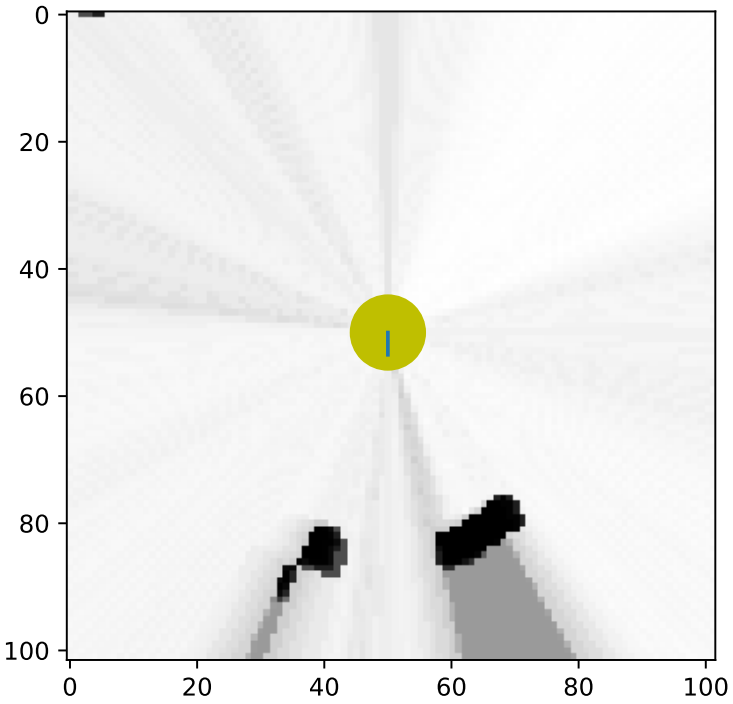


Real-time Object Detection and Tracking

Real-time Object Detection and Tracking using Sensor Fusion of LiDAR, Radar, and Camera sensors

Master's thesis in Systems, Control and Mechatronics

Johan Ivarsson
Per Wiklund

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

MASTER'S THESIS 2019

Real-time Object Detection and Tracking

Real-time Object Detection and Tracking using Sensor Fusion of LiDAR, Radar, and Camera sensors

Johan Ivarsson
Per Wiklund

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Real-time Object Detection and Tracking
Real-time Object Detection and Tracking using Sensor Fusion of LiDAR, Radar, and Camera sensors
Johan Ivarsson
Per Wiklund

Examiner: Lars Hammarstrand, Department of Electrical Engineering

Master's Thesis 2019:NN
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: **Bird view of platform in a certainty grid.**.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2019

Real-time Object Detection and Tracking
Real-time Object Detection and Tracking using Sensor Fusion of LiDAR, Radar, and Camera Sensors
Johan Ivarsson
Per Wiklund
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous solutions are a major field of research in modern society, and an important branch of such systems is collision avoidance. Collision avoidance is often based on multiple subsystems determining the environment and possible dangers in its surroundings. This project seeks to develop such an environment, based on object detection and tracking using sensor fusion.

In this thesis the environment was created as an occupancy grid using LiDAR, Radar, and ultrasonic sensors. This grid was then filtered using a Gaussian filter in order to determine objects within it. Thereafter, images from a camera were used in combination with the convolutional neural network tiny YOLOv3 in order to classify the objects within the grid. Lastly, when having determined a set of objects inside the grid a Kalman filter was used to track the objects.

The goal of the thesis is to be able to run the developed algorithm in real-time on a platform resembling an autonomous car. Therefore, a real-time test case was developed for the platform, along with a simple collision avoidance algorithm, allowing the platform to stop and restart. This platform came equipped with communications software in addition to software for the LiDAR, ultrasonic, and camera sensors. The test case consisted of four objects, which the platform was supposed to avoid based on information in the occupancy grid. This test case is also used to determine the importance of each sensor in the system and how well the system is able to track objects in real-time.

The hardware on the platform proved to be a limiting factor of the performance. Thus, the developed algorithm had to be run on an external computer and communication with the platform was done via WIFI. Using an external computer, a time per iteration of roughly 0.5 seconds was achieved for an occupancy grid resolution of 100x100 cells. The external computer and the poor quality of the sensors makes the system difficult to compare with state-of-the-art solutions. Despite this, the platform could complete the developed real-time test case. From the test case it could be concluded that the ultrasonic sensor provides no additional information compared to the Radar and could thus be excluded from the system.

Keywords: Autonomous solutions, Collision avoidance, Sensor fusion, LiDAR, Radar, Ultrasonic, Convolutional neural network, Kalman filter, Occupancy grid, YOLOv3.

Acknowledgements

We would like to thank Olle Norelius and Gabriel Wagner at Infotiv, for the helping hands they have offered during the project and their willingness to answer all of our excellent and not so excellent questions. Furthermore, a very special thanks is extended to Carl Wagné and Örjan Hagberg for their valuable feedback on the report and their ability to make event the dullest of moments a joy!

Johan Ivarsson, Gothenburg, June 2019
Per Wiklund, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1

Introduction

Autonomous systems are becoming more and more widely spread throughout society. Applications ranging from simple lawnmowers to self driving cars are constantly under research in order to take the next step in automation. At the moment, many car manufacturers around the world are researching the field of self driving cars with a goal of being the first with a fully autonomous car. To achieve this a number of intelligent subsystems have to be working close to perfection, both single-handedly and together as a lager system. Many of these subsystems in turn require several different sensors complementing each other to achieve as accurate reading of its surroundings as possible. Some of the most commonly used sensors are multi-purpose camera, short and long range Radar, LiDAR, Ultrasonic, GPS, and odometer sensors.

The various sensors used in autonomous systems perform differently depending on environment and application. Therefore, the usage of complementary sensors is vital in order to create a robust system, which allows a single application to perform in multiple environments. The choice of sensors in a certain application is far from obvious. For instance, Tesla motor believes autonomous cars are fully functional without data from a LiDAR and instead can rely more heavily on Radar and camera information [2].

1.1 Background

Even though the topic of fully autonomous cars is highly researched, there seems to be quite some time before a fully autonomous car will be launched. It is not difficult to fathom the complexity of the entire system which is a fully autonomous car. With subsystems such as collision avoidance, self localization, multiple object tracking, and many more to fuse into a single, major system is not an easy task.

Another important aspect and difficulty in such systems today, is enabling them to run in real-time. This is essential in order to deploy such systems outside testing ranges, as real world scenarios are prone to rapid changes. For instance, if a pedestrian walked in front of an autonomous car the response time before the car registers it and stops is crucial to ensure safety.

However, as stated, progress is being made and systems such as *ADAS* (*Advanced driver-assistance systems*) are becoming increasingly intelligent every year. This is enabled by progress within different techniques such as *simultaneous localization and mapping*, and *multi-sensory data fusion*. The latter of these two examples seek to combine a set of different sensors' read-

ings into a common framework and is more commonly referred to as *sensor fusion* [3]. This framework is often created as a map or grid, which is then commonly used as a basis in different applications such as object detection. Many of these roles then come to play vital parts in larger systems. For instance, as object detection plays a major role in collision avoidance. Therefore, the importance of these maps cannot be understated and as sensor fusion is a frequently used tool when creating them, a deep understanding of it is required.

Which set of sensors to use in what application is a key aspect, but also how the information from different sensor is combined. Some sensors are light sensitive, whilst others struggles when the environment is dark. The objects detectable by some, are not as easily detected by others. Examples of such an object is transparent ones, which a LiDAR would have difficulties detecting, but an ultrasonic sensor would not.

This project seeks to create a map covering the surrounding world of an autonomous platform resembling a miniature self driving car using sensor fusion. This map is created using LiDAR, ultrasonic, Radar, and camera as the set of sensors. Thereafter, the map is used as a basis for a object detection and tracking algorithm also using the aforementioned sensors. All of this is done in real-time.

1.2 Purpose

The aim of the project is to develop an algorithm that aids the decision making in collision avoidance by providing multiple object detection and tracking in real-time. The algorithm is to calculate width of and distance to detected objects, while also classifying them and displaying the confidence level of the classification. The detection and classification part of this project is formulated as a Sensor fusion problem, where the sensors used are LiDAR, Radar, ultrasonic and camera. The algorithm is implemented on a small autonomous platform, with the goal of making decisions based on detected objects in real-time.

When being able to detect and classify objects using the sensors mentioned, the ability to perform multiple object tracking in real-time is to be implemented. This is done through filtering. Therefore, the project will investigate the suitability of different motion models and filters in real-time applications, using the autonomous platform as a test vehicle.

1.3 Problem Description

This section aims to clarify the premises and the final product of this project. It will shortly motivate the project from the point of view of the company where the project is held, *Infotiv*. Furthermore, specifications regarding the sensors used and some software already implemented on the autonomous platform is presented. Lastly, the main questions to answer is stated in a short manner.

1.3.1 Problem introduction

This project will design a real-time object detection and tracking system on an autonomous platform, resembling a self driving car. While there exists a system today based on a 2D, single layer LiDAR and a camera, it is not optimal. The object detection algorithm is not sufficient to ensure robust collision avoidance, because it lacks the ability to see transparent objects and those under the LiDAR's layer of view. Therefore, the main idea of the project is to design a new system using additional information from different sensors. The sensors introduced are ultrasonic sensors and two stationary Radar sensors. By introducing these sensors the new system will be capable of detecting a larger variety of objects. In order to improve the tracking ability of the existing system, a Kalman filter will be implemented.

1.3.2 Problem Details

The current LiDAR and camera system uses the neural network structure *YOLOv2* to classify objects within the field of vision and keeps track of them using the LiDAR's distance and angle measurements. This results in rough bounding boxes around objects in the images from the camera, with a corresponding certainty in its classification. The tracking is then performed by using distances between objects in adjacent iterations. This allows for a rough tracking of objects, however it blindly follows the measurements provided. This could result in less accurate predictions than would the measurements be compared to some kind of model. Furthermore, objects which the LiDAR and camera cannot detect are invisible to the autonomous platform.

Therefore, this project seeks to deliver a sensor fusion algorithm which outputs bounding boxes around detected objects with a certainty of the classification, complemented by an occupancy grid that shows the position, size and prediction of the objects. This is to be achieved using the sensors defined in section A.1 in appendix A.

When considering tracking of detected objects a Kalman filter will be used. Therefore, issues regarding motion models and linearity have to be addressed. This will be done by implementing a linear Kalman filter along with a linear model and a nonlinear Kalman filter with a nonlinear model. Here the trade-off in gained performance and computational complexity is key in determining the most suitable filter to implement.

As the the system is to run in real-time, some constraints on the used methods exists. Namely, that the computational complexity has to be taken into account at all times. The trade-off in precision and the time to run an iteration of the entire algorithm has to be evaluated. As the performance of the entire system will be dependent on that time.

Furthermore, a real-time test case is to be developed. The purpose of the test is to validate the systems behavior and performance. In order to perform the test a simple collision avoidance algorithm will be developed, which allows the platform to stop and restart.

1.3.3 Aim

The following questions will be investigated during the project:

- What is the increase in robustness, regarding object detection, by introducing distance measurements from ultrasonic and Radar sensors to compliment a LiDAR and camera? Are any of the information sources redundant? Can the redundant sensor be excluded from the system?
- How can the width and distance information of a detected object help with the classification of it? Is it good enough in order to navigate through darker areas where the camera is unreliable?

Furthermore, the project is to perform tracking of the detected objects in real-time. This is to be done using a suitable Kalman filter.

1.4 Limitations

This master's thesis is bound to some limitations in different forms. A major limitation on the system arise as it is designed for real-time usage. This limits the project to rely on less computational heavy techniques and models. Since all the computations are performed on an external computer, there is a slight delay that is introduced when information is sent via WIFI.

When it comes to the sensors of the platform, the project is limited to work with low-end sensors. To be more specific, the LiDAR uses a single layer and is therefore incapable of detecting objects beyond those on the same height as the sensor. The two Radar sensors are able to detect multiple objects but on the other hand unable to obtain a reasonable angle to the detection.

1.5 Related work

There exists some different solutions when combining grid mapping with multiple object tracking. One of the more known are the simultaneous localization, mapping and moving object tracking (SLAMMOT) which was used by Wang et al, [4]. An extended variant of the SLAMMOT is the SLAMMOT-SP which also takes scene prediction into account. This was tested by Chung et al, [5] which resulted in increased performance in an indoor environment. However, the method struggled with the prediction of irregular structures.

Another solution for real-time tracking applications using Occupancy grid was tested by Nuss et al, in [6] where they represent each cell as a random finite set of particles. This was performed together with a probability hypothesis density / multi-instance Bernoulli (PHD/MIB) filter. This solution proved to be able to run in real-time with sufficient computational power. Another proposed approach to mapping and tracking simultaneously was tested by Vu et al, [7] where they combine SLAM with detection and tracking of moving objects (DATMO). They also track the objects with a multiple hypothesis tracker (MHT) together with an adaptive interacting multiple model (IMM) filter.

The method used in this thesis has been inspired by the mentioned solutions and is based on an occupancy grid that is designed with the Bayesian inference theory framework. The occupancy grid will act as the main source of information where all the individual sensors are combined. Tracking of objects will be performed with aid of a Kalman filter. Since it is desirable that the algorithm can run in real-time on a limited amount of computational power, many of the individual approaches to sub tasks has been taken with performance in regard to the run time. What differs this thesis project from most others are that it is performed in real-time for small scale systems.

1.6 Thesis outline

After this introduction, the thesis is structured as follows. In chapter 2 a theoretic background for the concepts and methods used throughout the project is presented. The idea with this chapter is to aid the reader with useful knowledge in the topics concerned. The method for this project is divided into four different chapters. Firstly, chapter 3 introduces the system architecture of the platform this project is applied on. Secondly, chapter 4 will introduce the occupancy grid and how it is interpreted. Thirdly, Chapter 5 is about how the created occupancy grid is coupled with the object classification that is obtained with a camera sensor. Lastly, chapter 6 presents different filtering models and corresponding states. The results for the project is then presented in chapter 7, which in turn is discussed in chapter 8. The thesis is then summed up in a conclusion in chapter 9.

2

Theory

This chapter will present relevant theory to the different parts of the project. Namely data association, machine learning, Bayesian filtering, and sensor fusion at different levels. The theory aims to ease the understanding of the content of this report, with more emphasis on theories with higher relevancy to this project. Therefore, some sections are presented without the same deep dive as in others.

2.1 Data Association Theory

Data association is a method to define sub clusters from a larger cluster. A cluster here refers to a cluttered environment based on measurements from sensors. In other words, data association problems seek to, from a cluttered environment, define what measurements correspond to what targets [8]. There are numerous approaches to perform data association and this section will cover two of the most common.

2.1.1 Nearest Neighbors

Nearest neighbor is the simplest technique for data association. The nearest neighbor approach is based on the assumption that adjacent measurement points in a cluster are related. Therefore, nearest neighbor associates measurement points based on distances to nearby points. This distance is often an absolute distance or the euclidean distance, but it could also be some other statistical distance function [8]. Moreover, as nearest neighbor is the simplest data association method there exists some disadvantages. For instance, as it is based on statistical distance measurements, it suffers from error propagation when applied to dense clusters or noisy measurement points.

2.2 Machine Learning

Machine learning solutions in various applications are rising trends. One of the most common applications of machine learning is image recognition and object detection in images. In this project object detection in images is done using a *convolutional neural network*. This section presents the main concepts behind neural networks and then introduces convolutional neural networks.

2.2.1 Neural Networks

A neural network is a tool for *machine learning*. These can be designed as a *supervised* machine learning tool. This in turn means that a neural network can be trained to perform classification with aid of external knowledge. During the training stage it is told when it is correct, when it is wrong, and what the correct classification would have been.

A neural network is a network composed of artificial neurons, which in turn create different *layers*. For a vanilla neural network the most common structure of layers is an *input layer*, an *output layer*, and a number of *hidden layers*. When all neurons between two consecutive layers are connected the second layer is called a *fully connected layer*. If all layers are fully connected layers the network is called a *fully connected neural network*. Furthermore, these networks are known as *feedforward neural networks* [9]. An example of such a neural network is shown in figure 2.1.

