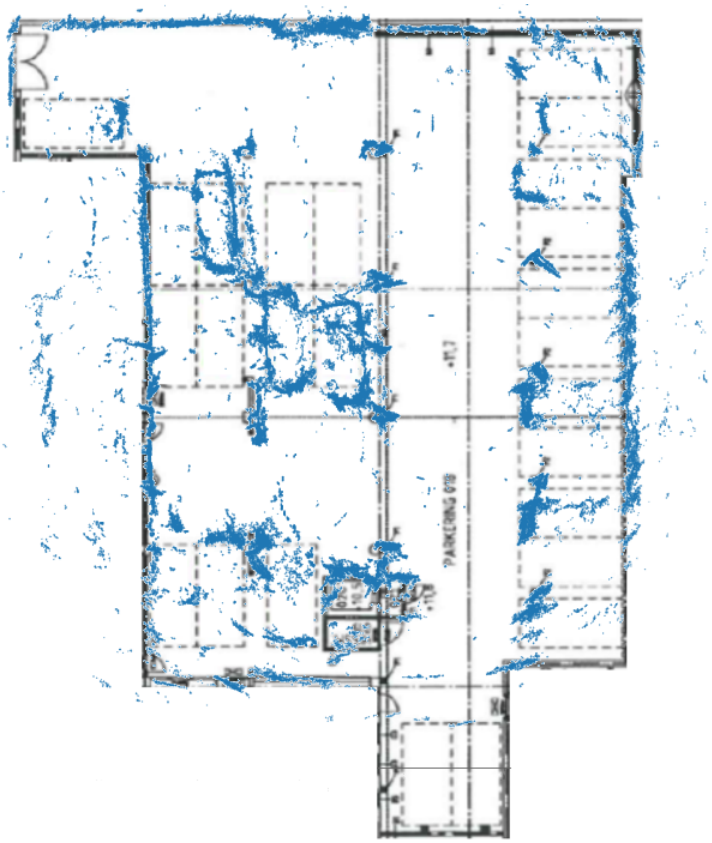
CHALMERS
UNIVERSITY OF TECHNOLOGY



Indoor Radar Localisation

Detection Density Based SLAM

MARCUS HOLMSTRÖM
MARTIN LIDÉN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's thesis 2021

Indoor Radar Localisation

Detection Density Based SLAM

MARCUS HOLMSTRÖM
MARTIN LIDÉN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
Gothenburg, Sweden 2021

Indoor Radar Localisation
Detection Density Based SLAM
Marcus Holmström
Martin Lidén

Supervisor: Rickard Nilsson, Aptiv
Supervisor: Björn Langborn, Department of Electrical Engineering
Examiner: Tommy Svensson, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Floor plan of the Aptiv garage with an overlay of detections from one lap using the Detection Density Augmentation Filter (DDAF).

Typeset in LaTeX
Gothenburg, Sweden 2021

Indoor Radar Localisation
Detection Density Based SLAM
Marcus Holmström & Martin Lidén
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In this thesis indoor localisation using Simultaneous Localization And Mapping (SLAM) based on radar data has been investigated. As radar already is available in many vehicles in the automotive industry, improving the radar functionality to work better indoors would increase the confidence in the vehicle's overall positioning. The data used has been recorded at Aptiv's facilities and the data set is from a basement parking garage where the material in the area mainly are concrete and sheet metal, with the addition of a few corner reflectors.

An analysis of the Radar Cross-Section (RCS) from these materials is presented as well as an algorithm for filtering, augmenting and processing the radar detections. This algorithm, called Detection Density Augmentation Filter (DDAF), was designed for compatibility with an open source SLAM module and is compared to a preexisting processing standard.

A SLAM module has been implemented, into the Aptiv project City Mobility On Demand (CMOD), and tested. The SLAM results are subpar as the module was found to not fully account for higher velocities of the host vehicle. The faulty output could be reproduced and the issue traced down to the open source module not interpolating correctly between vehicle positions. Despite the subpar results the algorithm DDAF mentioned above shows promising results for indoor SLAM using radar.

Keywords: Radar, Indoor, Localisation, GraphSLAM, ROS, Aptiv

Acknowledgements

First and foremost we would like to thank Rickard Nilsson, our supervisor at Aptiv, for setting up this project and giving us the opportunity to work with such an interesting thesis. His patience, enthusiasm and constant support help us through our ups and down. We wish him and his family with the newly born the very best.

A big thank you to William Ljungbergh and Gerog Hess, who also worked on the tuk-tuk but with visual SLAM, for their help with all from installation guides to implementation of ROS modules and their knowledge of the tuk-tuk. A quick video chat always saved us many hours of potential headache.

Then we would like to thank Björn Langborn, our supervisor at Chalmers, for his cheery mood and constant support. His insightful thoughts and keen eye truly helped us finishing this report.

Lastly, we would like to extend our gratitude to Tommy Svensson, our examiner, for his putting his time and effort into this project and seeing it through to the end.

Marcus Holmström & Martin Lidén, Gothenburg, June 2021

Contents

List of Figures

List of Tables

Glossary

ACC Adaptive Cruise Control. 7
AD Autonomous Driving. 1
AEB Autonomous Emergency Braking-system. 7
AV Autonomous Vehicle. 1, 2

CAN bus Controller Area Network bus. 19, 21
CMOD City Mobility On Demand. 2, 4, 23, 30

DDAF Detection Density Augmentation Filter. 2, 27, 31, 34, 40, 45, 47–50, 55

EKF Extended Kalman Filter. 13, 46, 47

GPS Global Positioning System. 1, 26

IMU Inertial Measurement Unit. 3, 11, 14, 19, 21, 22

LiDAR Light Detection And Ranging. 2, 11, 26
LNIB Least Number of detection In Bin. 29, 45, 49

MRPT Mobile Robot Programming Toolkit. 23, 29–31, 40, 43, 44, 46, 47, 50–52, 55

R&D Research and Development. 2, 23
radar Radio Detection And Ranging. 1–3, 7–10, 21, 27, 35
RCS Radar Cross-Section. 8, 9, 27, 28, 35, 46, 47, 55
ROS Robot Operating System. 2–4, 21–26, 29, 30, 47, 50, 52
RViz ROS Visualization. 26

SLAM Simultaneous Localization And Mapping. 1–3, 14, 21, 23, 25, 29–31, 35, 40, 43–52, 55
SNR Signal to Noise Ratio. 3, 7, 8

VCS Vehicle Coordinate System. xii, 19, 21, 24–27, 30, 50

1
Introduction

Within the automotive industry Autonomous Driving (AD) has been a milestone since the World's fair in New York, 1939. The exhibit *Futurama* was the most popular and brought the futuristic concept of AD to a broader demographic [2]. The model at the World's Fair was the image of a utopia held by the future, thought to be a reachable goal within 20 years. In hindsight they were off by quite a few years. Not until 1977 was any major advances made, when Tsukuba Mechanical Engineering Lab displayed the first Autonomous Vehicle (AV), the car was able to follow specific white street markers and could do this while driving at a speed of 20 mph [3]. This is far from what we expect today but a small step on the way toward fully AV.

The levels of autonomy is today divided into six different levels, where human interaction and the vehicles capabilities varies for each level. Level 0 means fully controlled by a human driver, no autonomous driving, and level 5 is fully autonomous, no human interaction [4]. The top end cars from the major manufacturers today are usually level 2, but level 3 cars are approaching the market [5]. Cars with level 3 will give the user an "eyes off" experience where sensors on the vehicle will interpret and react to the surrounding, under certain circumstances. The idea is to minimize human decision-making as a means of reducing the number of accidents. As stated in [6, 7] many accidents occur due to drivers being distracted, which is something that can be greatly reduced by AD.

This thesis proposal was made by the automotive company Aptiv PLC who are actively working on the current and next generations AD systems. The proposal was to investigate how to solve the problem of indoor localization based on Radio Detection And Ranging (radar) detections and whether the implementation of Simultaneous Localization And Mapping (SLAM) is feasible. A full indoor SLAM solution is a challenging task but would create a solid ground for further development of AD indoors and also answer questions about what applications radar technology can have in AV.

This project is mainly set in an indoor environment, a confined space with poor Global Positioning System (GPS) signal. One of the most prominent examples for this use case would be a common parking garage, where there are different kinds of materials and objects to detect. The natural challenge for this environment comes from the fact that radar tends to reflect multiple times back and forth from the vehicle and the objects we are trying to detect. This phenomena is referred to as the radar multipath wave propagation problem and creates a lot of false detection's indoors.

Another application for radar indoor localisation is as first responders, using a

robot as eyes and ears, in hazardous situations [8]. The possibility to first deploy a robot to scout the area and sending a map in real time to the first responders could potentially save lives. The reason radar technology is used stems from the fact that other detection technologies, like Light Detection And Ranging (LiDAR) and vision systems, work poorly under conditions where regular vision is impaired. With inspiration from the first responders one can see how radar localization can be crucial for AV navigating in hazardous situations such as a burning garage or on roads close to forest fires.

1.1 Purpose

The purpose of this thesis is to investigate the possibilities of indoor radar localisation and whether or not the environmental constraints makes a radar SLAM solution feasible. This is of interest for a number of reasons:

- As radar units are relatively cheap compared to LiDAR this would be beneficial for car manufacturers.
- As radar units are part of the industry standard in other applications the hardware is already installed and the implementation could possibly work on already existing models.
- Using radar adds an extra layer of localisation, with the benefit of working under conditions with poor visibility.

1.2 Our contribution

This thesis presents an implementation and evaluation of a SLAM module adapted for Aptiv's hardware and software stack running in an indoor environment. The project is done in collaboration with Aptiv's active safety department and the City Mobility On Demand (CMOD) project. Our work has been the first step in Aptiv's future Research and Development (R&D) on indoor radar SLAM where the main contribution from us has been the implementation of a well tested Robot Operating System (ROS) SLAM module for their CMOD Tuk-Tuk project and a novel radar filtering and augmentation technique we call Detection Density Augmentation Filter (DDAF).

2
Scope and setup

2.1 Scope

This project aims to determine whether or not radar is suitable for localisation in an indoor area. The definition of "indoor" will need two clarifications. For the purpose of this report it means a confined area with structures interfering with the GPS signal and the confined area has walls on all sides closer to the vehicle than the max range of the radar units. This will be a noisy environment as the radar units will find plenty of multipath wave propagation and the Signal to Noise Ratio (SNR) for the detections is worse than in an open environment, with less multipath wave propagation [9]. In addition to that, the investigated area will be considered static without any moving objects. However, displacement of objects between the different data collections are considered within the scope.

2.2 Delimitations

The following limitations are set on the scope of this project:

- The scope will only be considered in the context of the hardware, software and the two data collection sites provided by Aptiv.
- The SLAM modules and algorithms used will only be available, and tested, open source modules adapted for ROS.
- The amount of computing power to solve the SLAM problem will not be considered as a metric.

2.3 Vehicle setup

The tuk-tuk provided by Aptiv comes with all necessary equipment for this project. For ease of use and economical reason no changes were made to the installed hardware. Therefore the study was restricted to already installed radar units even if the resolution was deemed too low for the intended target of localising the tuk-tuk accurately enough to not hit any obstacles. This decision was made as data processing and augmentation were considered part of the project. Thus radar data and data from the Inertial Measurement Unit (IMU) were the only inputs to the SLAM-model.

Figure 2.1: The tuk-tuk on the sidewalk outside Aptiv's facilities.

2.4 Hardware and Software setup

The software stack provided by Aptiv was a combination between ROS software developed in their Active Safety stack and the CMOD project. ROS is an open source project that started development in 2007 [10] and have become widely used for its high functionality and ability to create complex robot-like prototypes. The technology stack can be summarised with the illustration in Figure 2.2 and consist of a bottom hardware layer together with a top software layer. In the top software layer is the Linux operating system Ubuntu on which the ROS environment was installed. ROS operates in a network like fashion where different nodes run different applications or part-applications. In addition to the ROS nodes a core node is also required for the nodes to be able to talk with each other. The bottom hardware layer consists of a network router, two computers and the sensor and control related automotive hardware. In complement to that git was used for version control and Aptiv's cloud was used for backing up the collected data.

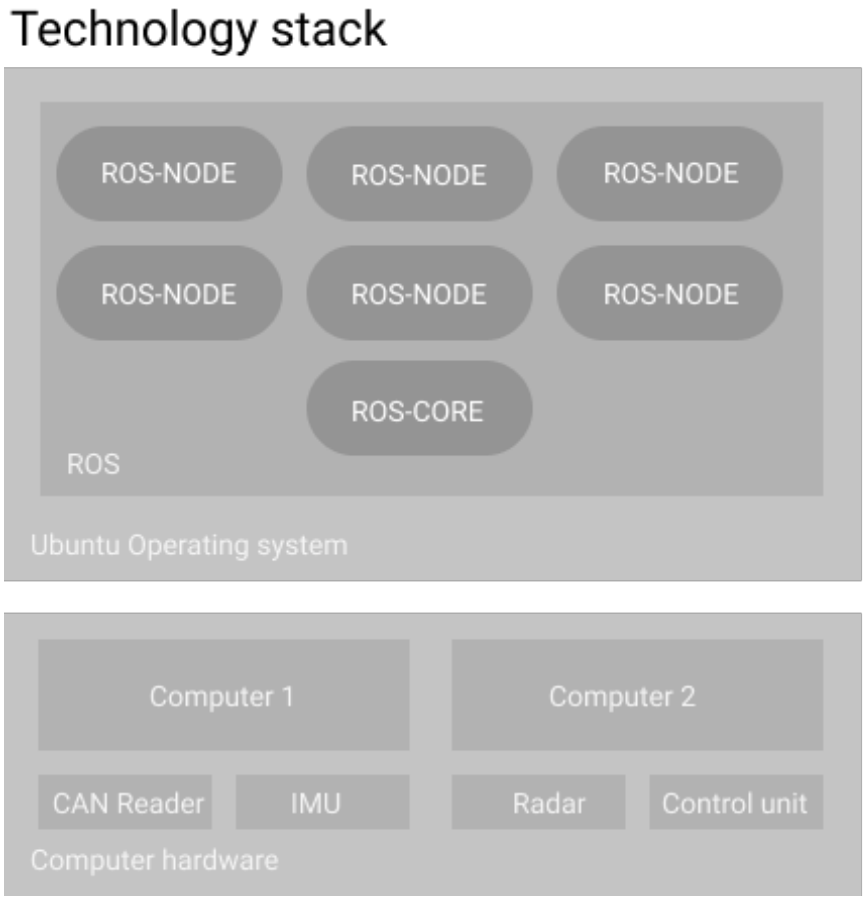


Figure 2.2: A generic representation of the technology stack used during this thesis.

3
Theory

3.1 Radar

Radar is a sensor used to measure the distance and angle of objects relative to the radar by means of electromagnetic reflection. The fundamental theory of a radar was first thought of by James Clerk Maxwell [11], but the classical radar as a concept was the work of Heinrich Hertz in the late 19$^{\text{th}}$ century [11]. The technology did not see wide use until the 1930s when the military in several countries started to use it to detect aeroplanes. Since then radar units have gained a multitude of applications and are commonly used in the automotive industry. Adaptive Cruise Control (ACC) and Autonomous Emergency Braking-system (AEB) are two features modern cars provide and which rely on radar measurements.

3.1.1 Signal processing

The basics for how these measurements work follow a few simple steps. The radar sends an electromagnetic wave from the antenna. The electromagnetic wave propagate in space with the speed of light until a target is hit, which will return an echo. Once the echo reaches the receiver the measurement is complete. The slant range, R_{sl}, is the distance from the radar to the targets position in space. The slant range is then simply calculated with

$$R_{sl} = c\frac{\tau}{2}, \tag{3.1}$$

where c is the speed of light, τ is the time it takes the wave to make a round trip and 2 is for calculating the one-way distance.

