FACULTY OF SCIENCE AND TECHNOLOGY

BACHELOR'S THESIS

Study programme / specialisation: Bachelor in Computer Science	The spring semester, 2023 Open
Author: Julie Vy Tran, Eirik Reiestad and Asbjørn Stokka	
Supervisor at UiS: Karl Skretting	
Thesis title: An Introduction to Computer Vision in Autonomous Driving Norsk tittel: En Introduksjon til Kunstig Syn i Autonom Kjøring	
Credits (ECTS): 20	
Keywords: Computer vision, Autonomous driving, Python, Image processing	Pages: 74 + appendix/other: 58 Stavanger, 15.05.2023

Abstract

Autonomous driving is one of the rising technology in today's society. Thus, a wide range of applications uses this technology for the benefits it yields. For instance, an autonomous driving robot will free up the labor force and increase productivity in industries that require rapid transportation. However, to gain these benefits, it requires the development of reliable and accurate software and algorithms to be implemented in these autonomous driving systems. As this field has been growing over the years, different companies have implemented this technology with great success. Thus, the increased focus on autonomous driving technology makes this a relevant topic to perform research on.

As developing an autonomous driving system is a demanding topic, this project focuses solely on how computer vision can be used in autonomous driving systems. First and foremost, a computer-vision based autonomous driving software is developed. The software is first implemented on a small premade book-size vehicle. This system is then used to test the software's functionality. Autonomous driving functions that perform satisfactorily on the small test vehicle are also tested on a larger vehicle to see if the software works for other systems. Furthermore, the developed software is limited to some autonomous driving actions. This includes actions such as stopping when a hindrance or a stop sign is detected, driving on a simple road, and parking. Although these are only a few autonomous driving actions, they are fundamental operations that can make the autonomous driving system already applicable to different use cases.

Different computer vision methods for object detection have been implemented for detecting different types of objects such as hindrances and signs to determine the vehicle's environment. The software also includes the usage of a line detection method for detecting road and parking lines that are used for centering and parking the vehicle. Moreover, a bird-view of the physical world is created from the camera output to be used as an environment map to plan the most optimal path in different scenarios. Finally, these implementations are combined to build the driving logic of the vehicle, making it able to perform the driving actions mentioned in the previous paragraph.

When utilizing the developed software for the driving task, hindrance detection, the result showed that although the actual hindrances were detected, there were scenarios where blockades were detected even though there were none. On the other hand, the developed function of stopping when a stop sign is detected was highly accurate and reliable as it performed as expected. With regard to the remaining two implemented actions, centering and parking the vehicle, the system struggled to achieve a promising result. Despite that, the physical validation tests without the use of a vehicle model showed positive outcomes although with minor deviation from the desired result. Overall, the software showed potential to be developed even further to be applicable in more demanding scenarios, however, the current issues must be addressed first.

Preface

First, we would like to thank our supervisor Karl Skretting for his advice and support throughout the project. Also a special thanks to the University of Stavanger for letting us use Lyspæren as an area for testing and a workspace with access to tools like 3D printers and laser cutters to be able to do modifications to the vehicle. Thus, making this project possible. Last but not least, we would like to thank our family and friends who have supported us throughout the project.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

In today's society, autonomous driving technology is widely used in many aspects. For instance, the most well-known usage of autonomous driving is in self-driving cars. This technology is also used in other applications for both private and industrial usage. For private application areas, there are for example robots such as lawn movers and vacuum cleaners that use this technology to carry out tasks that are simple and often need to be performed. On the other hand, for instance, in industries with logistics operations, automated driving has been used for internal logistics such as moving packages in a warehouse. One of the advantages of using autonomous driving for this use case is that it is possible to increase the employees' productivity by freeing up their time for more valuable tasks. Overall, there are a lot of opportunities where autonomous driving can be applied, both for private and industrial usage.

One of the core technologies of autonomous driving is computer vision. This method has been used in autonomous driving for tasks such as detecting and classifying the vehicle's surroundings. Thus, with the ability to identify their own surroundings, the vehicles can safely navigate around their environment. Despite that, computer vision used alone in autonomous driving has its challenges. Nevertheless, it is a fundamental start to the process of developing an autonomous driving system.

1.2 Aim and Objectives

The aim of this thesis is to develop computer vision-based software for autonomous driving and implement it on prototype vehicles as a proof of concept for further development of computer vision in the field of autonomous driving. The core objectives of the thesis are the driving actions the autonomous driving system should be able to deal with without human interactions such as:

- Detecting hindrances within an appropriate distance in front of the vehicle and reacting to this information by stopping the vehicle to avoid front collisions.

- Detecting stop signs and stopping the vehicle when the sign is within an appropriate distance, e.g., to indicate in an intersection that other autonomous vehicles are nearby to avoid collisions.
- Centering the vehicle on a simple road indicated by two-lane lines for autonomous vehicles to, for instance, move around in marked driving tracks.
- Parking in a desired spot that is indicated by two parking lines and a QR code that can be used for autonomous vehicles to, e.g., drive to a charging spot, parking spot, or destination point for moving products in a warehouse.

1.3 Limitations

The automated driving functions that will be developed are limited to the functions listed in Section 1.2. In addition to these limitations, the proposed solution takes into consideration other limitations such as:

- The vehicles that will be used to implement the software are simple prototype vehicles.
- The vehicles will be tested in a controlled environment, thus the obtained result may not translate to real-world scenarios.
- The vehicles will only be driving on a simple road that does not include sharp turns or intersections.
- The vehicles will only perform perpendicular parking.
- The vehicles will only perform parking in scenarios where only one QR code is visible in the vehicle's field of view.
- The vehicle will only be driving in flat environments.

Overall, by following the objectives and limitations of this project, a simple yet operational autonomous driving system can be achieved.

1.4 Methodology

The project is executed with the following methodology to achieve the objectives of the project.

Data Collection

The test vehicle will use a camera for all the objectives as its primary data collection method except for hindrance detection which uses two cameras. The cameras will in both cases be restricted to a front-facing view.

Data Analysis

Before the information can be extracted from camera frames, they will go through a calibration process to remove lens distortions. Thereafter, in some scenarios, the relevant information will be retrieved from the calibrated camera frame directly. In other cases, additional image processing methods will be performed before the relevant information is perceptible.

Hardware Design

Although the vehicle's hardware design has been developed prior to this project, minor modifications on the hardware needed to be done to improve the accuracy of the vehicle's motion control. Different computer-aided design (CAD) software like Fusion 360 and kiCad was used to design additional components for the vehicles and electronics. These modifications were necessary to customize the vehicles for the project's specific applications.

Software Development

Working as a software development team, the project team will utilize a LEAN methodology by continuously following the cycle of:

- Identifying the needed concept.
- Discussing and planning what would give the most value to the project.
- Implementing the most valuable feature at this time.
- Run tests to get feedback on the result.

This is to optimize development time and other resources [52]. In addition, Github is used as a version control and project collaboration tool to have a structured way of collaborating.

Scalability

The software is aimed to be modular and scalable. First and foremost, it should be easy to implement new autonomous actions for the software without any modifications to existing implementations. Furthermore, the software should be able to be utilized for different vehicle constructions and motion control systems with a minimal amount of modifications of the software.

Implementation and Testing

The software implementation will be divided into modules. This makes the software scalable by making it easier to implement additional autonomous functions as new modules. The modular

design also makes it easier to create a set of unit tests for each module to test the core functionality. This will help to locate errors early, as well as secure functionality when refactoring codes.

Integration tests will be performed as it is an important testing method to see if the software works as intended when multiple modules are combined. There will also be conducted experimental tests for the different autonomous driving functions and the test results will be used to analyze and determine essential improvements in the system for future works. The experimental tests will be first conducted on a small test vehicle. Afterward, the functions that perform sufficiently will be tested on a larger test vehicle to verify that they can operate for other vehicle models as well.

Evaluation Criteria

The results of this project's experimental tests will be assessed against some metrics to determine their accuracy and reliability, making it easier to conclude if the result of the project is sufficient and acceptable.

Accuracy

Through the experimental tests, the accuracy of the modules will be tested. The accuracy will be measured with the criteria that the calculated values should only have a maximum of 10% deviation from the desired metric information. This number is selected for several reasons:

- For hindrance detection, the accuracy of calculating the distance to the object is not critical, as long as the interval of distances the vehicle should stop when the object is within this interval is large enough. However, it is still desirable to approximately know what interval of distances the vehicle will stop when an obstacle is detected, e.g., if the vehicle is instructed to stop when an object is observed within 50cm distance from the vehicle. Then, by using a 10% deviation, it will not affect the result significantly as the deviation is minimal compared to a higher deviation value. Thus, the result will be consistent. The same reasoning is valid for stop sign detection.
- For the other objectives such as lane centering and parking, a high deviation value is more critical as the desired result is more complex. A high deviation value can for example cause the vehicle to not center itself in the lane, and in the worst-case scenario move out of the lane. Whereas for parking, a high deviation value can cause the vehicle to park outside of the desired parking slot, which is also not an ideal outcome. Thus, a maximum deviation of 10% will ensure that the function of centering and parking the vehicle will have high accuracy.

Although a maximum deviation of 10% for each module is set as a criterion, it is important to be alerted that when combining the different modules together, a higher deviation value can emerge.

Reliability

The reliability of the autonomous driving system including the usage of the driving logic and the computer vision modules is also tested through experimental tests. The system must be reliable by being predictable and functioning as planned. Therefore, for detecting hindrances and stop signs, the vehicle should stop a minimum of 90% of the time. In terms of driving between lanes, it should not deviate more than 10% from the center. Furthermore, it should be able to park at a specific spot 90% of the time at different start positions. By defining such a high success rate and low deviation rate, it is possible to be certain that the vehicle will be consistent.

1.5 Thesis Outline

The rest of the thesis is structured as follows:

Chapter 2 - Computer Vision and Algorithms: This chapter gives an introduction and overview of computer vision, the theoretical basis of computer vision algorithms, and other algorithms needed for the software.

Chapter 3 - System Design and Development: This chapter presents the system design in both hardware and software aspects, the use cases of the algorithms given in Chapter 2 in the software, the motion control of the vehicle, and tests that have been performed to determine and evaluate the result of the project.

Chapter 4 - Results and Analysis: This chapter presents the result of the tests that were introduced in Chapter 3 and analyzes the test result of each test to evaluate the performance of the solution.

Chapter 5 - Conclusion and Future Works: This chapter summarizes the findings of the project and deduces a conclusion based on whether the objectives have been achieved. It also proposes future works based on the project's result.

Chapter 2

Computer Vision and Algorithms

This section will first present an introduction and overview of computer vision. Thereafter, the algorithms and techniques that will be used to develop the software for the autonomous driving system, including computer vision and path planning algorithms are presented.

2.1 Overview of Computer Vision

Computer vision is a subfield of computer science that aims to extract useful information from images [46]. Through computer vision, computers can analyze and interpret the world visually, enabling them to determine their environments and take appropriate actions based on various situations. Examples of computer vision techniques include image segmentation, object detection, facial recognition, edge detection, pattern detection, image classification, and feature matching [16]. These techniques are used to extract the desired information from the images in different scenarios.

Computer vision is used in various fields. In the context of autonomous vehicles, the subfield is critical for detecting and classifying objects within the vehicle's field of view. However, computer vision is not limited to the automotive industry. It has also found applications in the medical field where it has been used for X-ray testing [24] and detecting different types of cancer. For instance, Howard Lee and Yi-Ping Phoebe Chen's study, "Image-based computer-aided diagnosis system for cancer detection," demonstrates how computer vision can improve the accuracy of cancer diagnosis [15].

Computer vision is also being integrated into the retail industry. One example is Amazon Go, a chain of self-checkout stores that rely on computer vision, sensor fusion, and deep learning technologies to track the products customers purchase [44]. As such, computer vision has the potential to revolutionize not only transportation but also various industries and applications.

2.2 Stereoscopic Vision

Stereoscopic vision is the brain's ability to perceive three-dimensional shapes and forms from visual inputs [41]. By using this method on vehicles, it allows the vehicles to also be able to detect three-dimensional shapes. Thus, it can accurately measure the location of objects and obstacles in the physical world, which is essential for making informed decisions regarding the best path to navigate around them [4]. In computer science, this process is referred to as stereo vision when a computer does these calculations.

Although the method has the capacity of performing the previously mentioned action, for this project a simpler use case of stereoscopic vision will take place in Section 3.2.2. The goal of using stereoscopic vision in this project is to only detect objects that get too close in front of the vehicle, not measuring the distance of the objects. This approach is sufficient enough to showcase that this method can prevent collisions and other accidents which results in improving the safety of autonomous vehicles.

Even though stereoscopic vision is ideal for object detection, it has limitations in real-life scenarios. For instance, objects must have texturally rich features and be of a different color from the surface background [57], making them unsuitable for some depth scenarios. Therefore, other methods may be necessary for achieving more accurate results for specific tasks like calculating the distance to lines.

Concept

The concept of stereoscopic vision is akin to human visual perception, where we rely on the simultaneous input of two eyes to determine depth and distance to objects. In the case of stereoscopic vision, two cameras are used instead of eyes, and a triangle is created between the object and the cameras to calculate the object's distance. This process is visualized in Figure 2.1.

In the figure, point P of an object is detected in both image frames given by the two cameras. The distance between the cameras is the baseline, which can be adjusted for improved accuracy or view by reducing or increasing the baseline. For each image frame, the object with the point P is spanned over an area in the frame. Thus in this area, there exist numerous pixels that correspond to positions on object P which the other image frame can also spot although at a different pixel position. These pixels are then matched to their corresponding pixels in the other image frame. To expedite the computational process, only the pixel at the epipolar line is checked which is the section where the corresponding pixel is likely to be located in the image. This line represents the intersection of the epipolar plane with the image plane, where the epipolar plane is the plane defined by object P and the epipoles [1] [1].

[1] An epipole is where the center of one camera shows up in the picture of another camera.

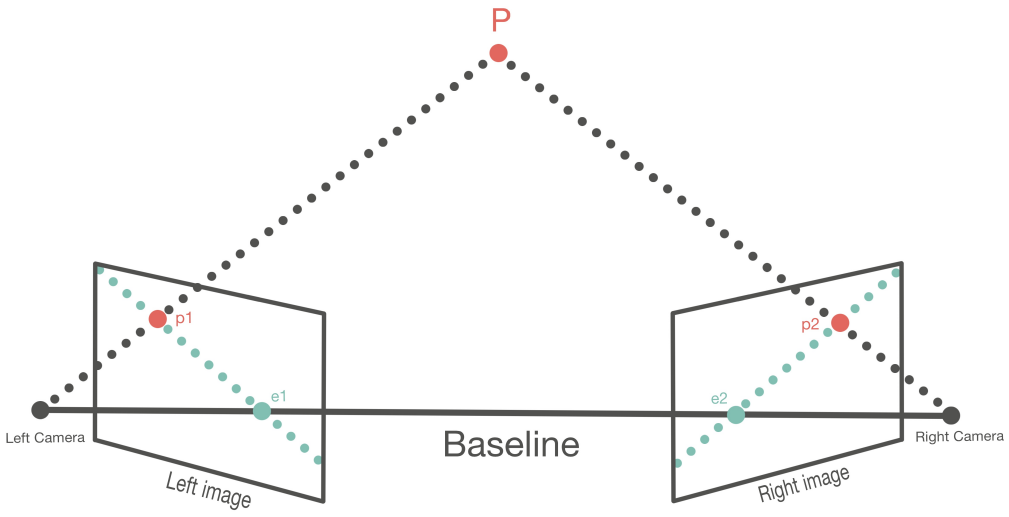


Figure 2.1: P represents a point on the desired object for detection. p1 and p2 represent the point P in the left and right camera images. The baseline is the distance between the two cameras. The green line in the image indicates the epipolar line, which is where the epipolar plane intersects with the image plane defined by point P and the epipoles.

Once the corresponding points on the images have been identified, the positional difference between the corresponding points is used to calculate the distance to the object. In short, objects closer to the cameras give a greater positional difference in camera images. This positional difference can then be used to generate a disparity map[2]. This map is thereafter used to create a depth map[3] that can be used to determine the distance to the object. More details about the process of creating the different maps will be presented in Section 3.2.2.

2.3 Local Binary Pattern Cascade

The ability to detect signs is crucial for autonomous driving systems because different signs can be used to inform what actions the system need to take. In order to detect these signs, the computer vision method called object detection can be used. For autonomous driving system use cases, high computational speed and accuracy are needed in object detection to achieve real-time performance with it such that systems can react as soon as possible. In this project, as one of the objectives is to detect stop signs and react to them, the following method of object detection presented in this section has been used in Section 3.2.4 to detect and locate the stop signs.

There exist many approaches to achieve object detection, for instance, Convolutional Neural Network is a widely used approach. However, Local Binary Pattern Cascade is selected to be used in this project because of its simplicity in creating the model with less need for GPU resources and time-consuming training compared to the previously mentioned approach [3]. A consequence is that the result of using LBP is slower detecting and less accurate than the CNN methods [23]. However, the decline in speed and accuracy is tolerable in this project because

[2]The disparity is the distance between a pixel and its corresponding horizontal match in the other image [40].
[3]A depth map is an image reflecting the depth of the objects in the frame.

the tested environments are controlled.

By using LBP Cascade, it is possible to provide the method with a camera frame and in return, obtain the location of the signs if there exist any in the camera frame. The locations obtained can thereafter be used to calculate the distance from the vehicle to the different signs if the sizes of the signs are known. Thus, the vehicle can then react to stop signs if the signs are within a certain distance.

The original version of LBP is based on calculating the 3-by-3 neighborhood threshold of each pixel based on the center pixel of the 3-by-3 [6]. Table 2.1 shows the LBP calculation for a 3-by-3 neighborhood. Table 2.1a is the original 3-by-3 neighborhood while Table 2.1b is the threshold calculated by comparing each element in the neighborhood to the center. If it is larger or equal to the center, then the threshold will be one, otherwise zero. Table 2.1c is the weighted binary value for each cell which is multiplied by the corresponding value in Table 2.1b. The result of the multiplications is shown in Table 2.1d. Summing up the cell values in Table 2.1d gives the LBP value of the center cell. In the following example the sum is equal to 209.

6	5	2
7	6	1
9	3	7

(a) Original

1	0	0
1		0
1	0	1

(b) Threshold

1	2	4
128		8
64	32	16

(c) Weight

1	0	0
128		0
64	0	16

(d) Result

Table 2.1: An example of an LBP calculation.

For each 3-by-3 neighborhood of the image, an LBP value is calculated with the method above. Therefore, a histogram can be made out of all these LBP values which counts how many times the same LBP value is calculated. By comparing the different histograms, it is possible to detect the different types of features such as edges or corners, etc [45].

After calculating all the features, the next step is to classify which feature is relevant as most of the features that have been calculated are irrelevant for detecting the desired object. A training method called AdaBoost that can be used to classify which feature is relevant is therefore utilized. The training method uses positive and negative images which are respectively images with the desired object and images without the desired object to detect the applicable features. Consequently, all the relevant features detected by AdaBoost combined together will be a classifier model that can be used to detect the desired object [28].

The last step in this algorithm is the concept of Cascade of Classifiers. Instead of applying all the relevant features to every sub-part of the image, the features are divided into different stages for classifying. For example, if the first stage's features do not pass on a region of an image, then the next planned stages on that region are aborted. By aborting, the method saves time in applying the remaining features as it already knows that it is not the desired object. Those regions which have all features passed are the desired object and thus the location of the target has been obtained [28].

2.4 Line Detection with Hough Transform

In this project, two of the objectives concern adjusting the autonomous vehicle to physically indicated lines: centering the vehicle within two-lane lines and parking the vehicle at a spot enclosed by two parking lines. Line detection is therefore necessary. By performing line detection on an image frame, the method will return all the visible lines in the frame which can then be used for further use cases. The newly mentioned method will therefore be used in Section 3.2.6 and Section 3.2.7 as one of the steps to determine lane lines and parking lines.

An approach to achieve line detection is to utilize two image processing methods, edge detection and Hough transform [32]. Besides these two methods, there exist image processing methods that are prerequisites for edge detection and also image processing methods that are optional to improve the performance of the two main techniques.

For edge detection, there are multiple algorithms for the same purpose. For instance, the Canny edge detection method is favorable for edge detection in noise conditions [51]. Thus, as camera frames usually includes noise, the Canny edge detection is selected to be used. The algorithm is divided into five steps: Noise reduction, gradient calculation, non-maximum suppression, double threshold, and edge tracking by Hysterisis. More information about the different steps can be found at [27] as this is out of the scope of the thesis. However, it is important to note that the first step Noise reduction uses Gaussian Blur to reduce noise on the image and that the Hysterisis step consists of two threshold values, *minVal* and *maxVal*. The edges that have intensity gradient values above *maxVal* are considered real edges. The edges that are below *minVal* are discarded while those that are between these two thresholds are considered edges too if they are connected to the edges that are above *maxVal* [27].

There are multiple steps that need to take place before applying the edge detection method to the image. First, the image should be converted to grayscale as this is a prerequisite for edge detection [2]. After performing the edge detection, the output of the edge detection step should go through an extra image processing method called morphological closing. This is a morphological transformation method used to fill elements without affecting the shape and size of the elements [34]. The method is based on performing two other morphological transformation methods. Dilation is performed first in the process and is an image-processing method used to grow the edges of an image. Afterward, erosion is carried out to discard pixels near the edges [34]. By applying morphological closing to the Canny output, the lines given from the Canny edge detection method will be more visible. Thus, it will improve the result of the next step.

After performing edge detection, the next step is to perform the Hough transformation. This is a transformation technique that can be used to detect straight lines. In short, the method is based on creating a parameter space using either the Cartesian or polar coordinate system and counting how many times an intersection point is intersected. Thereafter, a threshold is defined and it holds the value of how many times an intersection point needs to be intersected for it to be determined as a point. More information about how the transformation works can be found in [32].

Finally, although the algorithm returns all visible lines in the image frame, a drawback of this algorithm is that in cases where the physical lines are wide, the algorithm will return multiple lines which together actually represent one physical line. The algorithm, therefore, needs additional merging or classifying algorithms to resolve this problem.

2.5 Classifying and Merge Lines in Lane Detection

As mentioned in Section 2.4, the line detection method has the drawback of returning multiple lines that present the same physical line when the physical lines are wide. The following algorithm will therefore be used in Section 3.2.6 to resolve this drawback. By using this algorithm, the set of lines returned by the line detection algorithm given in Section 2.4 will be reduced to two lines that correspond to the lane lines.

The solution is proposed by Kanagaraj, et. al [10] and is based on using the value of the slope to determine one single line for each lane line from a set of lines. By defining a slope value range for each side of the lane, it is possible to classify the lines into these two groups. Thereafter, the average line of each group is calculated and used as the representative of the lane lines. The algorithm, however, it is notable that the method is susceptible to noise lines[4] if the slope range is too wide, which can cause the result lines from this algorithm to deviate from the physical lines.

2.6 Perspective Transform

An approach for centering the vehicle within a road is to create a point in the center at further parts of the path and use this point as the destination for the vehicle. However, when two-lane lines have been detected, depending on the vehicle's position, the two-lane lines might be of different sizes. This makes it difficult to determine the center of the road at further places as the camera image is in a three-dimensional view. The perspective transformation method which can be used to change the perspective of an image or a part of an image to a desired perspective will be used in Section 3.2.6 to resolve this problem and determine this point.

The transformation method is based on creating a transformation matrix and using this matrix to convert points to points in the desired perspective. The details of the linear algebra part of creating the transformation matrix and the derived equations for usage will not be presented as it is beyond the thesis' scope but can be read about in Chapter 2.2 from [18]. However, it is essential to mention that the matrix needs eight points to be created: four points defining the source perspective and four points defining the destination perspective. Furthermore, Equation 2.1 can be used to transform a single point from the source perspective to a destination perspective [31].

$$(x', y') = \left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right) \tag{2.1}$$

Equation Description

- **(x', y'):** the transformed point
- **M:** transformation matrix
- **(x, y):** the source point that needs to be transformed

[4] Noise lines are the same as irrelevant lines in the image frame.

2.7 Merging Line Segments

In Section 3.2.6, the proposed method of classifying and merging lines given in Section 2.5 was presented to resolve the drawback of the line detection method given in Section 2.4. However, the proposed solution only takes into account cases where there are only two physical lines with different slope signs. On the other hand, in the use case of parking, there might exist more than two physical parking lines. In addition, the two parking lines which together indicate one parking spot might have the same slope sign depending on the vehicle's orientation to the lines. The following approach of merging two close lines presented by Tavares and Padilha [55] will therefore be implemented in Section 3.2.7 to resolve the drawback from the line detection algorithm.

The method is based on comparing two lines with each other and returns a merged line if the method's criteria are met. It starts with defining a centroid x_G, y_G which the merged line will go through, and the orientation θ_r of the merged line by using Equations 2.2 and 2.3, respectively. The two lines that are being evaluated for merging are defined by two points each. a and b are defined as the points for the first line and c and d are defined as the points for second line. The length and orientation of the first line are defined as l_i and θ_i. Similarly, the length and orientation of the second line are defined as l_j and θ_j.

$$\begin{aligned} x_G &= \frac{l_i(a_x + b_x) + l_j(c_x + d_x)}{2(l_i + l_j)} \\ y_G &= \frac{l_i(a_y + b_y) + l_j(c_y + d_y)}{2(l_i + l_j)} \end{aligned} \tag{2.2}$$

$$\theta_r = \begin{cases} \frac{l_i\theta_i + l_j\theta_j}{l_i + l_j}, & \text{if } |\theta_i - \theta_j| \leq \pi/2 \\ \frac{l_i\theta_i + l_j\left(\theta_j - \pi\frac{\theta_j}{|\theta_j|}\right)}{l_i + l_j}, & \text{otherwise} \end{cases} \tag{2.3}$$

It is now possible to define a new coordinate system G where the centroid is the origin and the orientation calculated earlier is the orientation for the x-axis of the new coordinate system. The four given points representing the two lines can be converted to this new coordinate system by using the equations from Equation 2.4.

$$\begin{aligned} \delta_{XG} &= (\delta_y - y_G)sin\theta_r + (\delta_x - x_G)cos\theta_r \\ \delta_{YG} &= (\delta_y - y_G)cos\theta_r - (\delta_x - x_G)sin\theta_r \end{aligned} \tag{2.4}$$

An example of the setup, after the points have been converted to the new coordinate system G, is shown in Figure 2.2.

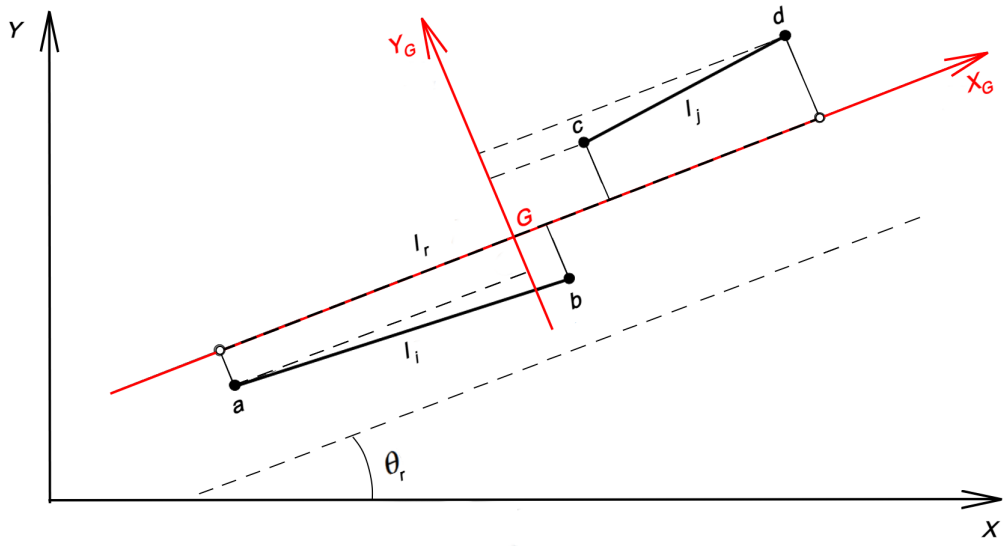
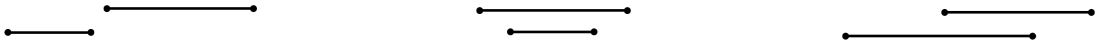


Figure 2.2: An example with the new coordinate system G represented by the red lines. The black dashed line on the X_G-axis is the merged line. The black lines define the two lines defined by respectively a and b, and c and d.

For the following calculations, the converted values of the four points will be used in the calculations, denoted with a G in the subscript. During the merging of two lines there exist three scenarios as presented in Figure 2.3.



(a) No overlap **(b)** Total overlap **(c)** Partial overlap

Figure 2.3: An example of the three different cases when comparing two lines.

The first case is when the two lines have no overlap which can be presented by condition 2.5. l_r is the length of the merged line where the merged line. The start and the end point of the merged line are defined by the two furthest orthogonal projections onto the new coordinate system's x-axis of the four points a, b, c, and d.

$$|l_r| \geq (|a_{X_G} - b_{X_G}| + |c_{X_G} - d_{X_G}|) \tag{2.5}$$

The two criteria of proximity for accepting the merging of the two lines are shown in Equation 2.6. $d_{MAX_{XG}}$ and $d_{MAX_{YG}}$ are two self-defined threshold to accept the merge. The higher these values are, the further apart the lines can be but still within the criteria for merging. δ_{YG} and κ_{Y_G} are the two furthest points from each other relative to the new coordinate system's y-axis.

$$\begin{gathered} |l_r| - (|a_{X_G} - b_{X_G}| + |c_{X_G} - d_{X_G}|) \leq d_{MAX_{XG}} \\ \text{and} \\ |\delta_{YG} - \kappa_{Y_G}| \leq d_{MAX_{YG}} \end{gathered} \tag{2.6}$$

The second case is when the two lines totally overlap each other. This case can be presented by the condition given in Equation 2.7. The criterion of proximity for accepting the merging of the two lines in this case is the second criterion in Equation 2.6.

$$|l_r| = |a_{X_G} - b_{X_G}| \quad \text{or} \quad |l_r| = |c_{X_G} - d_{X_G}| \tag{2.7}$$

The third case is when the two lines only partially overlap each other. This case can be presented by the condition in Equation 2.8. Similar to case two, the criterion of proximity for accepting the merging of the two lines is the second criterion in Equation 2.6

$$|l_r| < (|a_{X_G} - b_{X_G}| + |c_{X_G} - d_{X_G}|) \tag{2.8}$$

Thus, if the two lines do not conform with the criteria in its case, then the lines will not be merged.

Although this method resolves the drawback of the line detection algorithm in the use case of parking, it does not propose a solution to classify the different lines but solely focuses on merging. This limitation is therefore the reason why this method has not been used in Section 3.2.6 to resolve the same drawback as the method will presumably return more than two lines because of noise lines. By using the method given in Section 2.5 instead, it is possible to classify and merge the lines with fewer and simpler calculations. Furthermore, the most fundamental equations and description of the algorithm have been included to show what the implementation in Section 3.2.7 has been based on. However, a more descriptive description of the method can be found at [55].

2.8 Bresenham's Line Algorithm

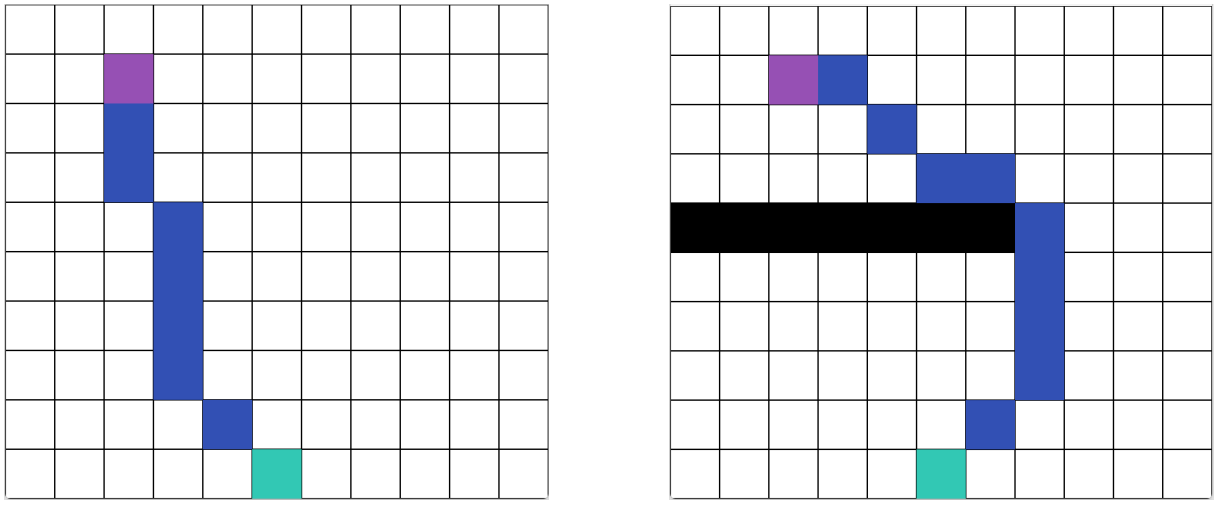
In Section 3.3, a two-dimensional matrix will be used to present the vehicle's environment. To map the lane and parking lines from Section 3.2.6 and Section 3.2.7 respectively into the two-dimensional matrix, the lines need to be converted into a list of integer points corresponding to the matrix indexing approach. Bresenham's line algorithm is therefore suitable for this use case as it can convert two endpoints of a line to a list of points that combined together represents the original line. Thus, this algorithm is used in Section 3.3 to map lines that are represented by two endpoints into the two-dimensional environment.

The algorithm is based on selecting each subsequent pixel of the line using integer calculations based on the least distance between the pixel and the actual line within the two endpoints [9]. Since further details about the algorithm are beyond the scope of this study, only a brief description of the algorithm has been presented here. More information on the algorithm can be found in [17].