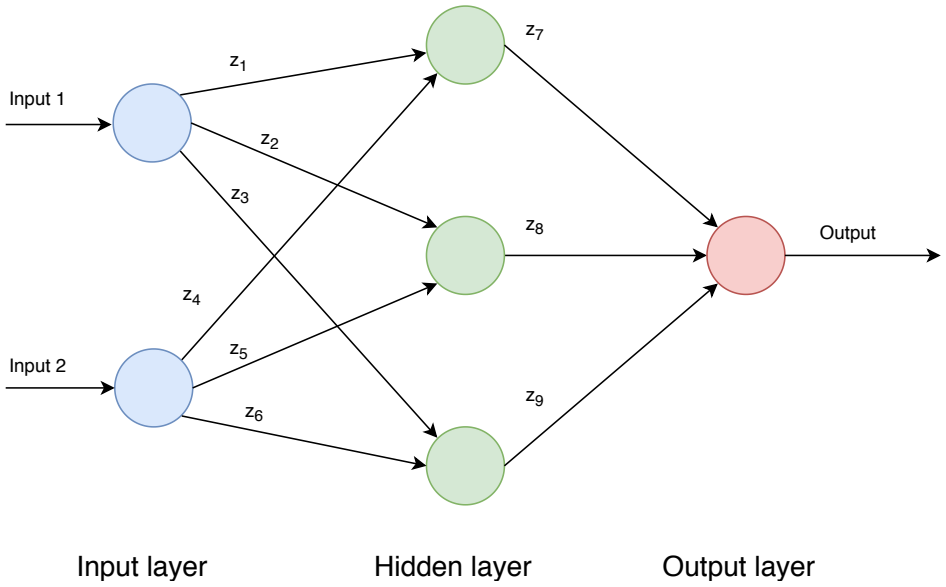


Figure 2.1: The figure illustrates an example of a fully connected feedforward neural network. The input layer is denoted as blue, the single hidden layer is marked as green and the output layer is marked as red. The parameters z_1 to z_9 are the outputs of each layer.

The input to a neuron in a neural network is a vector based on the number of neurons in the previous layer. This input is denoted z_k for the *k:th* layer in a neural network. The neuron, which consists of a set of adjustable parameters W, known as the *weights*, and b, known as the biases, then calculates its output accordingly:

$$z_{k+1} = F(Wz_k + b) \tag{2.1}$$

z_{k+1} is derived using an *activation function*, F, which adds nonlinear features to the linear set of equations $Wz_k + b$ [10].

2.2.2 Activation function

A feedforward neural network usually seeks to fit a polynomial to a certain set of data. A linear neural network would only be able to use polynomials of order one. Therefore, the introduced nonlinearity of the activation function is essential in order to perform complex fitting to data [11].

There are multiple different activation functions being used in different neural networks today. However, a common type of activation function is the sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

The sigmoid function is commonly used when performing for instance binary classification. Another activation function commonly used in binary classification is the *ReLU* function:

$$ReLU(x) = max(0, x) \tag{2.3}$$

Furthermore, *tanh(x)* is frequently used in many neural network application. However, when performing multiple classification the *softmax* function is often chosen, and follows as:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.4}$$

Here the activation function seeks to determine a probability associated with each class, x_i, [11].

2.2.3 Training a neural network

The adjustable parameters of each layer is referred to as weights, W, and biases, b. It is these weights and biases which are sought to be optimized in order to achieve a desired behavior of the neural network. This is done through *training*. Training means that a neural network is fed with inputs and then the outputs are compared to a ground truth. The parameters are then updated with an optimization function, and a common choice is stochastic gradient descent, which updates the parameters accordingly:

$$W_+ = W - \alpha \frac{\delta E(W)}{\delta W} \tag{2.5}$$

Where E a loss function, which is sought to be minimized, corresponding to the difference of each output compared to its ground truth. This is applied to every weight. The biases are updated according to (2.5) however W is substituted to b. This process is referred to as *backpropagation* [11]. In (2.5) the parameter α is the *learning rate* and determines the step size during training.

2.2.4 Convolutional Neural Networks

Convolutional neural networks (*CNNs*) are highly successful when processing grid-like data such as images. These neural networks are also common in many different applications today [11]. CNNs operate using *convolutional layers*, which utilizes the linear operation *convolution* instead of matrix multiplication. This is done by applying a *weighting function* in order to average over a certain input.

2.2.4.1 Convolutional layer

The weighting function in a convolutional layer is sometimes referred to as a kernel and the output of such a layer is commonly called the layer's *feature map* [11]. The kernel in a CNN layer is often designed as a *sliding window classifier*. This sliding window is then convoluted with the input to the convolutional layer. It is the size of the kernel, which then determines the number of weights in the layer should there only be one output from the layer. If multiple outputs are desired, a number of kernels have to be used, which in turns makes the total number of weights determined by the number of kernels as well. However, the sliding window technique allows for less weights than should a neural network using matrix multiplication have been applied to a similar input. This is called *sparse weights* or *shared weights* and is a reason why convolutional neural networks scale well with the number of layers, *depth*, [11].

CNNs are not bound to be designed using the sliding window approach. Another approach is to use kernel sizes equivalent to input size of the convolutional layers. This is, compared to the sliding window technique, more computationally expensive as convolution is performed at a higher resolution.

2.2.4.2 Pooling layer

Thus far only the convolutional layer has been introduced. However, other important layers in CNNs are *pooling layers*, which feature upsampling and downsampling of an input. There are multiple types of pooling layers, however when downsampling the *maxpooling layer* is one of the more common.

The max pooling layer uses a kernel, or sliding window, in order to extract the highest value within the window's size. The layer can then be designed to skip a certain amount of the input by implementing a *stride* (step size) [11]. The maxpool method is depicted in figure 2.2.

As maxpooling is one of many downsampling methods, there exists a couple of methods for performing upsampling. This is known as *unpooling*. Two common unpooling techniques are *bed of nails* and *nearest neighbor*. Both methods are based on padding techniques and the difference can be viewed in figure 2.3 and figure 2.4.

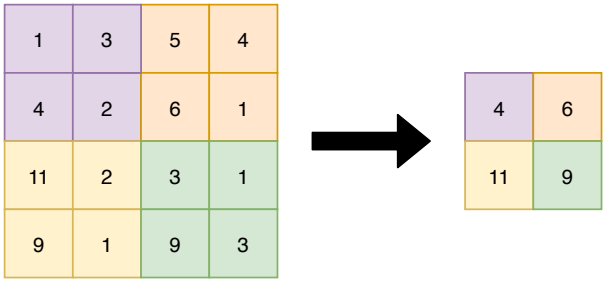


Figure 2.2: The figure illustrates the effect of a maxpool layer on an input. The maxpool layer has a kernel size of 2x2 and a stride of 2. The output is the map to the right of the arrow.

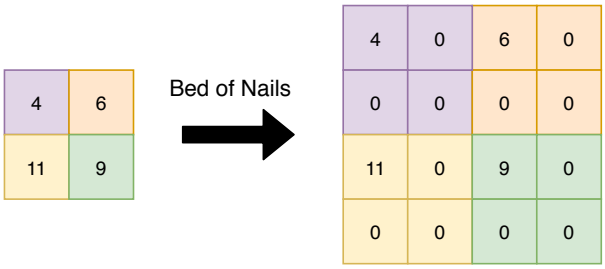


Figure 2.3: The figure illustrates an the effect of a bed of nails unpooling layer on an input. The bed of nails layer has a kernel size of 2x2. The output is the map to the right of the arrow.

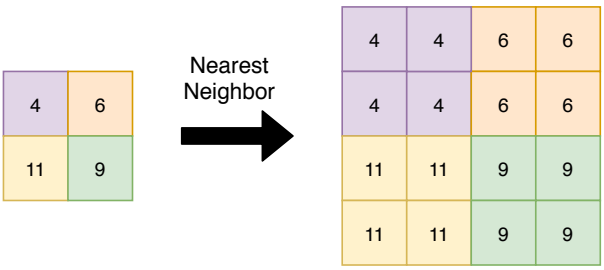


Figure 2.4: The figure illustrates an the effect of a nearest neighbor unpooling layer on an input. The nearest neighbor layer has a kernel size of 2x2. The output is the map to the right of the arrow.

The similarities and differences between the two unpooling layers are clear from figure 2.3 and figure 2.4. Namely that where the bed of nails layer adds zeros the nearest neighbor adds the input value to the entire unpooled area.

2.3 Image Recognition using CNNs

Some common applications for CNNs are object detection, classification and semantic segmentation in images. Semantic segmentation is the process of labeling every pixel of an image, while object detection and classification aims at classifying (and localizing) an area inside an image. The latter is often the approach used in object detection systems using multiple sensors. Since it is less computationally expensive to classify parts of an image instead of its entirety.

YOLO

YOLO (short for *You only look once*) is an object detection model using deep convolutional neural networks. The model uses a single CNN, trained on a set of images, to predict bounding boxes with corresponding class probabilities in input images [1]. The classes are determined using a softmax activation function in the output layer. Furthermore, the number of boxes predicted is determined by a threshold based on the confidence level of a prediction. Namely that bounding box predictions with a class probability lower than a certain value are discarded.

An advantage of YOLO is that it is designed to minimize the time to process an image and find detections. Apart from many other solutions YOLO formulates the detection issue as a regression problem [1]. The output from a YOLO network consists of a label corresponding to a detected object's class and a confidence level for the detected object based on IoU (intersection over union). Along with these outputs, the coordinates for the bounding box is given, which is useful when computing the height and width of the detected objects [1]. This makes it a common network in real-time applications, such as collision avoidance systems.

2.4 Bayesian theory

Bayesian theory is based on the probability of a specific event *A* being dependent on prior knowledge of conditions related to the event. By giving information about conditions correlated to events, a more accurate prediction of said event can be achieved. A simple example of Bayesian theory is that the age of a sick individual influences the probability of which disease is causing it. Mathematically, Bayes' theorem is stated as:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{2.6}$$

Here $P(A)$ is the probability of event *A* and $P(B)$ is the probability of event *B*. $P(A \mid B)$ is the conditional probability of event *A* given that event *B* is true and $P(B \mid A)$ is the opposite.

2.4.1 Bayesian Filters

A Bayesian filter utilizes Bayesian theory in order to recursively estimate the current value given past and current measurements. By observing the current measurement in relation to its priors a higher accuracy in the new estimate can be achieved. With all the available measurements $y_{1:k}$, the new estimate x_k can be determined by calculating the posterior $P(x_k \mid y_{1:k})$ as in (2.6). This filtering is closely related to both prediction and smoothing. While filtering estimates the current true value, prediction estimates the future value, and smoothing corrects the old estimates by using future data. [12]

2.4.2 The Kalman filter design

A common *Bayesian filter* is the *Kalman Filter*, which seeks to recursively calculate a state of an object based on measurements and a prior state. An initial step when designing a Kalman filter is to determine the states x, a model describing their change between two consecutive time steps, and a noise model describing the uncertainties. This is commonly referred to as a *motion model*. Furthermore, the measurement model has to be determined. This is a model describing the relation between the measurement y and the current state vector x, along with a measurement noise model. This model describes the uncertainties in the measurements. Together these two models describe the system accordingly:

$$x_k = f(x_{k-1}) + q \tag{2.7}$$

$$y_k = h(x_k) + r \tag{2.8}$$

The motion model would in this case include x, a model for f, and an estimation of the process noise q. The measurement model should on the other hand only contain h, and an estimation of the measurement noise r. If both of these models describe linear systems the *linear Kalman filter* can be applied.

However, should any of the two models describe a nonlinear system, i.e. the system's order is not 1, the linear Kalman filter is often an impossible choice and a different filter type has to implemented . Examples of such filters are the *Extended Kalman filter* and the *Unscented Kalman filter*. These filters will be discussed in section 2.4.2.2 and 2.4.2.3. First, however, the linear Kalman filter will be looked further into.

2.4.2.1 Linear Kalman filter

The Kalman filtering procedure can be divided into two parts, the prediction step and the update step. The prediction step can be formulated as:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} \tag{2.9}$$

$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q \tag{2.10}$$

Here A is the state transition matrix and corresponds to f in (2.7), Q is the process covariance model, $\hat{x}_{k-1|k-1}$ is the prior state estimation of the state, x_{k-1}, and ¶$_{k-1|k-1}$ is the covariance matrix of the prior estimation. $\hat{x}_{k|k-1}$ can be viewed as the initial guess of the next state estimate and $P_{k|k-1}$ the corresponding covariance matrix. The A is commonly designed such that $P_{k|k-1} > P_{k-1|k-1}$, which means that there is a larger uncertainty in the guess of the estimation than in the prior. In order to refine the estimation in the prediction step, the update step is performed. The equations of the update step follows as:

$$v = y_k - H\hat{x}_{k|k-1} \tag{2.11}$$

$$S = HP_{k|k-1}H^T + R \tag{2.12}$$

$$K = P_{k|k-1}H^T S^{-1} \tag{2.13}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + Kv \tag{2.14}$$

$$P_{k|k} = P_{k|k-1} - KSK^T \tag{2.15}$$

Here H is the matrix relating the states to the measurements, and can be viewed as h in (2.8), and R is the covariance matrix of the measurement model. $\hat{x}_{k|k}$ is the state estimation of the posterior state x_k and y_k is a measurement corresponding to x_k. Furthermore, S is the predicted covariance in y_k, K is the kalman filter gain, and $P_{k|k}$ is the covariance of estimation $\hat{x}_{k|k}$. v is the innovation matrix, which is the difference between the estimate from the prediction step and the measurement. The update step reduces the covariance in the estimation once again ($P_{k|k} < P_{k|k-1}$) by comparing with a measurement of the predicted state estimation, $\hat{x}_{k|k-1}$ [13].

2.4.2.2 Extended Kalman Filter

The Extended Kalman filter (*EKF*) is a method to solve state estimations for nonlinear models, under the assumption that the states and measurements are Gaussian random variables. This is performed as an optimization problem using a recursive maximum likelihood estimation (*MLE*). EKF is applied to systems on the form described by (2.7) and (2.8). Where either the motion model or the measurement model, or both, describe a nonlinear system. The EKF algorithm seeks to linearize these nonlinear models h and f as first-order Taylor Expansions around the current estimation of x_{k-1} [14].

2.4.2.3 Unscented Kalman Filter

The Unscented Kalman filter (*UKF*) is also a method for state estimation when the measurement or motion model is nonlinear. UKF uses samples of the prior state probability distribution, which as in the EKF case also is assumed to be Gaussian. However, UKF manages an estimate of the posterior with an accuracy corresponding to the third-order Taylor Expansion for any nonlinearity, compared to the first-order linearization of the EKF [15]. The concept of the UKF is based on an extension of the *Unscented transformation*. This is a method for statistical calculations of random variables propagated through nonlinear models, called σ-points and weights.

In the UKF these σ-points and weights are formulated using the prior state estimate, $\hat{x}_{k-1|k-1}$ and $P_{k-1|k-1}$ in the prediction step (and then using the predicted estimates $\hat{x}_{k|k-1}$ and $P_{k|k-1}$ in the update step) accordingly [15]:

$$\begin{aligned} \chi^0_{k-1} &= \hat{x}_{k-1|k-1} \\ \chi^i_{k-1} &= \hat{x}_{k-1|k-1} + \sqrt{n+\lambda}\sqrt{P_{k-1|k-1}} \\ \chi^{i+n}_{k-1} &= \hat{x}_{k-1|k-1} - \sqrt{n+\lambda}\sqrt{P_{k-1|k-1}} \end{aligned} \tag{2.16}$$

$$
\begin{aligned}
W^0 &= 1 - \frac{n}{n+\lambda} \\
W^i &= \frac{1}{2(n+\lambda)} \\
W^{i+n} &= \frac{1}{2(n+\lambda)}
\end{aligned}
\tag{2.17}
$$

Where *i* ranges from 1 to *n* which is the number of states in x_{k-1}. This creates a σ-vector of $2n+1$ σ-points with corresponding weights W, where the χ^0 is the prior $\hat{x}_{k-1|k-1}$, with the corresponding weight W^0. Propagating these σ-points through a nonlinear model gives the mean and covariance of the state estimation. This is done in the prediction step of the unscented Kalman filter, which follows as:

$$
\begin{aligned}
\hat{x}_{k|k-1} &= \sum_{i=0}^{2n} W^i F(\chi^i_{k-1}) \\
P_{k|k-1} &= \sum_{i=0}^{2n} W^i [\chi^i_{k-1} - \hat{x}_{k|k-1}][\chi^i_{k-1} - \hat{x}_{k|k-1}]^T + Q
\end{aligned}
\tag{2.18}
$$

Here Q is the process covariance, and f corresponds to the state transition function from (2.7). As in the Kalman filter case, the prediction step formulated in (2.18) is followed by an update step. This step is formulated accordingly:

$$
\begin{aligned}
\hat{y}_{k|k-1} &= \sum_{i=0}^{2n} W^i h(\chi^i_k) \\
P_{xy} &= \sum_{i=0}^{2n} W^i [\chi^i_k - \hat{x}_{k|k-1}][h(\chi^i_k) - \hat{y}_{k|k-1}]^T \\
S &= \sum_{i=0}^{2n} W^i [h(\chi^i_k) - \hat{y}_{k|k-1}][h(\chi^i_k) - \hat{y}_{k|k-1}]^T + R \\
K &= P_{xy} S^{-1} \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K(y_k - \hat{y}_{k|k-1}) \\
P_{k|k} &= P_{k|k-1} - K P_{xy}^{-1}
\end{aligned}
\tag{2.19}
$$

Important to note in the update step is that χ_k is σ-points based on the predicted distribution from (2.18). Furthermore, h corresponds to the model describing the measurements in (2.8), and R is the measurement covariance. P_{xy} is the cross covariance between the σ-points and the predicted state, and the same σ-points propagated through the measurement model and the resulted mean. The next state estimation $x_{k|k}$ and its covariance P_{k+1} is then derived using the Kalman gain K and the approximate covariance in measurements S.