As seen in Figure 3.1 a transmitted and a received wave can have a slight offset in the frequency. This difference is called the Doppler shift, f_D, and is caused by relative motion between target and the radar. By using (3.2), where f_c is the operating frequency relating to the wavelength $\lambda = c/f_c$, the relative radial velocity can be calculated from the Doppler shift [12] as

$$f_D = \frac{2v_{rel}}{\lambda} = \frac{2f_c v_{rel}}{c}. \tag{3.2}$$

A commonly used concept is SNR. It is a relation between the received signal and background noise. It can be described by

$$SNR = \frac{P_{signal}}{P_{noise}}, \tag{3.3}$$

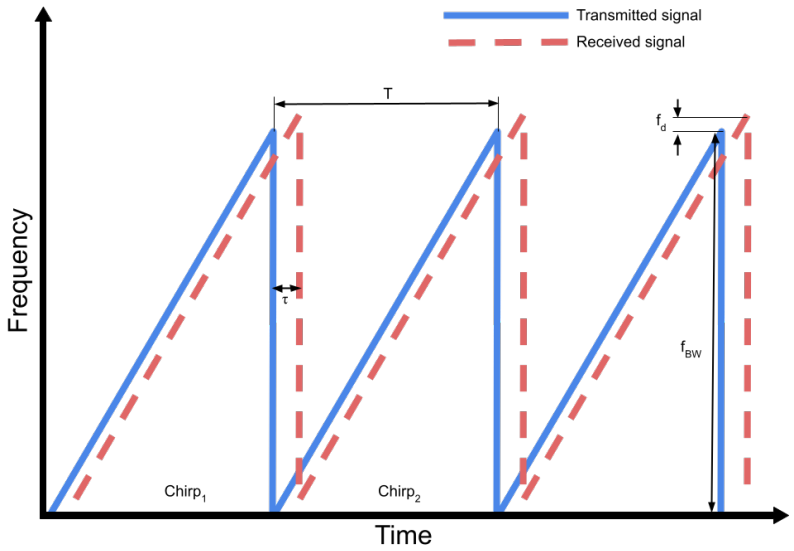


Figure 3.1: Ranging with FMCW radar. T is the time between chirps, τ is time for a round trip, f_d is the Doppler Shift and f_{BW} is the bandwidth.

where P_{signal} is the amplitude from the target signal and P_{noise} is the interference baseline. A common disturbance is Johnson-Nyquist noise, also known as thermal noise. This is found in all electronic measurements and originate from current carriers vibrating [13]. Applying the concept of SNR on radar signals the received power is of interest. Study the radar equation [12],

$$P_r = \frac{P_t G_t G_r \lambda \sigma}{(4\pi)^3 R_{sl}^4 L}, \tag{3.4}$$

where

$$\begin{aligned} P_r &= \text{receive power}, \\ P_t &= \text{transmit power}, \\ G_r &= \text{receive antenna gain}, \\ G_t &= \text{transmit antenna gain}, \\ \lambda &= \text{wavelength}, \\ \sigma &= \text{target Radar Cross-Section (RCS)}, \\ R_{sl} &= \text{slant range, and} \\ L &= \text{loss}. \end{aligned}$$

The SNR can be rewritten as

$$SNR = \frac{P_r}{P_{noise}} \tag{3.5}$$

where P_r is given by (3.4) and the thermal noise is given by

$$P_{noise} = kT_0BF. \tag{3.6}$$

The parameters for P_{noise} are

$$\begin{aligned} k &= \text{Boltzmann's constant,} \\ T_0 &= \text{reference temperature,} \\ B &= \text{bandwidth, and} \\ F &= \text{noise figure.} \end{aligned}$$

Almost all of the parameters in (3.5) and (3.6) are given by the state of the radar. There are only two depending on the target. The first one is, R_{sl}, the slant range, which is the distance to the target. The second parameter, σ, is the RCS. The RCS relates to the target's physical properties, such as material and shape, and is in a sense the quantity for how easily an object is detectable by a radar unit. This parameter is measured in dBsm, which relate the target in dB to one square meter.

3.1.2 Angle of arrival and the multipath problem

The basics for how to find the distance to a target has been explained, but without knowing from which direction the signal came from we can not pinpoint the target. For a radar unit to be able to detect the direction of an object an array of at least two antenna elements is needed. This follows from how the angle of arrival for the incoming wave is calculated. The wavefront from the echo, returning at an angle, will hit the antenna elements at different points in time, as there is a distance, d, between the elements. This will cause the phase of the wave to have slight differences, $\Delta\Phi$ [14]. This is illustrated in Figure 3.2. By measuring these differences it is possible to tell the angle of arrival using

$$\theta = \cos^{-1}\left(\frac{\lambda\Delta\Phi}{2\pi d}\right). \tag{3.7}$$

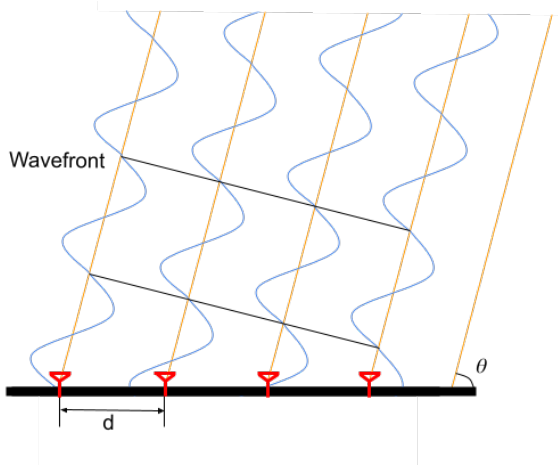


Figure 3.2: A simple illustration of how to get the direction of a target. Using the wavelength, λ, the distance d, between the antenna elements and the measured phase shift $\Delta\Phi$, the angle of arrival can be calculated. In the illustration the blue lines give a visual reference for how the phase differs between antenna elements in red. The orange lines give the angle of arrival.

Multipath wave propagation is when the same object is detected through two, or more, paths. This results in the same object having two, or more, positions in space from the point of view of the radar. See Figure 3.3 for a visualisation of this problem.

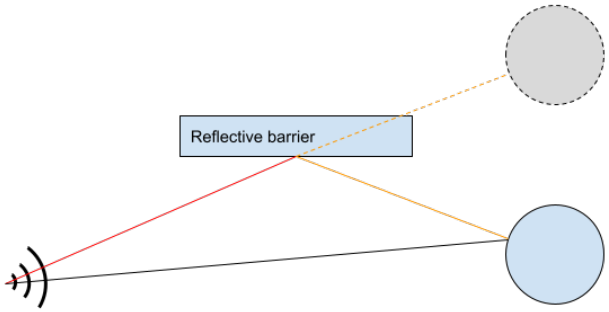


Figure 3.3: Illustration of multipath wave propagation. The direct path to the main target, the blue circle, is represented by the black line. The indirect path is represented at first by the red line and it propagates until it hit a target, the reflective surface. An echo propagates back to the radar and this is a correct detection. As the barrier is reflective, some of the wave do not propagate back following the red line yet, but instead follow the orange line. This will send an echo from the main target. This echo will have the total length of both red and orange lines combined, thus tricking the radar into seeing two objects from the same direction, the reflective barrier and the grey dashed circle.

3.2 Bayesian position estimation

One of the core requirements for an autonomous vehicle is to be able to navigate in its surroundings. To successfully achieve this the vehicle needs to gather information about the environment via hardware sensors like radar, LiDAR and IMU. However, in practice all sensors are prone to noise and will never be able to always accurately depict a perfect representation of reality. To counteract this problem methods like Bayesian filtering can be applied [15]. This kind of filter use probabilistic models to distinguish noise and false detections from desirable information.

3.2.1 State-space representation

For Bayesian filters to work a representation of states in space is required. When working with discrete time intervals the state-space can be represented by,

$$x_{t+1} = f(x_t, u_t), \tag{3.8}$$
$$y_t = g(x_t, u_t), \tag{3.9}$$

where x_t represents the systems state vector, u_t the input and y_t the output. The variable x_{t+1} represents the discrete time derivative of the state which is a function of current state x_t and the input u_t. Similarly, the output state y_t takes the same inputs. For a discrete time system where the functions f and g are chosen to be linear [16] the state-space model can be expressed as,

$$x_{t+1} = Ax_t + Bu_t, \tag{3.10}$$
$$y_t = Cx_t + Du_t. \tag{3.11}$$

3.2.2 Bayesian filtering

The position, also referred to as pose, of the vehicle is dependent on the current state x_t. When the system transitions from state t to $t+1$ due to, for example some control command u_{t+1} sent to the vehicle, the new state produces a measurement z_{t+1}. The relationship between controls, states and measurements are illustrated in Figure 3.4. In the figure we can also see x_0 as the initial pose, x_T as the ending pose and all poses together will be denoted as $\mathbf{x}$.

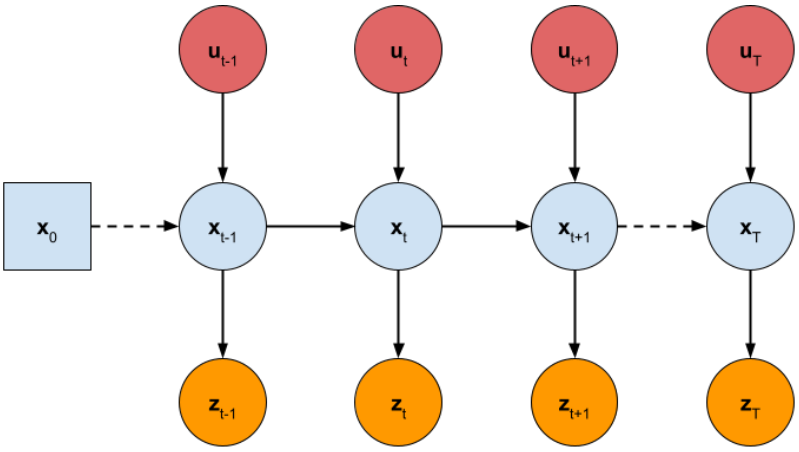


Figure 3.4: A schematic overview of the relationship between states (blue), controls (red) and measurements (orange) in Bayesian filtering.

Bayes filter is one of the more general algorithms for estimating the states in $\mathbf{x}$. The filter uses something called beliefs, which represent the vehicles knowledge of its position. As mentioned earlier these beliefs are estimates and not the same as true positions. The algorithm calculates a belief $bel(x_t)$ at time t based on the previous belief $bel(x_{t-1})$ at time $t-1$ together with the control u_t and measurement z_t. This calculation is done recursively and in its most basic form consists of two steps. First step is the prediction step which calculates a belief of x_t based on only the previous belief $bel(x_{t-1})$ and control u_t,

$$bel'(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx. \tag{3.12}$$

The next step is called the update step and it is using the results from the prediction step, the current pose x_t and the measurement z_t,

$$bel(x_t) = c * p(z_t|x_t)bel'(x_t). \tag{3.13}$$

The mathematical relationship shown in these equations can be summarised with step one spreading the resulting state distribution with noise from the control data u_t and step two tightens the state distribution by using the measurement data z_t. The constant c is a normalisation constant which is required for the results to correctly represent a probability. By providing an initial condition for $bel(x_0)$ the algorithm can solve simple localisation problems for a vehicle.

3.2.3 Kalman filtering

One of the most common techniques that use Bayes filter is a Kalman filter. It was develop by R.E Kalman and R.S Bucy in their work on linear filtering and prediction theory [17]. This filter uses linear stochastic processes with added white Gaussian noise. The basic idea is that the belief of the state $bel(x_t)$ at some time t can be expressed by its mean and covariance, and the Kalman filter estimates optimal values for them. For Kalman filter to work there are three properties the system has to have as prerequisites [18]:

1. The current state x_t needs to be represented by only linear arguments,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{3.14}$$

   where A_t, B_t are constants and the added noise ϵ_t is Gaussian distributed.
2. The current state z_t needs to be represented by only linear arguments,

$$z_t = C_t x_t + \delta_t, \tag{3.15}$$

   where C_t is a constant and added noise δ_t is Gaussian distributed.
3. The start belief $bel(x_0)$ has to be Gaussian distributed.

The implementation details for the Kalman filter algorithm can be found in the equations below. The Bayes belief function $bel(x_t)$ is represented by the mean state vector $\boldsymbol{\mu_t}$ and the covariance matrix $\boldsymbol{\Sigma_t}$. The prediction step includes,

$$\begin{aligned} \bar{\boldsymbol{\mu}}_t &= \boldsymbol{A_t \mu_{t-1}} + \boldsymbol{B_t u_t}, \\ \bar{\boldsymbol{\Sigma}}_t &= \boldsymbol{A_t \Sigma_{t-1} A_t^T} + \boldsymbol{R_t} \end{aligned} \tag{3.16}$$

and the update step is executed by

$$\begin{aligned} \boldsymbol{K_t} &= \bar{\boldsymbol{\Sigma}}_t \boldsymbol{C_t^T} (\boldsymbol{C_t} \bar{\boldsymbol{\Sigma}}_t \boldsymbol{C_t^T} + \boldsymbol{Q_t})^{-1}, \\ \boldsymbol{\mu_t} &= \boldsymbol{\mu_t} + \boldsymbol{K_t}(\boldsymbol{z_t} - \boldsymbol{C_t} \hat{\boldsymbol{\mu}}_t), \\ \boldsymbol{\Sigma_t} &= (\boldsymbol{I} - \boldsymbol{K_t C_t}) \bar{\boldsymbol{\Sigma}}_t. \end{aligned} \tag{3.17}$$

Capitalised symbols are representing matrices and non-capitalised symbols are representing vectors. Equation (3.16) result in the state transition probability and is equal to the predictions of the mean state vector and the covariance matrix for the next time step. Notice also the Gaussian distributed noise is added to the covariance prediction. These predictions are then used in the second update step (3.17) to get the Gaussian corrected Kalman gain $\boldsymbol{K_t}$ and the final mean state vector $\boldsymbol{\mu_t}$ and covariance matrix $\boldsymbol{\Sigma_t}$ [18]. These steps are recursively calculated for every time step t.

3.2.4 Extended Kalman filtering

Since the normal Kalman filter require linear relationship to work optimally there are other solutions to handle non-linear relationships. One such solution is called Extended Kalman Filter (EKF) and it is handling smaller non-linearities by linearising the model before applying the Kalman filter [15]. This is normally done by expanding the function to the first order Taylor expansion around the estimated state at each time step. Since this is only a first order approximation EKF works best for smaller non-linearities and will perform poorly if the first order approximation deviates too much from the non-linear function.

Here is an example of the continuous case where the functions g and h in,

$$\begin{aligned} x(t) &= g(x(t), u(t)) + \epsilon(t), \\ z(t) &= h(x(t), u(t)) + \delta(t), \end{aligned} \tag{3.18}$$

are non-linear. First approximate them with the first order Taylor expansion, noted g^* and h^*, at time step t and secondly apply the two steps of the Kalman filter algorithm [18],

$$\begin{aligned} \bar{\boldsymbol{\mu}}_{\boldsymbol{t}} &= g^*(\boldsymbol{\mu}_{\boldsymbol{t-1}}, \boldsymbol{u}_{\boldsymbol{t}}), \\ \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}} &= \boldsymbol{G}_{\boldsymbol{t}} \boldsymbol{\Sigma}_{\boldsymbol{t-1}} \boldsymbol{G}_{\boldsymbol{t}}^{\boldsymbol{T}} + \boldsymbol{R}_{\boldsymbol{t}}, \end{aligned} \tag{3.19}$$

and

$$\begin{aligned} \boldsymbol{K}_{\boldsymbol{t}} &= \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}} \boldsymbol{H}_{\boldsymbol{t}}^{\boldsymbol{T}} (\boldsymbol{H}_{\boldsymbol{t}} \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}} \boldsymbol{H}_{\boldsymbol{t}}^{\boldsymbol{T}} + \boldsymbol{Q}_{\boldsymbol{t}})^{-1}, \\ \boldsymbol{\mu}_{\boldsymbol{t}} &= \boldsymbol{\mu}_{\boldsymbol{t}} + \boldsymbol{K}_{\boldsymbol{t}} (\boldsymbol{z}_{\boldsymbol{t}} - h^*(\hat{\boldsymbol{\mu}}_{\boldsymbol{t}})), \\ \boldsymbol{\Sigma}_{\boldsymbol{t}} &= (\boldsymbol{I} - \boldsymbol{K}_{\boldsymbol{t}} \boldsymbol{H}_{\boldsymbol{t}}) \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}}. \end{aligned} \tag{3.20}$$

The only difference in the algorithm compared to the standard Kalman filter is the application of the approximated non-linear functions, g^* and h^*, at the beginning of the two steps.

3.3 Simultaneous Localisation And Mapping

The basics for any Simultaneous Localisation And Mapping (SLAM) problem is to estimate the trajectory and the map of the surroundings while moving. All measurements have noise and thus most SLAM modules use tools based on probabilities for producing results.

The tuk-tuk moves along on a trajectory represented by a sequence of random variables $\mathbf{x}_{1:T} = \{\mathbf{x}_1, ..., \mathbf{x}_T\}$. While moving along this trajectory, the IMU provides odometry measurements $\mathbf{u}_{1:T} = \{\mathbf{u}_1, ..., \mathbf{u}_T\}$ and data of the environment from the radar $\mathbf{z}_{1:T} = \{\mathbf{z}_1, ..., \mathbf{z}_T\}$. Then solving the SLAM problem is to estimate

$$p(\mathbf{x}_{1:T}, m | \mathbf{z}_{1:T}, \mathbf{u}_{1:T}, \mathbf{x}_0), \tag{3.21}$$

the posterior probability of the trajectory $\mathbf{x}_{1:T}$ and the map m of the surroundings given $\mathbf{u}_{1:T}$ and $\mathbf{z}_{1:T}$, and an initial $\mathbf{x}_0$.

