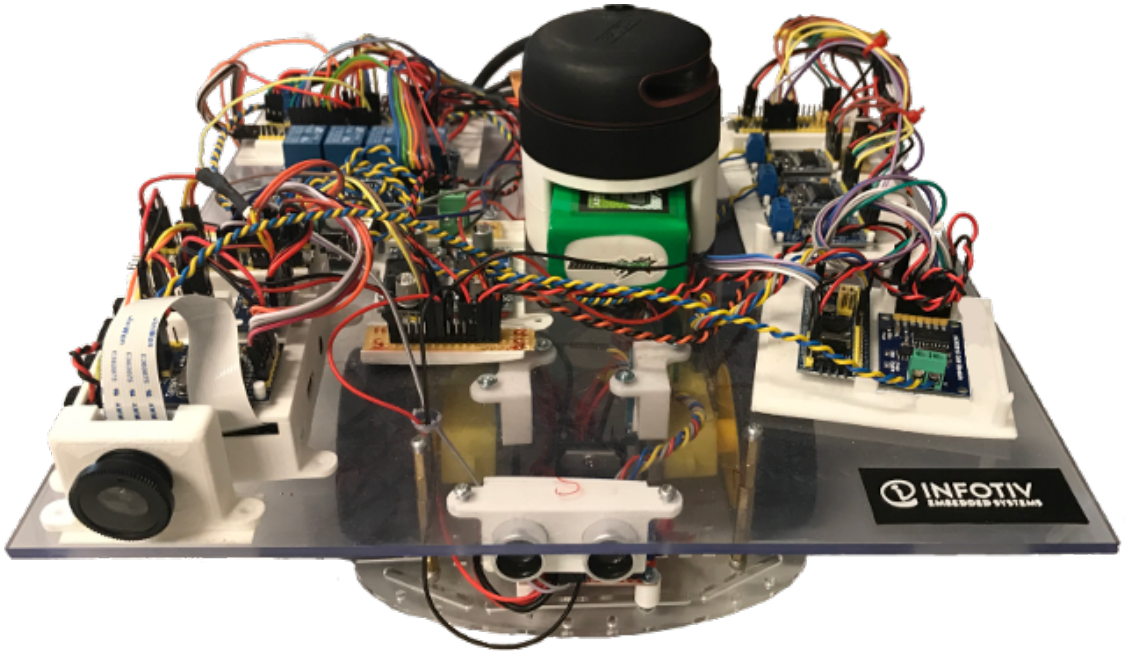
CHALMERS
UNIVERSITY OF TECHNOLOGY



Smart Collision Avoidance for Autonomous Vehicles

Thesis Report.

Complex Adaptive Systems

Gabriel Wagner

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2018:SEEX30

This report describes the process of researching methods and algorithms that can be used in a collision avoidance system. Furthermore it documents how such a system was implemented using some of the researched methods. The system uses a image recognition and lidar sensors mounted on a miniature vehicle. The system produces a steering signal which is sent back to the vehicle to make it stop and avoid collisions.

GABRIEL WAGNER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Chalmers University of Technology
Gothenburg, Sweden 2018

Supervisor: Olle Norelius, Infotiv AB
Examiner: Claes Andersson, Physical Resource Theory, Department of Space, Earth and Environment

Master's Thesis 2018:NN
Department of Space, Earth and Environment
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Master Thesis report of the implementation of a Collision Avoidance System, using lidar and Image recognition.
GABRIEL WAGNER
Department of Department of Space, Earth and Environment
Chalmers University of Technology

Abstract

One of the hottest topics in tech right now is self-driving cars. A controversial subject which will no doubt bring great change to our traffic systems. But before autonomous vehicles can take over the roads they must meet a high level of safety standards. To do so they require multiple systems and sensors that can detect and react to their surroundings. The systems can be categorized into what is known as active safety, as opposed to passive safety like seat belts. In this thesis the development of a collision avoidance system is documented, which is a type of active safety. First some initial research is done on what algorithms and techniques can be used to build a collision avoidance system. Then a proof of concept system is developed, in cooperation with the company Infotiv. The system is developed for a miniature vehicle built by Infotiv, which has lidar and camera sensors. The input to the system is sensor measurements and the output is a steering response, which either stops or turns depending on the object in front of the vehicle. By running the camera feed through a image recognition network the system is able to tell what type of objects are present, and using lidar find the distance to them. The finished product is a distributed modular system, which successfully allows the miniature vehicle to avoid colliding with obstacles. It can identify specific types of objects and react in different ways depending on the type of object. In conclusion the project shows that it is possible to build a robust light-weight, modular collision avoidance system. The system is based on techniques such as sensor fusion, occupancy grid mapping and image recognition.

Keywords: autonomous vehicles, image recognition, sensor fusion, occupancy grid, collision avoidance, active safety

Acknowledgements

I would like to thank Pierre Ekwall and Infotiv Embedded Systems for giving me the opportunity to work on this project. I would also like to thank my examiner Claes Andersson for helping me see the project through an academic perspective. Lastly a big thanks to Olle Norelius for being an excellent supervisor and mentor, interesting discussion and helping me during the entire project.

Gabriel Wagner, Gothenburg, June 2018

Contents

List of Figures

1
Introduction

This is the introductory section in which contains the background of the subject and a small presentation of some of the cutting edge systems we have today. Additionally the purpose and problem statement of the thesis with delimitations and a disposition are presented.

The modern cruise control was invented 1948 by Ralph Teetor[1]. It is one of the earliest systems which allowed drivers to automate a small part of driving. Since then many new systems have been added to modern vehicles to make driving easier. The next big leap on the horizon may eliminate the need for a human driver completely. With the help of advanced sensors and new developments in image recognition autonomous vehicles are no longer strictly science fiction.

Companies like Google and Volvo already have prototypes driving autonomously in traffic[2]. But they are not perfect, and there are are still many risks with rushing them to market, even if a human passenger is ready to intervene. Proven by multiple mishaps, such as the unfortunate accident in Tempe, Arizona, in which a prototype vehicle was part of a fatal accident[3] (March 18 2018). Many issues remain that need to be solved before autonomous vehicles outperform the average human driver.

This thesis will take a closer look at a system commonly referred to as collision avoidance system. The process contains research of the different methods and algorithms that can be used and leads into documenting and developign an implementation using some of the researched methods. To test the system a miniature vehicle will be used, developed by Infotiv AB, which has distance sensors and video camera capabilities. The final product will be a smart collision avoidance system which tells the miniature vehicle to stop or turn to avoid obstacles, by determining not only distance but what the object is.

1.1 Background

It's not uncommon to see headlines about autonomous vehicles, one of the hottest topics in tech right now. Industry and academia are both researching how to create a self-driving vehicle using modern technology. Perhaps one of the biggest reasons is that autonomous vehicles are said to be not only safer but also better for the environment[4]. One example how they are safer is by eliminating human errors such as falling asleep or sending text messages to name a few.

The environment aspect comes largely from having car to car communication. Traffic information can instantly be transmitted and incorporated into automated

path planning. By giving cars an awareness of large pools of cars around them, they can start travelling more like schools of fish which is far more efficient and can alleviate traffic jams. More efficient driving requires less fuel, which is better for the environment.

However before getting seduced by the long list of benefits there are many issues that need to be dealt with. Currently autonomous vehicles are in a dangerous middle ground. Prototype vehicles usually have human passengers that can intervene, but fail to do so based on believing the vehicle is completely autonomous. In addition there are still technical issues with sensors and while they are perfected there is a high risk of more casualties. The new technology and autonomous vehicles will also require new laws and regulations to make sure they are used safely. Needless to say autonomous vehicles will bring many changes to how we interact with and handle transportation.

