Autonomous Robot for Pavement Cleansing

A Personal Project

Christian Lykke Jørgensen

Lykke - Be Happier

Department of Electronic Systems
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:

Abstract:

Howdy

Project:
Autonomous Robot for Pavement Cleansing

Project Period:
Fall 2024

Project Group:

Participants:
Christian Lykke Jørgensen

Supervisor:
Kirsten Mølgaard

Page Numbers: 158

Date of Completion:
December 9, 2024

Contents

Preface

This report searches to develop a Roomba-like autonomous robot capable of exterminating weeds in pavement for private users.

Aalborg University, 19-12-2023

Christian Lykke Jørgensen
cljo22@student.aau.dk

1 Introduction

Every homeowner with some variation of pavement has a common thorn in their eye; weeds, moss, and grass. It is a prevailing problem experienced by homeowners, city councils, public buildings, etc. Currently, the only "easy" solutions are harmful to the pavement, water reservoirs and/or release a wide range of greenhouse gasses. Apart from their environmental impact, most of these methods are fairly time-consuming and labor-demanding.

Without a doubt, a better solution is needed. However, these must first be examined and analyzed in depth to understand how to improve upon existing solutions. The individual environmental impact of each solution must be examined before a new approach can be proven a better solution. Therefore this project's initial problem statement can be expressed as:

"What is the currently most efficient method for removing unwanted plants and plantlike material from pavement, and what are its environmental impacts?"

2 Problem Analysis

2.1 Existing Solutions

Before a new solution can be developed, current methods must be investigated, to understand their benefits and shortcomings. Each method will also be evaluated regarding its environmental impact and average time used to clean one square meter of 14x21cm "herregårdssten". The environmental impact will be a subjective scale of "small/medium/large" based on public guidelines set by Miljøstyrelsen and public consensus from institutions such as Bolius, Taenk, and Idenyt. Aforementioned guidelines and consensus can be accessed at: [1, 2, 3, 4, 5, 6, 7, 8, 9]. The average time used to clean one square meter of pavement will be based upon numbers from appendix A.1 on page 96 containing personal data collection from using different methods in my own driveway. Furthermore, a secondary number based upon Miljøstyrelsens numbers from their paper "Ukrudsbekæmpelse på Belægninger" will be available in cursive for some methods, [1].

2.1.1 Manual Pavement Cleaning

Manual pavement cleaning refers to dragging a stick such as seen in figure 2.1 through every groove between the individual paving tiles. This is an incredibly time-consuming task, as it requires the user to cleanse every mm of groove by hand. However, it is rather effective for cleaning pavement and is one of the most popular modes of cleaning pavement for private users.

Figure 2.1: Manual brush for cleaning pavement, [10].

Pros	Cons
Effective	Hard labour
Thorough	Not efficient
	Time consuming

Environmental Impact	Average Minutes Used to Clean $1m^2$
Small	1.554

Table 2.1: Pros and cons of manually cleaning the pavement.

2.1.2 Pressure Washer with Patio Cleaner

Using a pressure washer with a patio cleaner is one of the most effective methods of removing anything unwanted between tiles in the pavement. Unfortunately, it is almost

impossible not to remove the wanted parts as well, as the washer jets are strong enough to remove the sand between the tiles while removing weeds and algae. A positive byproduct of using a pressurewasher correctly on pavement is, that the pavement is cleaned and most of the time given a "new" look. This only occurs if and when the pressure washer is used correctly, as incorrect use can damage the pavement to such an extent, that it has to be replaced.

After treating pavement with a pressure washer, it will need to be re-sanded, and in some cases will benefit from having preservatives added, to discourage algae growth in the future. This part of the process is rather expensive, as new sand has to be bought, and the preservative coating has to be distributed correctly. Another downside to using preservatives is, that some products still contain substances, that are damaging to the environment

Pros	Cons
Effective	Need to refill with sand afterwards
Thorough	Expensive (new sand)
Quick execution	Dirty
Very easy	Loud
	Damages the pavement
Environmental Impact	Average Minutes Used to Clean $1m^2$
Small - Medium[1]	5.197, *0.2, without refilling sand*

Table 2.2: Pros and cons of pressure washing the pavement.

2.1.3 Rotating Steel Brush

Like with manual pavement cleaning, using steel brushes is quite effective at removing unwanted plants. There even exist motorized versions, minimizing the amount of manual labor necessary, and further improving the effectiveness. However, steel brushes are prone to damaging the surface of the pavement, especially if used incorrectly or if it is overused. Apart from the risk of damaging the pavement, steel brushes still require a lot of manual labor to be an effective method for removing unwanted plants in the pavement. Furthermore, cheap steel brushes could lose bristles while being used, leaving steel wires in the vicinity of the cleaned area, which will eventually create rust stains if they are not removed. At a larger scale products such as the Kwern Greenbuster exist, but are generally aimed at professional use rather than the average homeowner.

Pros	Cons
Thorough	Hard labour
Easy to execute	Not efficient
	Time consuming
	Damages the pavement
	Possible rust patches from damaged wires
Environmental Impact	Average Minutes Used to Clean $1m^2$
Medium	1.479, *0.12*

Table 2.3: Pros and cons of cleaning the pavement with a rotating steel brush.

2.1.4 Chemicals

Chemicals are sadly one of the easiest and most widely used methods for handling unwanted plants in pavement. The chemical glyphosate found in many weed-removing solutions is illegal for private people to use. Apart from commonly approved chemicals, it is fairly common that homeowners use a mix of water and salt, vinegar, or other chemicals found in the cleaning cabinet. These homebrewed elixirs and remedies are in most cases more damaging to the environment than people think. Salt and vinegar is so harmful, that they're illegal to use for weed extermination in Denmark.

Even though most of the chemical solutions do a good job at killing the plants, they have to be applied several times over such long periods of time, that they are practically inefficient compared to manual removal. Furthermore, solutions such as salt is damaging to the water reserves and could possibly ruin the pavement as well, leading to premature replacement of the pavement.

Pros	Cons
Effective	Not efficient
	Damage to natural water reserves
	Damages the pavement
Environmental Impact	Average Minutes Used to Clean $1m^2$
Large	0.496 * number of passes for the plant to perish

Table 2.4: Pros and cons of using chemicals to clean the pavement.

2.1.5 Sweeping

The easiest way of keeping pavement free from weeds is to not let the weeds settle and sprout. By this, it is meant as sweeping the pavement at least once a week, if not more often, to disrupt any seeds settling into the crevices, and if any succeed, stressing them by continuous sweeping. However, this method is most effective at the early stages of any weed's life cycle.

Pros	Cons
Thorough	Hard labour
Easy	Not efficient
	Time consuming
	Mostly efficient against new weeds
Environmental Impact	Average Minutes Used to Clean $1m^2$
Small	0.346

Table 2.5: Pros and cons of sweeping the pavement.

2.1.6 Weed Burning

Burning weeds is the prevalent method used in private homes, professional cleaning services, and public institutions. The reasons being ease of use, low labor commitments, and instant results (if used wrong). When it comes to burning off weeds, most solutions

have a large area of effect, which may not be optimal. In some cases, the effective area is quite a lot larger than the greenery being burned off.

Pros	Cons
Barely any labour	Fire
Very efficient if used correctly	Not efficient if used wrong
Easy	Time consuming
	Needs several passes to kill the plant
Environmental Impact	Average Minutes Used to Clean $1m^2$
Large	1.376, *0.12* * number of passes for the plant to perish

Table 2.6: Pros and cons of burning off weeds.

2.1.7 Laser-weeding

Laser-weeding is mostly known in agriculture, and still such a new concept that it has barely got a foothold. The first commercial laser weeding solution is made by Carbon Robotics based in Seattle and was launched during the summer of 2023, [11]. Their flagship product, the "LaserWeeder" is a carriage pulled by a tractor, with 30 150W CO2 lasers, 12 high-resolution cameras, and the capability of killing up to 300.000 weeds every hour, [12]. The "LaserWeeder" is currently the only commercially available technology designed to kill weeds with lasers, while WeedBot and WeLaser are alternatives still in the prototype stages, [13, 14]. Nevertheless, the effects of using lasers to kill weeds are being examined to a great extent across different use cases and under different circumstances. In general, laser weeding is scientifically approved as a concept, but is constrained by several factors such as limited knowledge regarding the long-term effects of using laser on plants and affiliated subjects, such as insects being hit as a byproduct, [15]. Another constraint regards machine learning and the current stage of artificial intelligence, which is mostly relevant for agricultural use where a machine has to discern between plants in different growth cycles and in different substrate compositions, [16]. It is consensus that laser-weeding is most efficient early in the growth cycle, especially at the cotyledon stage and two-leaf stage, [16, 17, 18]. A final constraint is the lack of established safety procedures following new technology. Even though lasers by no means are new technology, equipping autonomous robots with lasers capable of damaging organic matter, is quite new.

Pros	Cons
Barely any labour	Laser
Very efficient if used at optimal growth stages	Not as efficient on established plants
Easy	Expensive
Autonomous	Needs several passes to kill the plant
Fast	
Very eco-friendly	
Environmental Impact	Average Minutes Used to Clean $1m^2$
Small	0.0074^2

Table 2.7: Pros and cons of burning off weeds.

2.2 Private Autonomous Robots

Observing pavement de-weeding as a chore that has to be done, makes it possible to observe other chores that have been automated in a private home. Autonomous lawn mowing and Roombas are common in increasingly more homes, as they undertake a fairly simple, but time-demanding chore.

Autonomous lawn-mowing robots have evolved from bumping into everything and getting stuck in tall grass, to being adaptable and fit for almost any garden. Top-of-the-line robots is fit with GPS coordination, Bluetooth, rain sensors, four-wheel-drive, and batteries large enough for more than a thousand m^2 per charge. Combined with the newest advancements within the control of autonomous robots, they are capable of obstacle avoidance, rain-detection, optimizing routes, and dividing a lawn into multiple zones, including "no-go" zones, all within perimeter cables or other physical "restrictions". An example could be the LUBA 2 AWD 5000, which has all of the above features, [19].

Changing the focus to indoor use near people, pets, and other predicaments, robotic vacuum cleaners have advanced a lot as well. As with autonomous lawnmowers, top-of-the-line vacuum cleaners has evolved from "simple" robots bumping into everything and getting stuck in socks, to an "intelligent" robot. A Roomba Combo 10 Max has many of the same features as the LUBA 2, but with the addition of more advanced AI, capable of categorizing rooms, changing settings to fit a certain cleaning task, and schedule cleaning to fit a lifestyle, such as cleaning the kitchen after dinner each night, [20].

Both system types have integrated safety systems. Such a system could be the automatic blade-stop on the LUBA 2 or the scrub-stop in the Roomba, where it shuts down operation if a sudden change in slope or other suspicious movement is detected. Another safety feature available in both systems is obstacle avoidance. As neither system has to bump into objects before a change in direction is done, the chance of them tipping stuff over or hitting a person is minimized drastically.

Both systems are made to continuously do a simple task, in a semi-static environment near objects and beings. This draws several similarities to the task of cleaning pavement from weeds, and it is possible to draw inspiration from both areas, especially if they could be paired with the autonomous weed-killing available in the "LaserWeeder".

2.3 Problem Statement

Based upon current existing solutions, none of them pose as the most efficient method, without several drawbacks having varying importance dependent on the end-user. Returning to the initial problem statement:

"What is the currently most efficient method for removing unwanted plants and plantlike material from pavement, and what are its environmental impacts?"

The currently most efficient methods available to private users are pressure washing or burning the weeds if the determining metric is time used to remove the weeds (using Miljøstyrelsens numbers). Looking at personal experience with cleaning pavement of weeds, the most effective method is sweeping. However, sweeping pavement clean requires constant cleaning routines, rather than fewer routines of higher intensity for extended periods, such as manual cleaning. If the prosperity of the pavement is not important, faster results can be achieved by pressure washing or using a rotating steel brush. Chemicals are the sore thumb when it comes to common methods, as they are easy to use and mostly very effective, but ecologically not sound, as most users tend to overuse the chemicals.

This leaves burning the weeds and laser-weeding as the remaining methods. As both methods similarly stress the plant, another metric to determine superiority has to be used. Using price, the simple gas-burning solution is far superior, at least on initial cost, but using cost will eventually become larger than the combined acquisition and use cost for a laser system, based upon current gas and electricity prices. If time spent matters most, the possible autonomy of a laser-based system far outperforms a gas-burning solution. As shown by companies such as Carbon Robotics, it is possible to make a system capable of identifying weeds and discerning between different plants. Now, their solution is driven by a tractor and far from a private house-owner use case, but what if it could be adapted to fit the needs of a private individual? Combining the laser-weeding element with known solutions within lawn care such as the LUBA 2 AWD 500 and the iRobot Roomba Combo 10 Max could potentially be a solution, resulting in the following problem statement:

"How can an automated laser-weeding robot be developed to fit the needs of a private user wanting to clean their pavement from weeds?"

However, there is a catch regarding this, as only one person is working on this project. To make it more reasonable, the problem statement is reduced significantly. Nevertheless, the demand specification and general idea will still regard a complete system, until the technical analysis and system design is started, whereafter the project scope will be limited to fit the final problem statement:

"How can an automated pavement-following robot platform be developed?"

3 Demand Specification

3.1 Limitating the Project

As this project only stretches for a single semester and is being done by a single student, some limitations must be made. Instead of focusing on developing a full system meeting a large number of demand specifications from the start, the project will be divided into iterations - A demand specification for a full system is available in the appendix at sections B.7 and B.8. Therefore each iteration will have its own distinct goal(s) and specifications that it should meet. The first iteration will be the minimum viable product (MVP) and goal of this project. Further iterations will contain increasingly advanced features not described in depth in this project.

3.2 High Level Specification

This section describes a high-level overview of the functionality desired for the pavement cleansing robot. It is written as seen by a private person wanting to ease cleaning their pavement. Detailed specification can be found in section 3.3 starting page 9.
As a house owner looking to ease removing weeds from my pavement, I want:

- *An autonomous robot capable of moving around.*
- *A charge point/home at which the robot can dock when not in use/when it has to charge.*
- *An autonomous robot that updates me of its whereabouts and I can call "home" in case it is in my way.*
- *An autonomous robot that returns "home" before the battery dies.*
- *An autonomous robot that systematically cleans my driveway and other pavement, and therefore does not just bump around like a Roomba.*
- *An autonomous robot that detects cars, outdoor furniture, and the like, so that it will only operate around objects where it is safe.*
- *An autonomous robot capable of mapping what parts of the driveway that has been cleaned.*
- *An autonomous robot avoiding obstacles without bumping into them.*

A high-level specification for a complete system can be found in appendix B.7 on page 143 along with a functional specification in appendix B.8 on page 144.

3.3 Functional Specification

This section describes the functional criteria of the product. The criteria are seen from an end-user perspective and made as user stories, where accept criteria (AC1 & AC2 e.g.) must be fulfilled. A test of the functional specifications is made in section 7.

3.3.1 Operate the Robot

As a house owner, I want an autonomous robot capable of moving around.
Accept Criteria:
AC1:
The robot should be able to drive forward.
AC2:
The robot should be able to drive backward.
AC3:
The robot should be able to turn around its own axis (Z-axis).
AC4:
The robot should be able to follow a line in the pavement.

3.3.2 A Home for the Robot

As a house owner, I want a charge point/home at which the robot can dock when not in use/when it has to charge.
Accept Criteria:
AC1:
The charge point should be able to contain the robot.
AC2:
The robot should be able to communicate with the charge point.

3.3.3 User-interface

As a house owner, I want an autonomous robot that updates me of its whereabouts and I can call "home" in case it is in my way.
Accept Criteria:
AC1:
At all times the robot should broadcast its whereabouts.
AC2:
If the robot is in the way, a user-interface should enable me to send it "home" or to another area.

3.3.4 Robot Go Home

As a house owner, I want an autonomous robot that returns "home" before the battery dies.
Accept Criteria:
AC1:
At all times enough battery power is left to drive "home" + 10% extra distance, meaning that a 20-meter travel home, requires power for at least 22 meters.
AC2:
The robot should be capable of mapping a route "home" with obstacles such as corners, cars, and lawnchairs added, to accommodate non-direct routes.

3.3.5 Systematical Procedure

As a house owner, I want an autonomous robot that systematically cleans my driveway and other pavement, and therefore does not just bump around like a Roomba.
Accept Criteria:
AC1:
The robot has to map out all paved areas.
AC2:
A systematic approach following lines in the pavement has to be used.
AC3:
If several types of pavement are present, different areas must be mapped to distinguish and optimize routes for each area.

3.3.6 Spatial Awareness

As a house owner, I want an autonomous robot that detects cars, outdoor furniture, and the like, so that it will only operate around objects where it is safe.
Accept Criteria:
AC1:
The robot must be capable of determining whether or not, it can fit within a space.
AC2:
The robot must keep a minimum clearance of 10cm to anything above, so as to not wedge itself beneath anything.

3.3.7 Memory Capabilities

As a house owner, I want an autonomous robot capable of remembering what parts of the driveway have been cleaned.
Accept Criteria:
AC1:
The robot must map out which areas have been cleaned at which point, to rotate between areas.
AC2:
The robot must be capable of increasing its speed when no greenery is present.

3.3.8 Obstacle Avoidance

As a house owner, I want an autonomous robot avoiding obstacles without bumping into them.
Accept Criteria:
AC1:
The robot must not hit anything to change its course.
AC2:
The robot must not be closer than 5cm to anything in any direction, other than the pavement below it.
AC3:
The robot must not drive off of a ledge and tumble down.
AC4:
If the robot cannot turn around its own axis, it must reverse out from its current spot.
AC5:
If presented in a corner with no way out, the robot must turn off and notify the owner.

