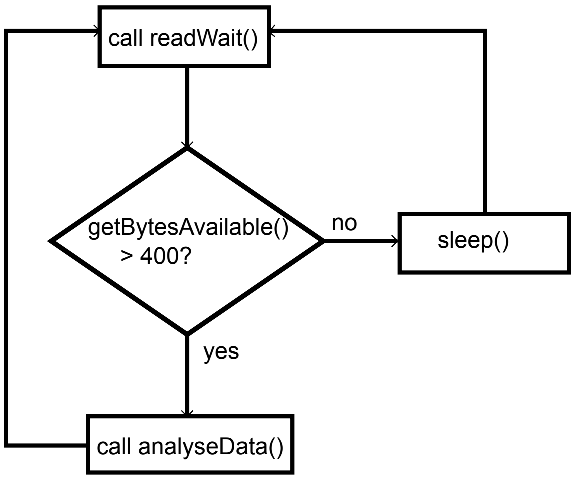


Figure B.1: Basic mode of operation of class ContinousSerialport. A thread from the main program continuously calls the object functions.

B.2.1.2 AnalyseData statemachine

The function AnalyseData extracts the Xsens data from received dataframes. Figure B.2 shows such a dataframe. First a preamble 0xFA,0xFF is always sent at the beginning of the frame. Then a byte follows which identifies the type of data whioch is sent. The next bytes gives the number of data bytes to expect, after that the databytes follow and the frame is closed with checksum byte. The checksum makes the sum of all bytes 0, if an 8-bit adder is used.

0xFA	0xFF	ID	LEN	D0	D1	..	DN	CHSUM

Figure B.2: A typical dataframe from the xsens sensor.

The state machine is ran on every byte in the FIFO buffer when it is called. The state machine is implemented with a switch statement, and the source code of this switch case is shown here:

```
switch (analyseState)
{
        case 0:         //No preamble found yet;
        {
                tmpCnt =0;
                byte = readNext();
                if(byte == 250) //Looks like a preamble...
                        analyseState = 1;
```

```
            else
                analyseState = 0;
        break;
        }
        case 1:         //Preamble could be found here;
        {
            byte = readNext();
            if(byte == 255) //still looks like a preamble...
                analyseState = 2;
            else
                analyseState = 0;
            break;
        }

        case 2:         //Datatype should now be found to be 50;
        {
            byte = readNext();
            if(byte == 50)  //Ok, data type is also correct
                analyseState = 3;
            else
                analyseState = 0;
            break;
        }


        case 3:         //Datalength should now be found to be 36;
        {
            byte = readNext();
            if(byte == 12)  //Ok, datalength is consistent.
                analyseState = 4;
            else
                analyseState = 0;
            break;
        }

        case 4:         //All clear to start reading 36 bytes;
        {
            byte = readNext();
            tempdata[tmpCnt]=byte;
            tmpCnt++;
            if(tmpCnt==12)
            {
                analyseState = 5;
            }
            break;
        }
        case 5:
        {
            checksum = (255+50+12)%256;
            byte = readNext();
            checksum+=byte;
```

```
                for(j=0;j<12;j++)
                {
                        checksum+=tempdata[j];
                }
                if(checksum==0) //Checksum is OK,
                //now transport all tempdata to more permanent data
                {
                        for(k=0;k<3;k++)
                        {
        //Have to reverse the byte order because somepeople
        //somewhere, sometime, did not agree on this.
                        tst[0] = tempdata[k*4+3];
                        tst[1] = tempdata[k*4+2];
                        tst[2] = tempdata[k*4+1];
                        tst[3] = tempdata[k*4+0];
                        *(XsensData+k)=*(float *)(tst);
                        }

                }
                analyseState = 0;       //Return to the first state,
                //waiting for a new preamble.
                break;
        }
        default:
        {
                break;
        }
}
```

This describes the most important functions of the class ContinuousSerialport. The rest of the source code can be found in the file listed in Table G.1.

B.2.2 Class: SerialPort

This class is used to communicate with the microcontroller board. The serial port read mode for this class is the waiting mode variant. This means that when the read function is called, the function waits until the specified number of bytes are received. In the initialization of objects of this class, the number of bytes to be read is set to 4. A thread in the main program contains a while loop which continuously calls the read function. This function then waits for data until the main program sends a byte to the microcontroller using this class. The microcontroller responds with four bytes, representing a 32-bits signed integer of the encoder counter value. The source code of this class is in the file SerialPort.cpp listed in Table G.1.

B.2.3 Class: SocketServer

The socketserver class contains the functions to read and write from the network as a UDP server. The class contains basically two functions. Send() and Receive(). The Receive function waits until a UDP packet of data is received. It then returns this data. In the main program, a thread is started which calls receive, and upon receiving a message, replies with the corresponding data. Figure B.3 shows the basic operation mode of this thread.

As can be seen in Figure B.3, five types of messages can be returned by the server. The messages are now explained here:

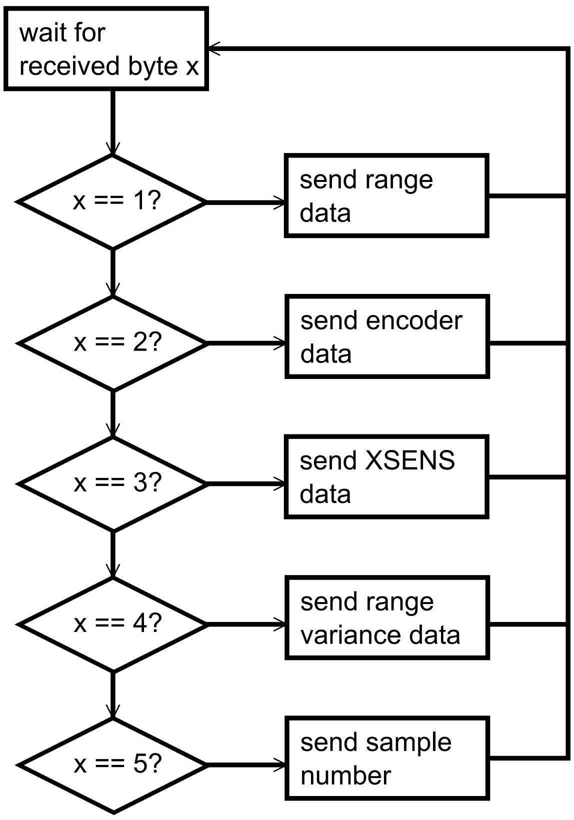


Figure B.3: Basic mode of operation of class SocketServer. A thread from the main program continuously listens for data and replies.

