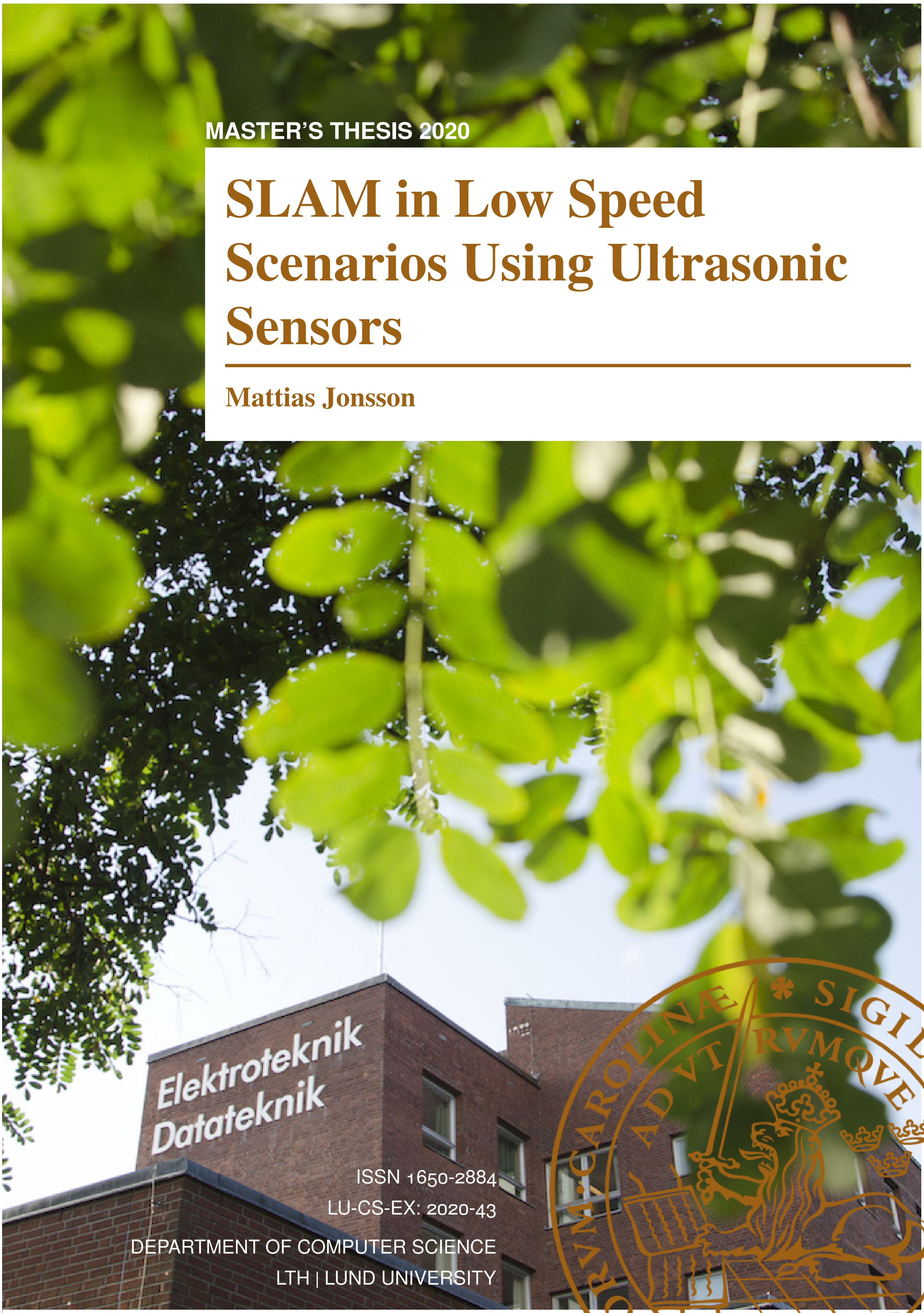

MASTER'S THESIS 2020

# SLAM in Low Speed Scenarios Using Ultrasonic Sensors

Mattias Jonsson

ISSN 1650-2884
LU-CS-EX: 2020-43

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-43

# SLAM in Low Speed Scenarios Using Ultrasonic Sensors

**Mattias Jonsson**

# SLAM in Low Speed Scenarios Using Ultrasonic Sensors

Mattias Jonsson
dic15mjo@student.lth.se

June 17, 2020

Master's thesis work carried out at Volvo Cars.

Supervisors: Elin Anna Topp, elin anna.topp@cs.lth.se
Magnus Wendt, magnus.wendt@volvocars.com

Examiner: Jacek Malec, jacek.malec@cs.lth.se

## Abstract

This master's thesis investigates how ultrasonic sensors on cars can be used to assist a driver in low speed parking scenarios. A simple odometry model based on rear wheel rotation sensors is implemented. Using this model to approximate the car's position, a Bayesian map of the environment is built from ultrasonic sensor data. Based on the results from this, the possibility of using SLAM with data from ultrasonic sensors is investigated.

The simple odometry model is sufficiently accurate for the use cases in this thesis. The map built from ultrasonic sensor data is accurate enough for aiding the driver in some not too tight low speed parking scenarios. SLAM using ultrasonic sensor data is found to be feasible given a known simple structure of the environment according to which the sensor readings can be interpreted before being used as input to the SLAM algorithm.

**Keywords**: SLAM, ultrasonic sensors, occupancy grid, robot odometry

# Acknowledgements

I would like to thank Magnus Wendt, Daniel Larsson and Amer Mustajbasic at Volvo Cars for their help and feedback toughout the thesis.

I would also like to thank Axel Pandolfi Elmi and Ricardo Lucas, who also wrote their thesises at the Volvo Cars office in Lund at the same time as I did, for great cooperation.

Finally I want to thank my supervisor at Lund University, Elin Anna Topp.

# Contents

# Chapter 1
# Introduction

The whole automotive industry is going through a revolutionizing digital transformation. Volvo Cars is moving away from sourcing complete functions, hardware and software, that gets integrated into the car. This is normally a waste of resources since that hard- and software is only utilized by one function. The next generation of Volvo's platform will have a centralized architecture where sensors will be shared among all applications. This brings a lot of new possibilities in terms of vision and perception capabilities. In the R&D team within Autonomous Drive in the VCC Lund office Volvo wants to utilize the available sensor set as much as possible to develop low speed maneuvering assisting features.

One of the most important sensor modalities is the ultrasonic sensors. It has traditionally been used to inform and warn the user when the car closes in on an object.

We want to explore the possibility to extend the usage of the sensor to also perform a rough simultaneous localization and mapping; SLAM; the problem of creating a map of an environment using sensors on a robot while at the same time localizing this robot within the map.

## 1.1 Thesis Scope

The goal of this thesis is to investigate how ultrasonic sensors on cars can be used to assist a driver in low speed parking scenarios. In particular, we investigate the possibility to use SLAM in conjunction with the ultrasonic sensors to create a map of the car's surrounding, while at the same time localizing the car within this map. As of today, most state-of-the-art solutions to a similar problem utilize lidar sensors or cameras. These sensors produce a higher quality scan of the environment in most cases. Ultrasonic sensors, while being considerably less accurate, are cheaper and are already mounted on most new cars.

We focus on the scenario of a parking lot or a parking garage, where the goal is to map the environment and estimate the car position with sufficient accuracy to find a free parking lot and to navigate the car into or out of it without driving into any obstacles.

## 1.2 Research Questions

The work is divided into four general problems. We use the wheel rotation sensors to build an odometry model; a model that estimates the position of the car over time. Using this odometry model, we utilize the light radar (lidar) sensor to implement a state-of-the-art SLAM algorithm to be used as a baseline for evaluating further results with the ultrasonic sensors. After this, we use the ultrasonic sensors to build and maintain an occupancy grid of the environment and evaluate its quality. Finally, we reason about the possibility of using the ultrasonic sensors as input for a SLAM algorithm, based on the results of both the odometry model and the occupancy grid.

- How can we create an odometry model using the car's existing sensors?
- How well does state-of-the-art SLAM algorithms with lidar work in our use case?
- Is it possible to create an occupancy grid sufficient for low speed parking scenarios using the ultrasonic sensors?
- Is it possible to construct a model of the environment sufficient for performing SLAM using the ultrasonic sensors?

## 1.3 Limitations

We limit the scope of the project to a static environment without moving objects such as other cars, pedestrians, etc. We also assume optimal conditions for the sensors, as to be able to focus more on the SLAM model than on data formatting. Both the odometry, the mapping and the SLAM is limited to a 2D plane with no regards to height.

## 1.4 Related Work

In [12], an occupancy grid is created using ultrasonic sensors and methods for mitigating the impact of the high uncertainty of the sensors are described. in [20], different occupancy grid mapping methods are compared. In both papers, the environment, robot, and sensors are simulated. The environment is small and the sensors seem to have a considerably higher accuracy than the ones mounted on our test car. Most notably, no unpredictable behavior like what we experience when there is no obstacle in range, described in section 2.3, is described.

As far as we can tell, the problem of performing SLAM with only ultrasonic sensors is largely unexplored. When ultrasonic sensors are used, it is often in small indoor environments and in fusion with other sensors, as in [21] and [11]. In [13], a polynomial based SLAM is performed using only ultrasonic sensors for scanning the environment. [26] presents a feature chain based approach for SLAM using ultrasonic sensors in an indoor environment.

While none of the papers above offer solutions to our specific problems, they offer insight into the possibilities and limitations when working with noisy ultrasonic sensors.

## 1.5 Contributions

This thesis were carried out in close collaboration with Axel Pandolfi Elmi, Umeå University, who carried out his own thesis project on a closely related topic in parallel to this thesis. Approaches and solutions to shared problems were shared and discussed at length. The solutions in chapter 3, above all the odometry model in section 3.1.2 and the mapping algorithm described in section 3.3 were developed in close collaboration with Axel.

## 1.6 Outline

Chapter 2 covers theoretical background on the SLAM problem, and describes the hardware and software tools used in this thesis. Chapter 3 presents theory of the solutions that were implemented to answer the research questions. This includes an odometry model for the test car, a state-of-the-art SLAM algorithm using lidar, an occupancy grid mapping algorithm using the ultrasonic sensors, and an approach to using the ultrasonic sensors for SLAM. Chapter 4 describes how these solutions are implemented and evaluated. In chapter 5, the results are discussed and conclusions are drawn.

# Chapter 2
# Background

## 2.1 SLAM

Simultaneous Localization and Mapping (SLAM) is the problem of creating a map of an environment using sensors on a robot, while at the same time localizing the robot within this map. A SLAM algorithm takes sensor data and odometry data as input. Regardless of the specific algorithm used, the process of solving the problem looks roughly the same on a very high level. For each iteration, the SLAM algorithm first estimates a new robot position based on the odometry data and estimates what, based on old sensor data, the sensors should see from this new position. It then looks at the actual sensor data from the new position and uses it to correct the estimation of the robot position and update the map.

There are multiple different ways to approach the SLAM problem. Each method has its pros and cons regarding speed, accuracy, tolerance of faulty odometry- or sensor data, etc. In [15], a brief history of the SLAM problem is presented and a some approaches to the problem are described, discussed, and compared. In this section, a few different SLAM algorithms and their respective pros and cons, both in general and specifically for the use case in this thesis, are described.