3.4 Electrical Specification

The electrical specification is currently based on preliminary assumptions and will remain flexible until a comprehensive technical analysis and system design is completed. The following initial specifications are proposed, based on similar systems and available reference manuals:

- Battery Capacity: 10 Ah (capacity may be adjusted based on measured energy consumption requirements).
- Battery Voltage: 12-18V (for compatibility with motors and auxiliary systems).
- Operating System Voltage: 3.3V (suitable for microcontrollers and low-power electronics).
- Motor Voltage: 12-24V (standard voltage for robotic drive motors).
- Auxiliary Equipment Voltage: 12V (for components such as cooling fans, lights, etc.).
- Sensor Voltage: 3.3V (common voltage for environmental and navigation sensors).
- Charging System Voltage: 24V (for rapid charging circuits, depending on battery chemistry).
- Power Consumption: Estimated at 200-250W during peak operation (including motor, sensor, and laser operation).
- Communication Voltage: 3.3V or 5V (for wireless modules such as Wi-Fi, Bluetooth, or LoRa).
- Power Management Unit: 12-18V/5V/3.3V DC-DC converters to manage voltage distribution efficiently across different subsystems.

These specifications are informed by reference data from various RC projects, the motors mounted to the prototype given by AAU, and general knowledge of electrical systems, [21, 22]. Similar robotic platforms such as autonomous lawnmowers and vacuum cleaners generally operate at 12V with battery capacities ranging from 2.8-8.8Ah, [19, 23]. Future adjustments will depend on detailed load analysis and testing.

4 Technical Analysis

In an effort to analyze the technical needs of the project, the demands set for a prototype have been revisited. This identified the following areas of interest:

Area of Interest	Associated Functional Specification
General Control of Autonomous Robots	Operate the Robot
Sensors	Spatial Awareness, and Obstacle Avoidance
Computer Vision	Systematical Procedure, Obstacle Avoidance, and Memory Capabilities
Mapping	A Home for the Robot, Systematical Procedure, Robot Go Home, Memory Capabilities, and Obstacle Avoidance
Power Monitoring	A Home for the Robot, Robot Go Home and Memory Capabilities
Wireless Communication	A Home for the Robot, User-interface, Robot Go Home, and Memory Capabilities

Table 4.1: Overview of which areas will be covered in the technical analysis and their relation to various functional specifications.

With this overview, it is possible to analyze large parts of the system, before implementing the knowledge into the system-design phase. It should be noted that machine vision has been chosen over other methods for analyzing surroundings, as the environment in which the platform will operate is natural, and therefore poses situations where "simpler" methods have been deemed insufficient. Moreover, vision-based navigation is a passive method, whereas lasers, sonar, IR, etc. are active methods, possibly "altering" the environment by introducing waves or light, [24]. With increased computing power, a vision-based system could also extract far more information, than most unifications of other sensors, cheaper and more reliably, provided an effective algorithm is in place, [24].

4.1 Preliminary Solution

The technical analysis will be based on a preliminary solution, wherein demands, solutions, sensors, basic operations, and the like are already considered. This preliminary solution forms the basis for analyzing and developing an optimal system. The preliminary system will have some options chosen subjectively, as the option most likely will be needed/form the basis for a final system in future development. Therefore, some options/technical analysis elements might be overkill for developing a prototype matching

the problem statement in section 2.3.

The preliminary solution is a vision-based autonomous tracked vehicle, with distance sensors acting as backup "safety stops", in case the computer vision misinterprets its surroundings. Furthermore, the preliminary solution is approximately the size of a Roomba, and capable of roaming outdoor areas, distinguishing between pavement styles, asphalt, grass (as in a lawn), flower beds, and stairs. Moreover, the solution must be able to recognize patterns present in the surface it is traversing, to optimize path planning. On a final note, the system should also be able to map the area it is operating in, remembering obstacle placements and identifying areas of interest, such as weed placement.

The system shown in figure 4.1 is imagined to achieve all of the above. In the figure, it should be noted that the system has many sensors in the forward direction and barely any in reverse. The reason for this is that the system will mainly operate moving forward, mapping out its surroundings based on the images acquired by the camera. The distance sensors in front will be responsible for ensuring the robot does not hit anything misinterpreted by the camera, and the rear sensor is meant to expose if any obstacles not mapped have appeared behind the system. To monitor that the system is moving, encoders connected to idler wheels will be attached as well. To create a "home" for the system, a charge point is added externally, which the system will connect to by some sort of wireless communication. Apart from being a "home" the charge point will also act as the user interface, updating the user with battery percentage, whereabouts, etc.

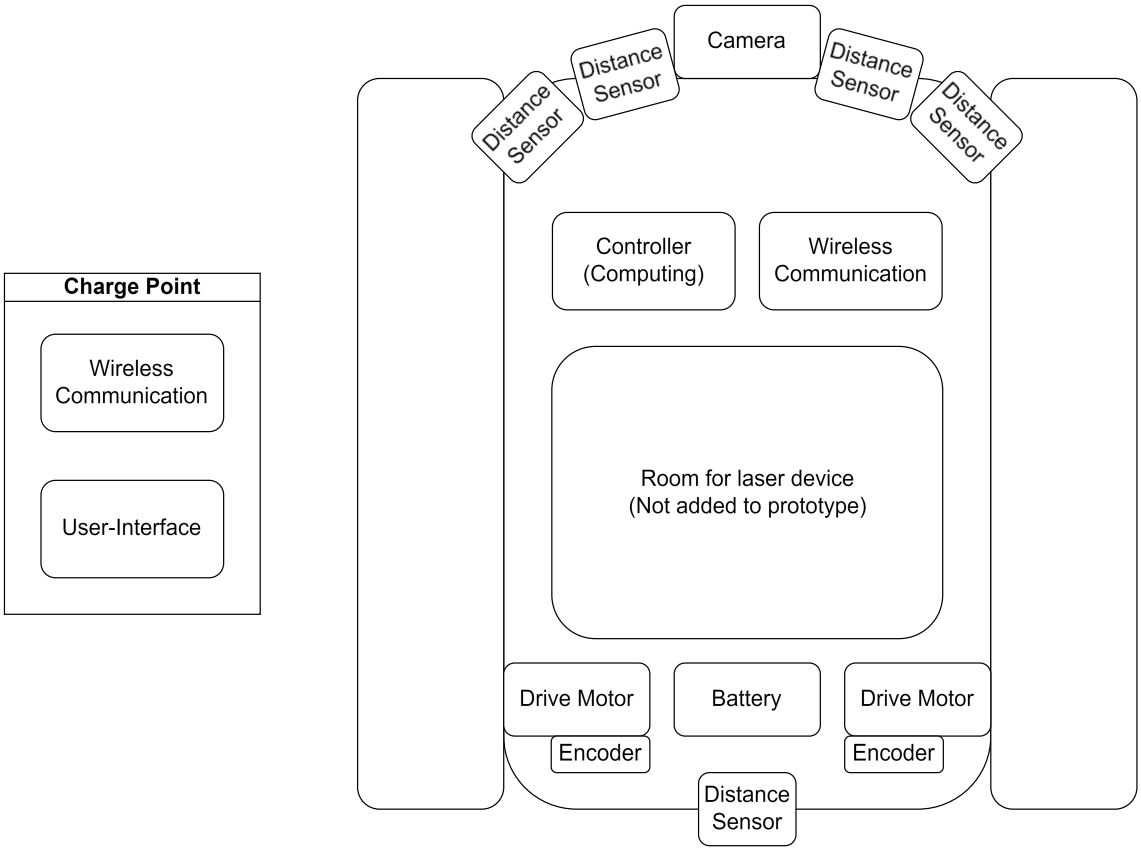
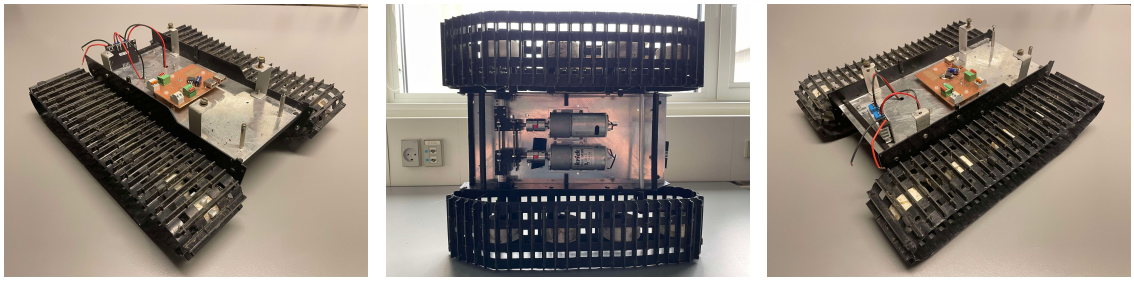


Figure 4.1: The preliminary solution, with the bare minimum of sensors/elements present. The camera will act as the robot's primary vision, while distance sensors will act as backup.

In figure 4.2 the available tracked vehicle on which the system will be built is shown. As this is only a prototype, the physical characteristics have not been considered when choosing a platform, and must therefore be modeled for this exact platform, before any control-logic can be implemented. This is done in section 4.1.2 on page 14.



(a) Front corner of the tracked vehicle. **(b)** Underside of the tracked vehicle. **(c)** Rear corner of the tracked vehicle.

Figure 4.2: The tracked vehicle which the preliminary solution will be built upon.

4.1.1 Information Hierarchy

A lot of information about the surroundings can be obtained, using the preliminary solution. To structure the available information and how it will be used, table 4.2 is made.

Camera	Computer Vision/Mapping: identifying edges in the pavement to plan paths, identifying "hidden" areas, mapping areas, mapping obstacles, mapping weeds, and planning return home path.
Encoder	One at each side will ensure information about: the speed of the system, if the system is turning, how far the system has traveled, and if the system is moving according to its input.
Wireless Communication	Whereabouts of the robot, connection to the user interface, Kalman filter localization, and connection to the charge point.
Distance Sensor	Backup for computer-vision. Will mostly act as additional information about surroundings, if the camera should miss anything.
Battery Monitoring	Ensuring enough power is left to return "home" at any point, could also be used to measure real-time power use in various environments.

Table 4.2: Information hierarchy for the system, with the most important information from the top.

4.1.2 Modeling of the Provided Platform

The provided tracked vehicle has the following physical dimensions:

- Length: 365mm
- Height: 140mm
- Width: 330mm
- Clearance: 33mm

- Track width: 90mm
- Body width: 140mm
- Body length: 300mm
- Motors: Krick Max Gear 50:1
- H-Bridge: ZK-BM1
- Top speed: 1.9 km/h

Furthermore, the platforms' correlation between PWM and speed has been tested with results shown in table 4.3 and figure 4.3. Further description of the correlation plots, correlation for each belt, test description and results are available in "Test Report 1 - Forward Motion of Robot" starting page 103.

PWM	Speed (km/h)	Speed (m/s)	Amperes Drawn	Peak Amperes	Voltage
11%	0.5	0.138	0.802	0.869	13.6
17%	0.6	0.167	0.877	0.955	13.6
24%	0.7	0.195	0.957	1.047	13.6
29%	0.8	0.222	0.987	1.118	13.6
37%	0.9	0.250	1.076	1.362	13.6
44%	1.0	0.278	1.143	1.336	13.6
50%	1.1	0.306	1.268	1.33	13.6
53%	1.2	0.333	1.331	1.42	13.6
56%	1.3	0.361	1.386	1.62	13.6
58%	1.4	0.389	1.392	1.72	13.6
60%	1.5	0.417	1.401	1.72	13.6
64%	1.6	0.444	1.426	2.11	13.6
71%	1.7	0.472	1.440	2.12	13.6
99%	1.8	0.500	1.451	2.12	13.6
99%	1.9	0.528	1.488	2.12	14.4

Table 4.3: Only the relevant speeds are shown in this table, as the treadmill used for testing can only increment at 0.1 km/h intervals. It should be noted that the power supply used cannot deliver more than 2.12A in constant current mode.

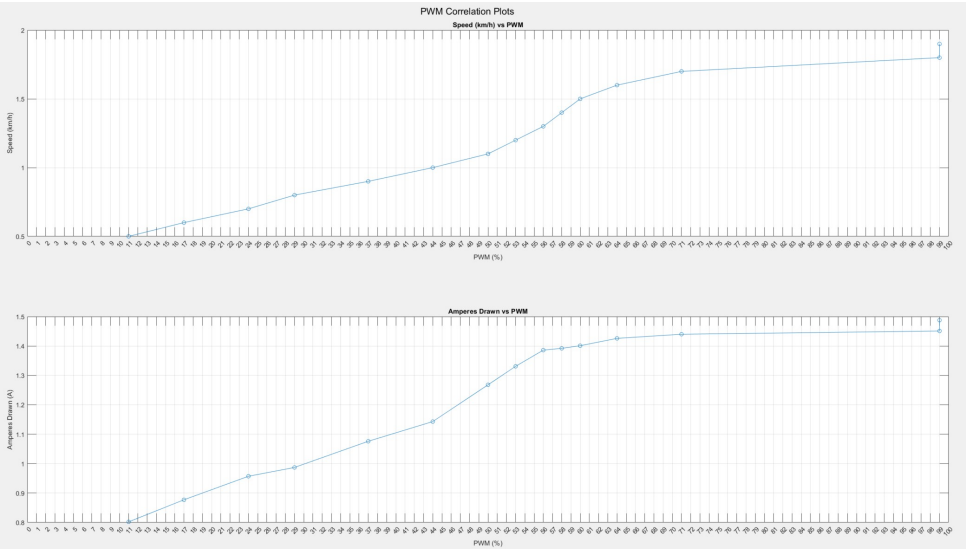


Figure 4.3: Graphs of PWM correlation between speed and amperes drawn, with PWM along the x-axis in both graphs.

As can be seen in figure 4.3 the speed and PWM response are near linear around the

53% mark, while a similarly linear ampere use is found at 53% PWM as well. In figure 4.4 a closer view is shown of the correlation. 53% is chosen as the point of interest as it is in the middle of a broader band with a near-linear correlation of PWM and vehicle speed. One could argue that 56% is more in the middle of the band, but the changes in both speed and amperes drawn, are not as linear here, as around 53%. Furthermore, a base level of 53% is desired to keep the speed low enough, that the vehicle poses no threat to its surroundings, while also allowing more processing time for the computer vision element, to be introduced in 4.4 starting page 38.

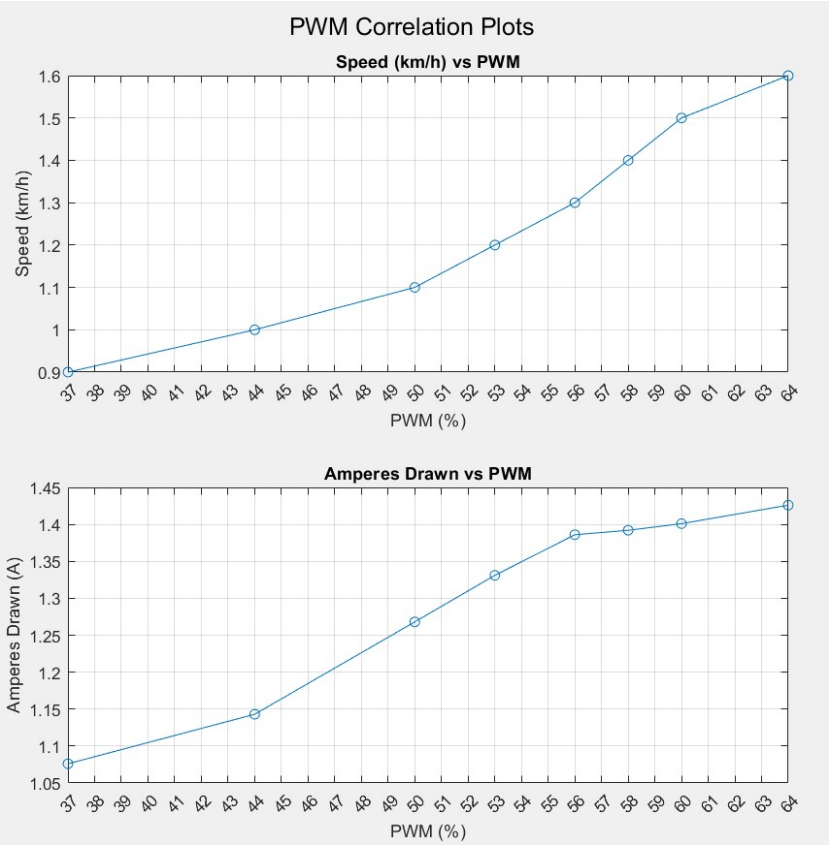
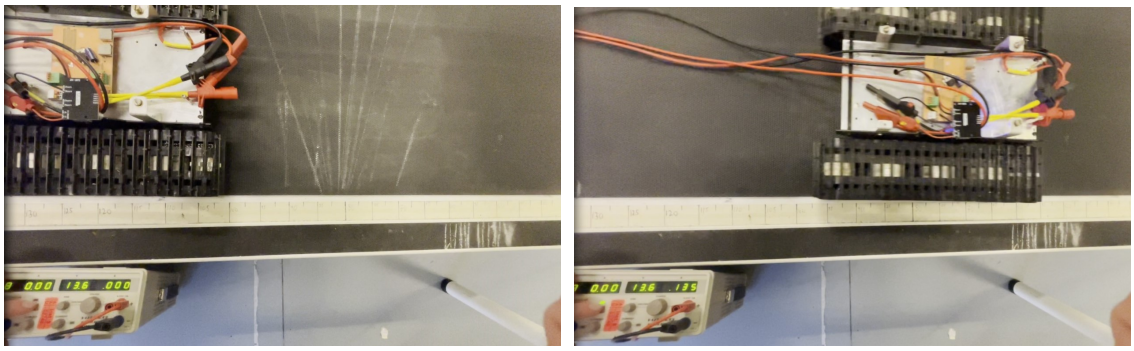


Figure 4.4: Zoomed in version of 4.3 around 44-64% PWM, to showcase linearity present around 53%.

All measurements in table 4.3 and figure 4.3 are measured using a Rodby treadmill incrementing in 0.1 km/h intervals. The PWM signal was adjusted until the robot became "stationary" on the treadmill, and then observed for at least a minute, before noting down what the correlation is. Moreover, all ampere measurements are estimates at nominal speed, as the ampere use would oscillate $+-$ 0.040 ampere. This is imagined to be because of the belts. Peak amperes observed were only seen in the setoff of each run. It should also be noted, that the robot veered to the right at all times, and worsened as PWM increased, suggesting that the motors do not exhibit the same response to PWM. All of the above is described in depth in B.1 starting page 103.