- message 1: Returns the 360 floating point values of the laser range finder data.
- message 2: Returns the 32-bit integer of the encoder counter value.
- message 3: Returns the 3 floating point numbers acc_x,acc_y and acc_z.
- message 4: Returns a variance calculated for each laser range data point (not used currently). 360 floating point values.
- message 5: Returns the index number of the sample currently sent. To prevent duplicate data if the client reads too fast.

The entire source code of this class can be found in the files listed in Table G.1.

B.2.4 Class: ImageProcessor

This class is a very important class, which performs the laser curve extraction of the camera images. The class makes use of the OpenCV library, which has been altered a bit, The alterations to the openCV library can be found in Appendix C. The class ImageProcessor contains one important function, called filter(), which does the image processing. The function filter captures a frame from the camera, then uses CVRemap() from OpenCV to make a polar coordinate mapping of the image. The center of the polar coordinate system is coded into the function, and resulted from calibration data. After remapping the image, a gaussian blur is performed, and a blob-detection in the vertical direction is performed, according the the procedure described in Section 4.3. After this, each column is regarded a histogram, and the weighted mean is calculated from every column of image data and is placed in array “mean”. “mean” contains 360 data points. Also, the standard deviation is calculated of every column, which is placed in the array “std”. In the constructor of the class, the polar coordinate mapping and convolution kernels are filled. The source code of this class can be found in the file listed in Table G.1.

B.2.5 Main of the CaptureCamCV

The main function of the program generates the threads which are used. It also creates objects of the four classes. One of the threads is the main function itself, which runs a while loop, continuously calling ImageProcessor::Filter(), and after that sends a byte with value 00 to the serial port of the microcontroller. This means that the LED lights are turned off. When this byte is send, the thread running the SerialPort::read() function receives the encoder value from the microcontroller. The source code of the main function can be found in the file CaptureCamCV listed in Table G.1.

B.2.5.1 Running the Program

To start the program, two commandline arguments need to be input. The first argument is the UDP port number for the server, the second argument is the number of the camera to use. So if port 5000 is to be used, and camera 2, then type: "./CapturecamCV 5000 2".n

C Alterations to the OpenCV Library

This chapter describes the alteration made to the OpenCV library.

C.1 Settings for Camera Resolution

The OpenCV source code 2.1.0 was downloaded, and it was found that no method is implemented to set the resolution of the camera. Therefore, the file “src/highgui/cvcap_libv4l.cpp” from the source code tree was modified.

```
229     #define DEFAULT_V4L_WIDTH   640
230     #define DEFAULT_V4L_HEIGHT  480
```

was altered into:

```
229     #define DEFAULT_V4L_WIDTH   1280
230     #define DEFAULT_V4L_HEIGHT  960
```

Also, in the file “/src/highgui/cvcap_v4l.cpp”,

```
231     #define DEFAULT_V4L_WIDTH   640
232     #define DEFAULT_V4L_HEIGHT  480
```

was altered into:

```
231     #define DEFAULT_V4L_WIDTH   1280
232     #define DEFAULT_V4L_HEIGHT  960
```

C.2 Settings for Manual Exposure

It was found that on this particular camera, exposure settings were not working from the openCV standard method. A property name for exposure had to be replaced by a property name for “Absolute exposure”. These settings are found in the file “/src/highgui/cvcap_v4l.cpp”. The following lines of code were altered:

```
2555 case CV_CAP_PROP_EXPOSURE:
2556     capture->control.id = V4L2_CID_EXPOSURE;
2557 break;
```

was changed in:

```
2555 case CV_CAP_PROP_EXPOSURE:
2556     capture->control.id = V4L2_CID_EXPOSURE_ABSOLUTE;
2557 break;
```

On another place:

```
2595 case CV_CAP_PROP_EXPOSURE:
2596     v4l2_min = capture->v4l2_exposure_min;
2597     v4l2_max = capture->v4l2_exposure_max;
2598 break;
```

The retreival of min/max values was hard-coded to:

```
case CV_CAP_PROP_EXPOSURE:
    v4l2_min = 0x8; //capture->v4l2_exposure_min;
    v4l2_max = 0x4000; //capture->v4l2_exposure_max;
break;
```

and finally:

```
    case CV_CAP_PROP_EXPOSURE:
        capture->control.id = V4L2_CID_EXPOSURE;
        break;
```

was changed to:

```
    case CV_CAP_PROP_EXPOSURE:
        capture->control.id = V4L2_CID_EXPOSURE_ABSOLUTE;
        break;
```

C.3 Compiling and Installing the updated Library

The updated library is compiled under ubuntu using gcc. First unpack the library, which was included in the files belonging to this project. Then to compile, go into the folder where the source was unpacked, and run:

```
make
```

This compiles the source code, and to install the new library to the operating system, run:

```
make install
```

This installs the new library in the folder "/usr/local/lib" and the file will be called "libhighgui.so.2.1.0".

D Template Matching Algorithm detailed description

This chapter describes the source code of the template matching algorithm.

D.1 Sum of Absolute Differences Scaled

The algorithm to calculate the sum of absolute differences is implemented using a mex file. This is actually a c++ code file, which can be compiled for matlab. This has the advantage that nested loops run faster than when using native matlab code.