1.1.1 Driver Assistance State of the Art

Modern vehicles already have multiple independent systems that assist the human driver. Some cars even go as far as overriding the driver and taking control in the event of a potential accident. Most systems rely on sensors of some kind, requiring an awareness of the vehicles situation. Below is a list of some of the recent and modern systems that can be found in cars today:

- **Adaptive Cruise Control** is one of the most popular systems for driver assistance. It utilizes radar placed in the front of the car, which can detect other vehicles in front of the car and adjust velocity to keep a set distance.
- **Lane Change/Departure Systems** use cameras and image processing algorithms to detect when the vehicle is leaving the lane. For example an algorithm can detect the white markings between lanes. When the distance to the marking is too low, the system can tug the steering wheel in the other direction. Some systems can even go as far as switching lanes autonomously.
- **Collision Avoidance** systems usually use a combination of radar and cameras to detect pedestrians, cyclists and other vehicles. When the system detects an impending collision it initially warns the driver. If a collision is imminent the system may even trigger an emergency brake.
- **Parking Assistance System** help drivers by detecting near-by objects and warn the driver. Most systems use an ultra-sonic sensor to detect the distance to nearby objects, issuing a warning to the driver when the distance is too small. Additionally many cars have a rear view camera to assist the driver further. Som manufacturers even provide fully autonomous parking systems.

1.2 Problem Statement

Collision avoidance is a broad term for any system that prevents collisions, it follows that there are multiple ways to build such a system. A specific implementation depends on the requirements of the project, such as hardware limitations and performance needs. The problem is knowing which methods and techniques fulfill those

requirements. In this thesis a focus will be given to having a light-weight application which could potentially run on a single-board computer.

1.2.1 Objectives

The objectives of the thesis are based on the requirements and scope given by Infotiv AB who requested this project.

- Research, design and implement a system which can determine whether or not a vehicle should stop, based on sensor readings and camera footage. The finished system should serve as a demo which can later serve as a platform for teaching and giving insight about collision avoidance technology.

To make the objective more clear and defined some sub-goals were chosen:

- Use lidar and image recognition to determine distance and type of objects.
- Implement the system with a modular approach, where each sub-system works as independently as possible.
- Research and use occupancy grid mapping for sensor fusion of image recognition and lidar sensor.
- The system should be able to recognize at least 2 objects, to which the system can react differently. Example brake for object 1 and keep driving for object 2.
- Use the system in a live-demo where the recognition and reaction is tested. Example: Traffic situation where the different objects are placed in front of the vehicle.

1.3 Purpose

The main purpose of this research is to implement and test a collision avoidance system. In addition some broad research is done of what methods and techniques can be used for different parts of the system. Lastly the project will serve as a proof of concept and learning platform for Infotiv.

1.4 Delimitations

This study is delimited to research different methods but only implementing and testing chosen methods that do not overlap. For example, while there are multiple ways to perform image recognition only one method will be implemented, while a few will be researched. The system is purely created as software, any hardware requirements are provided or built prior to this project.

1.5 Disposition

In the following chapter the theoretical framework behind the project are presented. Thereafter, in chapter 3 a method description is given of how the implementation was done of the collision avoidance system. Next the results of the project are

presented, showing what the final versions of each module produces and how they fit together. Lastly, a discussion is held analyzing the results in reference to the goals. Some ideas for future work, places that can be improved or extended.

2
Theory

This chapter presents the underlying theoretical concepts that lay the foundation for the rest of the thesis.

2.1 Autonomous Platform

The Infotiv Autonomous Platform (IAP) is a miniature vehicle system made for research and education at Infotiv. It has multiple electrical control units (ECU) which communicate on a controller area network. The different units are divided in a way which mimics the electrical system in a modern Volvo car. The IAP has three sensors; lidar, ultrasonic and camera which are connected to an on board Raspberry Pi. The vehicle can be seen in figure 2.1

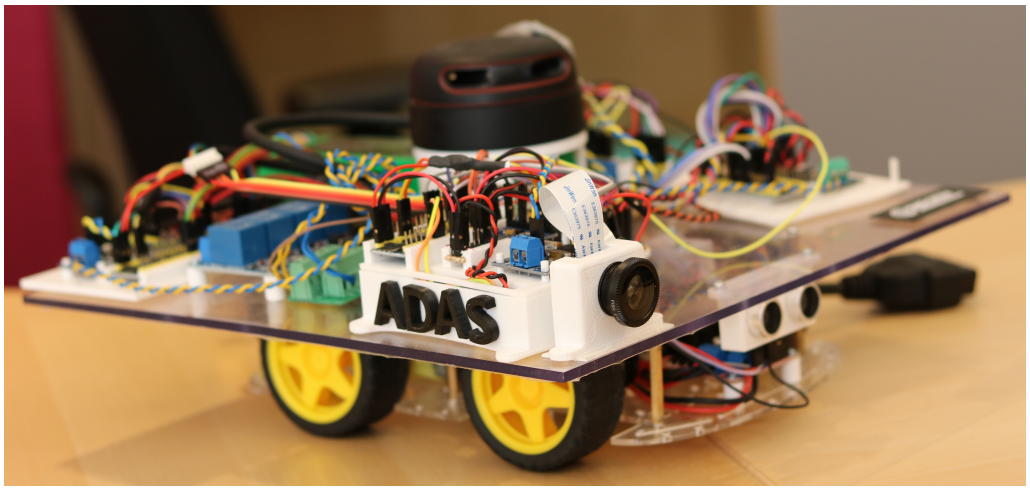


Figure 2.1: The Infotiv Autonomous Platform The IAP has four wheels for moving around the environment, each wheels has a small motor. Mounted on top of the bottom wheel-plate is a 30×30 cm plate which holds the different ECUs and sensors. In the closest corner the camera lens can be seen which is connected to the on board Raspberry Pi computer, marked ADAS (Advanced Driver-Assistance System). Behind the computer the lidar sensor can be seen, a black and red cylinder-like object.

Sensor readings from the different sensors are formatted and published using wifi and socket communication. Messages can also be sent to the on board computer to set speed, and turn rate among others. A Python package was developed by Infotiv prior to this project to make communication easy to and from the IAP. The package wraps the network communication into simple function calls using the

Python language. For outside applications to receive sensor data the Python package is imported and used to make requests. A reply is given when a complete reading has finished of the requested sensor type. It is important to know the different formats of the sensor replies. The camera reply is a single frame image with a resolution of 640×480, in jpeg format. Lidar data is formatted into an array of 400 readings, which is one rotation of the laser. Each reading is a tuple of three values: angle of the laser, distance to detected point and quality of the reading. If nothing was detected by the laser it is set to the max distance of the lidar sensor.

2.2 Image Recognition

Nearly all state-of-the-art image recognition algorithms are based on convolutional neural networks (CNN). That has not always been the case, in 2011 and 2012 CNNs started showing up in image recognition competitions. For example in the 2012 ImageNet challenge, a CNN crushed the competition with its error rate of 16%, which was 9% better than the previous year winner. Since then CNNs are the go to method for image recognition and object detection. The latest networks have improved the technique greatly and reach error rates as low as a few percent.

Convolutional neural networks are a class of deep, feed-forward networks that handle data which can be structured into two-dimensional shapes, such as images. Researchers all over the world have constructed different types of networks to try and find a high performing solution for rapid image recognition. When it comes to real time detection some of the most well known networks are RetinaNet, FastRCNN and Yolo. Initially most networks were so called two-stage detectors, but recently single-stage detectors have risen in use. Both types are region-based proposal system, meaning they first detect objects in an image and then classify the types of objects that have been detected.

2.2.1 Yolo

Yolo (short for ”you only look once”) is a single-stage region proposal classification network originally built by Joseph Redmon, Ali Farhadi Santosh Divvala and Ross Girshick[5]. The network works real-time, in other words the detection process is fast enough to work continously on an image stream. Speed often comes at the cost of accuracy, the speed at which detections can be made is lowered by deeper but more accurate network configrations. To handle this the network can be configured in multiple ways, most notably the creators have made a normal and tiny version. The normal version is a deep neural net, using more than 20 layers and is capable of performing detections at 45 frames per second. The tiny version has a lower depth, making it faster but less accurate, and can make detections at 155 frames per second. Both benchmarks are when used with a high teir graphics card.