The earliest SLAM algorithms utilize an extended Kalman filter (EKF). In 1986, the use of an extended Kalman filter to solve the SLAM problem was proposed in [27]. A solution is presented in [14]. EKF based SLAM algorithms are typically feature based, meaning that the environment is represented as a set of distinct landmarks. One large con of EFK SLAM is that the uncertainty is modelled as Gaussian noise, which rarely matches the real nature of uncertainty that depends on both faulty scans and imperfect odometry. The computational complexity is also quadratic with regards to the number of landmarks, which makes the approach infeasible for larger maps or environments with a very high number of landmarks. One way to combat this is to split a large map into smaller submaps, as investigated in [25]. Some convergence problems and inconsistencies in EKF SLAM is investigated and discussed in [19].

Another approach to the SLAM problem is to use particle filters. In [23], FastSLAM, a SLAM algorithm based on a Rao-Blackwellized particle filter, is introduced. As suggested by its name, the main pro of this algorithm over EFK SLAM is that it is faster; the computational complexity scales logarithmically with the number of landmarks in the map, as opposed to the quadratic scaling in EKF SLAM. Each particle in the filter contains one Kalman filter for each landmark in the entire map. In [24], the FastSLAM algorithm is improved by taking the most recent sensor measurement in addition to data from the odometry model into account when updating the robot's position. It is worth noting that this is particularly helpful as many robots have noisy odometry models, but relatively accurate sensors with which to scan the environment. However, this is not the case in this thesis, as the odometry model described in section 3.1.2 is considerably more accurate than the ultrasonic sensors.

In [17] and [5], gmapping, a grid-based version of FastSLAM, is introduced and implemented. As opposed to the feature-based algorithms described above, grid-based SLAM algorithms model the environment as occupancy grids. Gmapping was developed in 2007 and is still one of the most widely used SLAM algorithms. In [29], the loop closure of gmapping is improved using a Kalman filter based algorithm.

Yet another SLAM algorithm, GraphSLAM, is presented in [28]. GraphSLAM is a SLAM algorithm that is often used to create 3D maps of large areas containing tens of millions of features or more. It is a full, or offline, SLAM algorithm, which means that it takes pre-recorded data from the whole run of the robot as input, as opposed to online algorithms that works in real time. As this is far from the use case in this thesis, where we want to perform online SLAM in a small area such as a parking lot, we will not investigate GraphSLAM further.

Google Cartographer, presented in [18], is a SLAM system developed by Google in 2016. It is an online SLAM algorithm that generates 2D grid maps.

## 2.2 The Test Car

All data was recorded using a Volvo XC90. The car is equipped with wheel rotation sensors and ultrasonic sensors as described in sections 2.3 and 2.5. It also has an aftermarket front mounted lidar sensor, described in section 2.4. The CAN bus of the car, the bus to which all signals in the car can be accessed, and the lidar sensor are connected to a computer with Robot Operating System (ROS) installed.

## 2.3 Ultrasonic Sensors

Ultrasonic sensors measure the distance of objects by sending out a high frequency sound wave and measuring the time it takes for this wave to bounce off an object and reflect back to the sensor. In the simple scenario of a single sensor that acts as both the transmitter and the receiver, the distance is calculated with equation 2.1, where $L$ is the measured distance, $T$ is the time between transmission and reception, and $C$ is the sonic speed.

$$L = \frac{1}{2} * T * C \tag{2.1}$$

As opposed to lidar sensors and cameras, ultrasonic sensors like these and are already mounted on most new cars sold today.

There are 12 ultrasonic sensors placed around our test car, positioned as follows in relation to the nose of the car, where the *x* axis points from the front to the back of the car and the y axis points from the left to the right side of the car:

| sensor number | x [cm] | y [cm] | angle [degrees] |
|---|---|---|---|
| 1 | 38.2 | -90.8 | 265 |
| 2 | 28.9 | -83.1 | 211 |
| 3 | 03.8 | -33.0 | 180 |
| 4 | 03.8 | 33.0 | 180 |
| 5 | 28.9 | 83.1 | 149 |
| 6 | 38.2 | 90.8 | 95 |
| 7 | 436.0 | 93.5 | 82 |
| 8 | 480.0 | 76.2 | 43 |
| 9 | 491.5 | 32.1 | 6 |
| 10 | 491.5 | -32.1 | -6 |
| 11 | 480.0 | -76.2 | -43 |
| 12 | 436.0 | -93.5 | -82 |

These sensor placements are visualized in figure 2.3. The sensors we use has an update frequency of 10hz, an opening angle of 70 degrees, and a specified minimum and maximum detection range of 15 centimeters and 5.5 meters respectively.

Each sensor report three types of echos; a direct echo, a left cross-echo and a right cross-echo. A direct echo is reported when the same sensor that transmitted a signal receives it and a cross-echo is reported when the sensor to the left or to the right of the transmitting sensor receives the echo. For each type of echo, the sensor reports the first, second and third echo received. For each echo reported, the sensors also report a quality factor between 0 and 100, representing how certain the sensor is that the reported value is correct.

Over all, the sensors have a very low accuracy. During our tests, we found that even tough the sensors have a specified maximum range of 5.5 meters, readings at a greater distance than about 3.5 meters were largely unreliable. We also found that only the first echos were accurate enough to be of use. An example of ultrasonic sensor data in the real world is visualized in figure 2.1, which contains visualizations from test run 1, described further in section 4.1.1. The arcs around the car are visualizations of the sensor readings. For each reported reading from each sensor, we visualize the average of the first direct echo, right cross-echo and left cross-echo, to mitigate the impact of eventual errors in any one of them. Sensor readings with a high reported quality factor are visualized as dark red, and readings with a lower reported quality factor are visualized as lighter yellow. In the left image where the car is close to a wall, we can see that the sensors report this fairly accurately. In the right image where there is no wall within range of the ultrasonic sensors, they report seemingly random values.

The accuracy of the sensors decline with both the distance between the sensor and the object, and the angle between the sensor's sonar axis and the object. We elaborate further on this in section 3.3.1.

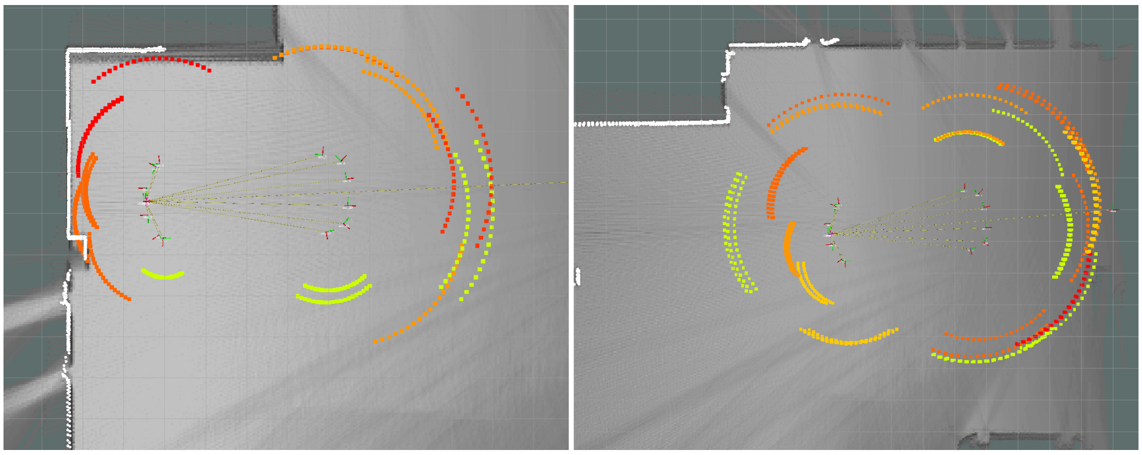

**Figure 2.1:** A visualization of ultrasonic data in two scenarios

## 2.4 Lidar Sensors

A light detection and ranging (lidar) sensor measures distances by emitting laser light and measuring its reflections. The sensor used in this thesis has a maximum range of 100 meters and an accuracy of about 3cm [6]. The sensor is capable of creating a 3d scan with a 30 degree vertical field of view and a 360 degree horizontal field of view.

We use this lidar sensor for scanning the environment in our state-of-the-art SLAM model in section 3.2. In figure 2.1, the map used as a ground truth is created using lidar. The white lines along the walls in front of the car are visualizations of the latest lidar scans.

On the test car, the lidar is mounted on the nose of the car and the horizontal field of view is limited to 180 degrees. As we limit the scope of the thesis to 2D SLAM, we only consider the scans with a 0 degree vertical angle.

## 2.5 Wheel Rotation Sensors

Each wheel of the test car has a sensor that reports the wheel's rotations. The sensor reports a number starting at 0 and increasing to 255 before starting over at 0 again. The value increases by 1 for every 1/96 full rotations of the wheel. We use these sensors for the odometry model, described in section 3.1.

## 2.6 Robot Operating System

Robot Operating System (ROS) [9] is used for all implementations in this thesis. ROS is a framework for writing robot software. The basics of ROS relies on a few core concepts: *nodes* communicating with each other by sending or receiving *messages* on *topics*. Nodes are self-contained pieces of code, in our case written in Python. For example, the odometry model described in section 3.1.2 runs as a ROS node that subscribes to sensor data from the appropriate topics, processes the data, and publishes car coordinates on the `/odom` topic.

Another concept in ROS worth describing further is the *tf transform* [10]. Tf is a ROS package that lets the user keep track of multiple coordinate frames over time. Figure 2.2

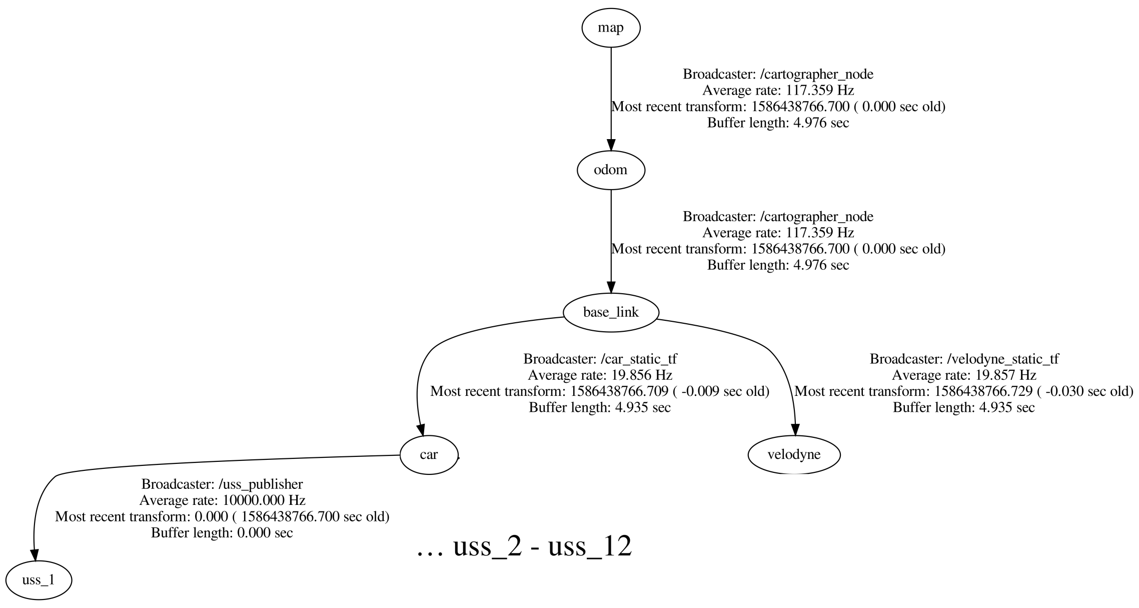
map
Broadcaster: /cartographer_node
Average rate: 117.359 Hz
Most recent transform: 1586438766.700 ( 0.000 sec old)
Buffer length: 4.976 sec
odom
Broadcaster: /cartographer_node
Average rate: 117.359 Hz
Most recent transform: 1586438766.700 ( 0.000 sec old)
Buffer length: 4.976 sec
base_link
Broadcaster: /car_static_tf
Average rate: 19.856 Hz
Most recent transform: 1586438766.709 ( -0.009 sec old)
Buffer length: 4.935 sec
Broadcaster: /velodyne_static_tf
Average rate: 19.857 Hz
Most recent transform: 1586438766.729 ( -0.030 sec old)
Buffer length: 4.935 sec
car
velodyne
Broadcaster: /uss_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 ( 1586438766.700 sec old)
Buffer length: 0.000 sec
uss_1
… uss_2 - uss_12