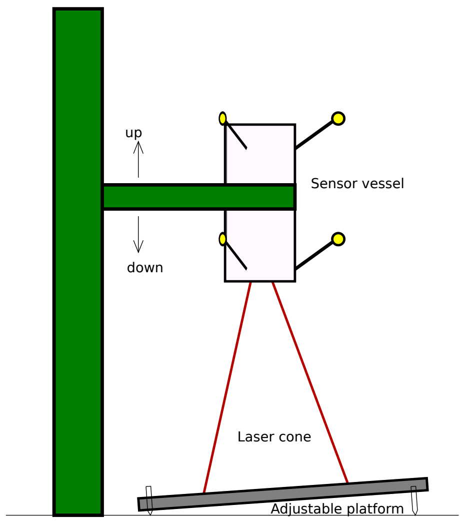
```
#include <mex.h>
#include <math.h>

double AbsDiff(double *m,double *t,double *um,int x, int y,int sx,int sy,
    int Sx);

void mexFunction(int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *
    prhs[ ])
{
    /*/function needs inputs: map[][], template[][], unmapped[][]
    //map is the source image
    //template is the template to match
    //unmapped is the location of not yet mapped points
    //sx, sy: size x and y of template
    //Sx, Sy: size x and y of map*/
    double *map;
    double *template;
    double *unmapped;
    double *output;
    double tmpsum;
    int Sy,Sx,sx,sy,i,j;
    /*prhs input pointer
    //plhs output pointer*/
    Sy = mxGetM(prhs[0]);
    Sx = mxGetN(prhs[0]);
    sy = mxGetM(prhs[1]);
    sx = mxGetN(prhs[1]);
    map = mxGetPr(prhs[0]);
    template = mxGetPr(prhs[1]);
    unmapped = mxGetPr(prhs[2]);


    if(Sy==sy)
    {
        plhs[0] = mxCreateDoubleMatrix(Sy,Sx,mxREAL);
        output = mxGetPr(plhs[0]);
        for(i=0;i<Sx;i++)
        {
            for(j=0;j<Sy;j++)
            {
                *(output +j +i*Sy) = AbsDiff(map, template ,unmapped, i , j , sx
                    , sy , Sx) ;
            }
        }
    }
    else
    {
        plhs [0] = mxCreateDoubleMatrix(1 ,1 ,mxREAL) ;
        output = mxGetPr(plhs [0]) ;
        *output = 0;

    }


}

double AbsDiff(double *m,double *t ,double *um,int x, int y,int sx ,int sy ,
    int Sx)
{
    int i , j , tj , ti ;
    double sum;
    double tmp;
    double count;
    tmp=0;
    count = 0;
    for( i=0;i<sx; i++)
    {
        for( j=0;j<sy; j++)
        {
            tj = (j+y)%sy;
            ti = (i+x)%Sx;
            sum = *(m+tj +ti*sy)- *(t+j + i*sy);
            if(*(um+tj +ti*sy)==1)
            {
                tmp += abs(sum);
                count+=abs(*(t+j + i*sy));
            }
            else
            {
                tmp+= abs(sum)/2.0;
                count+=abs(*(t+j + i*sy));
            }

        }
    }

    return tmp/count;
}
```

E Manual for Laser Range Finder Calibration

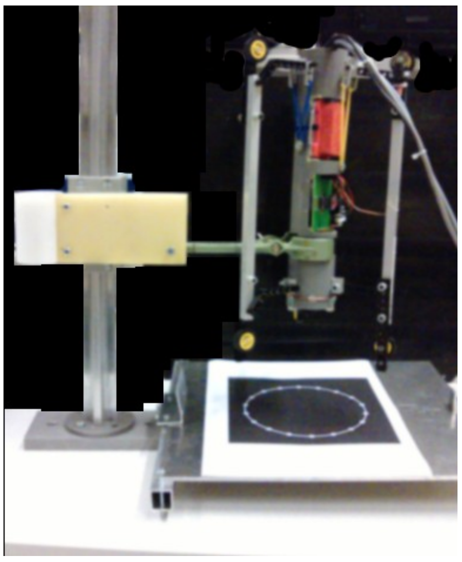
This chapter contains the manual telling how to calibrate the laser range finder.

E.1 Setting up the Calibration Rig

The first step of the calibration of the laser range finder is to mount the laser range finder in the calibration rig. Figure E.1 Shows the vessel mounted inside the rig. It is preferred that the vessel is mounted such that the center axis of the laser range finder is approximately perpendicular to the ground plane.



(a) Schematic display of the calibration hardware.



(b) Photo of the calibration hardware, the sensor vessel is held by a clamp.

Figure E.1: The calibration hardware, the height of the projector is adjustable, the ground plane's orientation is also adjustable

E.1.1 Adjusting the Projection Plane

The projection plane now needs to be adjusted such that it is exactly perpendicular to the laser projector. To test this, the cylindrical section of pipe displayed in Figure E.2 can be used. First adjust the height of the calibration rig to approximately align the laser line with the line inside the cylindrical pipe section. Then adjust the projection plane orientation with the three screws to align the laser line precisely with the black line inside the cylinder.

Note that it must be verified that the cylinder shape is circular. This can be done by using a sliding gauge, and if necessary, loosen or tighten the screws on the cylinder to make its shape more circular.

E.1.2 Capturing Calibration Images

To calibrate the optical setup, a set of calibration images is used. Print out the files listed in Table G.9, this are the calibration patterns. One of these patterns is shown in Figure E.3.