The estimation of the posterior can be done with two assumptions, the Markov assumption and the static world assumption, using a dynamic Bayesian network [19]. A dynamic Bayesian network is a way to represent variables over time. This is done in a directed graph as seen in Figure 3.5. There is one node for each variable, a directed edge between two nodes imply a conditional dependence between them.

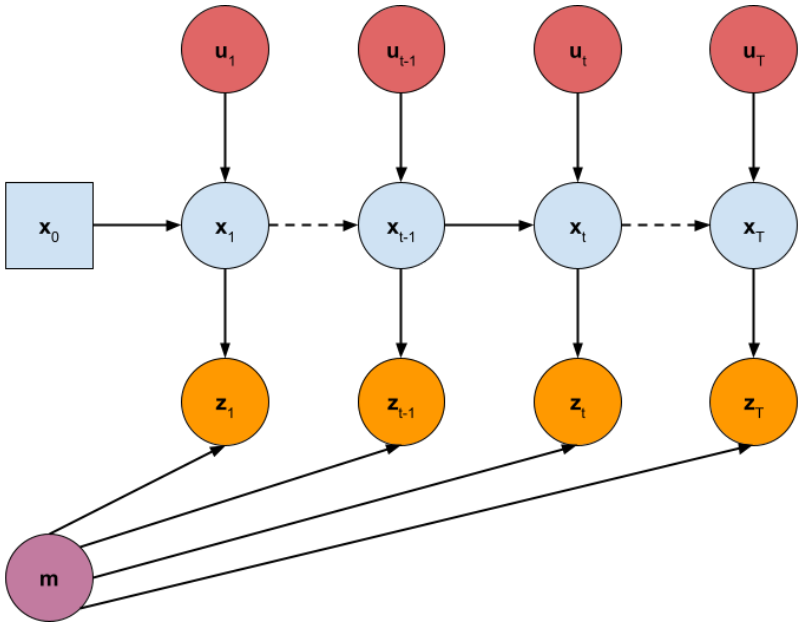


Figure 3.5: A Dynamic Bayesian Network relating to the SLAM process. $\mathbf{u}_{1:T}$ and $\mathbf{z}_{1:T}$ are the observed variables, whereas $\mathbf{x}_{1:T}$ and m are hidden variables.

There are two models building this graph. The state transition model and the observation model. The transition model gives the probability that at a time, t, the tuk-tuk is at position $\mathbf{x}_t$ given that in the previous time step, $t-1$, the tuk-tuk was in position $\mathbf{x}_{t-1}$ and got $\mathbf{u}_t$ from the odometry. Essentially, the two edges leading to $\mathbf{x}_t$ is used for calculating the probability using $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$. This works analogously for the observation model. With $\mathbf{z}_t$ depending on $\mathbf{x}_t$ and $\mathbf{m}$, the current position and the static map, the probability to perform observation $\mathbf{z}_t$ is given by $p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m})$.

3.3.1 GraphSLAM

There are alternative ways of representing the dynamic Bayesian network. One is in a "graph-based" approach which put emphasis on the spatial structure. The different poses of the tuk-tuk are stored as nodes in a graph and labelled after their position in the surroundings. The observations, $\mathbf{z}_t$, or odometry measurements $\mathbf{u}_t$ are spatial constraints and represented in edges between two nodes. In a sense the graph is constructed purely by the measurements as each node represent the position of the tuk-tuk and a measurement from that position. When a whole graph is constructed it is an optimisation problem to find the best configuration of poses given the constraints. This means that the problem could be divided into two, the graph construction, usually is called Front-End, and the graph optimisation, usually called Back-End.

3.3.1.1 Front-End

As mentioned before, the Front-End constructs the graph of nodes, from poses, and the edges, from the measurements. This builds a chain of nodes with an edge

between the next and previous node. Edges can also be added to nodes that are not adjacent to each other in the chain if there are similarities in their measurements, or constraints. If the constraints are similar enough, without being adjacent, a loop closure is formed placing these positions in close proximity of each other. A simple illustration of this is shown in Figure 3.6.

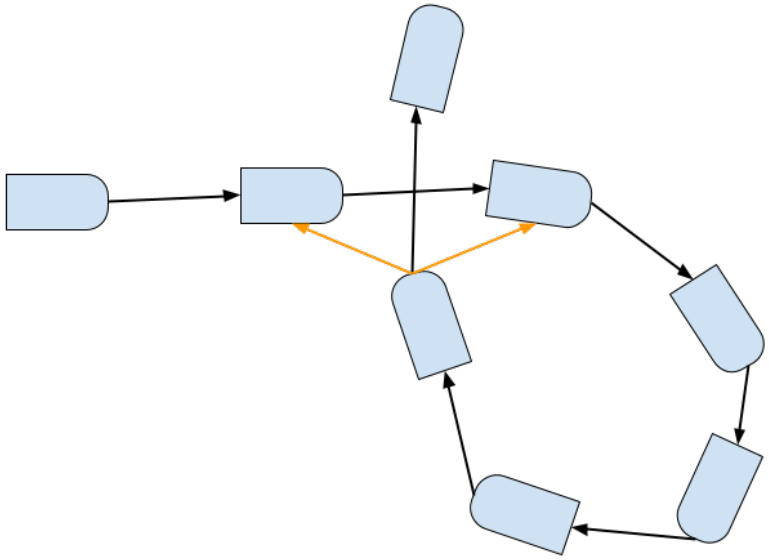


Figure 3.6: A set of tuk-tuk poses, or nodes, in blue. The edges, or constraints, are the black arrows and the orange arrows suggest loop closures for finding similarities in the measurements.

Only using dead reckoning, an estimation of the new positions of a vehicle based on steering angle and wheel rotation since the previous position, give poor results. As drifting occur over time, additional information is beneficial for tracking the position and that is why loop closures are desirable. Performing a correct loop closure is however not easy. If objects move there will be features missing and if there is structural resemblance, such as a corner, one might be hard to differ from the other.

A way to increase the accuracy of loop closure is grouping the suggested loop closures [20], also known as hypotheses, h. As seen in Figure 3.7 there are several hypotheses in two groups. The idea is to, in one of these groups, form a pair-wise consistency for each pair of hypotheses, resulting in a consistency matrix A. Each element in the matrix represent the probability of the paired hypotheses h_i and h_j being in agreement. A vector $\mathbf{v}$, called an indicator vector, can be formed by assigning $\mathbf{v}_i = 1$ if the i^{th} hypothesis should be accepted, and otherwise $\mathbf{v}_i = 0$ [20]. The average of the pair-wise consistency, λ, can then be calculated using $\mathbf{v}$ and A with the intention to maximise λ using

$$\lambda = \frac{\mathbf{v}^T \mathbf{A} \mathbf{v}}{\mathbf{v}^T \mathbf{v}}. \tag{3.22}$$

From

$$A\mathbf{v} = \lambda \mathbf{v} \tag{3.23}$$

the value of $\mathbf{v}$ that maximises λ is seen to be the dominant eigenvector of A and the dominant eigenvalue is λ [20]. If the ratio of the two largest eigenvalues, λ_m/λ_n, is large, $\mathbf{v}_m$ is considered an accepted loop closure. If there is no eigenvalue large enough, include more hypotheses until one is found.

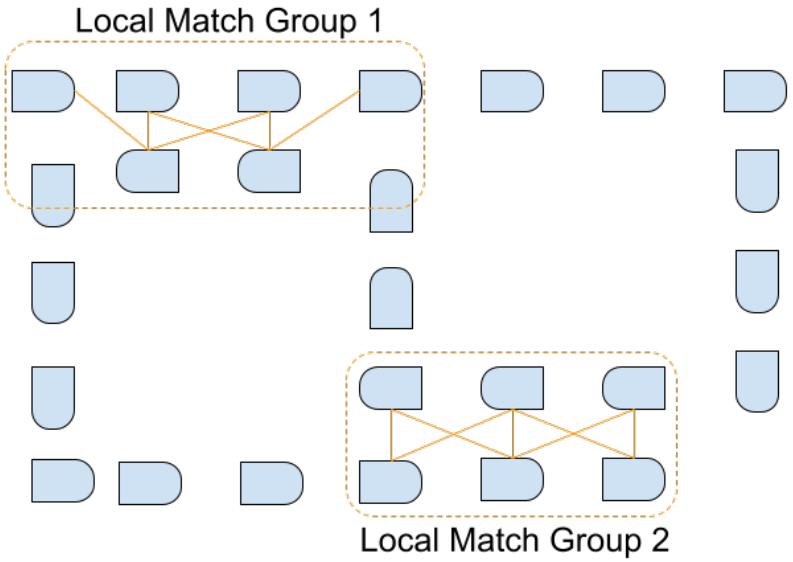


Figure 3.7: Example of how Local Match Groups are formed.

3.3.1.2 Back-End

The Back-End use the fully built graph, consisting of poses and observations, and optimises this by finding the maximum of the objective function,

$$[\mathbf{p}^*, m^*] = \underset{\mathbf{p},m}{\operatorname{argmax}}\, p(\mathbf{p}_{0:T}, m|\mathbf{z}_{1:T}, \mathbf{u}_{1:T}). \tag{3.24}$$

(3.24) can be rewritten as a minimisation problem [21],

$$[\mathbf{p}^*, m^*] = \underset{\mathbf{p},m}{\operatorname{argmin}} \sum_k \mathbf{e}_k^T \mathbf{\Omega}_k \mathbf{e}_k \tag{3.25}$$

and solved using Levenberg-Marquardt. Here, $\mathbf{e}$ is the error, k is the specific constraints and $\mathbf{\Omega}$ contain the constraint information.

The optimised graph might be subject to distortions as it is sensitive to outliers and an incorrectly performed loop closure could be detrimental. Therefore the removal of outliers greatly increase the robustness of the optimisation. Preferably, the removal of outliers is made in both the Front-End and the Back-End. One way of removing outliers in the Back-End is by using dynamic covariance scaling, which scale the edges in proportion to the objective function [22].

4

Implementation

4.1 Hardware and software integration

In the automotive industry the integration between the hardware and the software have become an increasingly more important topic as cars get more dependent on software for increased performance and functionality. This section will go through the general schematic shown in Figure 4.1 of how the radar hardware in the tuk-tuk is connected to the software.

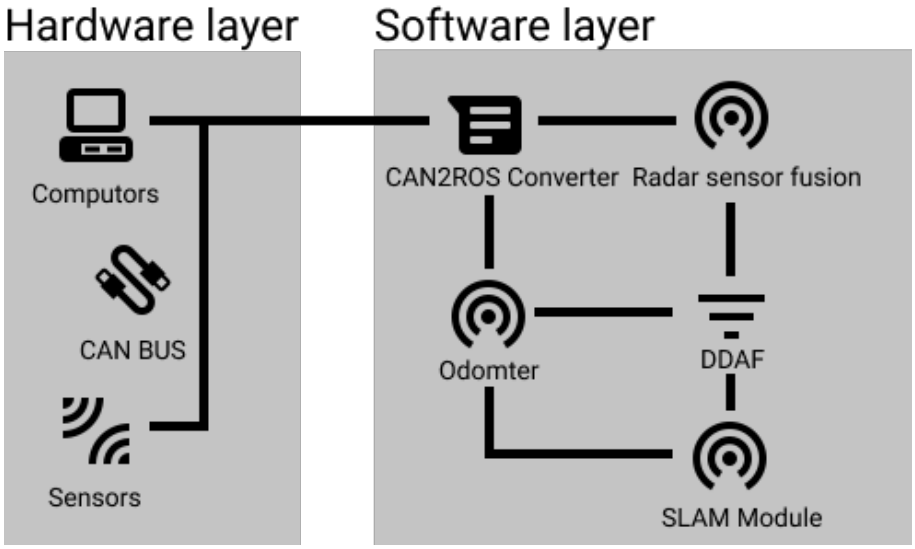


Figure 4.1: General connection schematic of the integrated hardware and software used in this thesis.

4.1.1 Hardware

There are four radar units, using the 77 GHz band, in the tuk-tuk providing data. An approximation of placement and orientation can be seen in Figure 4.2. They are named after their placement on the tuk-tuk, Front Left (FL), Front Right (FR), Rear Left (RL) Rear Right (RR). The radar units positions are given in VCS coordinate frame and can be found in Table 4.1. All these units are connected via the Controller Area Network bus (CAN bus) onto a CAN bus reader which in turn is connected to one of the computers. All the other vehicle related hardware, like the IMU and control unit, also goes via the CAN bus.

Table 4.1: The positions of the radar units in VCS and the orientation given in an Euler angles.

Radar unit	x [m]	y [m]	z [m]	Pitch	Roll	Yaw
FL	-0.24	-0.45	-0.64	180°	0°	309.0°
FR	-0.24	0.45	-0.64	180°	0°	50.5°
RL	-2.11	-0.37	-0.64	180°	0°	198.5°
RR	-2.11	0.37	-0.64	180°	0°	164.0°

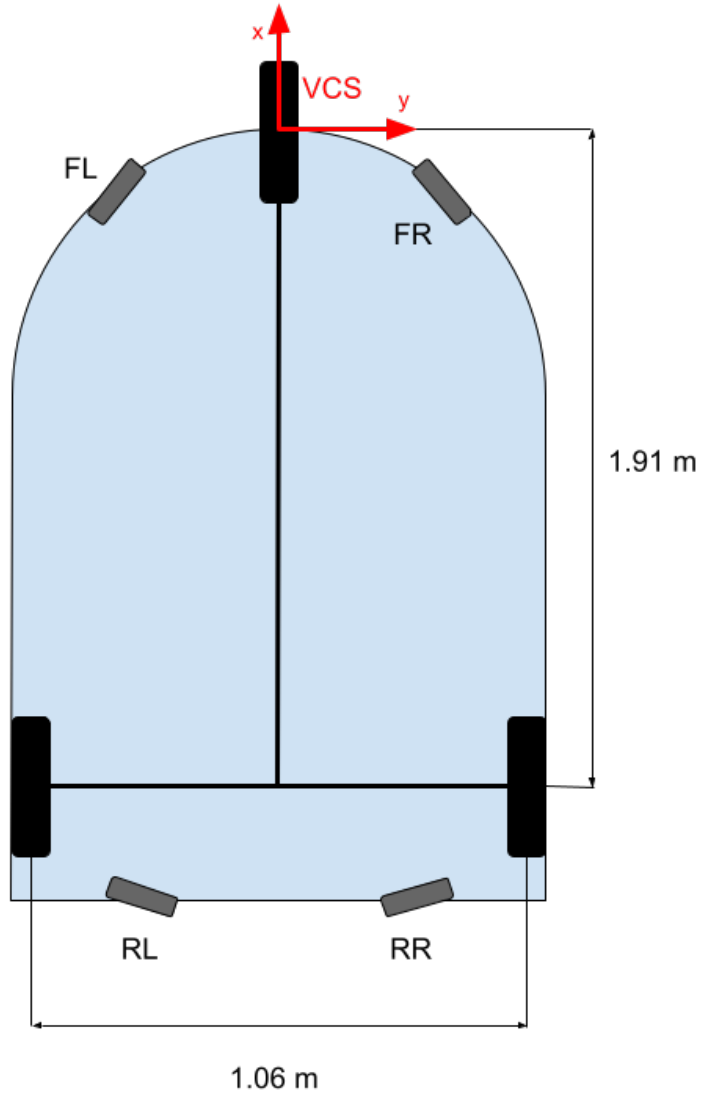


Figure 4.2: A simple schematic overview of the tuk-tuk. The four radar units are named FL, FR, RL, RR. The origin of the VCS is show in the image. Note that VCS follow the North, East, Down (NED) convention.

4.1.2 Software

Like the schematic in Figure 4.1 illustrates there is a ROS node handling the conversion between the information on the CAN bus from the CAN protocol to ROS messages. This makes it possible for other nodes to take part of only the information that is relevant for that particular node. Since the radar data comes from four different detectors, thus four different coordinate systems, one ROS node is responsible for aggregating and publishing the data in the VCS coordinate frame. After that the messages are feed into our custom ROS node which in turn is input to the SLAM module together with the odometry data from the IMU. More details on how ROS and the custom node works can be found in section 4.3.