**Figure 2.2:** A diagram of the tf frames

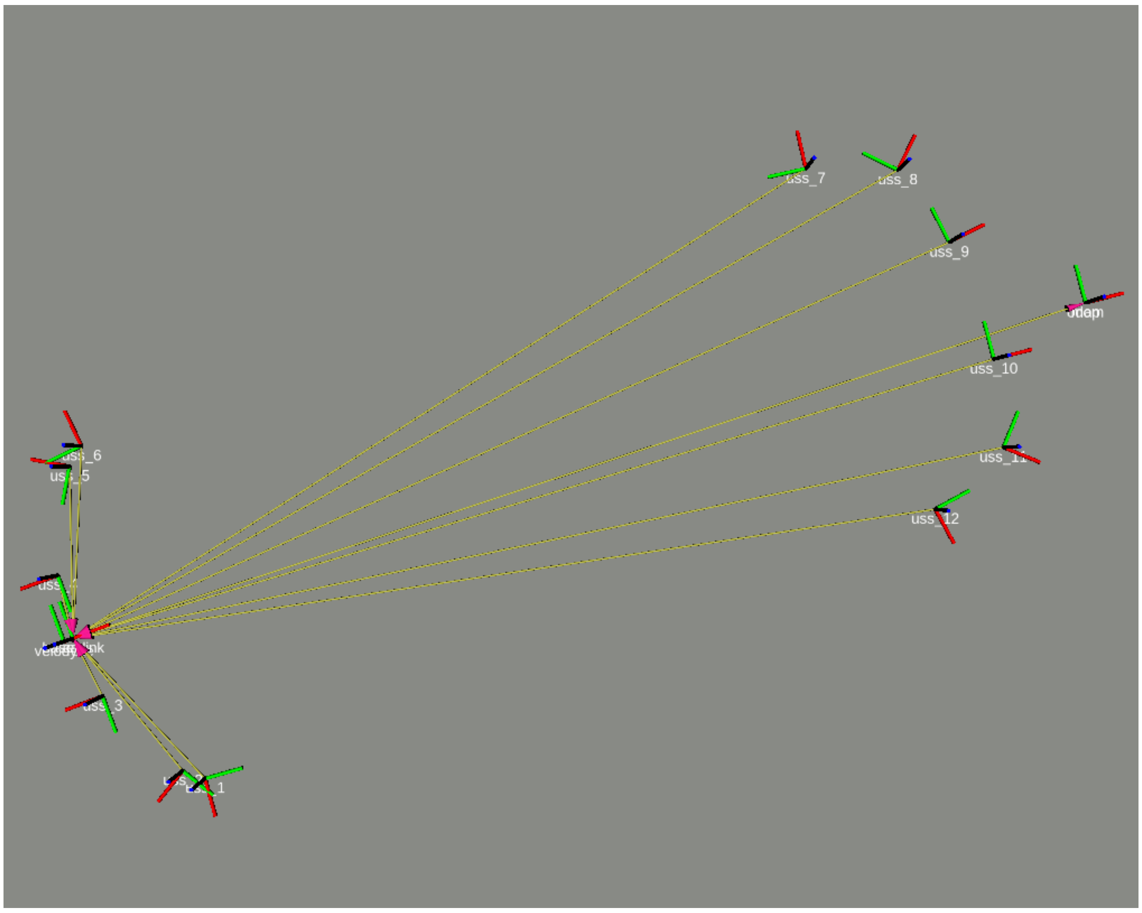

**Figure 2.3:** A visualization of the tf frames

shows a diagram of the tf frames we use when running Google Cartographer as described in section 3.2. `map` is the reference world frame, in relation to which everything else is organized. The `odom` and `base_link` frames are required for the SLAM algorithms as specified in [7]. The other frames make up our model of the car. The car has a position and angle in relation to the map and the lidar sensor and the ultrasonic sensors have positions and angles in relation to the car. The results are visualized in figure 2.3. The tf frames makes it easy to, for example, publish ultrasonic sensor readings in relation to the ultrasonic sensor, without having to worry about where the sensor is located in relation to the car, the map, or anything else.

One of the tools used extensively in this thesis is rosbag [8]. Rosbag is a node that allows for recording of messages on ROS topics to a file, which can later be played back in real time. When doing our test runs, described further in sections 4.1.1 and 4.1.2, we run only small nodes that read and decode data from the CAN bus of the car using `cantools` [1], convert the data to ROS data types, and publish it on appropriate ROS topics. The rosbag node subscribes to these topics to record the data, allowing us to replay the exact same data any number of times later.

# Chapter 3
# Approach

In this chapter the approach to solving the problems needed to answer the research questions is described. This includes building an odometry model for the car, a SLAM algorithm using the lidar data, an algorithm for creating an occupancy grid using the ultrasonic sensor, and a method for performing SLAM using the ultrasonic sensor.

## 3.1 Odometry

### 3.1.1 Traditional Odometry Model

Figure 3.1 shows a visualization of a traditional car odometry model, used for example in [22]. The car position is represented by the $x$ and $y$ variables at the center of the front axle, and the car rotation is represented by the $\theta$ variable. As the steering wheel is turned, $\phi$ increases or decreases and the center of rotation $C$ (in the coordinate system of the car) can be found in the intersection between the lines through the front- and the rear wheel axles using equation 3.1.

$$\begin{aligned} x_C &= \frac{L}{\tan\phi} \\ y_C &= -L \\ R &= \sqrt{x_C^2 + y_C^2} \end{aligned} \tag{3.1}$$

When the car moves, it moves along the circle with its center at $C$ and a radius of $R$. When the car is not turning at all, the special case of $\phi = 0$ and $R = \infty$ applies. In this case, the equations below are undefined due to a division by zero in equation 3.1, and we simply move the car in a straight line instead. In all other cases, the new car position in the old car coordinates can be found using equation 3.3, where $d$ is the distance the car has moved,

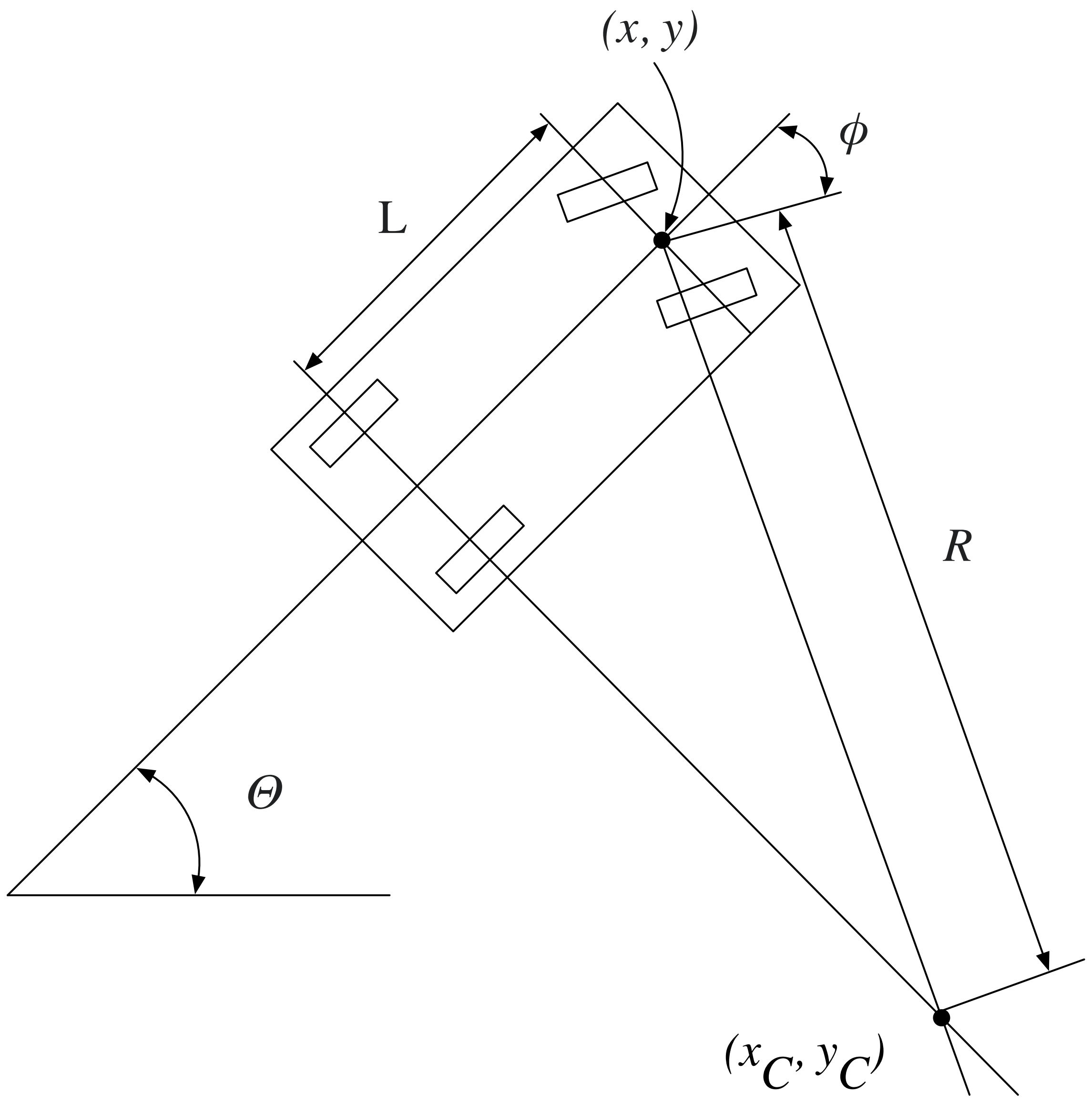

**Figure 3.1:** A traditional car odometry model

which can be found using equation 3.2.

$$d = \frac{n\_wheel\_ticks * wheel\_circumference}{96} \tag{3.2}$$

$$\begin{aligned} x_{new} &= x_C + R * \cos(\phi + \frac{d}{R}) \\ y_{new} &= y_C + R * \sin(\phi + \frac{d}{R}) \end{aligned} \tag{3.3}$$

The car position can then be translated to world coordinates using equation 3.4.

$$\left( \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x_{new} & y_{new} \end{bmatrix} + \begin{bmatrix} x & y \end{bmatrix} \right)^{\mathsf{T}} \tag{3.4}$$

The new car angle in world coordinates can be found using equation 3.5.

$$\theta_{new} = \theta + \frac{d}{R} \tag{3.5}$$

Using this odometry model, the new position and the new angle depend on both the wheel circumference of all four wheels, the steering ratio, and the car's track. The dimensions and the specified steering ratio of our test car can be found in [3] and [4]. The wheel circumference varies with tire pressure, how heavily the car is loaded, and other factors. We found that the dimensions and steering ratio found in the specifications, while close, does not exactly match our test car with sufficient accuracy. Because of this, we found it hard to set the parameters in a way that produced an accurate odometry model.