As example, the procedure will be explained for the 70 mm pattern. The same procedure must be repeated for all the patterns. First, put the pattern on the projection plane. Now the height of the projector must be adjusted by turning the knob on the calibration rig, such that the radius

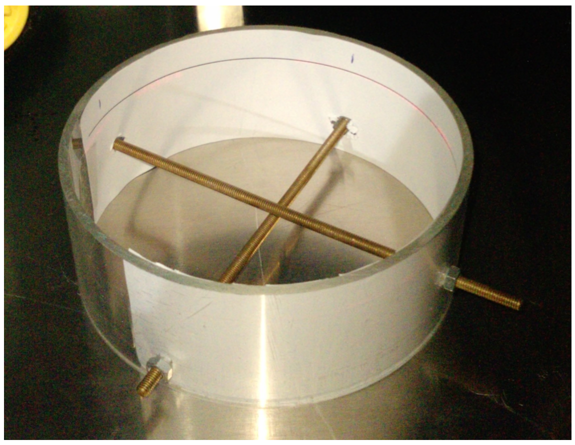


Figure E.2: Cylinder shape used to assist in aligning the Projection plane with the projector.

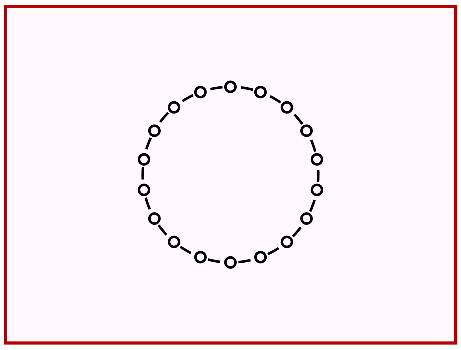


Figure E.3: The calibration pattern, for the 70 mm radius circle.

of the laser circle is the same as the circle on the calibration pattern. Then the pattern must be rotated so that one of the 18 small circles is at the reference point, as shown in Figure E.4.

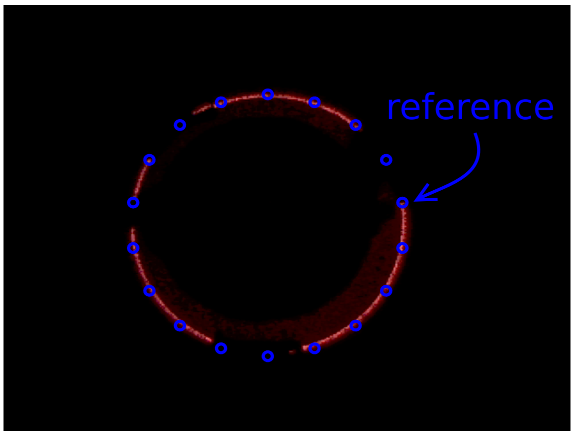


Figure E.4: Laser cone projected on a flat surface perpendicular to cone, with reference marked, and locations of the 18 points along the circle also marked.

If the calibration pattern is aligned correctly, the laser can be turned off, and the image from the camera must be captured. Any capture program can be used, in this project, the linux program “guvcview” was used. The resolution of the camera must be set to 1280x960, and the image must be saved in the folder “FilesUsed/Matlab/Calibration”. The captured image should look like the image in Figure E.5. (The metal nuts were used as a paper weight to fix the paper position). The captured file should be saved as “bmp”, and should be named 70mmCAL.bmp. (Replace 70mm by the appropriate diameter).

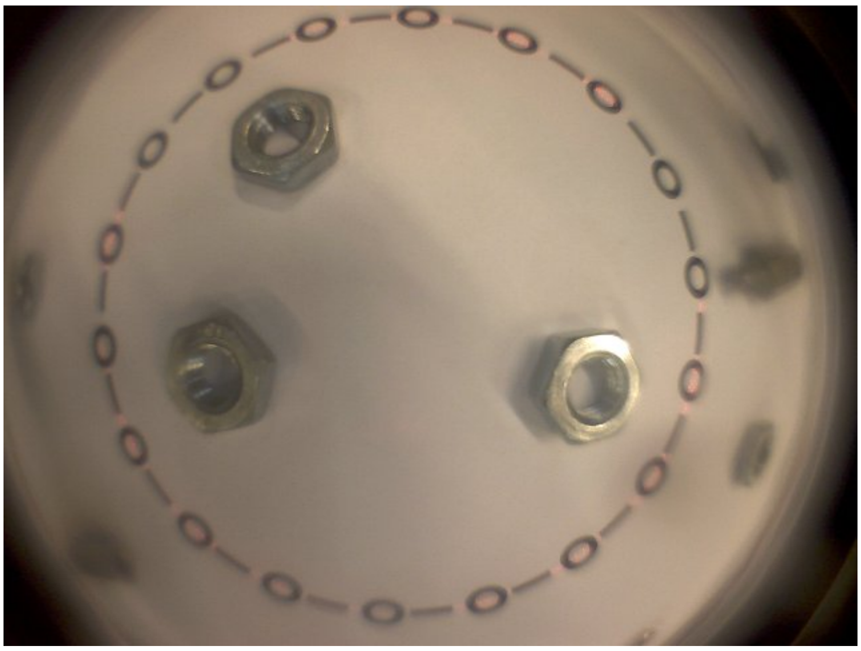


Figure E.5: Example of 70mm calibration pattern captured

E.2 Importing the Calibration Data

After all the calibration patterns are captured, now the images must be imported into the calibration data files. The Matlab script “Graphinput.m” from Table G.3 can be used for this. First open the script in the editor. This shows the following code:

```
clear all;
im = imread('70mmCal.bmp');
[xc,yc] = meshgrid(1:1280,1:960);

imshow(im);
for(i=1:18)

[x,y] = ginput(2);
x = round(x);
y = round(y);

W = 1-sum(im2double(im(y(1):y(2),x(1):x(2),2:3)),3);
W = W-min(W(:));
W = W/max(W(:));
W(find(W<0.6))=0;
W = W./sum(W(:));

mx(i) = sum(sum(W.*xc(y(1):y(2),x(1):x(2),:)));
my(i) = sum(sum(W.*yc(y(1):y(2),x(1):x(2),:)));
hold on;
plot(mx,my,'*');
hold off;
end;
```

```
C = [mx;my;190*ones(1,18)];
%Array C contains the Coordinates. This should be
%saved to a .mat file with correct name.
C = [mx;my;120*ones(1,18)];
%Naming convention for saving variable
% C: 135mm.mat, 090mm.mat or XXXmm.mat
```

The second line load the image. Replace the image name with the desired file to import data from. The script will show the image, and the mouse must be used to draw a rectangle around every calibration point (the 18 small circles in the pattern), in a counter clockwise fashion, starting at the circle which was marked with an arrow in Figure E.4. It can be beneficial to mark the first circle with a arrow on the calibration pattern, so that it can be recognized. In this example, a metal paperweight was placed near the first circle. After each rectangle drawn, the script searches the center position of the circle inside the rectangle, and marks this with an asterisk. Figure E.6 shows what this looks like.