A step response has also been calculated for the system using video-material and MATLAB. The tracked vehicle was recorded when subjected to a PWM of 53, correlating to the nominal speed of 1.2 km/h (0.333 m/s) at 13.6 volts (optimal battery voltage, explained in section 4.6 on page 55). The recording was made at 60 FPS in 1080x1920 resolution, making it possible to read off a measuring tape in the video. Using the video "12kmhResponse.MOV" the observations showcased in figures 4.5 has been made. How-

ever, do note that all position measurements are done by eye estimation, and some error will therefore be present.



(a) Start position 1 frame before amperes are drawn. **(b)** End position after 100 frames (0.442m from start).

Figure 4.5: The start and ending places after 100 frames (1.65 seconds) at 53% PWM.

The resulting transfer function has been derived by observing the velocity plot, showcased in figure 4.6. Furthermore, the transfer function (green) is shown in the plot as well. The transfer function has been made by finding the gain, and then estimating a τ and delay from the velocity plot. The values found are:

$$Gain = \frac{0.33m/s}{53\%PWM} = 0.0062 \tag{4.1}$$

$$\tau = 0.15 \tag{4.2}$$

$$Delay = e^{-0.2*s} \tag{4.3}$$

A better transfer function could be achieved if a higher framerate were available and more precise measurements were made. Nevertheless, the error compared to the real system is negligent, and the following transfer function is considered true, for a step response from 0 to 1.2 km/h. The exact steps to end at this result are described in depth in "Test Report 2 - Starting the Robot From a Standstill", starting page 108. Section B.2.2 on page 109 describes the results, while the complete test report describes the purpose, test setup, procedure, etc.

$$12kmh_{TransferFunction} = e^{-0.2s} * 53 * \frac{0.0062}{0.15s + 1} \tag{4.4}$$

The mean velocity of the system after reaching nominal speed at 60 frames is 0.332 m/s, compared with the 0.333 m/s expected. This deviation is likely due to measurement inaccuracies. It should also be noted, that for the first 29 frames, voltage was only at 4.7 volts, before reaching 12.2 volts till frame 49, where it reached 13.6 as expected. This is due to constant current being activated on the power supply, to keep it from going into current protection. In "Test Report 2 - Starting the Robot From a Standstill", the slowest possible speed (0.5 km/h) is measured as well, and a transfer function for it has been derived. The results can be found in section B.2.2 on page 109.

To complete the description of the preliminary system, it is necessary to know how fast it can turn while moving forward, and at what speed difference it should turn, to not lose noticeable speed. In "Test Report 3 - Turning the Robot" a series of tests has

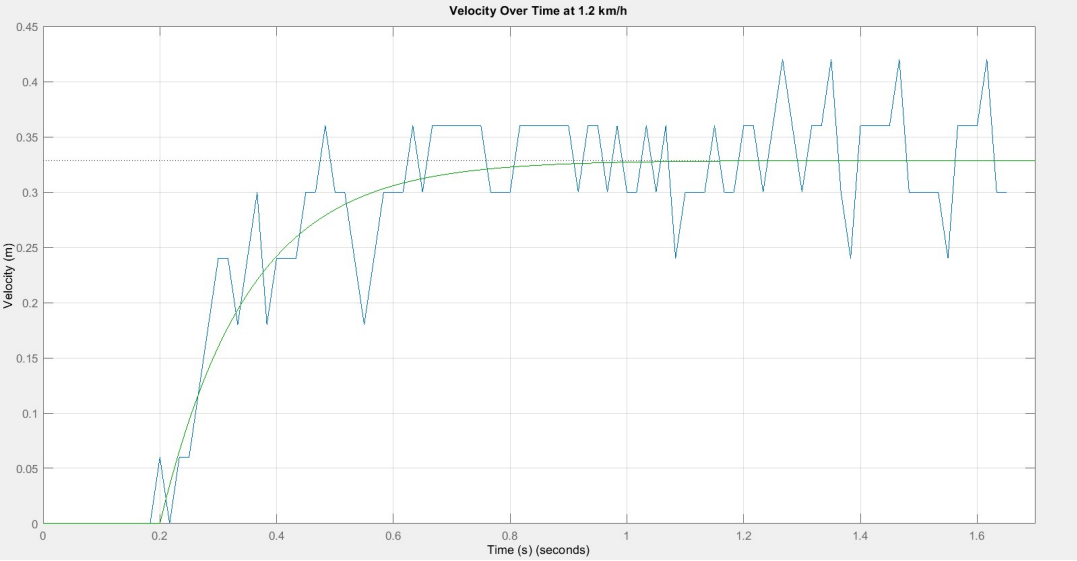


Figure 4.6: Estimated transfer function of the system (green) with velocity plot (blue).

been conducted, examining differences in belt speeds and their impact on the vehicle. All tests have been conducted from a starting speed of 1.2 km/h, and all differences are made in relation to 1.2 km/h. This means that the first test with a difference of 0.1 km/h has been made, by setting the speed of the left belt to 1.2 km/h, and the right belt to 1.1 km/h. For a difference of 0.2 km/h, the speeds are 1.3 km/h and 1.1 km/h, and so it continues up to 1.3 km/h in difference. This is done to lose as little forward momentum as possible when turning. The tests conducted in B.3 have all been evaluated, and the desired turning speeds have been set to 0.7 km/h and 1.3 km/h difference. 1.3 km/h is chosen, as it is the maximum possible turning while still moving forward. 0.7 km/h has been chosen as the operational turning speed, as it exhibits some turning, without under- or overexaggerating it. Moreover, the nominal speed of 1.2 km/h moving forward is almost completely preserved at this difference in speed between the belts.

In "Test Report 3 - Turning the Robot", it is only for 0.7 km/h and 1.3 km/h in difference that complete data analysis has been made. To describe how much the vehicle is turning, the vision function "vision.pointtracker" in MATLABs computer vision toolbox has been used, and its function has been described in detail in section B.3 page 112. In figure 4.7 and 4.8, the angle development over time is shown for both speeds. When observing the graphs, it should be noted, that the first 5 seconds are spent driving straight ahead before the vehicle begins to turn.

To better describe how the angle changes over time, transfer functions has been estimated by observing the angle difference between each frame. The resulting graphs and estimated transfer functions are shown in figure 4.9 and 4.10. The transfer functions derived for both step responses are given in equation 4.12 and 4.8. As with the step response for moving forward, the transfer functions are estimated with qualified guesses and trying different values. In section B.3 at page 115 a more in-depth description is made, of how exactly these transfer functions are derived.

The short version of each parameters estimation is given in table 4.4.

For 1.3 km/h difference:	**For 0.7 km/h difference:**
Delay: counted 300 frames (4.988 seconds) from the button being pressed in the video.	Delay: counted 300 frames (4.988 seconds) from the button being pressed in the video.
Gain: Average angle difference between the last 100 frames.	Gain: Average angle difference between the last 386 frames (larger dataset).
Time constant: Reading off the graph, and trying different values until a suitable match was found.	Time constant: Reading off the graph, and trying different values until a suitable match was found.

Table 4.4: Short overview of how the variables used for the transfer functions describing angle development has been found. Full explanation is available in section B.3 on page 115.

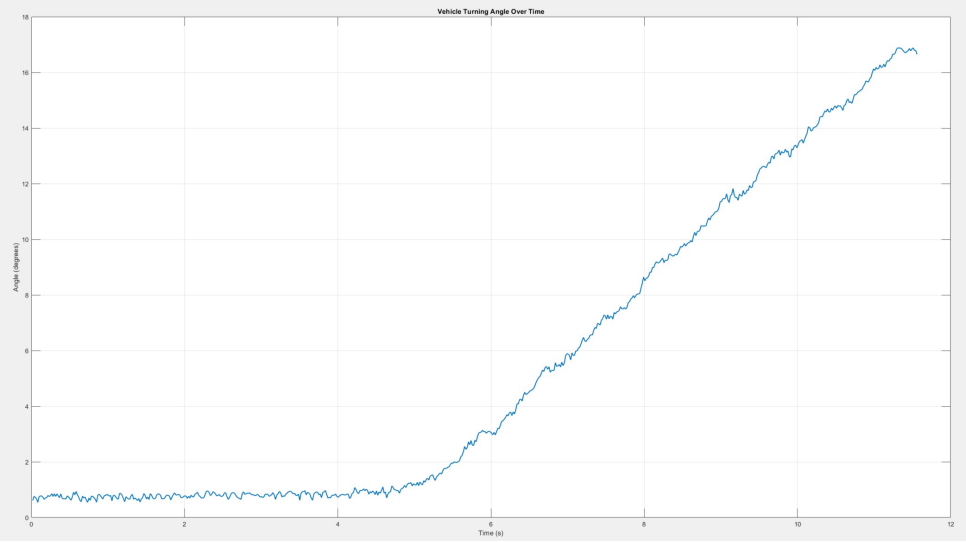


Figure 4.7: Angle development/change over time when the vehicle is subjected to a speed difference of 0.7 km/h. Note that the first 5 seconds are driving straight ahead, as there is a 5-second delay, between pressing the button and the PWM being altered.

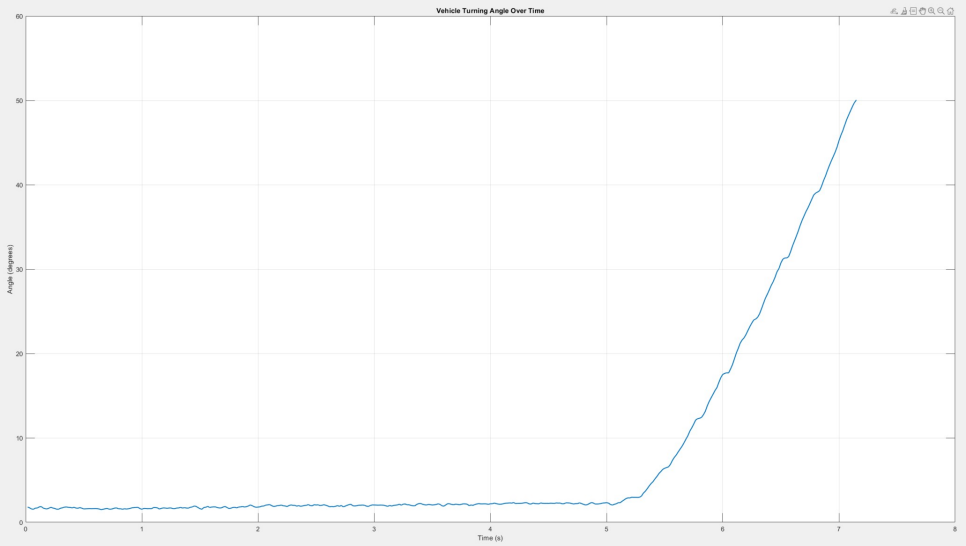


Figure 4.8: Angle development/change over time when the vehicle is subjected to a speed difference of 1.3 km/h. Note that the first 5 seconds are driving straight ahead, as there is a 5-second delay, between pressing the button and the PWM being altered.

With all variables in place, the transfer functions can be plotted in conjunction with the original signal, as shown in figure 4.9 and 4.10. For figure 4.10, the visual interpretation has been significantly harder to manage, due to the amount of noise present. Therefore the signal has been averaged by using the average value over the next 10 frames to describe the current frames' value (shown in red). While this dataset isn't strictly necessary and is an unorthodox manipulation, it makes the visual interpretation much more intuitive. The settling time was found very precisely by notifying the average angle differences in an Excel sheet. A snippet of said Excel sheet is available in section B.3 on page 117 in table B.3.

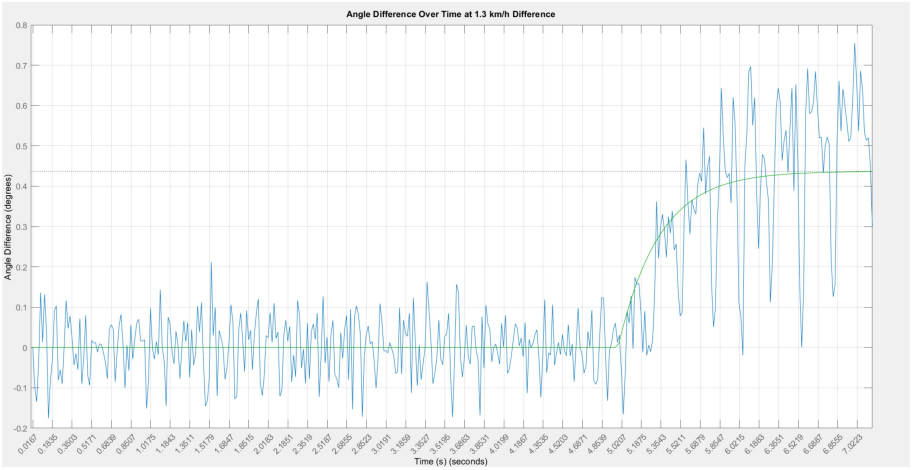


Figure 4.9: Angle difference over time (blue) with estimated transfer function (green) at 1.3 km/h difference.

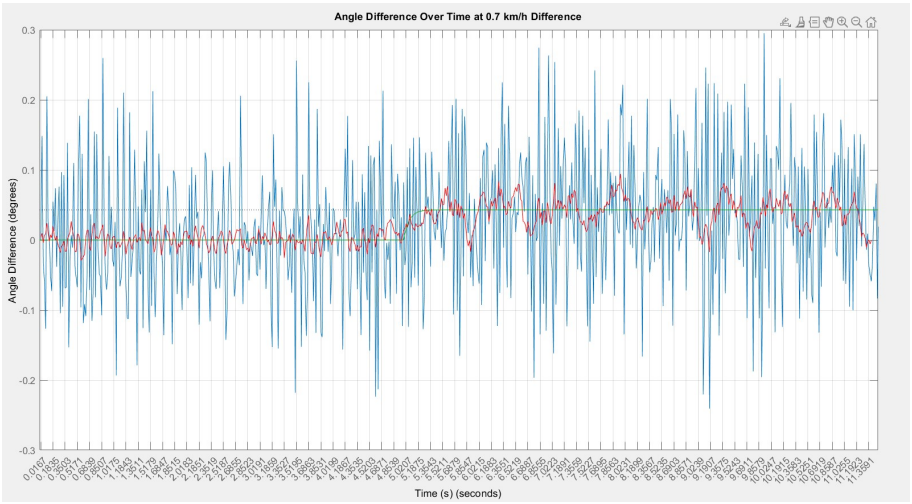


Figure 4.10: Angle difference over time (blue) with estimated transfer function (green) and average angle differences across the next 10 samples (red), with 0.7 km/h difference.

For the step response of 1.3 km/h difference, the estimated variables for a transfer function describing the system are:

$$Gain = 0.437 \quad - \textit{Found by calculating the average value across the last } 100 \textit{ frames of video} \tag{4.5}$$

$$\tau = 0.35[s] \tag{4.6}$$

$$Delay = 0.0166 * 300 = 4.988[s] \tag{4.7}$$

$$G(s) = e^{-4.988*s} * \frac{0.437}{0.35 * s + 1} \tag{4.8}$$

For the step response of 0.7 km/h difference, the estimated variables for a transfer function describing the system are:

$$Gain = 0.040 \quad - Found\ by\ calculating\ the\ average\ value\ across\ the\ last\ 386\ frames\ of\ video \tag{4.9}$$

$$\tau = 0.075[s] \tag{4.10}$$

$$Delay = 0.0166 * 300 = 4.988[s] \tag{4.11}$$

$$G(s) = e^{-4.988*s} * \frac{0.04}{0.075 * s + 1} \tag{4.12}$$

Observing equation 4.8 and 4.12 in their corresponding graphs and by their time constants, it becomes apparent that a speed difference of 0.7 km/h is obtainable much faster than 1.3 km/h, as their respective settling times are 0.3 and 1.4 seconds. As the settling times are so different, they must be accounted for individually in later system design. Depending on the sample time for the final system, 0.3 seconds might even become an issue using a feedback system, as faster sampling could mean the system has not reached a difference in angle, before the next sample. However, looking at the system when everything is settled, the 0.7 km/h difference yields an angle change of 2.48 degrees every 0.5 seconds, while the 1.3 km/h difference yields an angle change of approximately 27.5 degrees every 0.5 seconds.