2.9 A* algorithm

Considering the system will use a path-planning approach to help with decision making such as letting the system plan ahead in case no decisions can be made based on the information from the real-time moment [53]. A path-finding algorithm is therefore needed. There exists different graph-traversal and path-finding algorithms that are suited for this use case. For example, the Depth-first-search algorithm or Dijkstra's algorithm can be both used to find a path between a desired start and endpoint [53]. However, the A* algorithm is selected to be used in this project because of its simplicity and optimal efficiency [7]. A descriptive outline for this approach with modifications will take place in Section 3.4.

The A* algorithm is a modification to Dijkstra's algorithm and is optimized for a single destination [53]. Moreover, it is a shortest-path algorithm that creates a graph with an edge between the eight neighbors. In this project, a matrix is used, where each square represents a graph node. Figure 2.4 illustrates an example of an A* algorithm use case.



(a) Path without hindrances.

(b) Path with hindrances.

Figure 2.4: Result of using the A* algorithm to calculate a path between two points.

Matrix description

Green square is the start node.

Purple square is the end node.

Black square are the hindrances.

The calculation process of the A* algorithm is illustrated in Figure 2.5. The algorithm is based on calculating three values to decide the next node in the path. The first value denoted as g is the distance from the start to each specific node. This value is calculated by each node adding the distance to the parent with the parents' distance g. The second value denoted as h is calculated concerning the direct distance to the end position with no regard for obstacles, which is found with the use of the Pythagoras theorem. The third value denoted as f corresponds to the sum of the g and h values of each node and defines the total cost to travel to that node, which is used to find the cheapest path.

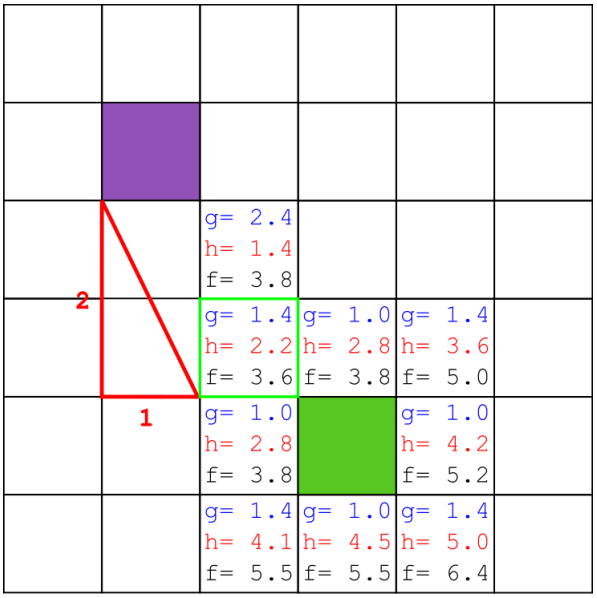


Figure 2.5: The values calculated for each node around the start node illustrate the calculation of the g value, h value, and f value. The red triangle showcases the distance from the green-outlined square to the purple square.

2.10 Catmull-Rom Spline

The path that is obtained by the A* algorithm presented in Section 2.9 consists of points corresponding to squares in the two-dimensional matrix environment map in Section 3.3. By moving through these squares, it will cause the vehicle's movement to be strained to eight different movements corresponding to each neighboring square. It is therefore preferred to move between the squares in a smoother way which is not too sharp and not too flat. In addition, as each square in the matrix is identified using indexes and will correspond to a real distance, a continuous path is needed for performing motion control on the vehicle. Thus, the Catmull-Rom spline is used in Section 3.4.2 to define a path with a smooth trajectory that is continuous and goes through the points given by the shortest path algorithm for path planning.

Splines, in general, are formed by several curve segments with each one being defined by a set of points. There exist several types of spline such as B-spline, Bezier spline, and Hermite spline, the last being the most suitable for this use case because this type of spline passes through all the points that have been used to calculate the spline. Although this characteristic is not necessary for environments without obstacles, however, in environments with obstacles, a path defined by a spline type without this characteristic could lead to collisions. Therefore, by using Hermite splines in path planning, the possibility of the spline function being deviated from the desired path is removed. However, a drawback with this type of spline is that generally when calculating the spline, the derivative is needed to be able to calculate it [11]. The Catmull-Rom which is a specialized version of the Cardinal spline which itself is a type of Hermite spline, on the other hand only needs four points to define a curve [42]. Thus, easier computations can be made. Furthermore, the Catmull-Rom spline also takes into consideration that there should not be too flat sections and not too sharp sections which are suitable for use cases like pathing.

As previously mentioned, the Catmull-Rom spline defines each curve segment using four points:

P_0, P_1, P_2, and P_3, as illustrated in Figure 2.6.

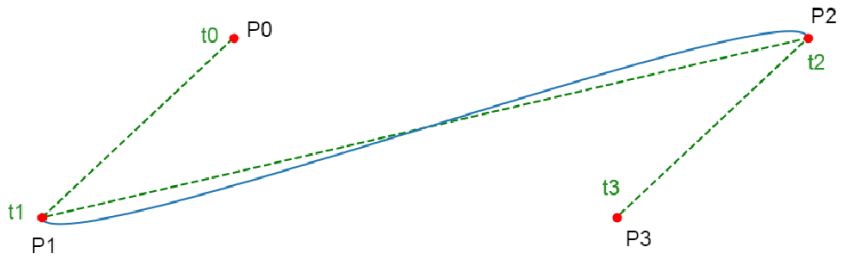


Figure 2.6: An example of a Catmull-Rom spline. The blue line is the curve made by the four points $P0$, $P1$, $P2$, and $P3$. The green lines is structured by intersecting linear lines through $t0$, $t1$, $t2$ and $t3$, which are parameterized points from Equation 2.10.

The curve constructed by the four points can be calculated using a set of equations as shown in Equation 2.9 [58]. The set of equations consists of three levels, denoted subsequently as A, B, and C in order. Level C represents the final curve. The calculation starts from the first level and is then iteratively inserted into the next level until the final curve C is obtained.

$$\begin{aligned}
C &= \frac{t_2 - t}{t_2 - t_1} B_1 + \frac{t - t_1}{t_2 - t_1} B_2 \\
B_1 &= \frac{t_2 - t}{t_2 - t_0} A_1 + \frac{t - t_0}{t_2 - t_0} A_2 \\
B_2 &= \frac{t_3 - t}{t_2 - t_0} A_2 + \frac{t - t_1}{t_3 - t_1} A_3 \\
A_1 &= \frac{t_1 - t}{t_1 - t_0} P_0 + \frac{t - t_0}{t_1 - t_0} P_1 \\
A_2 &= \frac{t_2 - t}{t_2 - t_1} P_1 + \frac{t - t_1}{t_2 - t_1} P_2 \\
A_3 &= \frac{t_3 - t}{t_3 - t_2} P_2 + \frac{t - t_2}{t_3 - t_2} P_3
\end{aligned} \tag{2.9}$$

Parameter t_0 from the equations above is equal to 0 while t_1, t_2 and t_3 are calculated by using Equation 2.10 [58]. The parameter α from Equation 2.10 is a value between 0 to 1 where the different values of α correspond to different types of Catmull-Rom splines. The value α set to 0.5 corresponds to the centripetal version. This Catmull-Rom spline type guarantees no self-intersections within the curve segment which is suitable in pathing use case. The t parameter in all the equations above is defined as the values between t_1 and t_2 and depends on how many points are desired on this curve segment.

$$t_{i+1} = |P_{i+1} - P_i|^{\alpha} + t_i \tag{2.10}$$

In order to fully understand how Equation 2.9 and Equation 2.10 are derived, a greater understanding of the algorithm is needed, which is beyond the scope of the thesis. However, as it will be used for the implementation in Section 3.4.2, it has been included in this section to illustrate what the implementation will be based on. More information about the equations can be read at [58] and [14].

Chapter 3

System Design and Development

3.1 System Architecture and Components

This section presents the system architecture and components that have been developed or selected for usage to achieve the objectives.

3.1.1 Hardware Requirements

Certain hardware is required to be able to test the software practically. The proposed hardware requirements are therefore as follows:

- Vehicle with a size and weight that can be easily physically stopped if it goes out of control when testing.
- Vehicle with motors that can be autonomously controlled through the onboard computer.
- A low-cost, small form-factor computer with decent performance that can run on a battery power source as an onboard computer.
- A setup of one and two camera sensors, preferably cameras with low latency.

3.1.2 Hardware Design

Based on the hardware requirements, and a previous bachelor thesis construction of a vehicle given in [54], the following hardware has been used in testing and development.

Vehicle models

The software will first be implemented and tested on the previous bachelor thesis construction of a vehicle, referred to as the *desktop test vehicle* henceforth. This setup uses the differential drive

for steering with limitations on camera hardware because of its size. Thereupon, if the software works as intended, it will be implemented on a larger test vehicle, referred to as *small car test vehicle*, which uses servo steering and higher resolution cameras. More information about the two vehicles setup is described in the following paragraphs.

Desktop Test Vehicle

A modified version (Figure 3.1) of the proposed vehicle from the bachelor thesis is used. As most parts of the desktop test vehicle were 3D-printed or laser cut replacement parts for modifications could be manufactured quickly. The modified vehicle consists of a plastic plate body mounted on top of two stepper motors with wheels for movement and a ball caster wheel for balance. This setup uses the differential drive for steering. For more precise motion control, it is modified to use two stepper motor controllers and two NEMA17 stepper motors instead of the proposed gearbox motors. The motors and motor controllers are powered by a small lightweight 14 V drone battery. The main body plate carries the drone battery, the stepper motor control board, and the single-board computer. To improve the view of the camera it is mounted on top of a platform. With this setup, a small-scale, lightweight, indoor usable prototype vehicle is obtained.

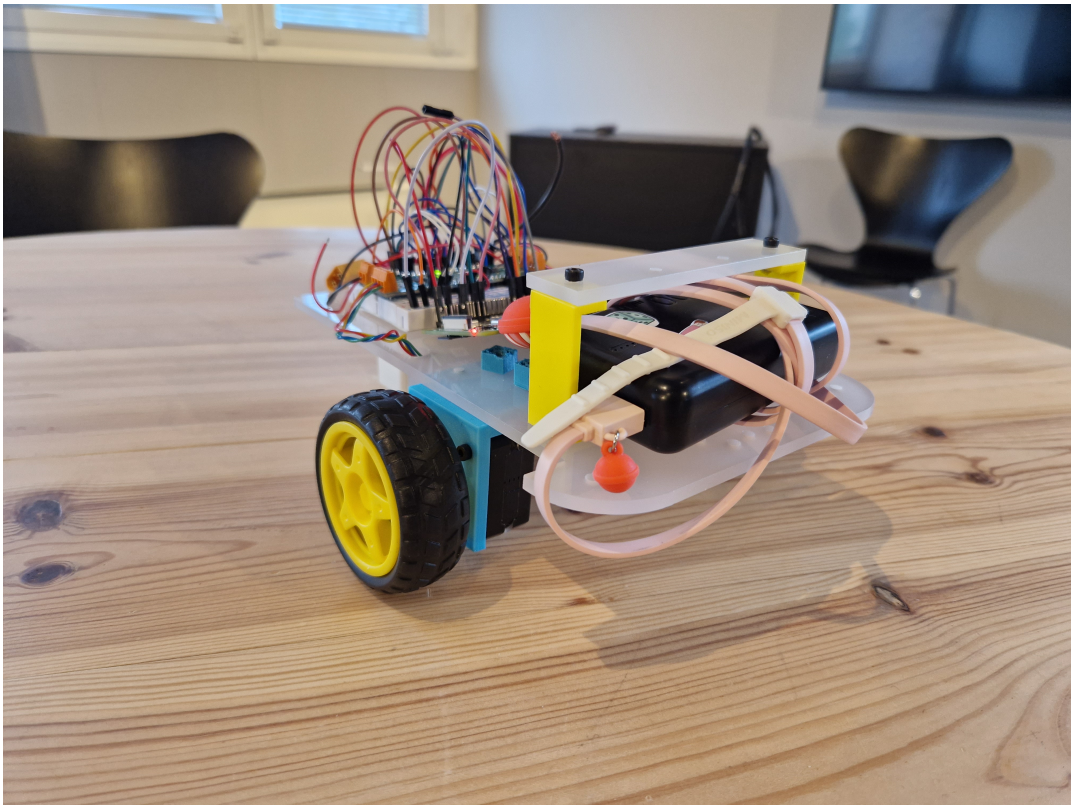


Figure 3.1: The desktop test vehicle

Small Car Test Vehicle

The small-size car (figure 3.2) is a 4-wheel children's bicycle that uses stereo steering instead of differential drive. It controls turns with a servo motor, and uses a stepper motor to drive the

chain. This frame can carry more weight and is more suitable for attaching other sensors, as well as using stereoscopic vision for object detection. It is also closer in both size and configuration, to larger pre-existing three- and four-wheel electric bicycles developed in this region. With its size, it is suitable for driving on "Sykkel og aktivitetsgården" in Sandnes to test the centering of lane function. In addition, it would in most ways be equivalent to the full-scale electric three- and four-wheel bicycles, thus if the software works on this model then, then the software theoretically is possible to use any functional logic from this project on other existing similar vehicles.

Figure 3.2: The small car test vehicle

Single Board Computer

As proposed in the previous bachelor thesis a budget option, the Raspberry Pi 4b single-board computer [43] has been used as the platform board. It features a 64-bit 1.5 GHz ARM CPU, is energy efficient, has multiple USB ports for connecting cameras, etc, features a feature-rich GPIO including SPI for connecting, has built-in Wi-Fi, and is available with either 1, 2, 4, or 8 GB RAM depending on the variants. The Raspberry Pi OS (Debian Bullseye at the time of writing) has a rich set of available software packages including OpenCV and Python 9. With a more powerful board or given hardware acceleration, the real-time performance would improve. However, within the scope of the project, the Raspberry Pi 4b should perform well enough to run the software such that the small-scale vehicle is within the speed limit that is set. In addition, with the limited board performance of the Raspberry Pi 4 compared to other powerful boards, writing efficient code will become an important area of focus.

Camera

Any Linux-supported USB or SPI camera can be used. With the desktop prototype vehicle, Suyin HD cameras with a resolution of 1280x800 have been used. The camera was chosen primarily because of the cost and the size to fit the vehicle. For the small car test vehicle, size was less of an issue, thus a Logitech web camera with a resolution of 1920x1080 was therefore used.

3.1.3 Software Requirements

To create an autonomous driving system there is a need for a *'vehicle control software'* running on the onboard computer to gather camera inputs, perform calculations and control the vehicle. In addition for testing purposes, a software with a graphical user interface for visualizing the gathered information and remotely monitoring the vehicle and the vehicle control software is also needed. The graphical user interface software should also be able to utilize the different computer vision modules without affecting the vehicle control software.

3.1.4 Software Design

Based on the software requirements, the following design has been implemented to accommodate the requirements.

Programming Language and Libraries

OpenCV will be one of the main libraries used in the software for performing computer vision techniques. In terms of programming language, for rapid prototyping and ease of use, Python was chosen as the main programming language to be used. The reason behind this selection is that it is a popular programming language in machine learning because of the good OpenCV-Python bindings despite the fact that it is an interpreted language. As interpreted languages run on a virtual machine (interpreter), an implementation of a software in this language would be slower than similar software in a compiled language such as Rust and C++. However, as most of the demanding algorithms and calculations for computer vision will be performed by OpenCV C++ modules, Python's slower calculation performance will be less noticeable. Moreover, the software relies on threads for asynchronous tasks. Python's reduced performance in multi-threading due to GIL is less ideal, but GIL reduces some of the complexity and pitfalls of creating multi-threaded software (More about this in the next paragraph) [47]. As performance is outside the scope of this bachelor thesis, the reduced complexity makes Python a good choice. And, any well-working algorithm developed can also be ported to C++ or Rust at a later stage for improved performance. The graphical user interface is built by using Qt as it is feature-rich and offers useful tools for interface creation. Furthermore, Poetry is used as a dependency manager for easy installation and package creation, and to keep track of libraries and library versions. To keep consistency in code formatting across the codebase, PyLint is used both locally and in the version control system to ensure readability.

Threads, timers and callbacks

As some tasks have to be performed at the same time, or out of order from the main program loop, some tasks have been put in threads or timers. Multi-threading in Python is not multi-threading because of GIL (Global Interpreter Lock), a mutex that allows only a single thread to hold the control of the interpreter [47]. By using GIL, it simplifies writing multi-threaded code as GIL will work as a global mutex to protect code and make it thread-safe. However, it is not as efficient as running threads in various other programming languages like C++, Rust, or Go. Running threads can still be extremely useful in some places, where code should run in a loop, waiting to act as soon as the thread observes a change. In addition to threads, timers are used where the task should be performed at a set time interval. And callbacks are used when the software is waiting for a change to happen before performing an action or updating the user interface.

Communication

For communication between the headless control software run on the single-board computer on the vehicle and the development GUI run by the developer on a computer, basic UDP sockets are used for communication. The development GUI receives sensor data and motor output from the vehicle and can send instructions to the vehicle. It is possible to stream camera frames from the vehicle and see data output and overlays on the frames by running the same computer vision modules on them as performed on the single-board computer.

The headless control software opens a socket server when it starts up, listening on a port for incoming connections. Depending on configuration it will also open socket servers on configured ports for streaming camera frames. If there are connected clients, it will send data to all the connected clients.

The development GUI computer control system uses a thread to connect to the driving software and listen for incoming data. All incoming sensor data is placed in a database, and at a set interval, a timer will trigger a function to update the display. When connecting to the socket camera server a thread will spawn to listen for incoming frames. Whenever a frame is received it will run the selected OpenCV functions, and display the new image in the GUI with or without overlay lines to highlight what is found. A label is also used to display relevant data.

An initial performance test on streaming frames K showed that with ideal conditions for performance, running a stripped-down Python script that would only stream image frames to a single client, it achieved up to 5 FPS.

OpenCV Videostream

OpenCV video capture method was used for capturing frames using a video stream. An unexpected delay was observed from a physical change done to the headless control software. This was noticed while the headless control software was running at a rate of one frame being processed per second, and it would take five frames before the change was observed in the QR-code logic. This delay is caused due to the buffer size in OpenCV [38] [48]. To counteract this delay,

a loop to clear the buffer was added.

Overall Architecture

The code has a modular design to make it reusable between the headless control software for the vehicle and the development GUI software. It is also designed to be modular in the aspect of communication type for motor control, communication, and other tasks. There is an abstract class for car communication, and separate modules for serial communication (UART), stepper motor communication (GPIO output and PWM), and CAN Bus communication. Additional communication modules can easily be created and added by using the abstract class as a template. This allows the implementation of computer vision algorithms to be unaffected by hardware modifications. Thus, by performing hardware modifications, the autonomy logic does not need to be rewritten.

Both the headless control software and the development GUI are launched through the `__main__.py` file. It takes in any given command line argument. The command line arguments can modify the path for the standard configuration file, theme/Qt design file, storage option, and the selection of window mode for the GUI. The information in the standard configuration file (and the installed dependency modules) determines if the software will run development GUI or the headless control software. The file is a YAML type of configuration file and a template for this file can be found in the GitHub repository given in Appendix A. Furthermore, all the computer vision and calculation modules are shared for both the development GUI and the headless control software. The overall flow of the software is presented in Figure 3.3.

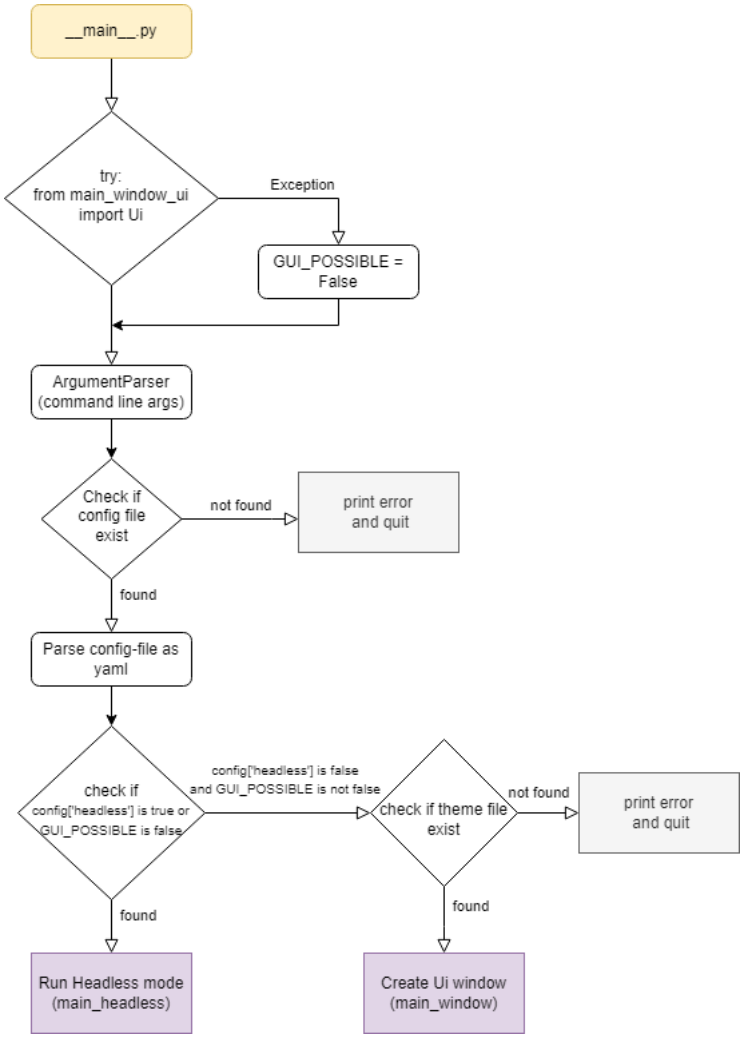


Figure 3.3: The flowchart of the starting file (`__main__`) for the software. The program flow for the blocks *Run Headless Mode* and *Create UI Window* are presented respectively in Figure 3.4 and Figure 3.5.

Headless Control Software

The headless control software is made to be run on an embedded system in the vehicle. It is modular, and using the configuration file it is possible to configure what communication modules are going to be used to communicate with motors and sensors. It starts by loading and initializing the required modules and then starts a loop where it gathers a new frame from the camera. For each frame, it will run the appropriate computer vision modules depending on the current state. The term "Frames Per Second" (FPS) is used to describe the performance of the system for how many frames the software can take and process per second. In terms of this software, that calculation refers to how many times this loop is processed per second. A flowchart describing the flow of this headless control software is given in Figure 3.4.

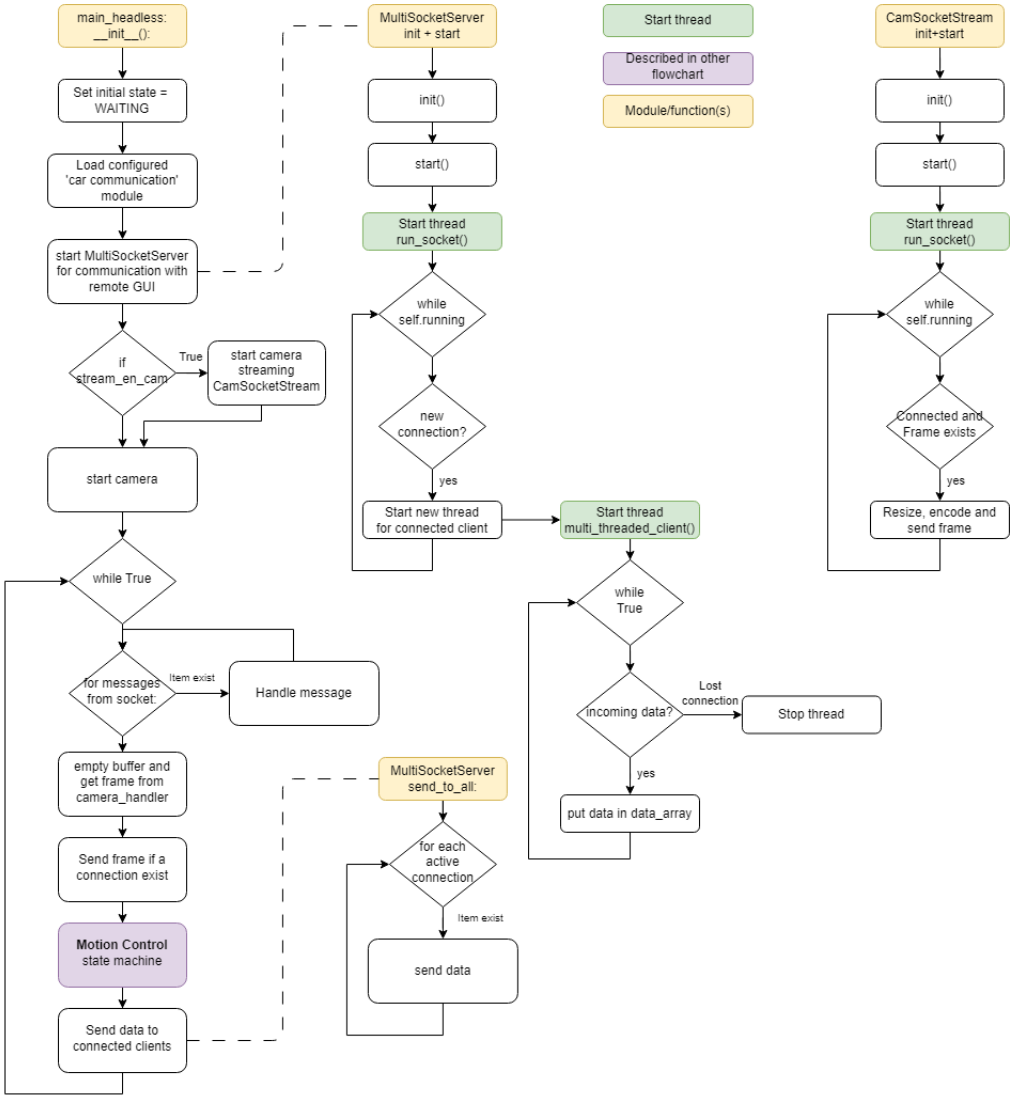


Figure 3.4: The headless control software flowchart (`main_headless`). The purple box *Motion Control state machine* refers to Figure 3.6.

Development GUI Software

The Qt-based development GUI software is made to be used for testing computer vision modules, controlling the vehicle's actions, and gathering valuable data from the vehicle while it is running the headless control software. The gathered data can be useful in debugging situations where the vehicle does not act as expected.

As shown in Figure 3.5, the software initializes the graphical interface by filling the Qt interface with options in QtComboBox drop-down lists and checking for locally connected cameras on the computer to connect to. This includes reading available states from an enumerator found in the shared `Definitions.py` file, and reading IP addresses and ports for possible socket servers (vehicles) from the configuration file. In addition, it also prepares a timer for updating text output from the computer vision modules.

After being initialized, the user interface can be used to run computer vision modules on the camera frames from a connected camera, or from the vehicle over a UDP connection. It can control the vehicle and receive data from the vehicle software over a network connection. In addition, the graphs of relevant logged data will be presented to the user. The design for this user interface is presented in Section 3.6.

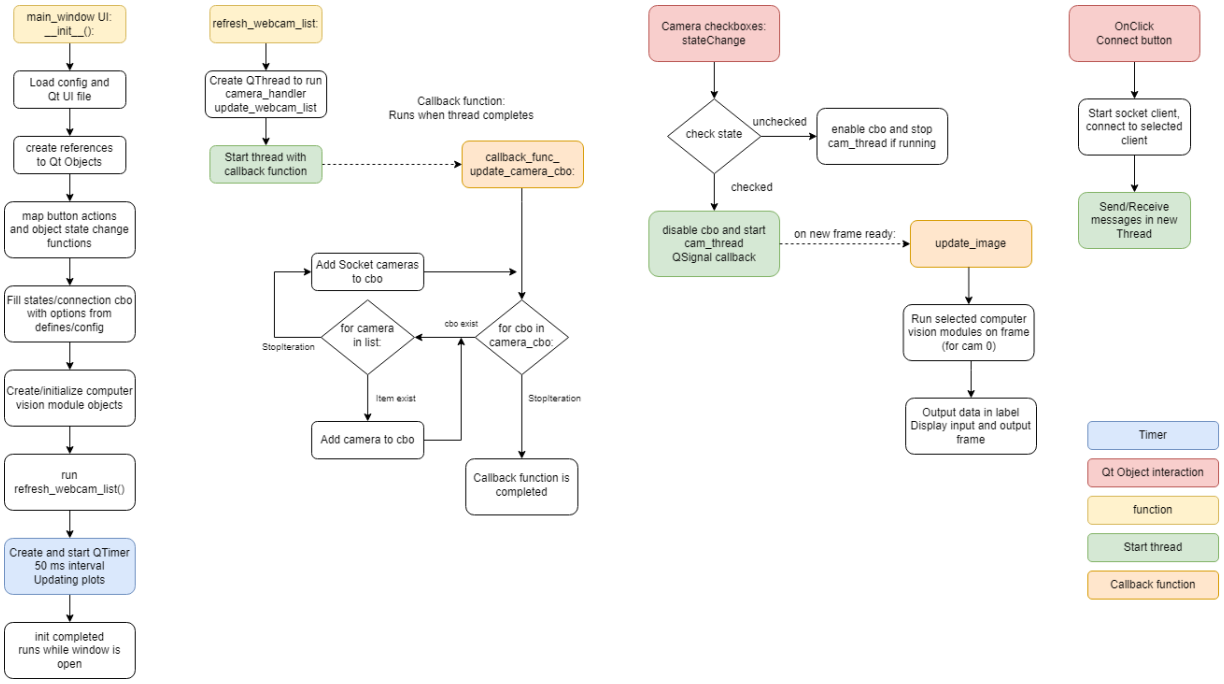


Figure 3.5: The development GUI software (`main_window`) flowchart

Modules

The software will include a collection of modules for different computer vision techniques and other relevant concepts needed for autonomous driving functions. The different modules are listed below with a short description of their usage:

- **Stereoscopic Vision**: Uses depth detection method to do hindrance detection.
- **QR Code Detection**: Uses object detection method to detect QR-code which represents empty parking slots.
- **Stop Sign Detection**: Uses LBP Cascade object detection technique to detect stop signs.
- **Line Detection**: Uses image processing methods such as edge detection to detect lines.
- **Lane Detection**: Detects the lane lines by using the Line Detection module, and calculates a point that is in the middle of the lane that can be used as a destination point for the vehicle to stay in the lane by using perspective transformation.
- **Parking Slot Detection**: Uses the QR Code Detection module to determine a region of interest and uses the Line Detection module to extract the lines from the region of interest.

- **Environment Mapping**: Creates a two-dimensional matrix that represents the vehicle's environment and maps lines and objects into this environment.
- **Path Planning**: Uses a combination of a shortest path algorithm, spline, and the Environment Mapping module's information to calculate path information.

Headless control software driving logic

The vehicle driving logic consists of six states. The first two are used for testing purposes and the last four are the objectives of the thesis:

- **WAITING**: The vehicle will not drive when in this mode, but will only print out information about objects or lines detected.
- **MANUAL**: Manual control of the vehicle from the development GUI software. Manual driving
- **STEREO**: The vehicle will drive with a constant speed when it is in this mode as long as it does not detect any hindrances from the Stereoscopic Vision module.
- **STOPPING**: The vehicle will drive with a constant speed when it is in this mode as long as it does not detect any stop signs from the Stop Sign Detection module. The vehicle will reduce its speed when a stop sign is detected and will stop the movement completely when it is within a given distance.
- **DRIVING**: The vehicle will use the Lane Detection module to detect the lane lines and define a point in the center of the lane. Thereafter map this point to the 2D environment and use the Path Planning module to calculate a path from the vehicle to the point. The calculated path will then be used to adjust the motors as needed.
- **PARKING**: The vehicle will use the Parking Slot Detection module combined with the QR Code Detection module to detect the parking slot. The parking slot lines will be mapped as obstacles in the Environment module and the QR code will be the desired destination in the Path Planning module. The calculated path will then be used to perform motion control.

The flow chart of the proposed driving logic is presented in Figure 3.6. The driving logic accommodates the problem highlighted in Section 3.3 by implementing an action queue to save calculated decisions from the previously given image frame. Moreover, the driving logic focuses on constantly using the new frame if there is enough information to calculate a new path such that deviations from the motion control are taken into account for the next driving actions.