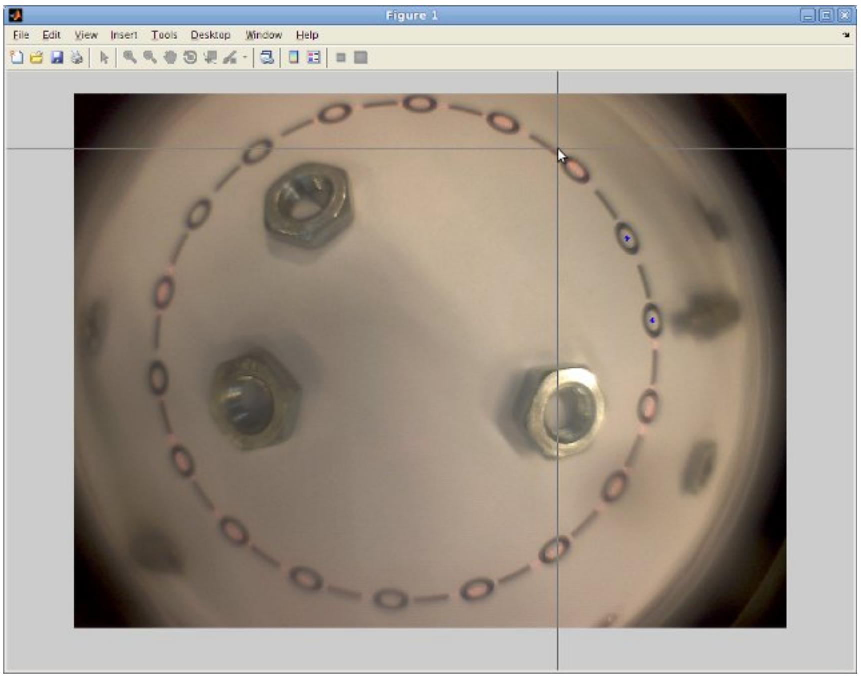


Figure E.6: The process of importing captured calibration files

To draw a rectangle around a calibration point, first click on a upper left above the circle, then lower right below the circle. The rectangle needs not to be very tight around the circle, but it is important that the white space inside the circle is approximately in the center of the rectangle. Sometimes the asterisk is estimated on the wrong location. If this happens, then the script must be re-run and it must be attempted to make a tighter fitting rectangle around the circle.

E.2.1 Saving the Calibration Data

After the script “GraphInput.m” has been completed, and all 18 points are marked, the workspace now contains a matrix called “C”. This matrix must be saved as a .mat file, called “???mm.mat”. (in this case, 070mm.mat). After this is done for all calibration patterns, then the calibration is done. World coordinates can now be matched to camera coordinates.

E.3 Using the Calibration Data

To use the calibration data, first it must be known to the system which patterns were used in the calibration. In this project, circles with radii ranging from 70...135 mm were used with a step of 5 mm, plus two extra radii, 150 mm and 190 mm. Using more circles will provide more data for the interpolation to, which could make the system more accurate. All the ???mm.mat files are combined by the script called “Create_coords.m”. This scripts needs to be altered to read in every pattern which was captured. The script source looks like this:

```
%This script takes a set of .mat files and creates all world and camera
%coordinates out of it.

clear all;
th = (0:20:340)*pi/180;
u =[];
v=[];
z =[];
x=[];
y=[];
s = ['070mm.mat';
     '075mm.mat';
     '080mm.mat';
     '085mm.mat';
     '090mm.mat';
     '095mm.mat';
     '100mm.mat';
     '105mm.mat';
     '110mm.mat';
     '115mm.mat';
     '120mm.mat';
     '125mm.mat';
     '130mm.mat';
     '135mm.mat';
     '150mm.mat';
     '190mm.mat'];
diameters = [70;75;80;85;90;95;100;105;110;115;120;125;130;135;150;190];

for(i=1:size(s,1))

    C = load(s(i,:));
    %obtain the 18 camera coordinates
    u=[u C.C(1,:)];
    v=[v C.C(2,:)];
    r = diameters(i)/2;
    %create world coordinates belonging to the cone
    z = [z r/30*100*ones(1,18)];
    x = [x r*cos(th)];
    y = [y r*sin(th)];
end;

%make two arrays ,C is camera coordinates, W = world coordinates
C = [u;v];
```

```
W = [x;y;z];
```

The matrix “S” must be altered to include all files previously captured, and the diameter belonging to each file must be added to the array “diameters”. This script is used by various other functions, such as the function “makeMapping()”. The script creates two arrays, “C" and “W”, “C” contains camera coordinates and “W” contains world coordinates belonging to that. (world coordinates are actually in cone coordinate system, described in Subsection 4.2.1.2).

E.3.1 Using the Mapping Interpolation

With the set of camera coordinates and corresponding world coordinates, a interpolation can be made to find world coordinates from arbitrary camera coordinates. This is used when laser range finder data is input from the data acquisition server. The data acquisition client (“getNetwork.m”) will use a interpolation method for this. The interpolation method actually is the standard Matlab command “TriScatteredInterp”, and is implemented in the function “makeMapping.m” (see Table G.3). The function is called as follows:

```
[FR, FPhi] = makeMapping()
```

This yields two functions, “FR” and “FPhi”. The laser range finder data from the data acquisition client comes as a radius r and a angle φ. If now the radius of the circle in world coordinates is needed, use:

```
Rworld = FR(r,phi);
```

If the angle in world coordinates is needed, use:

```
Rworld = FR(r,phi);
```

(the radius and angle in world coordinates are the radius and angle from the mapping shown in Figure fig:ConeMap).