2.5 Gaussian Filtering

Gaussian filters are common types of filters used to remove noise from data. This is done as a convolution between an input and a Gaussian function, which turns deterministic values into

probabilistic distributions. This convolution is also known as the Weierstrass Transform which is a smoothed version of an original function $f(x)$, which is obtained by averaging f with a Gaussian distribution. For the case where smoothing is applied to a two dimensional grid, the filter has to be applied in both directions which leads to the following equation:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.20}$$

Here, x and y are the distances from the origin of the distribution and σ is the standard deviation for the distribution. An illustration of a two dimensional Gaussian distribution can be seen in figure 2.5.

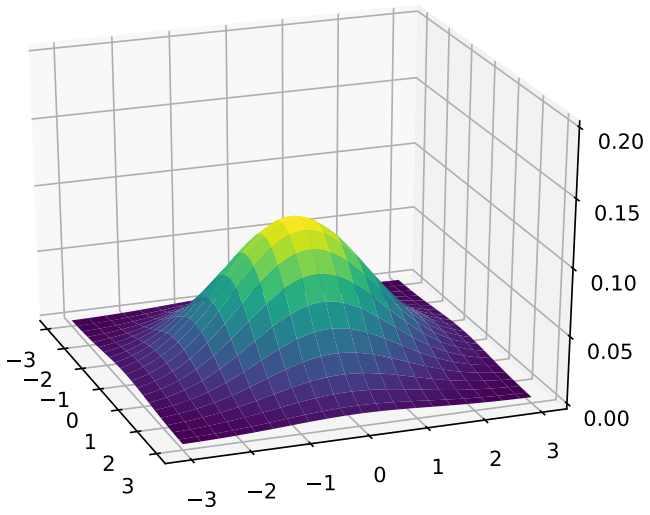


Figure 2.5: Illustration of a 2D Gaussian distribution.

2.6 Sensor Fusion

In order to increase the performance and robustness of sensor based systems one could combine the sensory information into one common information frame. This is known by a couple of different terms such as "sensor fusion", "data fusion", and "multi-sensor data fusion". Since there exists different words that roughly describes the same thing it quickly becomes hard to get a clear picture of what it actually means. It can vary from real-time sensor fusion for applications within the automotive industry to altitude estimation of an airplane. Moreover, it is important to distinguish between single sensor fusion where data from different time instances are fused together, and multiple sensor fusion where instead the fusion occur between different sensors at the same time instance [16]. In this report, the word sensor fusion will be used in order to describe the fusion of multiple sensors contributing to a common output.

Even though using sensor fusion generally results in an improvement of system performance, it also comes with some challenges that need to be addressed. Some of these challenges are Conflicting data, Data alignment, and Data association [17].

One way to further clarify what kind of fusion that is used in different cases is to separate it into a few categories. The Three-level Categorization is often used dividing sensor fusion into low, mid, and high-level fusion. The three different categories are also known as which type of fusion that occurs at that level and is in that case called, raw data fusion, feature level fusion, and decision fusion respectively [3]. A short introduction to each of the categories is given in the following sections.

2.6.1 Low-level sensor fusion

Low-level sensor fusion (LLF) is a very popular choice of sensor fusion due to the minimized information loss in the earlier stages. Here the decisions are happening in the last step. This includes the state of the system which could be positions, velocities, or the state of the surrounding map. To clarify this, LLF is used by combining raw measurement data from several sensors in order to achieve output data containing more information than the input ones. Articles like [18] shows that LLF outperforms higher levels of sensor fusion in certain cases due to the closer resemblance with the measured data.

2.6.2 Mid-level sensor fusion

Mid-level sensor fusion (MLF) which is also known as feature level fusion is based on combining different features of an object into a so called feature map. These features could for example be edges of an object or a frame that distinguish an object in an image. The features could either be given as a manual input to the algorithm or as an output from sensor data. This feature map can then be used in the segmentation part as well as detection of objects. MLF can struggle with contradicting features from different sensors which could result in worse performance.

2.6.3 High-level sensor fusion

High-level sensor fusion (HLF), also known as decision level fusion. At this level, decisions from different so called supervisors are combined in order to achieve a common decision. These decisions can, for example, originate from different subsystems in a vehicle. For instance, should a subsystem, A, recommend a vehicle to continue and another subsystem, B, recommends it to stop, the entire system have to respond accordingly. This is done by allowing some higher order system to fuse the recommendations in order to make a final decision depending on the situation. This kind of fusion is often said to be of the highest order when it comes to data fusion.

2.6.4 Multi-level sensor fusion

All these different levels can be used together when creating a sensor fusion architecture. There are several different architectures describing multi-level sensor fusion. Some of them are Boyd model, LAAS architecture, and the JDL fusion architecture. All of them are closely related with small difference in how the different layers are interpreted. Usually the sensor fusion will go from the lower levels first and then proceed upwards to the higher levels of fusion.

2.7 Occupancy Grid

An *Occupancy Grid* is a map describing the state of a finite area within the topic of probabilistic robotics. The derivation of an occupancy grid consists of creating maps based on noisy measurements. The Occupancy Grid defines a coordinate grid where each grid slot is evenly spaced and usually consists of a binary variables or boolean expressions. This means that each grid slot can either be occupied or not. An alternative approach to the occupancy grid is to make all the cells probabilistic. This turns the grid into a *Certainty grid*, which is a special case of an occupancy grid.

This special case contains more information than the regular occupancy grid and thus allow for better predictions. The derivation of an occupancy grid can be viewed as a filtering problems with two inputs, namely a measurement and a state. The literature presents several different frameworks for occupancy grid mapping where some of the more common are, Bayesian Inference Theory, Dempster-Schafer theory of Evidence, and Fuzzy-logic [19]. Of these three, the Bayesian Inference Theory method of updating an occupancy grid is the most straight forward and widely used. There are some advantages with using a more advanced framework such as the Dempster-Schafer. It improves the performance when it comes to contradictory information in comparison to the Bayesian Inference theory [20]. However, reports such as [21] found that the computational time difference between the two methods is significant and thereby is a key aspect when choosing framework.

2.7.0.1 Bayesian Inference Theory

The most commonly used probability framework for occupancy grids is the Bayesian Inference Theory. It calculates the posterior of the states in the occupancy grid, using Bayes rule in (2.6). The posterior state of the map is formulated as:

$$p(m|z,x) \tag{2.21}$$

Here m is the state of the map, z is the measurements and x is the grid's orientation. The prediction and update steps of the map thus follows as:

$$p(m_t|z_{1:t-1},x_{1:t-1}) = \int p(m_t|m_{t-1})p(m_{t-1}|z_{1:t-1},x_{1:t-1})dm_{t-1} \tag{2.22}$$

$$p(m_t|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_t, z_{1:t-1}, x_{1:t})p(m_t|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})} \tag{2.23}$$

Generally, it is quite difficult to estimate the dynamical part of the map which is described by $p(m_t|m_{t-1})$. It is however possible to make an assumption that the map is static at this point and later on compensate for it with a forgetting factor. This forgetting factor is a penalty for each of the cells such that old measurements eventually are discarded.

3

System Architecture

This chapter introduces the software structure and how information flow throughout the system. The sensor reading functions are referred to as servers, which are also briefly introduced. Furthermore, the occupancy grid is briefly introduced where the different levels of sensor fusion in the entire systems are stated. Lastly, a short introduction of how the Kalman filter is implemented is presented.

3.1 Software structure

The software of the system is divided into five main subsystems. These subsystems are based on already existing software structure and its information flow. By performing an initial system analysis the subsystems can be defined as following.

- An already existing car control system providing LiDAR, ultrasonic, and camera measurements. Furthermore, this is the system controlling the motors and uses a Raspberry Pi as computational unit. This system can be accessed via WIFI and thus algorithms are not required to be run on the Raspberry Pi.
- A Radar measurement server, which through socket communication can be subscribed to. The server provides numerous distance measurements. The number of measurements is based on how many peaks are above a certain threshold in the Radar signal's power spectrum.
- A LiDAR and ultrasonic measurement server, which is also subscribed to using sockets. This server is hosted on a separate socket compared to the Radar server. However, as the ultrasonic sensors provide no additional information compared to the Radar sensors it is initially excluded. When performing the real-time testing this assumption will be examined further.
- An image recognition server, subscribing to images from the vehicle and processes them using YOLO. However, as loading images require more CPU, it is desired to only access this stream when necessary. Therefore, the image stream is shut down after having processed an image.
- An occupancy grid which first fuses the data from the above mentioned measurement streams. This is a low-level sensor fusion system between the Radar and LiDAR subsystems. This information is then used in a mid-level fusion system to apply the classification from the image recognition server to the belonging measurements. This is the stage

where objects are defined.

- Lastly, a Kalman filter will subscribe to the occupancy grid and track the objects defined in it. Then the occupancy grid will be updated using the output of the Kalman filter.

The above system structure is illustrated in figure 3.1.

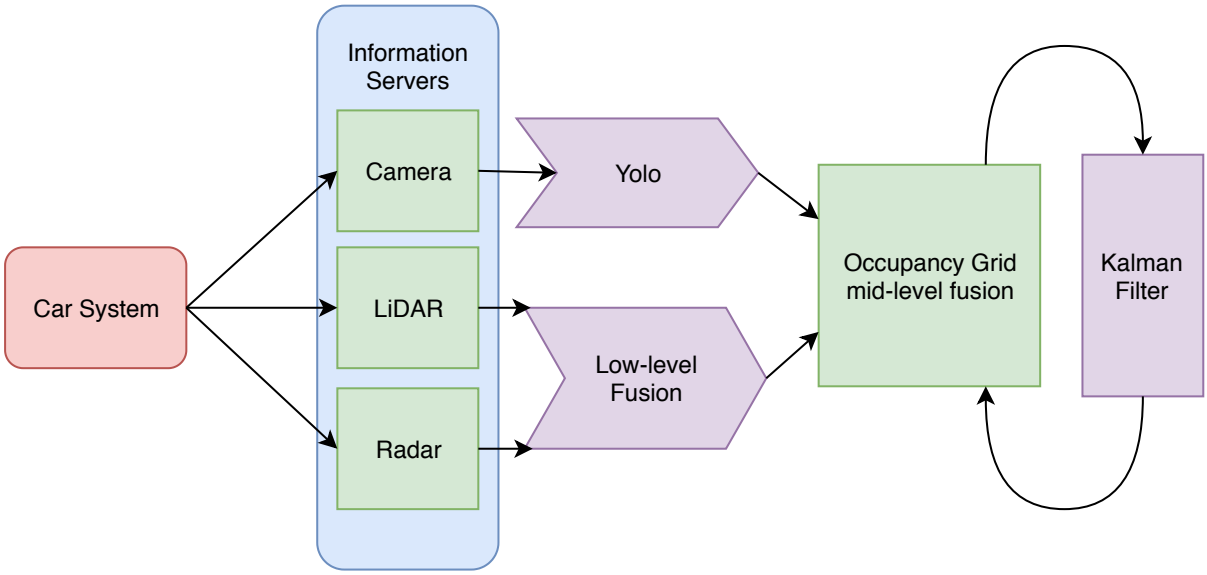


Figure 3.1: The figure describes the system's software structure. Here the arrows represents flow of information. The *Information Servers* collect the data from each of the sockets in the car systems, which are then subscribed to in order to process the data.

3.2 The object identification package

The object identification in this project is performed using YOLOv3. Instead of using the network from the YOLOv3 website [22], *Ultralytics's tensorflow* version was used [23]. As this package is designed to perform object identifications on already existing images, it had to be reworked slightly to accept images in real-time. Furthermore, the output was altered from an image with bounding boxes showing where and what objects were detected to how much of the image an object covers along with its classification.

3.3 The Autonomous Platform

The platform used in this project is depicted in figure 3.2. When it comes to sensors, the platform is equipped with a RPLiDAR-A2, two A111 Radar sensors from Acconeer, a Raspberry Pi camera, and a HC-SR04 Ultrasonic sensor. More detailed information about each of the sensors can be found in Appendix A.

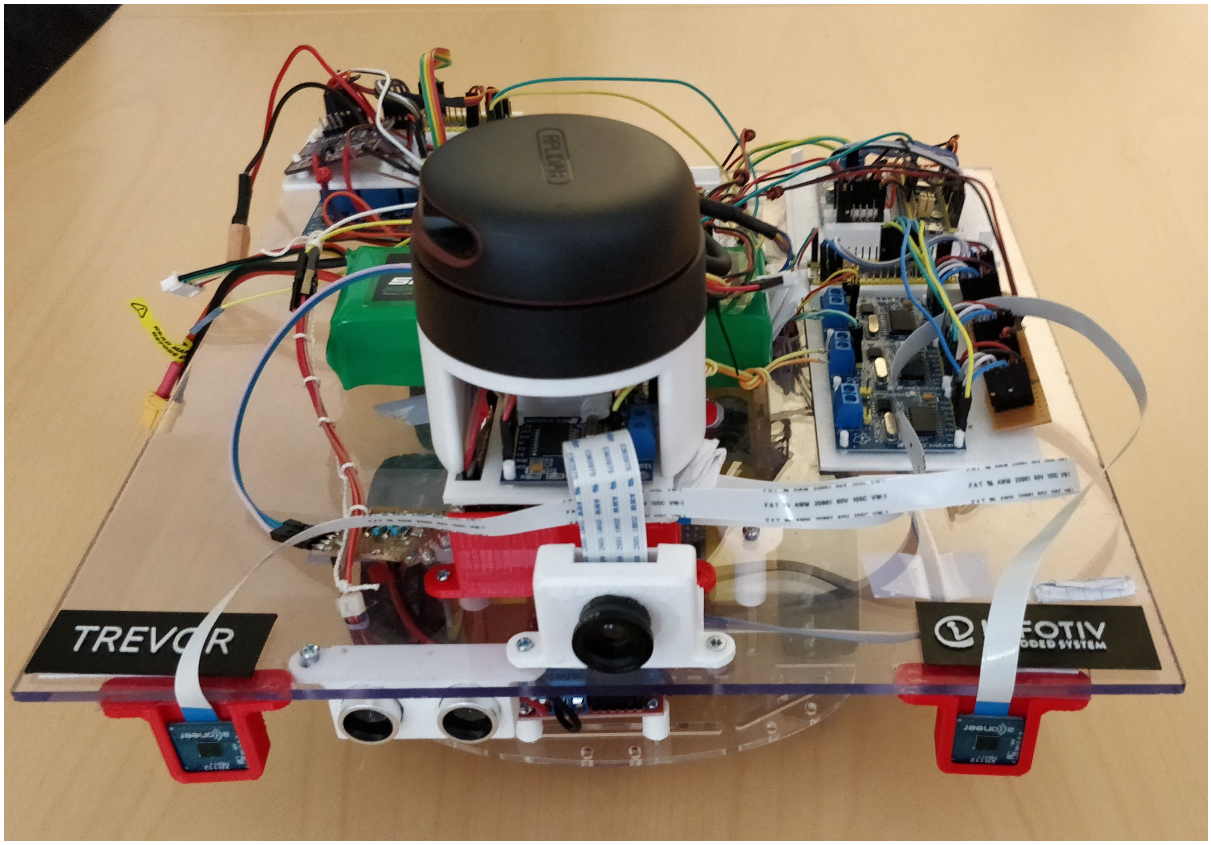


Figure 3.2: The figure shows the autonomous platform that has been used throughout the project. The platform is equipped with a single layer LiDAR, two Radar sensors, a raspberry pi camera, and an Ultrasonic sensor.