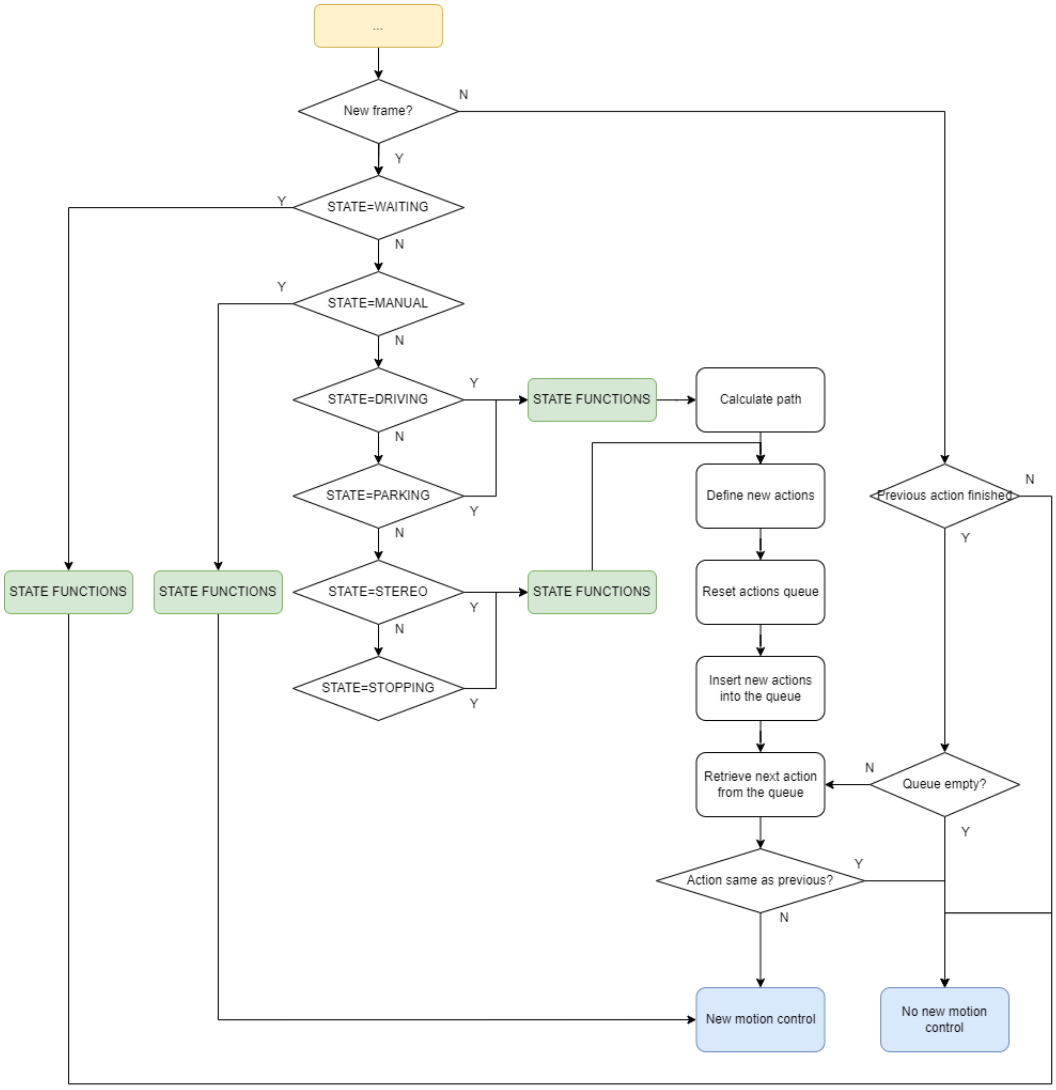


Figure 3.6: Flow chart of the driving logic. The yellow box refers to the process before the *Motion Control state machine* in Figure 3.4.

3.2 Computer Vision Techniques for Autonomous Driving

This section provides a comprehensive overview of the usage of the computer vision techniques and algorithms presented in Chapter 2. First and foremost, this section covers the process of calibrating the camera. Following that, the object detection implementations, which surround Stereoscopic Vision, QR Code Detection, and Stop Sign Detection, are presented. Finally, the section presents the development and implementation of the Line detection module and the use of Line Detection in Lane Detection and Parking Slot Detection modules.

3.2.1 Camera Calibration and Rectification

Camera calibration is a crucial step for computer vision applications that needs to retrieve metric information of a camera frame as it removes lens distortion [13]. It is a process for estimating a camera's internal and external parameters [1] that can be used to nullify image distortions. Two common types of distortion in cameras are radial distortion and tangential distortion, which can cause straight lines to appear curved and introduce curvature to objects in a photo [35]. The following steps describe the process of calibrating the cameras to resolve this. The steps are based on a calibration pattern approach involving a checkerboard pattern with sharp gradients in two directions for precise localization. A more detailed explanation is given in [50] and [37].

1. Define physical world coordinates for the checkerboard

To define physical world coordinates for the checkerboard, each point is assigned a set of physical coordinates (X, Y, Z) corresponding to a pixel location (u, v). As the checkerboard lays flat, Z can be arbitrarily chosen as 0 for every point. The equal spacing of the checkerboard allows each 3-dimensional point to be easily defined using a reference point (0, 0). Finally, multiple images are taken from different angles and distances to improve the calibration.

2. Find 2-dimensional coordinates of the checkerboard

The next step is to project a 3-dimensional world into a 2-dimensional image using functions from OpenCV's library. The first step is to find the corners of the checkerboard [20] with the `cv2.findChessboradCorners()`[2] function as presented below. The output of the previous function is then used in the function `cv2.cornerSubPix()`[3]. The goal of this function is to return an array of refined corners with sub-pixel accuracy. By using the function `cv2.drawChessboardCorners()`[2] and the output obtained from the previously performed function as the input in this new function, the calibration can be illustrated as shown in Figure 3.7.

[1] Information about the parameters can be found at [26].

[2] The parameters for the functions `cv2.findChessboradCorners()`, `cv2.drawChessboardCorners()`, `cv2.undistort()`, `cv2.drawChessboardCorners()`, `cv2.stereoCalibrate()`, `cv2.stereoRectify()` and `cv2.initUndistortRectifyMap()` are described in [26].

[3] The parameters for the function `cv2.cornerSubPix()` are described in [30]

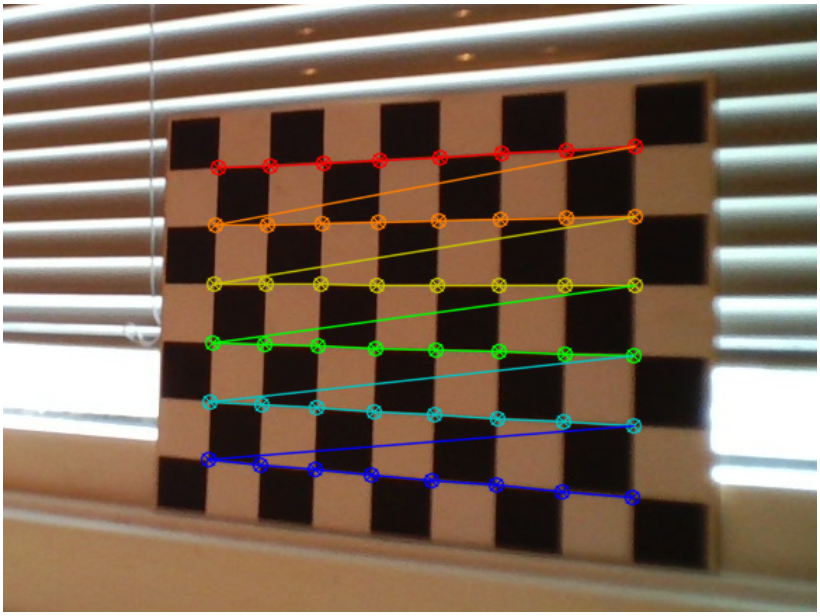


Figure 3.7: An illustration of the distortion on the checkerboard image using the OpenCV inbuilt function `cv2.drawChessboardCorners()` to draw the corners.

3. Calibrate the camera

Finally, to obtain a matrix that can be used to correct the iamges, the `cv2.calibrateCamera()`[2] is used with the calculated information from the previous steps. The obtained matrix is then used with the function `cv2.undistort()`[2] to resolve the distortions mentioned at the beginning of this section.

3.2.2 Stereoscopic Vision

As described in Section 2.2, stereoscopic vision uses two cameras to be able to detect obstacles. First and foremost, to achieve this, a stereo calibration is performed to calibrate the cameras concerning each other. This is done as the output is dependent on the correlation between the cameras. Various methods from OpenCV are then used to compare the two images from each camera. Because of its many applications, the implementation is not groundbreaking and the majority of the methods are inspired by [19] and [39]. However, changes are done to accommodate this project.

Stereo Calibration and Rectification

Stereo calibration is the process of determining the intrinsic and extrinsic parameters of a stereo camera system [22]. By calibrating a stereo camera, we can accurately determine the 3D position of objects in the scene by triangulation, which is a fundamental step in stereoscopic vision. Additionally, accurate stereo calibration can improve the accuracy and robustness of other computer vision tasks such as depth estimation, object tracking, and structure from motion. The solution is based on using the two OpenCV functions, correspondingly,

`cv2.stereoCalibrate()`[2] and `cv2.stereoRectify()`[2]. The result from these functions is used with the `cv2.initUndistortRectifyMap()`[2] which creates a map for the rectifications so the rectification can be performed on new images by remapping.

OpenCVs remap function remaps[4] the images based on the calculated stereo rectification map. With this function, there is multiple *interpolation* flags that may be used [31]. For this case, the *INTER_LINEAR* that is a bilinear interpolation is used because of the shorter computing time than other functions like *Lanczos4*. It is also less prone to blurring than *nearest-neighbor* interpolation function, and it is less sensitive to noise and distortion than a more complex function like *bicubic* or *Lanczos4*. By using this function, all new images can easily be rectified.

Creating a disparity map

Moreover, the rectified images are the input to a stereo block-matching algorithm which is used for computing disparity maps in stereo vision [56]. There are different block-matching algorithms that can be used. In this project, a semi-global block matching algorithm was utilized because it creates a more complete disparity map compared to a local block matching algorithm [21]. However, it is more computationally expensive compared to the local area block-matching algorithm [21]. More information about the semi-global block-matching algorithm and the local area block-matching method can be found at [25] and [8] respectively. The disparity map is created by calling the `compute()` method from the `cv2.stereoSGBM_create()` class with the stereo images as input.

Disparity Map to Depth Map

A depth map can be created by dividing the disparity map with a constant "M". This constant is calculated by registering the disparity value of an object at certain distances that combined corresponds to a linear system that can be solved. The OpenCV function `cv2.solve()` is used to solve this linear system.

Baseline

As mentioned in Section 2.2, the baseline is the distance between the two cameras. Setting the baseline of a stereoscopic vision system to the average interpupillary distance of humans is a common practice [12]. This provides a reasonable starting distance for detecting obstacles in front of a vehicle. Increasing the baseline improves depth resolution, while decreasing it improves accuracy [12]. However, in the current implementation, it is not necessary to increase the baseline for improved depth resolution, as the goal is to only detect obstacles that could be a hindrance close to the vehicle. Similarly, a decreased baseline is not necessary for detecting object shapes accurately. Therefore, to keep the baseline at a reasonable distance, the cameras will have the recommended baseline of 90mm.

[4]The process of taking pixels from one place in the image and locating them in another position in a new image [36].

Detecting Obstacles

Because the primary use of this technology in this project will be detecting obstacles at a close distance in front of the vehicle, the generated disparity map will be filtered by a minimum and a maximum distance. The maximum distance should depend on the speed of the vehicle. When the vehicle is driving slowly, a lower range is needed as opposed to a higher speed where a larger range is needed. This is so that the vehicle has time to stop before it hits the hindrance. The minimum distance is at what minimum distance it should detect the obstacles and should in this project be set to 0 cm as the goal is to detect obstacles close to the car. Thereafter, the contours of the filtered disparity map are extracted and iterated through. The size of the closest contour is checked and will return the depth if it is significant enough. If not, the next contour is selected and reviewed.

Implementation

A class **StereoscopicVision** has been made as an implementation of stereoscopic vision with the following methods:

- **read_stereo_map:** Gets camera calibration and rectification information from an XML file.
- **get_disparity:** Returns the disparity map created from left and right stereo images.
- **obstacle_detection:** Calculates the depth, position, and size of the detected contour.
- **get_data:** Returns the data from `obstacle_detection` with a boolean return value stating if an object could be found.

3.2.3 QR Code Detection

This section focuses on QR code recognition as one of the objectives is to park at a spot that is indicated by a QR code. The detected QR codes will be later on used to detect unoccupied parking spots and as the destination point in path planning. OpenCV's built-in functions are used for QR code detection, which identifies the corners' positions. These corner points are then utilized to calculate the angle and distance from the vehicle to the parking spot.

Theory

The four points in Figure 3.8 are used to calculate the sides of the squared QR-Code, which is then used to calculate the distance to the sign.

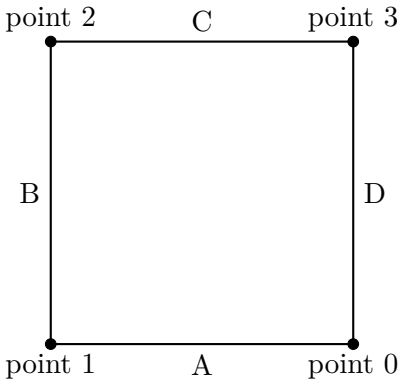


Figure 3.8: Illustration of a QR-Code.

When the QR code is viewed from an angle, as illustrated in Figure 3.9, the height-width ratio increases, which affects the perceived size.

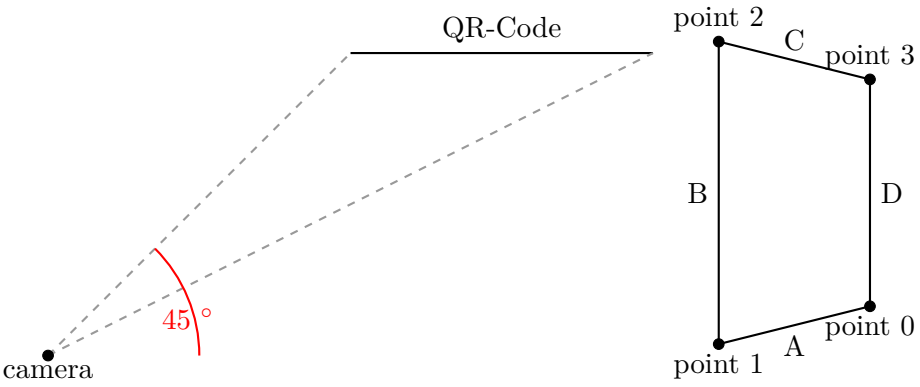


Figure 3.9: Left figure shows the QR-Code seen from a 45-degree angle. The figure on the right shows how the QR code would look from a two-dimensional perspective when looked at a 45-degree angle.

To calculate the angle, the following equation is used where ratio is the height-to-width ratio of the QR code and the height value used to calculate the ratio is the largest height value:

$$\text{angle} = (1 - \text{ratio}) * 90 \tag{3.1}$$

To calculate the distance, a reference value of the QR code at a specific distance denoted as y_{offset} is necessary. Equation 3.2 is used to calculate this reference value and the variables in the equation are determined by physically measuring the specified values in the equation at a certain distance.

$$y_{\text{offset}} = \frac{\text{height}_{px}}{\text{height}_{mm}} * \text{distance}_{mm} \tag{3.2}$$

The distance in the y-direction is then calculated by finding the ratio between the QR code height and the measured pixel height, thereafter multiplying it by the y_{offset} as shown in Equation 3.3.

$$y_{\text{distance}} = \frac{\text{height}_{mm}}{\text{curHeight}_{px}} * y_{\text{offset}} \tag{3.3}$$

The height_{mm} is the height of the QR code and curHeight_{px} is the measured number of pixels in the y direction of the QR code. The multiplication factor y_{offset} is calculated in Equation 3.2. Similarly, the x-distance is calculated using the following equation:

$$x_{\text{distance}} = \frac{\text{curWidth}_{px}}{\text{width}_{px}} * x_{\text{offset}} \tag{3.4}$$

where curWidth_{px} is the current width of the QR code in pixels, width_{px} is the measured width of the QR code in pixels at a specific position and x_{offset} is the pixel offset from the center of the camera.

Implementation

To detect the QR-Code, the methods from the OpenCV library presented in Code 3.1 have been used.

Code 3.1: OpenCV methods to detect QR-codes.

```
1 qcd = cv.QRCodeDetector()
2 retval, decoded_info, points, straight_qrcode = qcd.detectAndDecodeMulti(img)
```

Parameters Description

retval: Indicates if a QR code is detected.

decoded_info: Contains the decoded information about the detected QR code.

points: Contains the corner points.

straight_qrcode: Contains an optional output vector of the image including rectified and binarized QR codes.

A class **QRGeometry** is made to calculate QR code metrics and includes the following methods:

- **get_width:** Calculates the width of the QR code.
- **get_height:** Calculates the height of the QR code.
- **get_angle:** Calculates the angle to the QR code.
- **get_distance_y:** Calculates the distance to the QR code in the y direction.
- **get_distance_x:** Calculates the distance to the QR code in the x direction.

Another class **QRCode** has been made with the following method:

- **get_data:** Implements Code 3.1 and returns the data about the QR code, which includes the distance and angle to the QR code that is calculated by using class `QRGeometry`.

3.2.4 Stop Sign Detection

To be able to create the model based on the Local Binary Pattern method described in Section 2.3 in order to detect the stop signs, the guide by OpenCV at [29] has been used.

The initial step is to create two text files. One includes information about which image is the negative image. The other one includes which image is the positive image and the location of the object in the image. An example of the information file for negative images and positive images are respectively shown in Code 3.2 and Code 3.3.

Code 3.2: Example of information file for negative images.

```
...
negative/1.png
negative/2.png
negative/3.png
...
```

Code 3.3: Example of information file for positive images.

```
...
positive/1.jpg 1 12 22 323 332
positive/2.jpg 1 190 82 300 281
positive/3.jpg 2 10 71 137 148 183 58 106 100
...
```

To be efficient and not manually write down each negative image's filename, a simple Python code has been made to create this information file for negative images. This code is presented below.

Code 3.4: Code for creating a text file containing the path of each negative image in the folder with all the negative images.

```
import os

def gen_desc_for_neg():
    with open('bg.txt', 'w') as f:
        for filename in os.listdir('./negative'):
            f.write('negative/'+ filename + '\n')

if __name__ == "__main__":
    gen_desc_for_neg()
```

When selecting positive and negative images for training the following standards have been used:

- Positive images should include the sign in different background settings.

- Negative images should have similar background settings as the positive images, but without the sign.

Parameter Selection

All the parameters mentioned in this section vary from the amount of positive and negative images used in the training process. The parameters used in the different OpenCV applications are described at [29]. Suggestions from [5] have been taken into consideration when selecting the set of parameters to create the model.

For the information file of positive images, OpenCV includes an application that iterates through each image in the folder of positive images and lets the user mark the position of the desired object on the image. The command used in a terminal for this application is shown below.

```
opencv_annotation --annotations=info.txt --images=/positive/
```

After creating the positive information file, this file is used in another application of OpenCV to create a vector file that contains positive samples of the positive image collection. The usage of this feature is shown below.

```
opencv_createsamples -info info.txt -w 24 -h 24 -num 1000
-vec pos.vec
```

The vector file and the information for the negative images are then used in another OpenCV application for training the model. The usage of this feature is shown below where `pos.vec` is the vector file and `bg.txt` is the information file for the negative images.

```
opencv_traincascade -data cascade/ -vec pos.vec -bg bg.txt
-w 24 -h 24 -numPos 1000 -numNeg 2000 -numStages 7
-maxFalseAlarmRate 0.5 -minHitRate 0.995
```

The output of this step is the model in the XML format that is used in the OpenCV function `cv2.CascadeClassifier()` to perform the concept of Cascade of Classifiers to detect the signs as shown in the following code snippet. The `minSize` defines the minimum pixel size of the stop signs the model is able to detect.

Code 3.5: Using the model to detect the stop signs.

```
1 cascade = cv2.CascadeClassifier('cascade.xml', minSize=(X, X))
2 signs = cascade.detectMultiScale(frame)
```

Furthermore, to calculate the distance from the vehicle to the stop sign, the same method to determine the QR-code distance given in Section 3.2.3 has been used.

A class **StopSignDetector** has thereafter been constructed to utilize the cascade model. The class includes these methods:

- **detect_signs**: Uses Code 3.5 to detect the signs.
- **get_distance**: Calculates the distance to the stop sign by using Equation 3.2 and Equation 3.3.
- **show_signs**: Draws a rectangle around the signs on the image.

3.2.5 Line Detection

The implementation of detecting lines for the use case of parking and centering the vehicle is based on applying a set of functions from OpenCV that corresponds to the theory presented in Section 2.4. The code snippet of this implementation is presented below.

Code 3.6: Image processing methods to detect lines.

```
1  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
2  gaussian_blur = cv2.GaussianBlur(
3      gray,
4      (kernel_size, kernel_size)
5  )
6  canny = cv2.Canny(gaussian_blur, low_treshold, high_treshold)
7  erode = cv2.erode(dilate, (kernel_erode, iterations=num_iter_erode)
8  dilate = cv2.dilate(erode, (kernel_dilate, iterations = num_iter_dilate)
9  lines = cv2.HoughLinesP(
10         canny, rho, theta, threshold, np.array([]),
11         minLineLength, maxLineGap
12 )
```

A brief explanation of Code 3.6 is as follows:

- **Line 1:** The function `cv2.cvtColor()` converts the input image to grayscale as a prerequisite step for edge detection by using the parameter value `cv2.COLOR_BGR2GRAY` to indicate that it is a grayscale conversion.
- **Line 2-5:** The function `cv2.GaussianBlur()` uses the Gaussian Blur method to reduce the noise in the image. The parameters for this function are described in [33].
- **Line 6:** The function `cv2.Canny()` performs the Canny edge detection method on the input image. The parameters for this method are described in [30].
- **Line 7-8:** The function in Line 6 `cv2.erode()` starts the process of morphological closing by performing the morphological transformation method erode on the input image. Thereafter, the output of erode is in Line 7 used in the function `cv2.dilate()` to perform the second morphological transformation method. The parameters for these two methods are described at [33]. Although OpenCV provides a single method for morphological closing, as the implementation aims at the possibility of changing parameters independently for cases where one method needs to be applied more than the other, individual methods have been used instead.
- **Line 9-12:** The function `cv2.HoughLinesP()` retrieves the line in the input image by using the Hough transformation. The parameters for this method are described in [30].

Parameter Selection

All the parameters mentioned in Code 3.6 vary from use case to use case and therefore are selected through the try-and-fail method for each use case.

A class **LineDetector** has been constructed to include the previously presented implementation and other necessary operations. The class includes the following methods:

- **get_region_of_interest**: Returns the whole image as the region of interest. An abstract method to indicate applications of LineDetector can overwrite when another region of interest is preferable.
- **get_lines**: Includes the implementation given in Code 3.6 to retrieve the lines in an input image.
- **show_lines**: Shows the lines on the image based on lines given from `get_lines`.

3.2.6 Lane Detection

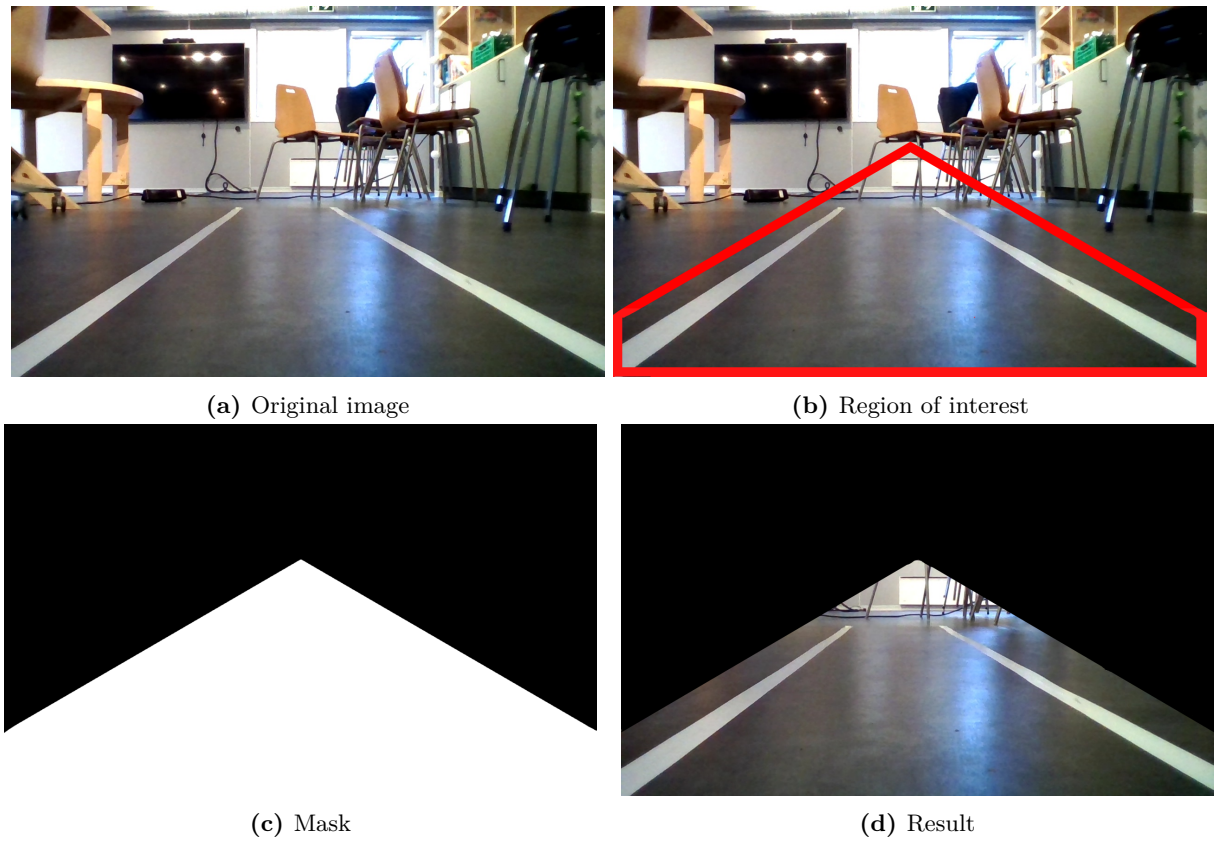
One of the objectives of the project is to keep the autonomous driving vehicle centered on the road when driving forward. Therefore, the system needs to be able to detect the lane lines which present the road in order to make further calculations and achieve the previously mentioned objective. The line detection method from Section 2.4 will therefore be used as one of the steps to detect the lane lines. However, in order to be able to detect the lane with the least possible error, the two steps, defining the optimal region of interest and classifying lines, must also be performed. Furthermore, to achieve the mentioned objective, an approach for centering the vehicle on a road is to create a point that is at further away from the vehicle. This point can then be used as the destination for the vehicle. The goal of the vehicle is to reach this point as this means it is now in the center of the road. Thus, the detected lane lines are needed to calculate this point in addition to the perspective transformation method given in Section 2.6.

Region of Interest

When driving forward on a road, there might exist other objects in the camera's field of view. By performing the line detection immediately on the whole camera's field of view, a large number of the returned lines are of the surroundings and not of the lane lines. These are therefore not relevant for detecting the lane lines. Furthermore, having a larger set of irrelevant lines will cause the step of classifying lines to be less accurate. An optimal region of interest is therefore needed to avoid this problem.

Since the camera is placed on the vehicle at a height above the ground, the lane lines will approximately form a triangle. The region of interest will therefore have the same form. However, the lane lines might not be within the triangle which expands from the two bottom corners of the camera frame if the lane is too wide. An extra rectangle added to the bottom of the triangle can be used so that the whole lane line is included in the region of interest. The size of the rectangle and triangle is adjusted based on the lane width and the camera height above the ground.

To extract the region of interest from the image, a mask is made out of the previously mentioned region and thereafter applied using the bitwise AND operation between the mask and the original image. The region of interest should be white on the mask while the other regions should be black. The reason behind this is due to the fact that white is (255, 255, 255) in RGB, meaning in bits white is all ones. By doing the AND operation on the regions, the result is the original values. Meanwhile, black is (0, 0, 0), meaning in bits black is all zeros. By doing the AND operation on the black regions, the result is equal to the mask, meaning this part will only be black. An example to visualize this operation is shown in Figure 3.10.



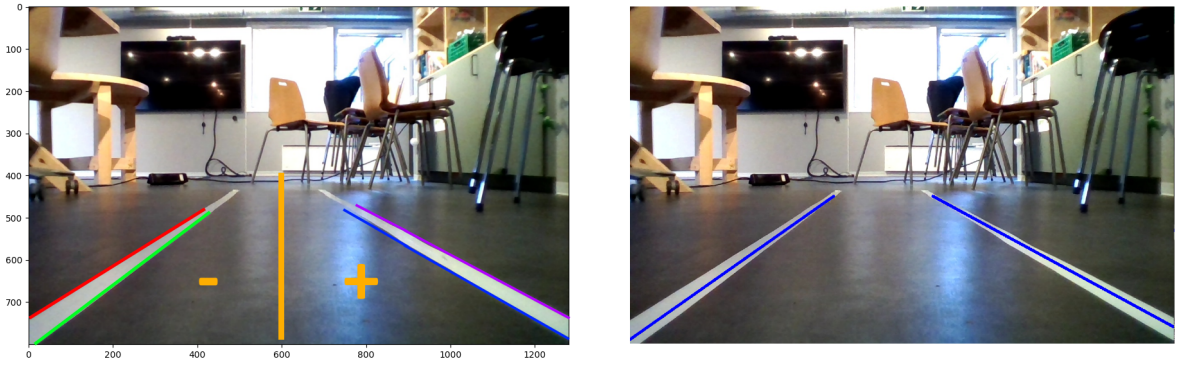
(a) Original image **(b)** Region of interest

(c) Mask **(d)** Result

Figure 3.10: The steps of extracting the region of interest using masking.

Classifying Lines

After extracting the region of interest, the line detection method from Section 3.2.5 can be used to retrieve only lines within the region of interest. The set of lines obtained through the line detection method is thereafter classified by using the slope value method given in Section 2.5. Two ranges for each lane line group are defined based on the fact that the origin of the coordinate system of OpenCV is in the top left corner. The first range corresponds to a positive slope value, implying it is on the right side of the vehicle. The second range corresponds to a negative slope value, which means it is on the left side. Figure 3.11a visualizes this established classification of lines. From the figure, it is possible to see that the lines that correspond to the left lane line (red and green) both have negative slope values. On the other hand, the lines that correspond the right lane line (blue and purple) both have positive slope values.



(a) Classifying lines into two groups

(b) The average lines of the two groups

Figure 3.11: The steps of determining the lane lines from a set of lines.

Afterward, for each group, the average slope and intersection of the lines belonging to the group are calculated. The result of doing this calculation for the two groups is the average lane lines that respectively are the left lane line and the right lane line as shown in Figure 3.11b.

Center Difference

The difference between the center of the lane and the center of the camera can be calculated by using the two-lane lines. This value is used to display it on the user interface software for an approximation of where the vehicle is in the lane. The center of the camera frame can be found by dividing the width of the frame in two. For the center of the lane, it is possible to calculate this center using one point on each line that is closest to the camera. Defining the point on the left line as (x_1, y_1) and the point on the right line as (x_2, y_2) the equation for this calculation is as shown below:

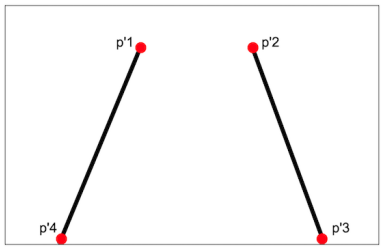
$$\text{diff} = x_1 + \frac{x_2 - x_1}{2} \tag{3.5}$$

If the result of Equation 3.5 is negative, it corresponds to the center of the vehicle being to the left of the center of the lane. In contrary, if the result is positive, then it corresponds to the center of the vehicle being to the right of the center of the lane.

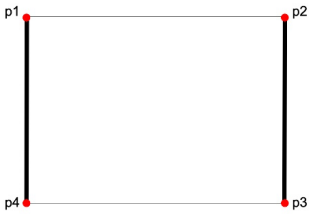
Checkpoint using Perspective Transformation

By using the perspective transformation method presented in Section 2.6 and the detected lane lines, it is possible to calculate the point that will be used as the destination for the vehicle to center itself on the road. In this scenario, there exist two perspectives: the 3D 1-point perspective, which is the original view seen in the camera frame, and the 2D world, which is the detected lane itself seen from above such that the endpoints of the lane lines correspond to the corners in this perspective. These perspectives are respectively shown in Figure 3.12a and Figure 3.12b. First, the original frame is converted from the 3D perspective with the detected lanes into the 2D perspective by defining the width of the 2D world frame as the difference

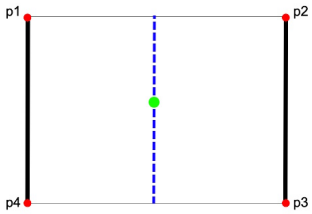
between $p'3$ and $p'4$ in the x-value. These two points are the two closest endpoints of the lane lines as shown in Figure 3.12a. The height of the frame on the other hand can be an arbitrary value depending on the desired distance from the vehicle to the point. The x value of the point will be half of the width while the y value is arbitrary as this value also depends on the desired distance away from the vehicle. Thereafter, to convert the point that has been calculated in the 2D perspective to a point in the 3D perspective, Equation 2.1 is used. The visualization of the steps are presented in Figure 3.12



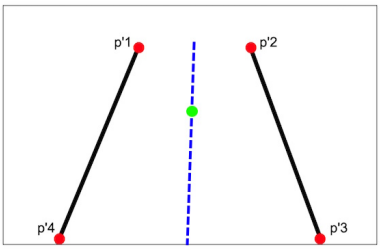
(a) Original image frame in the 3D perspective with the two detected lane lines.



(b) Convert to 2D perspective with the four endpoints of the two-lane lines as the frame corners.



(c) Define a point in the 2D perspective which is in the center of the lane.



(d) Convert the point from the 2D perspective to the 3D perspective.

Figure 3.12: The steps needed to define a point that is in the center of the lane in the original frame in 3D perspective. The blue dashed line presents the center of the lane. The red circles illustrate the endpoints of the two-lane lines. The black thick lines represent the lane lines. The grey lines represent the border of the frame. The green circle presents the point in the center of the lane.

Implementation

A class **LaneDetection** has been made as an implementation for this module. The methods in this class are based on the previously mentioned theory and are listed and described below:

- **get_region_of_interest**: Uses Numpy's logical AND function to retrieve the triangle region of interest.
- **get_average_lines**: Converts lines presented by two points to a line presentation using slope and intercept by using Numpy's `polyfit` method. Thereafter classifies the lines using the slope value and calculates the average line of the left and right groups.
- **get_line_coordinates_from_parameters**: Converts line representation from slope and intercept to two points.

- **get_diff_from_center_info**: Calculates the difference between the camera center and the lane center.
- **get_next_point**: Calculates a point on the center of the lane that is at further parts of the road defined by two lane lines by using perspective transformation.
- **get_lane_line**: Retrieves the lane lines from an image presented in coordinate form by using `get_average_lines` and `get_line_coordinates_from_parameters`.