In "Test Report 4 - Rotating the Robot" the rotational capabilities of the vehicle have been tested. As with the turning capabilities, transfer functions were derived by observing graphs and raw data, acquired with MATLABs computer vision toolbox and the "vision.PointTracker" tool from video material of the vehicle rotating. The transfer functions yielded insight into how fast the vehicle reaches maximum rotational velocity. However, they also revealed that the maximum rotational velocity is not reached before the vehicle has completed more than 1 full rotation. As a final system will not benefit from rotating more than 180 degrees at any point, a linear function has been derived as well, shown in equation 4.13. The linear function shows decent comparability with the graph shown in figure 4.11, as 90-degree and 180-degree rotation is obtained with little error. While the linear function is not far off in the exemplified instances, please do note that it is not exact, especially if the vehicle should ever rotate at full speed. If the vehicle rotates at full speed, it will rotate at approximately 143-153 degrees every second, as shown in figure B.14 on page 119.

$$f(x) = 60_{degrees} * 1_{Second} \tag{4.13}$$

With both the rotational, turning and speed capabilities known, all movement of the system can now be described mathematically. A theoretical example to be tested on a final system could be to traverse the pavement shown in figure 4.12. The path intended

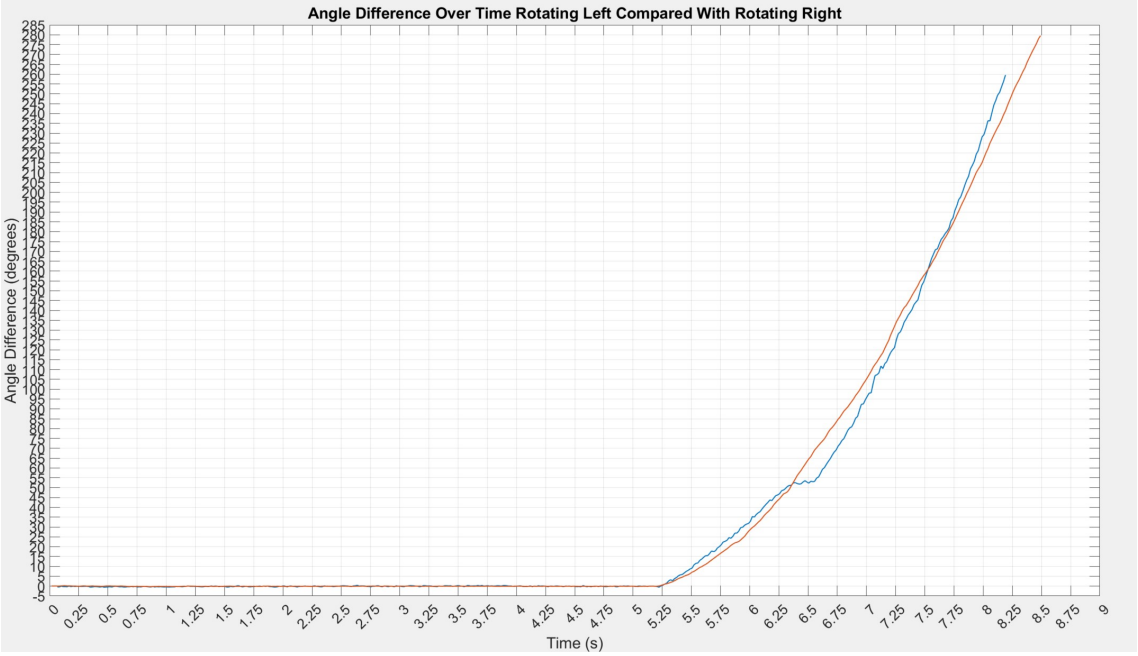


Figure 4.11: Comparison of how the vehicle rotates in both directions. Orange is rotating right, and blue is rotating left.

to use measures 3 meters of straight driving, then a turn measuring 45 degrees, followed by 2 meters of straight driving. Theoretically, the path can be divided into:

Initial straight driving from a standstill:
0.6 seconds is the settling time to reach 1.2 km/h, as derived from equation B.2. In those 0.6 seconds, the first 0.09 meters are traveled. From there, it will travel at 0.333 m/s for 8.74 seconds before it should change the angle by 45 degrees.

Changing the angle:
If it should change the angle with a speed difference of 0.7 km/h, it would take 9.07 seconds and 3.02 meters, which is not feasible at all, as the vehicle would have strayed too far from its path, as it is still moving forward. However, when upping the difference to 1.3 km/h it only takes 0.82 seconds and 0.27 meters. If the choice instead would be to rotate on the spot, it would need 0.75 seconds using equation 4.13 or 1.25 seconds, if the graph shown in 4.11 is read off of instead.

Straight continued driving:
To traverse the last two meters, it would take approximately 5.55 seconds, as the first 15 cm at least are traversed while turning if a speed difference of 1.3 km/h is used. If a speed difference of 0.7 km/h is used, its position is unknown, and therefore the final stretch cannot be assumed. If the vehicle rotates around its Z-axis, it will take 6.34 seconds to arrive at the destination.

Takeaway:
While the speed difference of 1.3 km/h would be too exaggerated in many situations, it seems to be optimal for use in some specific cases such as the one described above, as it is both the fastest and to some extent the most precise way to traverse the pavement, when the conditions are as described in this example.

Figure 4.12: Pavement to traverse theoretically and, in due time, practically.

4.2 General Control of Autonomous Robots

To narrow the subject down to relevant information, this section will focus on operating and designing the operation of autonomous wheel-driven vehicles. At first, a consensus on how autonomous vehicles move will be reached, with corresponding describing terms.

To break the control of autonomous robots even further down, their operation can be broken down into five segments:

1. Mapping: Even the simplest autonomous robots require some form of mapping to perform their task. Coordinates or physical boundaries could represent the mapping.

2. Data Acquisition: The robot has to "observe" the environment, and collect data from sensors.

3. Feature Recognition: Extracting distinct features will in most cases be significant for the operation, ensuring that certain textures, colors, or physical constraints are avoided or approached.

4. Landmark[1] Identification: Related to mapping, the robot should be able to match landmarks to coordinates or other preset criteria (mostly relevant for vision-based systems).

5. Self-localisation: As most autonomous robots move around, they have to know where they are. This can be achieved by measuring the distance traveled, or better yet, measuring distances to known landmarks. Paths can also be derived from knowing the current location.

[1] In this case, a landmark could be anything, such as a doorway, charge point, etc., and not necessarily a landmark such as a statue or a monument.

A good basis for autonomy is made with the previous segments in place. However, navigating the now-known environment requires the robot to solve four subproblems more, before embarking on its task:

1. World Construction: The above segments have to be combined to create a world perception wherein the robot can operate.
2. Path Planning: Based on the perceived/constructed world, the robot needs a task, which it can divide into an ordered sequence of subtasks.
3. Path Generation: With the ordered sequence of subtasks, a path between them must be made, considering the environment.
4. Path Tracking: While it may seem simple to "move 1 meter forward", the robot must at all times reassess its position with regard to known landmarks, as it could otherwise believe it has moved 1 meter, while in reality not moving at all.

Even with the added steps of constructing and following a path made to fit a task, autonomous robots can still meet obstacles hindering their operation. Avoidance of such obstacles will be discussed more in-depth in section 4.4 starting page 38. The above lists are derived from the first chapter of [24].

4.2.1 Electrical Systems in Autonomous Robots

All vehicular autonomous robots share the same basic electrical elements. While most robots contain far more elements than the 6 mentioned below, these 6 broad elements are present in all vehicular robots.

- Battery
- Motor(s)
- World Interpreting Sensors
- Memory (for mapping purposes)
- Wireless Communication
- Computing

4.2.2 Movement

The movement of an autonomous robot requires knowledge of its steering and wheel topology. To avoid discussing subjects unrelated to a final design, the reader should note that a **tracked topology** has been chosen for this project, as mentioned in 4.1 starting page 12. Nevertheless, a short introduction to various platforms will be made.

Typical platform topologies are:

- Two-wheel with a nose wheel
- Four-wheel with steering in one end.
- Multi-wheel with fixed axles.
- Articulated steering.
- Tracked.
- Steering at all wheels.

While these six topologies vary wildly from each other when observed, their steering and general movement are similar for at least three of them. The two-wheel, multi-wheel, and tracked topology all rely on different motor speeds at each side to steer, while the articulated and four-wheeled topology steers by orienting a set of wheels

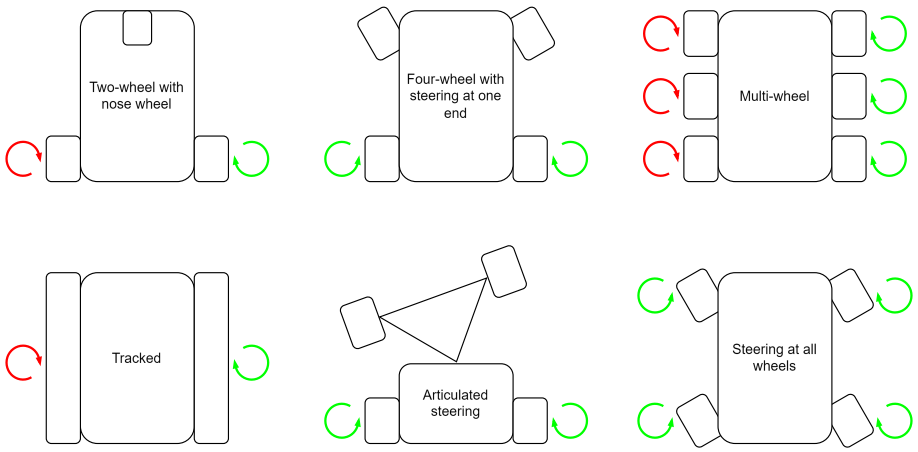


Figure 4.13: The six most common steering topologies used for vehicular robots. Arrows signify driving wheels, with green arrows signifying a forward movement and red signifying a backward movement. All six topologies are steering left in this drawing.

differently to the other set. The only true outlier is robots with the capability of steering at all wheels. In figure 4.13 the six topologies are shown turning to the left, exemplifying that to obtain the same movement, similar but still different approaches are needed.

From here, the focus will lie on a tracked vehicle's movement and proprietary behavior. As can be seen from figure 4.13, the tracked topology relies on either moving each track in opposite directions or at least at different speeds to steer in any direction. A benefit posed by sufficiently strong tracked vehicles is the ability to turn around its Z-axis if the tracks are rotating opposite. Unfortunately, an innate side effect is that when moving forward, the turning circle becomes rather large, dependent on the speed difference between the belts. In figure 4.14 different operations on a joystick are paired with the steering of a tracked vehicle. The purpose of the joystick visualization is to relate common RC operations, to the behavior of a tracked vehicle.

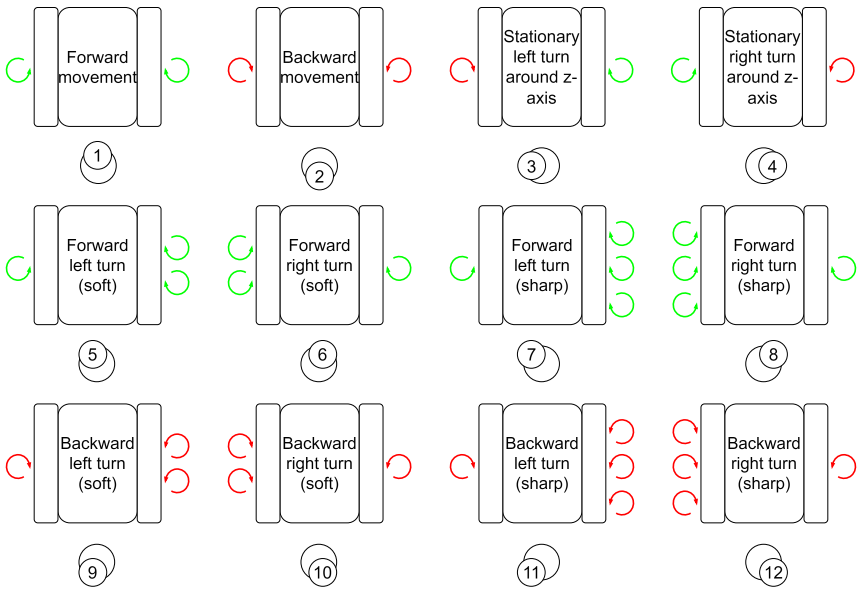


Figure 4.14: Twelve common operations for a tracked vehicle. Beneath each operation, a corresponding joystick placement is found. The number of rotating arrows signifies the speed of each track, and as in figure 4.13, red is backward and green forwards. A real example of what the figures symbolize could be, that joystick position 6 matches a speed difference of 0.7 km/h, while position 8 matches a difference of 1.3 km/h, introduced in 4.1.2 on page 14.

4.2.3 Reacting to the Environment

Now that movement of an autonomous robot has been described, it has to move intelligently around, meaning it has to take its surroundings into account - computer vision and its corresponding benefits, disadvantages, and uses will be described in detail in section 4.4 starting page 38. This section strives to explain common control loops related to the operation of an autonomous robot posed with a more demanding environment, than an empty floor.

In figure 4.15 a general control loop is introduced, to increase understanding of the control loop shown in figure 4.16. Any feedback control loop functions by comparing a reference to an error, feeding the result into a controller which then operates the system. Some kind of sensor will then measure the output of the system, and feed the result (error) back to the controller. Disturbances/noise can play a large or small role, depending on the stability of the system. In figure 4.16 a preliminary feedback system is made for the tracked vehicle. In this system, the controller sends a signal to each belt respectively, by increasing or decreasing the speed difference between them, thereby turning the vehicle. The sensor is based on the computer vision element to be introduced in section 4.4 page 38, where an image of the surface will be taken, and edges in the pavement will be found to create a guiding line for the tracked vehicle. The computer vision element will thereafter calculate an angle difference from the desired, which will be fed back as the error. With the knowledge gained in 4.1 it is evident that all operations of a tracked vehicle should be possible to model, using P controllers.

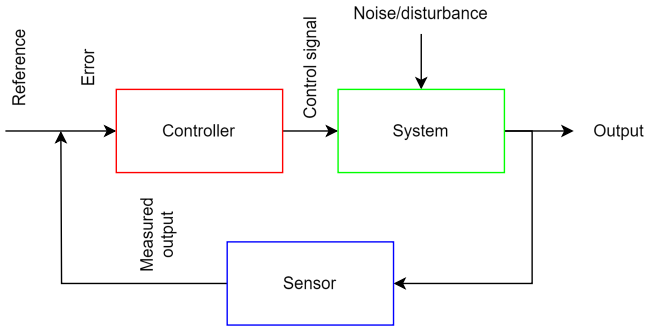


Figure 4.15: General control loop. A reference is introduced, whereafter a controller calculates which signal the system should receive. The system is affected by both the control signal and disturbances, yielding an output, which must be measured to compensate for errors. This is also known as a feedback loop.

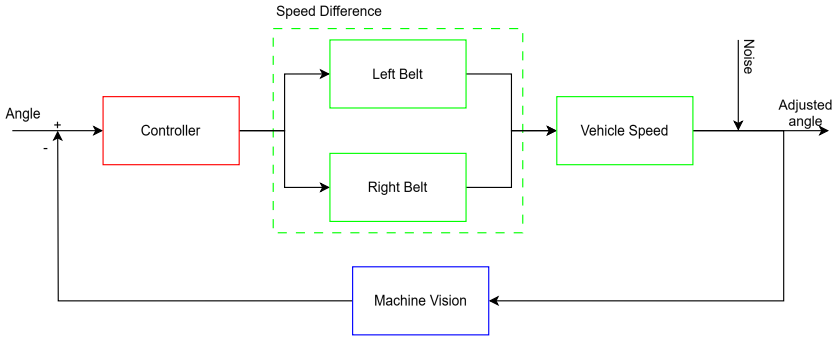


Figure 4.16: Control loop for the tracked vehicle, with the addition of a sensor in the form of computer vision to adjust the angle of the vehicle.

Unfortunately, this tells the controller nothing about the actual speed of the vehicle, as the motor speed is not guaranteed to match the vehicle speed, i.e. in the case of a slippery surface. Therefore another sensor has to be added to feedback the actual speed of the vehicle, unless the computer vision element has to become much smarter. Such a sensor could be an encoder connected to an idle wheel not turned by any of the belts. The idle wheel would have to drag along the ground, centered on the back of the vehicle. To create even more confirmation of how the vehicle behaves, an additional encoder could be added to the axle which then reveals the orientation of the idler wheel, and thereby how much/if the vehicle is turning. In figure 4.17 a sketch of what the system could look like is made. The green circles are encoders, while the red circle is the idle wheel dragging on the surface. The black rod is the axle connecting the idle wheel to the vehicle. By adding two encoders to the idle wheel, it becomes possible to confirm the movement of the vehicle by crossreferencing idle wheel speed and angle with input values.

The other option of increasing reliance on the computer vision element would require that the vision becomes able to track points in the pavement such as the end of tiles or similar fixed objects with known measurements. Furthermore, the vision element would have to compute how fast the fixed points are moving in relation to the vehicle and if they are changing direction (i.e. when turning). However, this is far beyond the scope of this project, and therefore the idle wheel solution will be used.

This introduces a new control loop, based on encoder readings. In reality, the control loop itself is identical to figure 4.16, except for the computer vision being exchanged for idle wheel encoders, as shown in figure 4.18.

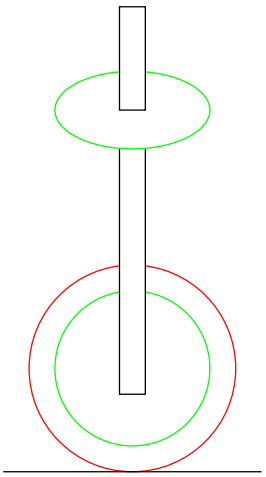


Figure 4.17: Possible idle wheel construction.

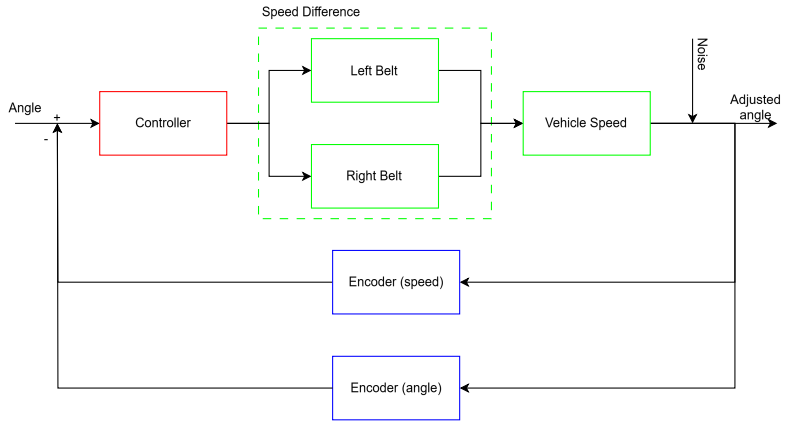


Figure 4.18: Control loop for encoder wheel, containing both angle and speed as feedback.

As can be seen from the control loops in figure 4.16 and 4.18, both loops are reliant on changing the speed of both belts to regulate speed and angle. This could become a problem if the systems are not decoupled. By decoupling the idea is to reduce the interaction between the two control loops present in the system. To do so, a TITO system is introduced in figure 4.19.

To make the TITO system more approachable an introduction to TITO modeling can be found in appendix Introduction to TITO Systems on page 97, where the math behind TITO systems is walked through, using variables, before it is adapted to fit the transfer

functions describing the movement of the tracked vehicle, introduced in 4.1.2. To convert the above to a system describing the tracked vehicle, figure 4.19 is introduced. In this figure, it is evident that one control loop relies on the angle of the tracked vehicle, while the other relies on speed.