The key difference from other state-of-the-art classification networks, such as fastRCNN is the number of stages. FastRCNN first stage produces object detections using one type of network, then runs the detection through a classification network. In contrast, Yolo runs the image through the network once, meaning it is more like a fully convolutional neural network.

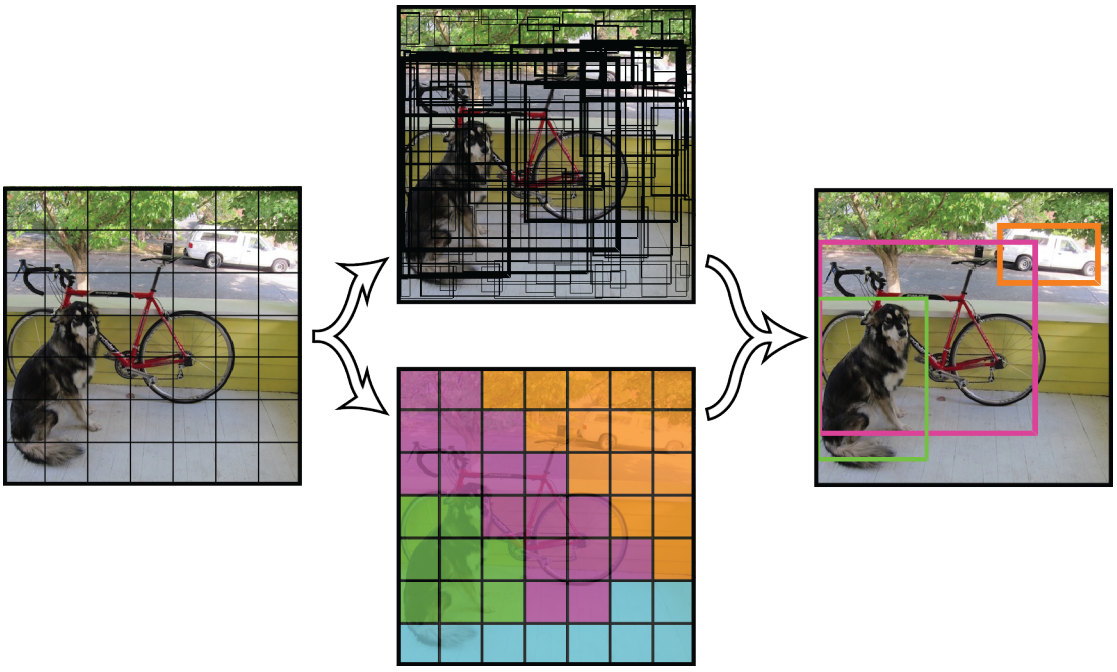


Figure 2.2: The two part process of Yolos detection. The first box shows the original image divided into grid cells. Each cell predicts a number of bounding boxes which have different confidence of containing an object, visualized by the thickness of the boxes in the top center box. In parallel each grid cell also predicts which type of object it contains, shown as colors in the bottom center box. Lastly the boxes with the highest confidence are matched with their respective labels to produce labeled bounding boxes, seen in the furthest right box.

Yolo accomplishes its one-stage process by splitting the image into a grid with multiple cells. For each cell a number of object detections are made, in the form of bounding boxes. Each box is given a confidence level, which corresponds to how likely the box is to contain an object. Next each cell is assigned a probability of belonging to a specific type of object, like a dog or a car. Finally the two parts are merged, meaning each bounding box is assigned a class, then filtered out based on a threshold for the confidence. The filtering removes any bounding boxes which are likely to be empty. The process has been visualized in figure 2.2.

The design choices behind Yolo also come with some limitations of the classifications made by the network. As previously mentioned the image is divided into cells and only one type of object can be assigned to a cell. This method has problems with objects that are in close proximity, leading to only one of the objects being detected properly. In addition the training model for bounding boxes depend on object orientation, meaning objects that are rotated or oddly scaled are harder to detect.

2.3 Occupancy Grid Mapping

Sensors are never ideal in reality, meaning readings come with a degree of uncertainty. By combining sensor data from different sensors the uncertainty can be

minimized. The technique is called sensor fusion, and is commonly used technique in robotics. A popular method to fuse sensory data is occupancy grid mapping, which creates a representation of an environment. The representation can be used by a robot to navigate in its surroundings.

2.3.1 Occupancy Maps

A common data structure used for mapping is a grid, which splits a real map into discrete cells. In occupancy grid maps the cells contain some value which corresponds to whether or not there is an object in the way. This is commonly done using a simple binary value; zero or one, *empty* or *filled*. This method could for example be used to represent a road, where the road is represented using *empty* and all other cells are *filled*. The state of the cells are what is called occupancy, if a cell is *filled* it is occupied and can not be traversed.

Before a robot can use an occupancy grid the map has to be constructed using some method. There are a few different approaches, perhaps the most simple way is to build a map "manually", in other words give the robot a built map created by some external process or person. Another way to produce a map is to let the robot use sensors to build a map on the go. Either way a grid map is built by assigning cells to either be *filled* or *empty*. If the robot is building a map with sensors, some method of conversion is required to turn sensor readings into occupancy values. The conversion is called an inverse sensor model, which will be explained later in section 2.3.3.

A simple inverse sensor model could be to simply have each cell be a counter of how many times an object has been detected in each respective cell. When a cell has many hits the probability is high that something is occupying that cell. Unfortunately this method quickly has problems with overflow, since the number of hits will continue to grow. To solve the overflow problem the occupancy grid map algorithm was developed by by Efes and Moravec in the 80s [6]. The method uses probability theory to assign each cell with probabilities instead of counters.

2.3.2 Recursive Bayesian Estimation

The method will be described in the context of a vehicle, which requires some notations:

- Vehicle pose - At a given time t the pose is a K-dimensional vector:

$$\mathbf{x}_t = \left(x_t^1, x_t^2, ..., x_t^K\right)^\intercal$$

- Sensor measurement - at a given time t a measurement from a sensor S is a N-dimensional vector:

$$\mathbf{s}_t = \left(s_t^1, s_t^2, ..., s_t^N\right)^\intercal$$

- Map - A two-dimensional grid:

$$\mathbf{g} = \{g_{ij} : 1 \leq i \leq G_H, 1 \leq j \leq G_W\}$$

  Where G_H, G_W are the number of columns and rows.

The main purpose of the occupancy grid algorithm is to estimate:

$$p(\mathbf{g}|\mathbf{x}_{1:t}, \mathbf{s}_{1:t}) \tag{2.1}$$

Which is the probability that the grid is in a specific state given all measurements taken and each vehicle pose. Given a grid of n binary cells there are 2^n possible grid configurations, which is computationally difficult to deal with.

The problem can be simplified using the assumption that each cell is independent of all other cells. The assumption is easy to disprove; if an object occupies more than one cell, the cells all depend on the same object, it follows that they are not independent. Despite being a flawed assumption, it is sufficiently accurate for the purpose of the occupancy grid algorithm. The goal of the algorithm is now to find the probability of each cell instead:

$$p(g_{ij}|\mathbf{x}_{1:t}, \mathbf{s}_{1:t}) = \frac{p(\mathbf{x}_{1:t}, \mathbf{s}_{1:t}|g_{ij})p(g_{ij})}{p(\mathbf{x}_{1:t}, \mathbf{s}_{1:t})} \tag{2.2}$$

Which can then be used to estimate the original probability of the entire grid:

$$p(\mathbf{g}|\mathbf{x}_{1:t}, \mathbf{s}_{1:t}) = \prod_{i}^{G_W} \prod_{j}^{G_H} p(g_{ij}|\mathbf{x}_{1:t}, \mathbf{s}_{1:t}) \tag{2.3}$$

The above expression is also known as a Binary Bayes Filter. The result is a recursive formula in the form of a log-odds expression[7]:

$$\begin{aligned} l_t(g_{ij}) &= \log\left(\frac{p(g_{ij}|\mathbf{x}_{1:t}, \mathbf{s}_{1:t})}{1 - p(g_{ij}|\mathbf{x}_{1:t}, \mathbf{s}_{1:t})}\right) \\ &= l_{t-1}(g_{ij}) + \log\left(\frac{p(g_{ij}|\mathbf{x}_t, \mathbf{s}_t)}{1 - p(g_{ij}|\mathbf{x}_t, \mathbf{s}_t)}\right) - \log\left(\frac{p(g_{ij})}{1 - p(g_{ij})}\right) \end{aligned} \tag{2.4}$$

The formula is also known as the occupancy grid update rule, because it defines how each cell of the grid is updated with each iteration. The first term on the right hand side is the previous cells grid map value $l_{t-1}(g_{ij})$. Next is the log-odds of the distribution of the current measurements which is known as the inverse sensor model. Lastly the prior probability which is a type of initialization, commonly set to a constant $p(g_{ij}) = 0.5$ which implies all cells are unknown at the start. The update rule only requires knowledge of the previous grid and the inverse sensory model, as opposed to keeping track of a full history.