4

Occupancy grid

The Occupancy Grid Framework was first introduced by Alberto Elfes and Hans Moravec in 1985 [24], where they used the visual framework in order to create a probabilistic map with the detections of sonar sensors. The framework is used in such a way that each cell is given a probabilistic value as occupied or unoccupied depending on a set threshold. Usually, a defined resolution of the grid is set depending on desired accuracy. However, it might need to be looked into if there are some constraints to the overall computational load for the calculations.

Several frameworks were considered and analyzed in this project in order to find a suitable one. The one that finally was chosen is the the Bayesian Inference Theory because it is easily implemented and has a low computational compared to such as Demspster-Shafer. This makes it suitable for applications that needs to be run in real-time. The Bayesian Inference Theory framework has proven to perform well for occupancy grids which can be seen in articles such as [25] and [26].

The continuation of this chapter describes how the sensors are modeled, how the grid is interpreted, and how the grid is updated at every time iteration. However, first comes a short introduction to all of the inputs of the grid which can be seen here below.

- For the project, a 2D grid has been used of size $[N_x, N_y]$ where N_x is the number of cells on the x-axis and N_y is the number of cells on the y-axis. Each of the cells has an individual value that is connected to the probability of it being occupied. This can be formulated as:

$$O_g = \{O_{gij} : 1 \leq i \leq N_x, 1 \leq j \leq N_y\} \tag{4.1}$$

  Where O_g refers to the occupancy grid and O_{gij} refers to the cell i, j.

- Data from the LiDAR sensor at a time t is given as a two by n-dimensional array containing the distance measurements r_1 to r_n, and their corresponding angles θ_1 to θ_n:

$$z_t = \begin{bmatrix} r_1 & r_2 & ... & r_n \\ \theta_1 & \theta_2 & ... & \theta_n \end{bmatrix} \tag{4.2}$$

- Both the radars and the sonar measurements only provide a distances which is not optimal for the occupancy grid. The radars are capable to retrieve several detections with different distances while the sonar only return one distance.

$$z_t = \begin{bmatrix} r & r_1 & ... & r_n \end{bmatrix} \tag{4.3}$$

- The position of the platform is described by three parameters which are x and y position in the grid as well as the heading of the platform, ϕ. For this project, ϕ is assumed constant and takes the value -90^o. Together, these parameters form the point p_t describing state of the platform:

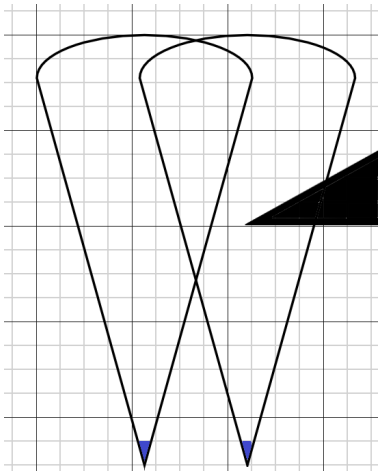
$$p_t = \begin{bmatrix} x & y \end{bmatrix} \tag{4.4}$$

4.1 Sensor models

Two of the more common sensor models that are used in occupancy grids are the forward sensor model and the inverse sensor model. The inverse sensor model is the simpler version of the models. It assumes every sensor in the system as independent of the other sensors. Thereby, each cell can be assigned a value based on the probability of the cell being occupied in regard to each of the sensors separately. It makes it a lot easier to update the grid due to the restriction of all the special cases occurring when different sensors give contradicting information.

A forward sensor model requires each sensor to be individually compared to all others. This is to avoid problems occurring when the model is too simple. However, this make the model more complex and thus also increasing the computational time considerably. Since, the objective of the thesis is to run the algorithms in real-time, which requires the run time of the algorithm to be as low as possible, the inverse sensor model has been used in the creation of the occupancy grid.

Scene | Inverese sensor model | Forward sensor model



The two sensors' reach and an object that is only detectable by one of them.

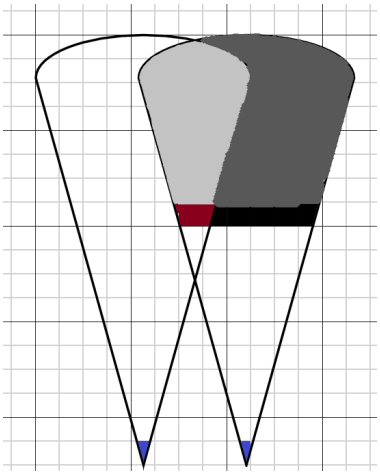


Illustration of an inverse sensor model in a probabilistic grid

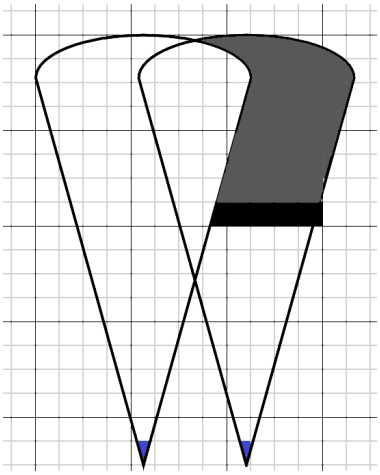


Illustration of a forward sensor model in a probabilistic grid

Figure 4.4: These figures show the different outputs depending on the sensor model that is used. The left sensor will not register any detection and will thereby believe that there is no object in its view. The right sensor on the other hand gets a detection and will fill the cells with the detection distance d as occupied and the cells behind it as "unknown". The red part shows the most crucial part where one sensors says it is occupied and the other says it is unoccupied.

Figure 4.4 shows the difference in the updated grid using the two different sensor models. The forward sensor model clearly adapts to the special case in the figure where one sensor sees an object and the other does not. This is the main reason why the forward sensor model outperforms the inverse sensor model. However, when a probabilistic grid is used, the cells where the different sensors give different results will have a less likelihood of being occupied. This compensates a bit for the lack of flexibility that exists in the inverse sensor model.

In order to represent the distance measurements from the radars and sonar sensors, they have to be modified such that it represent a circular slice of the complete angular span instead of just a distance. This is done by duplicating the angles ten times over the complete angular radius which the sensor is able to detect. When the measurements have been modified, it is time to apply the sensor model to them. The sensor model that have been used in order to update each of the cells uses log odds to reduce numerical errors when multiplying low numbers. The model is stated as in (4.5) where l_t represents the log odds ratio.

$$l_t(O_{gij}) = l_{t-1}(O_{gij}) + log\frac{1 - p(O_{gij})}{p(O_{gij})} + log\frac{p(O_{gij}|z_t)}{1 - p(O_{gij}|z_t)} \tag{4.5}$$

4.2 Initializing the grid

An occupancy grid is initialized by setting a size of the grid depending on the necessary range and accuracy. The number of cells N_x, N_y needs to be chosen depending on the physical size of the grid in meters and what resolution is desired for the specific application. All these cells are given a value midway between the maximum and minimum value a cell can obtain. This results in all cells being in an unknown state before any information is provided.

4.3 Cell interpretation

Cells are treated individually and in that sense are independent of each other. In reality cell values are in fact correlated due to measurement noise and its physical dynamics. However, to simplify the interpretation of the grid, each cell is treated independently. Since the value of a cell can be seen as the probability of it being occupied, it can be summarized with the following equation:

$$p(O_{gij}) = \begin{cases} \text{Occupied} & \text{if} \quad p \geq \alpha \\ \text{Unknown} & \text{if} \quad \alpha \geq p \geq \beta \\ \text{Unoccupied} & \text{if} \quad \beta \geq p \end{cases} \tag{4.6}$$

Here α and β are two different constants that defines the thresholds for the status of the cells. The values for α and β can vary quite a lot depending on safety levels etc. A low value β will make the platform more hesitant and thus lower the risk of running into something. However, it will also make it more vulnerable to single false detections.

4.4 Forgetting factor

Normally, an occupancy grid assumes the environment as static. This means that if a sensor has claimed a cell as occupied, it will stay occupied until another sensor says otherwise. However, this is not entirely feasible since a static grid is more likely to cause false positives and being slower to detect new objects in previously unoccupied cells. A solution to this problem is to apply a forgetting factor to the grid. This means that cells where no new information is given in the last iteration converges towards an unknown state. The forgetting factor should be chosen such that objects can move appropriately within the grid, while not penalizing lost data too heavily, as this would result in removal of relevant objects.

4.5 Updating the grid

Each sensor contributes to the same grid at every iteration instead of creating a separate grid for each of the sensors. This makes it less time consuming due to the decreased total amount of

cells that needs to be updated. However, this complicates handling of contradictory measurements. In figure 4.5, an overview of the grid updating algorithm can be seen. The algorithm requires that a grid already has been initialized which results in a prior grid.

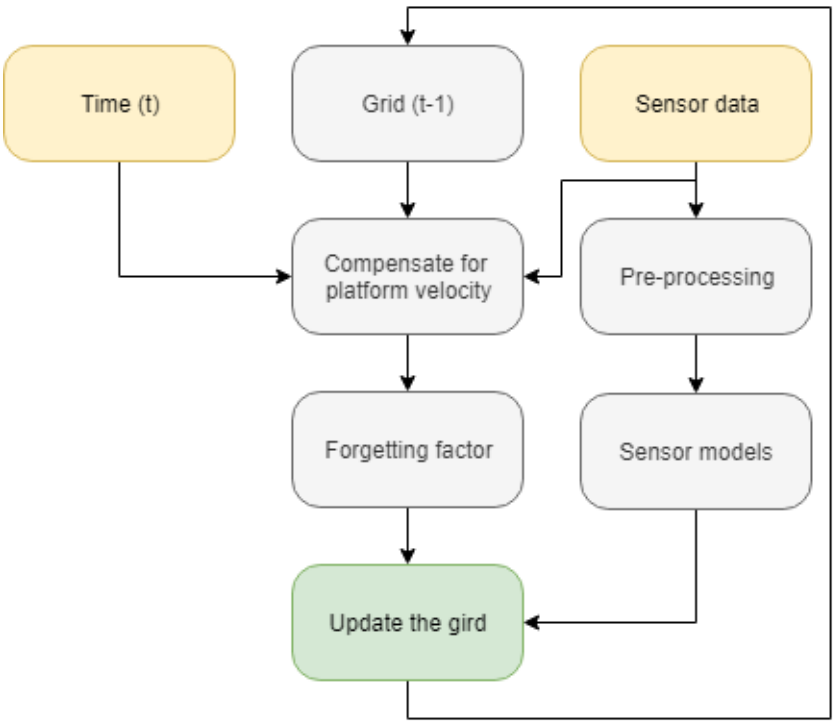


Figure 4.5: The figure describes the structure of how the grid is updated at every iteration. However, it requires that the grid has been initialized

The updating algorithm is based on two main flows that are connected in the last step which is the update of the occupancy grid.

- The first one is the most vital and consists of the sensor data and how it is processed. Firstly, the raw data is filtered such that outliers and other unnecessary data is removed. This data is then fed into the different sensor models that turns the measurements into usable data for the occupancy grid.
- The second main flow handles the previous grid and compensates for the platform's velocity. This is done by measuring the speed and then translating the occupied cells into new cells depending on how much each of the objects has moved in relation to the platform. In order to take care of dynamic environments, the grid is then subject to a forgetting factor making it more dependent on new detections and being able to return a cell into an unknown state.

4.6 Identifcation of objects

This section presents the procedure enabling label assignment to objects within the grid as well as creating a manageable working environment where further applications can be used. The procedure consists of two main steps, namely filtering of the grid and label assignment.

4.6.1 Filtering the grid

In order to distinguish between different objects in the grid, a Gaussian filter is applied to the complete grid. Beyond separating the cells into convenient objects which are easily manage, the filter will also remove outliers and insignificant occupied cells. The standard deviation σ is selected depending on the environment and how detailed the initialized grid is. A higher resolution on the grid will require a higher σ value.

The probability value in each cell has now changed based on the values of neighboring cells. A threshold is used in order to only keep the significant cells. All the adjacent cells are grouped together and can now be identified as one object. This part is quite crucial for the functionality of the platform. If an object that is supposed to be viewed as one object is instead represented by two, it might lose some valuable information or introduce false information.

Objects are created using the Nearest Neighbor method presented in section 2.1. In this project the objects are created as bounding boxes around the their adjacent, occupied cells. The middle point of a bounding box will then represent an object's center, which also is the point that will be observed and tracked using the Kalman filter. With help of the coordinates of the bounding box, it is easy to obtain an estimation of the object's width from the platforms perspective due to its absolute coordinates.

When all the objects has been separated, their history can be recorded and further information about the objects can be assigned. At this stage, each of the objects contain information about its position and size. However, a tractable dictionary that is easily managed has been created such that additional information, e.g. from the camera, can be assigned to the corresponding object.

4.6.2 Label and confidence update

This section presents handling of label assignments over time and alters the confidence level of a classification. These classifications and confidence levels are derived using the camera and the YOLO network. The handling is then performed by updating a list of all separated objects, L, in the previous iteration with the information gathered during the current. This information is stored in a similar list, L_{new}.

The comparison is done by using the euclidean distances between the center of the new objects, in L_{new}, and the center of the objects in L. In this sense it finds the minimum euclidean distances in consecutive iterations. This minimum distance is then used to decide which objects to update in L based on the labels and confidences of the objects in L_{new}.

The euclidean distances have to be below a certain threshold, Δd, in order to be considered a match. This is to allow new objects to enter the grid, without being classified as an already existing object. If an euclidean distance d is below the threshold, the threshold is updated for the current iteration. In this sense the function seeks to find:

$$d = \underset{c_{k,i}}{\operatorname{argmin}} \sqrt{\mid c_{k,i}^2 - c_{k-1}^2 \mid} \quad \textit{for} \quad c_{k,i} \in L_{new} \quad \text{and} \quad c_{k-1} \in L \tag{4.7}$$

The pseudo code for the implemented function is presented below.

Algorithm 1: Updating labels of objects over time

Result: Updated labels and confidence based on new measurements, L_{new}
initialization: L, L_{new}, i = 0;
min = min(size(L), size(L_{new}));
max = max(size(L), size(L_{new}));
while i < *length(min)* **do**
    Δd = 40, L_L = [ ], $e = -1$;
    for k = *0:length(max)* **do**
        if $min[i]_{label}$ == $max[k]_{label}$ **then**
            L_L <– k;
        else
        end
        d = norm(C_i, C_k);
        if $d < \Delta d$ **then**
            Δd = d, $C_{new} = C_k$ $e = k$;
        else
        end
    end
    if $e > -1$ **then**
        Update L with $max[k]$ or $min[i]$ depending on which is smallest, L or L_{new};
    else
    end
    i++;
end

The confidences and labels of the two lists are acquired using YOLO. The YOLO network architecture used is presented in chapter 5 along with how the YOLO predictions are combined with the occupancy grid.

5

Camera and Occupancy grid fusion

Until this chapter an occupancy grid based on the data from the LiDAR and radar sensors has been created. It has then been filtered using a Gaussian filter to create objects. This chapter will introduce the final data stream of the system, namely the camera. It will mostly focus on how the camera information is used in order to classify as many of the determined objects as possible.

5.1 YOLO

The main purpose of this part of the project is to classify the already defined blobs from the object occupancy grid. In this project it is done using the YOLOv3 architecture. YOLOv3 uses both fully convolutional layers and sliding windows for object detection along with a maxpool procedure to downsample the feature maps. However, YOLOv3 does not provide any upsampling. This is because it is designed with speed in focus. The YOLO architecture is presented in figure 5.1.

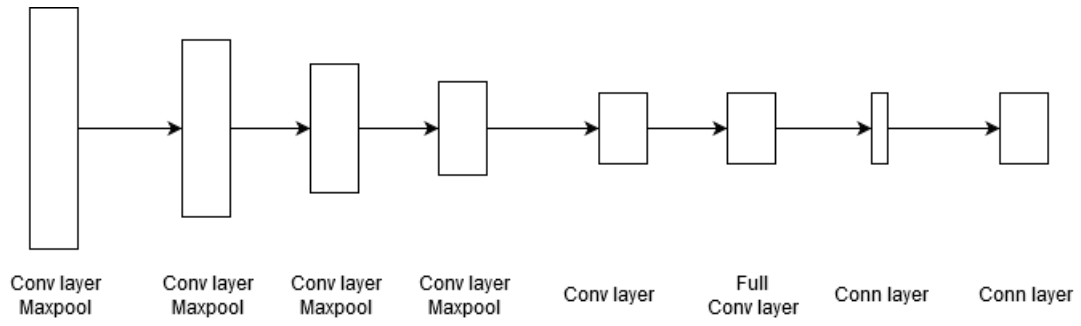


Figure 5.1: The YOLO network structure. Combined conv, convolutional layer, and maxpool layers represent two different layers. A conv layer using sliding window of a certain size (for more details see [1]) and a 2x2 maxpool layer with a stride of 2 pixels. The conv layer after all the maxpool downsampling, resembles numerous conv layers using sliding windows, which is then followed by two fully convolutional layers. The last conn layers are fully connected layers.