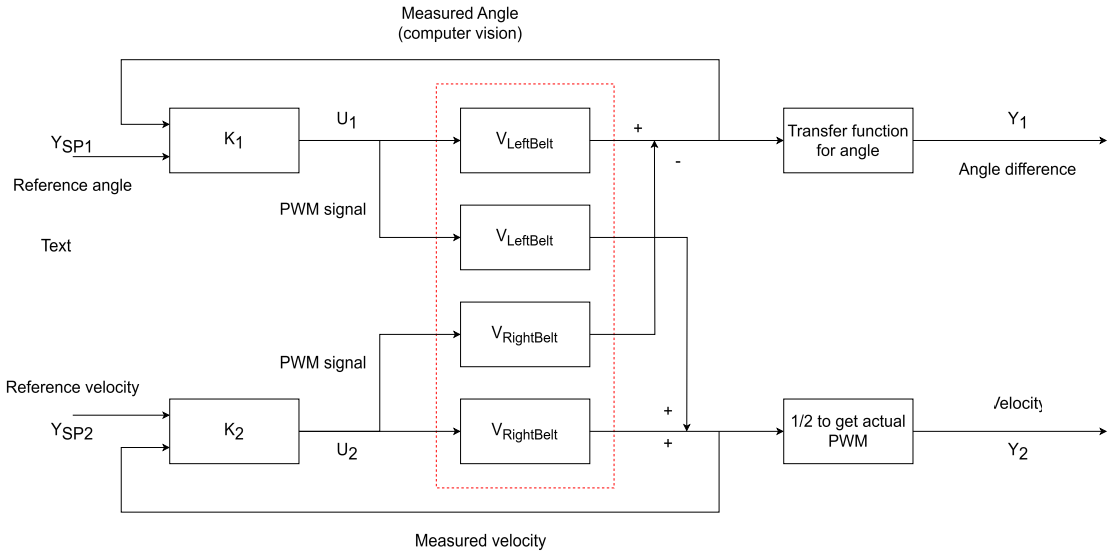


Figure 4.19: TITO system of the tracked vehicle. V_X is the velocity at the specified belt.

By interchanging the transfer functions in the example with the calculated transfer functions for turning and moving, the TITO system can be investigated. At steady state the system can be represented by the same variable, given in equation 4.14, with reference velocity matching 1.2km/h and reference angle being 0. To describe the flow of figure 4.19 starting from the left, an example behavior is utilized. The following will happen physically: the vehicle is driving forward at nominal velocity when a change in the pavement requires it to turn 7.5 degrees to the left. The following happens in the control loop: Measured velocity and measured angle match their references, resulting in no change to U_1 and U_2. Now the measured angle is off by 7.5 degrees, resulting in K_1 lowering the value of U_1, making $V_{LeftBelt} = \frac{0.29}{1}PWM$ to achieve the desired change in angle specified in 4.1.2. Within 3 seconds the measured angle should be back to ≈ 0. During these three seconds, the change in PWM on the left belt has resulted in $V_{RightBelt} = \frac{0.60}{1}PWM$ to compensate for the loss of velocity caused by the angle loop and measured in the feedback loop for velocity. Each belt is therefore represented by equations 4.15 and 4.16.

$$P_{11} = P_{12} = P_{21} = P_{22} = V_{LeftBelt} = V_{RightBelt} = e^{-0.2s} * \frac{0.53}{1}PWM * \frac{0.0062}{0.15s+1} \tag{4.14}$$

$$P_{11} = P_{12} = V_{LeftBelt} = e^{-0.2s} * \frac{0.0062}{0.15s+1} * \frac{0.29}{1}PWM \tag{4.15}$$

$$P_{21} = P_{22} = V_{RightBelt} = e^{-0.2s} * \frac{0.0062}{0.15s+1} * \frac{0.60}{1}PWM \tag{4.16}$$

Now, as the transfer functions describing the vehicle have been derived by testing the vehicle, several of the components used in the TITO system are known already. As U_1 and U_2 are the control signals used, they can be set equal to the PWM relations.

However, they have to be Laplace transformed to follow the proceedings introduced from equations A.1 to A.16. In equations 4.17 and 4.18 the Laplacians of the PWM inputs can be seen.

$$U_1(s) = \mathscr{L}(\frac{0.29}{1}) = \frac{\frac{0.29}{1}}{s} \tag{4.17}$$

$$U_2(s) = \mathscr{L}(\frac{0.29}{1}) = \frac{\frac{0.60}{1}}{s} \tag{4.18}$$

Furthermore, K is known for both expressions as $K_X = 0.0062$. Using the above information, the transfer matrix for the system can be made as shown in equation 4.19:

$$V(s) = \begin{pmatrix} e^{-0.2s} * \frac{0.0062}{0.15s+1} * \frac{\frac{0.29}{1}}{s} & e^{-0.2s} * \frac{0.0062}{0.15s+1} * \frac{\frac{0.60}{1}}{s} \\ e^{-0.2s} * \frac{0.0062}{0.15s+1} * \frac{\frac{0.29}{1}}{s} & e^{-0.2s} * \frac{0.0062}{0.15s+1} * \frac{\frac{0.60}{1}}{s} \end{pmatrix} \tag{4.19}$$

However, to derive the influence between the two systems, the proceedings from equations A.6 to A.12 are used, yielding:

$$Y_{1Angle} = \frac{0.0062}{0.15s+1} * U_1(s) + \frac{0.0062}{0.15s+1} * U_2(s) \tag{4.20}$$

$$U_2(s) = -K_2 * Y_2(s) = Y_2(s) \Rightarrow \frac{-U_2(s)}{K_2} \tag{4.21}$$

Which can then be inserted into equation 4.20, as seen in equation 4.22:

$$\begin{gathered} \frac{-U_2(s)}{K_2} = \frac{0.0062}{0.15s+1} * U_1(s) + \frac{0.0062}{0.15s+1} * U_2(s) \\ \Downarrow \\ \frac{-U_2(s)}{K_2} - \frac{0.0062}{0.15s+1} * U_2(s) = \frac{0.0062}{0.15s+1} * U_1(s) \end{gathered} \tag{4.22}$$

Which is simplified to:

$$\begin{gathered} U_2(s) * (\frac{-1}{K_2} - \frac{0.0062}{0.15s+1}) = \frac{0.0062}{0.15s+1} * U_1(s) \\ \Downarrow \\ U_2(s) * (\frac{-(0.15s+1) - K_2}{K_2(0.15s+1)}) = \frac{0.0062}{0.15s+1} * U_1(s) \\ \Downarrow \\ U_2(s) = \frac{-K_2}{(0.15s+1) + K_2} * U_1(s) \\ \Downarrow \\ U_2(s) = -\frac{K_2}{0.15s+1+K_2} * U_1(s) \end{gathered} \tag{4.23}$$

Which can then be used in 4.24:

$$\begin{gathered} Y_{1Angle}(s) = \frac{0.0062}{0.15s+1} * U_1(s) + \frac{0.0062}{0.15s+1} * (-\frac{K_2}{0.15s+1+K_2} * U_1(s)) \\ \Downarrow \\ Y_{1Angle}(s) = \frac{0.0062}{0.15s+1} * U_1(s) - \frac{0.0062 * K_2}{(0.15s+1) * (0.15s+1+K_2)} * U_1(s) \end{gathered} \tag{4.24}$$

$$\Downarrow$$

$$Y_{1Angle}(s) = \frac{0.0062 * (0.15s + 1 + K_2)}{(0.15s + 1) * (0.15s + 1 + K_2)} - \frac{0.0062 * K_2}{(0.15s + 1) * (0.15s + 1 + K_2)} * U_1(s)$$

$$\Downarrow$$

$$Y_{1Angle}(s) = \frac{0.0062 * (0.15s + 1 + K_2) - 0.0062 * K_2}{(0.15s + 1) * (0.15s + 1 + K_2)} * U_1(s) \tag{4.25}$$

Which becomes an equation, exhibiting the difference that K_2 has on the system if s=0, shown in 4.26:

$$Y_{1Angle}(0) = \frac{0.0062}{1 + K_2} \tag{4.26}$$

By equation 4.26 it becomes apparent that K2 will influence Y1. K2's influence will decrease as its own value increases, as equation 4.26 will then yield smaller values. This implies that the system is coupled, as seen in figure A.1. Similar equations could be derived to find K1's influence on Y2. The question now is, which loop should determine the behavior of the other loop?

A scientific way would be to use a relative gain array (RGA). The RGA is a matrix, that can reveal the effect of pairing input and output variables in TITO systems. It is derived from the steady-state gain matrix of the system denoted "P". Mathematically, the RGA is defined as:

$$RGA = V(0) \circ (V(0)^{-1})^T \tag{4.27}$$

Where:
$V(0) = \begin{pmatrix} V_{LeftBelt}(0) & V_{LeftBelt}(0) \\ V_{RightBelt}(0) & V_{RightBelt}(0) \end{pmatrix}$ is the static gain of the system. $\circ$ is piecewise multiplication. $(V(0)^{-1})^T$ is the inverse transposed of $V(0)$. Using it on the system yields the matrix shown in 4.28:

$$V(0) = \begin{pmatrix} e^0 * \frac{0.0062}{1} & e^0 * \frac{0.0062}{1} \\ e^0 * \frac{0.0062}{1} & e^0 * \frac{0.0062}{1} \end{pmatrix} \Rightarrow \begin{pmatrix} \frac{0.0062}{1} & \frac{0.0062}{1} \\ \frac{0.0062}{1} & \frac{0.0062}{1} \end{pmatrix} \tag{4.28}$$

The next step is finding the inverse, which becomes a harder task than showcased in the example, shown in the following:

$$V(0)^{-1} = \frac{1}{\frac{0.0062}{1} * \frac{0.0062}{1} - \frac{0.0062}{1} * \frac{0.0062}{1}} * \begin{pmatrix} \frac{0.0062}{1} & \frac{0.0062}{1} \\ \frac{0.0062}{1} & \frac{0.0062}{1} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{4.29}$$

As the matrix is singular, another approach has to be taken. The Uhlmann Inverse will be used to describe the singular system, which is a method that can handle singular matrices by generalizing the concept of inversion, [25, 26, 27, 28]. The Uhlmann inverse in equation 4.33 is calculated using singular value decomposition (SVD) shown in equation 4.30, which states that:

$$P = U * \Sigma * V^T \tag{4.30}$$

Where: U and V^T are orthogonal matrices and Σ is a diagonal matrix containing the singular values of P. To find the SVD of V(0), MATLAB is used, yielding:

$$U = \begin{pmatrix} -0.7071 & -0.7071 \\ -0.7071 & 0.7071 \end{pmatrix}, \Sigma = \begin{pmatrix} 0.0124 & 0 \\ 0 & 0 \end{pmatrix}, V^T \begin{pmatrix} -0.7071 & -0.7071 \\ -0.7071 & 0.7071 \end{pmatrix} \tag{4.31}$$

Then Σ is modified to obtain the generalized inverse, denoted Σ^+:

$$\Sigma^+ = \begin{pmatrix} \frac{1}{0.0124} & 0 \\ 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 80.6452 & 0 \\ 0 & 0 \end{pmatrix} \tag{4.32}$$

$$V^+ = U * \Sigma^+ * V^T \Rightarrow V^+ = \begin{pmatrix} -0.7071 & -0.7071 \\ -0.7071 & 0.7071 \end{pmatrix} \begin{pmatrix} 80.6452 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -0.7071 & -0.7071 \\ -0.7071 & 0.7071 \end{pmatrix}$$
$$V^+ = \begin{pmatrix} 40.325 & 40.325 \\ 40.325 & 40.325 \end{pmatrix} \tag{4.33}$$

Now, as the Uhlmann inverse of the matrix is still symmetric, the transpose is, therefore, the same, making V^+ equal to $(V(0)^{-1})^T$. The next step is piecewise multiplication, shown in equation 4.34

$$RGA = V(0) \circ V^+ = \begin{pmatrix} \frac{0.0062}{1} & \frac{0.0062}{1} \\ \frac{0.0062}{1} & \frac{0.0062}{1} \end{pmatrix} \circ \begin{pmatrix} 40.325 & 40.325 \\ 40.325 & 40.325 \end{pmatrix} = \begin{pmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{pmatrix} \tag{4.34}$$

The RGA can now be checked with the interaction index, introduced in A.17 and used in 4.35. With a lambda value of 0.25, the row and column checksum method still yields the correct interpretation.

$$\begin{pmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{pmatrix} == \begin{pmatrix} \lambda & 1-\lambda \\ 1-\lambda & \lambda \end{pmatrix} \tag{4.35}$$

As the gain is outside the interval of $0.67 < \lambda < 1.5$, decoupling should improve control of the system. To begin decoupling the system, a new loop "identical" to the original one is introduced in figure 4.20. To describe the decoupling mathematically, equations 4.37 and 4.38 is introduced. Using equation 4.39 and 4.40 also makes it possible to reduce the number of unknowns in the decoupling equations, as shown in 4.41 and 4.42.

$$Q = \frac{V_{RightBelt} * V_{LeftBelt}}{V_{LeftBelt} * V_{RightBelt}} \tag{4.36}$$

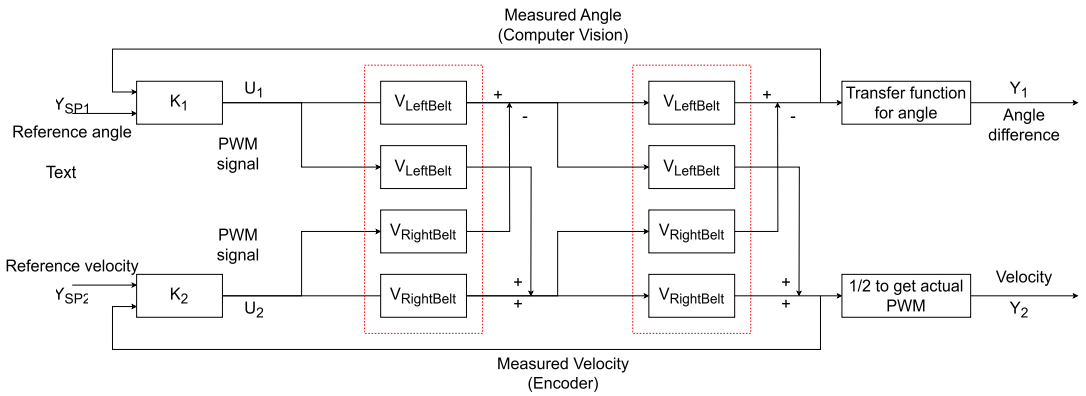


Figure 4.20: The decoupled TITO system for the tracked vehicle.

$$\frac{Y_1}{U_{1A}} = F_{12} * V_{RightBelt} + F_{11} * V_{LeftBelt} = 0 \tag{4.37}$$

$$\frac{Y_2}{U_{1A}} = F_{21} * V_{LeftBelt} + F_{22} * V_{RightBelt} = 0 \tag{4.38}$$

$$F_{12} = -F_{11} * \frac{V_{LeftBelt}}{V_{RightBelt}} \tag{4.39}$$

$$F_{21} = -F_{22} * \frac{V_{RightBelt}}{V_{LeftBelt}} \tag{4.40}$$

$$\frac{Y_1}{U_{1A}} = -F_{11} * \frac{V_{LeftBelt}}{V_{RightBelt}} * V_{RightBelt} + F_{11} * V_{LeftBelt} = 0 \tag{4.41}$$

$$\frac{Y_2}{U_{2A}} = -F_{22} * \frac{V_{RightBelt}}{V_{LeftBelt}} * V_{LeftBelt} + F_{22} * V_{RightBelt} = 0 \tag{4.42}$$

Equations 4.43 and 4.44 are used to continue the simplification. This simplification has left the equation system with 4 unknowns (F_{11}, F_{12}, F_{21}, *and* F_{22}) describable by 2 equations. Therefore, two of the unknowns can be chosen, and to simplify matters, they should be 1, as this makes it easier to find the remaining two, using equations 4.39 and 4.40 with F_{11} and F_{22} set to 1, yielding equations 4.45 and 4.46.

$$\begin{aligned}
\frac{Y_1}{U_{1A}} &= - F_{11} * \frac{V_{LeftBelt}}{V_{RightBelt}} * V_{RightBelt} \\
&\Downarrow \\
\frac{Y_1}{U_{1A}} &= (1 - \frac{V_{RightBelt} * V_{LeftBelt}}{V_{LeftBelt} * V_{RightBelt}}) * V_{LeftBelt} * F_{11} \\
&\Downarrow \\
\frac{Y_1}{U_{1A}} &= (1 - Q) * V_{LeftBelt} * F_{11}
\end{aligned} \tag{4.43}$$

$$\begin{aligned}
\frac{Y_2}{U_{2A}} &= - F_{22} * \frac{V_{RightBelt}}{V_{LeftBelt}} * V_{LeftBelt} \\
&\Downarrow \\
\frac{Y_2}{U_{2A}} &= (1 - \frac{V_{RightBelt} * V_{LeftBelt}}{V_{LeftBelt} * V_{RightBelt}}) * V_{RightBelt} * F_{22} \\
&\Downarrow \\
\frac{Y_2}{U_{2A}} &= (1 - Q) * V_{RightBelt} * F_{22}
\end{aligned} \tag{4.44}$$

$$F_{12} = -1 * \frac{V_{LeftBelt}}{V_{RightBelt}} \tag{4.45}$$

$$F_{21} = -1 * \frac{V_{RightBelt}}{V_{LeftBelt}} \tag{4.46}$$

This leaves the final two expressions to be given in equation 4.47 and 4.48, enabling modeling using SISO methods rules from here on out, as shown in figure 4.21.

$$Y_1 = U_{1A} * (1 - Q) * V_{LeftBelt} \tag{4.47}$$

$$Y_2 = U_{2A} * (1 - Q) * V_{RightBelt} \tag{4.48}$$

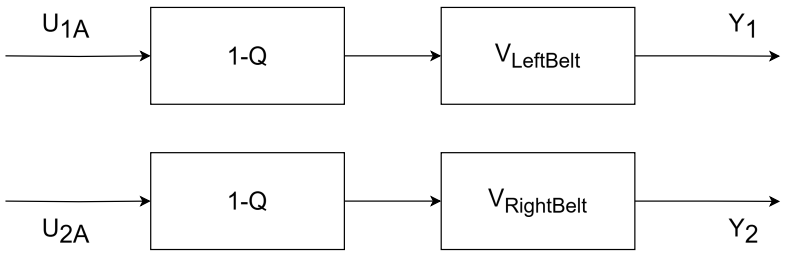


Figure 4.21: Final interpretation of how a the TITO system becomes a SISO system for the tracked vehicle.