3.2.7 Parking Slot Detection

Another objective of the thesis is to park in a spot indicated by a QR code and two parking lines. Like Lane Detection, the Parking Slot Detection module also needs to utilize the Line Detection module to detect the parking lines. Furthermore, as mentioned in Section 3.2.3, the QR code will be used as a destination point for path planning when parking. On the other hand, the parking lines will be used to define the shape of the parking spot. By perceiving the parking lines as obstacles in path planning, it is also possible to avoid driving onto the lines or that the ending position is in such a way that the wheels of the vehicle are on the parking lines.

Region of Interest

First, the region of interest is the whole image as it should be able to detect the QR code anywhere on the image. After the QR code is detected, the region of interest is reduced to being below the line that goes through point 0 and point 1 in Figure 3.8. An example of this is shown in Figure 3.13. The same method as in Lane Detection with bitwise masking is then used to extract the region of interest.

(a) QR Code detected **(b)** Extracted the region of interest

Figure 3.13: Example of a region of interest for detecting parking lines. The red marks in (a) present the points 0 and 1 in Figure 3.8 of the QR code and the blue line represents the line that goes through the two points.

Merging Lines

After identifying the desired region of interest, the method of detecting lines from Section 3.2.5 will be used to retrieve lines in the given region of interest. The set of lines returned by the line detection needs to be merged by using the algorithm in Section 2.7. This is due to the result from the line detection might return one physical line as multiple detected lines as mentioned in Section 2.4 and shown in Figure 3.14a. The proposed solution in the previously mentioned section is therefore used to achieve the desired outcome as shown in Figure 3.14b.

(a) Without merging

(b) With merging

Figure 3.14: Example of merging closed parking lines. In subfigure (a) there are two blue lines in each red oval. In (b) the two lines from (a) are now represented by one line.

Determine the Relevant Parking Lines

With the system using a shortest path algorithm for decision-making like mentioned in Section 2.9, the QR code must, first of all, be enclosed within the two parking lines that indicate the empty spot. In addition, an extra line indicating another obstacle between the edge of the two parking lines on the side to which the QR code is the closest to is needed for the path planning algorithm. This is to create a path in which the vehicle will not end up in the back of the QR code. This extra line is shown in Figure 3.15 as the orange line which passes through the points $(x2, y2)$ and $(x3, y3)$.

Although the project is limited to parking when only one QR code is detected, it does not eliminate the possibility of there might exist other parking spots without a QR code. Thus, there might be more than two parking lines in the camera's field of view. The two relevant parking lines enclosing the QR code, therefore, need to be determined among all the parking lines to be able to create the extra obstacle line. Since each line is represented by two endpoints, the endpoint that is the closest to the QR code needs to be determined. From Figure 3.15, it is possible to see that the point closest to the QR code and furthest away from the camera for each line is the one with the minimum y-value of the two points if we define the origin of the axis to be from the top-left. These points are marked with yellow circles in the figure.

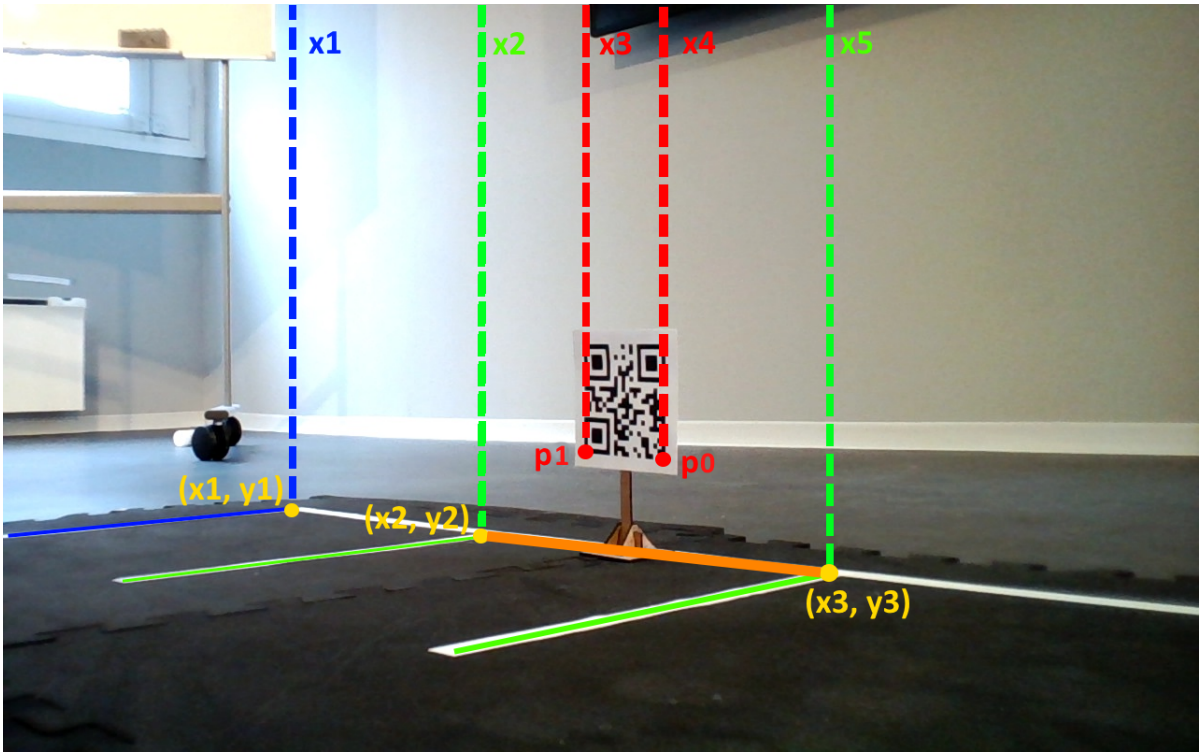


Figure 3.15: The idea behind determining the relevant lines of a parking line. The green lines are the relevant lines of the free parking slot.

Based on $p0$ and $p1$ from Figure 3.8, the x-value of these two points can be used to compare with the yellow points' x-values to determine the left and right parking line. The following equation can be used to calculate a difference value that can be used for this comparison where X will be the x-value of one of the yellow points and P will be the x-value of either $p0$ or $p1$.

$$diff = X - P \tag{3.6}$$

Furthermore, from Figure 3.15, it is possible to perceive that point $(x2, y2)$, which represents the left parking line, is the closest point to $p0$'s of the lines on the left side of $p0$. Similarly, it is possible to see that point $(x3, y3)$ which represents the right parking line, is the closest point to $p1$ which is on the right side of $p1$. Based on the equation, the previously mentioned observations and the fact that the x-axis in OpenCV's image coordinate system starts from the left of the image, it is possible to conclude this for $p0$. The closest point to $p0$ will have the minimum difference value that is positive. Whereas the closest point for $p1$ will have a minimum difference that is negative. Thus, these two closest points that are determined by the two previously mentioned criteria represent the two relevant lines enclosing the QR code.

It is important to note that this method may not give accurate results if there exists noise or irrelevant lines with endpoints that are closer to the QR code than the parking lines. This will cause the algorithm to determine the incorrect lines as the relevant lines. In this project, the parameters of the line detection method given in Section 3.2.5 have been used to reduce the amount of noise and irrelevant lines as a resolution for the previously mentioned problem.

Implementation

A class **ParkingSlotDetection** has been made as an implementation in this regard. The methods that are included in the class are listed and described in the following list:

- **get_closest_line**: Retrieves the two lines that represent the parking lines enclosing the QR code.
- **get_region_of_interest**: Gets the region of interest using the QR Code Detection module and Numpy's bitwise AND operation.
- **get_parking_lines**: Retrieves all the lines in the image in the region of interest and merges them together if they are close enough by using the class `MergeLines` that is implemented based on the theory from Section 2.7.
- **get_parking_slot**: Combines the methods above and the QR Code Detection module to retrieve the two lines closest to the QR code. In addition, it also returns all the parking lines.

3.3 Environment Mapping

One of the main challenges with autonomous driving is the problem of determining the actions needed to take to traverse to the destination point with regard to the vehicle's environment. A way to cope with this is the use of live measurements. For instance, in centering the vehicle within a road, a regulator based on the difference from the center can be used to correct its position. Another method, which is the approach used in this project, is to map the three-dimensional environment view that the vehicle's camera has, into a two-dimensional environment map with an aerial perspective. By using this approach, the two-dimensional environment map and a desired destination point are used to determine a path that the vehicle should follow to arrive at the destination. This method takes into consideration the challenges associated with the vehicle's field of view. For example, there are cases where the vehicle is not able to retrieve useful information from its camera, e.g., only one lane line is detected when trying to center the vehicle on the road as shown in Figure 3.16. With the use of path planning based on the environment map, the next several decisions are calculated well before the execution. Thus, in cases where no useful information can be retrieved, it can still continue forward based on the already calculated path from path planning, meaning it is not always dependent on live measurements, unlike regulators. Generally, environment map are used to continuously map new detected objects to the current map while still preserving the previously mapped objects which are no longer visible for the vehicle. However, as the project is an introduction to autonomous driving, a simpler approach of utilizing the map has been implemented by limiting the environment map to be reset on every new view as this is sufficient enough to perform path planning in the project's use case.

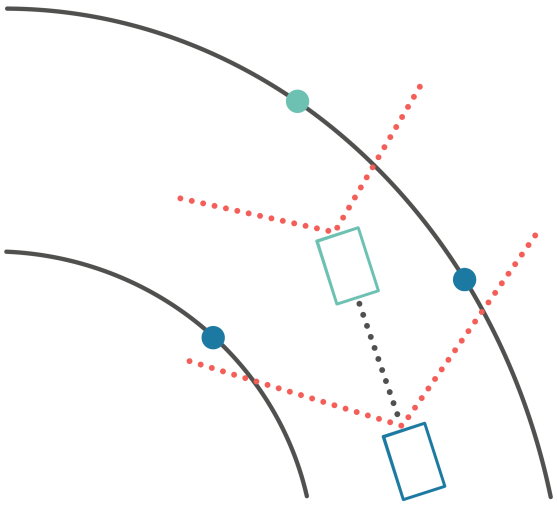


Figure 3.16: The blue vehicle in the figure is equipped to detect both lines and adjust itself accordingly. However, in the green vehicle's position, the camera can only detect one line, leaving it unable to align itself from the real-time information.

A $m \times n$ matrix can be used as a two-dimensional environment map for the vehicle's surroundings with an aerial perspective. Each index in the matrix corresponds to a desired distance in the x and y-axis, together defining the vehicle's environment as shown in Figure 3.17. The desired distance is used to adjust the accuracy by minimizing or enlarging the map. This is useful because a sub-pixel accuracy can be computationally demanding for some systems and therefore a smaller map can be used to reduce the needed computational process.

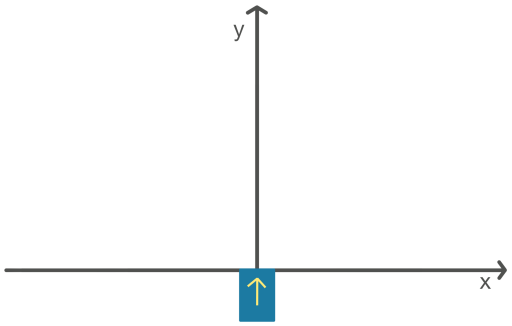


Figure 3.17: Coordinate system defining the distance to the different objects with the vehicle as the origin.

To map an object, the distance to the object in the x and y-axis needs to be converted into two index numbers, one for the row and one for the column. The equations for this conversion are given as follows:

$$\begin{aligned} r &= c_y - \lfloor d_y/s \rfloor \\ c &= c_x - \lfloor d_x/s \rfloor \end{aligned} \tag{3.7}$$

Equation description

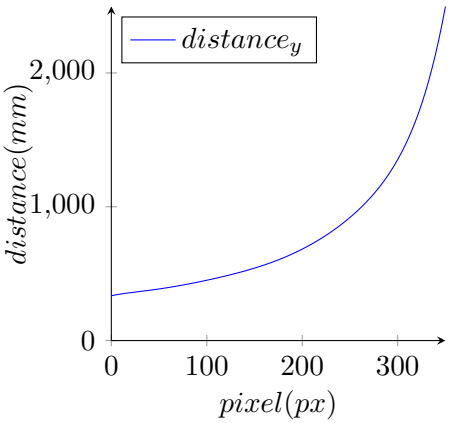
- r: The object's row index.
- c: The object's column index.
- c_x: The column index of the vehicle, equals to value $\lfloor n/2 \rfloor$.
- c_y: The row index of the vehicle equals to value m.
- d_x: The distance in the x-axis to the object.
- d_y: The distance in the y-axis to the object.
- s: The distance of each index in the matrix.

When mapping the stop signs and QR codes, as the size of both types is known, it is possible to calculate the distances from the vehicle to the sign as mentioned in Section 3.2.3. For parking lines and lane lines, the size of the lines can vary. Therefore, it is not possible to use the same method as the signs. The method of stereoscopic vision, which has been mentioned in Section 2.2 as a method to determine distances to objects, is also not suitable for detecting the distance to lines as lines are objects without texturally rich features. Thus, with the limitation of flat landform environments, to calculate the distances for the lines, two mathematical functions is used.

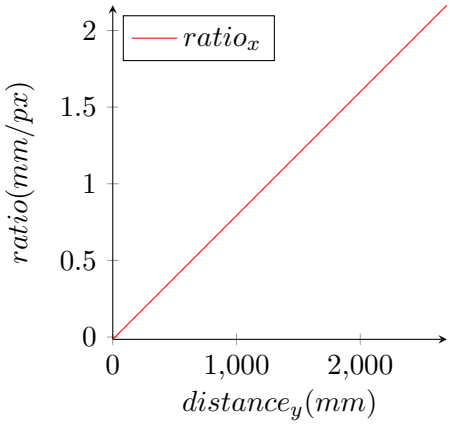
The first mathematical function describes the correlation between the distance in the y-axis and the pixel position in the vertical direction. The second mathematical function describes the correlation between the ratio in the horizontal direction which corresponds to the amount of distance in mm per pixel and the distance in the y-axis which is found by the previous mathematical function. If the ratio is constant in all places in the camera view, it is possible to use Equation 3.8 where $offset_x$ corresponds to the number of pixels between the camera center and a particular pixel. However, if the ratio is not constant in all places in the camera view, then a more complex function is needed to resolve the ratio at different distances on the y-axis.

$$d_x = \text{offset}_x * \text{ratio} \tag{3.8}$$

The test that was performed to determine these two mathematical functions is given in Appendix I and is based on regression analysis. The result of the test is then given in Figure 3.18. The mathematical function for the correlation between the distance in the y-axis and the pixel values in the vertical direction is presented in Figure 3.18a and is a seventh-degree polynomial. It is important to note that the function is only fitting at pixels height less than 350. It, therefore, has its own limitations which need to be cautioned of when used. On the other hand, the ratio for the x-axis as seen in Figure 3.18b is linearly dependent on the distance from the object to the vehicle in the y-direction and not constant. This is due to the fact that, the further away on the y-axis, the wider is the camera's field of view. It is also important to be aware that these two mathematical functions are limited to converting objects' ground positions and cannot be used to convert pixel points that correspond to positions above the ground.



(a) Regression model of the pixel values in the vertical direction against the distance in the y-axis.



(b) Regression model of the ratio in the x-axis which corresponds to the amount of distance in mm per pixel and the distance in the y-axis

Figure 3.18: Mathematical functions determined by the test given in Appendix I through regression analysis.

To use these two mathematical functions to calculate the distances in the y- and x-direction, first, the equations from Equation 3.9 are used to determine the difference in pixel position from the bottom-mid of the camera frame by using the camera image width and height.

$$\begin{aligned} \text{offset}_y &= \text{height} - \text{point}_x \\ \text{offset}_x &= \text{point}_y - \text{width}/2 \end{aligned} \tag{3.9}$$

offset_y should then be used as the parameter value in the mathematical function that corresponds to the correlation between the mm distance in the y-axis and the pixel position, which results in a calculated distance in the y-direction. Afterward, offset_y should be used as the parameter value in the mathematical function that corresponds to the correlation between mm per pixel horizontal ratio depending on the y-distance. This results in the ratio in the horizontal direction. The ratio should then be multiplied with the offset_x, resulting in a calculated distance in the x-direction.

Additionally, when adding a line to the two-dimensional map, only the two endpoints representing the lines are known. To map the entire line in the two-dimensional environment, Bresenham's line algorithm, which is presented in Section 2.8, can be used to achieve this.

A class **Environment** has been made as an implementation for the two-dimensional environment map.

- **reset**: Resets the matrix, only keeping the start node that represents the vehicle.
- **get_data**: Returns the environment matrix as a copy.
- **get_pos**: Retrieves the row and column indexes of an object with a given object ID.
- **point_to_distance**: Converts a point on a frame into distances in both the x and y-axis using the mathematical functions presented in Figure 3.18.

- **insert**: Inserts an object with a desired ID corresponding to the object type using distances in the x and y-axis respective to the vehicle.
- **insert_by_index**: Inserts an object with a desired ID corresponding to the object type using row and column indexes.
- **insert_objects**: Inserts a list of objects into the environment either if it is by distance or by indexing and uses the Bresenham algorithm if the object is a line.
- **remove**: Removes an object with a given ID corresponding to the object type from the environment.

Implementation Note

insert_objects method of **Pathfinding** class uses a Python package called **bresenham** for Bresenham's line algorithm.

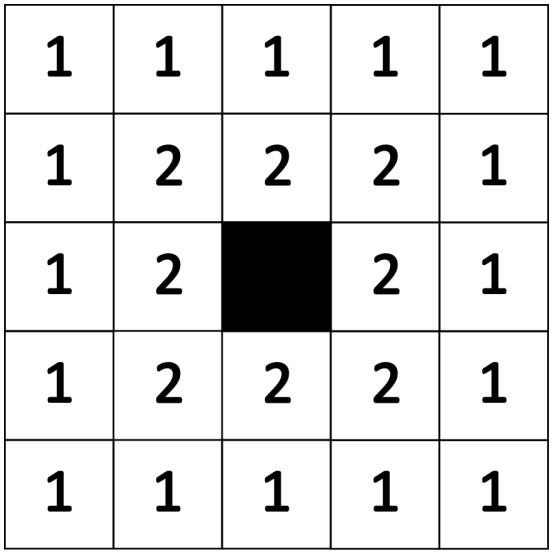
3.4 Path Planning

By using the information from the two-dimensional environment map in the previous section, it is possible to plan a path to reach a desired destination within the map. Thus, this section will be about developing a modified version of the A* algorithm to find an appropriate shortest path from the vehicle to the desired destination. Furthermore, the section will present the usage of the Catmull-Rom spline with the A* algorithm to achieve a smooth and continuous path.

3.4.1 Modified A* algorithm

When obstacles are present between the start and end points, the path will navigate around them with the constraint that a hindrance is not a valid node to traverse. However, as Figure 2.4a shows, the path may not be suitable for a vehicle due to sharp turns around hindrances. Therefore, a modification is made to the A* algorithm given in Section 2.9 to resolve this problem.

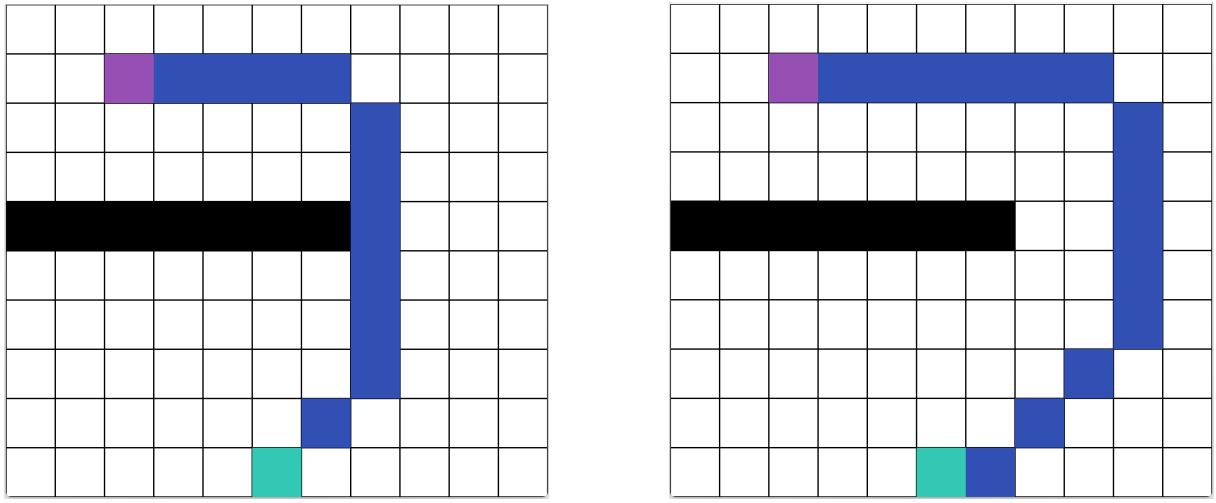
To achieve an algorithm that refrains from creating a path that closely follows the hindrances, the algorithm can be adjusted to assign a larger f value (referred to as the weight) to surround nodes in proportion to their distance from the obstacle. Additionally, a penalty can be applied to further force the path away from the obstacle. Figure 3.19 shows how the surrounding nodes will be affected. By applying weights and penalties for the hindrances, the path will have a smoother way around the corners as shown in Figure 3.20. It is possible to argue that a margin of safety would be a better solution for this scenario which will ensure that the car never touches the hindrances. However, because the hindrances in this project objective are limited to lines, this will ensure a smooth path where the consequences of driving too close to a line are none.



(a) Weights around an obstacle.

(b) Weight around an obstacle with a penalty of 4.

Figure 3.19: Showcase the weights from a hindrance.



(a) Pathfinding with a weight of 2.

(b) Pathfinding with a weight of 2 and a penalty of 4.

Figure 3.20: Pathfinding with weights.

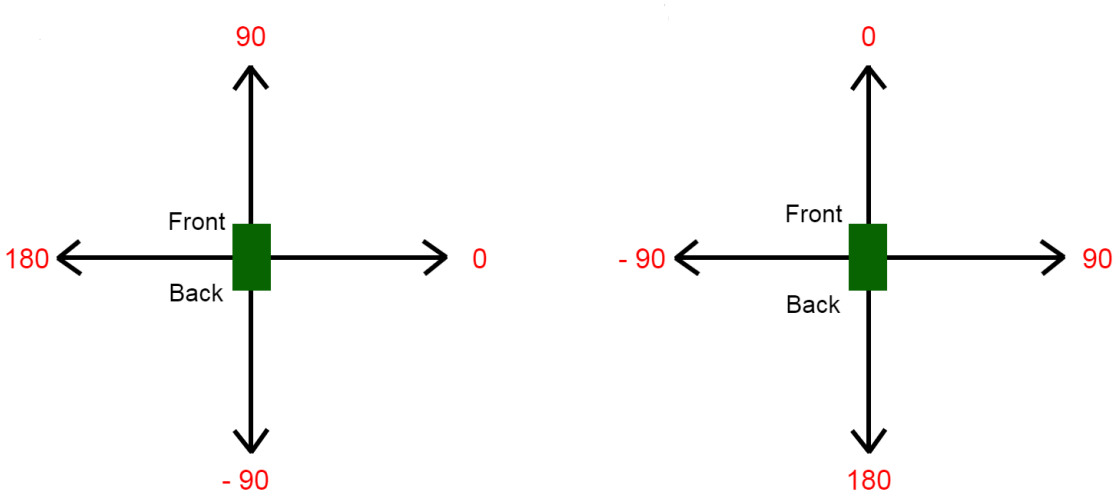
3.4.2 Catmull-Rom Spline

The obtained path from the modified A* algorithm is then used to calculate the Catmull-Rom spline presented in Section 2.10 to achieve a continuous path with a smooth trajectory for later use in motion control. The equations in the previously mentioned section are used to calculate the spline and the obtained path from the A* star are used as input points to create the curve segments.

Moreover, for each pair of two points given by the spline, the length between the two points is calculated. This information is for the vehicle to know how far to move from the first point to

arrive at the next point. On the other hand, for the direction it needs to face to move to the next point, a new polar coordinate system is defined for easier usage of the information obtained from the spline in motion control.

Given that the camera is located at the front of the vehicle, the vectors with the same direction as the camera view should have an angle of 0 degrees. To differentiate between the left and right sides of the 0 viewpoints, the clockwise side of 0 will be defined as positive and the counterclockwise side of 0 as negative. The value range is, therefore, $(-180, 180]$, where the 0 angle is on the positive side of the y-axis. The previous and the new polar coordinate system is presented in Figure 3.21.



(a) Original **(b)** Converted

Figure 3.21: Two different 0 starts on the polar coordinate systems. The green box represents the vehicle.

If the angle at a point is given directly to the motion control, then it needs to determine how much it to turn based on the previous direction to arrive at the new direction. This implies that the previous angle needs to be saved. To avoid doing this in motion control, this module will give the angle difference between the current point and the next point based on the new polar coordinate system definition. Hence, a negative angle difference corresponds to a left turn and a positive angle difference corresponds to a right turn.

To calculate this angle difference, for each point, the direction vector to the next point on the spline is calculated. The vector obtained enables the calculation of the direction vector's angle based on the initial orientation of the vehicle's camera. The direction vector has two components, v_x along the x-axis and v_y along the y-axis. The adjacent cathetus is represented by v_x, while v_y represents the opposite cathetus. To convert the angle of the direction vector to the new angle definition, the initial step is to employ the values of the components acquired through the inverse tangent based on the tangent definition. The result of the previous action is thereafter evaluated in Equation 3.10, where θ is the result of the tangent inverse and θ' is the desired direction in the new coordinate system. The equation takes into consideration that the points given from the spline are inverted on the y-axis. This is due to the matrix the spline is based on having the first row in the matrix as the furthest away from the vehicle whereas the last row is where the vehicle is as mentioned in Section 3.3. Finally, after the angles of all the direction

vectors have been calculated, the angle differences are calculated sequentially.

$$\theta' = \begin{cases} 90 + \theta, & \text{if } x > 0 \\ 90 - \theta, & \text{otherwise} \end{cases} \tag{3.10}$$

3.4.3 Implementation

A class **AStar** has been made as an implementation of the modified A* algorithm with the following methods:

- **find_path**: Calculates the path from an object to another object based on two input object IDs.
- **create_weight_matrix**: Creates a weighted matrix based on the current environment.

Furthermore, a class **Pathfinding** has been made as an implementation for path planning with the methods presented below:

- **catmull_rom_segment**: Calculates the points of the segment made by four input points using Equation 2.9.
- **catmull_rom_spline**: Creates a segment using `catmull_rom_segment` for each overlapping sequence of four points.
- **catmull_t_j**: Calculates the t_1, t_2 and t_3 values for the Catmull-Rom spline using Equation 2.10.
- **approx_segment_lengths**: Approximates the length of a segment between two points.
- **get_angle**: Calculates the angle based on the new polar coordinate system given by Equation 3.10.
- **get_angle_diff**: Calculates the sequential angle differences from a list of angles.
- **calculate_path**: Uses the `AStar` class to calculate the path from one object to another. Thereafter, it uses the `catmull_rom_spline` method to calculate the continuous path and then uses this path to obtain the angle changes and distances to move in each direction for motion control usage.
- **merge_similar_angles**: Merges similar angles to reduce redundancy in the list of angles and times.

3.5 Motion Control

The desktop test vehicle model presented in Section 3.1.2 employs for each wheel a boolean value to assert which direction the wheel should turn (DIR pin of the stepper driver), and pulses to

step the motor (PWM with 50 % duty cycle on the STEP pin). Thus, a simple motion model is constructed with respect to the vehicle's approach to controlling the vehicle's wheels and the information obtained from the Path Planning module that consists of an angle change and a distance that the vehicle should move forward in the new direction.

The motion model is divided into two steps. The first step is to change the direction the vehicle is facing by turning the vehicle. The proposed solution is to let one wheel turn faster. The wheel which is going to turn is decided by the angle given by the Path Planning module. Based on Figure 3.21b, if the angle is positive, then the left wheel should turn faster, and otherwise, the right wheel should turn faster. The second step is to drive forward at a constant speed in the new direction. To achieve this, both wheels should turn at the same speed. Figure 3.22 visualizes this motion control approach.

(a) **(b)**

Figure 3.22: The concept of the motion control model. The black dashed line represents the previous direction. The red dashed line describes the step of changing direction. The blue dashed line represents the step of keeping constant speed in the new direction. $\triangle d$ illustrates the distance in the y-axis before the new desired direction is achieved and r represents the turning radius. (a) and (b) show the relation between $\triangle d$ and r.

$\triangle d$ in the motion model is affected by the different speeds of the wheels. Less difference in speed leads to a smaller $\triangle d$ value because the turning radius is smaller. Similarly, a large difference will lead to a larger $\triangle d$ value because the turning radius is larger. Thus, a large $\triangle d$ can lead to a large deviation from the destination position. On the other hand, a large difference in speed can cause unstable driving. A value for the difference in speed which compromises between these two problems is therefore necessary. Moreover, in the first step, by defining the difference in the wheels' speed as a constant value, the turning radius will also be a constant value as a consequence. The different orientations θ for the vehicle can then be achieved by calculating the distance the vehicle needs to travel on the basis of the turn radius. The following equation is used to calculate this distance.

$$d = 2\pi r * \frac{\theta}{360} \tag{3.11}$$

The velocity of the outer wheel during turning is calculated using the PWM value of the outer wheel. The value is then used to determine the duration the vehicle needs to hold this motor

power value to travel the previously calculated distance. The duration is achieved by dividing d by the velocity of the outer wheel.

The small car test vehicle model presented in Section 3.1.2 employs a boolean value to assert which direction the drive wheel should turn (DIR pin of the stepper driver), and pulses to control a servo motor connected to the front wheels to adjust the direction. Only motion control for straight driving was implemented for the small car test vehicle. Straight driving was implemented in order to test the function of detecting stop signs and reacting to them. However, due to prioritizing tasks with LEAN methodology and the lack of a need for turning the small car test vehicle, that functionality was not implemented.

Motion control can be tested separately and manually using a gamepad with the test vehicle in the MANUAL state to see if it turns as desired.

3.6 User Interface Design

The user interface design of the development GUI software is presented in Figure 3.23.

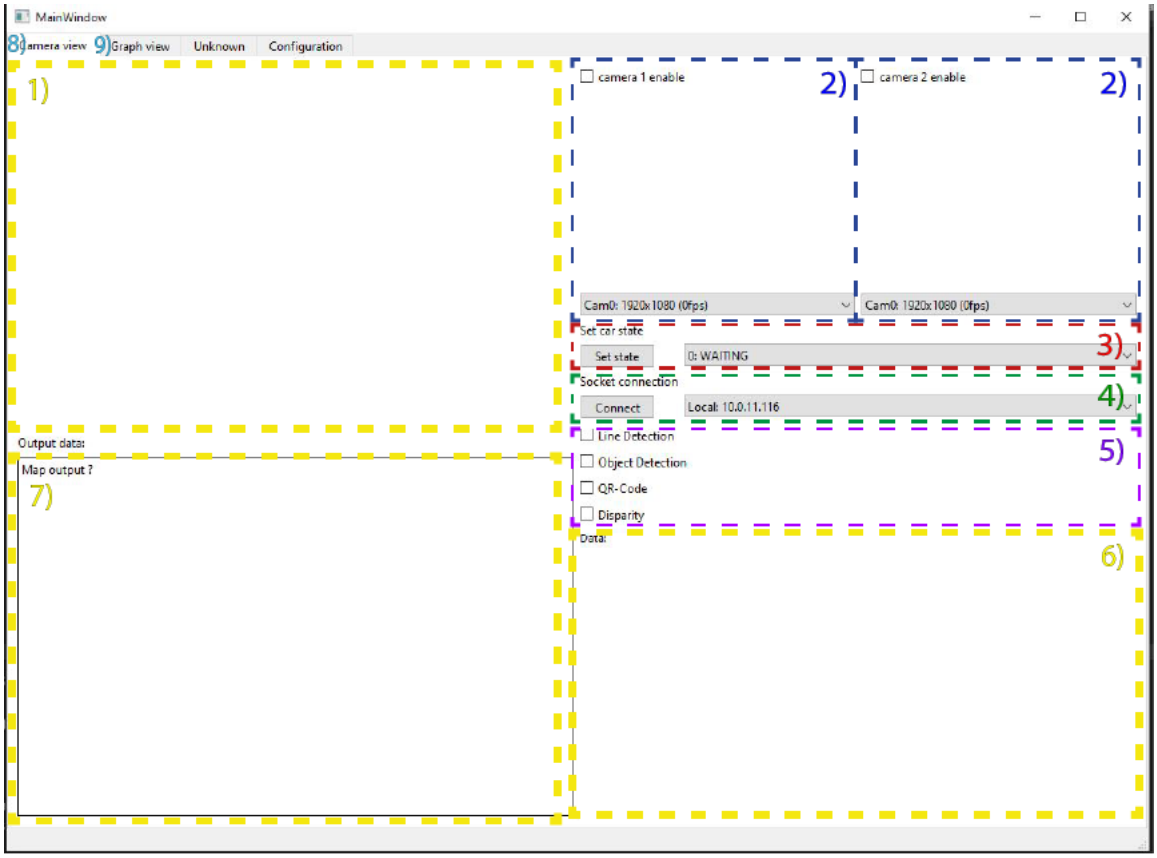


Figure 3.23: The user interface design of the Qt-based development GUI software. The different colors and labels describe the different frames in the design.

A description of the design is given below:

1. (Yellow) Contains image output area with relevant drawings from selected computer vision modules.
2. (Blue) Contains check boxes, camera frame output, and camera selection drop-down list (Combo box/QComboBox). The drop-down list includes the available cameras and network socket camera selection (from an array in the configuration file). By marking the checkbox the selected camera from the drop-down list is enabled. Frames from Camera 1 are used for the computer vision modules, and frames from Camera 2 are only used for stereoscopic vision.
3. (Red) Contains a drop-down list (combo box/QComboBox) to select the desired driving state and a button to send the chosen state to the vehicle. See Section 3.1.4 for information about the available states.
4. (Green) Contains a drop-down list (combo box/QComboBox) to select a socket and a button to connect or disconnect the socket connection. A socket connection is needed for the GUI to send/receive data from the vehicle control software.
5. (Purple) Contains checkboxes to select the computer vision algorithms to run on the received camera frames. Output data is shown in the output sections.
6. (Yellow) The region with the *Data* label shows a text description of data from the enabled computer vision modules.
7. (Yellow) Presents an arrow indicating the direction for the next vehicle movement.
8. (Turquoise) Tab item for the currently shown page (Camera view).
9. (Turquoise) Tab for graphs.