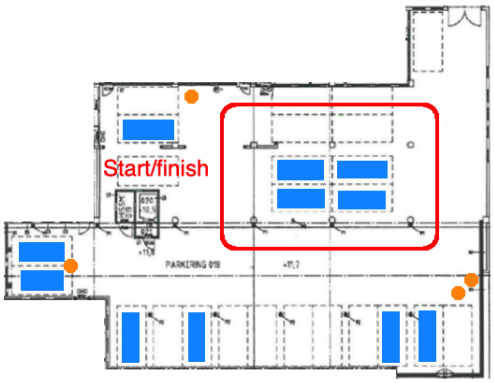
4.2 Data collection

For this thesis several data sets have been collected in two different settings. The main focus is on the garage data set, see section 4.2.1. As a reference, one additional data set was collected at a parking lot outside the Aptiv office, see section 4.2.2.

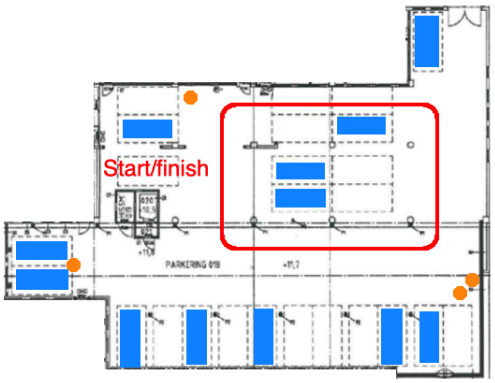
4.2.1 Aptiv office - Garage

The garage data set was collected in the basement of Aptiv's facilities. The majority of the surroundings were either metal or concrete. In this setting two different parameters were tested for a total of 6 scenarios and 30 laps in the garage.

Data was collected with two different configurations of parked cars, as seen in Figure 4.3. In addition, the effect of corner reflectors were tested in three steps. No reflector at all, in the height of the radar units, and at the maximum height of the tripods (approximately 1.2 m).



(a) The first configuration of parked cars during testing.



(b) The second configuration of parked cars during testing.

Figure 4.3: Configuration of parked cars in the garage during testing marked in blue. An estimation of the route in red and placement of corner reflectors in orange.

4.2.2 Aptiv office - Parking lot

The second site for data collection is an outdoor environment and works as a reference to the indoor data set. This setting was less controlled as there were no pool cars to move and the parking lot was under the influence of the weather. A few days before the data collection there was heavy snowfall resulting in snow banks at the parking lot. With these prerequisites the corner reflectors were tested as before with five laps each.

Figure 4.4: Altered Google maps image of the area used for data collection at Aptiv's parking lot. As seen in the figure an approximate route has been added in red. Since the data collection was mid February there was plenty of snow which is not in the image and the car does not represent the collected data. Image from [1].

4.3 Robot operating system (ROS)

As opposed to what the name implies ROS is not an operating system, but an open source software framework for developing robotics projects in C++ and Python. The intended use of ROS is more specifically to be a framework that effectively bridge different kinds of sensor data to a uniform format together with the functionality for a robot to understand and navigate in an environment [23]. As such it is suitable for a vehicle as well since it needs to perform tasks like reading the IMU and providing steering and throttle signals. This is one of many reasons that Aptiv decided to port

their CMOD R&D project to ROS. ROS also has the benefit of being open source which in turn has created an ecosystem of open source community and commercially developed ROS modules. One such project is Mobile Robot Programming Toolkit (MRPT) SLAM [24] which will be discussed further in section 4.5.

4.3.1 Network structure

ROS operates in a network-like fashion consisting of so called nodes. Nodes work like computational containers and can have different kinds of responsibilities. In general a ROS node usually has one or more of three different kinds of responsibilities. The responsibilities are called Publishers, Subscribers and Services. They can all communicate with each others via messages. Publishers can publish different kinds of messages, subscribers can listen to publishers for messages and services have the ability to work with a request/reply structure. A message is a specified type of data structure that associates a data type and name with a data value. This makes it possible for the nodes to always know how to interpret the data received on the network. Another important ROS concept is the called topic, which is simply a name for the message stream published or subscribed on the network.

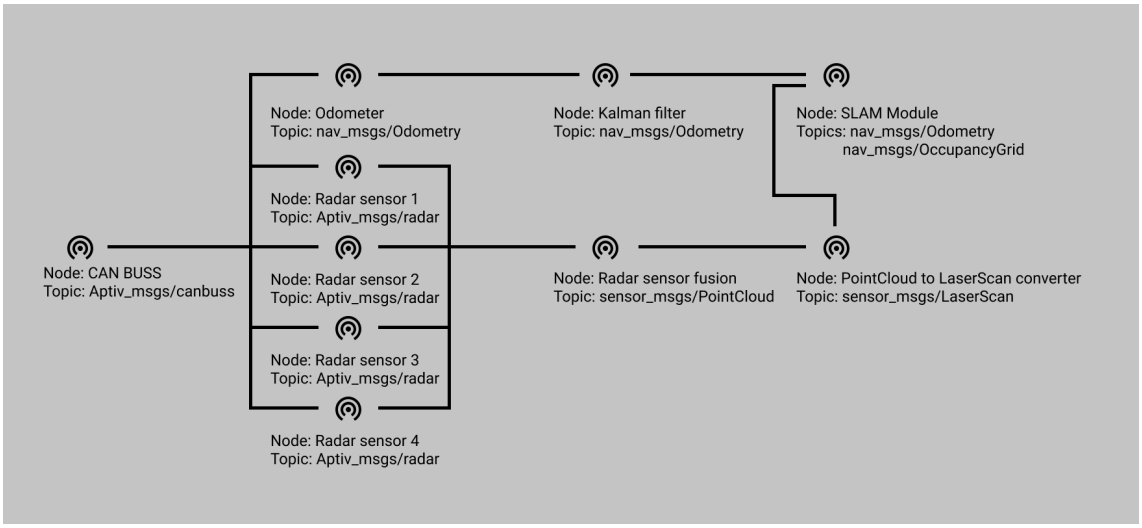


Figure 4.5: An overview of the node schematics used in this thesis.

To interface with the SLAM modules described in section 3.3 a new publisher node was needed to convert the radar data in sensor-msgs/PointCloud format to sensor-msgs/LaserScan format. More details on how the conversion was done will be described in section 4.4. In addition to the converted message another sensor-msgs/PointCloud message was also published with the same radar data in a new coordinate system for easier data analysis. More details on how transforms work is provided in section 4.3.2. One of the very powerful features in ROS is their extensive library of standard messages types. In addition to that, developers also have the ability to create their own message type defined according to their desire. This combination enables high modularity and compatibility with both proprietary and open source software.

4.3.2 Transforms

One other central functionality of ROS is the ability to define and transform one coordinate system to another. In addition to topics publishing different messages the ROS network also broadcast these so called transforms. This works via defining how different coordinate system relate to each other and if the relationship is of static or dynamic type. With ROS transform listeners it is possible to listen to transforms and go between coordinate systems, referred to as frames, in code runtime. When all the frames, and their relationships, have been defined ROS builds a transformation tree like the one shown in Figure 4.6. As long as there is a path between one part of the tree to another the framework allows for a transform between the frames.

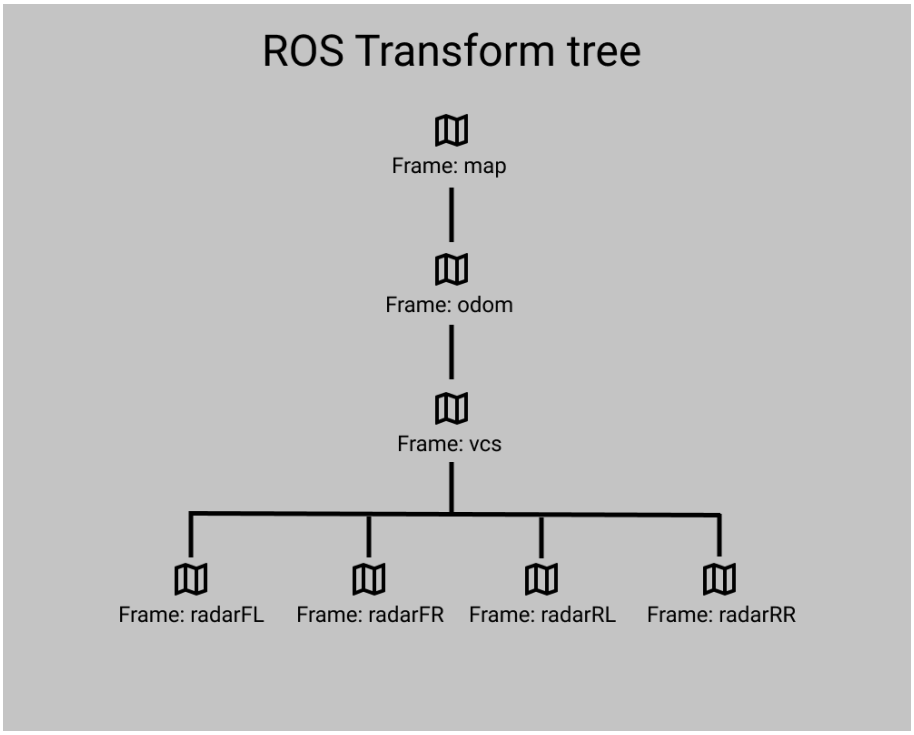


Figure 4.6: ROS transform tree.

There are three coordinate systems of importance in this thesis. The first is the odometry frame, or in short odom. The odom frame has its origin at the vehicles starting position and is static. The second coordinate system is called VCS and has its origin attached to the front bumper of the tuk-tuk and moves freely in the odom frame, see Figure 4.7. The third is called map and is corresponding to the global map. In the case of one vehicle the odom frame and the map frame are practically the same. If there are several vehicles or robots in one system then each unit gets their own odom frame while there will only be one global map frame. The four radar units output detections in the VCS frame. The odometry measurements gives an estimate of the position and heading based on dead reckoning in the odometry frame. After the dead reckoning is calculated, based on last known position and velocity, the measurements are run through a Kalman-filter to stabilise the vehicles path. This data is then published on the topic called /odometry, see Figure 4.5.

This topic is subscribed to by the SLAM module and the topic is also used to define the transform between the VCS frame to the odom frame.

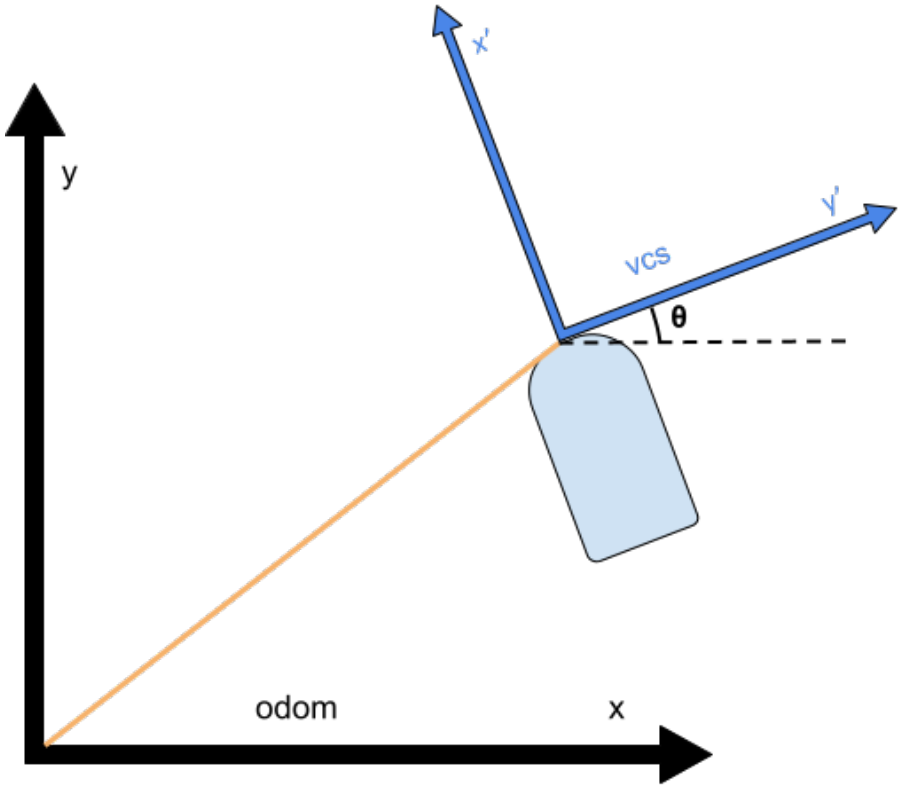


Figure 4.7: The odom frame is static and act as the coordinate system of the 2D world. The VCS frame is attached to the tuk-tuk, moves and rotates freely in the odom frame.

4.3.3 Rosbags

Rosbags enables us to record and replay ROS standardised data sets. A rosbag contains several topics, each with messages that have their own timestamp. By replaying the timestamps in order rosbags can simulate the original data collection. This is suitable for projects that can benefit from tuning hyperparameters since the same input data can be recreated in a semi deterministic manner. The reason why rosbags are semi deterministic has to do with the network structure of the framework. The hardware resources are distributed to each node in real time and thus, depending on the workload, different nodes can get assigned work in a different order between replays of the rosbag. This is commonly not a problem since the difference between replays is small. Despite that the ROS Aptiv software stack was designed to achieve perfect synchronisation between the nodes. This required a sophisticated C++ template structure incorporating parallel programming that ensured time synchronisation between nodes. To ensure rosbag replayability on that

level for the new radar ROS node was outside the scope of this thesis but the benefits of consistent input data to that node could still be used.

Rosbags allows us also to tune parameters in the system for specific data collection sites. Since the garage had bad GPS reception a ground truth based on GPS is not possible. Therefore parameters were tuned for the dead reckoning compensation on the odometer data such that it was sufficiently good to use as ground truth.

4.3.4 Visualisation

Visualising the data in real time was made with ROS Visualization (RViz), a program specifically made for this task. As the transforms used in ROS are accessible by RViz without any manipulation it was a convenient choice of software. In addition to this the different topics that are publishing data can be found in RViz and changed intermittently. Thus, making an analysis of the data in real time easy.

Visualising downloaded data, from rosbags, was not made using any ROS specific software, as these files can be read in python. The package pandas [25] was used for most of the data manipulation and plotting.

4.4 PointCloud to LaserScan converter

In order to work with the SLAM modules the input has to be in a specific message format called LaserScan. This format is native to, for example, LiDAR and the two most important features of the message are the detection range and azimuth angle. The data received from the four radar units is of PointCloud format, which uses Cartesian coordinates, and the message is published in the VCS frame. ROS does not support a standard conversion between the two message types so to go from the raw radar data of message type PointCloud a custom converter node is needed.

4.4.1 Standard LaserScan converter

The standard way of converting a PointCloud message to a LaserScan message can be found in the pseudo code in Algorithm 1 below. In short the LaserScan message divides the surounding space into circle segments, see Figure 4.8, in which only one detection per circle segment can be represented by a range value. To convert the Cartesian coordinate system to polar coordinates the standard implementation always chooses the detections closest to the detector. The range of the detections is restricted to 12 m since this was a more realistic range compared to the hardware limitations of about 100 m. The circle segment angle α was chosen to be 2° which makes the message fit a total of 180 detections per LaserScan message.

Algorithm 1 Standard PointCloud to LaserScan converter

```
for point in PointCloud do
    Calculate range and azimuth angle from x and y
    Convert the azimuth angle to corresponding LaserScan circle segment
    if range < range inside circle segment then
        circle segment range = range
    end if
end for
```

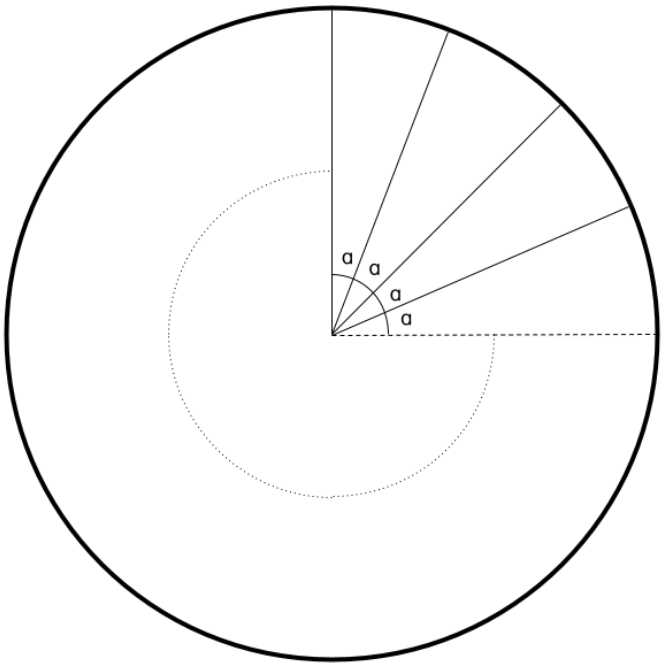


Figure 4.8: The area around the tuk-tuk was divided into a set amount of circular sectors, $n_{sec} = 360/\alpha$. Each sector could then represent the measurement of a specific angle in the LaserScan message.