The YOLOv3 design makes it an advantageous choice in real-time systems. Additionally, using tiny YOLO instead of the full version also allows for faster object detection. This is because tiny YOLO is designed to use an optimal amount of parameters while still retaining a performance roughly equivalent to that of YOLOv3.

5.2 Object Classification

After having implemented a method for extracting relevant information from the camera stream, it is time to combine the image data with the occupancy grid. This procedure is presented in this section.

5.2.1 Camera view

The first step when performing classification of the objects in the filtered occupancy grid, as presented in section 4.6.1, is to determine which objects in the grid the camera can see. This is done using the position of the vehicle p_t, as described in chapter 4. As p_t describes the center of the vehicle (LiDAR position), a translation to express the grid from the camera's perspective is done accordingly:

$$\begin{aligned} \Delta x &= p_c^x - p_t^x \\ \Delta y &= p_c^y - p_t^y \\ O_g^c &= O_g + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \end{aligned} \tag{5.1}$$

Here p_c is the coordinate of the camera in the occupancy grid with the LiDAR as origin, O_g, and O_g^c is the new grid using the camera's position as origin. From this point the coordinates in the filtered occupancy grid within the camera's field of view is determined. Determining the field of view is done by observing an image stream of an object moving out of the field of view. The angle to the object when it exits the field of view is determined by:

$$\begin{aligned} c_r^\theta &= p_t^\theta + \theta_{view} \\ c_l^\theta &= p_t^\theta - \theta_{view} \end{aligned} \tag{5.2}$$

Depending on whether the object exists to the right or left (*r* or *l*) c_r^θ or c_l^θ can be determined. c_l^θ represents the angle at which the camera's field of view starts, and c_r^θ represents its end. Together they represent the field of view as an angular span on the unit circle with the camera position as origin. The angle θ_{view} is half of the angle at which the object exits the field of view, independent of the camera's orientation. This span describes which objects in O_g^c the YOLOv3 network is able to classify.

5.2.2 Object labeling

When having determined the field of view of the camera and the cells in O_g^c described by the camera's view. The next step is to determine what objects within the view corresponds to which cells in the grid. This is done using the bounding boxes in an image around detected objects created by the YOLOv3 network. The image is described by the angle span $\theta_{object} \in [p_t^\theta - c_r^\theta, p_t^\theta + c_l^\theta]$ in the grid. This means that the bounding box edges, given as the top left and bottom right pixel pairs, can be described as angles in O_g^c given the total image size p_{htot}xp_{wtot}:

$$\begin{aligned} \theta^i_{object} &= 2\theta_{view}\frac{p_w}{p_{wtot}} \\ \theta^i_{start} &= -(180^\circ - 2\theta_{view})/2 - 2\theta_{view}\frac{p^t_x}{p_{wtot}} \\ \theta^i_{end} &= \theta^i_{start} - \theta^i_{object} \end{aligned} \tag{5.3}$$

This will result in a covered angular area of the object, θ_{object}, in the occupancy grid. θ_{object} is dependent on the width in pixels, p_w, of a bounding box over an object. p_w can be calculated from the top left and bottom right pixel pairs outputted from the YOLOv3 network. This span will then start at θ_{start}, which assumes a forward direction of $p^\theta_t = -90^\circ$, and ends at θ_{end}.

After transforming the bounding box pixel values to angles in the occupancy grid a comparison with the center of the objects in the grid can be made. This is done by comparing the angle to the center of an object, determined using the LiDAR, in O^c_g with the angle span $[\theta^i_{start}, \theta^i_{end}]$. Only one object in O^c_g can occupy the aforementioned span. Therefore, the object whose center angle is inside $[\theta^i_{start}, \theta^i_{end}]$ can be assigned the label corresponding to the bounding box with pixel width p_w.

5.2.3 Radar and Camera detection

Until this section the assigning of labels have been dependant on the angle to the center of an object. This angle is only available from the LiDAR measurements. However, as the LiDAR is not able to detect all objects and as the Radar sensors do not provide angular measurements, a special case was developed. This special case handles the objects detected by the camera which the radars see but are undetected by the LiDAR. As mentioned, the Radar sensors do not provide angular measurements. This makes the previous method for object classification presented in the equations in (5.3) unfeasible. However, as there are two Radar sensors mounted on the autonomous platform triangulation can be performed. By using triangulation the angle from each radar θ_1 and θ_2 to an object detected by both radar sensors can be determined accordingly:

$$\theta_1 = arccos(\frac{d_1^2 + \Delta d^2 - d_2^2}{2d_1\Delta d}) \tag{5.4}$$

d_1 and d_2 are the distance measurements from radar sensor 1 and 2 respectively and Δd is the distance between the two sensors. When having determined either of the angles from the Radar sensors, the distance and angle can be used to determine a position in the occupancy grid. Note that the position will be presented with the radar sensors used as origin, meaning that a translation to the camera's point of view, as in (5.1) has to be made.

After moving the origin to the camera the new angle determined through triangulation can be compared in the same fashion as the centers determined by the LiDAR. Thus, a detection made by both radar sensors can be classified.

5.3 Radar detection refinement

A Radar detection is represented by a parabola in the certainty grid since no angular measurement is provided (unless triangulation can be used), as presented in section 4.1. Although the information is vital to the system as a whole there is room for improvement. The assumption that all Radar detections are of the same size and covers the entire angular span of the Radar sensor's lobe is naive. Unfortunately the situation is inescapable unless the two Radar sensors each detect the same object or if the LiDAR also finds the object. However, most of the detectable objects by one Radar in the autonomous platform's way is detectable by the other. This is because of the width of the Radar signal's lobe, which is roughly 45^o.

As previously explained an object detected by both Radar sensors can be assigned a coordinate and this coordinate can be assigned a label using the same principle as for the LiDAR measurements. However, the resulting coordinate from the triangulation is just a coordinate and very few objects are of that small a size. Therefore, the width of the objects in pixels help determine the size of the detected object. A Radar sensor provides the distance and the camera provides the angular size θ^i_{object}. Thus the initial two parabolas (since both sensor's are required to detect the object) can be refined to only occupy cells in the grid between the angles θ^i_{start} and θ^i_{end}, from the equations in (5.3).

6

System models

In order to perform multiple object tracking over time, while moving, a Kalman filter is implemented. This requires a defined measurement and process model as well as a compatible choice of filter structure. The procedure to determine these hyperparameters is presented in this chapter.

6.1 Motion model

A necessity when attempting to track objects is to derive a model describing the movement of an object between two time steps. This model is the motion model and this section presents the two models looked into in this project.

6.1.1 Constant velocity model

The first step when creating a Kalman filter is to determine what motion model to use. Then the states x, state transition matrix, A, and input matrix, B, acting on the input u can be determined. The state transition matrix is determined by observing how the objects are moving in relation to the platform. As the goal is to perform this tracking while the vehicle is moving a constant velocity model is suitable for the task at hand. Even more so since the velocity of the platform will be more or less constant throughout the entire testing sequence. The constant velocity model for the system is formulated as:

$$A = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} T & 0 \\ 0 & T \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad z = \begin{bmatrix} x & y & \dot{x} & \dot{y} \end{bmatrix}^T, \quad u = \begin{bmatrix} \Delta v_x & \Delta v_y \end{bmatrix}^T \tag{6.1}$$

Where the states in z are the Cartesian position x, y and the velocity along each axis $\dot{x}$, $\dot{y}$. The input, u, is the change in velocity along each axis between two iterations, meaning that he input is 0 as long as the velocity is constant. Lastly, T represents the sampling time. Together these form the state transition model:

$$x_{k+1} = Ax_k + Bu_k + q \tag{6.2}$$

Furthermore, to complete the motion model a process noise model has to be chosen. In this project the process noise model taken from [27] and follows as:

$$Q = \begin{bmatrix} \sigma_1^2 \frac{T^3}{3} & 0 & \sigma_1^2 \frac{T^2}{2} & 0 \\ 0 & \sigma_2^2 \frac{T^3}{3} & 0 & \sigma_2^2 \frac{T^2}{2} \\ \sigma_1^2 \frac{T^2}{2} & 0 & \sigma_1^2 T & 0 \\ 0 & \sigma_2^2 \frac{T^2}{2} & 0 & \sigma_2^2 T \end{bmatrix} \tag{6.3}$$

$$q \sim \mathcal{N}(0, Q) \tag{6.4}$$

Where σ_1^2 and σ_2^2 are the variances along each axis in the process noise.

6.1.2 Coordinated turn model

A weakness of the constant velocity is that objects subjected to a yaw rate will be poorly tracked. In the previous section it was mentioned that the autonomous vehicle would move with a constant velocity, which remains true. However, when considering the turns of the vehicle the velocity will change and objects will no longer move as described by the constant velocity model. This requires a more advanced model that can take angular velocity into account. Therefore, a cartesian coordinated turn model was designed accordingly:

$$x_{k+1} = \begin{bmatrix} x + T\dot{x}\cos(\omega T) - T\dot{y}\sin(\omega T) \\ y + T\dot{x}\sin(\omega T) + T\dot{y}\cos(\omega T) \\ \dot{x}\cos(\omega T) - \dot{y}\sin(\omega T) \\ \dot{x}\sin(\omega T) + \dot{y}\cos(\omega T) \\ \omega \end{bmatrix} + q, \quad x_k = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \omega \end{bmatrix} \tag{6.5}$$

This model is based on the coordinated turn model in [28]. Where q is the process noise and has to be modeled. This model was chosen from [29] and follows as :

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \delta v_x \sigma_v^2 T & 0 & 0 \\ 0 & 0 & 0 & \delta v_y \sigma_v^2 T & 0 \\ 0 & 0 & 0 & 0 & \sigma_\omega^2 T \end{bmatrix} \tag{6.6}$$

$$\delta v_x = \frac{\dot{x}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \tag{6.7}$$

$$\delta v_y = \frac{\dot{y}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \tag{6.8}$$

$$q \sim \mathcal{N}(0, Q) \tag{6.9}$$

Where σ_v^2 is the velocity noise variance, and δv_x and δv_y describes its contribution along each axis. σ_ω^2 is the angular velocity noise variance.

6.2 Modelling of measurements

While the previous section defined the motion models used in this project, this section seeks to introduce the measurement models corresponding to the different choices of state variables, z. Furthermore, a method for determining the measurement noise is presented.

6.2.1 Measurement model

The measurement model in this project is determined by what is sought to track. In this project the centers of detected objects are subjected to tracking. This is suitable since the center provide a cartesian measurement easily comparable to both of the aforementioned motion models. Therefore, the measurement model matrix is simply:

$$H_{cv} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad H_{ct} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{6.10}$$

The structure of the measurement model is adjusted to fit the state vector x_k for the two cases. Where H_{cv} is applied to the state vector in the constant velocity model and H_{ct} is applied to the state vector in the coordinated turn model. This creates the two measurement models accordingly:

$$y_k = H_{cv} x_k^{cv} + r \tag{6.11}$$

$$y_k = H_{ct} x_k^{ct} + r \tag{6.12}$$

Where x_k^{cv} and x_k^{ct} denotes the state vectors for the two measurement models, and r is the measurement noise described as $r \sim \mathcal{N}(0, R)$, where R is the measurement noise covariance matrix.

6.2.2 Measurement noise

As opposed to the models of the noise acting on the motion model, the noise of the measurements is not a design choice. This measurement noise is the inaccuracy in the created occupancy grid. This inaccuracy is determined by the process to create the input measurements to the Kalman filter, which is the center of objects. Therefore, the inaccuracy in the process of determining an object's center, as described in section 4.6.1, is a more precise description of the measurement noise.
After having formulated the measurement noise as the deviations in an object's center, a simulation to determine the noise covariance matrix, R, can be made. The simulation is to have the autonomous platform standing still, while calculating the center of a stationary object. From the calculated centers a mean and covariance can be derived. The derived covariance is the measurement noise covariance used by the Kalman filter.

6.3 Kalman filter type

Two types of Kalman filters were studied in this project. The regular, linear Kalman filter along with the constant velocity model presented in section 6.1.1 and the unscented Kalman filter with the coordinated turn model from section 6.1.2. As to why the UKF was chosen over the extended Kalman filter, is based on the accuracy differences described in the sections 2.4.2.2 and 2.4.2.3. Further support for choosing the UKF filter can be found in [28] where the UKF is claimed to be preferred when having a cartesian coordinated turn model, which is the case of this project.

7

Results

7.1 Kalman filter study

This section presents the performance of the Kalman filter versus the unscented Kalman filter for different process noises as well as different measurement noises. Furthermore, the determined measurement noise is also presented in this section.

7.1.1 Measurement noise covariance

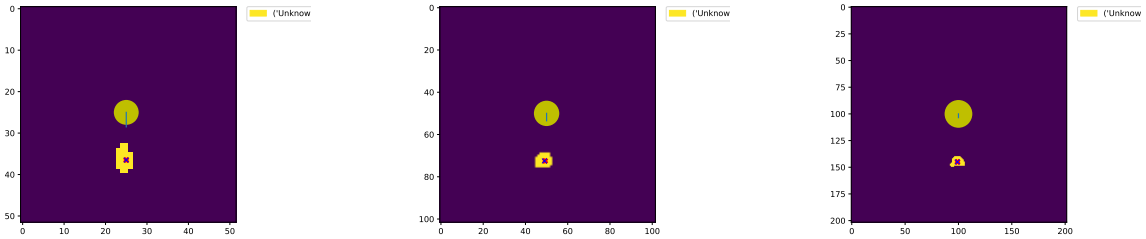
As described in section 6.2.2 the measurement noise was determined by measuring the center (x, y) of a stationary object over time. In order to gain a more accurate description of the measurement noise two different objects were used, one small and one large. Furthermore, the impact of the resolution of the occupancy grid was studied. Conclusively, three different covariance matrices were determined for each of the objects. At a resolution of 50x50, 100x100 and 200x200 which results in a cell size of $6x$6cm, $3x$3cm, and $1.5x1.5$cm respectively. The following covariance matrices, where the variance is expressed in cells, were determined for the smaller object:

$$R_{50}^{S} = \begin{bmatrix} 0.17 & 0.01 \\ 0.01 & 0.05 \end{bmatrix}, \quad R_{100}^{S} = \begin{bmatrix} 0.26 & 0.02 \\ 0.02 & 0.06 \end{bmatrix}, \quad R_{200}^{S} = \begin{bmatrix} 1.61 & 0.25 \\ 0.25 & 0.48 \end{bmatrix} \tag{7.1}$$

The covariance matrices for the large object were determined as:

$$R_{50}^{L} = \begin{bmatrix} 0.16 & 0.00 \\ 0.00 & 0.00 \end{bmatrix}, \quad R_{100}^{L} = \begin{bmatrix} 0.77 & 0.11 \\ 0.11 & 0.07 \end{bmatrix}, \quad R_{200}^{L} = \begin{bmatrix} 5.22 & 0.57 \\ 0.57 & 0.36 \end{bmatrix} \tag{7.2}$$

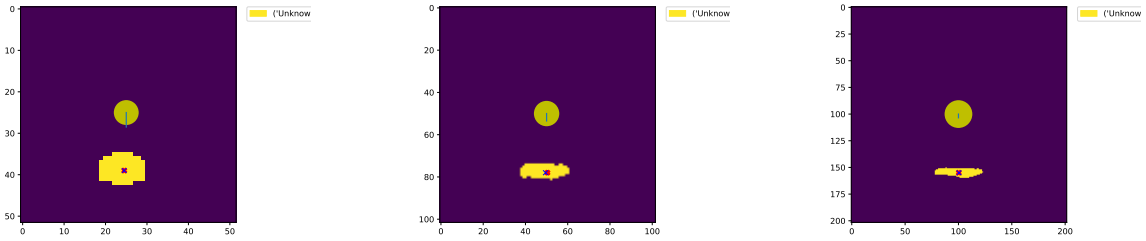
The above covariance study was performed with both objects in front of the platform. At a location where roughly $\theta = -\frac{\pi}{2}$. The small object and the car can be viewed at the different resolutions in figure 7.1.