3.7 Testing and Validation

To ensure the quality and functionality of the software implementation, a rigorous testing and validation process was conducted. The testing process was divided into several steps, including unit testing, integration testing, and system testing.

Unit tests were created to test small pieces of code in isolation. This enabled the identification and resolution of any bugs or errors early in the development process. Integration testing was also made, where the various components of the software were tested together to assess how they interacted with each other. For example, the parking line detection was tested with the QR code scanning feature to catch any issues that might arise when different modules were combined.

In addition to the code testing, a validation phase took place after each module was completed. This phase involved the modules being tested physically based on the objectives of the modules and ensuring that they performed as required. The results of these tests were documented in individual test reports to provide a record of the testing and validation process for future reference. For the different validation tests that were performed, a detailed test report is given in each of its own appendix chapters. The purposes of each test are as follows:

- **Functionality Testing of Stereoscopic Vision (Appendix B, C, D):** The purpose of the tests is to verify the implementation of stereoscopic vision works as intended and

determine how it gets affected by noise. First of all, a test was conducted to see how the lighting affected the calibration process. Thereafter, a test determining the parameter values takes place to get the best possible results. A final test was conducted, showing how blurring the images changed the results.

- **Functionality Testing of QR Code Detection (Appendix E):** The purpose of the test is to verify the ability to detect QR codes by using the QR Code Detection module. Furthermore, it is also to determine the detectable distances and angles. Finally, it also tries to determine the minimum size the QR code needs to be so that it is detectable at certain distances.
- **Functionality Testing of Stop Sign Detection (Appendix F):** The purpose of the test is to verify the ability to detect stop signs by using the Stop Sign Detection module. Another objective is to determine the relevant range of distance and angle when using the module. As the module allows one to choose the minimum size of detection based on the size of the pixels on the camera frame, different values of this minimum size may affect the relevant range of distance and angle.
- **Functionality Testing of Lane Detection (Appendix G):** The purpose of the test is to verify that the Lane Detection module works as intended based on the functions: detecting lane lines, calculating the distance between the center of the vehicle and the center of the lane, and providing a checkpoint in the center of the lane. Another purpose is to determine the uncertainty in the calculation of the center difference and the checkpoint.
- **Functionality testing of Parking Slot Detection (Appendix H):** The purpose of the test is to verify the Parking Slot Detection module's methods work as intended based on the function: detecting and determining the parking lines that correspond to an empty slot when a QR code is detected. The test disregards the distance to the QR code, but rather focuses on the different angles as this will be tested in the QR code test.
- **Determining Mathematical functions to Map Lines into a 2D Environment (Appendix J)**: The purpose of this test is to determine the mathematical functions for the method `point_to_distance` of the Environment Mapping module and the accuracy of the mathematical functions when used in the method `point_to_distance`.
- **Functionality Testing of Mapping QR Code with Lines (Appendix J)**: The purpose of this test is to determine how the mapping of QR code and the mapping of lines operate together for parking path planning.

System integration testing was the final step in the testing process. When performing system integration testing the software as a complete system was tested to see how well it performs measured towards the four functional objectives described in Section 1.4. This process is used to quantify the reliability of the software with the developed algorithms when implemented on a vehicle.

Chapter 4

Results and Analysis

4.1 Evaluation of System Performance

This section will evaluate the system's performance. This includes evaluating the accuracy of the computer vision techniques, the system's reliability, and the system's speed measured in frames per second based on the evaluation criteria given in Section 1.4.

4.1.1 Accuracy of Computer Vision Techniques

The different modules will be presented separately based on the presented validation tests in Section 3.7 to identify the accuracy of the modules.

Stereoscopic Vision

For stereoscopic vision, tests were conducted to tune the module's parameters and to see how different light conditions can affect the result of detecting objects.

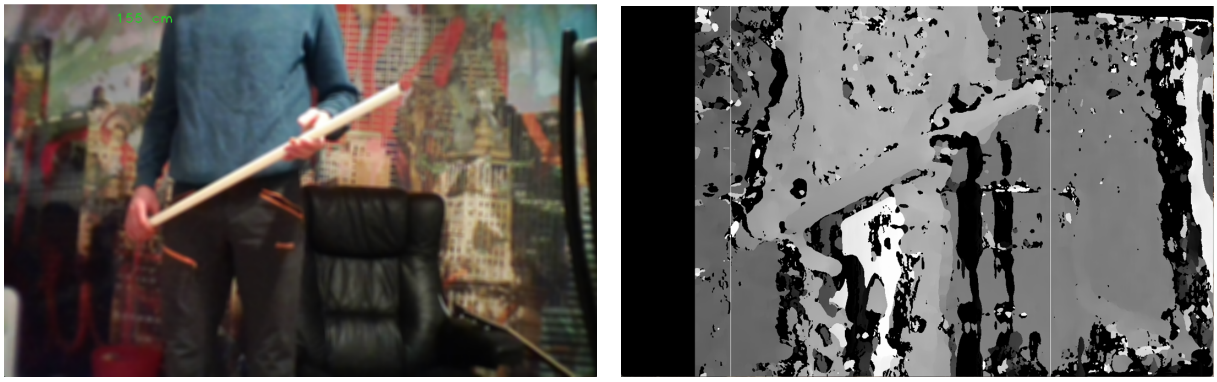
First, the calibration process is tested, which is given in Appendix B. It was done to see at which light condition the calibration would give the best results. Different light settings were tested such as only sunlight, only indoor light, and a combination of sun and indoor light. Furthermore, different settings of the standard parameters (number of disparities and block size) were tested such that the results were based on different adjustments. The results showed that an environment with no glare was preferred as the glare creates noise for the camera. That means it is beneficial to limit conditions causing glare such as reflective surfaces and direct sunlight.

The documentation from [26] and [49] indicated that there is a numerous amount of parameters that can be tuned in order to get a representative disparity map. However, the result from the test given in Appendix C showed that only adjusting the number of disparities and the block size gave a better result than adjusting every available parameter. A possible explanation for the

parameter adjustment limitation is that the given testing environment is not optimal as it still contains glare, thus also leading to difficulty in achieving good results with the other parameters in stereoscopic vision. For instance, the testing photos might not be ideal because of the lack of patterns in which stereo vision is not well suited as mentioned in Section 2.2. In addition to the external factors, there were a lot of parameters to tune, which makes this process complex. Thus, some parameters might not be set to a value which gives an ideal outcome. However, the results were clear enough to be able to arrive at a conclusion. It showed that the results when only adjusting the two parameters, the number of disparities and block size, create a more accurate and descriptive map compared to the result when adjusting every possible parameter.

Additionally, another test was conducted, establishing if blurring the image was beneficial to the result obtained in the test from Appendix D. The results from this test showed that a small amount of blur gave a smoother disparity map because it reduced the noise in the image.

Furthermore, with the use of the results from the tests, a depth map was created. One example result is illustrated in 4.1, showing the left stereo image and the resulting depth map.



(a) The blurred left image. The green text is the calculated distance to the object.

(b) The depth map. The white box, indicated by a thin white line, is the area that is detected as the object.

Figure 4.1: The result of using the disparity map to determine the distance to an object. The measured distance to the object was approximately 145cm.

Figure 4.1a is a blurred image from the left camera with the green text displaying the measured distance at 155cm. The real distance in this scenario is about 145cm. However, when standing still, the measured distance would vary by around ± 15cm. That means the distance will deviate by about 17% in the worst-case scenario. Nevertheless, as this method is not used as an accurate object detector and is restricted to detecting objects between a threshold, this result is acceptable.

Although the distance accuracy is acceptable in this scenario, this is not always the case. The disparity map is presented in Figure 4.1b, where a thin white line is drawn around the detected object. In this frame, the disparity map detects the correct object and shows a high level of accuracy where it can distinguish the person and the pipe from the back wall. However, for other frames, it can have problems detecting the correct object, especially at a close distance. Moreover, one of the most outstanding challenges was that the program detected an obstacle, even if there were none. This was possibly due to the fact that the parameter tuning might not be perfectly optimized. In addition, the blurring effect reduced the distinction in the image which can cause part of the image mistaken for an object. Another factor would be the testing environment as this was not representative of a driving vehicle in every scenario, thus, it might

benefit from a more open environment with fewer objects.

Despite the challenges, the module shows a result, that with further development can be a good tool regarding object detection. From these tests, it is possible to perceive that stereoscopic vision is a vulnerable technology that needs proper calibration processes based on different light conditions. In addition, the parameters might be difficult to tune as it is a complicated task, and only using the standard parameters will give a better result for this application. Even if the measured distance is acceptable, the inconsistent disparity map restricts its reliability.

QR Code Detection

For QR code detection, a test given in Appendix E was conducted to verify the accuracy of the module's calculations and detection. An example photo of a detected QR code is shown in Figure 4.2.

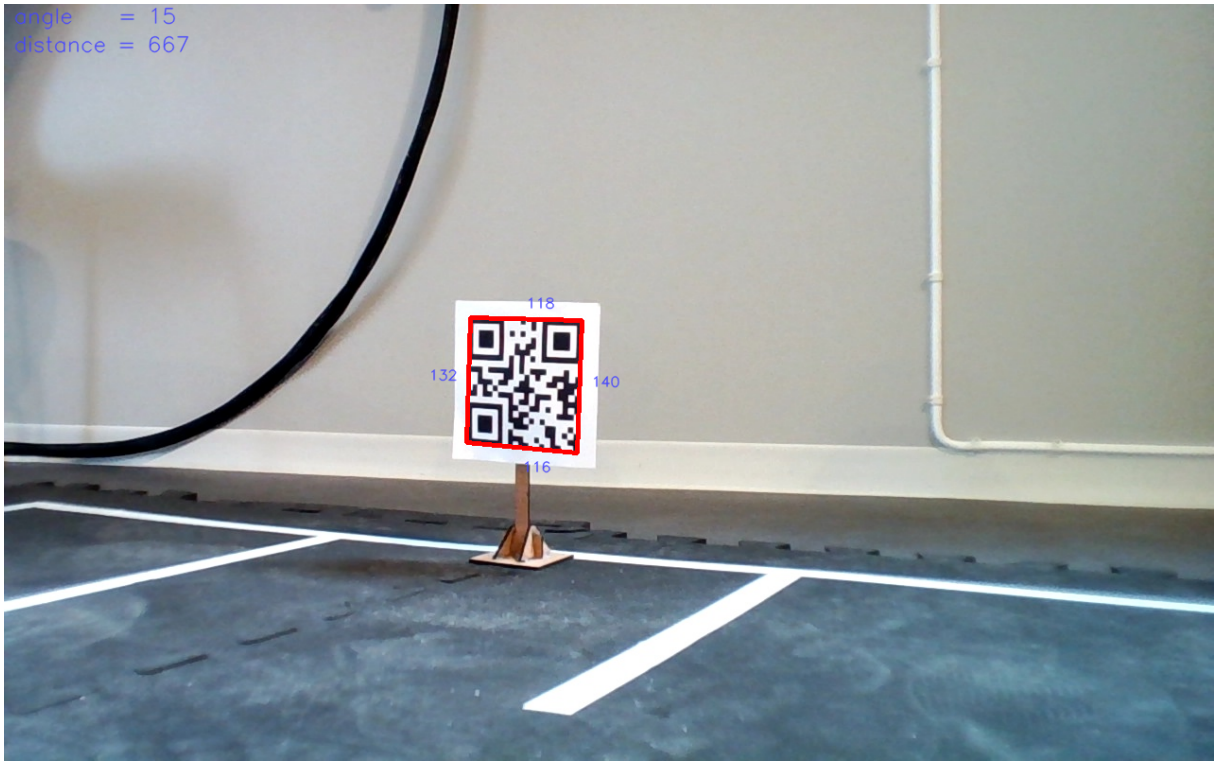


Figure 4.2: The detected QR code. The number around the QR code is the number of pixels on the side. The numbers in the top left display the values of angle and distance.

The result of the test showed that the calculation of the distance to the QR code sign was quite accurate with deviations of a maximum of 8% from the actual measurements. These deviations were possibly influenced by the accurate measurements of the size of the QR code and its distance to the camera. Thus, the mapping of the QR code into the environment map will have high accuracy.

Furthermore, the test results indicated that the range of distance in which the module is able to detect the QR code is dependent on the resolution of the camera and the size of the QR

code. With a higher-resolution camera, the module was able to detect the QR code at a further distance. Similarly, a larger size of the QR code gave a further maximum distance of detection. The correlation between the size of the QR code and the maximum distance the QR code can be detected is given in Figure 4.3. From the figure, it is possible to observe that the maximum distance that the QR code can be detected is a linear function of the size of the sign.

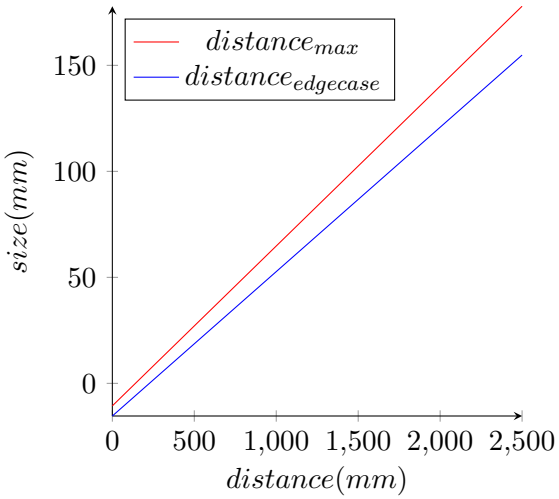


Figure 4.3: The result of the maximum distance the module is able to detect the QR code with different QR code sizes and a camera with the resolution 1280x800. The red line ($distance_{max}$) is the max distance at which the QR code is constantly detected. The blue line ($distance_{edgecase}$) is the max distance the QR code can be detected.

For the accuracy of detecting the QR code at different angles, the test result showed that this accuracy was not affected by distance, except when the QR code was placed at the maximum distance of detection. The general range of angles that the module was able to detect the QR code was $\pm 35°$ from the camera being perpendicular to the QR code.

Given the previously mentioned results with the camera the vehicle presented in Section 3.1.2 is equipped with, it required a QR code with the size of 65mm to be detected at a distance of 1000mm consistently. As the size of the QR code should not exceed the size of the parking space, a QR code size of about 100mm would be sufficient to have a detectable range above 1500mm for a vehicle of dimensions 228mm by 180mm in a parking lot of dimensions 345mm by 330mm.

Stop Sign Detection

The test given in Appendix F was performed to verify the functionality of the Stop Sign Detection module regarding the accuracy of detection and distance calculation. An example of the usage of the Stop Sign Detection module to detect a stop sign is presented in Figure 4.4

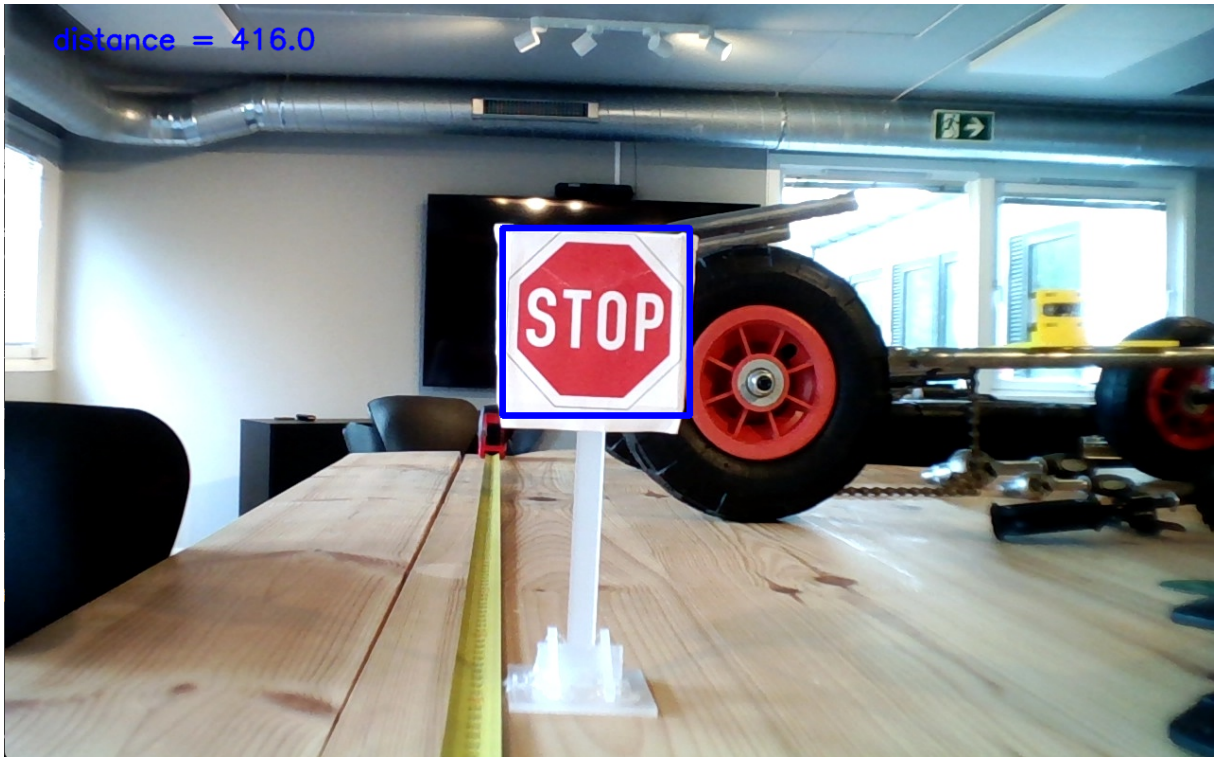
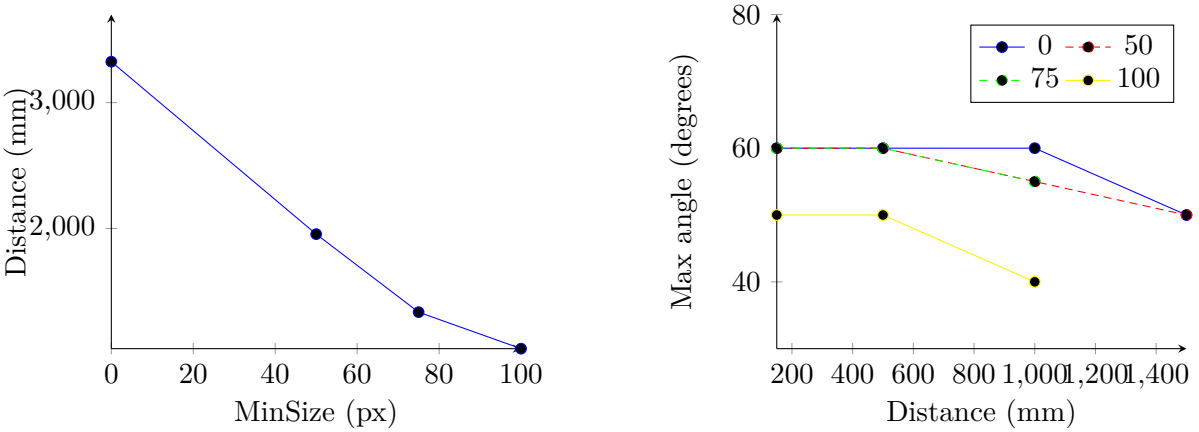


Figure 4.4: An example of the usage of the Stop Sign Detection module to detect a stop sign. The number in the top left corner displays the distance to the stop sign. The measured distance to the stop sign is 416mm.

First, the result from the test showed that the maximum distance the module was able to detect the stop sign was dependent on the parameter `minSize` in the step of Cascade of Classifiers.A large `minSize` value will cause the maximum distance the module is able to detect the sign to be smaller. This correlation is presented in Figure 4.5a.

Furthermore, Figure 4.5b presents the result of determining the detectable angles with different `minSize` and at different distances. The plot indicates that the relevant angles do not vary significantly but will decrease slowly when the distance is nearing the maximum possible distance of detection for the different minimum sizes. However, the result of `minSize` 100 is an exception as the possible angle to be detected deviates clearly from the other `minSize` values.



(a) Maximum detectable distance.

(b) Maximum detectable ± angle

Figure 4.5: Result of determining the maximum distance and the angles the module is able to detect the stop signs for different `minSize` values. The different colors in (b) correspond to different `minSize` values.

Although the furthest maximum distance is desired such that it can detect the signs as quickly as possible, the test result showed that the performance was affected by the `minSize`. In addition, it was observed that the processing frames per second decreased when the `minSize` increases. Therefore, the most suitable `minSize` should be a value that considers mainly FPS and maximum detectable distance. The detectable angles did not vary much, thus, it does not need to be focused on when deciding the `minSize` value. The `minSize` with the value 75 is therefore chosen for this module since it has a reasonable maximum distance of detection and FPS on the single board computer that the vehicle is equipped with.

Finally, it was observed that the module had a high accuracy in calculating the distance to the stop sign with a maximum of 10% deviation from the measured value. Thus, the module will have high accuracy when utilized in the driving logic based on having a distance that is set as a threshold for determining when the vehicle movement should stop. If the distance between the vehicles and the stop sign is less than this threshold, then the vehicle will stop moving.

Lane Detection

The functionality of the Lane Detection module was verified through the test given in Appendix G. The functionalities were tested to determine the accuracy of detecting the lane lines and calculations using the lane lines. An example of using the Lane Detection module is shown in Figure 4.6.

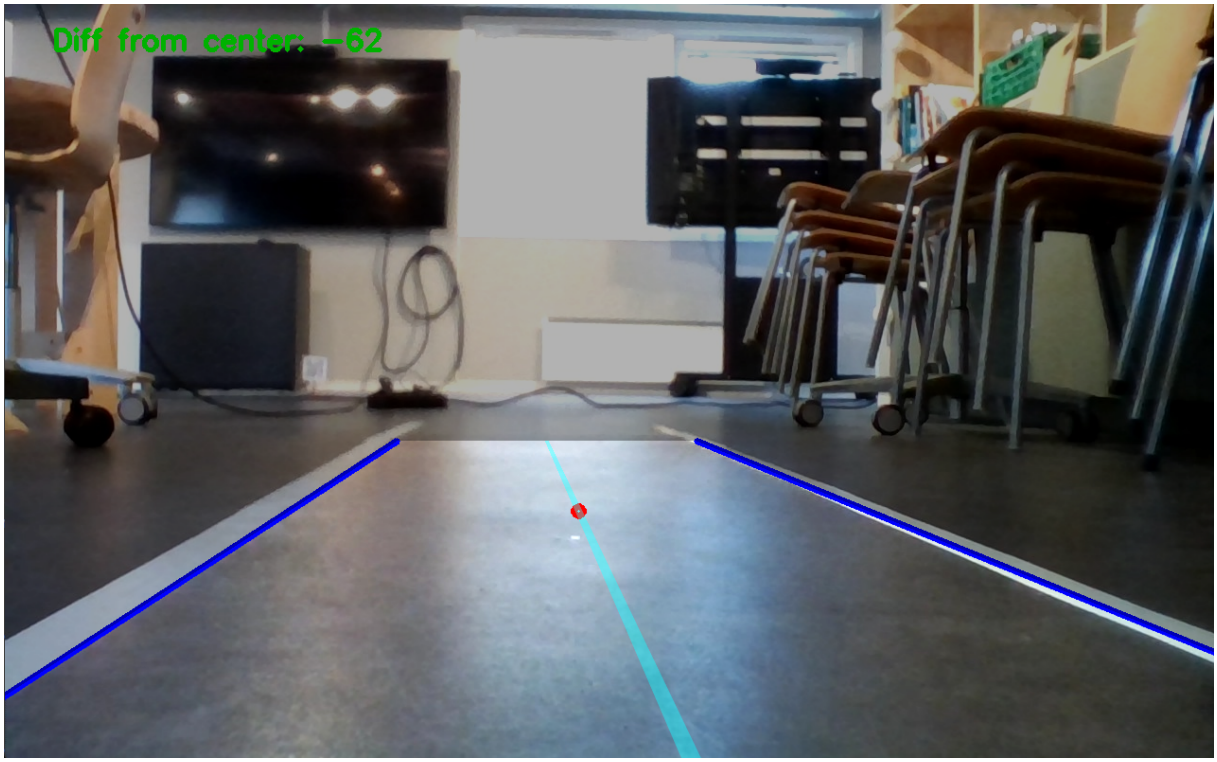


Figure 4.6: An example of the usage of the Lane Detection module. The number in the top left corner displays the values of the difference between the center of the lane and the center of the vehicle. The light blue line between the two lane lines defines the center of the lane. The red point presents a point on the center of the lane line which is further away from the vehicle. The lighter area in the image indicates the area in which perspective transformation has been performed.

The result of checking if both lane lines are detected at different distances from the left lane line is presented in Table 4.1. From the table, it is possible to perceive that a wide range of relevant distances can detect both lane lines. However, the maximum distance from the left lane line from the test was less than the desired range. The reason behind this deviation from the desired range was an error in the camera the vehicle was equipped with, which caused the camera's lens center to not be centered in the camera frame. This led to the left side being seen more than the right side of the lens center. Therefore when the right wheel was placed on the right line, the left line was out of the region of interest. Since the center of the lane will have a distance of 230.5mm from the left lane line, there exists a safety of margin with a value of 124.5mm, which is within the range that the vehicle would usually be when lane centering is activated. This error from the camera will therefore not affect the functionality of detecting lane lines, but it is important to be aware of so additional computations can be performed on other affected functionalities to take this issue into account.

Desired range		**Result range**	
Min	**Max**	**Min**	**Max**
52	409	52	355

Table 4.1: Result of checking if both lane lines are detected in a lane with a width of 461 mm and lane line width of 38 mm. `Min` and `Max` correspond respectively to the minimum and the maximum distance from the left lane line. The desired `Min` and `Max` also correspond respectively to the left wheel of the vehicle being on the left lane line and the right wheel of the vehicle being on the right lane line.

Furthermore, the test also includes the result of determining the deviation in calculating the distance between the center of the lane and the center of the vehicle compared to the measured distance. When the vehicle was on the left side of the center of the lane, the maximum deviation was approximately 12.5%. However, when the vehicle was on the right side of the center of the lane, the maximum deviation was approximately 42.0%, which is a quite large deviation percentage. Despite this large deviation, this deviation corresponds only to a value of difference that is less than 30mm which is only 6.5% of the lane width. A factor to the high deviation could have possibly been caused due to the width of the lane line, which is in this case 38mm. The measured center difference was measured from the inside of the lane lines. However, in some cases, the detected lane lines might be in between the outer and inner lane lines instead, which can give it an extra offset of up to 38 mm. Although the deviation in percentage is large, the calculated deviation in mm is not large compared to the lane width. In addition, as the method is only used to be displayed in the user graphical interface, it is still acceptable, but the deviation must be aware of when it is being used.

The deviation between the center of the lane and the point that is calculated to be at the center of the lane and at a further distance away from the vehicle is also given in the test. The deviation in all cases is less than 10%, thus, this calculation has high accuracy. It is therefore possible to conclude that when the two-lane lines are detected, the point given by the module is acceptable to utilize in the path planning algorithm to center the vehicle when driving within the road.

Parking Slot Detection

The test given in Appendix H was performed to verify the functionality of the Parking Slot Detection module. The accuracy that was tested in this module was the ability to detect the correct parking lines in order to create an extra obstacle line for further usage in the Path Planning module. An example of a result obtained by using the module is given in Figure 4.7.

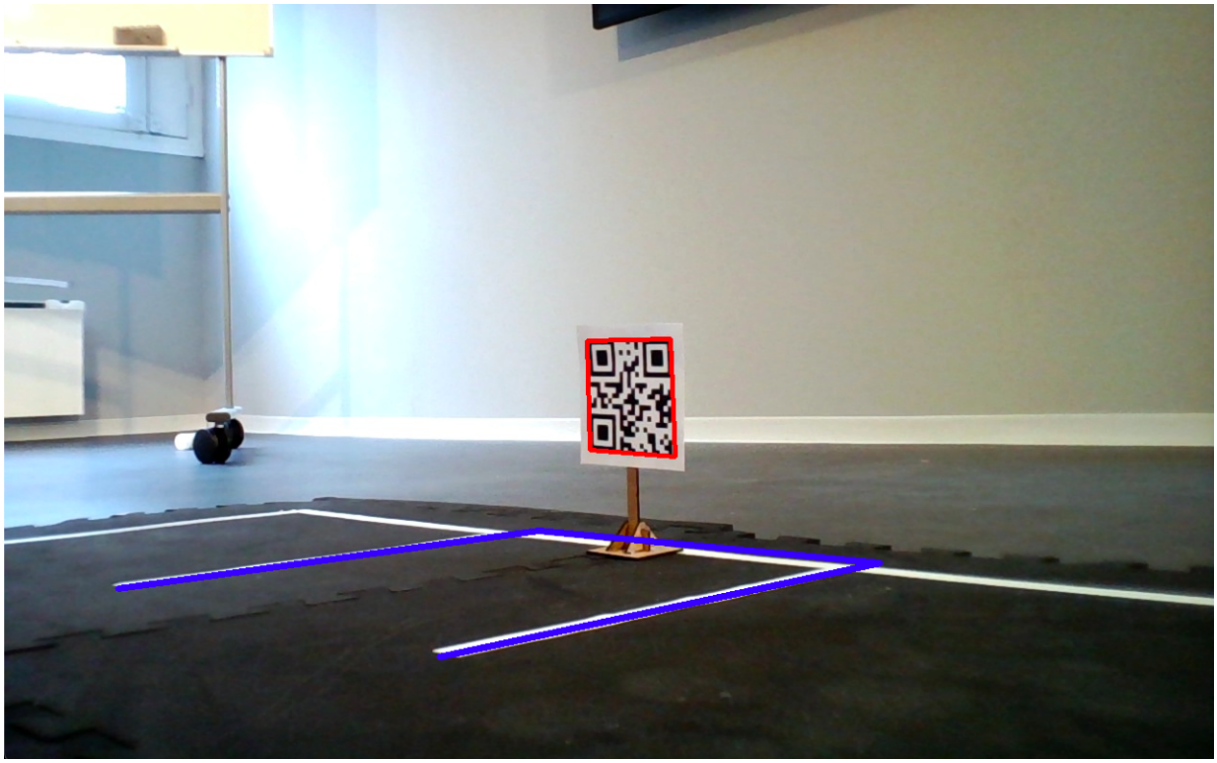


Figure 4.7: An example of the usage of the Parking Slot Detection module. The correct parking lines are detected and an extra obstacle line was created when a QR code is detected.

The result of the previously mentioned test indicated that within the relevant angles for detecting the QR code which defines the desired range in this scenario, it detected the parking lines enclosing the QR code robustly. However, it was possible to observe that noise lines could affect the result as mentioned in Section 3.2.7. It is, therefore, quite dependent on the environment the parking lines are placed in and the parameters being fine-tuned to be able to remove the noise lines.

4.1.2 Accuracy of Environmental Mapping

The test given in Appendix J was performed to verify the accuracy of the `point_to_distance` method of the Environment Mapping module.

The result of the accuracy test is given in Figure 4.8. From the previously mentioned figure, it is possible to perceive that the maximum deviation was around 15.4% for the y-distance and 12.3% for the x-distance. As the deviation is less than 20%, but larger than 10%, the deviation should be aware of when the method `point_to_distance` is used. On the other hand, only 20% of the points in the y-direction had a deviation larger than 10%. The result is similar for the deviation in the x-direction. It is therefore possible to affirm that the method has a moderate accuracy.

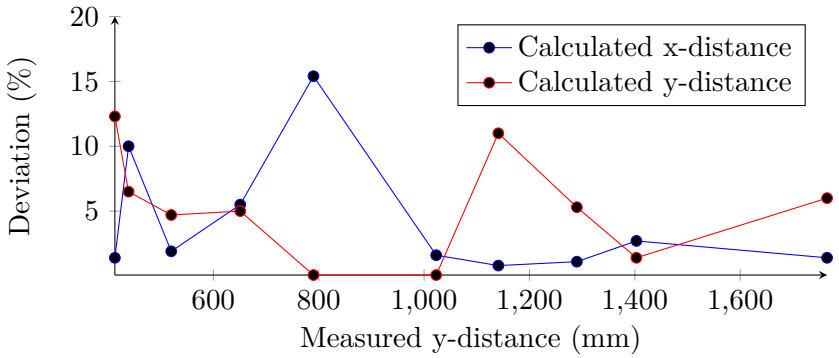
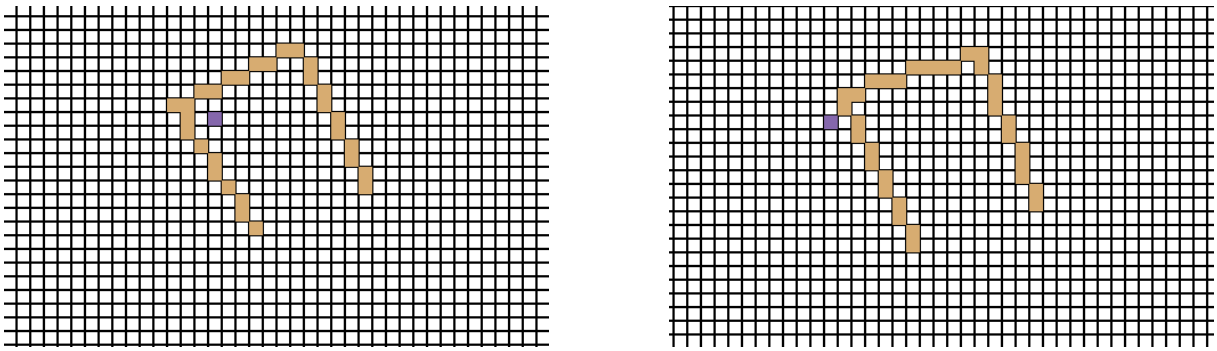


Figure 4.8: Result of comparing measured distances to distances calculated by the method `point_to_distance`.