This concludes the manual of the calibration procedure.

F Hardware Components used in the Sensor Vessel

This chapter lists the hardware components used in the sensor vessel.

F.1 Ordered Parts

Several parts of the Sensor vessel had to be bought.

supplier	article number	description
Conrad Electronics	971975 - 89	USB webcam
Holo-Eye	DE-R 219	Diffractive Optical Element
Maxon Motor	110521	Encoder 100 pulses
Gamma	?	Doorspy, fish-eye lens

Table F.1: Several parts ordered for this project

The price of the ordered parts totals to an amount of about 150 Euro. Figure F.1 shows a picture of the ordered parts.

(a) "Doorspy", used as a source for wide-angle lens

(b) Diffractive Optical Element

(c) Camera

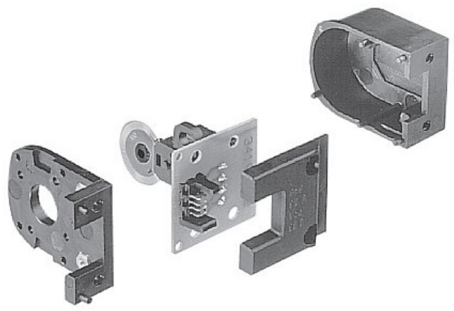

(d) Encoder

Figure F.1: Parts used for the optical sensor

F.2 Already available Parts

Several parts were used which were already available at the control laboratory. The most notable parts are listed in Table F.2

Supplier	Article Number	Description
Melles Griot	–	Laser diode 635 nm
Control Lab.	–	Motor controller (Ansink (2007))
Xsens	Xsens MTi	Inertial Measurement Unit

Table F.2: Parts used which were already available

The laser diode was taken from the laser range finder made by Drost (2009). The motorcontroller is the breakout-version of the pirate motorcontrollers.

G Files and their description

G.1 Files belonging to the data acquisition software

The files belonging to the data acquisition server are listed in Table G.1. They are found in the folder "/filesused/CaptureCamCV"

file	description
SerialPort.cpp	Source code of the SerialPort class
SerialPort.h	Header file of the SerialPort class
CaptureCamCV.cpp	source file containing the main function
ContinuousSerialPort.cpp	Source code of the ContinuousSerialPort class
ContinuousSerialPort.h	header file of ContinuousSerialPort class
ImageProcessor.cpp	source file of the ImageProcessor class
ImageProcessor.h	header file of the ImageProcessor class
SocketServer.cpp	Source file of the SocketServer class
SocketServer.h	header file of the SocketServer class

Table G.1: Description of source code files belonging to the Data acquisition program

The matlab files belonging to the data acquisition client are in the same folder are the calibration matlab files. This is done because the data acquisition client depends on some of these files. The folder is called "/filesused/Matlab/Calibration"

file	description
getNetwork.m	Daq Client, gathers data from the UDP server.
CreateCoords.m	loads the pre-recorded calibration data.
MakeMapping.m	creates mapping functions between camera and world coordinates.

Table G.2: Description of matlab source code files belonging to the Data acquisition program

The data acquisition client is set to use UDP port 5000. It can be run from matlab after the server has been started. The main script is getNetwork.m, this is the client.

G.2 Files belonging to the Calibration Procedure in Section 4.4

This section describes the Matlab files which were used during calibration of the laser range finder. The files are found in the folder "FilesUsed/Matlab/Calibration". The files with their description are listed in Table G.3

G.3 Files Belonging to the Simulations

This section lists the files that belong to the simulations from Chapter 6. The main folder where the files for the simulations reside is: "/FilesUsed/Matlab/Simulations".

G.3.1 Files Belonging to Section 6.1

In Section 6.1, the performance of the particle filter is tested. This is done using the matlab files found in the folder "/FilesUsed/Matlab/Simulations/6.1". The files are listed in Table G.4. The simulation is started by running SIR_gaussian.m

file	description
Create_coords.m	Reads in data files with calibration data.
makeInvMapping.m	Makes the inverse mapping. (world -> camera)
makeMapping.m	makes mapping functions, Camera->world
Graphinput.m	inputs a captured calibration image. select coords with mouse.
getNetwork.m	Data acquisition client
makeDiffKernel.m	generate 2nd differential of gaussian kernel
transformImage.m	Verification of the calibration procedure.
FlatPlateR104.bmp	Image inside calibration rig, used by transformImage.m
xxxmmCAL.mat	Data files containing calibration data, for each radius xxx.

Table G.3: Description of source code files belonging to Section 4.4

file	description
RSR.m	Residual Systematic Resampling algorithm
drawfromproposalPdf.m	function to draw samples from proposal density.
FitPipe.m	Ray cast algorithm observation model for straight pipes.
getRot.m	Returns a rotation matrix from a state or world vector input.
plotLaser.m	Plots the laser dots in the world coordinates.
PlotT1.m	Draws a straight section of pipe with a T-junction type 1.
testHypothesis.m	Calculate probability density $p(r_k\|x_k)$ using fitPipe
SIR_gaussian.m	Main program to test the particle filter.
SampleData.mat	data file containing struct array of measurements.
err*.mat	Various data files containing error performances from tests.