Resolution = 50x50 Resolution = 100x100 Resolution = 200x200

Figure 7.1: The stationary autonomous platform, marked as a circle, measuring the center of a stationary object at different resolutions. The object is the yellow blob in the occupancy grid, the center is denoted by a blue x, and the red dot is the (x, y) states of the Kalman filter.

The same view as in figure 7.1 is presented for the larger object and can be observed in figure 7.2.



Resolution = 50x50 Resolution = 100x100 Resolution = 200x200

Figure 7.2: The stationary autonomous platform, marked as a circle, measuring the center of a stationary object at different resolutions. The object is the yellow blob in the occupancy grid, the center is denoted by a blue x, and the red dot is the (x, y) states of the Kalman filter.

The shape of the objects in the both figure 7.1 and 7.2 reveals the reasons behind the calculated covariance R^S and R^L. As the vehicle's heading, in this case, is parallel to the y-axis the width is determined by the x-axis values. In order to allow the designed filter to perform independently of angle to the object, the noise covariance has to be altered. The new covariance is then set to:

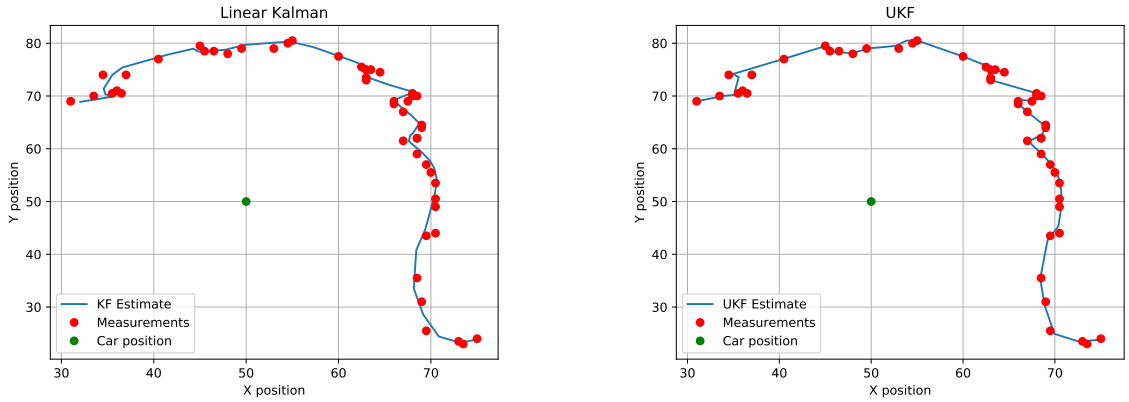
$$R = \begin{bmatrix} R_x & 0 \\ 0 & R_x \end{bmatrix} \tag{7.3}$$

Where R_x is the x variance. For instance, R^S_{50} yields a R_x of 0.17. The zeros is set since the measurement noise is not desired to be correlated between the two axes. However, important to note is that the measurement noise here is only expressed in the ability to calculate centers in the occupancy grid. Therefore, it might be reasonable to increase the R_x value.

7.1.2 UKF and KF performance

In order to evaluate the performance and suitability of the UKF and KF an attempt to track an object moving around the platform was made. This is done by measuring the center of an object

over time and then applying the Kalman filters afterwards. So that multiple noise variances can be tested on the same data. The grid resolution is 100x100 cells for all test cases and therefore and $R_x = 0.77$ was initially chosen. The result of such a study can be viewed in figure 7.3.

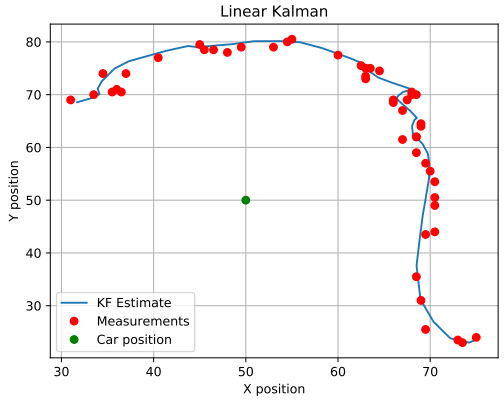


KF performance when tracking an object circulating the vehicle

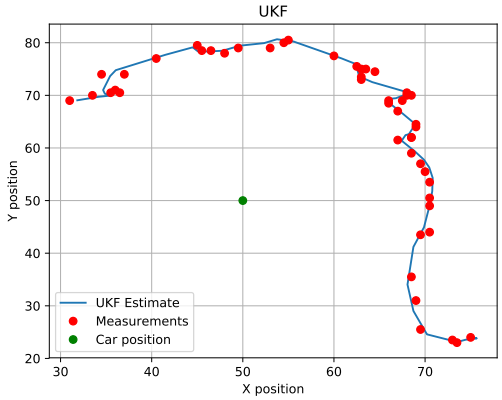
UKF performance when tracking an object circulating the vehicle

Figure 7.3: The figure displays the difference in performance when tracking an object circulating the platform. The platform is represented with a green dot. The blue line in both cases is the estimated trajectory and the red dots are measurements of the object. The process noise variance in this case is $\sigma_1^2 = 1$, $\sigma_2^2 = 1$, $\sigma_v^2 = 1$ *unit/s*, $\sigma_\omega^2 = 1$ *rad/s*. The measurement noise variance in this case is $R_x = 0.77$. The initial state is $(75, 24)$.

In section 7.1.1 the determined covariance matrices was deemed possibly optimistic, therefore it is of interest to study the performance when R_x is high. Figure 7.4 presents the result when $R_x = 5$ (which would roughly correspond to the R_{200}^L covariance matrix).



KF performance when tracking an object circulating the vehicle



UKF performance when tracking an object circulating the vehicle

Figure 7.4: The figure displays the difference in performance when tracking an object circulating the platform. The platform is represented with a green dot. The blue line in both cases is the estimated trajectory and the red dots are measurements of the object. The process noise variance in this case is $\sigma_1^2 = 1$, $\sigma_2^2 = 1$, $\sigma_v^2 = 1$ *unit/s*, $\sigma_\omega^2 = 1$ *rad/s*. The measurement noise variance in this case is $R_x = 5$. The initial state is $(75, 24)$.

From the figures 7.3 and 7.4 it is visible that the UKF captures the turning motion in a more accurate way. Especially when the measurements noise variance is increased and the filters rely more heavily on the predicted states $x_{k|k-1}$ from (2.18) and (2.9). Therefore, this filter is chosen as the tracking tool during the test of the systems in section 7.2.

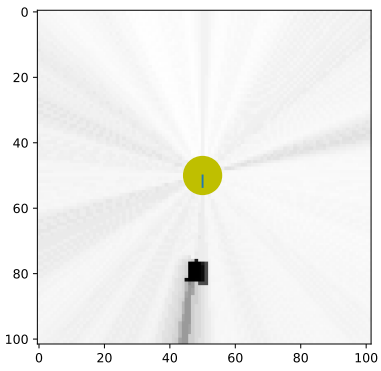
7.2 Object detection and tracking

This section presents the results of the combined system at each of the major steps in the algorithm which can be viewed in chapter 3. The results in this section is the behavior of the algorithm when performing object detection, and tracking on a vase. The algorithm uses a grid resolution of 100x100 cells covering a 3x3 m^2 area surrounding the platform.

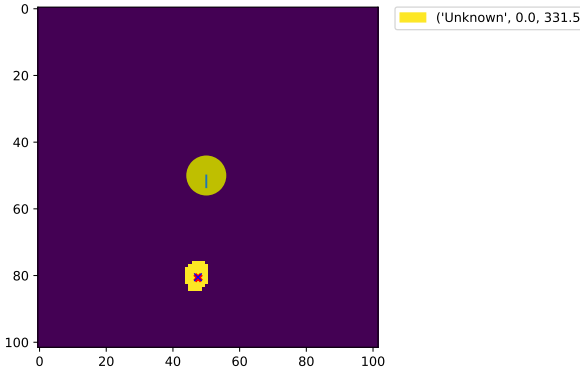
7.2.1 Object detection

This section shows a study of the algorithm's ability to perform object detection. By starting with the occupancy grid containing the fused LiDAR and Radar measurements, presented in figure 7.5, and then displaying operations on this occupancy grid. Thereafter, the output from the tiny YOLOv3 network is shown, and finally the resulting filtered occupancy grid based on camera information, and LiDAR and Radar measurements is presented. The initial state of this study is presented in figure 7.5.

The occupancy grid is depicted to the left in figure 7.5 and is as stated the result of the probabilistic cell view of the fusion between the LiDAR and Radar sensors. This grid is filtered



Occupancy grid based on LiDAR and Radar measurements.

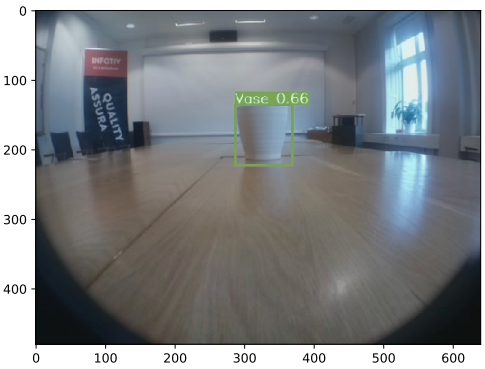


Filtered occupancy grid based on LiDAR and Radar measurements.

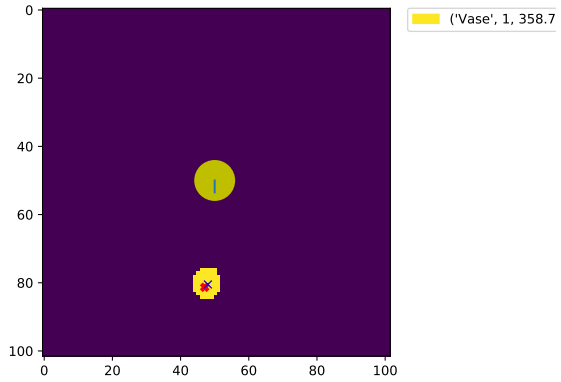
Figure 7.5: The figure shows the occupancy grid based on LiDAR and Radar measurements when it has detected an object in front of it. The blackness of the cells to the left represents the probability of an individual cell being occupied. Hence, the white and light grey are unoccupied with a high certainty while the black area is likely to be occupied. The grid has been filtered in the right figure and is showing the grid in only occupied/unoccupied state for each cell. The blue cross to the right represents the center of the object while the red dot represents the estimated center by the Kalman filter.

using a Gaussian filter introducing a standard deviation of 2 cells to each cell. This results in the filtered occupancy grid which is presented to the right in figure 7.5.

When having determined an object in the filtered occupancy grid, the next step is classification of it. As can be seen from the legend in figure 7.5, the label is *Unknown* and the confidence level of the label is 0.0. This is not strange since the tiny YOLOv3 output is not fused with the grid. This output image from the tiny YOLOv3 network is presented to the left in figure 7.6.



Output image from the YOLO network



Ocuppancy grid where the classification has been assigned to the object

Figure 7.6: The figure displays the state of the system, when a vase is detected and standing in front of the platform. To the left the vase is visible through the camera and to the right an object labeled vase is present. The center measurement is denoted using as x and the initial UKF state is the red dot.

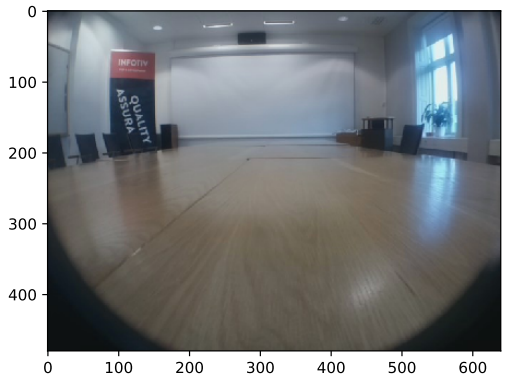
The output of the tiny YOLOv3 network to the left in figure 7.6, shows a predicted bounding box around a detected object with a corresponding label, *vase*. The confidence of the network is displayed on the bounding box and is a value between 0 and 1. For this vase it displays a confidence of 0.66. When combining this output to the filtered occupancy grid in figure 7.5, the final object detection behavior is achieved. This behavior is presented to the right in figure 7.6.

Between the left and the right image displayed in figure 7.6 there is a difference in the confidence levels. This is because in the camera image the confidence level of the tiny YOLOv3 network, while the occupancy grid shows the confidence level of the entire object detection system, also ranging between 0 and 1. The system's confidence level is updated with every time iteration. Therefore, if it finds the same object within consecutive iterations and matches it to an equivalent label the confidence of the object detection algorithm is increased.

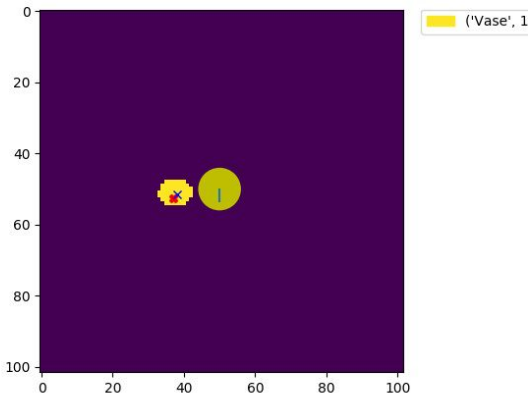
7.2.2 Tracking

In order to display the tracking ability of the algorithm, the system attempted to track a vase. The vase is initially positioned in front of the platform to allow classification to be made. The goal is then to move the vase out of view, and check its ability to retaining the vase label. I.e. accurately track the object, where the initial state of the vase is presented in figure 7.6. The initial grid coordinates are roughly $(x, y) = (48, 81)$ for both the UKF and the center of the object. The center is denoted as a cross and the estimated UKF state is marked by a red dot.

After having set up the experiment, the vase displayed in figure 7.6 is moved until it is to the side of the platform. The final state is presented in figure 7.7.



Camera vision



Filtered occupancy grid

Figure 7.7: The figure displays the state of the system, when a vase is believed to be standing beside the platform. To the left the camera view can be seen and to the right an object labeled vase is present in the filtered occupancy grid. The center measurement is denoted using as x and the initial UKF state is the red dot.

The final state of the experiment, depicted in figure 7.7, is the object in the filtered occupancy grid besides the platform. The object in the grid has retained its label, vase, and therefore has been accurately tracked. The tracking trajectory is presented in figure 7.8.

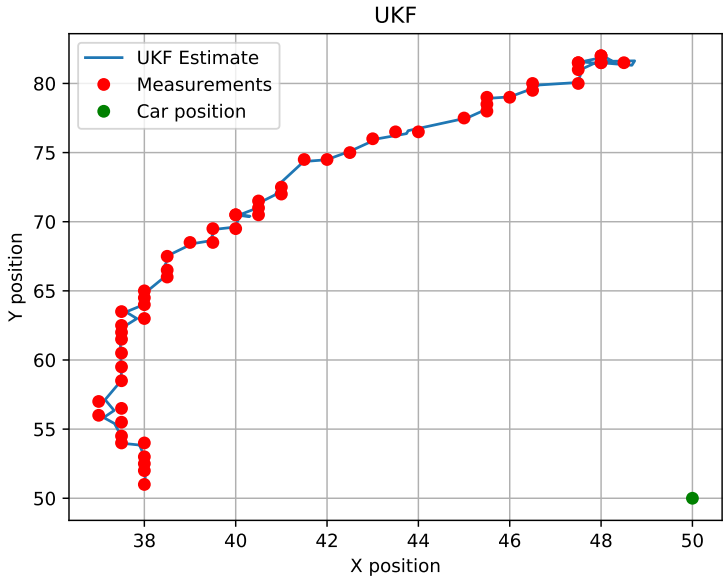
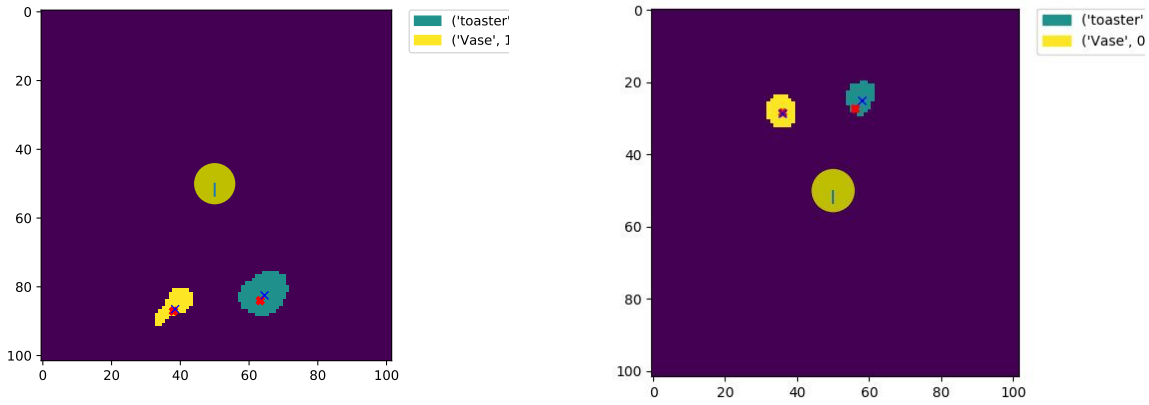


Figure 7.8: The figure displays the tracking performance of the UKF when following a vase . The blue line is the estimated trajectory, the red dots are measurements of the vase, and the car is represented by a green dot in $(50, 50)$. The process noise variance in this case is $\sigma_1^2 = 1$, $\sigma_2^2 = 1$, $\sigma_v^2 = 1$ *unit/s*, $\sigma_\omega^2 = 1$ *rad/s*. The measurement noise variance in this case is $R_x = 0.77$. The initial state is $(48, 81)$.