### 3.1.2 Rear Wheel Rotation Model

Another way to produce an odometry model is to consider only the rotations of the two rear wheels. When using this approach, we use the center of the rear axle as the car position, as opposed to the center of the front axle like in section 3.1.1.

The distance the car has travelled is the average of the distance the rear wheels has rolled, and the turning angle of the car is derived from the difference between the the rear wheels using equation 3.6, where $d_r$ is the distance the right wheel has rolled and $d_l$ is the distance the left wheel has rolled respectively. These are both calculated using equation 3.2.

The difference between the old and the new $\theta$ is calculated using equation 3.6. The distance $d$ travelled by the car is calculated as the average between $d_r$ and $d_l$, and the new car position in the old car coordinates is calculated using equation 3.7. The new car position in the world coordinates is calculated in the same way as for the traditional odometry model.

$$\theta_\Delta = \arctan\left(\frac{d_l - d_r}{wheel\_base}\right) \tag{3.6}$$

$$\begin{aligned} x_\Delta &= d * \cos(\theta + \theta_\Delta) \\ y_\Delta &= d * \sin(\theta + \theta_\Delta) \end{aligned} \tag{3.7}$$

While this approach is arguably further from an accurate model of the real life physics of a moving car, one big pro of this approach is that the only changeable parameters the model depends on are the wheel circumferences of the two rear wheels, as opposed to the many other parameters in the traditional model.

## 3.2 State-of-the-art SLAM with Lidar

As a baseline for our tests with the ultrasonic sensors, we run a state-of-the-art SLAM algorithm using the front-mounted lidar sensor of the test car for scanning the environment.

The most common SLAM algorithms are briefly described in section 2.1. In this thesis, we consider the case of 2D mapping in a relatively small and static environment, without any considerable hardware limitations. In [16], a few different SLAM algorithms are compared in different use cases, and Google Cartographer is recommended for 2D lidar SLAM. Based on this, we chose to use the Google Cartographer ROS implementation [2].

The lidar sensor on the test car, described in section 2.4 is used for scanning the environment, and the model described in section 3.1.2 provides the odometry data.

## 3.3 Mapping Using Ultrasonic Sensors

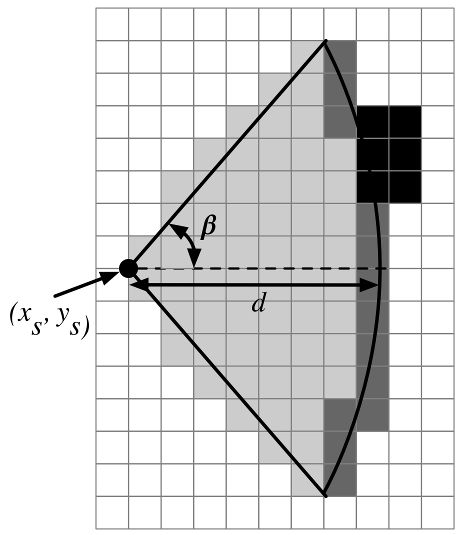

**Figure 3.2:** An ultrasonic reading in the occupancy grid

As a first step in evaluating how the ultrasonic sensors can help a driver in a parking scenario, we use their data to create an occupancy grid of the environment surrounding the car. In [20], a few different sonar based occupancy grid mapping methods are compared. In this thesis, we choose to create a Bayesian map, since it is not very complex and produces good results.

A sensor reading is interpreted according to figure 3.2, where $(x_s, y_s)$ is the sensor position, $\beta$ is half the sensor's opening angle, and $d$ is the distance measured to an obstacle. The black cells are the actual obstacle seen by the sensors. The light grey tiles are marked as free and the dark grey tiles are marked as occupied. The white ones are not in the sensor's range, and are not updated at all. For each sensor reading, the probability of all cells marked as occupied is increased and the probability of all cells marked as free is decreased. As the car moves around and all 12 sensors report values, the goal is that this will produce a sufficient occupancy grid despite the low accuracy and large opening angle of each individual reading.

For every sensor reading, we run algorithm 1 to update the probabilities of each cell.

$p_{t-1}$ is the prior probability of the cell, $p_t$ its updated probability, $l_{occ}$ is the value with which we increase the probability of the cell if it is reported as occupied, and $l_{free}$ is the value with which we decrease the probability of the cell if it is reported as free.

```
foreach cell c do
    $p_{t-1} \leftarrow$ probability of c being occupied before this sensor reading
    if c is reported as occupied then
        $p_t = p_{t-1} + l_{occ}$
    else if c is reported as free then
        $p_t = p_{t-1} - l_{free}$
    else
        $p_t = p_{t-1}$
    end
end
```

**Algorithm 1:** Algorithm for updating the Bayesian map with data from a new sensor reading

The distance $d_c$ and angle $\phi$ between a cell and the sensor is calculated using equation 3.8, where $(x_c, y_c)$ is the cell position and $\theta$ is the sensor angle.

$$\begin{aligned} d &= \sqrt{(x_s - x_c)^2 + (y_s - y_c)^2} \\ \phi &= \operatorname{atan2}(y_s - y_c, x_s - x_c) - \theta \end{aligned} \tag{3.8}$$

For a sensor reading with a reported range $d$, a cell $c$ is reported as free if it is within the sensor's opening angle and at a smaller distance from the sensor than the reported range; $\phi < \beta$ and $d_c < d - \alpha/2$, where $\alpha$ is the size of a cell in the grid. A cell is reported as occupied if it's within the sensor's opening angle and, within the margin of the size of a cell, at distance $d$ from the sensor; $\phi < \beta$ and $d_c - d < \alpha/2$.

### 3.3.1 Improvements to the Bayesian Map

In this section, we investigate ways to increase the quality of the simple Bayesian map described in section 3.3.

As described in section 2.3, the accuracy of the ultrasonic sensors decrease with the distance from an eventual obstacle both to the sensor and to the sensor's sonar axis. Using this information, we construct a model that decrease the probability that is added to a cell marked as occupied by a sensor reading as these distances increase, as suggested in [20].

The angular modulation function, 3.9, describes the reduction in certainty based on the distance between a cell and the sensor's sonar axis. $\phi$ is the angle between the sonar axis and a given cell and $\beta$ is half the sensor's opening angle.

$$\alpha(\phi) = \begin{cases} 1 - \left(\dfrac{\phi}{\beta}\right)^2 & 0 \le \phi \le \beta \\ 0 & |\phi| > \beta \end{cases} \tag{3.9}$$

The radial modulation function, 3.10, describes the reduction in certainty based on the distance between a cell and the sensor. $\Delta(d_c)$ is the radial modulation function, $d_c$ is the

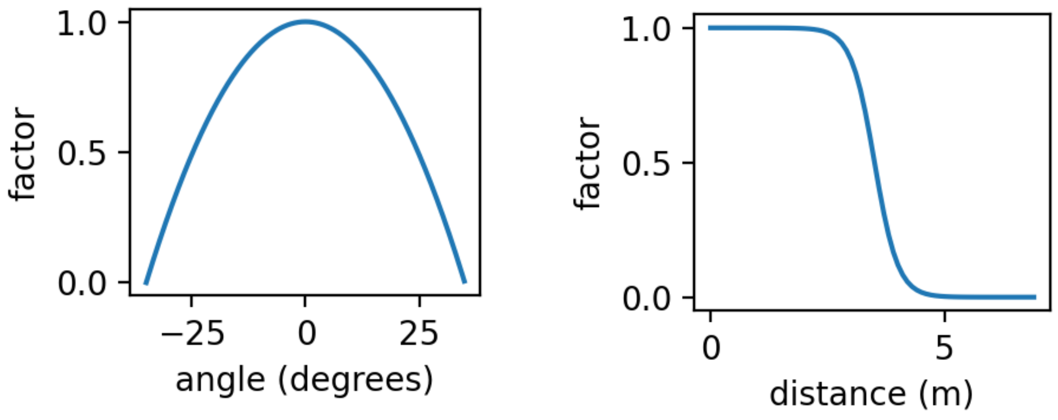

**Figure 3.3:** Angular and radial modulation

distance from the sensor to a given cell and $\rho_v$ is the point at which the drop off in certainty is 50%. We set this to 3.5 in our tests, since we observed a considerable drop in the quality of readings for obstacles further away than that.

$$\Delta(d_c) = 1 - \frac{1 + \tanh(2(d_c - \rho_v))}{2} \tag{3.10}$$

Equation 3.9 and 3.10 are plotted in figure 3.3.

The vast majority of false ultrasonic sensor readings occur when there is no obstacle in range of the sensor. When there is an obstacle within 3.5 meters, the sensor reports this with a reasonably high accuracy. In the case of false readings when an obstacle is within range of the sensor, it's extremely rare that the reported value is greater than the actual range to the obstacle; for each sensor reading, we can be reasonably confident that there is no obstacle closer than the reported value. Because of this, we only use these functions to decrease the probability of occupancy added to grid cells for each ultrasonic sensor reading. The probability subtracted when a sensor reports a cell as free stays constant regardless of the range reported or the angle between the cell and the sensor axle.

## 3.4 SLAM Using Ultrasonic Sensors

As discussed in sections 2.3 and 2.4, lidar sensors and ultrasonic sensors work in fundamentally different ways. Apart from the lower accuracy of the ultrasonic sensors, the largest concern for us is their large opening angle. In this thesis, for example in figure 3.2, ultrasonic sensor readings are represented as arcs representing the range of points at which the sensor could have sensed an object. The probability of the object being at any one point on these arcs is modelled according to equations 3.9 and 3.10. For use in a SLAM algorithm, however, we need precise points to track and scan match, not arcs and probability distributions. The low accuracy is a problem for scan matching in particular. In our case, the odometry model described in section 3.1.2 produces more reliable data than the ultrasonic sensors. This means that even if the sensors reported points at specific coordinates instead of arcs, the scan matching would introduce more errors to the odometry instead of correcting it.