Table G.4: Description of source code files belonging to Section 6.1

G.3.2 Files Belonging to Section 6.2.1

In Section 6.2.1, a simulated map is generated of a pipe with a 90-degree bend. The Matlab files belonging to this simulation are listed in Table G.5. The files are found in the folder "FilesUsed/Matlab/Simulations/6.2.1"

file	description
sumAbsDifferens.c	Source code from Section D.1.
findPeak.m	function which finds the peak of the template match.
fitBend.m	ray casting algorithm for a bend.
fitPipe.m	ray casting algorithm for a straight pipe.
getRot.m	Returns a rotation matrix from a state or world vector input.
makeOccgrid.m	Main simulation file. Simulates the map with a bend.
sumAbsDifferens.mexglx	binary mex file of sumAbsDifferens.c
SampleData.mat	data file containing struct array of measurements.
Bend.mat	data file containing Bend template.
T1.mat	data file containing T-junction type 1 template.
T2.mat	data file containing T-junction type 2 template.
MAP.mat	Example data file containing a simulated map.

Table G.5: Description of source code files belonging to Section 6.2.1

G.3.3 Files Belonging to Section 6.2.2

In Section 6.2.2, a simulated map is generated of a pipe with a T-junction type 1. The Matlab files belonging to this simulation are listed in Table G.6. The files are found in the folder "FilesUsed/Matlab/Simulations/6.2.2"

file	description
sumAbsDifferens.c	Source code from Section D.1.
findPeak.m	function which finds the peak of the template match.
fitBend.m	ray casting algorithm for a bend.
fitPipe.m	ray casting algorithm for a straight pipe.
fitT1.m	ray casting algorithm for a T-junction.
getRot.m	Returns a rotation matrix from a state or world vector input.
makeOccgrid.m	Main simulation file. Simulates the map with a bend.
sumAbsDifferens.mexglx	binary mex file of sumAbsDifferens.c
SampleData.mat	data file containing struct array of measurements.
Bend.mat	data file containing Bend template.
T1.mat	data file containing T-junction type 1 template.
T2.mat	data file containing T-junction type 2 template.
MAP.mat	Example data file containing a simulated map.

Table G.6: Description of source code files belonging to Section 6.2.2

G.3.4 Files Belonging to Section 6.2.3

In Section 6.2.2, a simulated map is generated of a pipe with a T-junction type 2. The Matlab files belonging to this simulation are listed in Table G.7. The files are found in the folder "FilesUsed/Matlab/Simulations/6.2.3"

file	description
sumAbsDifferens.c	Source code from Section D.1.
findPeak.m	function which finds the peak of the template match.
fitBend.m	ray casting algorithm for a bend.
fitPipe.m	ray casting algorithm for a straight pipe.
fitT1.m	ray casting algorithm for a T-junction.
getRot.m	Returns a rotation matrix from a state or world vector input.
makeOccgrid.m	Main simulation file. Simulates the map with a bend.
sumAbsDifferens.mexglx	binary mex file of sumAbsDifferens.c
SampleData.mat	data file containing struct array of measurements.
Bend.mat	data file containing Bend template.
T1.mat	data file containing T-junction type 1 template.
T2.mat	data file containing T-junction type 2 template.
MAP.mat	Example data file containing a simulated map.

Table G.7: Description of source code files belonging to Section 6.2.3

G.4 Files belonging to the Experiments

This section describes the Matlab files used to run the experiments. All experiments are performed from one script, but different measurement data is loaded at the beginning of the script. (uncomment the desired data). Table G.8 shows the files used in Section 7.1.

file	description
SIR_gaussian.m	Main program to run the experiment.
RSR.m	Residual Systematic Resampling algorithm
SampleData.mat	data file containing struct array of measurements.
sumAbsDifferens.c	Source code from Section D.1.
sumAbsDifferens.mexglx	binary mex file of sumAbsDifferens.c
findPeak.m	function which finds the peak of the template match.
fitBend.m	ray casting algorithm for a bend.
fitPipe.m	ray casting algorithm for a straight pipe.
fitT1.m	ray casting algorithm for a T-junction.
getRot.m	Returns a rotation matrix from a state or world vector input.
makeOccgrid.m	Main simulation file. Simulates the map with a bend.
Bend.mat	data file containing Bend template.
T1.mat	data file containing T-junction type 1 template.
T2.mat	data file containing T-junction type 2 template.
plotLaser.m	Plots the laser dots in the world coordinates.
PlotT1.m	Draws a straight section of pipe with a T-junction type 1.
PlotBend.m	Draws a 90-degree Bend
testHypothesis.m	Calculate probability density $p(r_k|x_k)$ using fitPipe
SampleBend.mat	Measurement data for pipe with bend
SampleT1.mat	Measurement data for a pipe with T-junction type 1
SampleT2.mat	Measurement data for a pipe with T-junction type 2
drawfromproposalPdf.m	function to draw samples from proposal density.
matchFeatures.m	Function to find the match value of templates on map
findPeak.m	function to find the peak match of a template
updateOccGrid.m	Function that updates cells in the occupancy grid.

Table G.8: Description of source code files belonging to Section 7.1

G.5 Calibration patterns

A number of calibration patterns used for calibration of the laser range finder is saved into the folder “FilesUsed/CalibrationPatterns”.

The naming convention is “*mmPattern.svg”, where * is replaced by the diameter of the circle.

file	description
70mmPattern.svg	Inkscape file with the 70 mm diameter calibration pattern
75mmPattern.svg	Inkscape file with the 75 mm diameter calibration pattern
80mmPattern.svg	Inkscape file with the 80 mm diameter calibration pattern
85mmPattern.svg	Inkscape file with the 85 mm diameter calibration pattern
90mmPattern.svg	Inkscape file with the 90 mm diameter calibration pattern
95mmPattern.svg	Inkscape file with the 95 mm diameter calibration pattern
100mmPattern.svg	Inkscape file with the 100 mm diameter calibration pattern
105mmPattern.svg	Inkscape file with the 105 mm diameter calibration pattern
110mmPattern.svg	Inkscape file with the 110 mm diameter calibration pattern
115mmPattern.svg	Inkscape file with the 115 mm diameter calibration pattern
120mmPattern.svg	Inkscape file with the 120 mm diameter calibration pattern
125mmPattern.svg	Inkscape file with the 125 mm diameter calibration pattern
130mmPattern.svg	Inkscape file with the 130 mm diameter calibration pattern
135mmPattern.svg	Inkscape file with the 135 mm diameter calibration pattern
150mmPattern.svg	Inkscape file with the 150 mm diameter calibration pattern
190mmPattern.svg	Inkscape file with the 190 mm diameter calibration pattern

Table G.9: Description of the files with the calibration patterns for the laser range finder.