The formula works well for static objects, since each iteration will compound the likelihood that the cell is occupied. But for moving objects there is a problem with "switching" cells, and not building up a confidence. To remedy the problem a decay factor δ can be applied after the update rule:

$$p(g_{ij}) \leftarrow (p(g_{ij} - 0.5) * e^{-\frac{\Delta}{\tau}} + 0.5 \tag{2.5}$$

Where Δt is the update interval and τ is the time constant of decay. The decay factor is a measure of how fast the grid forgets old measurements and goes back to the *unknown* state, in the absence of new measurements.

2.3.3 Inverse Sensor Model

To convert sensor readings into occupancy grid values an inverse sensor model is required. The sensor models are in general a probability distribution, but may differ greatly between different types of sensors. For example there is a big difference in how a lidar and an ultrasonic sensor behave, meaning their models will behave differently as well. A lidar sensor spins rapidly and sends laser beams at set intervals. The beams bounce off objects and return to the sensor, which is then able to calculate the distance each beam travelled. The result is a 360 degree distance measurement.

Each measured distance can be seen as a point $\mathbf{p} = (\alpha_i, r_i)$, where α is the angle the beam was sent at and r_i the measured distance. A fitting model for the lidar is a probability distribution which is applied for each point. Instead of only setting the probability of occupancy near the point, the model can be applied on the entire line leading up to the point. A simple example could be a rectangular step at the distance of the detected object:

$$p(g_{ij}|\mathbf{x}_t, \mathbf{s}_t) = \begin{cases} 0.1 & r < r_i - \epsilon \\ 0.9 & |r - r_i| <= \epsilon \\ 0.5 & r > r_i + \epsilon \end{cases}$$

Where r is a trace along the line, and r_i the distance to the detected point. ϵ, the width of the rectangle, can be seen as the resolution of the detection. A better model is usually a Gaussian, instead of a rectangular step, which gives a smooth distribution. The different types of models can be seen in figure 2.3.

2.4 Path Finding

Path finding is a crucial part of a robots navigation capabilities. Most path finding algorithms are a subset of graph search algorithms, a simple example is the breadth first search. Breadth first search will check all nodes to eventually find a path, which leaves room for improvement in performance. Perhaps most famous is Dijkstra's algorithm for finding the shortest path between two nodes. Another example is the Artificial Potential Field algorithm, which models obstacles as potentials. The path is then generated by letting a virtual particle move while under the influence of the artificial potentials.

2.4.1 A*

A* (pronounced A-star) is a commonly used algorithm for finding the shortest path between two nodes in a graph. It uses a heuristic function to extend Dijkstra's algorithm, by doing so it can reach the solution faster. The algorithm falls under the category best-first search, where the goal is to find the path between two nodes that have the lowest cost. To find the solution the algorithm tests all possible paths, but it does so in an order that is always the lowest total cost first, described by the function:

$$f(n) = g(n) + h(n) \tag{2.6}$$

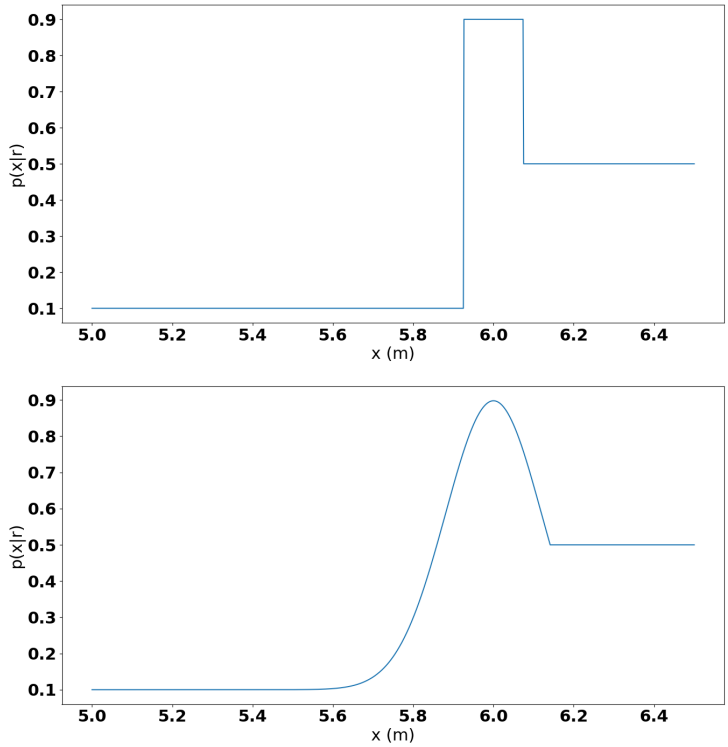


Figure 2.3: Example of two inverse sensor model distributions. Top: rectangular distribution around a detected object at $x = 6$m. Bottom: to account for noise a Gaussian is used instead of a rectangular distribution around the detected object at $x = 6$m.

Where $g(n)$ is the cost from the start node to the node n and $h(n)$ is the heuristic cost. The heuristic function estimates the cost of the remaining path leading to the goal. If the algorithm is a blindfolded person searching for a goal node, then the heuristic function acts as a guide saying hot or cold as the person tests different paths. When the end node is reached $f(n)$ has been minimized and the nodes that were traversed make a minimal path.

In most robot applications nodes exist in a two or three dimensional space, in which case a natural heuristic function would be the euclidean distance to the goal node:

$$h(n) = |\vec{x}_n - \vec{x}_g| \tag{2.7}$$

$\vec{x}_n$ is the coordinate of a node n and $\vec{x}_g$ the coordinate of the goal node g. By minimizing the distance to the goal the next tested node will be guaranteed to be the closest to the goal, which has a good chance to be the best path.

2.5 Bresenham's line algorithm

Bresenham's algorithm is used to approximate a continuous line between two points into discrete segments. Commonly used to draw line primitives in bitmap images on screens, where pixels are the smallest discrete segment. By using integer addition, subtraction and bit shifting the algorithm is able to efficiently compute the approximated line. The algorithm is one of the earliest to be developed in computer

graphics.

A line is defined by its two endpoints: (x_0, y_0) and (x_1, y_1). These can be seen as discrete indexes in the image matrix, where the first value is the column and the second the row. Assume that $x_0 < x_1$, $y_0 < y_1$ and that $|x_1 - x_0| > |y_1 - y_0|$. In other words that the line has a positive slope with an absolute value of less than one. For each column x_i between x_0 and x_1 there is exactly one row y_i which matches the fractional line value. The opposite is not true, each row between y_0 and y_1 may contain multiple rastered pixels, for example a near horizontal line. The algorithm chooses the column index y_i which is closest to the fractional line value, in other words the best approximation. A visualization can be seen in figure 2.4. The fractional value of the line can be found by constructing the linear function which tangents the original line:

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0 \tag{2.8}$$

Any line can be transformed to work with the initial assumption, which allows any type of line to be approximated.