4.4.2 Detection Density Augmentation Filter (DDAF)

DDAF takes the raw radar detections as input and process them to a suitable format for the SLAM module. This starts by dividing the area surrounding the tuk-tuk into circular sectors, see Figure 4.8, where the angular resolution of the LaserScan is $\alpha = 2°$. Each of these sectors can only have one range value as this is required for the LaserScan format. Choosing the range for each sector is done in four steps.

1. Fill a buffer of radar messages in ROS frame map.
2. If the buffer is full, transform the messages from frame map to frame VCS.
3. Check the detections density of each bin in each circle sector.
4. Choose the average range and RCS of the most dense region.

The buffer is filled by transforming the raw radar detections from the VCS frame to the global map frame. The buffer is set to 1 second which is equivalent to 20 raw radar messages. Each time a new radar message is published the oldest message

is replaced in the buffer. When the buffer is filled, start executing the rest of the algorithm. Each LaserScan sector is divided into bins of 0.5 m, see Figure 4.9. The bin contains the detections within the area it represents. The detections density is calculated by dividing the number of detections in the bin by the area of the bin, for each bin. The bin with the highest detection density determines the range by calculating the mean range of all detections in the bin. Since the radar RCS values is of unit dBsm they can not be averaged and the highest RCS value in the bin was chosen instead. In addition to this the algorithm can also choose to reject bins of some fixed size, if that size was considered to low. This way it can effectively filter out for instance single detections. This is not possible for the standard LaserScan algorithm since almost all detections are a single detection for that specific circle segment.

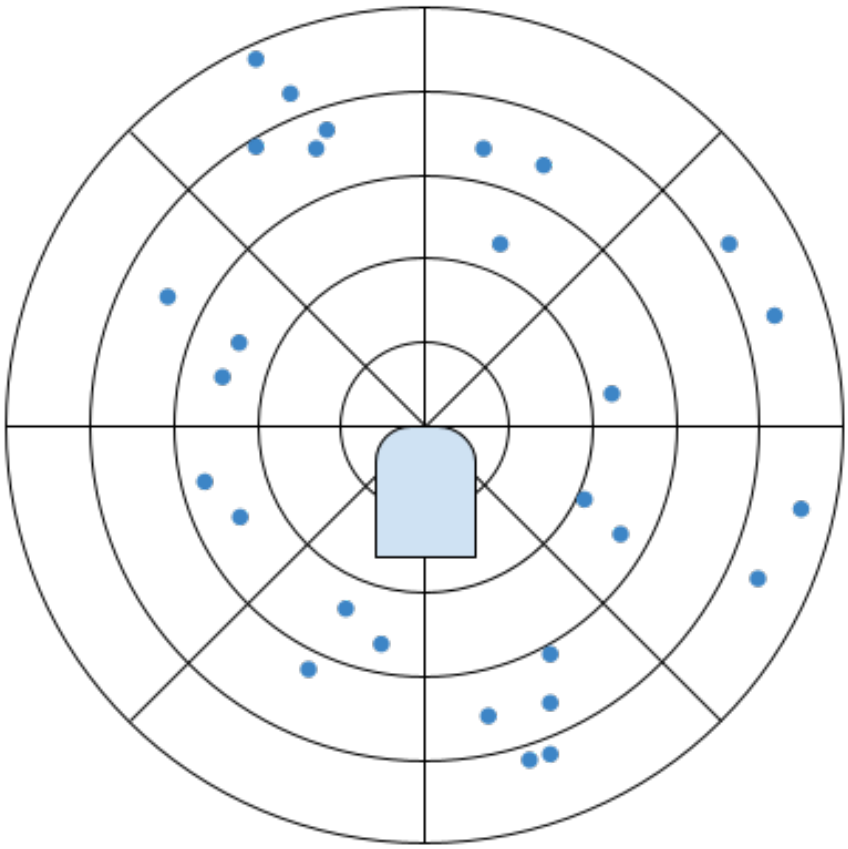


Figure 4.9: Each sector was divided into bins of length 0.5 m, where each bin contained the number of detections in the range. As the lengths are known, the area of each bin could be used to calculate the density of detections. The range passed to the LaserScan message is the mean of the detections in the bin with the highest density.

Max range [m]	Buffer Size	Angular resolution α [°]	LNIB
12	20 (1 s)	2	2

The hyperparameters of the algorithm is the following:

- Max range
- Buffer size
- Angular resolution α
- Least Number of detection In Bin (LNIB)

All of these parameters will change the resulting LaserScan message and will be the main parameters for balancing the filtering and augmentation effect of the algorithm. A more detailed discussion about the hyperparameters can be found in the discussion section 6.3. Optimal buffer size has been investigated and was chosen to be when the DDAF algorithm produced the same amount of detections as the raw radar detections published.

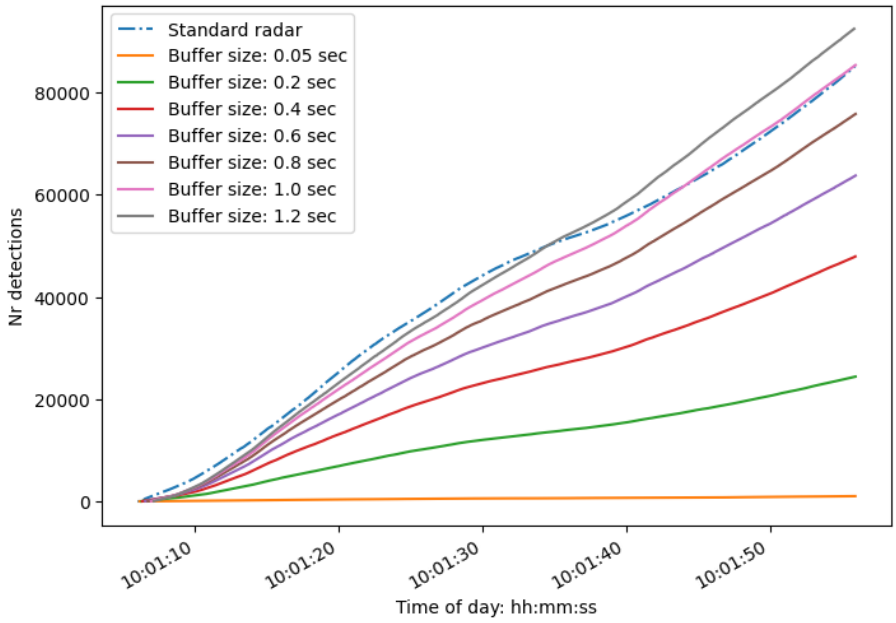


Figure 4.10: As seen in the figure larger buffer sizes corresponds to more data being produced per message. This is the data count from one lap in the garage corresponding to approximately 50 seconds.

4.5 SLAM implementation

Adding new existing modules to a ROS project generally works in the same way. First the required module dependencies must be installed via the sudo apt-install command for the correct ROS version. Then the module can be placed in the ROS work-space folder and compiled. The ROS MRPT SLAM modules are an open source project released under the BSD license with core libraries written in plain

C++. The MRPT libraries are commonly used by academics within the robotics research areas. The ROS MRPT Graph SLAM module comes with example code and a demo rosbag but with no standardised documentation. The first step of integrating the module with an existing project is to define the baselink, odom and map frame. The baselink frame is defined as the coordinate system based on the vehicle and is set to the VCS frame. The other two frames are already the same name as in the existing project and thus no variables needs to be changed. The second step is to assign the correct topics to the corresponding subscriber nodes. In the case of MRPT Graph SLAM this corresponds to one LaserScan subscriber and one Odometry subscriber. In general this should suffice for a working module but additional changes was needed for integration with Aptivs CMOD project. The CMOD project already has integrated a Kalman Filter that takes care of the transforms between the VCS, odom and map frame and this module came into conflict with the MRPT Graph SLAM module. The problem that occurred was that both modules broadcasted transforms between the odom and map frame but based on different localisation data since both modules takes the raw odometry data as input but use different algorithms for producing the output. The solution to this problem was to simply remove the transform broadcaster in the MRPT module and let the Kalman filter handle the transforms. With a working module the last thing to do is to optimise the hyperparameters for the Graph SLAM algorithm.

5
Results

The results of this thesis are presented in this section with focus on the feasibility of solving the SLAM problem with existing and commonly distributed radar hardware. The results are divided in to three main sections attempting to give insights that can benefit over all localisation indoors using radar.

- Raw radar detection improvements
- Reflector landmark amplification
- MRPT Graph SLAM output

5.1 Raw radar detections improvements

Radars suffer greatly from the multi-path-propagation problem and radar reflections in an indoor environment like a garage. Therefore different methods for filtering and augmenting the radar detections have been presented below. DDAF effectively filters out single noise detections at the same time as it is producing more data on the desired positions. The algorithm was also tested in a noisy outdoor environment to showcase that the usability is not only restricted to noisy indoor environments.

5.1.1 LaserScan vs full range radar

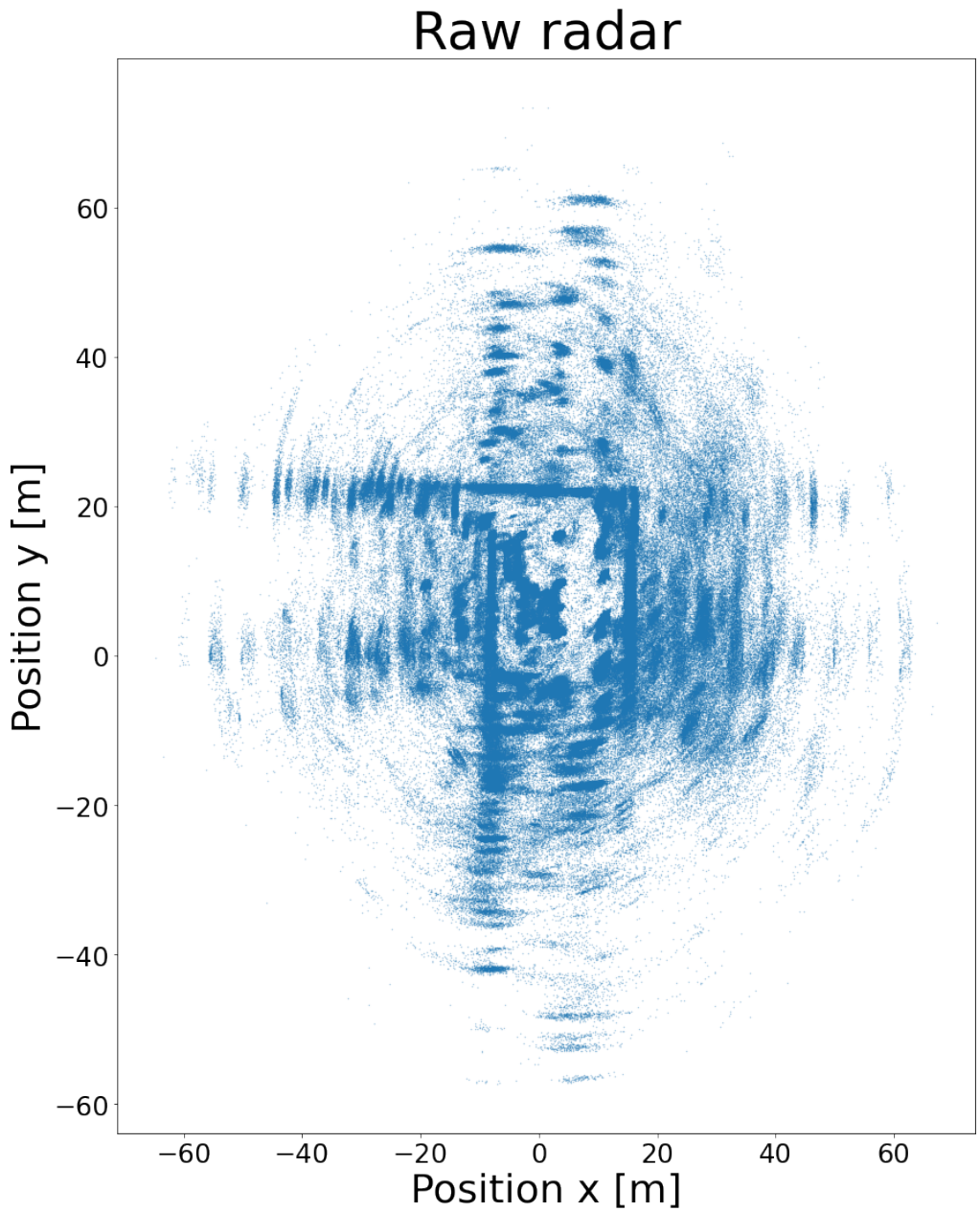
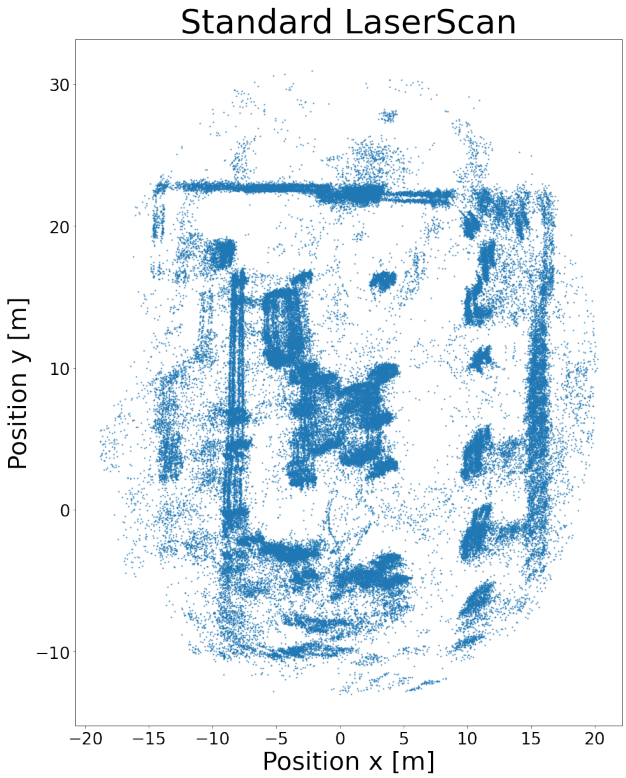
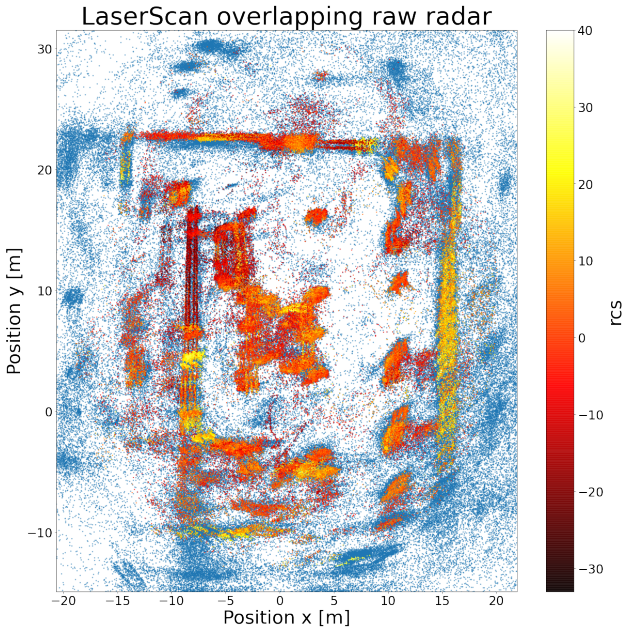


Figure 5.1: Raw radar data from five laps in the garage. There are approximately 470 000 detections in the image. These are all of the detections unfiltered and what all of the results are based on.



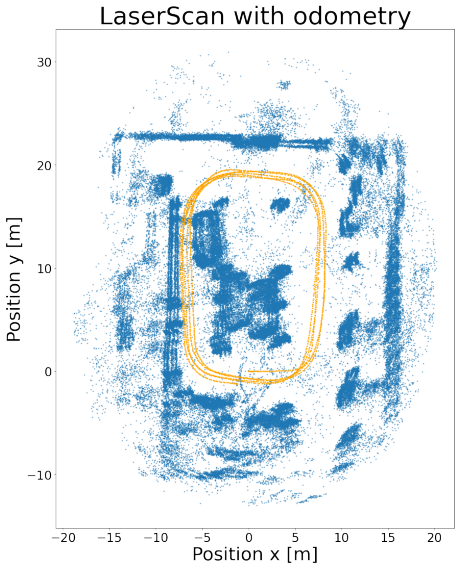
(a) LaserScan from five laps in the garage. There are approximately 120 000 detections in this image.