Bibliography

Ansink, J. (2007), *Electronic design of a gas pipe inspection robot*, Master's thesis, University of Twente.

de Boer, H. (2008), *Analysis, optimization and evaluation of a pipe inspection robot*, Master's thesis, University of Twente.

Bolić, M., P. M. Djurić and S. Hong (2004), Resampling Algorithms for Particle Filters: A Computational Complexity Perspective, *URASIP Journal on Applied Signal Processing*, **2004**, 15, pp. 2267–2277.

Burkink, B. (2009), PIRATE Propulsion Unit, Technical report, University of Twente.

Dertien, E. (2006), *System specifications for PIRATE*, Master's thesis, University of Twente.

Dertien, E. (2007), Design of PIRATEMainboard, Technical report, University of Twente.

Doggen, C. (2010), *Wireless state feedback and control of a pipe inspection robot*, Master's thesis, University of Twente.

Douc, R. (2005), Comparison of resampling schemes for particle filtering, in *In 4th International Symposium on Image and Signal Processing and Analysis (ISPA*, pp. 64–69.

Doucet, A., S. Godsill and C. Andrieu (2000), On sequential Monte Carlo sampling methods for Bayesian filtering, *Statistics and Computing*, **10**, 3, pp. 174–188.

Drost, E. (2009), Measurement system for pipe profiling, *MSc-Report 003CE2009*.

Duran, O., K. Althoefer and L. D. Seneviratner (2003), Automated Sewer Inspection Using Image Processing and a Neural Classifier, *Transactions on Mechatronics IEEE/ASME*.

Elinas, P., R. Sim and J. Little (2006), σSLAM: stereo vision SLAM using the Rao-Blackwellised particle filter and a novel mixture proposal distribution, *ICRA 2006. Proceedings 2006 IEEE International Conference on Robotics and Automation*.

Fairfield, N., G. A. Kantor and D. Wettergreen (2006), Towards Particle Filter SLAM with Three Dimensional Evidence Grids in a Flooded Subterranean Environment, in *Proceedings of ICRA 2006*, pp. 3575 – 3580.

Grisetti, G., C. Stachniss and W. Burgard (2007a), Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, *IEEE Transactions on Robotics*, **23**, 1, pp. 34–46.

Grisetti, G., G. D. Tipaldi, C. Stachnissr, W. Burgard and D. Nardi (2007b), Fast and Accurate SLAM with Rao-Blackwellized Particle Filters, *Robot. Auton. Syst.*, **55**, pp. 30–38, ISSN 0921-8890.

Holz, D., C. Lorken and H. Surmann (2008), Continuous 3D sensing for navigation and SLAM in cluttered and dynamic environments, *11th International Conference on Information Fusion*, **June 30 2008-July 3 2008**, pp. 1 – 7.

Laet, T. D., J. D. Schutter and H. Bruyninckx (2008), Rigorously Bayesian range finder sensor model for dynamic environments, in *ICRA*, pp. 2994–3001.

Lu, F. and E. Milios (1997), Globally Consistent Range Scan Alignment for Environment Mapping, *Autonomous Robots*, **4**.

Newman, P. M. (1999), *On the Structure and Solution of the Simultaneous Localisation and Map Building Problem*, Ph.D. thesis, Australian Centre for Field Robotics,The University of Sydney.

Pandey, A. K., K. M. Krishna and H. Hexmoor (2007), Feature Chain based Occupancy Grid SLAM for Robots Equipped with Sonar Sensors, in *IEEE-International Conference on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS), 2007*.

Perwass, C. and G. Sommer (2006), The Inversion Camera Model, Technical report, Institut fü r Informatik, CAU Kiel.

Pfaff, P., C. Plagemann and W. Burgard (2007), Improved Likelihood Models for Probabilistic Localization based on Range Scans, in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA.

Plagemann, C., K. Kersting, P. Pfaff and W. Burgard (2007), Gaussian Beam Processes: A Nonparametric Bayesian Measurement Model for Range Finders, in *Robotics: Science and Systems (RSS)*, Atlanta, Georgia, USA.

Reemeijer, H. (2010), *Control of a pipe inspection robot*, Master's thesis, University of Twente.

Reilink, R. (2008), *Realtime Stereo Vision Processing for a Humanoid*, Master's thesis, Control Laboratory, Unversity of Twente.

Ristic, B., S. Arulampalam and N. Gordon (2004), *Beyond the Kalman Filter*, Artech House Publishers, ISBN 158053631.

Rodriguez-Losada, D., F. Matia, A. Jimenez and R. Galan (2006), Consistency improvement for SLAM - EKF for indoor environments, in *IEEE International Conference on Robotics and Automation, 2006*, pp. 418 – 423.

Schroeter, C. and H.-M. Gross (2008), A sensor-independent approach to RBPF SLAM - Map Match SLAM applied to Visual Mapping, in *In IROS 2008. IEEE/RSJ International Conference on intelligent Robots and Systems, 2008.*, pp. 2078 – 2083.

Vennegoor op Nijhuis, J. (2007), *Development of a pipe inspection robot*, Master's thesis, University of Twente.

Wu1, C.-J. and W.-H. Tsai1 (2009), Adaptation of Space-Mapping Methods for Object Location Estimation to Camera Setup Changes: A New Study, Technical report, Institute of Computer Science and Engineering, National Chiao Tung University, Taiwan.

Zhang, Z. (1999), Flexible Camera Calibration By Viewing a Plane From Unknown Orientations, Technical report, Microsoft Research.