Furthermore, the test given in Appendix J was performed to determine the alignment between the mapping of the QR code and the mapping of lines for parking scenarios. The result of the test indicated that the mapping was dependent on the accuracy of the QR code distance calculation and the accuracy of the `point_to_distance` method of the Environment Mapping module. Out of five test cases, four gave a result where the QR code was within the parking slot as given in Figure 4.9a. However, although the QR code was within the parking slot, the QR code in the mapping is not in the position in which the QR code is physically placed. For example in the previously mentioned figure, the QR code is closer to the left side of the parking slot. Furthermore, one of the test cases gave a result where the QR code was not within the parking slot as shown in Figure 4.9b. In addition, the test cases only included a few combinations of positions from the QR code. Thus, there might exist some other scenarios where the mapping of the QR code mapping and the parking lines do not align with each other for example causing the QR code to be behind the parking slot instead.



(a) QR code is within the parking slot.

(b) QR code is not within the parking slot.

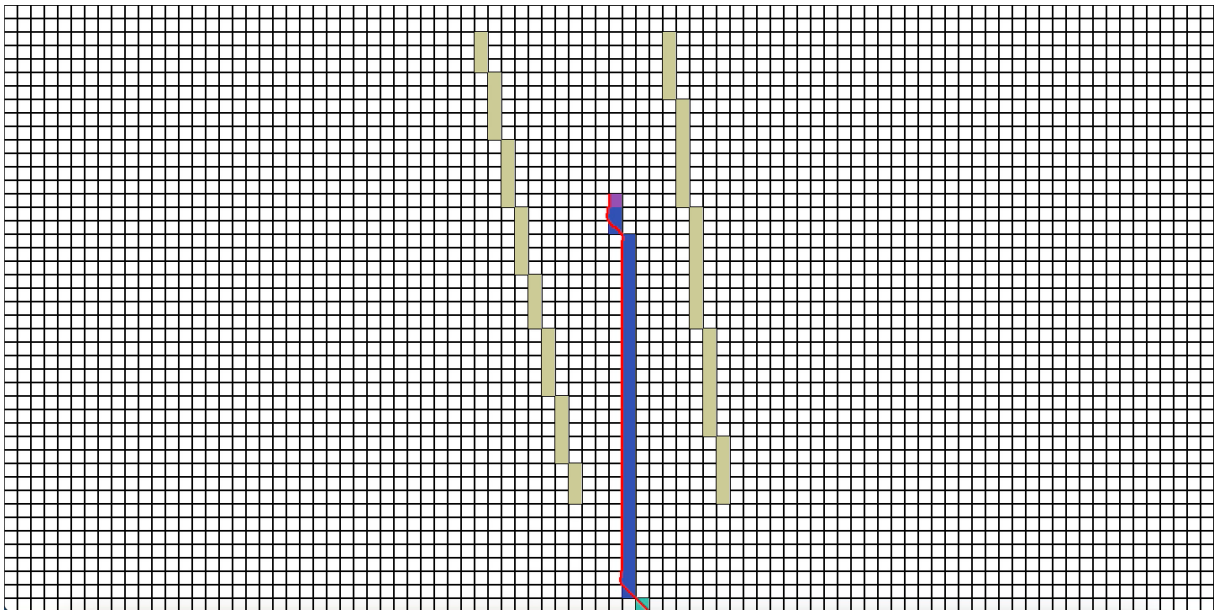
Figure 4.9: An example of the result from the test given in Appendix J. The yellow lines correspond to the parking lines and the purple square indicates the QR code.

4.1.3 Accuracy of Path Planning

As the Path Planning module builds upon the Mapping module and does not include any new retrieval of information from the camera frame, the accuracy of the module is based on the accuracy of the other modules and if the implementations of the algorithms are correct. Unit and integration tests have primarily been used to enforce high accuracy in the implementations of

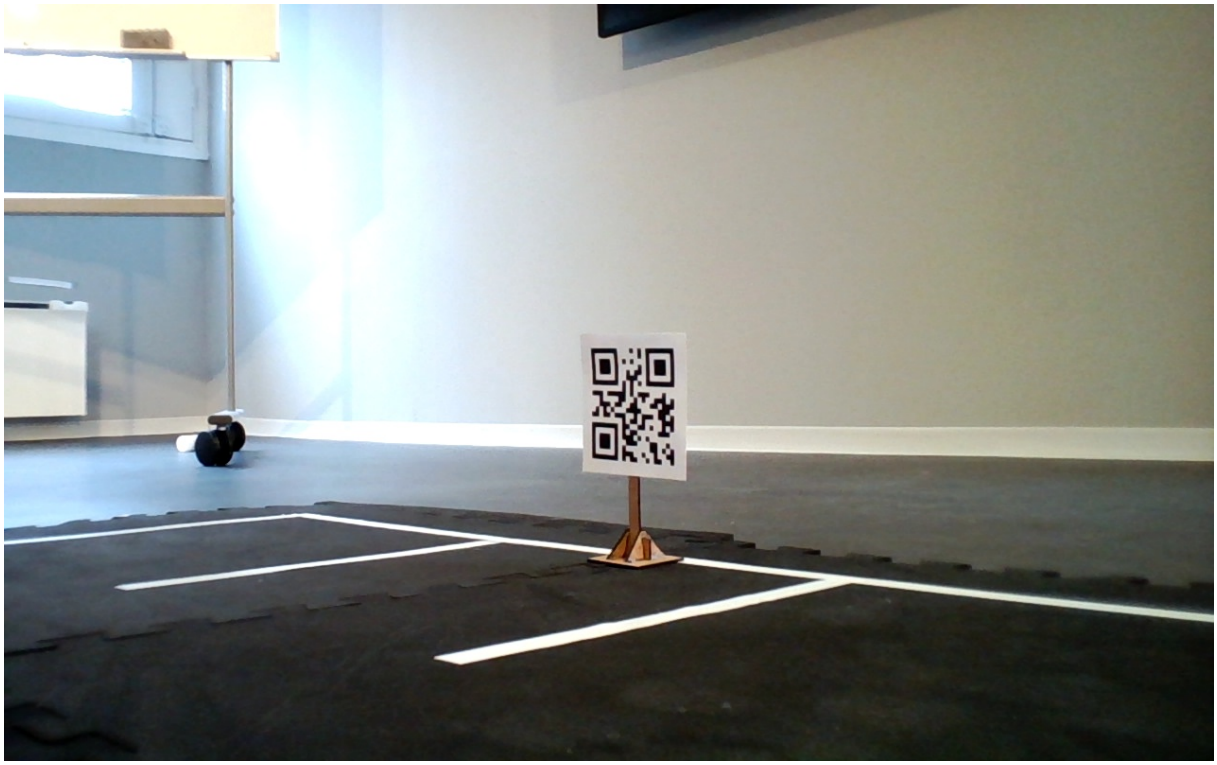
the algorithms. The unit tests have been used on the implementations to determine if the module alone performs as intended. Whereas, integration tests have been carried out to comprehend how this module works with other modules in the Mapping module. An example of the result of the Path Planning module used with the Mapping module and other computer vision modules is presented in Figure 4.10 and Figure 4.11.

(a) Input camera frame.

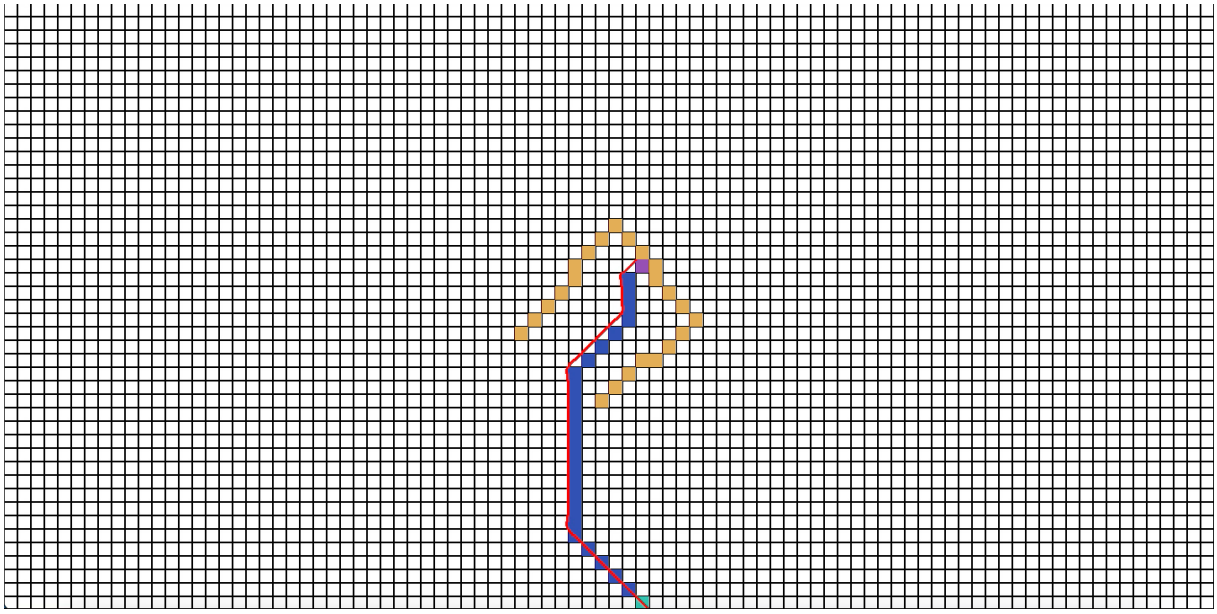


(b) Result from Path Planning module.

Figure 4.10: Result of using the Path Planning module with the Environment Mapping and Lane Detection modules for the lane centering function. The green and purple square corresponds to the vehicle and the destination point respectively. The blue squares are the path given from the A* algorithm and the red lines correspond to the Catmull-Rom spline.



(a) Input camera frame.



(b) Result from Path Planning module.

Figure 4.11: Result of using the Path Planning module with the Environment Mapping, QR Code Detection, and Parking Slot Detection modules for the parking function. The green and purple square corresponds to the vehicle and the destination point respectively. The blue squares are the path given from the A* algorithm and the red lines correspond to the Catmull-Rom spline.

The result and information from integration tests have then been used to perform a physical validation test of manually moving the vehicle. This was done so that the accuracy of the module can be tested without relying on the simple developed motion control, thus, it removes a factor of the source of error. Although moving the vehicle introduces human errors as a source of error,

the uncertainty of this source of error is presumably lower than a simple motion control if the actions are performed precisely. In addition, the results from this test can be used to compare with the system to see if the system's performance is optimal.

For lane-centering path planning, the physical validation test gave a promising result with minimal deviations from the center of the lane after arriving at the destination point. Whereas for path planning in terms of parking, the results were acceptable. However, at cases where the angle was larger than $\pm 20°$, then one of the wheels was on the parking line or slightly outside the parking slot. Overall, although the final position of the vehicle relative to the parking slot was not perfectly accurate in all cases, the result showed that the path obtained from the module was approximately heading in the right direction. However, because of the deviations presented in Section 4.1.2 with the QR code not always being perfectly aligned with the parking lines, this has led to the final position not being perfect.

4.1.4 Reliability of the System

The reliability of the system was tested by executing the software with the four objective states presented in Section 3.1.4. First of all, the `STEREO` state was used to test if the hindrance detection development worked reliably. While using this state, it was observed that when the software was able to detect the actual hindrances within the critical threshold, the desktop vehicle would stop moving, although with a delay in reaction time because of the hardware's low processing power. However, as mentioned in Section 4.1.1, the vehicle also stopped in most scenarios when there were no hindrances because the software mistakenly detected objects. Thus, this issue greatly reduces the reliability of the system in terms of hindrance detection. As this function is quite demanding regarding processing power, as well as working better on large objects, thus, this might be more suitable for larger systems with more processing power. Subsequently, the `STOPPING` state was applied to test the reliability of the capability of reacting to stop signs. The test result showed that the system performed with high accuracy and reliability by stopping whenever a stop sign was detected within a threshold distance. This state was tested on both test vehicles and performed well on both of them. The result of the state execution can be watched here https://youtu.be/xawxZPvk0ts. Subsequently, the state's performance time can be watched here https://youtu.be/bhT4F5p6ftg.

Regarding the two remaining states, `DRIVING` for lane centering and `PARKING` for parking both rely on the motion control presented in Section 3.5, unlike the two previously mentioned states that only need to stop the movement of the vehicle. Thus, when testing the function of centering itself with the desktop test vehicle, the deviation from the center of the lane lines is larger than 10% from the center most of the time, which is not an ideal outcome. Similarly, for parking, the system has a tendency to end with a final position where more than just one wheel of the vehicle is outside of the parking slot whenever the angle is larger than $\pm 20°$. Both states give therefore a worse result than the ones presented in Section 4.1.3. A possible explanation for this is as previously mentioned, the states rely on the simply developed motion control, which can also cause further deviations than the already existing ones from the Path Planning module. Although the driving logic presented in Section 3.1.4 is supposed to reduce the mentioned deviation, where the main purpose is to constantly use a new frame in order to calculate a new path if possible, the result was still imperfect. Hence, the system in terms of centering and parking the vehicle is therefore not reliable on the small test vehicle. As the system did not perform satisfactorily on the desktop test vehicle, these two states were not tested on the larger vehicle.

Overall, one of the four objectives performed reliably based on the evaluation criteria from Section 1.4 for both test vehicles. Whereas two of the objectives performed theoretically sufficiently, but not when combined as a system. Finally, the last objective did not perform well overall. Thus, the issues presented regarding the three objectives which did not perform well within a system must be resolved to make the entire system more reliable and operational. First for the desktop test vehicle, and then later to scale for the larger one.

4.2 Comparison with Existing Autonomous Driving Systems

First of all, a prototype vehicle has been used for the developed software to be implemented in this project. Furthermore, this prototype cannot bear direct comparison with other high-level existing autonomous vehicle systems as they integrate other sensors into their vehicle like LIDAR or RADAR to improve their autonomous functions. Meanwhile, this project focuses primarily on computer vision. Thus, regardless of the results in Section 4.1, this system still might not produce as accurate results and include a wide set of functionality as some of the existing autonomous systems. However, it provides an excellent starting point for further learning and development of a more advanced system.

4.3 Limitations and Areas for Improvement

The current system has some limitations in both the hardware and software aspects. First of all, the prototype vehicle is equipped with a small camera due to its size. Small cameras, especially cheap ones, have smaller image sensors and lesser-quality optics. This affects the quality of the captured images regardless of resolution, and can to some degree lead to the software to struggling to detect objects and lines accurately. Our test location also had some issues with reflections sometimes causing glare in the captured images, preventing the algorithm to detect lines correctly. Overall, the image processing algorithms worked well when parameters were tweaked for the exact conditions, and the conditions were made as ideal as possible. The results might be different in other conditions and environments. The obtained results are gathered using the described setup of the vehicle model and motion control. Even though the modules can be used on other hardware, some tweaking and changes may be needed, and results may vary.

It is possible that more work on camera placement, shading, filters, and choice of lenses might improve the quality and consistency of the input images in changing conditions.

Stereoscopic vision has a lot of areas that can be improved to resolve the issue of detecting objects when there are none. First of all, the tuning might be adjusted so that the disparity map is not as smooth. That may also be accomplished by removing the blurring. Performing these two actions might resolve the previously mentioned issue. However, this will make it more prone to noise and increase the difficulty to detect obstacles and was not suitable for this project's scenarios. Secondly, the size of what contour is detectable can be increased so that the small irrelevant contours will not be detected.
From the software perspective, Python was used as the primary language in this project, although it is not suited for high-performance tasks as mentioned in Section 3.1.4 regarding threading. An area of improvement could therefore be to use a high-performance programming language

like Rust or C++, which is better at the previously mentioned issue. Furthermore, if the prototype vehicle is equipped with a more powerful onboard computer, the speed of the system can be improved. The Raspberry Pi has limited processing power. This was not ideal for the performance of stereoscopic vision object detection. If continuing on that route, the software could benefit from a more powerful computer.
Subsequently, as mentioned in Section 2.3, Local Binary Pattern Cascade has been used as the object detection method to detect stop signs because it requires fewer resources to be able to obtain a usable trained model. However, even though this module already has a high accuracy from the test result, the accuracy can still be further improved. Convolutional Neural Networks can be used instead to achieve the same purpose with higher accuracy. For instance, YOLO is a promising approach for this use case. In the aspect of path planning, as it is based on the environment, the mapping of objects and lines into the environment is also a critical factor for the accuracy of the path the vehicle will travel. Further improvements on the mathematical functions that were found between the correlation of pixels on the camera frame and the real distance that was used to map ground objects should be carried out to improve the accuracy of parking path planning. Improvement in the calculation of distance for the QR code is also important as it is the destination point for parking and needs to be aligned with the mapping of lines which did not perform as accurately as presented in the result in Section 4.1. Overall, addressing these limitations and areas of improvement will significantly increase the performance of the system as well as its accuracy and reliability.

Chapter 5

Conclusion and Future Works

5.1 Summary of Findings and Contributions

Throughout the thesis, different computer vision methods have been presented for the different autonomous function objectives. The proposed software solution for an autonomous vehicle that is solely based on computer vision is a combination of applications of existing algorithms, modification of algorithms, and self-developed algorithms.

First of all, an existing algorithm for stereoscopic vision has been used for the detection of large objects corresponding to hindrances in front of the vehicle. Thereafter, for the detection of stop signs, a Local Binary Pattern model has been trained to be used in a concept called Cascade of Classifier to detect the desired signs. Furthermore, for lane keeping and detecting parking slots, a combination of image processing methods to achieve edge detection and Hough transform was performed to detect the lines given by the vehicle's camera. Afterwards, for each application of line detection, a suitable region of interest was extracted and a suitable method of clustering lines was utilized to reduce the number of lines detected from the line detection to only necessary lines. For lane keeping, it was the lane lines, and for parking slots, it was the parking lines. Perspective transform was then used with the detected lane lines to obtain a point that is in the middle of the road and a desired amount of distance away from the vehicle. This point was further used as a destination point to handle the action of centering the vehicle within a road. For parking, a QR code was used with the parking lines to represent an empty slot and was applied as the destination point for autonomous parking. All the detected objects and lines were thereafter mapped into a two-dimensional environment that is represented by a $m \times n$ matrix. The mapping of ground elements as lines were based on calculating the distance from the vehicle to the object based on two mathematical functions that were discovered through regression analysis of the correlation between the pixel values and the physical distance. In terms of mapping signs to the environment, an algorithm based on using a reference value to calculate the distance to the signs has been used. Thereafter, a self-modified version of the shortest path algorithm A* was utilized to define a path to the desired destination. The result of this algorithm is then used to calculate a Catmull-Rom spline for obtaining a smoother trajectory. By using the information from this resulting path, it was possible to control the motion of the vehicle to obtain the project's objectives.

Tests were then performed to determine the accuracy of each module separately, and jointly in

environment mapping, path planning, and the reliability of the system. For each module and together in environment mapping, there existed a deviation from the desired result that caused a reduction in the accuracy. However, the deviation for most of the computer vision modules alone was within an acceptable amount. On the other hand, when the calculation from the computer vision modules was mapped together and used for path planning, the information achieved deviated slightly from the ideal result. This led to the fact that when performing a physical validation test on the Path Planning module for the parking function, the ending position of the parking was not always ideal. For instance, one of the vehicle wheels was often on the parking line. Subsequently, when testing the vehicle, the system was able to detect hindrances in front of the vehicle and stopped the vehicle's movement. However, the system also detected objects when none was visible in the vehicle's field of view. Thus, in terms of this objective, the system was not reliable. On the other hand, the system performed with high reliability regarding the function of stopping the movement whenever a stop sign was detected. For more complex actions such as centering the vehicle within a lane and parking, the system relied on a simple driving logic that led to deviations from the planned path. Thus, in both cases, the system struggled to achieve its objective. For centering the vehicle, the ending position of the vehicle has a tendency to deviate more than the set evaluation criteria. On the other hand, for parking, a part of the vehicle mostly ended up outside of the parking slot. Improvements are therefore needed to resolve these previously mentioned issues and also to enhance the already functional ones to increase their reliability, which is important to address in future development.

5.2 Recommendations for Future Research and Development

For further development of this system, it is important to address the limitations and areas of improvement mentioned in Section 4.3 as it turned out to be challenges that decreased the system's performance and accuracy. Furthermore, based on existing autonomous functions, possible future research could be to make the system more suitable for complex scenarios. For instance, the lane detection method can be further developed such that more complex roads can be detected, e.g., curved roads. Another direction is to expand stop sign detection to detect multiple types of signs instead of only stop signs and react to each one differently. Furthermore, research on how environment mapping can be further used is ideal for more complicated path traversal. As mentioned in 3.3 by memorizing previously seen objects in the environment, the vehicle has the possibility to travel to a destination on a path that is not currently visible in the vehicle's field of view. Finally, although this is beyond the general scope of computer vision, it is possible to integrate other sensors such as LIDAR and RADAR as additional sources of information to further research how to improve autonomous driving functions with other sensors.

Bibliography

[1] Epipolar (stereo) geometry. https://www.cse.unr.edu/~bebis/CS791E/Notes/EpipolarGeonetry.pdf. Accessed: April 20, 2023.

[2] Ehsan Akbari Sekehravani, Eduard Babulak, and Mehdi Masoodi. Implementing canny edge detection algorithm for noisy image. *Bulletin of Electrical Engineering and Informatics*, 9:1404–1410, 08 2020.

[3] Rosa Andrie Asmara, Muhammad Ridwan, and Gunawan Budiprasetyo. Haar cascade and convolutional neural network face detection in client-side for cloud computing face recognition. In *2021 International Conference on Electrical and Information Technology (IEIT)*, pages 1–5, Sep. 2021.

[4] Naveen Appiah and Nitin Bandaru. Obstacle detection using stereo vision for self-driving cars. *EE368/CS232: Digital Image Processing*.

[5] Ben. Training a cascade classifier. https://learncodebygaming.com/blog/training-a-cascade-classifier, August 2020. Accessed: January 29, 2023.

[6] Jo Chang-Yeon. Face detection using lbp features. volume 77, pages 1–4. Citeseer, 2008.

[7] Jeremy Cohen. Path planning for self-driving cars. https://thinkautonomous.medium.com/can-self-driving-car-think-6c9e8d939d60, July 2021. Accessed: April 12, 2023.

[8] Hamid Fsian, Vahid Mohammadi, Pierre Gouton, and Saeid Minaei. Comparison of stereo matching algorithms for the development of disparity map, 2022.

[9] Javatpoint. Bresenham's line algorithm. https://www.javatpoint.com/computer-graphics-bresenhams-line-algorithm. Accessed: March 28, 2023.

[10] Nitin Kanagaraj, David Hicks, Ayush Goyal, Sanju Tiwari, and Ghanapriya Singh. Deep learning using computer vision in self driving cars for lane and traffic sign detection. *International Journal of System Assurance Engineering and Management*, 12(6):1011–1025, 2021.

[11] Erdogan KAYA. Spline interpolation techniques. *Journal of Technical Science and Technologies*, January 2014.

[12] Mikko Kytö, Mikko Nuutinen, and Pirkko Oittinen. Method for measuring stereo camera depth accuracy based on stereoscopic vision. *SPIE Proceedings*, 2011.

[13] Gustavo Teodoro Laureano, Maria Stela Veludo de Paiva, Anderson da Silva Soares, and Clarimar José Coelho. Chapter 34 - a topological approach for detection of chessboard patterns for camera calibration. In Leonidas Deligiannidis and Hamid R. Arabnia, editors,

Emerging Trends in Image Processing, Computer Vision and Pattern Recognition, pages 517–531. Morgan Kaufmann, Boston, 2015.

[14] E.T.Y. Lee. Choosing nodes in parametric curve interpolation. *Computer-Aided Design*, 21(6):363–370, 1989.

[15] Howard Lee and Yi-Ping Phoebe Chen. Image based computer aided diagnosis system for cancer detection. *Expert Systems with Applications*, 42(12):5356–5365, 2015.

[16] Arm Ltd. What is computer vision. https://www.arm.com/glossary/computer-vision. Accessed: March 22, 2023.

[17] Lokesh Madan, Kislay Anand, and Bharat Bhushan. Bresenham's lines algorithm. volume 4. International Journal of Research in Science And Technology, 2014.

[18] Ezio Malis and Manuel Vargas. Deeper understanding of the homography decomposition for vision-based control. January 2007.

[19] Satya Mallick. Learnopencv. https://github.com/spmallick/learnopencv, 2015. Accessed: April 20, 2023.

[20] Satya Mallick. Geometry of image formation. https://learnopencv.com/geometry-of-image-formation/, February 2020. Accessed: April 20, 2023.

[21] MathWorks. Stereo disparity using semi-global block matching. https://se.mathworks.com/help/visionhdl/ug/stereoscopic-disparity.html, note = Accessed: April 20, 2023.

[22] MathWorks. What is camera calibration? - matlab & simulink. https://se.mathworks.com/help/vision/ug/camera-calibration.html. Accessed: April 20, 2023.

[23] Anusha Jayasree Mavilla Vari Palli and Vishnu Sai Medimi. A comparative study of yolo and haar cascade algorithm for helmet and license plate detection of motorcycles. *DiVA*, September 2022.

[24] Domingo Mery and Christian Pieringer. *Computer Vision for X-Ray Testing*. Springer Cham, 2021.

[25] Ekaterina Mezenceva and Sergey Malakhov. The study of the semi-global block matching algorithm implementing parallel calculation with gpu. In *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T)*, pages 649–652, 2021.

[26] OpenCV. Camera calibration and 3d reconstruction. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=stereobm#stereosgbm. Accessed: April 20, 2023.

[27] OpenCV. Canny edge detection. https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html. Accessed: February 10, 2023.

[28] OpenCV. Cascade classifier. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html. Accessed: January 29, 2023.

[29] OpenCV. Cascade classifier training. https://docs.opencv.org/4.x/dc/d88/tutorial_traincascade.html. Accessed: January 29, 2023.

[30] OpenCV. Feature detection. https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga354e0d7c86d0d9da75de9b9701a9a87e. Accessed: January 29, 2023.

[31] OpenCV. Geometric image transformations. https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html. Accessed: February 10, 2023.

[32] OpenCV. Hough line transform. https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html. Accessed: February 10, 2023.

[33] OpenCV. Image filtering. https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1.

[34] OpenCV. Morphological transformations. https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html. Accessed: February 10, 2023.

[35] OpenCV. Opencv camera calibration. https://docs.opencv.org/3.0beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration. Accessed: April 20, 2023.

[36] OpenCV. Remapping. https://docs.opencv.org/3.4/d1/da0/tutorial_remap.html. Accessed: May 5, 2023.

[37] OpenCV. Camera calibration. https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html, 2022. Accessed: April 20, 2023.

[38] OpenCV Contributors. Delay in videocapture because of buffer. https://forum.opencv.org/t/delay-in-videocapture-because-of-buffer/2755. Accessed: April 26, 2023.

[39] OpenCV Contributors. Introduction. https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/index.html. Accessed: April 26, 2023.

[40] R.R. Orozco, C. Loscos, I. Martin, and A. Artusi. Chapter 4 - multiview hdr video sequence generation. In Frédéric Dufaux, Patrick Le Callet, Rafał K. Mantiuk, and Marta Mrak, editors, *High Dynamic Range Video*, pages 121–138. Academic Press, 2016.

[41] A.J. Parker. Stereoscopic vision. In Larry R. Squire, editor, *Encyclopedia of Neuroscience*, pages 411–417. Academic Press, Oxford, 2009.

[42] Shirley Peter, Thompson William B, Willemsen Peter, Wyvill Brian, Marshner Steve, Ashikhmin Michael, Gleicher Michael, Hoffman Naty, Johnson Garrett, Munzner Tamara, Reinhard Erik, and Sung Kelvin. *Fundamentals of Computer Graphics, 3rd edition*. AK Peters, - edition, 2009. Other identifier: 2001309.

[43] Raspberry Pi. Raspberry pi 4. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Accessed: April 26, 2023.

[44] Elyse Betters Picaro. Amazon go and amazon fresh: How the 'just walk out' tech works. https://www.pocket-lint.com/gadgets/news/amazon/139650-what-is-amazon-go-where-is-it-and-how-does-it-work/, February 2023. Accessed: March 22, 2023.

[45] Esa Prakasa. Texture feature extraction by using local binary pattern. *Jurnal INKOM*, 9:45, May 2016.

[46] S. J. D Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, 2012.

[47] Python. Globalinterpreterlocks. https://wiki.python.org/moin/GlobalInterpreterLock. Accessed: March 27, 2023.

[48] RTSP. Can't set cv_cap_prop_buffersize #13s. https://github.com/dactylroot/rtsp/issues/13. Accessed: April 26, 2023.

[49] Kaustubh Sadekar. Stereo camera depth estimation with opencv (python/c++). https://learnopencv.com/depth-perception-using-stereo-camera-python-c/, April 2021. Accessed: April 20, 2023.

[50] Kaustubh Sadekar and Satya Mallick. Camera calibration using opencv: Learnopencv. https://learnopencv.com/camera-calibration-using-opencv/, May 2021. Accessed: April 20, 2023.

[51] M. Sharifi, M. Fathy, and M.T. Mahmoudi. A classified and comparative study of edge detection algorithms. *ResearchGate*, pages 117–120, 2002.

[52] Keith Shields. 4 best software development methodologies: Which is right for your project? https://designli.co/blog/4-best-software-development-methodologies-which-is-right-for-your-project/, October 2021. Accessed: March 22, 2023.

[53] Contributors Stanford. https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. Accessed: April 12, 2023.

[54] Asbjørn Stokka. Utvikle en mikrokontrollerbasert plattform for arbeid med reguleringsteknikk og autonom kjøring. *UiS Brage*, 2022.

[55] Joao Tavares and A. Padilha. A new approach for merging edge line segments. January 1995.

[56] Xilinx. Stereo local block matching user guide. https://www.xilinx.com/publications/user-guide/stereo-local-block-matching-user-guide.pdf. Accessed: April 20, 2023.

[57] Xiaoguang Xu, Jun Wang, Yi Wang, and Alexander Kokhanovsky. Chapter 1 - passive remote sensing of aerosol height. In Tanvir Islam, Yongxiang Hu, Alexander Kokhanovsky, and Jun Wang, editors, *Remote Sensing of Aerosols, Clouds, and Precipitation*, pages 1–22. Elsevier, 2018.

[58] Cem Yuksel, Scott Schaefer, and John Keyser. Parameterization and applications of catmull-rom curves. *Computer Aided Design*, 43(7):747–755, 2011.

Appendices A

Github Repository

The project's code implementation can be found here: https://github.com/stokka-elebac-22/RC-Project-Autonomous-Car

Appendices B

Functionality Testing of Stereoscopic Vision Calibration

Calibration Testing of Stereoscopic Vision

Eirik Reiestad
Asbjørn Stokka
Julie Vy Tran

Tested: February 1, 2023
Revised: February 1, 2023

Summary

The result from the calibration process is dependent on the light conditions it was calibrated with. To get the best possible results, no glare is preferred.

Introduction

This report documents the calibration process. The goal for this test rapport is to get an insight into which environmental conditions the best calibration results reside in and elaborate on the sources of error.

Equipment

- Logitech C920 Pro
- Suyin HD Camera
- 13x9 checkerboard
- Stereo vision pair
- 3D-printed case for the camera pair

Method

To perform this test, two cameras should be mounted in a fixed position to avoid movement as a source of error. On the lower-end cameras, a 3D-printed case is used to keep them in a fixed position. The *Logitech C920 Pro* cameras are just placed at a position and not moved for the whole testing period. A program is used to take photos with both cameras. Throughout the whole testing period, the same room and objects in the room are used. A checkerboard is moved around the room inside the frame of both cameras at different angles and positions. To have a solid amount of calibration data, at least 20 photos should be taken. This calibration method is tested in different light conditions:

- Sunlight
- Indoor light
- Indoor light with sunlight

The sunlight is passed through the window. The checkerboard is displayed on an iPad with a mat screen protector to reduce glare. For the parameter tuning, the following basic tuning is used:

Parameter	**Test 1**	**Test 2**	**Test 3**	**Test 4**
Number of disparities	16	16	80	160
Block size	11	51	51	61

With this, the goal is to reduce the noise in the image as much as possible and display a result that reflects the real-life positions of objects.