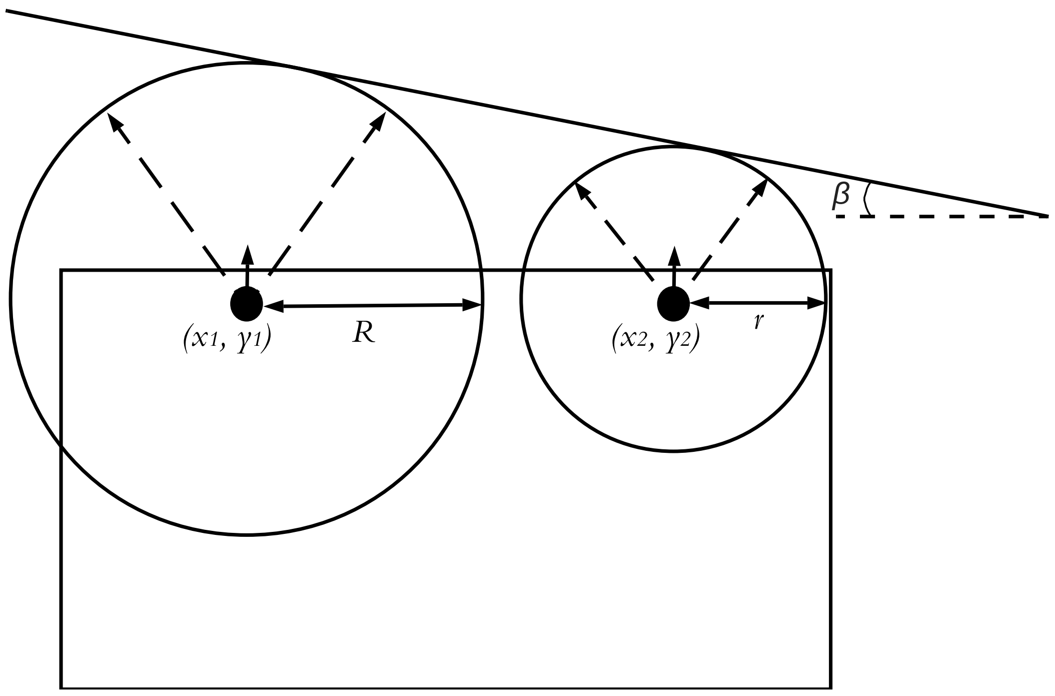

**Figure 3.4:** A model of the car with a straight obstacle to the side

Because of this, we need to pre-process the ultrasonic data in some way if we want to use it as input to a SLAM algorithm. The problem of the low accuracy can be tackled by only considering objects very close to the car, where the accuracy of the sensors is much greater and the random errors are considerably less common than at larger distances. As the scope of this thesis involves tight parking scenarios, this could be a car or a wall very close to a parking lot occupied by our car.

The problem of the large opening angles can be handled by assuming a known simple structure to which we can fit the ultrasonic sensor readings. In the scenario of a tight parking lot occupied by our car discussed above, we may assume somewhat straight obstacles to the left and the right of the car, at some distance and angle from the car.

These straight obstacles to the left or the right of the car are modelled according to figure 3.4. With the data from two side-facing ultrasonic sensors at positions $(x_1, y_1)$ and $(x_2, y_2)$, reporting an obstacle at distance $R$ and $r$ respectively, we model the wall as the tangent line of the two sensors' arcs. The angle between the car and the wall is calculated using equation 3.11, where the ± depends on if the wall is on the right side or the left side of the car.

$$\beta = \pm \arcsin\left(\frac{R - r}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}\right) \tag{3.11}$$

To change $\beta$ to be in relation to the world instead of the car, we simply add or remove, again depending on if the wall is on the right or the left side, the car's angle according to equation 3.12.

$$\beta_{world} = \beta \pm \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{3.12}$$

Using this method, we produce `sensor_msgs/LaserScan` messages corresponding to the wall in ROS, described further in section 4.6.

# Chapter 4

# Implementation and Evaluation

This section contains descriptions of the implementations of the solutions presented in section 3. The evaluation method is also described, and all implementations are evaluated.

## 4.1 Data Used for Evaluation

For evaluating our results, we relied on two runs of the test car. During both runs, we recorded wheel rotations, the vehicle motion state, the steering wheel angle, lidar data, and ultrasonic sensor data from the car's CAN bus using rosbag [8].

### 4.1.1 Test Run 1

In the first test run, we drove around in a nearly empty parking garage. A map of this is shown in figure 4.3. We started close to the parking house exit, side to side with one of the other cars; the topmost one that can be seen in the map. We then drove forwards slowly towards the wall of the garage, to the left in the map, stopped right in front of it, backed away, and turned around.

The relatively long distance we drove, including both wide and tight turns and driving both forwards and backwards, makes this data well suited for evaluating the quality of the odometry model. The section where we drove up to the wall were also useful for determining the behavior and quality of the ultrasonic sensors in different scenarios.

### 4.1.2 Test Run 2

The second test run is mapped in figure 4.4. We started on the parking lot at the bottom left of the map, with cars to the left and to the right, and a wall in front of the car. We slowly backed out of the parking lot, stopped, and then drove forwards (upwards in the map) a few meters.

This test run is a good example of the low speed parking scenario on which we focus in this thesis.

## 4.2 Data Formats

The data needs to be formatted as ROS messages in order to be saved in a rosbag. The wheel rotations, described in section 2.5, are represented by the ROS data type `std_msgs/UInt8`. The vehicle motion state is represented by `std_msgs/Int8`, where -1 represents the car moving backwards, 0 represents the car standing still, and 1 represents the car moving forwards. The steering wheel angle is reported in radians by a decimal number between roughly $-3/2\pi$ and $3/2\pi$, represented by a `std_msgs/Float32` in ROS.

The lidar data is more complex, and is represented by messages of the type `sensor_msgs/LaserScan` where each message contains a range of laser scans, each with its own measured range and angle from the sensor axle.

```
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

We implemented our own ROS data type to represent the ultrasonic sensor readings. The message does not contain any information about the reading's angle, like the laser scan message does. Instead, we only specify the sensor number in the message and specify the placement and angle of each sensor on the car directly in the source code. The different echos and qualities are described in more detail in section 2.3.

```
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
uint8 sensor_number
float32 dir_echo_1_distance
uint8 dir_echo_1_quality
float32 dir_echo_2_distance
uint8 dir_echo_2_quality
float32 dir_echo_3_distance
uint8 dir_echo_3_quality
float32 ce_left_echo_1_distance
```

```
uint8 ce_left_echo_1_quality
float32 ce_left_echo_2_distance
uint8 ce_left_echo_2_quality
float32 ce_left_echo_3_distance
uint8 ce_left_echo_3_quality
float32 ce_rgt_echo_1_distance
uint8 ce_rgt_echo_1_quality
float32 ce_rgt_echo_2_distance
uint8 ce_rgt_echo_2_quality
float32 ce_rgt_echo_3_distance
uint8 ce_rgt_echo_3_quality
```

## 4.3 Odometry

The goal of the odometry model is to use the sensors available on the car, described in section 2.5, to find a car pose, (*x* and *y* coordinates and a car angle), in a 2D plane. The results are formatted as ROS `nav_msgs/Odometry` messages for use as input to a SLAM algorithm.

```
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
    geometry_msgs/Pose pose
        geometry_msgs/Point position
            float64 x
            float64 y
            float64 z
        geometry_msgs/Quaternion orientation
            float64 x
            float64 y
            float64 z
            float64 w
    float64[36] covariance
geometry_msgs/TwistWithCovariance twist
    geometry_msgs/Twist twist
        geometry_msgs/Vector3 linear
            float64 x
            float64 y
            float64 z
        geometry_msgs/Vector3 angular
            float64 x
            float64 y
            float64 z
```

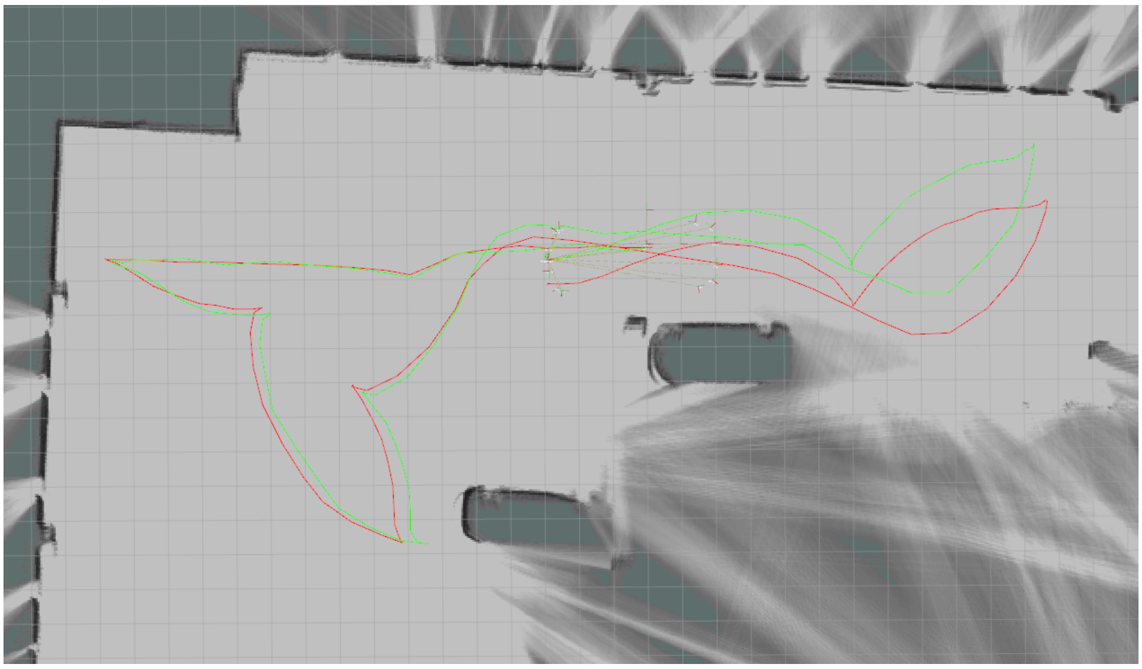

**Figure 4.1:** The real (green) and the estimated (red) path of the car during test run 1

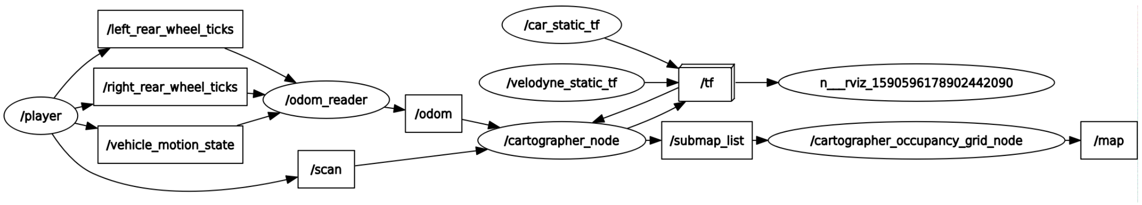

**Figure 4.2:** Graph of the ROS environment when running Google Cartographer