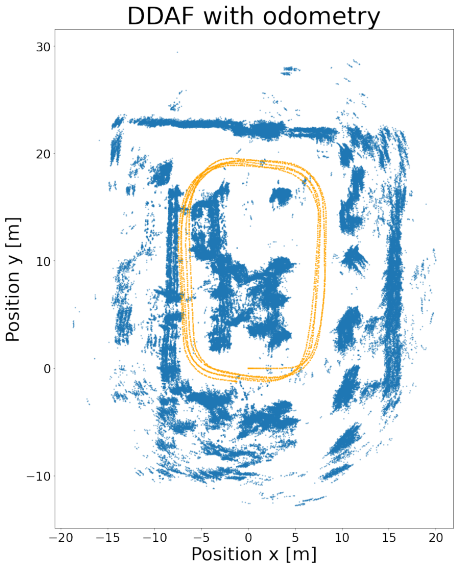
(b) Detections from LaserScan, colour coded in the RCS values, overlapping the raw radar data from Figure 5.1.

Figure 5.2: Comparison of the raw radar data and the standard LaserScan conversion.

5.1.2 LaserScan vs DDAF

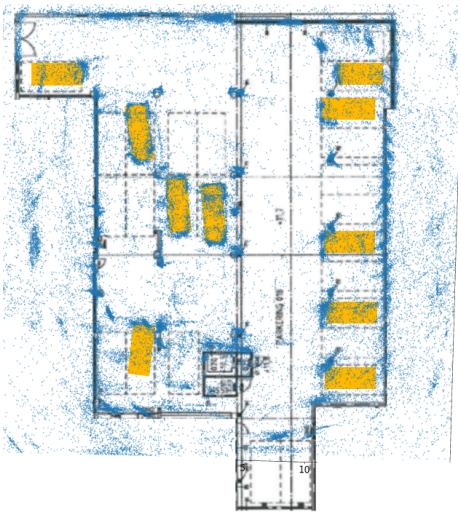


(a) LaserScan with approximately 120 000 detections.

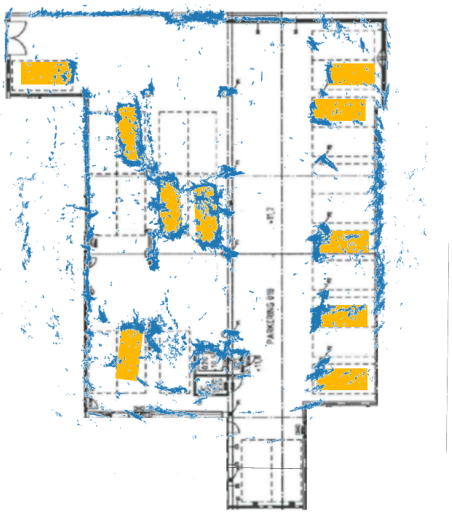


(b) DDAF with approximately 320 000 detections.

Figure 5.3: Comparison of standard LaserScan and DDAF conversion on raw radar data from five laps in the garage. As seen in the images the walls are more solid when using DDAF, we get less detections in the freespace and about 3 times more detections.



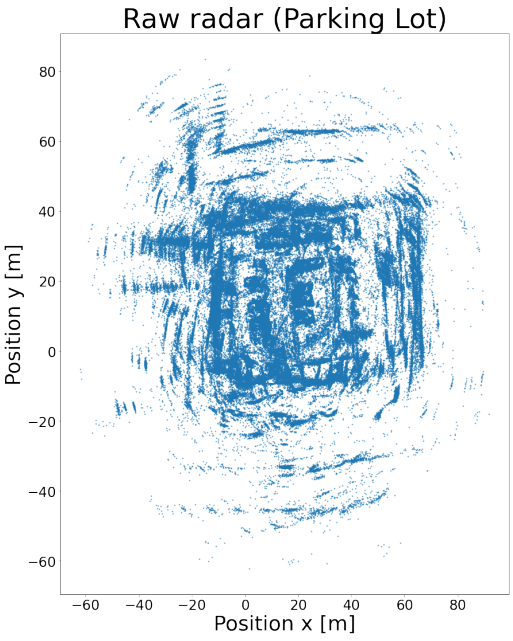
(a) An overlay of the raw radar on the garage floor plan.



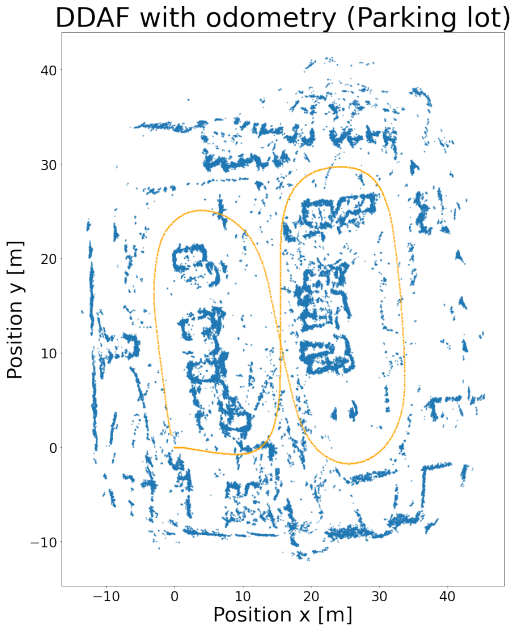
(b) An overlay of DDAF on the garage floor plan.

Figure 5.4: These two figures were chosen to showcase the contrast with DDAF and the original raw radar data for one lap around the garage.

5.1.3 DDAF outdoors



(a) Using raw radar there are 194 000 detections in the image.



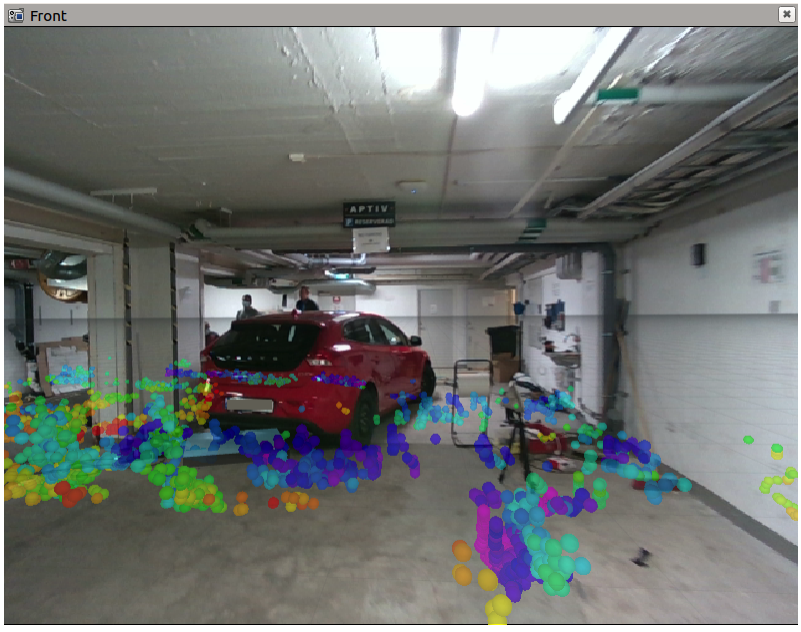
(b) Using DDAF there are 114 000 detections in the image. Trajectory from odometry in orange.

Figure 5.5: Showcasing the effect of DDAF in the outdoors parking lot after one lap. The outline of the cars are visible but the snow makes it hard to know what is solid objects in the figure.

5.2 Reflector analysis

In an attempt to improve radars detecting landmarks the use of corner reflectors have been investigated and are presented below. For corner reflectors to prove useful for indoor environments a very strong reflector amplification is recommended to give the RCS signature a clear distinction from the rest of the detections. Under these circumstances corner reflectors could possibly be used as a tool for better SLAM results. Looking at the RCS values in Figure 5.8 we can conclude that there is a lot of overlap between the RCS values of the corner reflectors and other common objects in the garage. If instead looking at the RCS cluster centroids for each object it is easier to notice that corner reflectors do have higher average RCS values than other objects.

(a) Image from data collection with an overlay of DDAF detections (reflector in northeast corner). Since the video feed and radar data are not in perfect sync we see the purple detections dots not aligned with the reflectors.



(b) Image from data collection with an overlay of DDAF detections (reflector in southeast corner).

Figure 5.6: These images showcases the two main reflectors locations analysed. In 5.6a the high RCS detections, in purple, from the corner reflectors stand out among the others. However, in 5.6b the detections from corner reflector are in a similar range as those of a car.

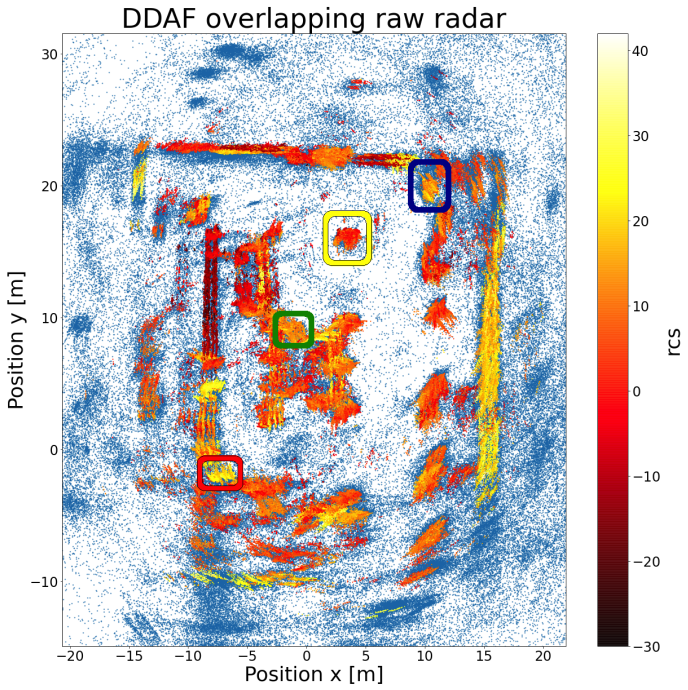


Figure 5.7: Raw radar overlapped with DDAF. The coloured squares are approximate areas of Figure 5.9 and Figure 5.10, where these regions are magnified.

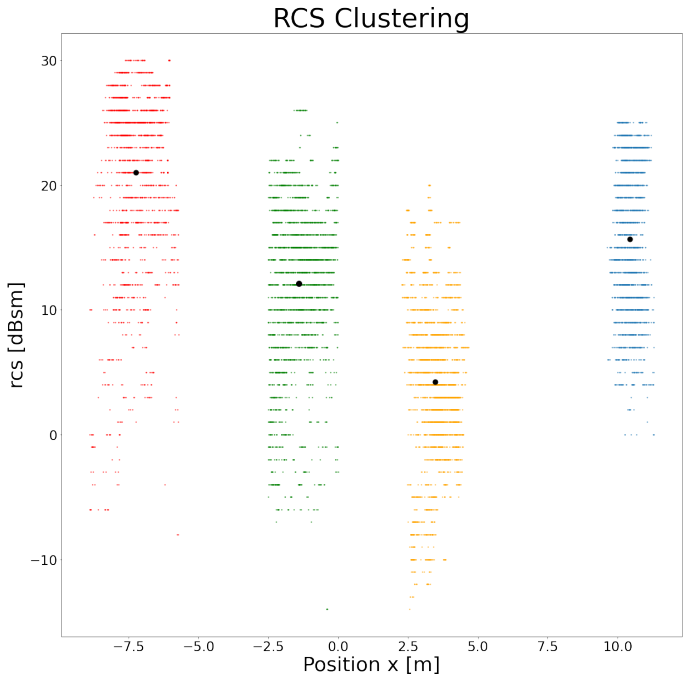
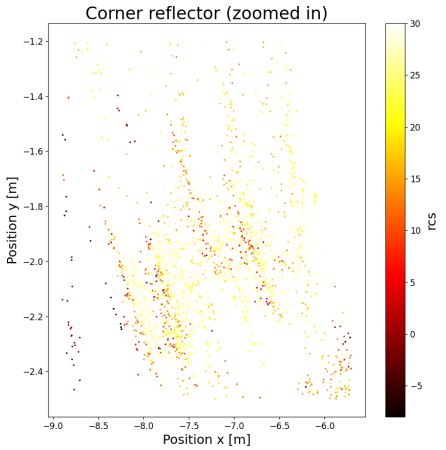
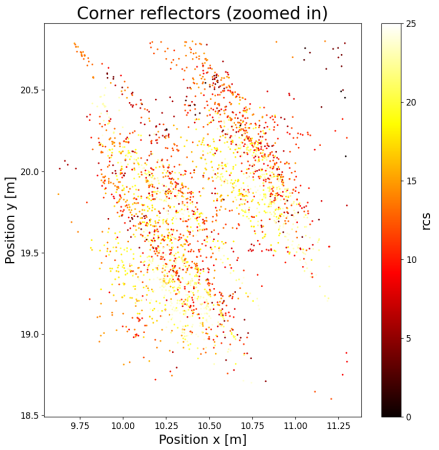


Figure 5.8: Plot of the RCS values for the four locations marked in Figure 5.7, colour coded with the same colours as the markings. The markings from left to right, are a single corner reflector, part of a car, a concrete column and two smaller corner reflectors. The mean RCS value of the corresponding areas are marked with a black dot.

5.2.1 Reflector characteristics

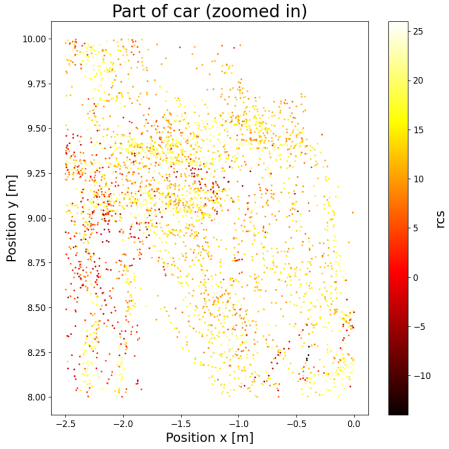


(a) Magnification of the bottom left red square in Figure 5.7. The detections represent a single corner reflector and the amount of detections shown in the figure is 2523.

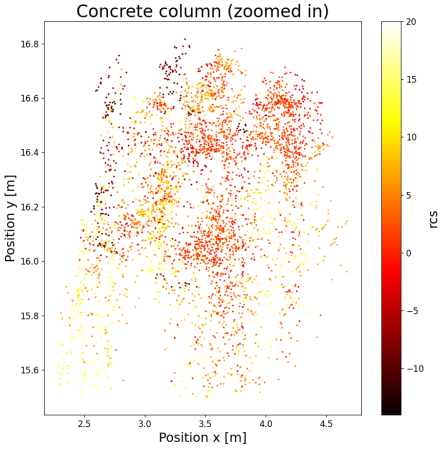


(b) Magnification of the top right blue square in Figure 5.7. The detections represent two corner reflectors and the amount of detections shown in the figure is 2877.

Figure 5.9: Closeup in the detection map of both the reflector locations.



(a) Magnification of the middle left green square in Figure 5.7. The detections represent part of a car and the amount of detections shown in the figure is 3511.



(b) Magnification of the middle right yellow square in Figure 5.7. The detections represent a concrete column and the amount of detections shown in the figure is 4265.

Figure 5.10: Close up of the detection map of a piece of a car and a concrete column.

(a) Image from data collection on outside parking lot with overlayed DDAF detections.

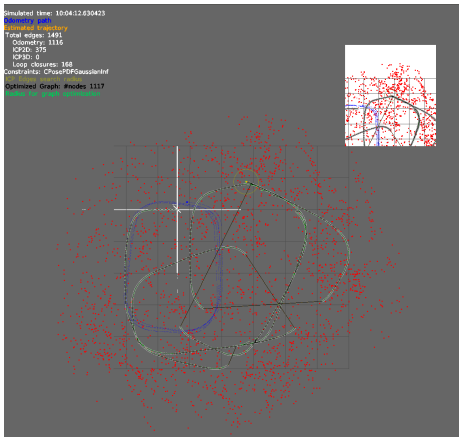
(b) Image from data collection on outside parking lot with overlayed DDAF detections.

Figure 5.11: Two images from the outside data collection using DDAF. We can distinguish the corner reflector much clearer in the overlay compared to the indoors data.