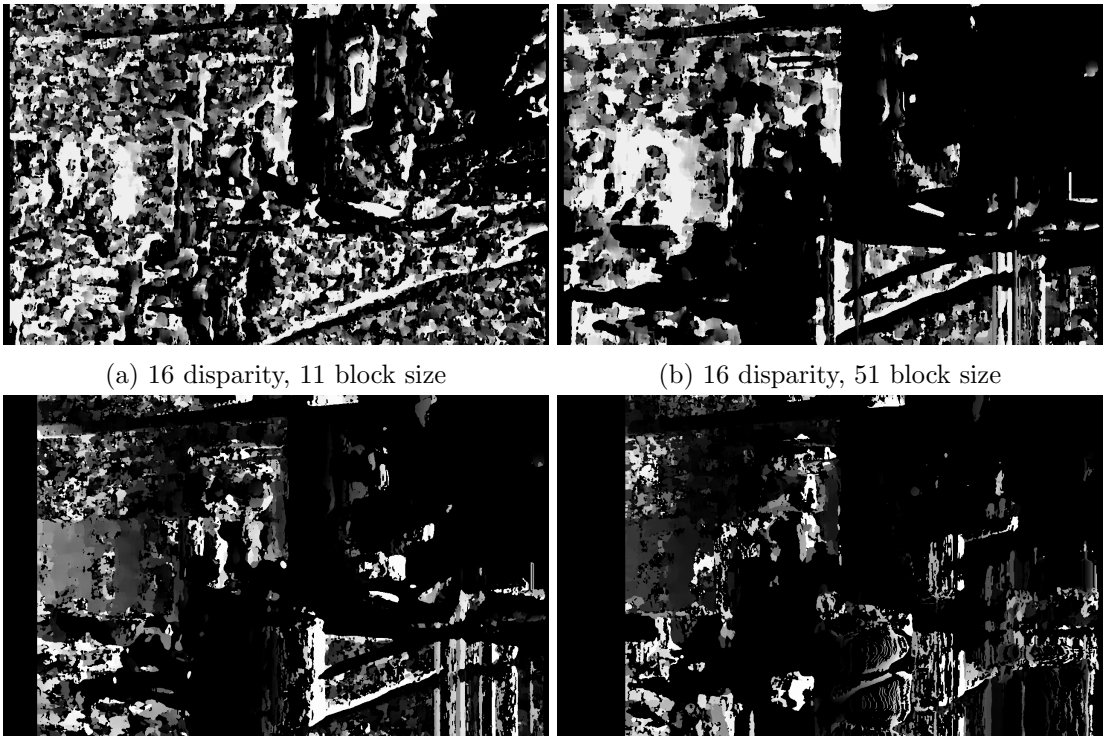
Result

For testing purposes, the cameras are not moved throughout the whole process, resulting in the same image as a stereo-vision pair shown in Figure 1. The photos might appear dark due to the photos being taken at night time to eliminate glare from the sun. The only light source is the indoor lights.

(a) Left image (b) Right image

Figure 1: The stereo image pair.

The following photos in Figure 2, Figure 3, and Figure 4 are disparity maps, the result of calibration in a specific environment with different calibration settings.



(a) 16 disparity, 11 block size

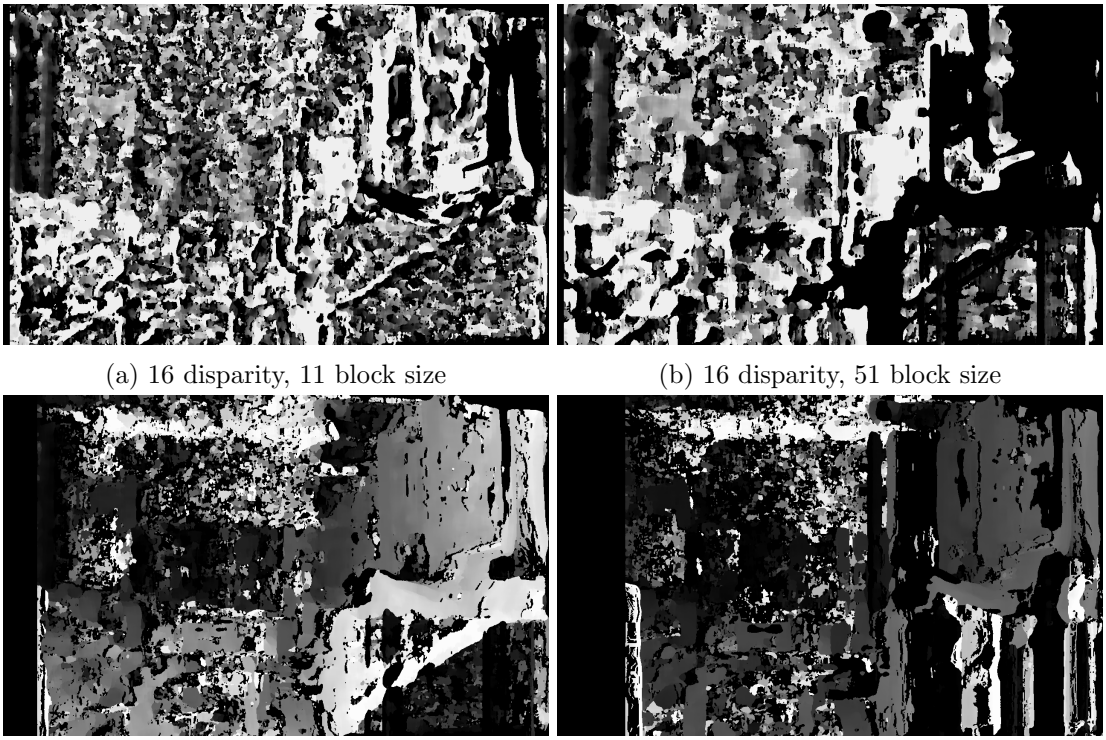
(b) 16 disparity, 51 block size

(c) 80 disparity, 51 block size

(d) 160 disparity, 61 block size

Figure 2: The calibration photos were taken with the sun beaming through the windows.

The disparity maps in Figure 2 show a map with low readability, implying that it is hard to see what it should represent. In addition, it does not show a clear difference from where the closest object is located.



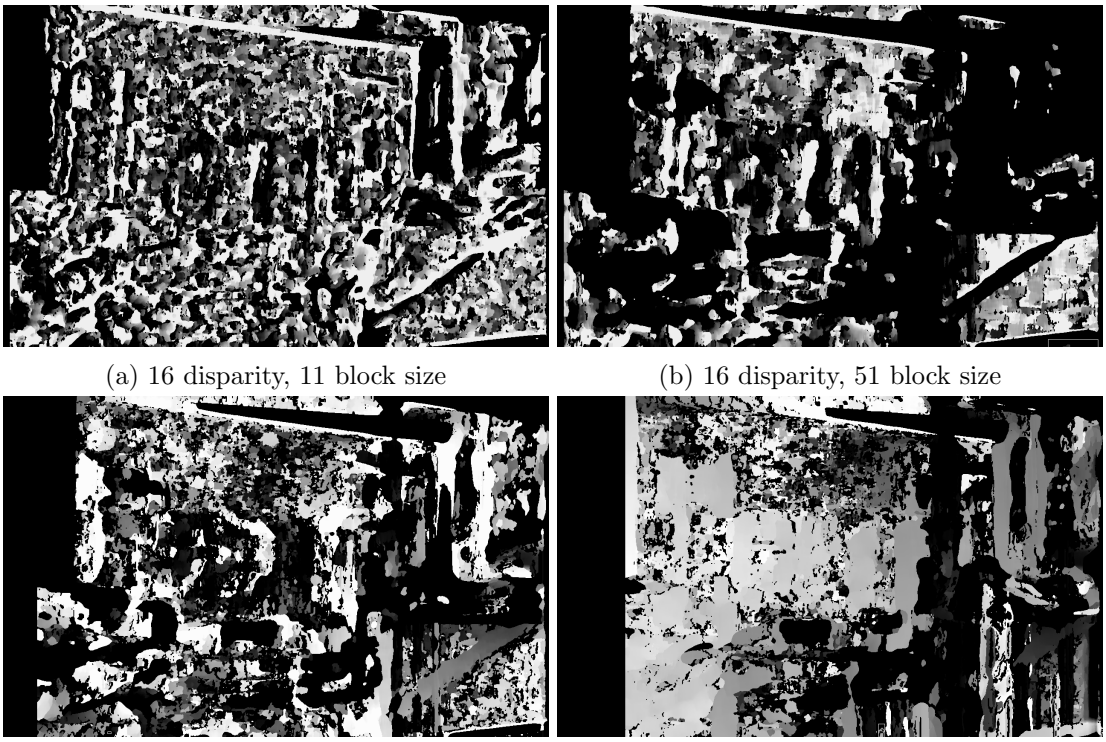
(a) 16 disparity, 11 block size

(b) 16 disparity, 51 block size

(c) 80 disparity, 51 block size

(d) 160 disparity, 61 block size

Figure 3: The calibration photos were taken with indoor lights on.



(a) 16 disparity, 11 block size

(b) 16 disparity, 51 block size

(c) 80 disparity, 51 block size

(d) 160 disparity, 61 block size

Figure 4: The calibration photos were taken with indoor lights on, and the sun beaming through the windows.

Discussion

Figure 3 shows a recognizable result. The calibration photos with light sources from indoors and the sun shown in Figure 4 are not as clear as Figure 3, but show a better result compared to the calibration photos with only the sun as a light source.

The previous figures are the disparity photos taken with the web camera. No advanced parameter adjustments were made, but some parameter changes to see how they affected the different maps in different environments. As the photos show, when there is a combination of both light and sun, there is a lot of noise in the disparity map. Reducing the glare will also reduce the noise and evoke a representative disparity map.

More test photos were taken to verify the results. Though, only a handful was shown to show the major differences in the different light conditions. Note however that this is not an optimal adjusted representation of the disparity map as it is a complex process where both light conditions and a subjective meaning are relevant factors.

Conclusion

The calibration process was tested in different environments in order to look into which light settings give the best calibration photos and disparity maps. From the test results, it was able to observe that the environment with no glare is preferred, meaning it is beneficial to limit the light from the sun and only use indoor lights.

Appendices C

Functionality Testing of Stereoscopic Vision Tuning

Tuning Testing of Stereoscopic Vision

Eirik Reiestad
Asbjørn Stokka
Julie Vy Tran

Tested: February 1, 2023
Revised: February 1, 2023

Summary

This test documents the parameter adjustment. The result of the test showed that adjusting every parameter is a complex task. In this case, only adjusting the main parameters, the number of disparities and the block size will give the best result.

Introduction

This report documents the parameter adjustment process. The goal of this rapport is to get an insight into which parameter values are suited.

Equipment

- A stereo image pair

Method

The following stereo image pair presented in Figure 1 is used for parameter adjustment.

(a) Left (b) Right

Figure 1: Stereo image pair

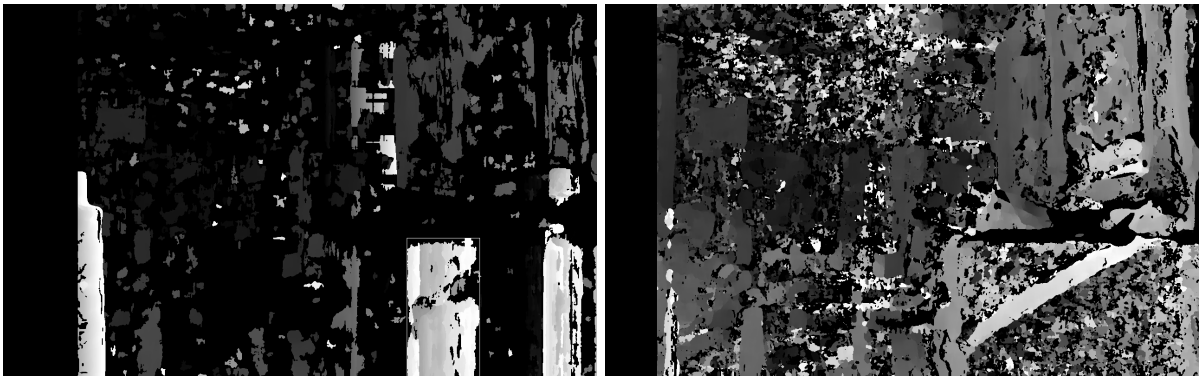
The following parameters will be adjusted:

Parameters	Description
Minimum disparity	The minimum value of the disparity to be searched
Number of Disparities	It establishes the range of values for the disparity to be searched for
Block Size	The size of the blocks in relation to the algorithm, in linear form.
P1	One of two parameters controlling the disparity smoothness. The larger the values are, the smoother the disparity is. This parameter is the penalty on the disparity change by plus or minus 1 between neighbor pixels.
P2	The other parameter controlling the disparity smoothness. The larger the values are, the smoother the disparity is. This parameter is the penalty on the disparity change by more than 1 between neighbor pixels.
disp12MaxDiff	Maximum allowed difference in the left-right disparity check. Setting it to a non-positive value will disable the check.
Pre-Filter Cap	Restricts the output of the filtering process to a specific value.
Uniqueness Ratio	The pixel is filtered out if the best matching disparity is not sufficiently better than every other disparity in the search range
Speckle Range	If the best matching disparity value is not significantly greater than the other values within the search range, the pixel will be removed from the output
Speckle Window Size	The threshold of the number of pixels, below which a disparity group is considered as "speckle" and discarded

An approach where one adjusts one parameter to the optimal and moves on to the next one is applied for this process. When reached the last parameter, the first one is adjusted again. When every parameter has been adjusted and no more gives any improvement, the adjustment is done. The criteria that say if a calibration parameter combination is better are the clarity of the map and if a function can detect the closest object. There will also be an additional test with only adjusting the number of disparities and the block size to see how to recommended settings perform.

Result

The result of parameter tuning is presented in Figure 2.



(a) Tuning every parameter (b) Turning only two parameters.

Figure 2: Result of parameter adjustment on the stereo image pair.

The most optimal tuning with the method of tuning every parameter is given below and the result of using this set of parameters on the test image is shown in Figure 2a.

Parameters	**Value**
Minimum disparity	0
Number of Disparities	160
Block Size	23
P1	100
P2	0
disp12MaxDiff	8
Pre-Filter Cap	0
Uniqueness Ratio	14
Speckle Range	1
Speckle Window Size	100

For the method of tuning only two parameters that are given below, the result is as shown in Figure 2b.

Parameters	**Value**
Number of Disparities	112
Block Size	19

Discussion

The result showed that adjusting every parameter was a complex task, resulting in a less representative disparity map. Only adjusting two parameters; the number of disparities and the block size, gave a better result.

Sources of Error

There are some sources of error that might alter the obtained result. The stereo photos might not be the best possible photos because of their lack of patterns, thus, resulting in single-colored shapes, which stereo vision is not best suited for.

Conclusion

The process of adjusting every parameter was complex, hence for this case, it was beneficial to only adjust the main parameters that is the number of disparities and the block size. The ideal values were 112 and 19 respectively.

Appendices D

Functionality Testing of Stereoscopic Vision Blurring

Blur Testing of Stereoscopic Vision

Eirik Reiestad
Asbjørn Stokka
Julie Vy Tran

Tested: February 1, 2023
Revised: February 1, 2023

Summary

This test looks into the effect blurring will have on a disparity map. Different degree of blurring was used and the smoothest and best disparity map was subjectively chosen. The result showed that blurring the images will create a smoother disparity map. However, it will result in a less detailed disparity map.

Introduction

This report documents the result of blurring the image for a stereoscopic vision pair. Different degrees of blurring are compared. This is done to find out if blurring will give a smoother disparity map and if it should be integrated.

Equipment

- Suyin HD Camera (1280 x 800)
- Stereo vision pair

Method

The same stereo-vision pair is used for all the tests. First, the blur is increased until the image gives unreadable or noticeably worse results. An acceptable adjustment of the stereo parameters should be found before the test. This should be used for the same image, where only the degree of blurring should be changed.

The blur function that is used is a function from the OpenCV library and is given below:

```
blurred_image = cv2.blur(image, (x_{blur}, y_{blur})
```

Result

The images as a result of different values of blurring are presented below. The photos might appear dark due to the photos being taken at night time to eliminate glare from the sun. The only light source is the indoor lights.

(a) Left image (b) Right image

Figure 1: Stereo image pair

(a) Left blurred image (b) Disparity

Figure 2: Blur = 0

(a) Left blurred image (b) Disparity

Figure 3: Blur = 5

(a) Left blurred image (b) Disparity

Figure 4: Blur = 10

(a) Left blurred image (b) Disparity

Figure 5: Blur = 15

(a) Left blurred image (b) Disparity

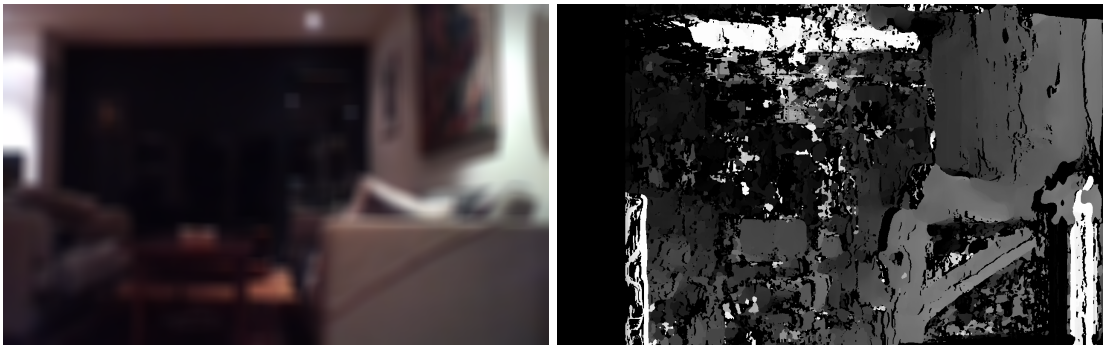
Figure 6: Blur = 20

(a) Left blurred image (b) Disparity

Figure 7: Blur = 25

(a) Left blurred image (b) Disparity

Figure 8: Blur = 30

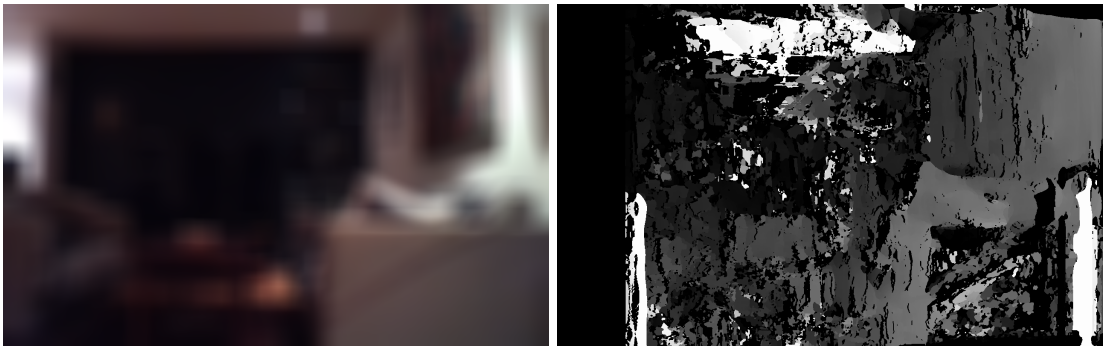


(a) Left blurred image (b) Disparity

Figure 9: Blur = 35

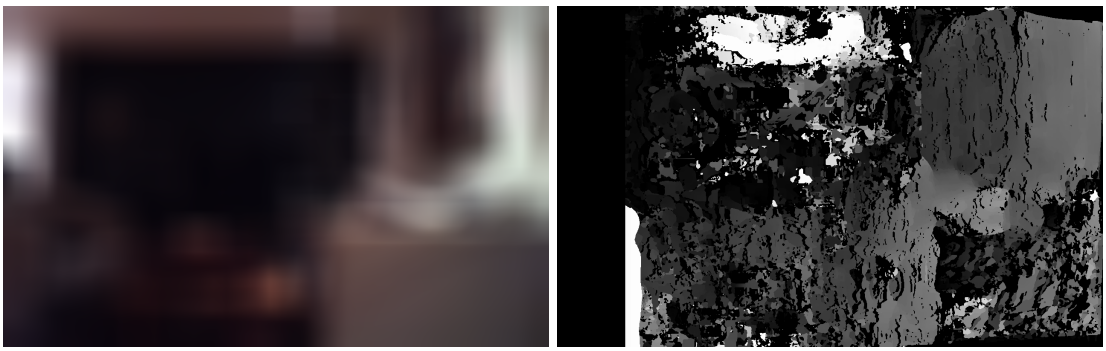
(a) Left blurred image (b) Disparity

Figure 10: Blur = 40



(a) Left blurred image (b) Disparity

Figure 11: Blur = 60



(a) Left blurred image (b) Disparity

Figure 12: Blur = 100

Discussion

From the figures in the result section, it is possible to observe that higher blur results gave a smoother disparity map. However, a blur greater than 25 led to a loss of details in the lower right corner.

Conclusion

For this scenario, a blur between 10 and 15 seems like a good tradeoff when it comes to accuracy compared to smoothness.

Appendices E

Functionality Testing of QR Code

QR code Accuracy Test

Eirik Reiestad
Asbjørn Stokka
Julie Vy Tran

Tested: March 13, 2023
Revised: March 13, 2023

Summary

The module can detect the QR code between $-35°$ and $35°$. The distance at which the QR code is detected is a linear function of the size of the QR code. A higher quality camera will also increase the distance it can detect the sign. The distance does not affect which range the angle detection has, although when the distance increases, the angle might differ more because of fewer pixels to calculate the angle from.

Introduction

This report documents functionality testing of the QR Code Module. The purpose of the testing is to make sure that the module is able to detect QR codes and determine the relevant range of distance and angle that the QR code can be detected. In addition, another goal is to check if the camera quality affects the result of the previous purpose.

Equipment

- Logitech C920 Pro
- Suyin HD Camera
- QR code
- Measurement tape
- Protractor

Method

A tape measure and a protractor are used to verify the distance and the angle. The program output was recorded and verified using both the Logitech C920 Pro camera and a lower-end webcam.

To perform this test, the QR code sign is placed perpendicular to the camera view. Thereafter, the sign is moved closer to the camera until it cannot be detected anymore. Afterward, the QR code is moved further away until it is not consistently detected. In the end, the QR code is moved further away until it is not detected at all. The angle is kept the same at all distances and the different distances are measured by the measuring tape while keeping the same angle.

Further, to perform the test for angle accuracy, the QR code sign is placed at a distance. First, it is rotated to one side, then to the other, so that the maximum angle the module can detect the QR code is measured. Then, the QR code is moved further away to see how the distance affects the angle.

Result

Figure 1 shows how large the QR code sign needs to be able to see it at a certain distance.

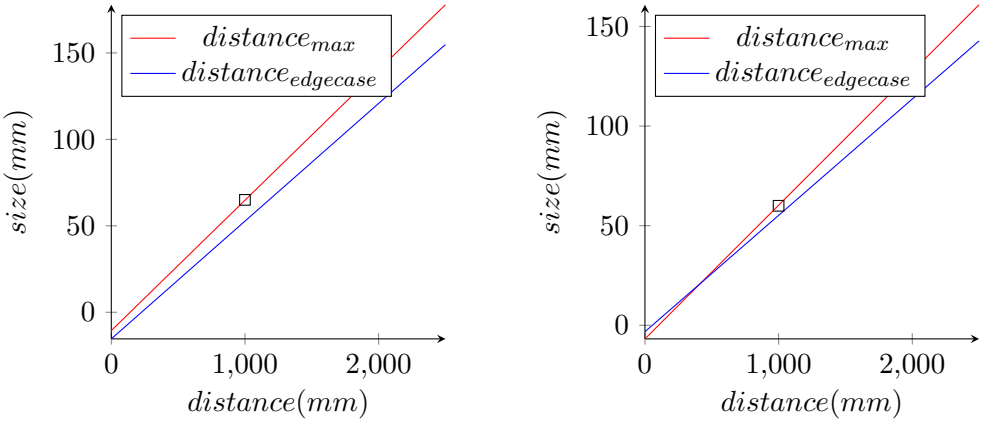


Figure 1: Web camera (1280 x 800) to the left. *Logi C920 Pro* (1980 x 1080) to the right. The red line ($distance_{max}$) is the max distance where the QR code is constantly detected. The blue line ($distance_{edgecase}$) is the max distance the QR code can be slightly detected. The black square indicates which size the QR code needs to be to detect it at a distance of 1 meter. The lower-quality web camera needs a larger size ($\approx 65mm$) compared to the higher-quality web camera ($\approx 60mm$).

For the lower-end webcam with a QR code size of 101 mm width and 101 mm height, the distance from 0 to 150 mm is irrelevant because the sign is too close to the camera. The distance between 200 mm to 1500 mm is the relevant part as it was able to detect the sign without any troubles. The distance between 1500 mm to 1700 mm is only detecting the QR code occasionally. All distances further away than 1700 mm are irrelevant as the module is not able to detect the sign.

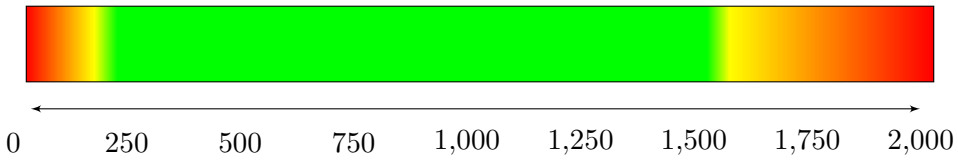


Figure 2: The red section corresponds to the values of the distance in which the camera cannot detect the QR code. The yellow section is hard to detect and the green is detectable.

Camera	Size [mm]	min %	max %	edge case %
Webcam (1280 x 800)	42	0	1.43	3.0
Webcam (1280 x 800)	52	0	3.7	4.0
Webcam (1280 x 800)	101	0	2.0	5.9
Webcam (1280 x 800)	141	0	4.0	13.0
Logi (1920 x 1080)	42	0	7.1	4.0
Logi (1920 x 1080)	52	0	5.2	5.3
Logi (1920 x 1080)	101	0	8.1	-6.4
Logi (1920 x 1080)	141	0	5.9	3.2

Table 1: The first column states which camera is tested. The second column is the size of the QR code in millimeters. The other columns show how big the deviation of the calculated measurements is from the real distance at the minimum, maximum, and edge case distances respectively.

The calculated distance is quite accurate as Table 1 shows that most of the calculation only deviates slightly from the measurements. There are multiple reasons for this deviation; Not accurate reading of the measurements or not an accurate camera calibration process.

When the QR code sign is perpendicular to the camera, it is at 0 degrees. The QR code is hard to detect when it is between $\pm 35°$. At a $\pm 45°$ angle, it is impossible to detect. The same results were found with different distances. However, when the QR code gets moved further away QR codes pixel size decreases and thus gives more unstable results.

Conclusion

The distance the QR code is detected is a linear function of the size of the sign. The test was done on two cameras with different resolutions, showing that a camera with a higher quality will detect the QR code at a further distance. The angle however is not affected by the change of the QR size, except when the distance is at the threshold where the QR code can not constantly be detected, the angle accuracy will differ more. At a normal distance, it will have a range of $-35°$ to $35°$ when looked at at a perpendicular angle.

Appendices F

Functionality Testing of Stop Sign Detection

Functionality Testing of Stop Sign Detection

Asbjørn Stokka (959810)
Eirik Reiestad (260753)
Julie Vy Tran (259195)

Tested: March 20, 2023
Revised: March 20, 2023

Summary

The interval of detectable distances and angles with different minimum sizes of detection was found by measuring at which angle and distance the module stops detecting the sign. The minimum size affected the range of the interval. A larger minimum size implied a smaller value of the maximum distance that was possible for the module to detect the signs. On the hand, the angle range did not vary when measured at different distances. However, it did decrease when the distance was close to the maximum detectable distance. FPS on the other hand increased significantly when the minimum size was increased. Lastly, the module had high accuracy in calculating the distance to the stop sign.

Introduction

This report documents functionality testing of the Stop Sign Detection module. The purpose of the testing is to make sure that the module is able to detect the traffic sign and to determine the detectable range of distance, angle and also frames per second. As the module allows one to choose the minimum size of detection based on the size of the pixels on the camera frame, different values of this minimum size may affect the detectable range of distance and angle. Finally, the test will determine the module's accuracy in calculating the distance to the stop sign.

Equipment

- Logitech C920 Pro
- Suyin HD Camera
- Stop sign
- Measuring tape
- Protractor

Method

To discover the detectable distances, the measuring tape is placed perpendicular to the camera lens. Thereafter, the traffic sign is placed as close as possible and perpendicular to the camera

such as it covers the whole frame, meaning no detection is possible. Afterward, the traffic sign is slowly moved further away until the detections are steady. This is then the minimum distance that the module is able to detect the sign. To discover the maximum distance that the module is able to detect the signs at, the traffic sign is moved further away until the detections become unsteady. This will be the maximum detectable distance.

To discover the detectable angles, the stop sign is placed at different distances that give steady detection. At each distance, the sign is rotated clockwise until the module stops detecting the sign. Thereafter, the sign is rotated counter-clockwise until the module stops detecting the signs. The angles before the module stops detecting the sign are the maximum and minimum detectable angles. 0 degrees indicates the position where the camera is perpendicular to the sign.

The given actions above are done for each type of camera and for different reasonable minimum sizes of detection. For the different minimum sizes, the frame per second should also be calculated by taking the number of frames within a time interval and dividing this sum by the time interval.

Result

The result of determining the detectable distances is given in Table 1.

	Suyin			**Logitech**			
Min Size [px]	**Min**	**Max**	**Int. size**	**Min**	**Max**	**Int. size**	**Diff.**
None	110	3325	3215	87	3570	3483	8.3%
50	120	1955	1835	87	2180	2093	14.1%
75	150	1335	1185	90	1500	1410	19.0%
100	160	1045	885	90	1090	1000	13.0%

Table 1: Result of detectable distances. The different columns except **Min Size** is presented in the unit millimeters.

The result of determining the detectable angles is given in Table 2.

		Cheap			Logitech			
Size [px]	**Dist. [mm]**	**Min**	**Max**	**Int.size**	**Min**	**Max**	**Int.size**	**Diff. [%]**
None	150	-60	60	120	-60	60	120	0%
None	500	-60	60	120	-60	60	120	0%
None	1000	-60	60	120	-60	60	120	0%
None	1500	-50	50	100	-60	60	120	20%
50	150	-60	60	120	-60	60	120	0%
50	500	-60	60	120	-60	60	120	0%
50	1000	-55	55	110	-60	60	120	9%
50	1500	-50	50	100	-60	60	120	20%
75	150	-60	60	120	-60	60	120	0%
75	500	-60	60	120	-60	60	120	0%
75	1000	-55	55	110	-60	60	120	9%
75	1500	-	-	0	-10	10	20	20
100	150	-50	50	100	-50	50	100	0%
100	500	-50	50	100	-50	50	100	0%
100	1000	-40	40	80	-40	40	80	0%
100	1500	-	-	0	-	-	0	0%

Table 2: Result of detectable angles. All the column values excluding the values in column **Size** and Dist. are presented in millimeters.

The result of frames per second for each minimum sign is given in Table 3.

Camera	**Size [px]**	**FPS**
Webcam (1280 x 800)	None	6.1
Webcam (1280 x 800)	50	13.4
Webcam (1280 x 800)	75	23.4
Webcam (1280 x 800)	100	29.7
Logi (1920 x 1080)	None	2.5
Logi (1920 x 1080)	50	6.4
Logi (1920 x 1080)	75	10.6
Logi (1920 x 1080)	100	14.3

Table 3: Result of frames per second for different minimum sizes.

The result of determining the deviation between the measured and calculated distance is given in Table 4.

Distance [mm]		Deviation	
Measured	**Calculated**	**[mm]**	**[%]**
250	271	21	8.3
334	346	12	3.6
412	416	4	0.9
510	521	11	2.2
667	698	31	4.6
729	784	55	7.5
855	885	30	3.5
963	1004	41	4.2
1035	1069	34	3.2
1260	1307	47	3.7

Table 4: Result of determining the deviation between the measured and calculated distance using the Suyin HD Camera.

Discussion

From Table 1, it is possible to perceive that the minimum distance will increase while the maximum distance will decrease when the size is larger. Table 2 shows that the detectable angles did not vary significantly but will decrease slowly when the distance was nearing the maximum detectable distance given in Table 1 for the different minimum sizes. Furthermore, in Table 3 it is perceivable that the frames per second value decreases when the minimum size increases. Therefore, the most suitable minimum size should be a value that takes both FPS and detectable distance into consideration. The minimum size with the value 75 is therefore chosen to be used for this module as it has a reasonable detectable interval of distance and FPS. Moreover, the result given in Table 4 shows that the module has a high accuracy in calculating the distance to the stop sign as all the deviations were less than 10%.

Sources of Error

Possible sources of error in the two tests could be errors in the equipment: Measuring tape and protractor. This could lead to the equipment giving incorrect values when utilized. Another source of error was the possibility of incorrect readings from the measuring equipment when performing the tests. This is an example of a human error. The last source of error could be the lighting. Some parts of the view might have more lighting than others which could cause the sign to be only partially visible. This may therefore lead to the module not being able to detect the sign when it was supposed to.

Conclusion

In conclusion, the minimum size of signs that can be detected also affects the result. The larger this size was, the less will the detectable interval size be. For detectable angles, it only depended on the minimum size when nearing the detectable distances, otherwise, the interval size was equal. The frames per second value was on the other hand dependent on the minimum size. The larger the minimum size was, the larger the FPS was. As the frames per second is crucial for the reaction time of the vehicle, the minimum size with the highest FPS value was the desired one. However, the higher the FPS was, the size of the interval of the detectable distances will be reduced. Finally, the module also had high accuracy in calculating the distance to the

stop sign. Overall, there might existed some uncertainties in all calculations because of errors such as human errors, equipment errors, etc.

Appendices G

Functionality Testing of Lane Detection

Functionality Testing of Lane Detection

Asbjørn Stokka (959810)
Eirik Reiestad (260753)
Julie Vy Tran (259195)

Tested: March 20, 2022
Revised: March 24, 2022

Summary

A set of parameters to use for the module was found using adjust one-by-one method, which is able to detect the vast majority of the relevant positions. Physical measurements were made to find the difference between the calculated and the measured values. For the module's calculation of center difference, there was a high deviation in the result of the calculation. However, the difference compared to the lane line width and the lane width was not significant and the direction of the difference was correct. Therefore, it is possible to use this value to identify which direction the difference is located. For the module's next checkpoint, the deviation was low and this point is therefore proper to be utilized in further usages.

Introduction

This report documents the method to determine the set of parameters that should be used in the module and the functional testing of the Lane Detection module. The purpose of the testing is to make sure that the module works as intended such as detecting the lane lines and that the next point given by the module is reasonable. In addition, the test checks the accuracy of the calculation of the difference between the vehicle center and the lane center and the difference between the next point and the lane center.

Equipment

- Suyin HD Camera
- White tape
- Measuring tape

Method

A lane is constructed by creating the left and right lane lines using white tape. The width of the lane should be equal at all regions on the lane. To determine a set of parameters that can be used to detect the lane in different positions, the camera is first placed in the middle of the lane. Afterwards, the parameters are slowly adjusted in the given order: Canny minimum threshold, Canny maximum threshold, Hough threshold, Hough minimum line length, Hough maximum line gap, and Gaussian blur. Each parameter is increased until it gives a satisfactory result.