```
float64[36] covariance
```

This means that in this message, we only used the `geometry_msg/Pose` part, and only the $x$ and $y$ coordinates and the orientation in one plane. The $z$ coordinates, the twist and the covariances were left out in our implementation.

Figure 4.1 shows the estimated path of the car during test run 1 using the odometry model described in section 3.1.2 in red. The green line is the real path of the car. As the car travelled 77 meters the difference between the real car position and the uncorrected position from the odometry model never exceeded 2 meters, and Google Cartographer never failed to properly correct the errors using scan matching.

## 4.4 State-of-the-art SLAM with lidar

Figure 4.2 shows a visualization of all nodes and topics active when running Google Cartographer in ROS. The `/player` node replays data recorded to a rosbag from the CAN bus of the car; rear wheel rotations, vehicle motion state, and lidar scans, and publishes it

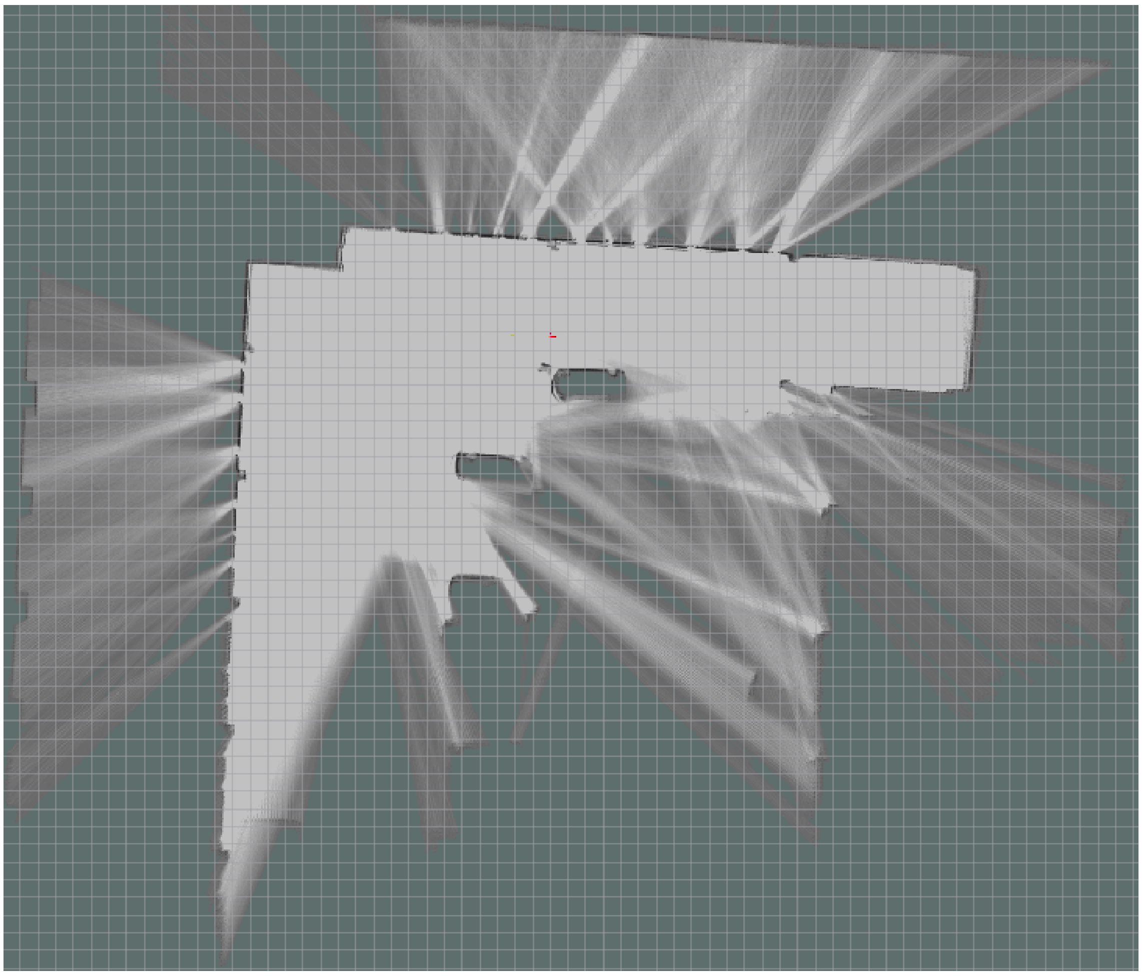

**Figure 4.3:** The map produced by Google Cartographer in test run 1

to appropriate topics. The `/odom_reader` node subscribes to the wheel rotations and the vehicle motion state and publishes odometry messages to the `/odom` topic. The SLAM algorithm runs in the `cartographer_node` node, which subscribes to the odometry messages and the lidar scans, and publishes a map and tf transforms for the car position.

The results of running Google Cartographer on test run 1 are shown in figure 4.3. All walls are straight and the parked cars are captured with great accuracy. An example of what a map can look like when the SLAM algorithm does not sufficiently correct the robot pose can be seen in [26], figure 1.

Figure 4.4 shows the results of running Cartographer on test run 2. Since the car only moved a few meters this run, we could not evaluate how well the scan matching corrected the odometry, but we could see that the lidar produced a scan of the surrounding with great accuracy, down to the wheel houses of individual cars.

## 4.5 Mapping Using Ultrasonic Sensors

The occupancy grid is represented by the `nav_msg/OccupancyGrid` type in ROS.