4.3 Sensors

A host of sensors are needed to enable any autonomous system to sense the world around it. This section intends to describe the sensors required to operate the limited version of the de-weeding autonomous robot. An ESP32CAM has been chosen to navigate the environment, while distance sensors serve as backup insurance against accidental hits with surrounding objects. Furthermore, a speed sensor will be implemented as an ensuring factor to correlate motor input with actual speed.

4.3.1 ESP32CAM

The ESP32CAM is a versatile development board in the ESP family, equipped with the Espressif ESP32 processor, which supports Wi-Fi, Bluetooth, and multiple GPIO interfaces. While it retains much of the functionality of a standard ESP32, the ESP32CAM is specifically designed for low-cost image and video streaming applications. It features an OV2640 camera module with a 2 MP resolution, making it suitable for lightweight machine vision tasks in robotic applications.

The board includes several useful components:

- SD card slot: Enables local storage of images, data logging, or pre-trained machine learning models.
- Camera socket: Supports multiple camera modules, including the default OV2640, allowing for easy hardware upgrades depending on the vision requirements of the robot.
- Flash: Useful for illumination in low-light conditions, enhancing the robot's vision performance.

However, due to the integration of the SD card and camera socket, the number of available GPIO pins is reduced, limiting the number of additional peripherals that can be connected. For applications requiring more sensor inputs or actuators, an external multiplexer or a dedicated I/O expansion board may be necessary.

Key Benefits for Autonomous Robotics:

1. Cost-Effectiveness:
   The ESP32CAM offers an affordable solution for incorporating vision capabilities into the robot, making it ideal for low-cost robotic platforms such as the autonomous laser-weeding robot.

2. Wireless Communication:
   With built-in Wi-Fi and Bluetooth, the ESP32CAM enables seamless integration with external devices, remote monitoring, and control. This wireless capability can be leveraged for remote data transmission, image processing offloading, and system updates.

3. Edge Computing for Machine Vision:
   The ESP32 processor is powerful enough to handle basic image processing tasks directly on-board, such as:

   - Object detection using libraries like OpenMV, OpenCV, or TensorFlow Lite for microcontrollers.
   - Line tracking, color recognition, and motion detection for navigating the robot or identifying pavement patterns in realtime.

   The board's ability to offload some vision tasks to the edge reduces the reliance on continuous wireless communication, enabling autonomous operation even in low-connectivity environments such as large properties with extensive paving.

4. Low Power Consumption:
   The ESP32CAM's low power consumption makes it well-suited for battery-operated systems, extending the operational time of the robot while ensuring efficient power use.

Vision-Based Autonomy:

The ESP32CAM plays a crucial role in enabling vision-based autonomy for the robot. Its machine vision capabilities allow the robot to:

- Identify optimal paths in real-time using the camera feed.
- Utilize image classification algorithms to differentiate between paving techniques.
- Provide feedback on the robot's surroundings, improving navigation and obstacle avoidance when paired with other sensors (discussed in section 4.4).

The large open-source codebase and Arduino IDE support offer a wealth of pre-existing machine vision examples, simplifying the development process. Additionally, existing libraries for image capture, machine learning inference, and streaming can be adapted to suit specific tasks required by the robot. Further use of the ESP32CAM in machine vision will be detailed in section 4.4.

4.3.2 Distance Sensor

As the ESP32CAM is not omnidirectional, secondary distance sensors will be incorporated as well. The distance can be measured by a wide range of active sensors as

mentioned in the introduction to chapter 4 on page 12, but as mentioned there as well, all of the most commonly used sensors are active sensors reliant on somewhat optimal conditions. Nevertheless, ultrasound or a laser-based ToF sensor will be utilized as the secondary sensor. These options are chosen over LIDAR, RADAR, and IR as they're cheaper types of sensors, while also fulfilling the need for object detection (albeit, only in one direction). Furthermore, ultrasound and ToF sensors are more precise at close range and use similar techniques, making interchangeability easy.

In short terms, ultrasound and ToF sensors work by sending a pulse out and waiting for the pulse to be reflected, deriving distance as a function of time elapsed between sending the pulse and receiving it again. An illustration can be seen in figure 4.22. A general formula is derived in equation 4.49:

$$Distance = \frac{T_{Elapsed} * Propagation_{Speed}}{2} \tag{4.49}$$

Where:
$T_{Elapsed}$ = Time of flight for each pulse.
$Propagation_{Speed}$ = The speed of each pulse (Speed of sound for ultrasound and speed of light for ToF).
Dividing by two yields the actual distance, as the pulse has to travel back and forth, making the total travel twice the distance.

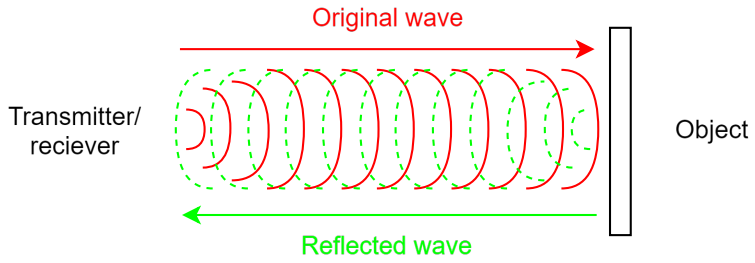


Figure 4.22: Illustration of how ultrasound and ToF sensors work.

A maximum working distance and signal spread will have to be calculated to create a more predictable system and ensure interference shouldn't be a problem. The sensor in this instance will only rely on proximity objects at a maximum distance of 1.5 meters. Therefore equation 4.50 can be set to find the maximum Pulse Repetition Frequency (PRF) at any distance (d):

$$PRF = \frac{0.5 * Propagation_{Speed}}{d} \tag{4.50}$$

As a failsafe, the PRF will be dependent on the maximum distance desired to read data, i.e. 1.5 meters. In figure 4.23 it can be seen how the signal should function, and be seen how a false positive could create interference. However, this can be mitigated by always waiting for a return signal. Unfortunately, this could create a dysfunctional system, as there is no guarantee that a return is ever received. Therefore, return signal strength should be another deciding factor, checking if the read signal is correct or a false positive.

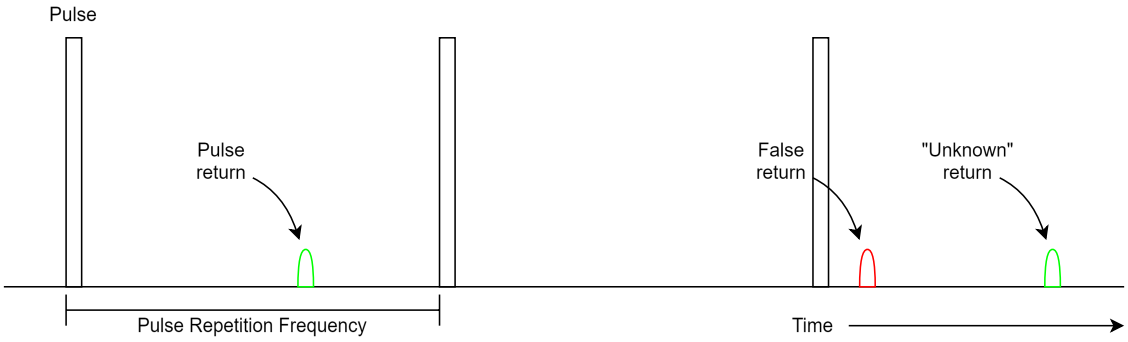


Figure 4.23: PRF correlation and how a too slow pulse return could result in a false positive.

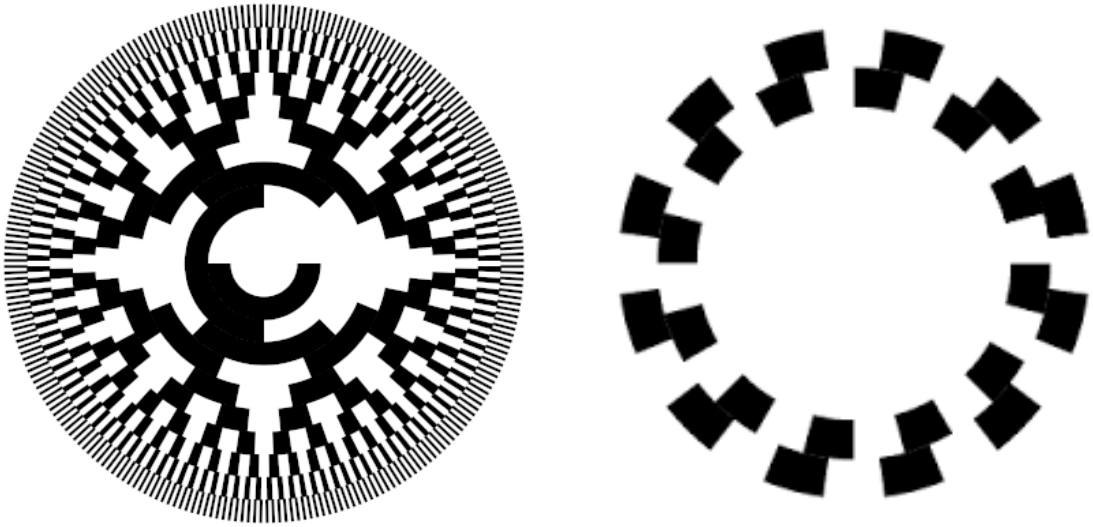
4.3.3 Speed Sensor

As discussed in section 4.2, it is detrimental to the operation of the robot that feedback is acquired regarding the actual speed of the robot. Without knowing the actual speed control will become largely impossible. To measure the speed several methods can be applied[2]:

- Encoders.
- Hall Effect Sensors.
- IMU.
- Tachometer.

Encoders

Encoders are commonly used to measure the rotation of a shaft by counting the reflecting lights off of a patterned disc. The patterned disc can have a distinct pattern all around, yielding knowledge of the exact position in which the shaft is, or simpler patterns yielding only knowledge of which direction and speed the shaft is turning, see figure 4.24 both encoder versions are illustrated.



(a) Absolute encoder disc. **(b)** Incremental encoder disc.

Figure 4.24: Encoder topologies.

[2]A fifth option exists using GPS and/or triangulating between local antennas, however, this option is ruled out for the prototype to decrease complexity.

The encoders can be placed anywhere in the drivetrain or on an idler wheel. Placing the encoder disc on the drivetrain can however result in false data, as the encoder only informs of the speed at one point in the drivetrain, and does not necessarily measure the actual speed of the robot. Placing the encoder on an idle wheel not connected to the track could instead yield a correct reading, as the idle wheel only turns when the vehicle is moving.

Hall Effect Sensors

Hall effect sensors work similarly to the encoder. Instead of counting reflections from an encoder disc, they detect the presence of magnetic fields when passing the sensor. Magnets are then placed on the track or a driveshaft, and then the sensor counts each magnet passing by. Similarly to the encoders, offset magnets can determine which direction the track is rotating. Measuring the time between each counted magnet allows for calculating the drive speed.

Inertial Measurement Unit

An IMU could also be utilized for speed monitoring. An accelerometer would at all times yield readings of what forces the robot is subjected to, and paired with a gyroscope it could yield precise measurements when integrated over time. Unfortunately, it is fairly susceptible to data drift, and it would be complicated to ensure correct readings when driving at an incline, as accelerometer values could be interpreted as fast movement, rather than the vehicle being on an incline.

Tachometer

A tachometer is a simplified version of the encoder, only having 1 point it measures on a shaft or along the track. It furthermore requires known gear ratios and track dimensions, to calculate the vehicle's speed. Yet again, this solution would rely too heavily on the track not being able to slip.

Overview

As several options exist, with each their own benefits, no optimal solution is shining through. Therefore, the following summarizing table has been made, to discern between the pros and cons:

Sensor Type	Pros	Cons
Encoders	High precision, can track speed and direction	Sensitive to dirt and slippage dependent on installation, mounting complexity.
Hall Effect	Simple, reliable	Requires magnet mounting, extra calculation for speed.
IMU	No physical contact, tracks acceleration and rotation	Prone to drift over time, prone to false readings at an incline.
Tachometer	Simple implementation, good for motor feedback systems	Requires calibration, affected by slippage.

Table 4.5: Comparison of speed sensors for tracked vehicle.

4.4 Computer Vision

To create a truly autonomous robot, machine vision/computer vision is detrimental to its operation. However, the prototype will mostly focus on utilizing computer vision. The distinction between machine and computer vision is found in their respective use cases. While both techniques broadly work by analyzing images, machine vision is mostly more goal-oriented, working in controlled environments focused on one task, typically quality control or similar tasks. All the while, computer vision is more of a dynamic version, working in uncontrolled and unpredictable environments, such as autonomous vehicles or augmented reality. Furthermore, computer vision is more oriented at recognizing images, objects, etc., while completing more complex tasks in a dynamic environment. Ergo, it is perfect for an autonomous robot roaming the great outdoors.

A Quick Note on Image Processing

Image processing is in this instance a specific form of signal processing, where the signal is images and the output is an interpretation of each image. The actual interpretation can be of varying character, depending on the desired result. Common image processing could be conversion to grayscale, color inversion, gaussian blurring, and in broad terms any sort of manipulation om images.

Regarding autonomous robots, image processing relies heavily on canny edge detection, object recognition, pattern recognition, and large training models. Canny edge detection sticks out amongst the other methods, as it does not rely on training models, but rather relies on image quality and an algorithm, which is further explained in section 4.4.1. Object detection and pattern recognition rely on large training models to learn how to interpret what is pictured in an image accurately.

4.4.1 Canny Edge

Canny edge detection was developed by John F. Canny in 1986 and is an image-processing operation designed to identify edges and other "structural" information that can be extracted. John has developed the general algorithm based on three criteria:

1. Detection of edges should be done with a low error rate, resulting in the maximum amount of edges being found.
2. Edges found must be accurately localized on the center of each edge.
3. Any edge should only be marked once.

To obtain optimal edge identification, the sum of four exponential terms should be used. However, the first derivative of a Gaussian is generally used as an approximation to decrease computing time. Canny edge detection can be done on any image of any size, but computing time will rise dramatically with increased resolution. The reason is found in the number of computations made for each pixel, determined by the size of the kernel used in each processing step, i.e. for a Gaussian blur, the execution time is given in big O notation[3] as:

$$GaussianExecutionTime = O(Width_{Kernel} * Height_{Kernel} * Width_{Image} * Height_{Image}) \tag{4.51}$$

[3]The worst case scenario for an algorithm to compute. In this case, the computation can be faster if enough pixels are of low value.

This means that a Gaussian blur imposed on a 1920x1080 image with a 7x7 kernel would take 101.606.400 computations, whereas a 3x3 kernel on the same image would take 18.662.400 computations. Reducing the image size from HD to the 160x120 pixels used in [24], the same kernel sizes yield 940.800 and 172.800 computations. More information regarding Gaussian blur and its workings can be found in 4.4.1.
The general algorithm for canny edge detection is divided into the following steps, [29, 30, 31]:

1. Apply Gaussian filters to remove any noise.
2. Localize intensity gradients within the image using the Sobel operator.
3. Decide and apply a gradient threshold to locate possible edges.
4. Apply double thresholding to determine actual edges.
5. Track and connect edges by hysteresis.

In computer vision, a sixth step is added previous to the above five: converting the image to grayscale. By converting the image to grayscale, all following computations are made easier, as pixels in the image now only contains high or low values from black to white, instead of varying values in RGB. While this is step is not strictly necessary, it eases every following step. A more detailed guide for each step is found in the following sections.

Grayscale Conversion

To convert an image from RGB to grayscale the first step is to find the RGB values of each pixel, as they will be used in all methods of conversion. With the corresponding RGB values found for each pixel, they have to be transformed. A common method is as simple as taking the average of each value:

$$Grayscale = \frac{R + G + B}{3} \tag{4.52}$$

Which yields a decent grayscale. However, it can be made better by using the weighted average instead. The weighted average takes human perception into account as well, as the human eye does not perceive all colors equally well. The weighted grayscale conversion is:

$$Grayscale_{Weighted} = \frac{R * 0.299 + G * 0.587 + B * 0.144}{3} \tag{4.53}$$

By using this grayscale conversion, a better perception is achieved as seen with a human eye. An unfortunate side-effect of doing a grayscale conversion is the loss of all knowledge regarding color in the image. Therefore, if this information is of any use (such as in the final version of this product, to identify greenery), the system either has to save a grayscale and original version of each image, requiring vast amounts of memory. Another applicational use which could result in lower memory needs, would be identifying greenery first, before beginning canny edge processing.

Gaussian Blur

The next step in the canny edge algorithm is applying a Gaussian blur filter to the image. The purpose of a Gaussian blur filter is to smooth the picture, removing noise and blemishes. A Gaussian blur smooths each pixel in an image, based on a normal distribution from the center of each pixel. To use filter terminology, a Gaussian blur acts as a lowpass filter, and its Fourier transform is another Gaussian. The effect a Gaussian blur has on an image is reducing the amount of high-frequency (high value) components/pixels, thus normalizing all pixels at a lower value. Before introducing more math, a visual representation is made in figure 4.25 of how a Gaussian filter processes an image.