From figure 7.10 the final position of the vase is roughly in the coordinates $(x, y) = (38, 51)$. When comparing this position to the object presented in the filtered occupancy grid in figure 7.7, it seems reasonable. As it is the center of the object the UKF is attempting to track.

Multiple objects

This section presents a study similar to the previous tracking study. However, this time the ability to track multiple objects is being tested. Initially two objects were placed in front of the platform and the goal is to drive in a straight line past them. Both the initial and final states are being presented in figure 7.9.



Initial state

Final state

Figure 7.9: The figure displays the initial and final state when tracking two different objects, one labeled *toaster* (dark green blob) and the other labeled *vase* (yellow blob). The autonomous platform is illustrated as the yellow circle in $(x, y) = (50, 50)$. The center measurements are denoted as blue x and the UKF states are the red dots.

One can see from figure 7.9 that the algorithm has kept the correct labels until the final state. Worth noting is that the confidence level of the classification for the *vase* is different for the two states (in the final state the confidence level is not actually 0 since then the label would have been discarded). This could be because of faulty readings in one or two iterations lowering the confidence level of its classification. However, it has clearly managed to keep track of both objects, which also can be seen in figure 7.10 where the estimated trajectories using the UKF filter are presented. Here the toaster's final position is roughly $(x, y) = (18, 62)$ and the vase's is around $(x, y) = (21, 37)$. When comparing this to figure 7.9 these final positions seem reasonable.

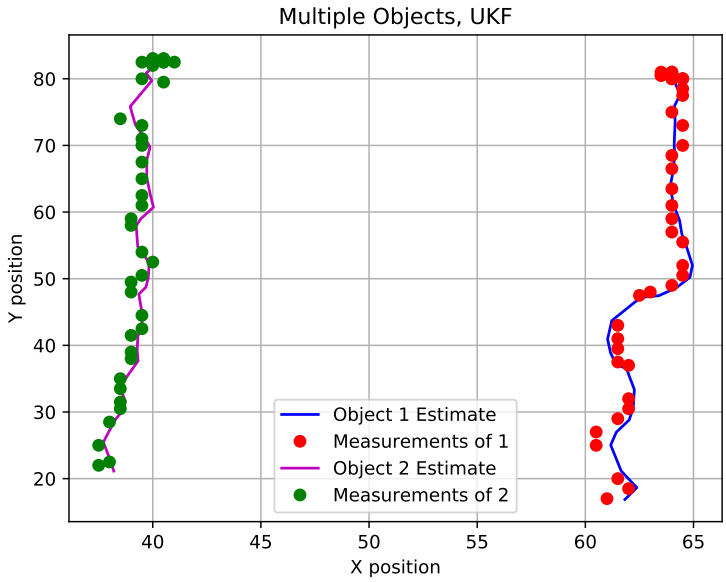
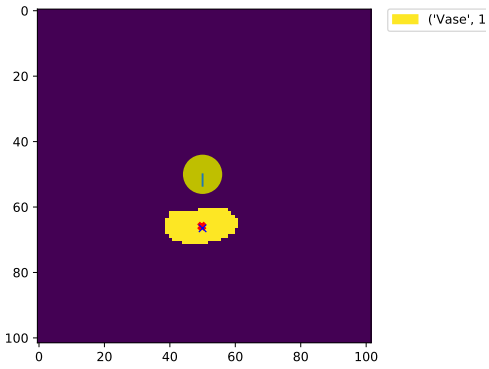


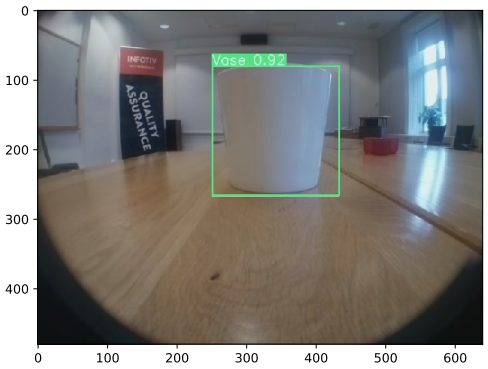
Figure 7.10: The figure displays the tracking performance of the UKF when estimating the center positions of the the objects in figure 7.9. The blue line is the estimated trajectory of the toaster and the red dots are the measurements of its center, while the purple line is the estimated trajectory of the vase with the green dots representing the measurements of its center. The process noise variance in this case is $\sigma_1^2 = 1$, $\sigma_2^2 = 1$, $\sigma_v^2 = 1$ *unit/s*, $\sigma_\omega^2 = 1$ *rad/s*. The measurement noise variance in this case is $R_x = 0.77$. The initial states are $(x, y) = (64, 81)$ for the toaster and $(x, y) = (41, 82)$ for the vase.

7.3 Radar detection refinement

This section presents an illustration of the method presented in section 5.3, where a camera detection is used to refine a detection by both Radar sensors. Firstly, the appearance of a detected *vase* when using only the Radar sensors and the camera is presented, see figure 7.11.



Filtered grid output



YOLO network output

Figure 7.11: The figure presents the shape of an object, *vase*, in the filtered object occupancy grid, to the left, and the output of the YOLOv3 network, to the right. The shape of the object in the grid (the yellow blob) is determined using only the Radar sensors. The circle in the grid represents the platform.

When then applying the method from section 5.3 a cropped version of the grid object in figure 7.11 is obtained. The new shape of the object is presented figure 7.12.

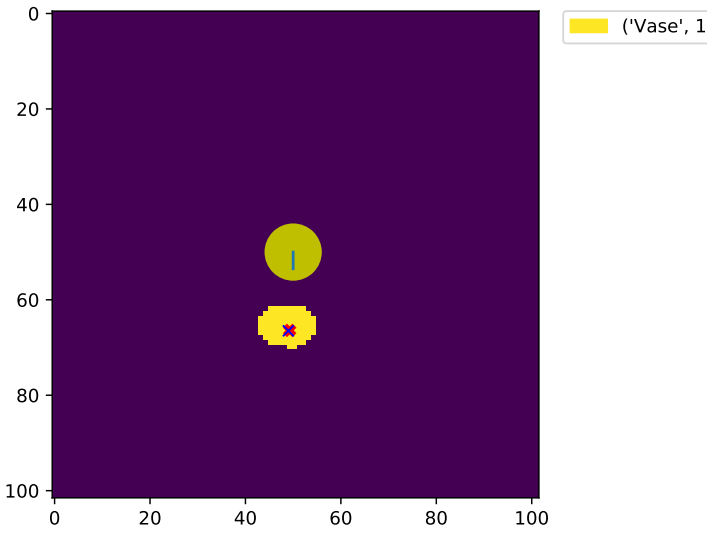


Figure 7.12: The figure displays the impact of the method in section 5.3 on an object determined by the Radar sensors. The object is presented as a yellow blob, and the circle represents the autonomous platform.

There is an apparent difference between the objects in the filtered grid in figure 7.11 and 7.12. In figure 7.11 the object covers the entire field of view of the Radar sensors. Whereas in the

figure 7.12 the observed object is refined to match the percentage of the camera's field of view it occupies.

7.4 Real-time tests

The platform was tested in multiple scenarios of a developed test cases in order to validate the contribution and necessity of each the different sensors. The test case consists of four different objects. The first object is placed in front of the platform such that all sensors should be able to detect it. The platform should therefore stop when approaching it. Thereafter, the object is removed and the platform should continue. The second object is a harmless object, which for instance could be a plastic bag or a fruit. However, the platform should determine it as harmless and thus actively run over it. The third object is a slice of plexiglass. The purpose of this object is to test the behavior when facing transparent objects. Furthermore, the last object is placed outside of the path such that it can be seen and classified, but deemed harmless as it is not in the platform's way. A drawing of the complete test case can be seen in figure 7.13. All the tests have been performed with a grid size of $3x3$ m in order to get a detailed view of the close surroundings. The algorithm limits the updated rate of the camera's detection to every fifth iteration, since running the image processing takes roughly 0.4 seconds. Furthermore, the algorithm has been run on an external computer, which is communicated with via WIFI.

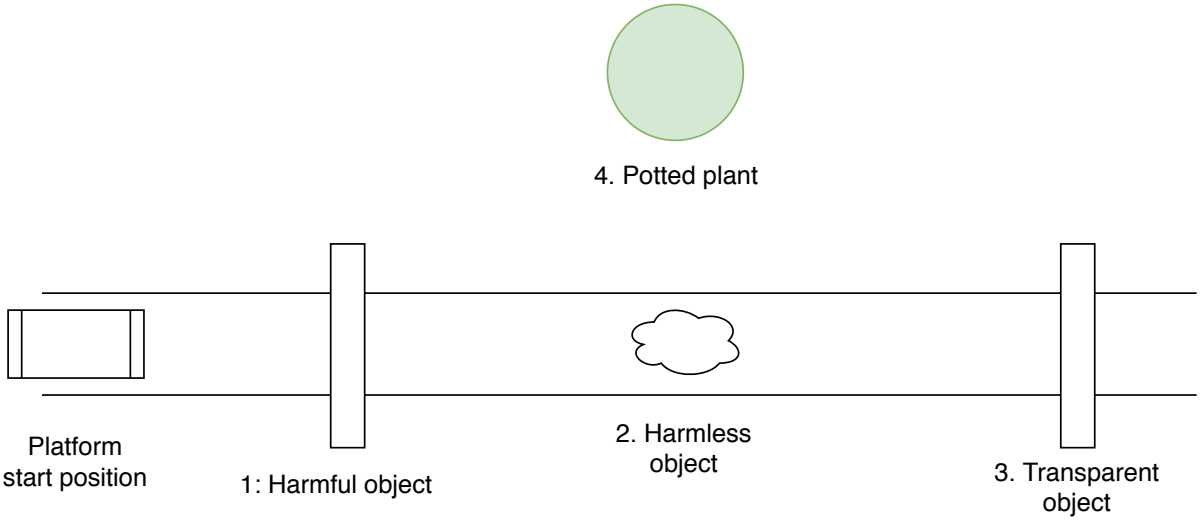


Figure 7.13: The figure shows a visualization of the developed test case. The platform is put to the left and then it follows the path to the right. Object 1 is a harmful object for the platform which it is supposed to stop for. Object number 2 is a harmless object and is therefore not supposed to be stopped for. The third object is a transparent slice of plexiglass and a fourth object (A potted plant) is placed at the side of the path for detection and tracking purposes.

7.4.1 Tests of complete system

For the complete system tests, all the sensors available to the algorithm were active. A summary of the performance of the system at different grid resolutions is presented in table 7.1. The

different resolutions were tested ten times each and the success rate represents how frequently the system passed the test. It shows that a grid resolution of 20x20 is too small to detect objects sufficiently. However, there is also an upper limit. At a resolution of 200x200 cells the performance decreased again, possibly due to the increased iteration time. In this case, the system had issues detecting the second object as it stopped instead of running over it.

Table 7.1: Table with the results from tests of different grid resolutions.

Grid size (cells)	Size of each cell [cm]	Success rate [%]	Time per iteration [s]
20x20	15x15cm	0%	0.3958s
50x50	6x6cm	100%	0.4106s
100x100	3x3cm	100%	0.4218s
200x200	1.5x1.5cm	90%	0.5254s

7.4.2 Tests without camera

The ability to classify objects was lost when removing the camera and the tiny YOLOv3 output. However, the platform was still able to detect the same amount of objects as the complete system. When running the test case the system indeed struggled with the second object. Since the system could not classify it and therefore could not determine whether it was harmful or not. However, the time per iteration of the algorithm is now lowered, since it need not run the tiny YOLOv3 network, which resulted in an average time per iteration of 0.3549s. The results of this test is presented in table 7.2. Here the success rate is the number of times the platform acted as intended when approaching an object.

Table 7.2: Table with the results from tests excluding the camera. These tests were performed with a grid of 100x100 cells.

Object	Success rate [%]
1	100%
2	0%
3	100%
4	100%

7.4.3 Tests without LiDAR

When the LiDAR is removed, the platform has lost it main source of distance measurements. Especially, for objects that are not in front of the platform. This means that the platform is unable to track previously classified objects after having passed them. In table 7.3 it can be seen that the platform is unable to detect the fourth object outside its path. However, it is able to pass the other three objects in its path, with a time per iteration of 0.2956s.

Table 7.3: Table with the results from tests excluding the LiDAR. These tests were performed with a grid of 100x100 cells.

Object	Success rate [%]
1	100%
2	100%
3	100%
4	0%

7.4.4 Tests without Radar

The Radar sensors add the ability to detect objects below the LiDAR's measurement layer, as well as transparent objects. These features are also added by the ultrasonic sensors. However, the ultrasonic sensors are only able to detect the transparent object 20% of the times. This supports the decision of ruling it out from the final system. Apart from that, the other sensors are able to retain the performance when it comes to the other three object, as can be seen in table 7.4, where a timer per iteration of 0.3804s was achieved.

Table 7.4: Table with the results from tests excluding the Radar. These tests were performed with a grid of 100x100 cells.

Object	Success rate [%]
1	100%
2	100%
3	20%
4	100%

7.4.5 Tests without Ultrasonic sensor

In table 7.5 one can see that the ultrasonic sensor is not adding any valuable information. Since the system performs equally as the complete system with all the sensors included. The average time per iteration for the system without the ultrasonic sensor was 0.4196s.

Table 7.5: Table with the results from tests excluding the Ultra sonic sensor. These tests were performed with a grid of 100x100 cells.

Object	Success rate [%]
1	100%
2	100%
3	100%
4	100%

8

Discussion

8.1 Kalman filter design

This section presents a discussion regarding the models used by the Kalman filter. These models are the motion models, and the the process noise model. Furthermore, the process and measurement noise covariance and the relation to the occupancy grid resolution is brought up and clarified.

8.1.1 Motion Model

The motion models investigated were a constant velocity model and a coordinated turn model. No further models were looked into as these two models were considered sufficient to describe the state transition for objects in this project. However, the choice between the two models are dependent on what the desired functionality is.

When keeping track of a stationary object as depicted in figure 7.13, and driving in a straight line, a constant velocity describes the objects sufficiently. Thus, the UKF is unnecessary in this case. However, when considering the tracking of objects with an angular velocity circulating the vehicle as in figure 7.3, KF and the constant velocity model drops in performance. Furthermore, the KF and constant velocity model is quite sensitive to large measurement noises compared to the UKF and coordinated turn model. Therefore, the UKF and the coordinated turn model is arguably a more versatile option in this project, and thus chosen is chosen as the implemented filter for the real-time test cases.

8.1.2 Occupancy grid resolution and noise

The measurement and process noise, if expressed as a covariance in grid cells, are correlated to the grid resolution and object size in the filtered occupancy grid. This phenomenon is discussed in this section. Furthermore, the choice of the process noise's magnitude, and the contribution of the implemented Kalman filter is discussed.

8.1.2.1 Measurement noise

The results presented in section 7.1.1 showed a correlation between occupancy grid resolution and the covariance of the measurement noise. That there is a correlation is not surprising. This is because the units of measurement in the grid is based on cells. Thus, as the resolution is increased the number of cells per meter is increased. I.e. the number of cells per object is increased, making the center cell calculation less accurate. As the center cell is the provided measurement to the UKF or KF, its covariance is increased with the resolution of the grid. As the algorithm run using a grid resolution of 100x100 the choice of appropriate noise was limited to R^S_{100} and R^L_{100}.