```
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
nav_msgs/MapMetaData info
    time map_load_time
    float32 resolution
    uint32 width
    uint32 height
    geometry_msgs/Pose origin
        geometry_msgs/Point position
            float64 x
            float64 y
            float64 z
        geometry_msgs/Quaternion orientation
            float64 x
            float64 y
            float64 z
            float64 w
int8[] data
```

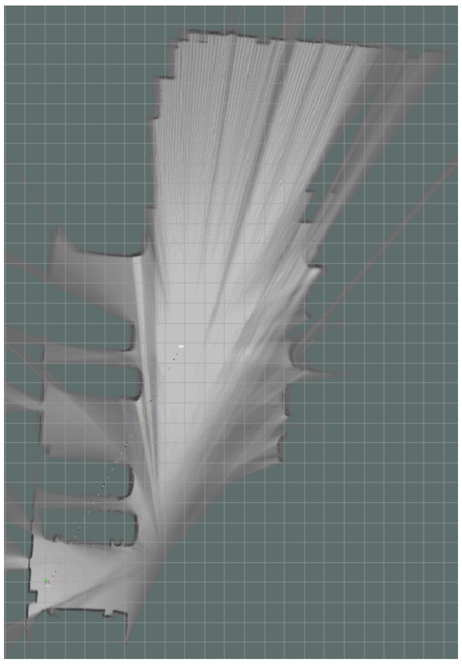

**Figure 4.4:** The map produced by Google Cartographer in test run 2

In the `info` field, the resolution, size, and middle point of the map is specified. The `data` field contains one `int8` for each grid cell. The values range from $-1$ to 100, where $-1$ represents an unknown cell and values between 0 and 100 represents the probability of a cell being occupied. In the visualization of the grid, for example in figure 4.5, the color of the cells range from white to black as the probability of occupancy range from 0 to 100. The cells marked as unknown get the grey color that can be seen along the edges in the figure.

Figure 4.5 shows the occupancy grid that was created using the ultrasonic sensor data during test run 1, without the improvements to the algorithm described in section 3.3.1. Cells occupied by obstacles, such as the wall on the top left and top right, and the two cars we drove close to, are correctly marked as occupied in this grid. The free space is mostly marked as free, with a few exceptions. As described in section 2.3, we set the maximum range of the sensors to 3.5 meters. In parts of the map such as the bottom left, the algorithm falsely marked the area about 3.5 meters from the car as occupied. When there is nothing in range of the sensors and they report seemingly random values, as visualized in figure 2.1, the faulty readings with a low reported distance are quickly compensated for by readings in the same direction but with a higher distance. The faulty readings with a distance close to 3.5 meters, however, will never be compensated for. Similar results can be seen in figure 4.6, which shows the corresponding occupancy grid in test run 2. The area at the bottom is free in reality, but marked as occupied in the occupancy grid.

Figure 4.7 shows the occupancy grid that was created using the ultrasonic sensor data during test run 1, with the improvements described in section 3.3.1. In this grid, the errors described above are mitigated. When we decrease the impact of readings far from the car, the values of the cells fade seamlessly from free to unknown in the bottom left of the grid. Similar results are seen in test run 2, figure 4.8. The wall on the bottom right of the grid is correctly marked as occupied while the unexplored free space on the bottom of the grid is marked as free fading into unknown.

## 4.6 SLAM Using Ultrasonic Sensors

After calculating $\beta$ as described in section 3.4, the wall was modelled as a `sensor_-msgs/LaserScan` message in ROS. As described in section 4.2, a laser scan message contains a range of points with some angle and distance from the sensor. We drew the line as 178 points with angles in the range between $\beta - 89$ and $\beta + 89$ degrees from the sensor's sonar axis. The range of a point with some angle $\alpha$ was calculated as $r/\cos(\alpha)$, where $r$ is the range reported from the sensor. The algorithm was tested on test run 2, and the results are visualized in figure 4.9, where the yellow to red lines are the raw ultrasonic sensor readings and the green lines are the lines constructed to run along the walls.

Lines like these can be used as input for a SLAM algorithm which will be able to correct the car pose with scan matching. We elaborate further the possibilities and limitations of this in section 5.

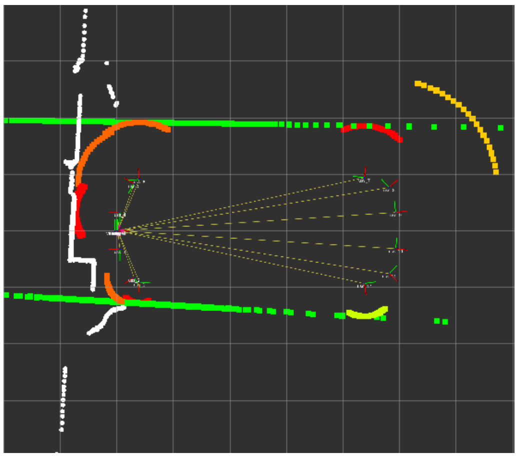

**Figure 4.9:** Visualization of the laser scan messages constructed using the data from the side-facing ultrasonic sensors

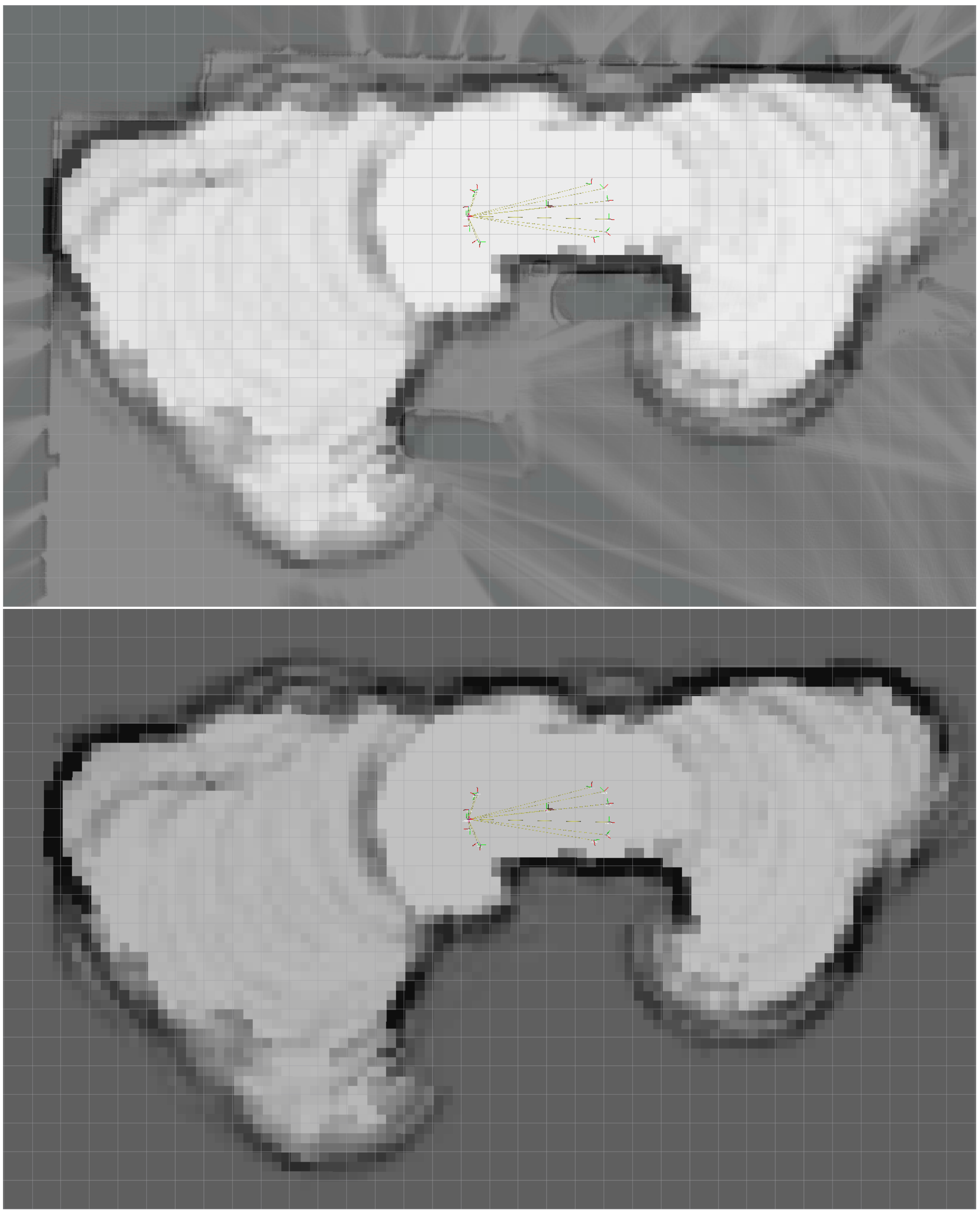

**Figure 4.5:** Test run 1 without angular and radial modulation, Google Cartographer map and occupancy grid on the top, only the occupancy grid on the bottom

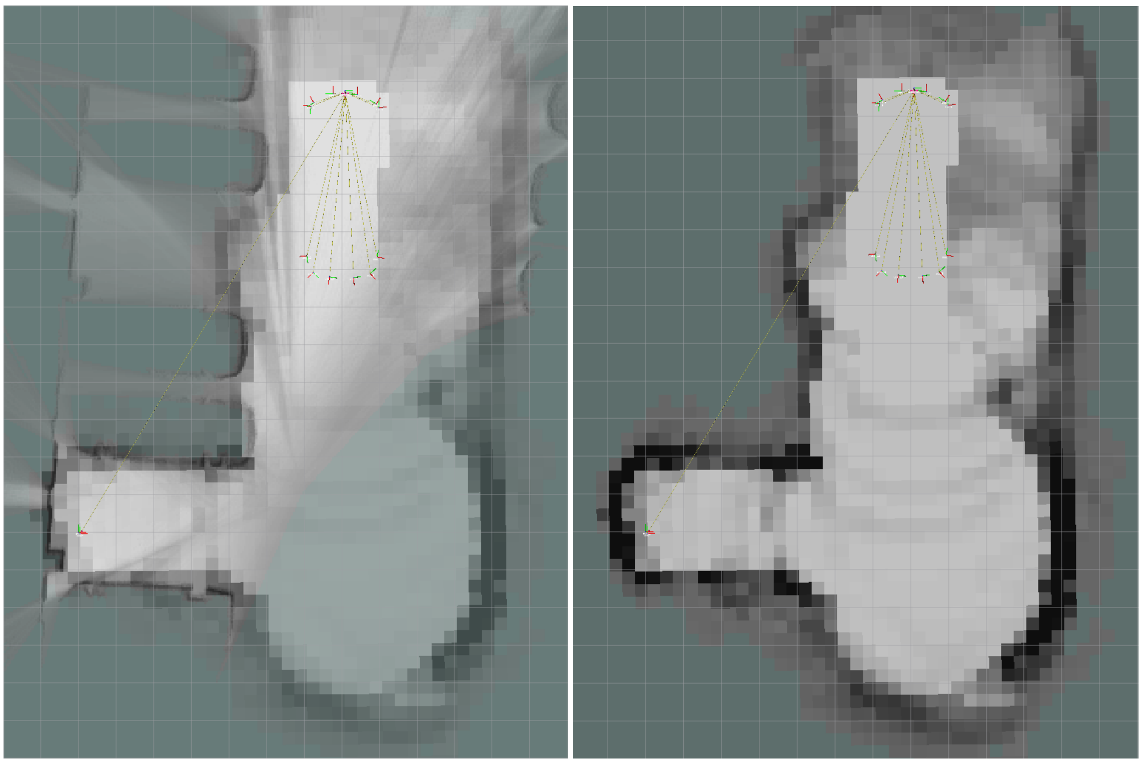

**Figure 4.6:** Test run 2 without angular and radial modulation, Google Cartographer map and occupancy grid to the left, only the occupancy grid to the right

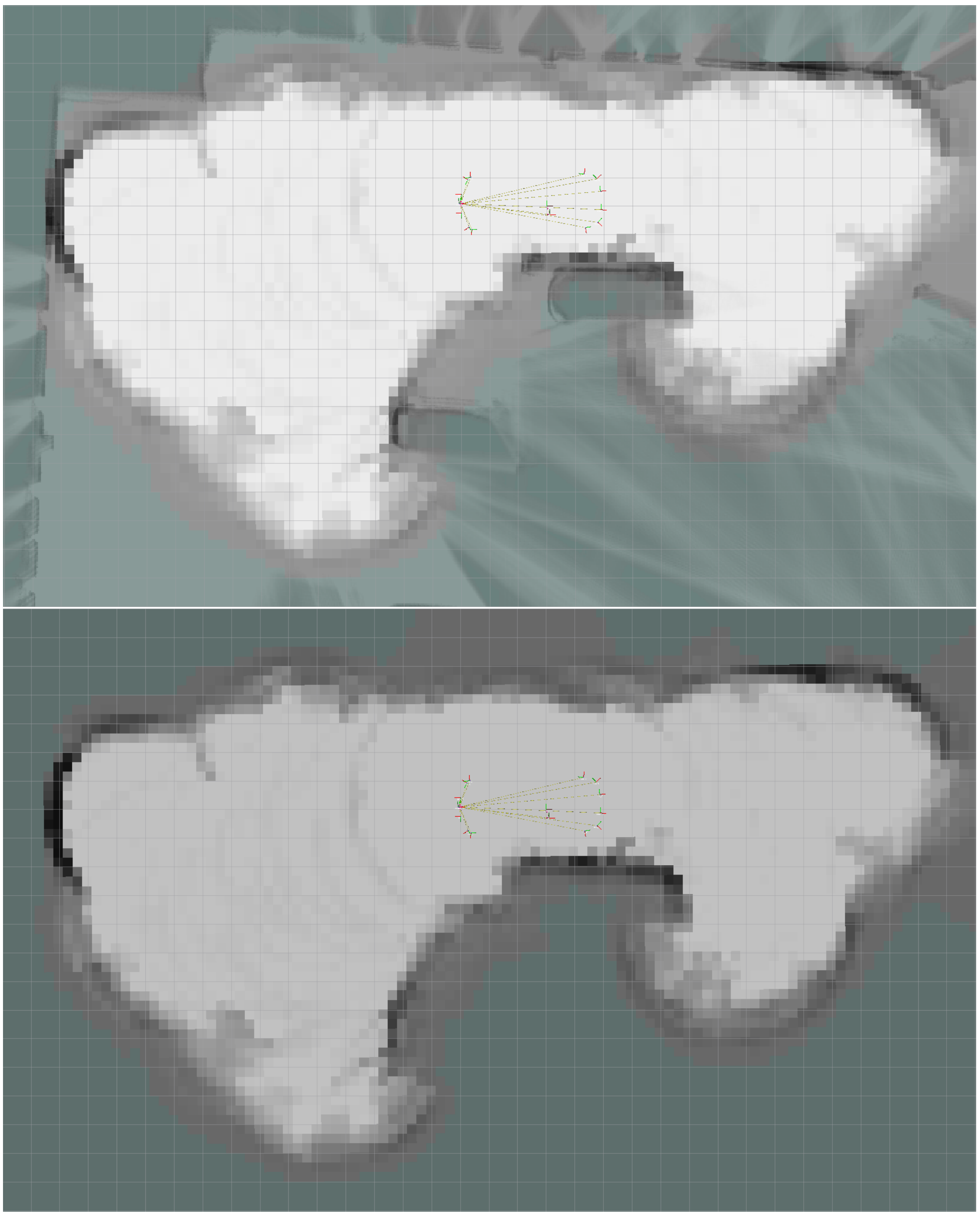

**Figure 4.7:** Test run 1 with angular and radial modulation, Google Cartographer map and occupancy grid on the top, only the occupancy grid on the bottom

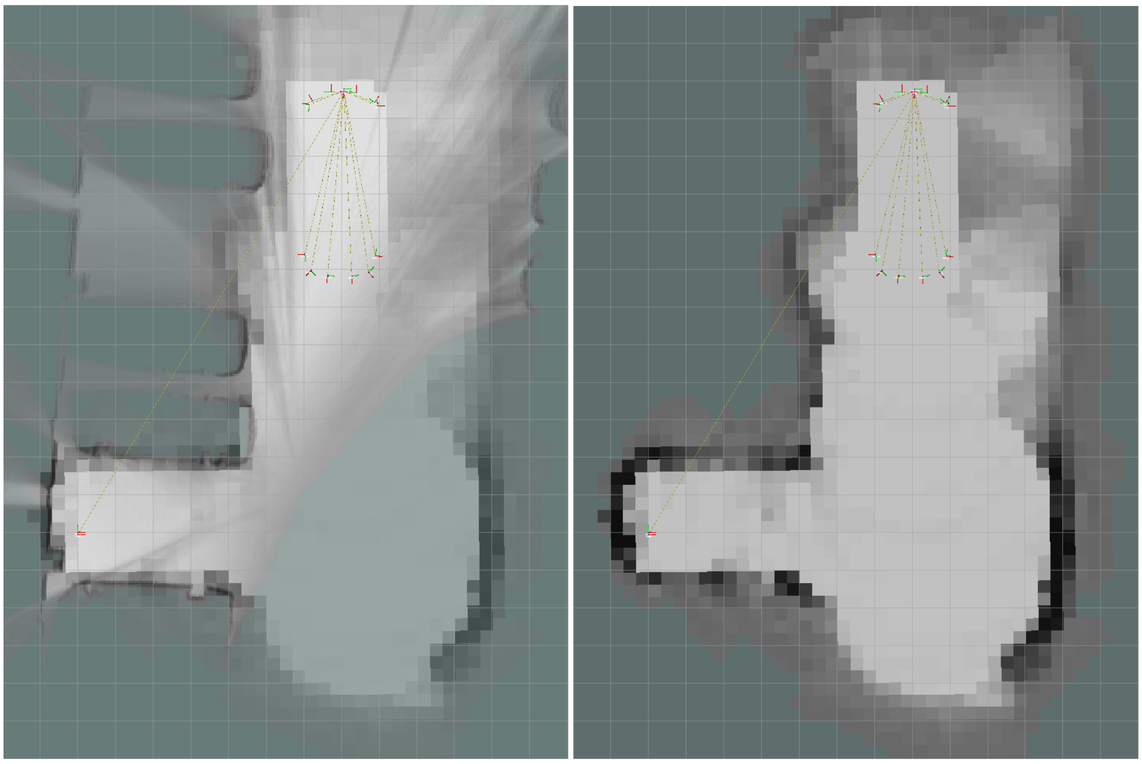

**Figure 4.8:** Test run 2 with angular and radial modulation, Google Cartographer map and occupancy grid to the left, only the occupancy grid to the right

# Chapter 5
# Discussion and Conclusions

Above all because of the uncertainty regarding wheel circumferences, building an accurate odometry model proved to be difficult. Nevertheless, all models we tried were sufficient for state-of-the-art SLAM using the lidar sensor. The scan matching in the SLAM algorithm quickly compensated for any error in the odometry model and prevented errors from accumulating, especially in the low speed scenarios we limited our research to. As we move to sensors with greater uncertainty and lower range, however, the quality of the odometry model becomes more important.

We found that the most important thing to consider when optimizing the odometry model was to decrease the number and minimize the impact of potentially faulty assumptions. While arguably being more elegant, the traditional odometry approach described in section 3.1.1 is dependent on both the steering ratio and the circumference of all four wheels. This made the model hard to tune, and made it very susceptible to changes in factors that have an impact on the wheel circumferences, such as tire pressures and the car load.

The simplified odometry model described in section 3.1.2, while being further from the reality of how the car moves, benefited greatly from only being dependent on the wheel circumference of the two rear wheels of the car. This made the model easier to optimize for the specific conditions of the test runs in this thesis, as well as less susceptible to changed preconditions. Overall, this model provided relatively accurate odometry, fully sufficient for the use case in this thesis.

Considering the poor quality of the data from the ultrasonic sensors, we were able to produce a fully usable occupancy grid in both our test runs. The key issue was to mitigate the impact of the seemingly random readings that the ultrasonic sensors produce when there is no obstacle within their range. The method of decreasing the probability for cells marked as free by sensor readings, in conjunction with the radial modulation described in section 3.3.1, handled this sufficiently well.

SLAM depends heavily on scan matching of high resolution lidar scans. Because of the poor quality of the ultrasonic sensor data, as well as the nature of ultrasonic sensors in general with their wide opening angles, low update frequencies, and low maximum ranges, it is infeasible to directly use their data as input to a SLAM algorithm.

In order to construct a setting where ultrasonic sensors can be used in conjunction with SLAM, as described in section 3.4, we relied on the assumption about a known structure of the environment we wanted to detect; straight obstacles to the left and the right of the car. From this, we pre-processed the ultrasonic sensor data to fit the assumed obstacles before using it as input for the SLAM algorithm.

Figure 4.9 shows the results when testing this in the first few section of test run 2, where our car stands on a parking lot between two other cars. It worked well here, but further work is needed for the solution to be truly useful. The most pressing issue in our minimal implementation is that when the car has backed out of the parking lot far enough for the rear side-facing ultrasonic sensors to be behind the cars to the side, the assumption of straight obstacles beside the car breaks down. If we want to keep track of our car in relation to the other cars throughout the whole parking scenario, we need to detect when this happens and switch over to only using the front sensors that are still detecting the other cars.