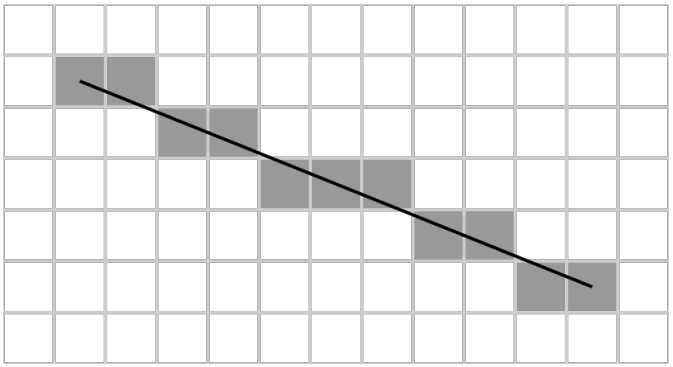


Figure 2.4: Visualization of Bresenham's line algorithm. The greyed grid cells are the best discrete approximation of the line.

3
Methods

This chapter describes the development of the collision avoidance system. The system was built in different parts which are each presented separately.

3.1 System Overview

Before development started an early sketch was made of how the parts of the systems could be built, based on the requirements and goals (presented in section 1.2). After a brief literature study and research phase, some early design decisions were made:

- The system will be divided into three main modules, which handle a image recognition, occupancy grid mapping and steering respectively.
- Modules will communicate using sockets, specifically a publish subscribe pattern.
- The image recognition module will use the Yolo network for visual object detection.

A sketch of how each module interacts has been visualized in a diagram in figure 3.1.

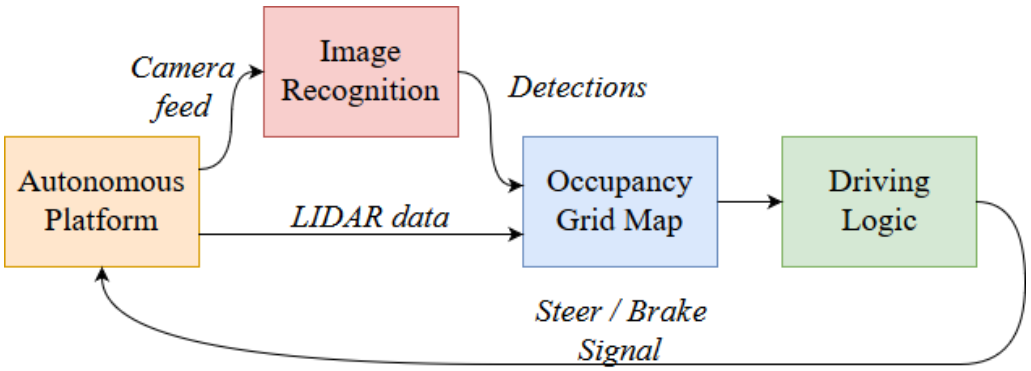


Figure 3.1: Block diagram of modules that together build the Active Safety system. The first block, from the left, is the IAP (orange) which outputs the camera feed and lidar data. The next block (red) manages image recognition and produces detections from the camera feed. The occupancy block (blue) uses the lidar data and detections to handle an occupancy grid. The last block uses the occupancy grid to make steering decisions which are sent back to the IAP.

3.2 Computer Vision

The Computer Vision module has three main tasks:

- Subscribe to an image stream service (like the IAP camera).
- Run the image through the Yolo network to produce detections.
- Publish the detections to all subscribers.

The module was built as a Python script which runs continuously at a set update frequency. To implement the Yolo network a library called Darkflow (a wrapper for TensorFlow) was used, which handles loading of the network and some pre-trained weights. The weights and configuration were taken from the website of the Yolo creator, as opposed to training custom weights. After loading the network and weights, a processing thread starts which handles incoming and outgoing communication.

When the IAP publishes an image it is received by the image recognition module. The image is processed through the detection network and a list of detections is produced. A detection in this case is a combination of a label and a bounding box. A label is a single word which describes what the detected object is, for example a person or car. The bounding box is actually just the pixel coordinates of the corners of the box, the box in turn specifies the location in the image of the detected object. When the list of detections is produced it is immediately published to all subscribing clients.

3.3 Occupancy Grids

The occupancy grid module is the focal point of the system, collecting different types of sensor data and fusing them into a single data structure. Three main parts make up the module; the sensor modelling, initialization and continuous updating.

3.3.1 Sensor Models

Both the lidar and camera sensor required inverse sensor models. As described in section 2.1, lidar data was published as a list of tuples. Each tuple was used to produce cell grid values in a line up to the detected point. The line was made using Bresenham's line algorithm, with the center of the grid (which is the sensor position) and the detected point as end points. The last part of the model was setting each cell along the line according to a probability distribution. At first a Gaussian distribution was used, but later proved unnecessary due to the relatively low resolution of the grid. It was replaced by a two-step rectangular distribution, which can be seen as a discrete Gaussian approximation. The distribution can be see in figure 3.2 in appendix.

The camera model was less straight forward due to not operating in the occupancy grid plane. The spatial component of the image detections were the bounding boxes, which exist in a plane perpendicular to the occupancy grid. Since there is no distance measurement for a detection, the possible space in which a detection can exist is a pyramid shape with an apex at the camera. The base of the pyramid is the bounding box of the detection. To better explain this relation it has been visualized

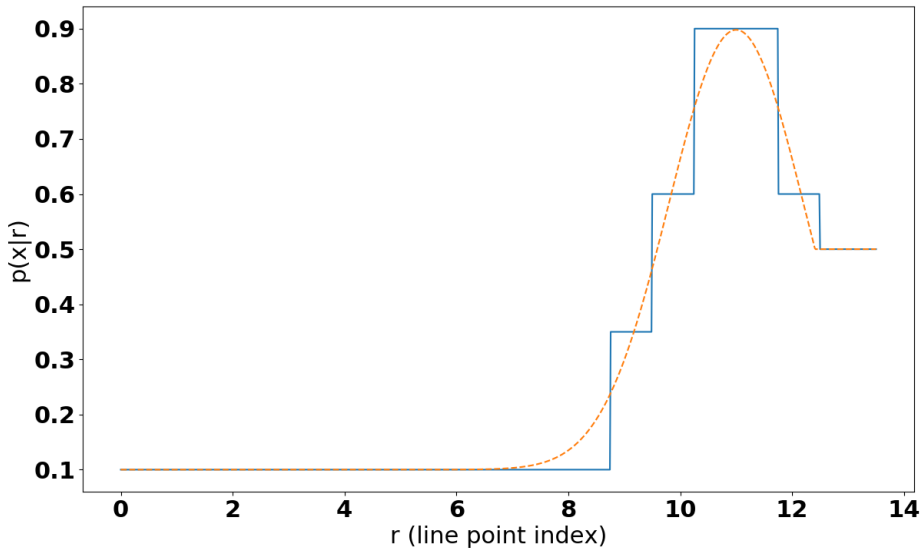


Figure 3.2: Inverse sensor model for lidar (line), approximating a smooth Gaussian (dotted). Sensor model applied on a discrete line with points indexed by r. The lidar sensor has detected an object at line index $R = 11$.

and can be seen in figure 3.3. If the pyramid shape is projected onto the occupancy grid plane a triangle shape is formed which. The triangle is the best approximation of where the detection can be, in the occupancy grid.

The triangle can be parameterized using two angles, the start and end of the apex. The max values of these parameters are the camera image itself, which in turn is bound by the field of view of the lens. Measuring the field of view is hard directly, but can be instead be done indirectly by measuring the distance to an object which covers the entire image. Next the object is measured, the two distances can be used to find the angle of the field of view, using simple trigonometry. After finding the distance to the object and the length of the object, the angle could easily be found using trigonometry. One last step was required to convert bounding boxes to the parameterized angles. The lens distorts the image to achieve a higher field of view by using a fish eye lens. To undistort the pixel coordinates a stereographic projection was used, which assumes the pixel coordinates lay on a sphere.