Section 7.1.1 also revealed that the measurement noise is related to the size of the object it is tracking. This is caused by the same mechanics as in the noise and resolution correlation. Namely that larger objects cover more cells. The measurement noise covariance was then chosen as R^L_{100}, since the system was considered less penalized by overestimating the noise, rather than underestimating it.

8.1.2.2 Process noise

In contradiction to the measurement noise, the process noise is a design choice based on a believed accuracy. This accuracy varies for the types of states. The process noise models in this project are designed such that velocities are a more uncertain state than positions.

Apart from σ^2_ω, σ^2_1, σ^2_2, and σ^2_v are correlated to the resolution of the grid. Thereby, a more accurate design would have a process noise model based on the resolution. However, the process noise model is in this project considered static and independent of the resolution, as the system is designed for a grid resolution of 100x100 cells.

8.1.3 Noise parameters

After having stated and discussed how the resolution and size of objects affects both the measurement noise and the process noise, how to choose the noise covariances becomes an issue to address. Firstly, the ratio between the noise model and the measurement noise determine whether the KF and UKF should rely more on the acquired measurement or the predicted state $x_{k|k-1}$. This means that the purpose of the designed Kalman filters play a role in how the process noise covariance is set. Therefore, prior to determining the noise parameters the role of the implemented KF and UKF ought to be discussed.

The KF and UKF is to compensate for the occupancy grid model and contribute to a more accurate determination of the location of the center of objects. This is especially important when the platform is subject to turning motions. During these conditions the designed systems performs poorly as the function presented in 4.6.2 updates the information of objects in the grid based on a euclidean distance. This is because the euclidean distance will change differently for objects close to the platform and those further away. Another issue with the euclidean distance is that the difference increases in adjacent iterations when the platform is subject to an angular

change, compared to when heading straight.

These turning motions are described more accurately by the coordinated turn model. As this model takes angular velocity into account. Therefore, the process noise model should feature a covariance less than that of the measurement noise covariance. This choice of noise ratio will encourage the UKF to weigh the motion model higher than the measurements, which are the centers in the occupancy grid.

8.2 Using a Compass

The method to estimate corresponding objects between adjacent time iterations presented in section 4.6.2 has been stated to perform poorly when objects are subjects of angular changes. One solution to this was to redesign this method to take the difference in consecutive readings from a compass, already mounted on the platform. However, the readings from the compass turned out to be zero at all times, and it was concluded that there existed some hardware issues with it. Therefore, the euclidean distance in section method 4.6.2 remained unchanged, even though it did not perform optimally.

8.3 Real-time tests

From the results in 7.4 one can see that for this specific test case, the platform performs quite well. However, even though an external computer was used, the processing time is still not fast enough. That being said, the external computer is not designed for these sorts of computations. This could of course be fixed by introducing a more powerful computational unit that does all the calculations etc.

It can also be seen that most of the individual sensors contribute to an increase in overall performance. However, the Ultrasonic sensor is redundant in this case due to the superior performance of the Radar in the same field of view. Therefore, the removal of the ultrasonic sensor is preferred since it reduces the amount of false detections as well as a slight reduction in processing time.

The LiDAR is a crucial part of the platform since it is the only sensor able to detect objects outside of the platform's path. This means that the platform's tracking ability is severely worsened with the removal of the LiDAR sensor. However, the platform is still able to keep away from all sorts of collisions due to the Radar sensors, which are able to detect all objects in front of the platform.

The contribution of the Radar sensors is that it enables the platform to both detect transparent objects as well as objects under the LiDAR's height. Naturally, it will also contribute to other objects in front of the platform as well. However, the LiDAR is superior in the domain where both of the sensors are able to get detections.

8.4 Running in real-time

As the algorithm should be implemented to run in real-time, important aspects are computational power and complexity. This restricts the system heavily. For instance, the occupancy grid is difficult to update every iteration at a resolution higher than 100x100. As a low time per iteration is key to the overall system's performance.

A high time per iteration reduces the performance of the algorithm significantly. Objects are detected and classified less frequently, which in turn yields a less frequent update rate of the occupancy grid. For functions such as the one presented in 4.6.2 an as high update rate as possible is desired. Since it uses a euclidean distance to find best matches between previous and current objects. This distance will increase with the time per iteration and some objects might travel further than the threshold allows. This causes it to find no matches for some objects or simply assign the wrong object some information.

The most exhaustive algorithm of the system is object detection using YOLOv3. Even though it uses tiny YOLOv3 its run time impacts the time per iteration significantly. Therefore, in order to increase performance of the system the camera information stream is only accessed every fifth iteration.

The run time of the system can be viewed in table 7.1. As the time per iteration is much longer than the update of the sensors, buffers are being filled with measurements from each individual sensor. This in turn requires these buffers to be emptied with every iteration in order to receive the most recent measurement sweep.

The buffers are seemingly created in the already existing car system, and when not emptied a delay rendering the algorithm incapable of running the test presented 7.4 is introduced. As this problem seem to appear in the already existing car system the issue was given a low priority. Due to a lack of time, the issue therefore also never became addressed.

8.5 Improvements on previous system

As have been stated, this project is based on a previous object detection and tracking system using a LiDAR and camera. For the company, Infotiv, this project sought to improve the detection ability by introducing new sensors. This the project succeeded on, as an issue with the previous system was its inability to detect transparent objects. Based on the performance when running the developed real-time test in section 7.4, the ability to detect such objects is clear.

Furthermore, an important aspect of the project was study how well the developed algorithm could perform in dark environments. This is also studied in the test case in section 7.4, and can be viewed as the tests run without the camera. From this study it shows that without the camera the system cannot run the test as intended. Since it lacks the ability to classify objects and thus cannot determine harmless from harmful objects. This being said, the algorithm prevents the platform from running into the objects by using the Radar and LiDAR measurements to stop.

Lastly, the algorithm calculated the width of detected objects. This information was never used in this project as a lack of time prevented developing of a collision avoidance algorithm able to turn and plan trajectories. However, the width determined from the predicted objects using the camera images did help reshape objects in the occupancy grid, as stated in section 5.3. Therefore, a conclusion that the knowledge of the width of an object can help refine detections of less accurate sensors and therefore aids in making this algorithm more robust.

8.6 Future work

This section presents some ideas that would be interesting to investigate in the future in order to improve the current algorithm or alternatively if one seeks to design a completely new. To begin with, as improvement to this project, it would be necessary to upgrade the existing hardware. It would enable utilization of the methods to their full potential. Thus enabling one to e.g., add an additional dimension to the occupancy grid used in this project. This would in turn enable height determination.

If it was not for the limitations in computational power, some more advanced methods when it comes to modelling of the occupancy grid would be interesting to try. The first step could be to use the Dempster-Shafer theory of evidence as framework for the grid. Furthermore, investigation of the benefits of implementing different frameworks would also be interesting. An idea would be to attempt to implement Proportional Conflict Redistribution rule #6 (PCR6) and ZPCR6, which takes the degree of intersection of focal elements into account [30].

Today the operating platform cannot run the algorithm. The computational complexity is far to high to run on a raspberry pi. Therefore, a computational node is required, the an external computer, which is communicated with using Wifi. With an upgraded computational unit on the platform the algorithm could be run on the platform without external aid. Reducing any delays and decreasing the number of lost information packages due to a poor connection. This would make the platform an independent vehicle, which is a more accurate resemblance of an autonomous car.

This system uses YOLO as the tool for object detection in images, trained on the COCO dataset. This allows for detection of specific objects, such as vase or toaster. An interesting approach to this part of the project would be to either use other datasets or reconsider the object detection approach in images. For instance, one could implement semantic segmentation and thus classify all cells in the occupancy grid within the cameras field of view. By then labeling the cells as either harmful or harmless, trajectory estimation could be performed using only the images. If this was to be combined with a self localisation algorithm, memory of the cells outside the cameras field of view could be stored. In this sense, an even more advanced grid could be achieved containing more information.

Another interesting feature would be if the algorithm was able to track objects directly within the camera. This could be done by using Markov Decision Processes (MDP) and decision policy implementation enabling tracking of objects even though one frame loose the classification

[31]. Thereafter, several parallel MDPs can be used in order to track multiple objects at the same time. The Tracking within in camera could be fused together with the tracking of object centers in order to further increase the certainty of where the objects are located.

For more realistic tests and verification of the algorithm it should be tested on a real sized vehicle in a realistic environment. The current system is ran in an indoor environment that makes most of the scenarios simpler than out in the real world. The possibility to test and verify the algorithm is crucial when comparing with the state-of-the-art methods.

The thesis has produced an environment, in the form of an occupancy grid, where objects appear along with a classification. The objects are then tracked using a Kalman filter. This environment can be implemented as a decision making base in collision avoidance systems.

One of the features of the project today is that the platform is assumed stationary inside the environment, while the objects are all dynamic. This could be reversed, creating a stationary environment where the platform navigates freely. If combining this with a self localization algorithm, for instance using a Kalman filter, a SLAMMOT algorithm could be developed.

9

Conclusion

This thesis proposes an algorithm for aid in collision avoidance. It uses an occupancy grid as the domain where LiDAR, Radar, ultrasonic, and camera sensory data is fused. The algorithm detects, classifies, and tracks the objects in its surrounding in real-time with a manageable grid. Furthermore, a validation test was designed in order to judge performance of the algorithm. The test case showed that the algorithm can perform object detection and tracking simultaneously. However, it would not be suitable to apply the algorithm in larger scale systems due to its simplistic and naive approach.

The aim of the thesis was to find out if sensor fusion of multiple sensors would improve the performance of an object detection and tracking algorithm, and if any of the sensors are redundant. It has shown that there is a significant increase in performance when combining additional complementary sensors, when comparing this project to a previous algorithm using only LiDAR and camera fusion. An example of such an improvement is the ability to detect transparent objects. However, it could be concluded that when using sensors of significant quality differences, in the same field of view, it introduces more conflicting information than valuable additional information. Thus, the ultrasonic sensor was excluded from the final system.

Conclusively, the developed algorithm requires more powerful hardware in order to run smoothly. A Raspberry Pi is not sufficient to support algorithms this computationally complex. Furthermore, the already existing car control system introduces strange behaviors, such as sensor data buffers or the bad readings from the compass, which worsened the performance of the algorithm. This makes it hard to compare this thesis work to state-of-the-art solutions.

Bibliography

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[2] M. Burns, "'anyone relying on lidar is doomed,' elon musk says," https://techcrunch.com/2019/04/22/anyone-relying-on-lidar-is-doomed-elon-musk-says/?guccounter=1, 2019, accessed: 2019-05-02.

[3] W. Elmenreich, "An introduction to sensor fusion," *Vienna University of Technology, Austria*, vol. 502, 2002.

[4] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte, "Simultaneous localization, mapping and moving object tracking," *The International Journal of Robotics Research*, vol. 26, no. 9, pp. 889–916, 2007.

[5] S.-Y. Chung and H.-P. Huang, "Slammot-sp: simultaneous slammot and scene prediction," *Advanced Robotics*, vol. 24, no. 7, pp. 979–1002, 2010.

[6] D. Nuss, S. Reuter, M. Thom, T. Yuan, G. Krehl, M. Maile, A. Gern, and K. Dietmayer, "A random finite set approach for dynamic occupancy grid maps with real-time application," *The International Journal of Robotics Research*, vol. 37, no. 8, pp. 841–866, 2018.

[7] T.-D. Vu, J. Burlet, and O. Aycard, "Grid-based localization and local mapping with moving object detection and tracking," *Information Fusion*, vol. 12, no. 1, pp. 58–69, 2011.

[8] F. Castanedo, "A review of data fusion techniques," *The Scientific World Journal*, vol. 2013, 2013.

[9] L. Francis, "The basics of neural networks demystified," *Contingencies*, vol. 11, no. 12, pp. 56–61, 2001.

[10] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[12] Z. Chen *et al.*, "Bayesian filtering: From kalman filters to particle filters, and beyond," *Statistics*, vol. 182, no. 1, pp. 1–69, 2003.

[13] R. G. Brown, P. Y. Hwang *et al.*, *Introduction to random signals and applied Kalman filtering*. Wiley New York, 1992, vol. 3.

[14] M. St-Pierre and D. Gingras, "Comparison between the unscented kalman filter and the extended kalman filter for the position estimation module of an integrated navigation information system," in *IEEE Intelligent Vehicles Symposium*. Citeseer, 2004, pp. 831–835.

[15] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. Ieee, 2000, pp. 153–158.

[16] D. Galar and U. Kumar, *EMaintenance: Essential Electronic Tools for Efficiency*. Academic Press, 2017.

[17] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[18] M. Haberjahn and K. Kozempel, "Multi level fusion of competitive sensors for automotive environment perception," in *Proceedings of the 16th International Conference on Information Fusion*. IEEE, 2013, pp. 397–403.

[19] F. Homm, N. Kaempchen, J. Ota, and D. Burschka, "Efficient occupancy grid computation on the gpu with lidar and radar for road boundary detection," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*. IEEE, 2010, pp. 1006–1013.

[20] A. Tchamova and J. Dezert, "On the behavior of dempster's rule of combination and the foundations of dempster-shafer theory," in *2012 6th IEEE International Conference Intelligent Systems*. IEEE, 2012, pp. 108–113.

[21] C. Fernandez, "Grid-based multi-sensor fusion for on-road obstacle detection: Application to autonomous driving," Ph.D. dissertation, Master's thesis, KTH Royal Institute of Technology, Stockholm, 2015.

[22] "YOLO: Real-Time Object Detection," https://pjreddie.com/darknet/yolo/, accessed: 2019-03-24.

[23] "YOLOv3 in Pytorch," https://github.com/ultralytics/yolov3, accessed: 2019-03-02.

[24] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings. 1985 IEEE international conference on robotics and automation*, vol. 2. IEEE, 1985, pp. 116–121.

[25] C. Coué, C. Pradalier, C. Laugier, T. Fraichard, and P. Bessière, "Bayesian occupancy filtering for multitarget tracking: an automotive application," *The International Journal of Robotics Research*, vol. 25, no. 1, pp. 19–30, 2006.

[26] T. Gindele, S. Brechtel, J. Schroder, and R. Dillmann, "Bayesian occupancy grid filter for dynamic environments using prior map knowledge," in *2009 IEEE Intelligent Vehicles Symposium*. IEEE, 2009, pp. 669–676.

[27] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, vol. 3.

[28] M. Roth, G. Hendeby, and F. Gustafsson, "Ekf/ukf maneuvering target tracking using coordinated turn models with polar/cartesian velocity," in *17th International Conference on Information Fusion (FUSION)*. IEEE, 2014, pp. 1–8.

[29] M. R. Morelande and N. J. Gordon, "Target tracking through a coordinated turn," in *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 4. IEEE, 2005, pp. iv–21.

[30] J. Dezert, J. Moras, and B. Pannetier, “Environment perception using grid occupancy estimation with belief functions,” in *2015 18th International Conference on Information Fusion (Fusion)*. IEEE, 2015, pp. 1070–1077.

[31] Y. Xiang, A. Alahi, and S. Savarese, “Learning to track: Online multi-object tracking by decision making,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4705–4713.

A

Appendix 1

A.1 Sensors

A.1.1 LiDAR

The LiDAR used in the project is a RPLIDAR-A2 which is a low cost 360 degree single layer LiDAR.

Distance range	0.12-12 [m]
Angular range	0-360°
Distance Resolution	0.5 [mm] - 1% of total distance
Angular resolution	$0.9°$ at 600 [rpm]
Sample frequency	2000-8000 [Hz]
Power Voltage	5 [V]

Table A.1: Specifications for RPLIDAR-A2

A.1.2 Radar

For radar detections, a sensor chip from Acconeer will be used. This is a short range pulse coherent radar sensor that can be connected to a connector board and then directly into a Raspberry Pi. The connector board supports four radar sensors which makes it possible to cover a large area.

Distance range	<2 [m]
Angular width	80°
Relative accuracy	$\mu m - level$
Frequency	60 [GHz]
Power Voltage	5 [V]

Table A.2: Specifications for XC112, XR112, and A111

A.1.3 Camera

The camera that is mounted on the vehicle has a macro fish lens with 0.67x magnification. This enables us to utilize a really wide space in front of the vehicle. The cameras resolution is 640x480 which is high enough to make good detection with the neural network YOLO.

A.1.4 Ultrasonic sensor

Distance range	0.02-5 [m]
Effective angle	$< 15°$
Distance Resolution	3 [mm]
Cycle period	> 50 [ms]
Power Voltage	5 [V]

Table A.3: Specifications for HC-SR04