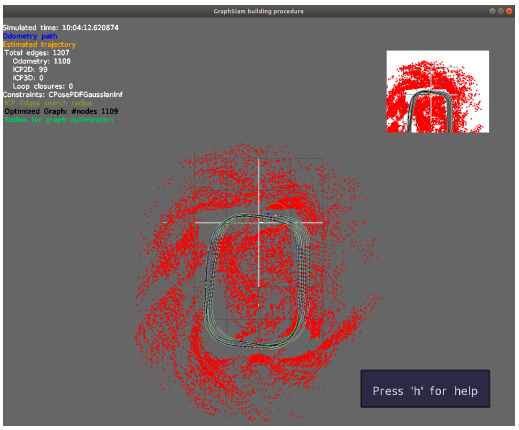
5.3 SLAM evaluation

For an algorithm to effectively solve the SLAM problem mainly two metrics are evaluated. The generated map of the environment and the estimated position, or trajectory, in that environment. The particular SLAM algorithm evaluated was MRPT Graph SLAM C++ software package ported to ROS. The results from the package was of varying nature. The trajectory generated by running MRPT Graph SLAM on the DDAF generated data was a great improvement compared to utilising the raw radar data. As seen in Figure 5.13 the trajectory SLAM output and the free space estimations looks a lot better for DDAF than it does for the standard LaserScan converter. On the other hand the map generated from the SLAM module was not comparable to the DDAF detections showed in section 5.1.2. After being able to replicate a similar figure to that of the SLAM generated map a possible explanation was found. More details in section 6.3.4.

5.3.1 Mapping



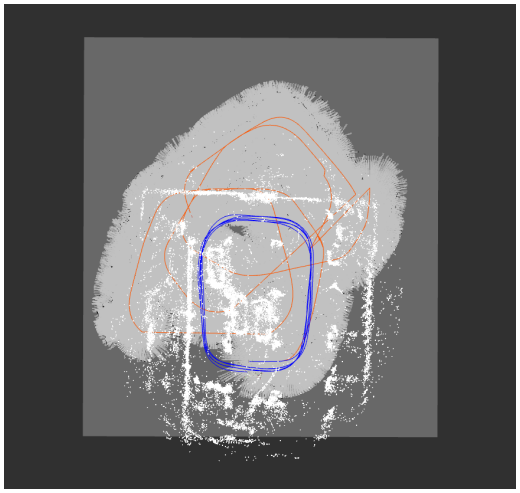
(a) MRPT visualisation software showing the generated map when using LaserScan as input.



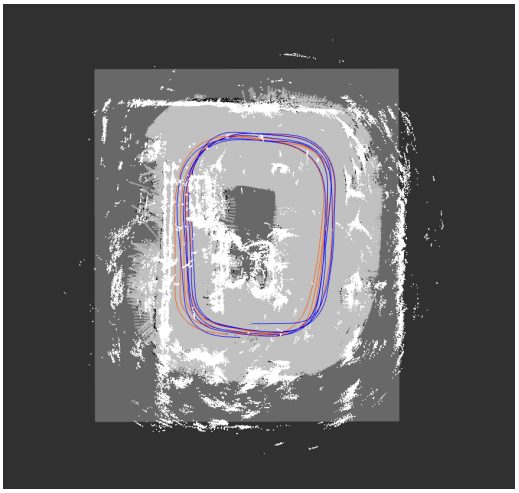
(b) MRPT visualisation software showing the generated map when using DDAF as input.

Figure 5.12: Both the inputs were unsuccessful in generating a map of the garage. Despite that DDAF did manage to consistently generate the distorted map at the same place while the standard LaserScan map was uninterpretable.

5.3.2 Navigation

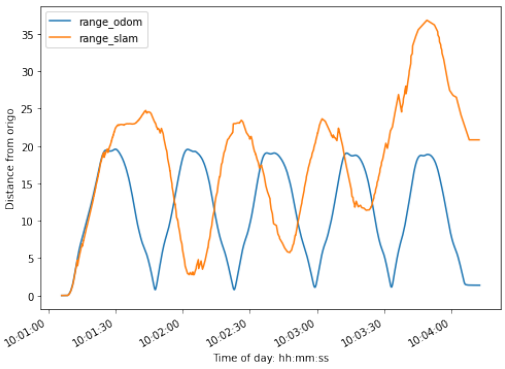


(a) Screenshot of RViz corresponding to LaserScan input and same MRPT output as in Figure 5.12a

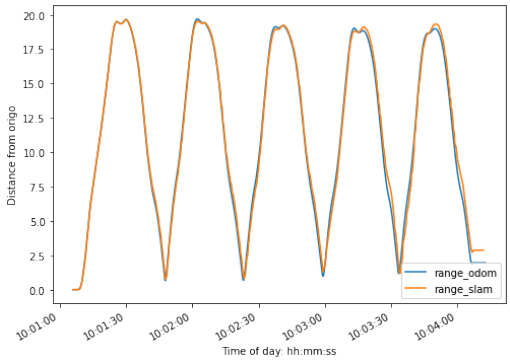


(b) Screenshot of RViz corresponding to DDAF input and same MRPT output as in Figure 5.12b

Figure 5.13: The images represent the two different MRPT SLAM outputs shown above. The blue trajectory is the odometry data, the orange trajectory is the SLAM output. The gray area is according to the SLAM output free space and the white dots are detections as seen by ROS and RViz. Comparing the trajectories and the free space there is a clear difference between the standard LaserScan converter (a) and DDAF (b).



(a) Distance from the origin of the two trajectories with LaserScan as input.



(b) Distance from the origin of the two trajectories with DDAF as input.

Figure 5.14: Two plots showcasing the differences in distance between the estimated trajectory of the two inputs. After about 30 seconds the standard LaserScan converter start to diverge from the odometry data while DDAF stick to the odometry output all five laps.

6
Discussion

6.1 Radar detection improvements

The problem with indoor radar localisation, as mentioned in the introduction and theory sections, is the fact that enclosed environments tend to increase the amount of false detections due to e.g. multipath reflections. This is especially true for environments with e.g. metal, that reflect electromagnetic waves well. Under these conditions, performing SLAM becomes increasingly difficult compared to many other environments. Therefore different methods have been developed, to prepare the raw radar data for integration with the MRPT Graph SLAM module and improving the quality of the detections from the desired objects.

6.1.1 Raw radar data

As seen in Figure 5.1 there are a lot of unwanted detections picked up by the radar detectors. Despite that, we can still make out clear contours of the garage and cars in the image. Worth mentioning is that the wall to the right in the figure is entirely made of metal, on top of that it is not completely flat but have the shape of a square wave.

Another place worth noting is the garage door which is located in top left of the figure. Both these places seem to have a lot of concentrated detections with periodical circular arcs behind the actual metal objects. We think that these kinds of detection patterns originates from the multipath problem. Since the garage door is flat it makes for a perfect reflection which in turn creates this periodical pattern.

In addition to that the multipath effect "smears out" the detections in the polar ϕ coordinate and might be the reason why lines behind the garage door is not flat like the door itself. The result of this behaviour is what we see in the figure, periodically spaced detection patterns of circular arcs behind the metal object we are trying to observe. We think that this behaviour of aggregated clusters of detections proves to be very problematic for the SLAM algorithm to handle. One very important note to make about this phenomena is that these clusters of false detections only occur behind the object we are trying to detect. This fact further enhances the idea of these particular detections have reflected between, for instance, the host vehicle and the object several times, since the added time of propagation with each reflection will make the detector think it is further away.

In addition to the shape of the false detections we can also take note of the range in which we find them. It is clear that detections further away than the size of the garage is of no use. Therefore doing something simple, like rejecting detections

at a set range, would most probably help navigation in indoor environments. One possibility could be to use some other technology to determine if we are inside or outside and with that information automatically set the max range of the detectors.

6.1.2 Standard LaserScan conversion

At first glance the standard LaserScan conversion works surprisingly well. In Figure 5.2a we can see that the majority of the noisy detections are gone, both inside the garage and outside. We can see the difference more clearly in Figure 5.2b with the LaserScan data, colour coded in RCS values, overlaying the raw radar data.

To understand the reason why the difference is so large one have to take into account the fact that the max range for LaserScan message is set to 12 meters. This means that detections on the other side of the garage, compared to the side of the host vehicle, are not counted which they are in the raw radar data since the range covers the whole garage at all times. This max range limitation improves detections quality drastically when it comes to not counting false detections but it also comes with a cost of a 3 times lower detection count. One thing that can not be seen in the aggregated data in the figures is how the detections look like inside the LaserScan message. Since the circle segment angle is set to $\alpha = 2°$ it means that the LaserScan message is divided into 180 circle segments. For a LaserScan message to represent something like a straight line, you want to have some segment in the message were all indices have relatively similar range values. An example of this can be found in Table 6.1. This would be equivalent to detecting something, like part of a straight wall.

The problem we see inside the standard LaserScan message is that it is sporadically filled with either gaps, between the indices, or with range values that are drastically different, see Table 6.2. This means that the message is representing "holes" in the object or that part of the object is detected far behind the real wall. We think that this is the reason why the MRPT Graph SLAM module made poor trajectory estimations.

Table 6.1: Example of how a straight line to the left of the host vehicle looks like inside the LaserScan message. Infinity signals a circle sector with no detections.

ϕ	**...**	**160°**	**158°**	**156°**	**154°**	**152°**	**150°**	**148°**	**146°**	**144°**	**...**
Range	∞	6.53	6.45	6.48	6.53	6.57	6.57	6.53	6.48	6.45	∞

Table 6.2: Example of what we see when converting the radar data with the standard LaserScan message conversion. Infinity signals a circle sector with no detections.

ϕ	**...**	**160°**	**158°**	**156°**	**154°**	**152°**	**150°**	**148°**	**146°**	**144°**	**...**
Range	∞	6.34	6.45	∞	∞	6.57	6.57	10.53	6.48	11.45	∞

6.1.3 DDAF conversion

Examining Figure 5.3a and 5.3b the general outlining of the garage looks very similar. We can distinguish the walls of the garage as well as the cars in the middle fairly well and the path of the host vehicle seems realistic. The largest difference between the figures is the amount of single detections present in the standard LaserScan figure compared to the DDAF figure. There are still spots where we find undesired noise, despite the fact that DDAF has the parameter LNIB set to 2, but compared to the standard LaserScan message they cover less area in the figure. One other important difference that is harder to notice in the figures is the detection density of the clusters and lines in the garage. For the DDAF conversion we noticed a much more dense pattern than in the LaserScan message. Despite that the algorithm still suffers from occasional detections behind the wall which is undesirable.

Looking at Figure 5.4a, with the DDAF detections overlapping the garage drawings, it seems like the proportions of the detections are well in line with the drawing. This is data from one lap around the garage with the time span being approximately 50 seconds. One of the more striking observations of the DDAF algorithm is the fact that it produces about the same amount of radar detections as the raw radar data. This is because the incoming data to DDAF is all the detections in the buffer and not only the detections from latest raw radar message. So in the process of filling the LaserScan messages, DDAF has a lot more detections to choose from compared to the standard LaserScan converter.

A plot of how the buffer size effects the amount of detections produced was shown in section 4.4.2 together with the values of the other hyperparameters. This feature of both augmenting and filtering data at the same time based on the most dense region of the circle segment seems to create a better foundation for solving the SLAM problem and we think that this is the main reason behind the results shown in section 5.3.

In addition to testing the performance of DDAF indoors we also tried it out outdoors. One question that came up was whether or not the algorithm would reduce performance, compared to the raw radar data, when not used in an enclosed environment. Since it was winter time and snow was present during the data collection a lot of noisy detections were observed. Replaying the rosbag with the raw radar data together with video feed it was very clear that a lot of the detections came from snow. Even though the raw radar data was very noisy DDAF managed to perform well and we can see clear outlines of the cars in the middle of the parking lot. The detections of objects, any object that was not a car, was much harder to differentiate since we can not know by looking at Figure 5.5b if it was a solid object or a bank of snow. In general we draw the conclusion that DDAF does not necessarily only produce good results in an indoor environment but could see benefits being used outside as well.

6.2 Corner reflectors

Corner reflectors are a known tool for amplifying radar signals and the question we asked ourselves was if this simple tool could make SLAM work better in an indoor

environment. The question turned out to be harder to answer than we thought. Radar reflectors amplify the signal in proportion to the area of the cone . In our data collection setup we used different kinds of reflectors because we thought this would result in more interesting data to analyse. In reality this turned out to be a mistake since it only made it harder to distinguish all the reflectors at once. In general we think that reflectors could be used to improve SLAM algorithms, via for instance investigating EKF SLAM which utilises specified landmarks, but for our case it is yet undetermined if the reflectors made a difference for the Graph SLAM algorithm to converge.

6.2.1 Distinguish reflectors via RCS

One of the properties we decided to analyse was the RCS values of the corner reflectors compared to other common objects we could find in the garage. Looking at Figure 5.7 we can see four highlighted areas. The red square in the bottom left is a corner reflector, the green square in the middle is a part of a car, the yellow square in the top is a concrete column and the blue square at top right is a pair of corner reflectors. Images of these areas, magnified, can be seen in Figure 5.9 and Figure 5.10.

If we keep the relative x position of the four object the same but use the y-axis for the RCS values we see what is shown in Figure 5.8. The original thought was that the RCS values of the radar reflector might be high enough to differentiate the reflectors by filtering only on the RCS values. Best case scenario would be if the difference in RCS would be so high that the clusters we see in Figure 5.8 can be separated by setting a certain threshold for the RCS values.

As we can see in the figure this is not possible. The clusters have a more widespread range than anticipated. This indicates that simple filtering based on only RCS would most probably not be a optimal way of finding corner reflectors or other objects in the garage. If one were to instead look at the mean of the clusters in the RCS direction they are more easily identified. We would be able to draw a distinct line between the two objects and the corner reflectors.

One other more advanced approach could be to cluster all the data, with some method like k-means or DBSCAN, in 3-dimensional space were the RCS value act in the z-direction and then do the filtering on all the clusters based on the centroids. This approach is outside the scope of this thesis since the implementation of such filtering would not realistically be input data for a SLAM algorithm but might be useful in other contexts.

6.2.2 SLAM with and without reflectors

The MRPT Graph SLAM module was tested extensively on the data with and without the radar reflectors. Since the mapping of the garage with the module was unsuccessful, more details on this in section 6.3, it was hard to quantify how much of a difference the radar reflectors made.

The original idea was that inside a noisy environment they could work as a landmark with detections of high confidence. The reflector did indeed give detections

of high confidence but as discussed in the section above the difference in RCS, for our specific corner reflectors, was not high enough to use simple filtering techniques to extract theses landmarks. In addition to that the number of detections was not either of high enough count, or density, to easily identify the reflectors in the data. The specific SLAM algorithm used was, as mentioned, Graph SLAM and the hope was that the Graph SLAM mapping, see theory section 3.3, of the algorithm would have an easier time locating the reflectors compared to other objects in the garage. This might still be the case since we did not come to any definitive conclusion but we think there might be other SLAM algorithms like EKF SLAM that could utilise corner reflectors better since this version of SLAM requires landmarks to operate.

It is also possible that the benefits of using these specific radar reflectors was small in comparison to the efficiency of DDAF on other object like walls and cars. We still believe that radar reflectors could be useful in indoor environments but that further testing needs to be done to make conclusions on that statement.

6.3 Radar indoor SLAM

As already stated solving the problem of indoor SLAM with common radar technology is a really hard problem. The following section provides a discussion about what angles of attack that is best based on the results presented in section 5.3. Judging by the map of the garage produced by the MRPT Graph SLAM module we did not succeed in sufficiently solving the indoor radar SLAM problem. Our goal is to by the end of this discussion present a path from where the project is today to how it can fully solve the radar indoor SLAM problem. We start by asking the question what input data is required for the SLAM algorithm to converge in the first place?

6.3.1 DDAF

Since the raw radar data clearly was not the answer to the question asked above, the next natural step was to try filtering the detections in some way. The simplest way of doing so was to set a max range for which the detections were valid. This worked surprisingly well, more details in the section 6.3.3 below, but going back to the question about SLAM convergence it was also clear that it was not enough. Because of the format of ROS' LaserScan message all the detections had to be divided into different circle segments. The standard way of converting radar data to LaserScan is simply to take the detections in each segment that had the lowest range value. What we realised was that in the radial direction of each circle segment you can plot the detection density for that segment by counting the detections per range unit. The only issue is that for any given raw radar detection message we will only find a hand full of detections.

For the detections density to make any sense, the raw radar messages had to be aggregated over some time. To make it easy to program we divided each circle segmenting in the radial direction into a finite number of bins dependent on the max range. Now all aggregated detections belongs to a specific bin, which is easy to count, calculate average range over and normalise against the area representing the

bin. Some details of coordinate system transforms are left out in this section but the details on those can be found in sections 4.4.2.

This algorithm effectively creates a circular "detections density map" with radius equal to max range and density in proportion to the aggregated time. One way of visualising this is to imagine a circle, divided into segments and bins like described above, and to let the count of each bin be a column in the z-direction creating a histogram for each circle segment. Normalise each bin count by the corresponding area and the histograms turns into a discrete density map. The last step is to choose which column in each circle segment you want to use in the LaserScan message, since you can only choose one, DDAF uses the highest column corresponding to the highest density.