Now each detection's bounding box could be turned into a pair of angles, which will be referred to as bounding angles. The bounding angles were used to create a circle segment with a set radius. To find the grid cells within the circle segment multiple lines were drawn at angles that interpolate the bounding angles, using Bresenham's line algorithm. While some cells due to rasterization, most of the circle segment area was covered. Each cell in each line was set to *occupied*, meaning a value of 0.9.

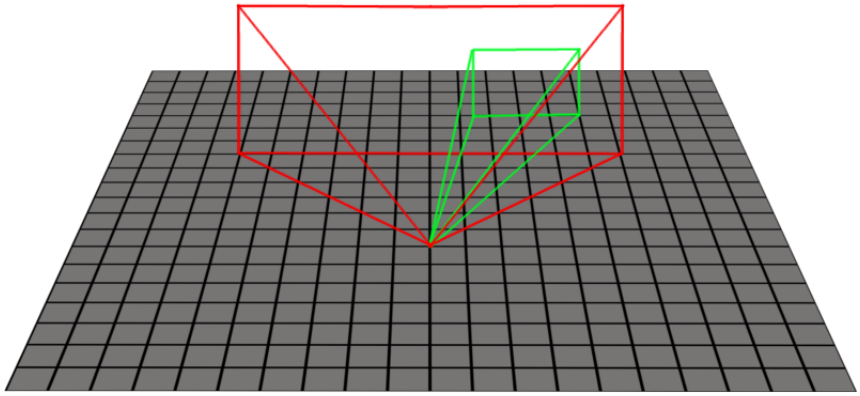


Figure 3.3: A visualization of how a camera view is perpendicular to the grid plane. The larger red pyramid represents the full camera view, while the smaller green pyramid a single detection.

3.3.2 Initialization

As mentioned earlier the module handles incoming sensor readings from the autonomous platform and detections from the computer vision module. If either the camera or the lidar for some reason was not producing data, the occupancy grid module should still do its best to produce grids. To manage this each incoming data stream was given a separate processing thread, for similar reasons a thread was made for the occupancy grid update process. In total the module contains three separate threads:

- Image Recognition Subscription
- Autonomous Platform Subscription
- Occupancy Grid Update

The subscription threads task is to wait for incoming data and assign the data to a container, which the update thread uses to create occupancy grids. To keep the container thread-safe the update thread only reads data, while the subscription threads always write to two separate variables in the container.

The initialization process uses some pre-configured values, such as lidar max range, grid size and ip-addresses. Another setting which is configured is a list of labels which determine which labels are used to build occupancy grids. Any other detections made by the detection module are discarded. The list is refered to as the ”label filter”. Next a dictionary of grids are initialized to the set size, with default cell values 0 (the log-odds value of *unknown*). After all the needed data structures and threads are created the threads are started and the iterative loops begin.

3.3.3 Updating

As mentioned the subscription threads handle incoming data and make sure the latest values are stored. The update thread reads the data and creates an occupancy grid for each label in the label filter and one grid from lidar data. The next step is to divide the lidar data among the camera grids, in other words fuse lidar and

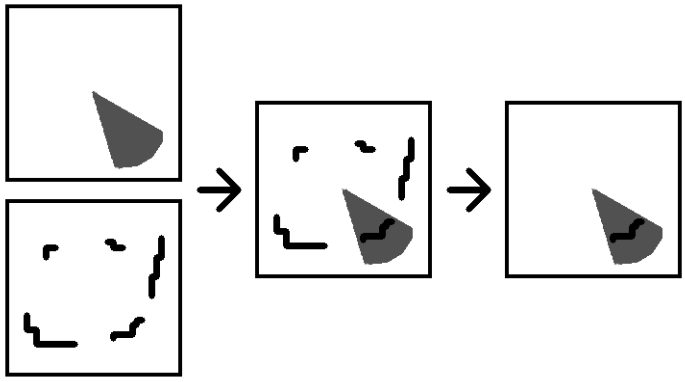


Figure 3.4: Sensor fusion of camera detection and lidar data. From left: Top box shows an occupancy grid with a camera detection circle segment. The bottom square shows an occupancy grid with lidar data. The middle square shows the two grids superimposed, notice the overlapping detections in the bottom right corner. The last box shows the fused grid, the remaining *occupied* cells are the overlapping cells from the two grids.

detections. By multiplying the lidar grid onto each labeled grid the fused grids maintain high probabilities in places that overlap, visualized in figure 3.4. This was based on Bayes rule, using the detection data as a prior distribution and the lidar data as a likelihood.

The newly created grids are based on only the latest data, meaning they need to be added to the previous grid according to the update rule. As shown in equation 2.4 the updated grid is produced by adding the newly produced grids to the previous grids. In the update process the grid cells have log-odds values, since that is the form they were needed to update. To convert the grids back to probabilities they were normalized. Since the decay was applied in each update, it can be seen as a geometric series and a final value can be calculated:

When the grids are requested from outside the module they are all converted back to regular probability values using an inverted log-odds function. Because of the continuous updating they first need

3.4 Steering Module

The steering module is the last module in the flow, which sends steering signals back to the autonomous vehicle. It was designed to have two different states; one for breaking (full stop) and one for steering. The states are switched between using a pre-defined condition of some kind. Specifically the condition is based on whether or not it is safe for the vehicle to keep driving. The module was designed to allow for the condition to be changed and still manage the rest of its functions.

3.4.1 Brake State

For the current implementation of the module, the condition was set to be based on a small square of cells in front of the IAP. In other words, if an occupancy grid had

multiple *occupied* cells in in close proximity of the vehicle, the module entered the brake state. The method to detect if there is many occupied cells is by finding the average occupancy and checking if it is above a threshold. The average occupancy is called ”danger level” and the threshold is a measure of how much danger level is tolerable. The function which calculates the danger level, was conveniently called a danger level function. As previously mentioned any condition and threshold can be used by designing new danger functions.

A danger function can only utilize the occupancy grids to determine a danger level, but can define any threshold or shape of cells to find an average. It can even discard an occupancy grid entirely, meaning the object type used to create the grid is irrelevant when it comes to collisions. For this project the two relevant labels were ”car” and ”potted plant”. The danger level function handles the car label as described above, with a relatively low threshold to make sure the IAP never collides with other cars. The potted plant on the other hand was considered safe to drive over, and therefore returned a danger level of 0 at all times. In other words the autonomous platform will stop for a car but drive over a potted plant.

3.4.2 Path Follow State

If the module is not in the brake state it defaults back to the path following state. In this mode the vehicle uses a heading function instead of a danger level function. The heading is a floating point value between -1 and 1, which represents turn rate balance between the wheels. If the function returns -1 the car only turns the wheels on the left side, 0 is equal on both and 1 means only the right side spins. Similar to the danger level function, the heading function can be implemented in many ways.

For this project an A* path path finding algorithm was used to create a heading. By taking all occupancy grids and collapsing them into a single merged grid a path can be found between all obstacles. This was done for simplicity, and works under the assumption that all objects should be avoided equally. The merged grid is a more traditional occupancy grid, with binary values where all objects are set to *occupied*. This provides the path finding algorithm with a clear graph structure where occupied cells are closed paths.

The algorithm produces a path in a specific compass direction, if an object appears the path will turn. The angle to the next node in the path is used to create the heading. Since there is no notion of a global position in the map, the vehicle can not get back on its original path. It can only use the heading to avoid vehicles, and then keep moving in the pre-defined compass direction.

4
Results

In this chapter the results are presented. The results of each module is shown separately to give a clear view of what they produce and how they work.

The summarized result of the project is a functional collision avoidance system. The system consists of modules which can run as a distributed program. The system has been tested on Linux and Windows operating systems and was written entirely in Python. All performance results are produced by using a high-end graphics card for image recognition.

4.1 Computer Vision Module

The final version of the computer vision module works by subscribing to an image stream. It runs each image through the image recognition network and publishes detections to any subscribers. Subscription is done to the ip-adress of the machine running the computer vision module and a port which is set in a configuration file.