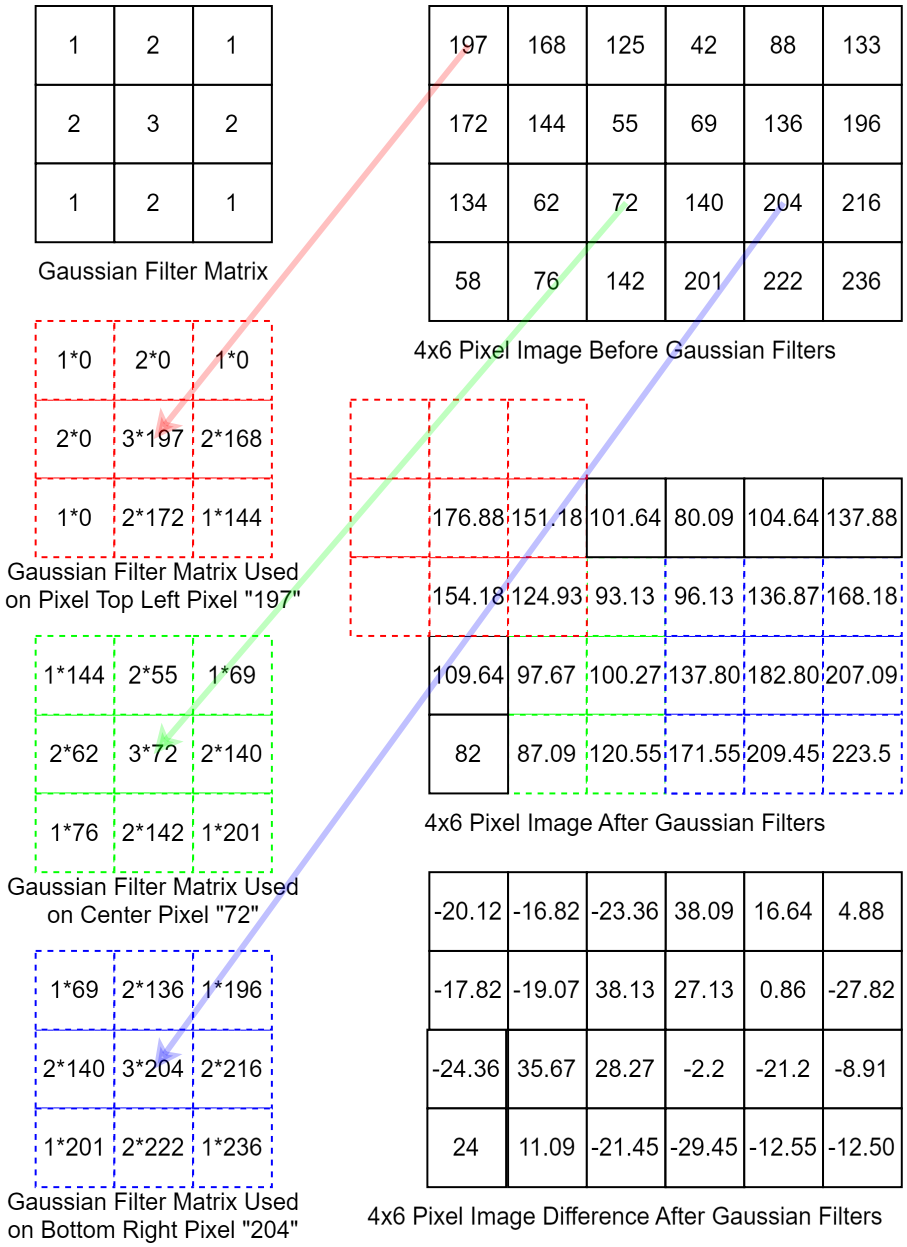


Figure 4.25: Caption

In figure 4.25, a Gaussian filter matrix is introduced, also known as the kernel of the Gaussian. In this case, the kernel is 3x3, but could theoretically be any odd integer

larger than 1, creating an even smoother effect at the cost of computing time. The kernel is then applied to each pixel in the image subjected to the Gaussian. The value yielded by the kernel then replaces the pixel value, after the kernel has computed a value for each pixel. In the three cases shown in figure 4.25, the calculations are:

$$\textbf{RED/197} = \frac{1*0+2*0+1*0+2*0+3*197+2*168+1*0+2*172+1*144}{0+0+0+0+3+2+2+1} \tag{4.54}$$

$$\textbf{GREEN/72} = \frac{1*144+2*55+1*69+2*62+3*72+2*140+1*76+2*142+1*201}{1+2+1+2+3+2+1+2+1} \tag{4.55}$$

$$\textbf{BLUE/204} = \frac{1*69+2*136+1*196+2*140+3*204+2*216+1*201+2*222+1*236}{1+2+1+2+3+2+1+2+1} \tag{4.56}$$

The values are found by multiplying the Gaussian kernel value with the value found at each pixel. The collected sum is then divided by each kernel value that has a pixel value in its corresponding spot. That is why equation 4.54 has multiple zeroes in it. One could argue that simply changing any pixels not present to 0 could skew the Gaussian blur for edge pixels, however, with a 3x3 or 5x5 kernel, it would only skew the outer-most 1-2 pixels, which in most cases would probably not make too much of a difference. Observing the middle pixel image in figure 4.25 it can be seen that each pixel has been changed by the Gaussian operator, creating smoother transitions between the pixels, which is desired.

Måske indsæt billeder af et normalt billede før efter gaussian blur?

Now that a visual representation has been established, the math can be presented as well. The Gaussian filter is defined by the following equation:

$$G(x,y) = \frac{1}{2*\pi*\sigma^2} * \exp -\frac{x^2+y^2}{2*\sigma^2} \tag{4.57}$$

Where:
$G(x,y)$ is the Gaussian function on a pixel coordinate x,y.
σ is the standard deviation of the Gaussian distribution, determining how much the filter should smooth. A common choice would be a value of 1.4.

The Gaussian kernel size is determined as a function of σ by the following rule of thumb:

$$k = 2*ceil(3\sigma)+1 \tag{4.58}$$

Yielding a 7x7 kernel, if a σ value of 1 is chosen. To explain the formula further, the 3 σ value is chosen, as 99.7% of the values in a Gaussian distribution are within 3 standard deviations. The ceil() function is there to ensure kernel size will be an integer, as the ceil() function rounds floating points up. Furthermore, the kernel has to be an odd size, therefore, 1 is added at the end. The kernel must be odd so that the convolution made with the Gaussian blur is symmetrically applied to radii from the central pixel. It is important to remember that this is just a rule of thumb, and the only true rules for a

Gaussian kernel is that it should be of odd size and symmetrical. A smaller kernel could very well work, if the smoothing quality does not have to be high and computational speed is of more importance. Smaller σ values will be sufficient if the goal is to blur the image minimally, while a larger σ will result in a stronger blurring effect and stronger smoothing, which could be beneficial for high noise images.

Now that the kernel size is determined, the actual kernel can be designed. Example kernels of 3x3 with sigma values of 1 and a 5x5 with sigma 2, is shown in equations 4.60 through 4.84. The Gaussian filter is calculated using the kernel values at each slot in the matrix. In the kernel, the centered value which all the values have to convolute around is located in 0,0, and all other values are located in integer coordinates away from 0,0. Using the Gaussian filter shown in equation 4.57, and using a sigma of 1 results in the following equation:

$$G(0,0) = \frac{1}{2 * \pi * 1^2} * \exp - \frac{0^2 + 0^2}{2 * 1^2} \tag{4.59}$$

Which gives the following kernel, when all coordinates are mapped accordingly:

$$\begin{bmatrix} \frac{1}{2*\pi*1^2} * \exp - \frac{-1^2+1^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{1^2+0^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{1^2+1^2}{2*1^2} \\ \frac{1}{2*\pi*1^2} * \exp - \frac{-1^2+0^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{0^2+0^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{1^2+0^2}{2*1^2} \\ \frac{1}{2*\pi*1^2} * \exp - \frac{-1^2+(-1^2)}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{-1^2+0^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{1^2+(-1^2)}{2*1^2} \end{bmatrix} \tag{4.60}$$

$$\downarrow \tag{4.61}$$

$$\begin{bmatrix} 0.059 & 0.097 & 0.059 \\ 0.097 & 0.159 & 0.097 \\ 0.059 & 0.097 & 0.059 \end{bmatrix} \tag{4.62}$$

These are the raw values of the Gaussian kernel, but unfortunately, floating point values increase computing time. Therefore it is important to normalize and scale the kernel. The first step to do so is summing all the values in the kernel:

$$(0.059 + 0.097 + 0.058) + (0.097 + 0.159 + 0.097) + (0.059 + 0.097 + 0.058) = 0.779 \tag{4.63}$$

Thereafter, the sum is normalized to 1. This is done to maintain the overall brightness of the image. In this case, without normalizing the image would only retain 77.9% brightness. To normalize each value, the easiest procedure is dividing the original value with the original sum, and then replacing the original value in the kernel with it. The normalized values are therefore:

$$\frac{0.058}{0.779} \approx 0.074 \tag{4.64}$$

$$\frac{0.097}{0.779} \approx 0.125 \tag{4.65}$$

$$\frac{0.159}{0.779} \approx 0.204 \tag{4.66}$$

Yielding a normalized kernel of:

$$\begin{bmatrix} 0.074 & 0.125 & 0.074 \\ 0.125 & 0.204 & 0.125 \\ 0.074 & 0.125 & 0.074 \end{bmatrix} \tag{4.67}$$

Now, while this might not look very normalized, the trick is to sum all of the values again:

$$(0.074 + 0.125 + 0.074) + (0.125 + 0.204 + 0.125) + (0.074 + 0.125 + 0.074) = 1 \quad (4.68)$$

With the matrix normalized, the next step is scaling the matrix values to integers. When scaling, factors that coincide with 2s complement are desirable, as these are much faster to process in most cases, as division in 2s complement can be done by bit-shifting rather than actual division. Having this in mind, higher powers of 2 demand higher processing power, meaning that the lowest possible power of 2 still yielding an integer value is desired. In this case, the value 16 is chosen, as it fits perfectly with the normalized value of: 0.125. Multiplying the entire matrix by 16 gives:

$$\begin{bmatrix} 0.074*16 & 0.125*16 & 0.074*16 \\ 0.125*16 & 0.204*16 & 0.125*16 \\ 0.074*16 & 0.125*16 & 0.074*16 \end{bmatrix} \rightarrow \frac{1}{16} * \begin{bmatrix} 1.184 & 2 & 1.184 \\ 2 & 3.264 & 2 \\ 1.184 & 2 & 1.184 \end{bmatrix} \quad (4.69)$$

Currently, the values aren't all integers, and this is where the math ceases to dominate, and rules of thumb take over yet again. Before doing so, please note the $\frac{1}{16}$ before the matrix, as this is added to preserve the original values. Instead of rounding off to the nearest integer, the relative proportions of the Gaussian curve have to be taken into consideration, while ensuring the total sum of the kernel is preserved. 1.184 is rounded down to 1, as it resembles a pixel "far" from the center, and therefore isn't as important in the Gaussian distribution. 3.264 is rounded up to 4 to preserve the Gaussian distribution, as this is the single most important pixel in each blur. These roundings assemble the final iteration of the Gaussian kernel, with integers at each coordinate:

$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (4.70)$$

Earlier it was mentioned that the sum should be preserved, and it has been, as both matrices 4.69 and 4.70 have a total sum of 16. This "coincidentally" matches the factor by which the normalized matrix is scaled, which is desired based on [32, 29, 30, 31]. The reason why the sum should match the scaling factor is found by looking at the brightness of the image. If the original sum is 1, 100% of the original brightness is preserved, and when scaling the values by i.e. 16, the sum should also be scaled by 16, to preserve brightness through scaling.

Now that the basic idea is instantiated, the procedure is replicated with a 5x5 matrice:

$$\begin{bmatrix} \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+2^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+2^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+2^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+2^2}{2*1^2}} \\ \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+1^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-1^2+1^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+1^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+1^2}{2*1^2}} \\ \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-1^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{0^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+0^2}{2*1^2}} \\ \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+(-1^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-1^2+(-1^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-1^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+(-1^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+(-1^2)}{2*1^2}} \\ \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+(-2^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+(-2^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+(-2^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+(-2^2)}{2*1^2}} \end{bmatrix} \quad (4.71)$$

$$\downarrow$$

$$\begin{bmatrix} 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \\ 0.013 & 0.059 & 0.097 & 0.059 & 0.013 \\ 0.022 & 0.097 & 0.159 & 0.097 & 0.022 \\ 0.013 & 0.059 & 0.097 & 0.059 & 0.013 \\ 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \end{bmatrix} \tag{4.72}$$

These are the raw values of a 5x5 Gaussian kernel. The next operation is normalizing and scaling the kernel. The first step to do so is summing all the values in the kernel:

$$(0.003 + 0.013 + 0.022 + 0.013 + 0.003) + (0.013 + 0.059 + 0.097 + 0.059 + 0.013) + (0.022 + 0.097 + 0.159 + 0.0 \tag{4.73}$$

Thereafter, the sum is normalized to 1. The normalized values are found in the same manner as for the 3x3 matrix and are therefore:

$$\frac{0.003}{0.987} \approx 0.00304 \tag{4.74}$$

$$\frac{0.013}{0.987} \approx 0.01317 \tag{4.75}$$

$$\frac{0.022}{0.987} \approx 0.02229 \tag{4.76}$$

$$\frac{0.058}{0.987} \approx 0.05876 \tag{4.77}$$

$$\frac{0.097}{0.987} \approx 0.09828 \tag{4.78}$$

$$\frac{0.159}{0.987} \approx 0.16109 \tag{4.79}$$

Yielding a normalized kernel of:

$$\begin{bmatrix} 0.00304 & 0.01317 & 0.02229 & 0.01317 & 0.00304 \\ 0.01317 & 0.05876 & 0.09828 & 0.05876 & 0.01317 \\ 0.02229 & 0.09828 & 0.16109 & 0.09828 & 0.02229 \\ 0.01317 & 0.05876 & 0.09828 & 0.05876 & 0.01317 \\ 0.00304 & 0.01317 & 0.02229 & 0.01317 & 0.00304 \end{bmatrix} \tag{4.80}$$

Now, while this might not look very normalized, the trick is to sum all of the values again[4]:

$$\begin{aligned} &(0.00304 + 0.01317 + 0.02229 + 0.01317 + 0.00304) + \\ &(0.01317 + 0.05876 + 0.09828 + 0.05876 + 0.01317) + \\ &(0.02229 + 0.09828 + 0.16109 + 0.09828 + 0.02229) + \\ &(0.01317 + 0.05876 + 0.09828 + 0.05876 + 0.01317) + \\ &(0.00304 + 0.01317 + 0.02229 + 0.01317 + 0.00304) \approx 1 \end{aligned} \tag{4.81}$$

To keep computation low, a scale factor of 16 is used.

[4] If the actual divisions are set up, the result is exactly 1.

$$\begin{bmatrix} 0.00304*16 & 0.01317*16 & 0.02229*16 & 0.01317*16 & 0.00304*16 \\ 0.01317*16 & 0.05876*16 & 0.09828*16 & 0.05876*16 & 0.01317*16 \\ 0.02229*16 & 0.09828*16 & 0.16109*16 & 0.09828*16 & 0.02229*16 \\ 0.01317*16 & 0.05876*16 & 0.09828*16 & 0.05876*16 & 0.01317*16 \\ 0.00304*16 & 0.01317*16 & 0.02229*16 & 0.01317*16 & 0.00304*16 \end{bmatrix} \tag{4.82}$$

$$\downarrow$$

$$\frac{1}{16} * \begin{bmatrix} 0.048 & 0.210 & 0.356 & 0.210 & 0.048 \\ 0.210 & 0.956 & 1.57 & 0.956 & 0.210 \\ 0.356 & 1.57 & 2.58 & 1.57 & 0.356 \\ 0.210 & 0.956 & 1.57 & 0.956 & 0.210 \\ 0.048 & 0.210 & 0.356 & 0.210 & 0.048 \end{bmatrix} \tag{4.83}$$

Now, while the values within the matrix are still floating points, the same un-mathematical deduction as earlier is made. The values seem proportional to what is desired of a Gaussian distribution, except for being floating point. Therefore, they're all multiplied by 10 and subjectively round to the nearest integer above or below 0.5. This yields a scaled and normalized matrix seen in 4.84:

$$\frac{1}{160} * \begin{bmatrix} 0 & 2 & 4 & 2 & 0 \\ 2 & 9 & 16 & 9 & 2 \\ 4 & 16 & 26 & 16 & 4 \\ 2 & 9 & 16 & 9 & 2 \\ 0 & 2 & 4 & 2 & 0 \end{bmatrix} \tag{4.84}$$

But what about matching the scale factor to the sum of the matrix? In this case, the matrix sums to 158, which is far from the 16 scaling factor, until the perspective is turned to a scale factor of 160 (16*10). Now the discrepancy is only "2", which can be found in the four corners of the matrix, each containing approximately 0.5. If they had not been rounded off, the matrix sum would match the scaling factor perfectly.