After doing this, the camera is moved to another position and checked if both lane lines can be detected in the new position given this set of parameters. If not, then a single parameter is re-adjusted at a time until both lane lines are visible. The camera is thereafter placed in the previous position to check if the set of parameters is also acceptable in this position. This action is done for at least three positions and the resulting set of parameters is used on the following actions.

To check if the module detects the lane for all relevant positions, the camera lens is placed perpendicular to the lane. The starting relevant position is when the left wheel of the vehicle is on the left lane line. On the other hand, the ending relevant position is when the right wheel of the vehicle is on the right lane line. The rest of the relevant positions are positions between these two previously mentioned positions. The camera is therefore moved in between this range to check if the lane is detectable by checking if both lane lines are detected. In addition, the distance is measured from the left lane line to the center of the camera at different relevant positions. This value is then used to calculate the difference from the camera center to the lane center. The result from this calculation is afterward compared to the given value from the module. In order for the module to be able to compute the center difference value, the width of the place that is possible to be seen on the camera frame is measured.

To check if the next point given by the module is reasonable, the point is displayed on the camera frame. The point is then located physically in the real world. After locating the point, the distance is measured from the left lane line to this point and compared with the distance to the center of the lane.

Result

The set of parameters that were found using the given method is presented in the table 1.

Gaussian	**Blur**	12
Canny thresh.	**Min**	75
	Max	89
Morph. Closing	**Min**	2
	Max	2
Hough lines	**Thresh.**	61
	Min length	59
	Max gap	49

Table 1: Result of the set of parameters

By using the set of parameters given in Table 1, the result of checking if both lane lines are detected at different distances from the left lane line is presented in Table 2.

Distance from left [mm]	**Detected both lane lines**
52 (left wheel on left line)	Yes
89	Yes
130	Yes
144	Yes
175	Yes
189	Yes
216	Yes
225	Yes
267	Yes
312	Yes
355	Yes
365	No
409 (right wheel on right line)	No

Table 2: Result of detecting both lane lines using the set of parameters given in Table 1

The result of the calculated distances obtained from the module and the comparison between the calculated distances and the measured distances are given in Table 3.

Center diff. [mm]		**Deviation [mm]**	**Deviation [%]**
Measured	**Calculated**		
-55.5	-62.45	6.95	12.5%
-100.5	-92.89	7.61	7.6%
-141	-141.76	0.76	0.5%
-178	-167.76	10.24	5.8%
-14	-13.20	0.80	5.7%
36.5	21.17	15.33	42.0%
81.5	52.39	29.10	35.7%
110.5	80.87	29.62	26.8%

Table 3: Result of comparison between measured distance from the camera center and the calculated given from the module with a lane width of 461 mm and lane line width of 38 mm

The result of the next point given from the module compared to the lane center at different camera center positions is given in Table 4.

Distance from left [mm]			Deviation [mm]	Deviation [%
Cam center	**Lane center**	**Next point**		
175.0	230.5	216.0	14.5	6.3%
130.0	230.5	218.0	12.5	5.4%
89.0	230.0	220.0	10.0	4.3%
52.0	230.0	220.0	10.0	4.3%
216.0	230.0	220.0	10.0	4.3%
267.0	230.5	227.0	3.5	1.5%
312.0	230.5	231	0.5	0.2%
341.0	230.5	231	0.5	0.2%

Table 4: Result of comparison between the lane center and the given next point at different positions of the camera center

Discussion

The set of parameters given by Table 1 is acceptable to be used for further processes as a wide range of relevant distances can detect both lane lines by using this set of parameters. However, the desired range would be that when the right wheel was on the right lane line, both lane lines were still visible. The reason why both lane lines were not visible at positions after 355mm from the left line is that the camera's lens center was not centered in the camera frame. This led to the left side being seen more than the right side of the lens center. Therefore, when the right wheel was placed on the right line, the left line was out of the region of interest. This also caused a problem when using the module to calculate the offset from the lane center. The module was implemented to use the middle of the camera frame as the center. However, because the camera lens center was not centered in the frame, an offset that corresponded to the difference between the real center of the camera lens and the current center has to be added to the original center of the camera frame for it to match the logic behind this module.

From Table 3, it is possible to see that the deviation is only a maximum of 12.5% on the left side, which is represented with negative values. However, on the right side, which is represented by positive values, the deviation was is which is normally considered as high. The value of the difference is less than 30 mm, which is only 6.5% of the lane width given as 461 mm. The direction on which side the camera center was compared to the lane center was also precise. The high deviation was potentially caused due to the width of the lane line which was in this case 38 mm. The measured center difference was measured from the inside of the lane lines. However, in some cases, the detected lane lines might be in between the outer and inner lane lines instead, which can give it an extra offset of up to 38 mm.

From Table 4, the deviation in all cases is less than 10%, which is an acceptable deviation. It is therefore possible to conclude that when the two-lane lines are detected, the next point given by the module is reasonable to utilize to stay within the lane when driving.

Sources of Error

Possible sources of error in the test could for example be errors in the equipment like the measurement tape itself. This could have led to wrong values on the measurements. Another example is noise lines in the camera frame, which could affect the calculation of the lane lines. Secondly, a possible source of error could be a human error such as incorrect reading from the measuring tape that leads to wrong values on the measurements.

Conclusion

In conclusion, the lane lines was detected at most of the relevant positions. However, because the center of the camera was shifted to the left, there are some positions that ideally should be able to detect the lane lines that do not detect the lane lines. For the method of calculating the center difference from the module, the deviation was high compared to the measured value. Despite that, it is acceptable for identifying the direction of the difference considering that the reason behind the deviation was because of the width of the lane line. For the next point given from the module that is going to be used as the destination for the vehicle in path planning, the deviation was low and the method is therefore acceptable to use for further use cases. From the values measured and calculated, there existed possible source of errors that could have led to inaccurate values or uncertainties. Thus, a higher deviation in the calculations. The possible sources of error could be such as equipment failure and human errors.

Appendices H

Functionality Testing of Parking Slot Detection

Functionality Testing of Parking Slot Detection

Asbjørn Stokka (959810)
Eirik Reiestad (260753)
Julie Vy Tran (259195)

Tested: March 20, 2023
Revised: March 21, 2023

Summary

A set of parameters were found through a one-by-one modifying method. Thereafter, a test was performed to verify the function of the Parking Slot Detection module to detect parking lines only when a QR code is detected. From the test result, the functionality works as intended. Parking lines in the view are only detected when a QR code was detected and the two parking lines enclosing the QR code are correctly determined. In addition, it was observed that the ability to determine the two parking lines enclosing the QR code was dependent on the set of parameters.

Introduction

This report documents functionality testing of the Parking Slot Detection module. One of the purposes of the testing is to discover a set of parameters that should be able to detect the parking lines with the restriction to only when a QR code is detected. Another purpose is to determine if the module is able to extract the correct parking lines that enclose the QR code from all the perceptible parking lines at different angles.

Equipment

- Suyin HD Camera
- QR code
- White tape
- Black mat

Method

First, three coherent parking slots are constructed by using white tape as parking lines. The QR code is then placed in the middle parking slot as shown in Figure 1. Afterwards, the camera frame at different angles based on the camera placement is captured.

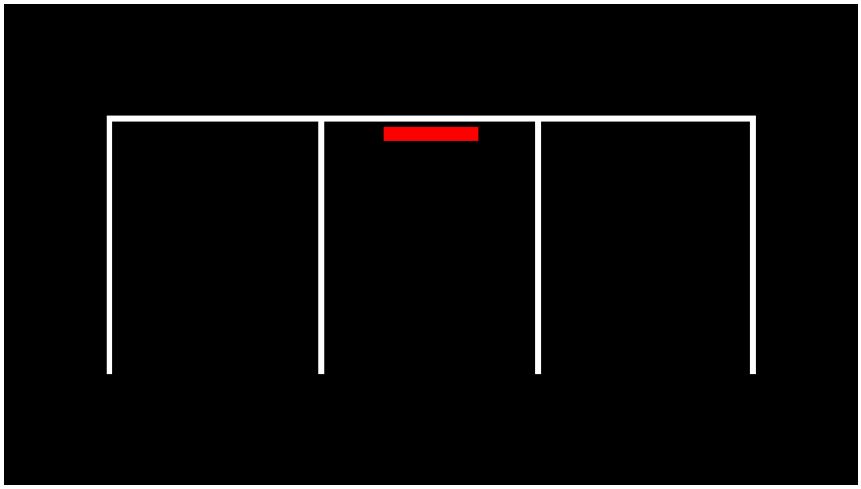


Figure 1: Three coherent parking slots, where the middle one that includes the QR code (red rectangle) is the desired slot to park.

At the angles which are within the relevant angles of detecting QR codes, these captured frames are used to determine a set of values to use as parameters for the module. The frame with the angle perpendicular to the camera is used as the initial frame with all the parameters set as 0. The parameters are thereafter increased in the order: Canny minimum threshold, Canny maximum threshold, Hough threshold, Hough minimum line length and Hough maximum line gap respectively. Each parameter is increased until a passable result is given before moving on to the next one. After finding a set of parameters for the perpendicular angle, these parameters are tested on the other frames with different angles. If the set of parameters also gives a passable result, then these parameters are used further for the next angle frame, else the parameters are increased in the same previously given order. If no set of parameters gives a passable result, the current set of parameters is decreased instead. Afterwards, these new parameters are used on the previously captured frame to check if it also gives a passable result for the previous captured. This procedure is performed for every captured frame. For the angles which are not within the relevant angle, the captured frames are used to confirm that no lines are detected.

Result

The set of parameters that resulted from the procedure mentioned in the previous section is presented in Table 1.

Gaussian	**Blur**	12
Canny threshold	**Min**	50
	Max	100
Morph. Closing	**Erode**	1
	Dilate	1
Hough line	**Threshold**	65
	Min Length	70
	Max gap	20

Table 1: Set of parameters for the Parking Slot Detection Module

From the set of parameters given in Table 1, the result from verifying if the parameters detect

available parking lines and extract the correct parking lines at different angles is presented in Table 2.

	Detected		
Angle [degrees]	**QR**	**Lines**	**Correct QR Code parking lines**
45	No	No	No
50	No	No	No
55	Yes	Yes	Yes
60	Yes	Yes	Yes
65	Yes	Yes	Yes
70	Yes	Yes	Yes
75	Yes	Yes	Yes
80	Yes	Yes	Yes
85	Yes	Yes	Yes
90	Yes	Yes	Yes
95	Yes	Yes	Yes
100	Yes	Yes	Yes
105	Yes	Yes	Yes
110	Yes	Yes	Yes
115	Yes	Yes	Yes
120	Yes	Yes	Yes
125	Yes	Yes	Yes
130	No	No	No
135	No	No	No

Table 2: Result of using the set of parameters from Table 1 to detect parking lines.

Discussion

As expected, based on the result from Table 2, lines were only detected when the QR code was detected. The set of parameters that were found in the test also operates adequately. Within the relevant angles for detecting QR-code, it detected the correct parking lines robustly.

In addition, the process of determining the set of parameters was time-consuming as a slight change of angle or in one of the parameters might cause the module to not be able to determine the correct two parking lines enclosing the QR code because of extra noise lines. The ability to detect the correct two parking lines was therefore highly dependent on parameter tuning. Although the set of parameters that were found through this test functions as intended. However, in different testing environments, new adjustments to the parameters are most likely required.

Sources of Error

Possible sources of error in the test could be the lighting, which can cause some parts of the lines to be less visible than others. This could lead to some parameter values being tuned more than it needs to. Another possible source of errors is the human error of reading the angle from the protractor.

Conclusion

In conclusion, the module works as intended through the set of parameters that were found in this test. The parking lines were detected when there was a QR code. Otherwise, when the QR code was not detected there will be no lines detected. In addition, when a QR code was detected by using the set of parameters, the correct parking lines enclosing the QR code were also obtained. However, the ability to detect the correct parking lines enclosing the QR code was quite dependent on the set of parameters.

Appendices I

Determining Mathematical Functions to Map an Object into a 2D Environment

Determine Mathematical Functions to Map Lines into a 2D Environment

Asbjørn Stokka (959810)
Eirik Reiestad (260753)
Julie Vy Tran (259195)

Performed: March 14, 2023
Revised: March 17, 2023

Summary

This section summarizes the findings of a test report aimed to establish the correlation between distance and pixel position in camera image frames. To achieve this, physical distances were measured, and relevant pixel positions were located on the image frame. Thereafter, regression analysis was performed to derive two mathematical functions to model this correlation. The first function represents the correlation between the distance in the y-axis and the pixel position in the y-axis of the point, which was modeled by a seventh-degree polynomial regression equation. The second function corresponds to the correlation between the ratio of millimeters per pixel on the x-axis and the distance on the y-axis, which was modeled by a linear polynomial regression equation. While some deviation is associated with both regression models, the size of the deviation is not critical. This test is based on the camera being placed 123mm above the ground.

Introduction

This report presents the results of determining the correlation between the distance from the vehicle to an object marked on the image frame for the method point_to_distance in the Environment module. The primary objective of this study is to establish a mathematical function that models this correlation in both the x-axis and the y-axis and determine the accuracy of the two mathematical functions.

The report discusses the methodology used to perform the testing and presents the findings of the study, including any uncertainties associated with the regression models.

Equipment

- Suyin HD Camera (1280 x 800)
- Measurement tape
- Marking tape

Method

This section outlines the methodology used to determine the correlation between the distance from the vehicle to an object marked on the image frame and the pixel coordinates on the x-axis and y-axis.

For the y-axis, the measurement tape is placed perpendicular to the camera lens with the start of the tape on the camera lens side. The marking tape is thereafter cut into pieces of reasonable sizes and placed at different distances on the y-axis from the vehicle. The distances to each tape are noted down. Next, the different tape pieces are located using a picture editor to determine their corresponding pixel positions on the y-axis using the camera frame. Using a graph tool, such as Geogebra, a regression model is then determined for the y-axis, where the x-values are the pixel positions and the y-values are the y-distances.

For the x-axis, two pieces of marking tapes are placed at each reasonable y-distance with a reasonable distance between them such that the pixels between the tapes in the frame do not overlap. This distance is measured and noted down. The pixel coordinates are then located and noted down for the two tapes at different y-distances, and the ratio of mm per pixel can then be calculated using the equation:

$$r_i = \frac{d_i}{|p_{i1} - p_{i2}|} \tag{1}$$

where p_{i1} and p_{i2} are the pixel coordinates in the x-axis and d_i is the distance in the y-axis in millimeters.

By using a graph tool, such as Geogebra, regression models are found by choosing the best-fitted model which in this scenario is defined as the model which passes through the most measurement points. Two models are found using this method. The first one is for the correlation between the distance per pixel ratio horizontally depending on the y-distance and the second is for the correlation between the distance in the y-axis and the pixel value vertically. The models are then implemented in the point_to_distance method.

To determine the accuracy of the two mathematical functions, random points are chosen within the camera frame and the distances to them are measured in both directions, where the origin of the coordinate system that corresponds to the distance in the x- and y-direction should be the camera lens. Thereafter, the point_to_distance method is used to obtain the calculated distances in both the x- and y-direction. These calculated distances are then compared with the measured distances to determine the accuracy of the method.

Result

The regression model of pixel and distance in the y-axis is given in Figure 1. This is a seventh-degree polynomial which is given by the equation below. The circles marked in Figure 1 represent the exact value that is measured.

$$\begin{aligned} f(x) = {} & 7.06075681116041 * 10^{-14} * x^7 - 7.07543466719694 * 10^{-11} * x^6 \\ & 2.84232325514655 * 10^{-8} * x^5 - 0.00000570571739397854 * x^4 \\ & + 0.000613607865916111 * x^3 - 0.0303482866336146 * x^2 \\ & + 1.55996291629615 * x + 334.781663853853 \end{aligned}$$

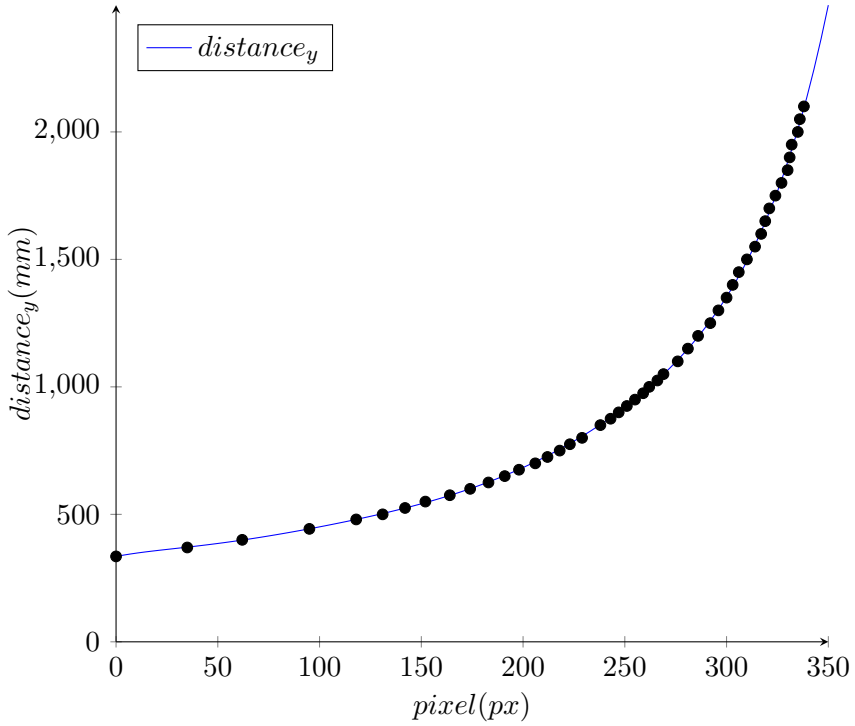


Figure 1: Regression model of pixel and distance in the y-axis.

Figure 2 presents the regression model for the distance ratio in the x-axis per pixel and distance in the y-axis. The equation for this model is given by Equation , and the circles in the figure represent the precise values measured that have been used to determine the regression model.

$$f(x) = 0.000807461x - 0.015133769$$

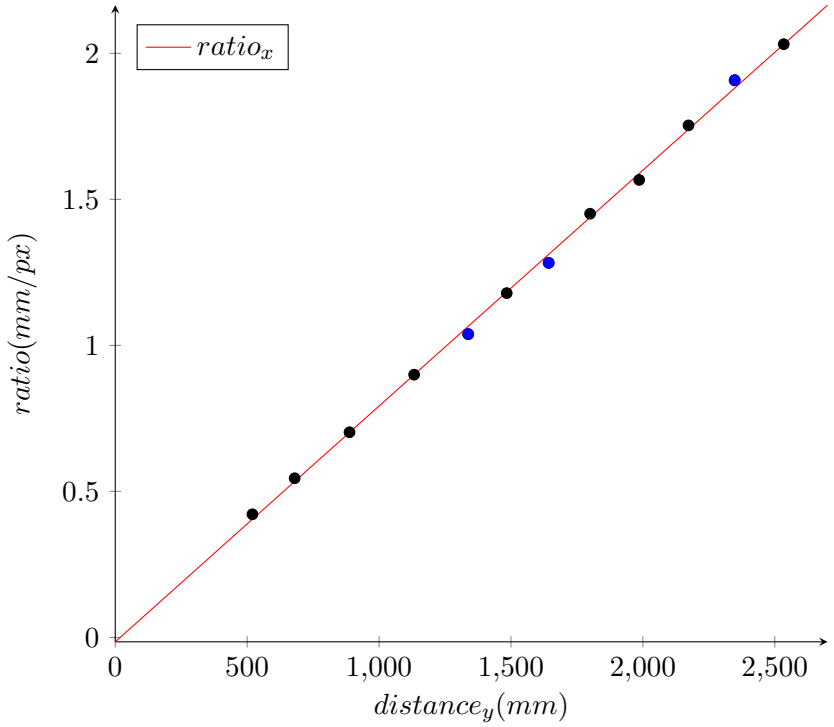


Figure 2: Regression model of the ratio of the distance in the x-axis per pixel and the distance in the y-axis.

The result of determining the accuracy of the two mathematical functions is given in Table 1.

Measured distances [mm]		**Pixel**		**Calculated distances [mm]**		**Deviation [%]**	
X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
232	1290	784	520	235.3	1131.4	1.4	12.3
-70	790	429	586	-63.0	738.3	10	6.5
-160	651	223	619	-157.0	620.7	1.9	4.7
22	413	607	744	23.2	392.2	5.5	5.0
-26	1023	505	545	-22.0	951.4	15.4	7.0
-195	1675	386	488	-191.9	1557.5	1.6	7.0
183	1403	711	509	184.4	1248.0	0.8	11.0
120	520	831	675	118.7	492.2	1.1	5.3
-140	439	142	712	-136.2	433.1	2.7	1.4
-163	1141	352	528	-160.7	1072.8	1.4	6.0

Table 1: Result of comparing measured distances to distances calculated by the method point_to_distance.

Discussion

Figure 1 shows that the model for the y-axis correlation passes through all of the measured points. This suggests that the model fits well with the data set. However, the model may provide incorrect distance values for pixel values larger than 350 since there are no measurement values in that range.

The model for the x-axis correlation in Figure 2 passes through 9 of the 12 points, which is approximately 75% of the points. While there may be some deviations in using this model since not all points lie on the model, the blue circles representing the points not traversed through are quite close to the line with a maximum deviation of only 0.03mm/px.

From Table 1 it is also possible to see that when applying the two mathematical functions for the method point_to_distance the maximum deviation is only at 12.3% for the y-distance and 15.4% for the x-distance. As it is less than 20%, the deviations are acceptable. However, as it is larger than 10%, one must be aware of when used.

Sources of Errors

There are several potential sources of error in this study. First of all, human errors during the measurement process could lead to inaccuracies in locating the different pixel values on the frame and thereby affect the regression models. Secondly, errors in the placement of the measurement tape and pieces of marking tape could also impact the accuracy of the measurements, particularly if the tape was not placed perpendicular to the camera lens or if the tapes were not positioned at equivalent distances along the y-axis. Finally, equipment errors, such as incorrect distances between measuring lines on the measuring tape or poor camera quality and camera calibration, could lead to blurry images or altered images. This could further lead to difficulties in accurately pinpointing pixel locations or an inaccurate constant ratio at the same y-distance because of lens distortion. In fact, the errors because of blurry images can be observed in the regression model of pixel and distance in the y-axis, where the range of pixel values is limited to only less than 400.

Conclusion

This study resulted in the discovery of two mathematical functions. The first is a seventh-degree polynomial that relates the distance on the y-axis to the y-value of the pixel coordinate. The second is a linear polynomial that relates the ratio of mm per pixel on the x-axis to the distance on the y-axis. However, there is a potential for deviation from accurate measurements due to various sources of error, such as human or equipment errors that causes uncertainties in the measurements.

Appendices J

Determine the Functionality of Mapping QR Code and Lines

Functionality testing of Mapping QR Code with Lines

Asbjørn Stokka (959810)
Eirik Reiestad (260753)
Julie Vy Tran (259195)

Performed: April 20, 2023
Revised: April 20, 2023

Summary

From the test cases, the QR code and the mapping of lines align with each other in that the QR code is enclosed within the parking lines. However, the result is restricted to these test cases and might not give the same result for other test cases.

Introduction

The purpose of the test is to determine how the mapping of QR code and the mapping of lines operate together. By determining this, it is possible to identify if path planning will work as intended. The desired result is that the QR code should be enclosed within the parking lines.

Equipment

- Suyin HD Camera
- QR code
- Protractor
- Measurement tape
- Marking tape

Method

A code implementation using the Pygame package is used to perform this test. The file is called example.py and is a part of the Pathfinding module.

Three parking slots are constructed using measurement tape. The QR code is then placed inside the middle parking slot, a little bit closer than the extra obstacle line of the parking slot but also not too far away from it. This is so that the QR code is still within the parking slot, but will not occupy the space the vehicle needs to be able to park in future tests. An example of how the QR code is placed is given in Figure 1 in the test report 'Functionality Testing of Parking Slot Detection'

The camera is afterwards placed at different angles and distances from the QR code. For each position, the code implementation is used to visualize if the QR code is enclosed within the parking lines of the middle parking slot.

Result

An example of the test result is presented in Figure 1.

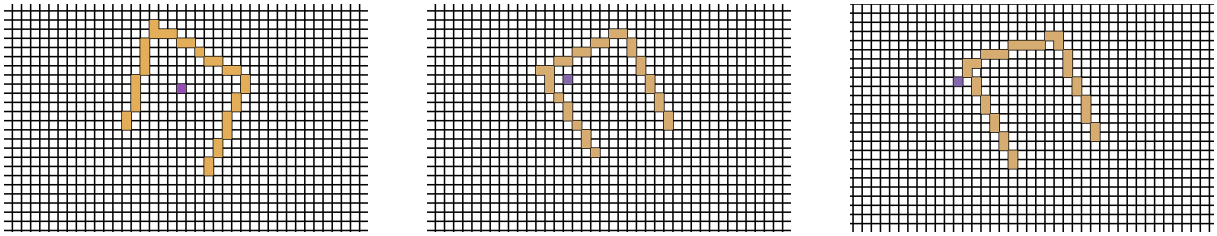


Figure 1: An example of the result from determining how the mapping of QR code and the mapping of lines operate together. Yellow lines represent the parking lines and the purple square is the QR code.

All test case results are given in Table 1.

Angle	**Distance**	**Enclosed**
-35	630	X
-22	652	-
0	600	X
22	770	X
35	735	X

Table 1: Result of all test cases to determine how the mapping of QR code and the mapping of lines operate together. The column 'Enclosed' describes if the QR code is enclosed in the parking lines or not. X means it is enclosed. Dash means it is not enclosed.

Discussion

From Table 1, it is possible to perceive that the QR code and the line mapping align with each other in 4 out 5 cases by that the QR code was enclosed in the parking lines. However, 1 of 5 cases resulted in the QR code being outside the parking slot. Furthermore, as seen in the first result in Figure 1, the QR code was mapped a bit far away from the obstacle line. On the other hand, the second result showed that the QR was close to the obstacle but closer to one side of the parking slot. Further, the third result showed an outcome where the QR was outside of the parking slot. Thus, it is possible to confirm that there are some deviations in the desired and obtained results. Overall, as these are only a few combinations of angles and distances, there might exist more scenarios where the QR code mapping and the parking lines mapping do not align with each other.

Sources of Errors

A possible source of error in the test cpuld be the placement of the QR code relative to the parking slot. The QR code could have been placed too far away from the obstacle line, thus there might exist a possibility that for example, the QR code in the edge cases like Figure **??** would be behind the obstacle line. However, since it was placed a little bit further away than the ideal amount from the obstacle line, it was observed that it is still within the parking slot.

Conclusion

In conclusion, from the test cases that have been performed, the QR code was enclosed within the parking lines in 80% of the test cases. However, there are some deviations from the desired position of the QR code in relation to the parking slot. In addition, as the test cases do not include all possible combinations of distance and angle, there is a possibility that this will not perform well in other test cases.

Appendices K

Camera Latency, Motor speeds, and Software frame rate tests

Camera latency, motor speeds, and software frame rate

Asbjørn Stokka (959810)
Eirik Reiestad (260753)
Julie Vy Tran (259195)

Performed: April, 2023

Summary

This section summarizes the findings of a test report for a bachelor thesis that aimed to quantify the latency of a selection of different cameras, PWM output vs motor speed of the gearbox motor, and current software performance in FPS.

Introduction

This report presents the results of testing the camera and motor hardware used and is aimed to give us quantified data on the performance. For the cameras, specifications on latency, resolution, and maximum FPS are relevant. For the motor system maximum and minimum speed is relevant, as well as check that they can handle the torque of driving the car. And, lastly, check the performance of our software running on the Raspberry Pi single-board computer. Even though the performance of the system is in general not important in regards to the accuracy of the computer vision algorithms, it is relevant for deciding what speed the test vehicle can safely operate at. An evaluation of these tests and evaluations will give important information on the physical handling of the test vehicles.

Equipment

- Logitech Camera (1920 x 1080)
- Suyin HD Camera (1280 x 800)
- Raspberry Pi camera
- Gearbox brush motor
- Motor driver board (H-Bridge)
- NEMA 17 stepper motor
- TMC2209 Stepper drivers
- Raspberry Pi 4B 1GB
- Desktop test vehicle
- Small test car

Camera Latency

Method

For testing camera latency two approaches have been used. First using a Python script that measures the time it takes from the script to change the screen color, until the camera notices this change. The script used for this was downloaded from: https://www.dlology.com/blog/how-to-measure-the-latency-of-a-webcam-with-opencv/

Secondly, the camera latency was approximated by filming the computer showing a millisecond timer, while also showing the camera output in a window next to it. At a random time, a screenshot was taken, and the latency was measured as the difference between the time shown in the timer window and the time shown in the camera output window.

Result

Camera latency was measured differently from the two approaches. Both indicated a camera latency between 100 and 200 ms. We can assume a worst-case latency of 200 ms.

Motors and torque

Method

Gearbox 1:48 brush motor

The datasheets and product information gives the following specifications (https://www.aliexpress.com/item/33017067692.html & https://www.adafruit.com/product/3777):

- Operating voltage: 3V 6V DC
- Maximum torque: 800gf cm min (when it is 3V)
- RPM:200(when it is 4.5V)
- Load current: 170mA (when it is 4.5V)
- Min. Operating Speed (3V): 90+/- 10% RPM
- Min. Operating Speed (6V): 200+/- 10% RPM
- Stall Torque (3V): 0.4kg.cm
- Stall Torque (6V): 0.8kg.cm
- Weight: 30.6g

Stepper motor

- 1.8 degrees per step
- 200 steps per revolution
- Desktop vehicle Wheel circumference: 213,52 mm (based on the outer diameter)
- Desktop test vehicle moves approximately 1 mm/step (0,94 mm)
- Small car wheel circumference 1318,8 mm (based on the outer diameter)
- Small car moves approximately 6,594 mm/step.

Weight measurements

Battery type	Battery [g]	Desktop test vehicle weight [g]	Total [g]
Gel	1 148	481	1 629
LiFePO4	491	481	972
Drone Battery	182	481	663
None	0	481	481

Results

The total weight of the desktop test vehicle led to high torque on the gearbox motor. At low voltages and slow speeds, the gearbox motor performed unevenly. This was noticeably better with the light batteries. Even with the lightest battery, the desktop test vehicle would not drive reliably enough to keep track of position and direction without additional sensors.

The TMC2209 stepper motor drivers allow for micro-stepping. Depending on the physical MS1/MS2 pin configuration the microsteps can be 8th, 16th, 32th, or 64th of a step. The frequency will need to be adjusted accordingly.

Wheels and frequencies for stepper motor

Method

By calculating the circumference of the wheels, and number of steps needed for a rotation of the motor it is possible to calculate the vehicle speed. The following table shows the needed PWM frequency needed for the stepper motor controller to achieve a given velocity of the vehicle. The values in the table has to be multiplied according to the micro-step setting of the motor controller as well.

Results

Frequency	**0.1 m/s**	**0.2 m/s**	**0.5 m/s**	**1 m/s**	**1.67 m/s**
Desktop test vehicle	106.38	212.77	531.91	1063.83	1 776.6
Small test car	16.13	32.27	80,67	161,33	269.43

Software performance

Method

Software performance was tested by running the output of the number of frames it calculated per second. These values can be found by downloading and running the Python software on a Raspberry Pi 4b.

Results

The socket camera streaming running on its own without any calculations gave an output of 5 FPS, counted on the remote computer receiving the frames on a wired network connection. And

depending on the running state it would run at approximately three to five FPS with the serial communication module, and up to 10 with the hardware PWM communication module.

Discussion

Note that 30 FPS is unrealistic when performing calculations on a Raspberry Pi, but it is the limit of what the cameras used in our project are capable of. In a production setting, it is possible to swap to cameras with higher frame rates and lower latency. In addition to the latency of the camera, the frame rate is a factor in how quickly the car will react to an obstacle or other visual feedback. Depending on the frame rate the car will move a distance each time a new image is processed, and new calculations are made. If any obstacles appear, this could add to the reaction time of the autonomous vehicle, and its chance to stop or turn to avoid a collision. And it will also affect how well it will be able to drive a course. The table above shows some of the relevant speeds and frame rates, and a calculation of how far the car will move between getting a new frame for calculations. It's worth noting that if an obstacle were to appear immediately after one frame was taken, it would be almost two frames (twice the distance) until the car acts on it, as it has to process the frame before any change is made to the motor output. Realistically it is expected to get up to around 5-10 FPS, one might want to avoid driving more than 10 cm between the moment a frame is taken to the car and acts on the new information it has retrieved. On a larger scale and given other sensors that can help detect obstacles it might be safe to increase this distance driven between frames/calculations.

	0.36 km/h	**0.72 km/h**	**1.8 km/h**	**3.6 km/h**	**6.0 km/h**
FPS	**0.1 m/s [m]**	**0.2 m/s [m]**	**0.5 m/s [m]**	**1 m/s [m]**	**1.67 m/s [m]**
1	0.10	0.20	0.50	1.00	1.67
3	0.03	0.06	0.17	0.33	0.56
5	0.02	0.04	0.10	0.20	0.33
10	0.01	0.02	0.05	0.10	0.17
15	0.007	0.013	0.03	0.07	0.11
30	0.003	0.007	0.016	0.03	0.06
Latency					
100 ms	0.01	0.02	0.05	0.10	0.17
200 ms	0.02	0.04	0.10	0.20	0.33

Sources of Errors

The time measurements done with a stopwatch are somewhat unreliable in nature and were also highly affected by the car not driving in a straight line.

Conclusion

From the set of tests performed in this report, it seems beneficial to change from gearbox brush motors to stepper motors as the accuracy of movement is essential. The frame rate and latency of the tested cameras, including the performance test with the raspberry pi suggest that it is possible to run the vehicle at speeds up to 6 km/h. However, slower speeds are advisable, and with stepper motors, just as reliable.