4.1.1 Format

The detections are formatted into a list of dictionaries and sent using JSON formatting, an example structure can be seen in listing 4.1. Each dictionary has 4 key-value pairs, the keys are: *label, confidence, bottomright* and *topleft*. The value belonging to *label* is a string which contains the class of the detection. The *confidence* value is a float value between 0 and 1 which measures the quality of the detection. The last two values, belonging to *bottomright* and *topleft*, are dictionaries. Both dictionaries contain pixel coordinates with keys x and y, which together define a bounding box around the detected object in the image.

Listing 4.1: Example detection data package in JSON format

```
[
    {
        "label" : "person",
        "confidence" : 0.521,
        "bottomright" : {"x": 100, "y": 100},
        "topleft" : {"x": 0, "y": 0}
    },
    {
        "label" : "pottedplant",
        "confidence" : 0.376,
        "bottomright" : {"x": 150, "y": 150},
        "topleft" : {"x": 110, "y": 130}
    },
    ...
]
```

4.1.2 Performance

On average the detection process runs $\approx$ 10 per second, which is dependent on the source data stream. Processing a single image through the Yolo-network takes approximately 0.017 seconds. The time it takes from an image being received in the image recognition module to being received on a subscribing client is on average 1.1 seconds. The majority of the delay caused by the module is therefore due to network delay. The frequency is bottle necked by the camera itself. The above is true for the default image resolution which is 640×480 pixels.

4.1.3 Detection Image Examples

Lastly some illustrations of how the module works can be see in figures 4.1, 4.2 below, and A.1 in appendix. The figures show the bounding boxes around detected objects and in the top left corner of each box is the label. The confidence level is not shown in the figure, but all detections made have a confidence level above 0.3.

4.2 Occupancy Grid Module

The final version of the occupancy grid module continuously updates a dictionary of occupancy grids. Each key in the dictionary is a *label* which corresponds to a detection. The keys are pre-configured to a sub-set of all the possible detections (the label filter):

- Person
- Car
- Potted plant

In addition a key is added called *default* which holds the occupancy grid of all lidar data which does not match any camera detections.

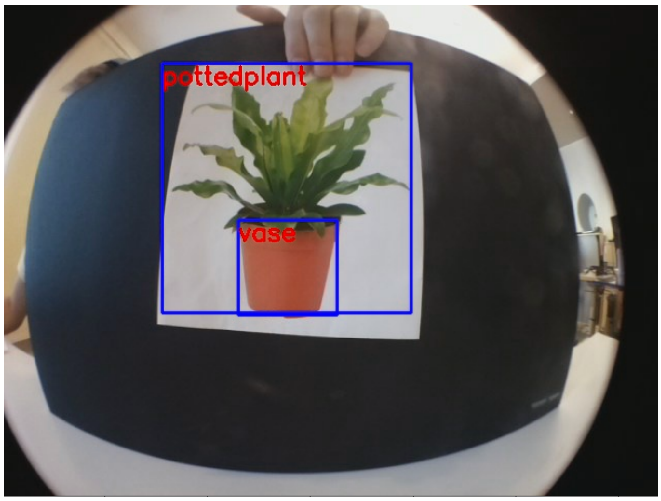


Figure 4.1: Example detection made by the computer vision module. A printed image of a potted plant is held in front of the camera. The module detects a potted plant (large box) and a vase (smaller box).

Currently the only supported input data streams are lidar and camera detections. The module does not publish any data but instead has a function for getting the latest occupancy grid dictionary. This means the occupancy grid module must be imported into other modules which require the occupancy grid dictionary.

Each occupancy grid is a two-dimensional numpy array, containing float values from 0 to 1. The size of the grid is set in a configuration file, currently defaulting to 80×80. An example of two grids can be seen in a visualization of the *default* in figure 4.3b and *person* in figure 4.3a.

4.3 Steering Module

The steering module imports the occupancy module and communicates directly with the IAP. By sending a speed update of zero the IAP can stop entirely, which happens when the module enters the brake state. While not in the brake state a speed update is set to a pre-configured max speed, in addition to the heading signal which makes the IAP turn.

4.3.1 Demonstration

The system was demonstrated at a Infotiv quarterly meeting. Each module was shown separately before a combined system demo was given live.

For the demo the path following state was deliberately turned off, to only show the stop functionality. The IAP was set on a table and started with a moderate forward speed. An image of a person was held at the edge of the table, when the autonomous platform was appr. 40 cm away from the image it stopped. The state changes were shown on a screen to visualize the changes made to the vehicles speed.

The path following functionality was not demonstrated due to time constraints, but was tested with a box and an image of a person. The vehicle managed to drive

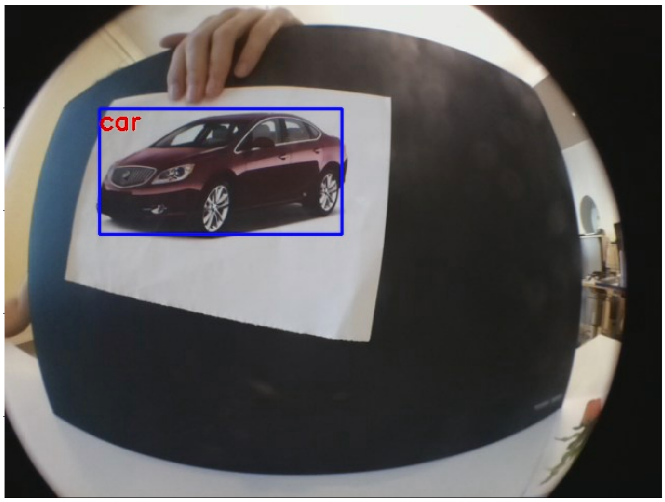
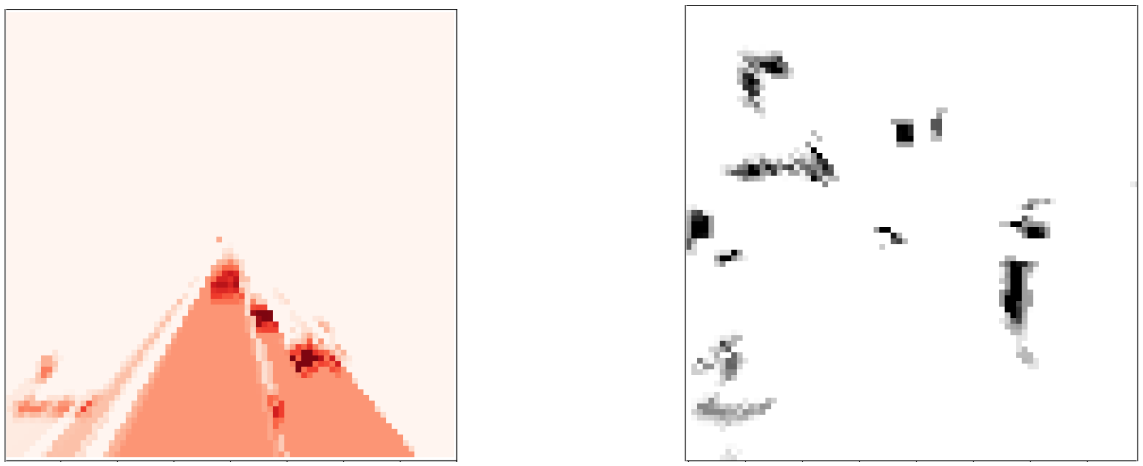


Figure 4.2: Example detection made by the computer vision module. A printed image of a car is held in front of the camera. It correctly identifies the image of a car.



(a) *Person*-occupancy grid.

(b) *Default*-occupancy grid.

Figure 4.3: Snapshot visualization of two occupancy grids. The left occupancy grid showing *person* detections by camera, superimposed with lidar data. Dark red pixels show the overlap between the camera detections and lidar. The conal, slightly lighter, shapes are the pure camera detections. The dark red pixel clusters are people. The left occupancy grid showing *default* detections made by lidar, shown as dark pixels. By definition there are no camera detections. It is difficult to distinguish objects, but the top left cluster of pixels is a corner and most straight lines are walls of the Infotiv office.

straight toward the box until the turn rate increased and it passed the box. After avoiding the box it continued forward as intended.