This selection can be done in other ways but using Occam's razor the column representing the highest density became the most natural choice. After the selection is done it practically collapses the density map into range values that will represent the radar detections for that unit of time. This makes it possible to publish DDAF LaserScan messages at the same frequency as the original radar data, except for the initial time it takes to fill the buffer.

6.3.2 DDAF Hyperparameter space

With the discrete density map in mind we can start to understand how the hyperparameters affect the algorithms ability to augment and filter detections at the same time. In summary we can put each hyperparameter in the category of filtering, augmenting or doing both. By going through each parameter one by one, and imagining what happens to the density map, we can take steps towards making it produce something that will better represent reality and thus benefit any SLAM algorithm.

Starting with the max range it might seem like it does the same thing for DDAF as it does for the standard LaserScan conversion but there is one major difference. When choosing the columns representing the highest density the max range plays a large roll since the largest density not always corresponds to the column best representing reality. One very good example of that can be seen in Figure 5.1, with raw radar data, since a lot of the times dense clusters are forming periodically behind metal objects in the garage. By changing the max range to some reasonable value we are also making sure to never calculate the density of columns beyond the max range and thus efficiently filter out most the detections outside of the garage.

The next hyperparameter, the angle α, had the surprising effect of filtering or augmenting data, depending on if you are increasing or decreasing α. One example of augmentation would be to divide α in half, which would created double amounts of circle segments thus also increasing the resolution of the density map by the double. As long as the buffer size is large enough to still fill most of the new bins this change in α would almost double the amount of detections generated but at the same time use less detections calculating the average of each bin. The same argument can be done for doubling α but will produce the results in reverse. Doubling α will generate about half as much data, thus acting as a filter, but at the same time use double the amount of detections for calculating the average of each bin. So α can

be summarised as a hyperparameter balancing resolution against accuracy.

The third hyperparameter is the buffer size. The height of the columns in the density map will be roughly in proportion to the amount of aggregated messages in the buffer. When choosing the buffer size it is important to use a large enough buffer to make sure that the columns representing the highest density is larger than the columns representing noise or unwanted reflections but at the same time keeping it as small as possible to reduce computational cost. Since it takes a finite number of range values to fill the LaserScan message there will also be a ceiling for how many detections that can be produced by the algorithm. Increasing the buffer size after that ceiling will only add computational cost to the algorithm and not produce any benefits. For each set of hyperparameter there is a sweet spot for the buffer size were it produces the desired results at the lowest computational cost.

The last hyperparameter is LNIB and is acting like a threshold filter for the density function. If the bin with the highest density does not at least have above some count value it will not be selected. We can visualise this with the density map by just deleting the columns that are not high enough across the whole map. This is the simplest hyperparameters to understand and also most effective for filtering data like single detections. The main idea for this discussion section was to explore this hyperparameter space using SLAM as evaluation metric. Unfortunately the implemented SLAM module underperformed compared to our expectations and making conclusions about optimal hyperparameters was unrealistic without any performance metric.

6.3.3 DDAF vs standard LaserScan

When evaluating SLAM results there are two categories that are of most interest, which is the mapping and the navigation results. As previously mentioned the mapping results from the algorithm was unsuccessful but the difference between the different inputs can still be partially evaluated. Looking at the results in Figure 5.12a, with the standard LaserScan conversion as input, in comparison with Figure 5.12b, with DDAF as input, we can clearly see that the map produced by DDAF is something resembling a square while the standard conversion map is not resembling anything.

By itself this might not say much since what we see in both maps is not what we expect but if we combine these maps together with the trajectory data it does tell us something about the different inputs. Looking at the difference in the trajectory data in Figure 5.14a and Figure 5.14b we can clearly see that DDAF does give a trajectory close to the odometer output, while the standard LaserScan conversion does diverge drastically. We would argue that what makes the DDAF trajectory similar to the odometry output is the fact that the map being produced might be distorted but consistently distorted for each lap. We have an idea of what makes the output distorted and is presented in section 6.3.4. These results are indicating that DDAF produces more consistent input values compared to the standard LaserScan conversion and could be the better alternative of the two.

6.3.4 MRPT Graph SLAM module bug

A map similar to the MRPT Graph SLAM map was produced by trying to manually convert the DDAF LaserScan message back to a PointCloud message and transform it from the VCS frame back to the map frame. The way this was done was to simply extract the range values and azimuth angles and convert them to Cartesian coordinates via,

$$x = r * \cos(\phi),$$
$$y = r * \sin(\phi),$$

where r is the range value and ϕ is the azimuth angle. After all the detections from LaserScan message was converted, they all got transformed via ROS transform functions from the VCS frame to the map frame. The results of this method are displayed in Figure 6.1a.

For reference the resulting MRPT Graph SLAM mapping is also provided in Figure 6.1b. This approach is the simplest way of converting a LaserScan message to Cartesian coordinates but does not take into account the fact that each detections in the messages was in reality registered at different times as the host vehicle was moving. To compensate for this a time is usually encoded inside the LaserScan message within a variable *time_increment*. Looking inside the official documentation for ROS LaserScan message we find the comment.

```
# File sensor_msgs/LaserScan.msg
...
float32 time_increment   # time between measurements [seconds]
                         # - if your scanner is moving, this will
                         # be used in interpolating position of 3d
   points
...
```

Listing 6.1: Part of the official LaserScan message definition.

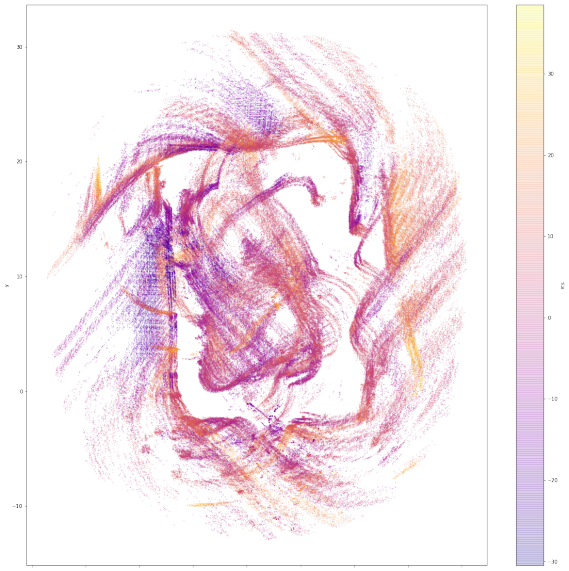
After digging through the source code of the MRPT package we found that inside their sub-module mrpt_bridge they never utilise any time variables, other than the timestamp, when converting the ROS LaserScan message to the MRPT equivalent C++ object. We also found that when they are converting back the MRPT object to ROS LaserScan format, for visualisation purposes, we can see them hard-coding the variable to zero.

```
// File mrpt_bridge/src/laser_scan.cpp
...
// setting the following values to zero solves a rviz visualization
    problem
  _msg.time_increment = 0.0;  // 1./30.; // Anything better?
  _msg.scan_time = 0.0;  // _msg.time_increment; // idem?
...
```

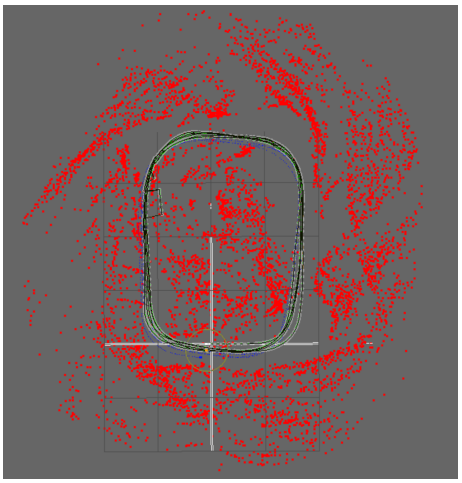
Listing 6.2: Part of laser_scan.cpp file showing hard-coded time variables.

This is to us a very clear indication that the MRPT Graph SLAM module does in fact not take any time variables into account when producing the transforms for the detections. We believe strongly that this is the reason for why the mapping

output of the module is unsuccessful and we will in the following sub-section 6.3.5 describe how the problem can be fixed.



(a) Recreation of error in python, colour gradient coded to the RCS values.



(b) Detection pattern using MRPT.

Figure 6.1: Showing similarities between the output from the SLAM module and the recreated behaviour plotted in in python.

6.3.5 MRPT Graph SLAM bug solution

The root problem with the MRPT Graph SLAM implementation stems from the assumption of the robot, or in our case a vehicle, having a fairly small velocity and yaw rate. For small velocities and yaw rates all the detections in the LaserScan message can be transformed with the same transform and still yield good enough results that the location of the detections can be used with Graph SLAM algorithm successfully. However, if the velocity and yaw rate is large enough each detection in the message will need to have its own transform corresponding to the place of the vehicle at the time of the detection. There is a point where the error from using the same transform on all the detections is to large and the Graph SLAM algorithm can not compensate for the robot/vehicle moving.

To account for this error we have to interpolate the transformations between the original timestamp of the LaserScan message and the total time it takes to preform one whole scan. In our case we do not have a scan time since the hardware we are using is not a laser. In practice the effect of aggregating the detections from all four radar units produce a similar behaviour to a laser sweep and thus this interpolation is still needed for our hardware setup as well. One example on how this interpolation can be implemented is described in the python-styled pseudocode below.

```
def interpolate_laserscan_message(ls_msg, tf_listener):

    # Getting start and end time for the interpolation
    start_time = ls_msg.timestamp
    end_time = ls_msg.timestmap +
               ls_msg.time_increment * len(ls_msg.ranges)

    # Getting start and end transforms for the interpolation
    start_transform = tf_listener(start_time)
    end_transform = tf_listener(end_time)

    # Empty transform object
    current_transform = tf.new_transform()

    # Empty list to save the result
    points_transformed = []

    for index, point in enumerate(ls_msg):
        # Get the start point coordinates and quaternion
        start_point, start_q = transform(start_transform, point)
        # Get the end point coordinates and quaternion
        end_point, end_q = transform(end_transform, point)

        # Calculate interpolation ratio
        ratio = get_ratio(start_transform, end_transform, index)

        # Interpolation of the translation
        current_transform.origin =
            interpolate_trans(start_point, end_point, ratio)
        # Interpolation of the rotation
        # Using Spherical Linear Interpolation (SLERP)
        current_transform.rotation =
            interpolate_slerp(start_q, end_q, ratio)

        # Transform the point with the new transform
        point_transformed = current_transform * point
        points_transformed.append(point_transformed)

    return points_transformed
```

Listing 6.3: This psudo code omits a of technical details and its only purpose is to highlight how this interpolation can be done with the tools provided by the ROS transform modules.

In the case of actually correcting the interpolation bug in the existing code, the implementation details is much more complicated. Since the MRPT Graph SLAM module is a software package stand alone from ROS, the calculations done in this package wont have access to the transforms and interpolation functions provided by ROS. Therefore the solution for this particular package will be to extend the source code to either A: implement the same functionality as ROS provides from scratch, or B: extend the mrpt_bridge functionality to also include ROS transforms and send the interpolated transforms together with the LaserScan message. Both of these solution will also have to make changes at the location where the detection transformations are calculated. To execute any of the mentioned solutions to this

problem is outside the scope of this thesis and the details are provided to Aptiv for further development of the project.

7
Conclusion

The objective of this thesis was to investigate and analyse the feasibility of using common automotive radar technology for indoor localisation. The question if it is possible to fully solve the indoor SLAM problem was investigated, given the environmental and hardware constraints. We also investigated if tools like corner reflectors could help in the localisation processes. The conclusions of the thesis comes with varying results. Regarding reflectors we managed to conclude that simple filters on RCS was not good enough for high confidence localisation. The RCS values from reflectors tends to mix in with other objects like cars and makes it hard to distinguish solely on the RCS. Despite that we think that using reflector with very high RCS could be useful as landmarks and, possibly, also benefit SLAM algorithms but to confirm this more testing is needed. To solve the SLAM problem, we first had to process the noise out of the radar data. The solution to this problem became our own algorithm we called Detection Density Augmentation Filter (DDAF) which successfully managed to both filter unwanted detections and at the same time increase the density of wanted detections. Using DDAF as input to the MRPT Graph SLAM module we got mixed results. Even though the mapping produced by the module was unsuccessful, it was consistently generating the same distorted map. This in turn seemed to be enough for the module to produce a decent estimation of the trajectory. These results indicate that DDAF might be a suitable solution for radar SLAM pre-processing but without a fully functioning module we can not validate the confidence of our conclusion. The distortions in the SLAM generated map was manually reproduced and the issue could be traced back to the MRPT source code not accounting for vehicle velocity when computing transforms in between vehicle positions. The solution to the problem was presented in text but executing the required changes lies outside the scope of this thesis. Our main conclusion is that solving the indoor radar SLAM problem, using DDAF as pre-processing, looks promising but needs a fully functioning SLAM implementation to be evaluated and tested to make any definite statements.

Bibliography

[1] Google Maps, 2021. [Online]. Available: https://www.google.se/maps/@57.68827,11.9967582,105m/data=!3m1!1e3

[2] M. Maurer, J. Gerdes, B. Lenz, and H. Winner, *Autonomous Driving: Technical, Legal and Social Aspects*. Springer Berlin Heidelberg, 2016. [Online]. Available: https://books.google.se/books?id=HdtCDwAAQBAJ

[3] K. Gammon. Future past: Self-driving cars have actually been around for a while. (accessed: 2021-03-22). [Online]. Available: https://www.caranddriver.com/news/a15343941/future-past-self-driving-cars-have-actually-been-around-for-a-while/

[4] Aptiv. What are the levels of automated driving? (accessed: 2021-03-21). [Online]. Available: https://origin.aptiv.com/en/insights/article/what-are-the-levels-of-automated-driving

[5] Synopsis. The 6 levels of vehicle autonomy explained. (accessed: 2021-03-22). [Online]. Available: https://www.synopsys.com/automotive/autonomous-driving-levels.html

[6] NHTSA. Crash factors in intersection-related crashes: An on-scene perspective. (accessed: 2021-03-22). [Online]. Available: http://www-nrd.nhtsa.dot.gov/Pubs/811366.pdf

[7] NHTSA. Distracted driving 2014. (accessed: 2021-03-22). [Online]. Available: http://www-nrd.nhtsa.dot.gov/Pubs/812260.pdf

[8] A. Kleiner and C. Dornhege, "Mapping for the support of first responders in critical domains," *Journal of Intelligent and Robotic Systems*, vol. 64, pp. 7–31, 10 2011, (accessed: 2021-03-21). [Online]. Available: https://www.researchgate.net/publication/220061523_Mapping_for_the_Support_of_First_Responders_in_Critical_Domains

[9] ElectronicsNotes. Multipath propagation. (accessed: 2021-03-25). [Online]. Available: https://www.electronics-notes.com/articles/antennas-propagation/

propagation-overview/multipath-propagation.php

[10] ROS. History. (accessed: 2021-05-21). [Online]. Available: https://www.ros.org/history/

[11] Britannica Academic. Radar. (accessed: 2021-05-13). [Online]. Available: https://academic-eb-com.eu1.proxy.openathens.net/levels/collegiate/article/radar/109463

[12] W. Melvin and J. Scheer, *Principles of Modern Radar: Vol. 3.* SciTech Publishing, 2014.

[13] ElectronicsNotes. Rf thermal noise: Johnson-nyquist noise. (accessed: 2021-06-01). [Online]. Available: https://www.electronics-notes.com/articles/basic_concepts/electronic-rf-noise/thermal-johnson-nyquist-basics.php

[14] L. Brennan, "Angular accuracy of a phased array radar," *IRE Transactions on Antennas and Propagation*, vol. 9, no. 3, pp. 268–275, 1961.

[15] M. Lundgren, "Bayesian filtering for automotive application phd thesis," 2015.

[16] W. L. Brogan, *Modern Control Theory (3rd ed.).* Prentice-Hall Inc, 1991, p. 173.

[17] R. B. R. Kalman., "New results in linear filtering and prediction theory," *J. Basic Eng 83(1), 95-108 (Mar 01, 1961)*, vol. 2, p. pp. 95–108., 1961.

[18] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics.* MIT Press, 2005, p. 41-42.

[19] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, pp. 31–43, 2010.

[20] E. Olson, "Recognizing places using spectrally clustered local matches," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1157–1172, 2009, inside Data Association. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889009001018

[21] R. Lindholm and C.-J. Pålsson, "Simultaneous localisation and mapping," 2015.

[22] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard, "Robust map optimization using dynamic covariance scaling," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 62–69.

[23] ROS. About ros. (accessed: 2021-04-30). [Online]. Available: https://www.ros.org/about-ros/

[24] MRPT. Empowering c++ development in robotics. (accessed: 2021-05-24). [Online]. Available: https://www.mrpt.org/

[25] The pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134