Another thing worth noting is that the map produced by the SLAM algorithm in this scenario will be a representation of our assumption of straight obstacles beside the car, not the real environment. If we want to create a map of the environment, we need to fall back on the mapping methods used in section 3.3 and only use the straight wall constructions for correcting the car's pose.

# References

[1] CAN BUS tools — cantools 34.0.0 documentation. https://cantools.readthedocs.io/en/latest/. [Online; accessed 8-June-2020].

[2] cartographer - ROS Wiki. http://wiki.ros.org/cartographer. [Online; accessed 8-June-2020].

[3] Dimensions | XC90 | Volvo Cars. https://www.volvocars.com/intl/cars/new-models/xc90/specifications/dimensions. [Online; accessed 8-June-2020].

[4] Models - XC90 - Technical Data - Volvo Cars Global Media Newsroom. https://www.media.volvocars.com/global/en-gb/models/xc90/2017/specifications. [Online; accessed 8-June-2020].

[5] OpenSLAM.org - gmapping. https://openslam-org.github.io/gmapping.html. [Online; accessed 8-June-2020].

[6] Puck™. https://velodynelidar.com/products/puck/. Library Catalog: velodynelidar.com.

[7] ROS API reference documentation — Cartographer ROS documentation. https://google-cartographer-ros.readthedocs.io/en/latest/ros_api.html#required-tf-transforms. [Online; accessed 8-June-2020].

[8] rosbag - ROS Wiki. http://wiki.ros.org/rosbag. [Online; accessed 8-June-2020].

[9] ROS.org | Powering the world's robots. https://www.ros.org/. Library Catalog: www.ros.org.

[10] tf - ROS Wiki. http://wiki.ros.org/tf. [Online; accessed 8-June-2020].

[11] SungHwan Ahn, Jinwoo Choi, Nakju Lett Doh, and Wan Kyun Chung. A practical approach for EKF-SLAM in an indoor environment: fusing ultrasonic sensors and stereo camera. *Autonomous Robots*, 24(3):315–335, April 2008.

[12] Ilze Andersone. Probabilistic Mapping with Ultrasonic Distance Sensors. *Procedia Computer Science*, 104:362–368, January 2017.

[13] Luigi D'Alfonso, Antonio Grano, Pietro Muraca, and Paolo Pugliese. A polynomial based SLAM algorithm for mobile robots using ultrasonic sensors - Experimental results. In *2013 16th International Conference on Advanced Robotics (ICAR)*, pages 1–6. IEEE, November 2013.

[14] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, June 2001. Conference Name: IEEE Transactions on Robotics and Automation.

[15] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, 13(2):99–110, June 2006.

[16] Maksim Filipenko and Ilya Afanasyev. Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment. In *2018 International Conference on Intelligent Systems (IS)*, pages 400–407. IEEE, September 2018.

[17] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23(1):34–46, February 2007.

[18] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-Time Loop Closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[19] Shoudong Huang and Gamini Dissanayake. Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM. *IEEE Transactions on Robotics*, 23(5):1036–1049, October 2007. Conference Name: IEEE Transactions on Robotics.

[20] Edouard Ivanjko, Ivan Petrovic, and Misel Brezak. Experimental Comparison of Sonar Based Occupancy Grid Mapping Methods. *Automatika: Journal for Control, Measurement, Electronics, Computing & Communications*, 50:65–79, December 2009.

[21] Patrick Lazik, Niranjini Rajagopal, Oliver Shih, Bruno Sinopoli, and Anthony Rowe. ALPS: A Bluetooth and Ultrasound Platform for Mapping and Localization. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, pages 73–84, Seoul, South Korea, November 2015. Association for Computing Machinery.

[22] Kooktae Lee, Woojin Chung, and Kwanghyun Yoo. Kinematic parameter calibration of a car-like mobile robot to improve odometry accuracy. *Mechatronics*, 20(5):582–595, August 2010.

[23] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fast-SLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. *Aaai/iaai*, 593598, 2002.

[24] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. 593598:1151–1156, 2003.

[25] Kai Ni, Drew Steedly, and Frank Dellaert. Tectonic SAM: Exact, Out-of-Core, Submap-Based SLAM. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1678–1685. IEEE, April 2007. ISSN: 1050-4729.

[26] Amit Kumar Pandey, K. Madhava Krishna, and Henry Hexmoor. Feature Chain Based Occupancy Grid SLAM for Robots Equipped with Sonar Sensors. In *2007 International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pages 283–288. IEEE, April 2007.

[27] Randall C. Smith and Peter Cheeseman. On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, December 1986. Publisher: SAGE Publications Ltd STM.

[28] Sebastian Thrun and Michael Montemerlo. The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research*, 25(5-6):403–429, May 2006.

[29] Peng Wang, Zonghai Chen, Qibin Zhang, and Jian Sun. A loop closure improvement method of Gmapping for low cost and resolution laser scanner. *IFAC-PapersOnLine*, 49(12):168–173, January 2016.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-06-15

**EXAMENSARBETE** SLAM in Low Speed Scenarios Using Ultrasonic Sensors
**STUDENT** Mattias Jonsson
**HANDLEDARE** Elin Anna Topp (LTH), Magnus Wendt (Volvo Cars)
**EXAMINATOR** Jacek Malec (LTH)

# SLAM i låghastighetsscenarier med ultraljudssensorer

POPULÄRVETENSKAPLIG SAMMANFATTNING **Mattias Jonsson**

Detta arbete undersöker möjligheten att använda data från ultraljudssensorer för att hjälpa föraren av en personbil i låghastighetsscenarier såsom parkering. Möjligheten att använda sensorerna för Simultaneous Localization and Mapping undersöks och diskuteras.

Självkörande bilar har de senaste åren visat sig vara en av de största nya utmaningarna för bilindustrin. Våra personbilar får fler och fler avancerade funktioner. Allt från avancerade säkerhetsfunktioner såsom automatisk inbromsning om ett barn springer fram framför bilen, till att automatisera monotona moment såsom motorvägskörning eller fickparkering, till försök att realisera visionen om en helt självkörande bil.

Gemensamt för de flesta av dessa funktioner är att bilen på något sätt måste vara medveten om sin omgivning. Det allra vanligaste är att använda kameror eller ljusradar, LIDAR, för att med hög precision och upplösning skanna omgivningen. Med dessa blir resultatet ofta bra, men sensorerna är både dyra och komplexa. Ett alternativ är ultraljudssensorer. Ultraljudssensorer är billiga och finns redan installerade på de flesta nya bilar som säljs idag. De har dock väsentligt lägre räckvidd, upplösning och precision än kameror och LIDAR och används idag främst till enkla funktioner såsom att varna om föraren backar för nära ett hinder.

I mitt examensarbete har jag undersökt möjligheten att använda data från ultraljudssensorer för mer avancerade hjälpmedel för föraren. En occupancy grid; en karta över en omgivning med information om var det finns hinder och var bilen är fri att köra, byggs. Utifrån detta undersöks möjligheten att använda data från ultraljudssensorer i Simultaneous Localization and Mapping; SLAM. SLAM är problemet att bygga en karta av en omgivning med hjälp av sensorer på bilen och samtidigt bestämma bilens position i denna karta. Det beskrivs typiskt som ett hönan och ägget-problem. Man behöver veta bilens position för att bygga kartan och man behöver ha en karta för att veta bilens position. Det finns ett antal algoritmer som kan approximera en lösning på detta problem, men i dessa implementationer används nästan uteslutande LIDAR för att skanna omgivningen.

Resultaten visar att det går att med hjälp av ultraljudssensorer bygga en karta av en trång omgivning, till exempel ett parkeringshus, med en tillräcklig upplösning och kvalité för att hitta och navigera till en ledig parkeringsplats. Det går även att i begränsade fall använda datan från ultraljudssensorerna med SLAM för att till exempel parkera på eller lämna en trång parkeringsplats utan att köra in i något hinder.