5
Discussion

In this final chapter the results are discussed, in reference to the initial problem statement. Different aspects of improvement or extension will be discussed as well.

The goal was to create a collision avoidance system which could recognize different types of objects and the distance to them. There are no large deviations from the original plan nor were there any major hindrances to the project. However there are plenty of things that can be improved or expanded upon in each module. The discussion will be divided into separate parts for each module and finally a general part for the system as a whole. I believe the objectives of the project have been met, which is supported by the results shown in the previous chapter.

5.1 Computer Vision Module

The module is built to be open for any application, only being coupled to the video source. Some simple examples could be system that count different type of objects encountered, or face recognition for people detected. The module is versatile since the source can be chosen freely and allows for any number of subscribers.

The module has an acceptable performance for real-time applications, however it is easily bottle-necked by bad wifi connections. The processing time is heavily dependent on the hardware used, a high end graphics card is required to reach maximum performance. Using a Nvidia GTX 1080 the processing takes mere milliseconds, which is the reason delays are more likely to be caused by the network communication.

There are plenty of ways to improve the module, and possibly increase performance. For example the Yolo network uses pre-trained weights, which are very accurate, but there are versions that are faster. The trade-off between speed and accuracy should not be hard to balance for each application, but may require custom training and network configuration.

Another area of improvement is encryption, in a real life situation communication which controls vehicles must be secured in some way. However since it is out of the scope of this project it was not even researched.

5.2 Occupancy Grid Module

The occupancy grid module is the key to fusing different sensor readings into one data structure. It manages to fulfill the objectives of the project with only lidar

and a camera, but could benefit from adding other sensors. To add a new sensor an inverse sensor model is required and adding it to the fusion process. There is potential for improvement in the automation of fusing distance sensors with camera data.

The module is the only one that does not publish an output onto a socket. It should not be difficult to add this functionality, however it is questionable if it is efficient to send such a large data structure using network communication. The system already has problems with network delays.

The choice to use a label filter to reduce the number of grids which need to be updated is questionable. Some type of solution is required, keeping track of and updating too many grids is computationally difficult for most computers. Since each labeled object has its own grid, to not lose information, the number of grids increases rapidly if many objects need to be tracked. For the objectives of this project only a few objects were required to be tracked, which made the label filter into a simple solution.

There should be smarter and faster ways to handle all the information, a simple example to explore could be using a single grid but having each cell store more information. In some ways it is equivalent to having multiple grids, but could prove simpler programmatically. Each cell would not have to contain likelihoods for every type of object, and could decay individually.

To allow dynamic objects to be seen the decay rate at which old measurements are forgotten is quite high. Since a low decay rate would mean that static objects would dwarf the probability of moving objects which do not have time to ”stack” over time. Moving objects also pose a difficulty for the system simply due to frequency. The sampling rate is not fast enough to smoothly track objects, but this issue may stem from the sensors themselves. However delays in the system will only add on to the issue. There is also a method involving predicting movement based on the cells, but it is costly and was not explored in this project. It could potentially help enhance the tracking of rapid moving objects.

5.3 Steering Module

The steering module works for its simple purpose, avoiding collisions by turning or stopping. In a larger scope some type of global mapping (like SLAM) would allow the vehicle to drive between two points, and only using the collision avoidance system as an emergency interruption. In this case there would not be a path following state. The A* path following algorithm would not be required at all in such a scenario. Overall the module is a simple application of the occupancy grid module, but there are more interesting ways to utilize the environment state information.

5.4 Viability

An important part of the system is the ability to react quickly, successfully stop and avoid collisions. In the results chapter some performance metrics are presented, and it is found that a delay of approximately a second exists in the computer vision

module. It was difficult to measure delays in other modules, but they could be seen while running real time. There is an issue with the delayed communication, it takes roughly a second to stop when testing the system with images of objects. In a real world application a second is far too long to adequately react to objects in the road while driving at higher speeds. If a vehicle drives 70 km per hour, a single second of delay equates to 19 meters travelled before the system reacts.

An argument can be made that while driving at higher speeds most objects pose a threat to the vehicle and the object, meaning there is no real need for a ”smart” collision avoidance system. Instead the system shines in city environments, where there are far more people and other objects to keep track of, and consequently vehicle speeds are lower. A delay of 1 second is still far too high to be considered safe even at speeds like 20 km per hour but is closer to viable.

If the whole system is run on a single machine, removing the networked communication, the system should have less delay. However this puts more pressure on the machine running the system, especially considering the current image recognition module requires a graphics card to run quickly. There are smaller and faster networks which could be explored in conjunction with running on a single machine. Also implementing the system in a language such as C++ should also speed up the systems across the board. Python offers quick prototyping possibilities but can come at the cost of computational speeds.

For the purpose of this project and the autonomous platform the system is viable. Since the system was built for the platform and at speeds which allow it to traverse an office, the latency is not unacceptable.

5.5 Other applications

From the very start a key word was modularity, therefore there are many potential uses for the system and the individual modules. The system can in short be described as a visual filter for the lidar sensor. Because objects can be given a high or low risk in the event of a collision. Lidar data can be discarded if it belongs to a low risk objects.

One interesting application of the occupancy grid is tracking objects. Clusters of high probability cells can be used to find the position of the different objects. If an object moves out of the cameras field of view it can no longer be identified as a specific type of object. However, by using predictive methods, clusters that have once been identified can be tracked and assigned their old label. Effectively making the system able to track and identify previously identified objects. This was in fact tested during the projects development, however a further use for it was not in the scope of the project.

5.6 Conclusion

The system works well as a proof of concept, further development should delve deeper into optimization and calibration. The key feature of the project is creating a platform of ”awareness” for the vehicle, which other systems can use to create

different types of reactions. Additionally, being able to classify lidar data by fusing it with detections. This allows filtering and other operations that are otherwise impossible on pure lidar data.

References

[1] R. Teetor, "Speed control device for resisting operation of the accelerator.", 2 519 859, Nov. 8, 1948.

[2] J. Markoff, "Google cars drive themselves, in traffic", *New York Times*, Sep. 10, 2010.

[3] E. Sacks, "Self-driving uber car involved in fatal accident in arizona", *NBC News*, Mar. 2018.

[4] W. D. Jones, "Self-driving cars: Saving lives and energy", *IEE Spectrum*, Jun. 2014.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[6] A. Elfes and L. Matthies, "Sensor integration for robot navigation: Combining sonar and stereo range data in a grid-based representataion", in *26th IEEE Conference on Decision and Control*, vol. 26, Dec. 1987, pp. 1802–1807. DOI: 10.1109/CDC.1987.272800.

[7] C. Gálvez del Postigo Fernández, "Grid-based multi-sensor fusion for on-road obstacle detection: Application to autonomous driving", Sep. 2016.

A

Appendix 1

A.1 Camera Detections

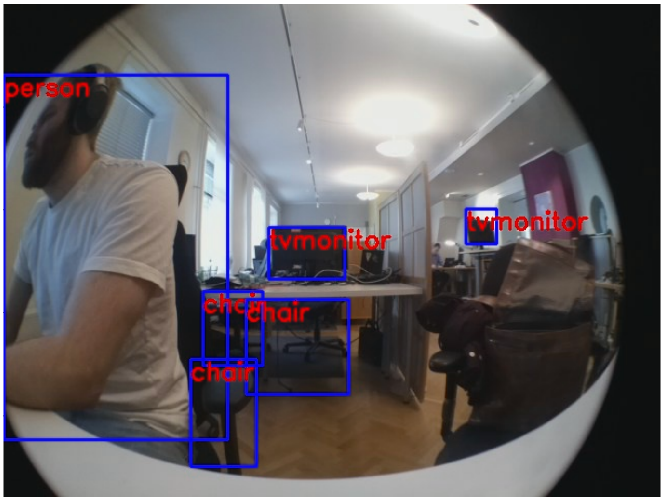


Figure A.1: Example detection made by the Computer Vision Module. A single frame image of the office with multiple detections..