Sobel Operator

As the image has been smoothed, the next step is localizing the actual edges. This will be done using the Sobel operator. Similar to the Gaussian blur, the Sobel operator functions with a 3x3 matrix centered on each pixel and is used on all pixels before their values are updated. A deviance from the Gaussian blur is that the Sobel operator is compromised of two operations, as it has a distinct operator for the X and Y directions:

$$G_{(x)} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \; and \; G_{(Y)} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{4.85}$$

After having used both operators on the initial values, the results can be put together for a final result. If the operations are not summed together, the resulting images will be quite different, as can be seen in figure 4.26:

To sum the results correctly, the Pythagorean theorem must be used, as takes both new values and converts them to a square-rooted sum:

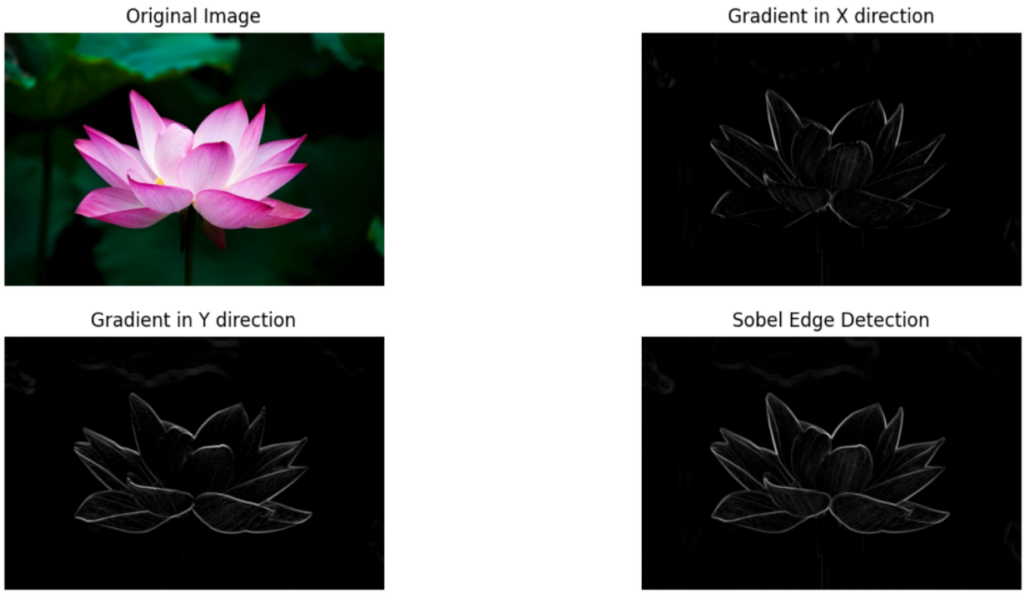


Figure 4.26: Comparison of images exposed to the Sobel, [33].

$$G = \sqrt{G_{(X)}^2 + G_{(Y)}^2} \tag{4.86}$$

Now that a final sum has been found for each pixel, it means that every edge should be visible in the image. But the Sobel operator is capable of yielding even more information, as the inverse tangent to each X and Y value can expose which orientation the edge has at that coordinate:

$$O = \arctan(\frac{G_{(X)}}{G_{(Y)}} \tag{4.87}$$

The result of using inverse tangent on all edges is shown visually in figure 4.27. While this information might not seem useful, it becomes indispensable in the next step, found at page 47.

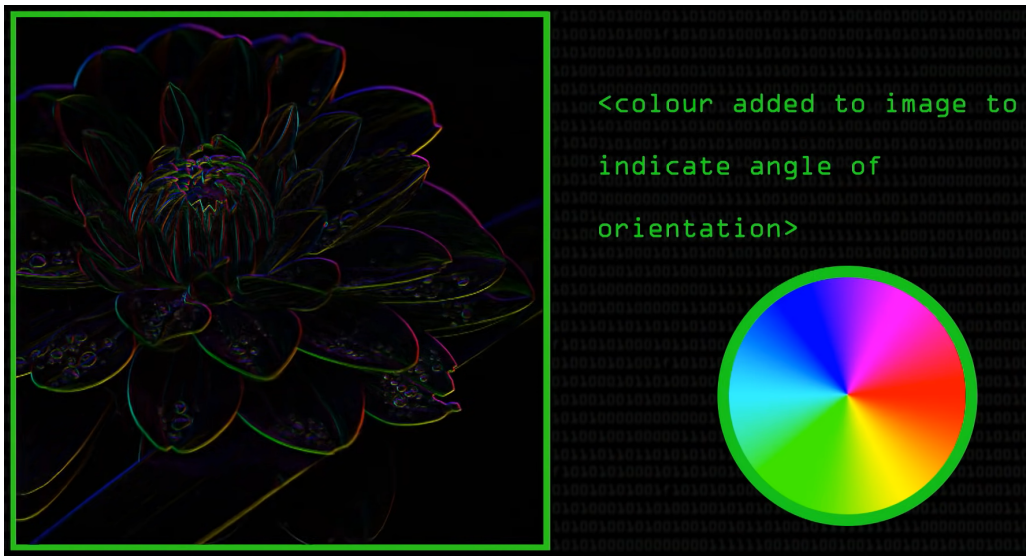


Figure 4.27: The circular RGB disc is meant to represent which orientation each edge has. The actual image is in grayscale, the colors are only added to show orientation. Image courtesy of [30].

Non-maximum Suppression

With the edge found and its orientation known, the next step is to thin the edges to 1-pixel width. This is beneficial for further processing as it sharpens the edge and removes a lot of "noise" that up until now was useful.
To thin the edge, the edge's orientation and its value are needed. The orientation tells us which pixels should be compared, and the value signifies which should be the remaining pixel. In figure 4.28 an illustration of how each pixel is compared to its adjacent pixels in its respective orientation is shown.

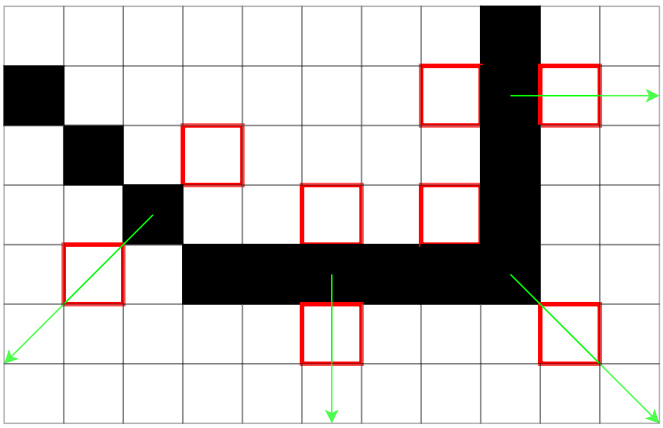


Figure 4.28: The black line has been reduced to 1 pixel wide, by comparing all the adjacent pixels (red border) in the correct orientation (green arrow) with each pixel along the line.

It should be noted that the decision-making is purely based on which pixel has the highest value and that no other operators are used on the pixels. When the highest-valued pixels are found for each orientation, the rest are set to 0, leaving only a thin edge in the picture, where the remaining pixels have retained their values, [31].

Double Thresholding and Edge Tracking by Hysteresis

The next step is meant to sort between strong edges, weak edges, and any remaining noise. Double thresholding sets a definitive value for which values should be kept and which to sort away. The "double" part works by a second threshold being set lower than the initial value. This threshold is meant to connect "weaker" edges with strong edges, as a requirement for being within the second threshold is that at least 1 adjacent pixel must be connected with a "strong" edge, identified by the 1st threshold. By hysteresis, the pixel connected to a strong edge can be 100 pixels away, but as long they are connected, hysteresis will join them as an edge, [34]. All pixels with values between the first and second threshold not connected to a "strong" edge, will be sorted out, along with any pixels with values lower than the second threshold.

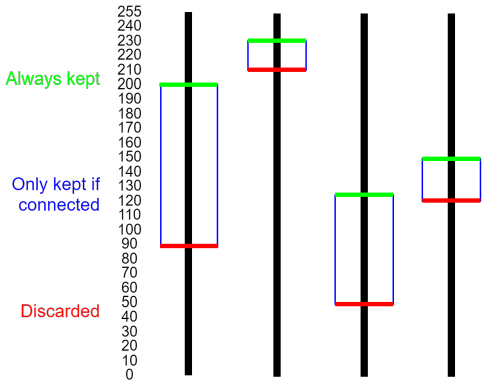


Figure 4.29: Various double threshold scenarios. The pixel values are along the Y-axis, ranging from 0-255 (8-bit).

The threshold utilized for this step is subjectively decided for the first iterations, depending on what edges the user would like to extrapolate from the image. By recursion, it is possible to find suiting values that fit exactly for each implementation. Image quality will have a rather large impact on this step, as larger resolution images inevitably have more edges, and depending on brightness, threshold values could be near each other and in general high. Examples of how these thresholds can be made are seen in figure 4.29.

4.4.2 Object Detection

Object detection is not necessary per se for the prototype, however, it would benefit greatly from being able to identify possible obstacles. Apart from classifying obstacles, object detection will be a key operation to create optimal paths. The objects in question related to a de-weeding robot are:

- Weeds in the pavement
- Grass - as in a lawn
- Asphalt
- Sticks
- Leaf-piles

In a final system, several other objects should be added as well, the above are the essential objects, as these could have a great impact on the operation.

To avoid developing object detection software from scratch, Edge Impulse will be used to train a model capable of detecting essential objects. Furthermore, a general approach will be discussed, handling the subjects within object detection on a surface level. The overall steps in any object detection application are:

1. **Data Acquisition:** The first step is acquiring photos of each object from multiple angles, and if different versions exist (such as for sticks) they should be added as well.
2. **Preprocessing Acquired Data:** Preprocessing consists of noise reduction, contrast correction, and various other image filters.
3. **Feature Extraction:** Extracting the features is meant to identify textures, shapes, colors, edges, etc. so that the model can be trained based on relevant features.
4. **Training the Model:** When using a machine learning model such as Convolutional Neural Networks (CNN) larger datasets of labeled images are better than smaller datasets. With a larger dataset, the model learns to associate features with specific objects with greater certainty.
5. **Testing the Model:** When the model has been trained, the model is tested on different datasets containing the same objects. The results of this will be the location and name of each object within the image, along with a confidence score.
6. **Deployment of the Model:** When deploying the model to a system, system constraints has to be accounted for. Larger systems can often contain better-trained models capable of detecting more objects.

When deployed and in use, the object detection will return an image with bounding boxes containing each object. Bounding boxes are the colored boxes surrounding each object, often seen when seeing object detection in action, and are crucial to describing which objects are present and where they are. Accompanying the bounding boxes is a confidence score, revealing how much or little the algorithm believes it is correct regarding the object.

A general struggle present in object detection is discerning objects from a cluttered background. A countermeasure to this is using the Deformable Parts Model (DPM) introduced by Felzenzwab et al in 2008, as this model objects as a collection of parts, where any part is detected individually. Unfortunately, this model is computationally expensive and hard to scale, especially for more complex detection tasks. Therefore, a Haar cascade will be used, as it is a computationally cheaper model, and is faster to implement with smaller training models. Moreover, Haar cascades use edges and line information, which is already obtained from Canny Edge detection. The actual implementation of machine learning models will be done in the system design phase, see section 5.3 on page 65, where the specific models used are also explained further.

4.4.3 Pattern Recognition

In this specific case, pattern recognition refers to recognizing patterns in pavement, and distinguishing between different tile sizes and pavement styles. This capability will optimize path planning, which is further described in section 4.5.2 starting page 54. The entirety of this section is based upon: [35, 36, 37, 38, 39, 40, 41, 42].

The Hough transform will be used to recognize patterns and create paths of interest. The Hough transform is capable of analyzing the edges extrapolated from edge detection, and combining them in a logical pattern. Moreover, the Hough transform can be used to find geometric patterns in an image based on known parameters, which is desired for recognizing pavement styles. The Hough transform transforms edges in an image from "image space" to "parameter space" (also known as "Hough space"). In parameter space, instead of being a connected line, points along the edge in image space will be translated to lines in parameter space. The lines in parameter space will intersect at 1 point, which contains the mathematical description of the line in image space. In figure 4.30 the correlation between image space and parameter space is visualized.

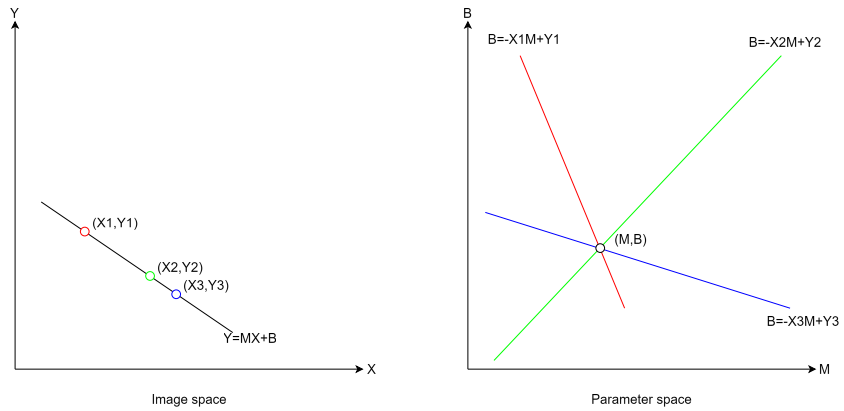


Figure 4.30: Correlation between image space and parameter space when the Hough transform is used.

Mathematically, the lines in parameter space can be seen as "lines" of value 1 in a matrix, where intersecting points will receive a larger weight, as more lines crossing the same cubicle increases the value by 1 each time, visualized in figure 4.31.

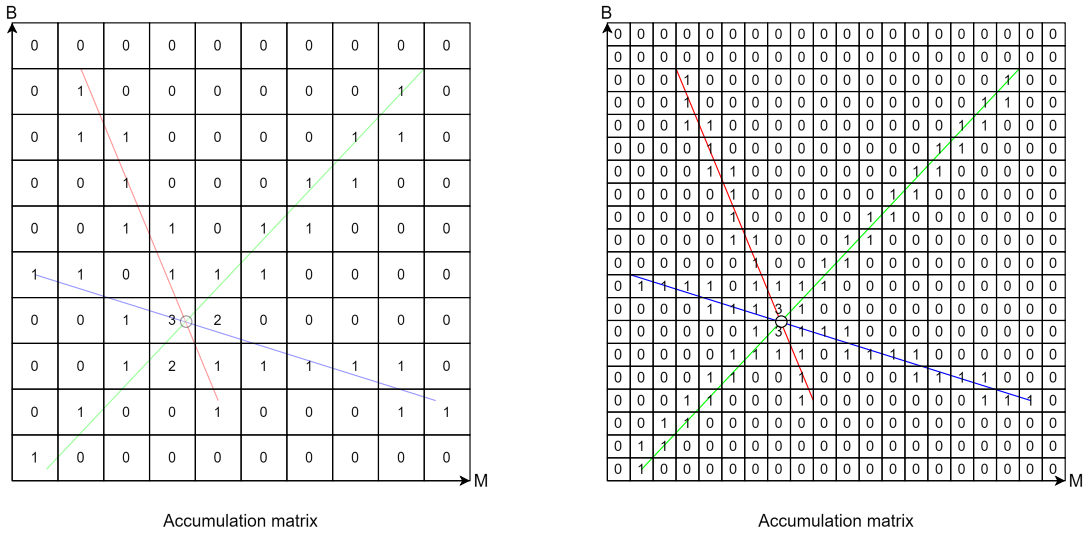


Figure 4.31: As increasing amounts of lines intersect, the common intersection weight also increases, revealing a maximum. To the right is an accumulation matrix with 4 times the resolution, but the maxima have been spread now, leaving ambiguity for which value is the correct.

This is fine as long as the Hough transform isn't met with a vertical line, as a vertical line is impossible to represent in an x,y coordinate system since its slope will be infinite. Therefore, polar coordinates are used instead. Every point in a cartesian coordinate system x,y such as an image, can be described using the angle (θ) and distance (ρ). The resulting parameter space will be a sinusoidal function.

When observing the line in image space, any line can be described with:

$$y = mx + b \tag{4.88}$$

With m being the slope and b the y-axis intercept. Using polar coordinates instead, the line is described as:

$$x * sin(\theta) - y * cos(\theta) = \rho \tag{4.89}$$

Where ρ is the perpendicular distance from the origo to the edge (basically a single point along the edge). θ is the angle between the perpendicular line and the x-axis. X and y are the cartesian coordinates along the edge, where the perpendicular line meets the edge. For each point in the image space, equation 4.89 produces a sinusoid in parameter space. Just as with cartesian coordinates, maxima will be revealed in an accumulation matrix, as intersections between sinusoids will increase the weighting. Therefore, the resolution of said accumulation matrix is important to consider, as the values have to be discretized. A regular interval will be in whole degrees (0-180) and r will be the diagonal maximum value of the image in pixels. While it may seem preferable to just increase resolution, figure 4.31 shows possible complications. In figure 4.32, sinusoids derived from an edge can be seen, along with corresponding maxima.

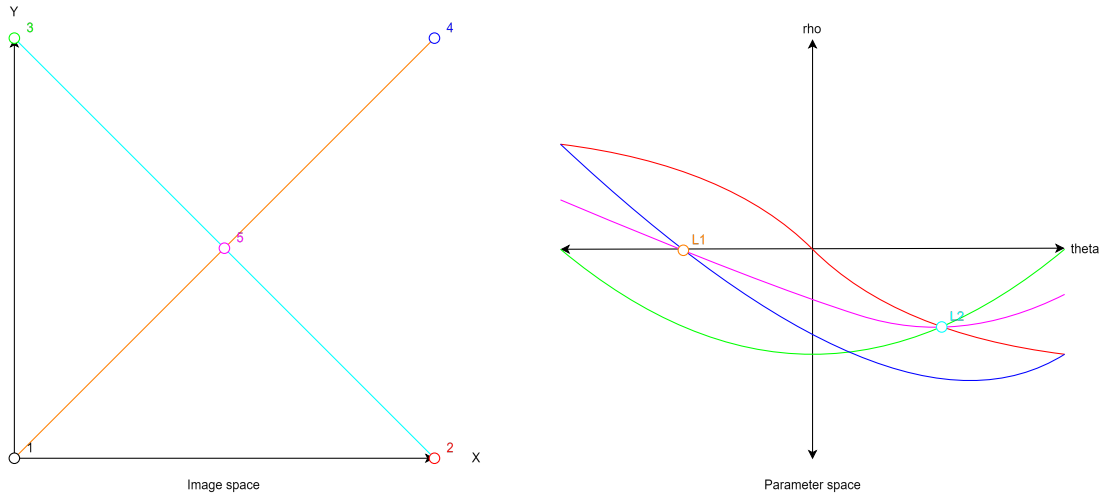


Figure 4.32: Sinusiodal representation in parameter space. The intersections (maxima) in parameter space translate to lines in image space, as shown by L1 and L2.

4.5 Mapping

Mapping the area in which the system will operate is preferable. The reason for this is the possibility of path planning and optimizing said paths, which otherwise would not be possible. Mapping out an area consists of finding boundaries, localizing a home, remembering obstacles, and possibly dividing into zones to accommodate the different needs of each area, allowing a systematical approach.

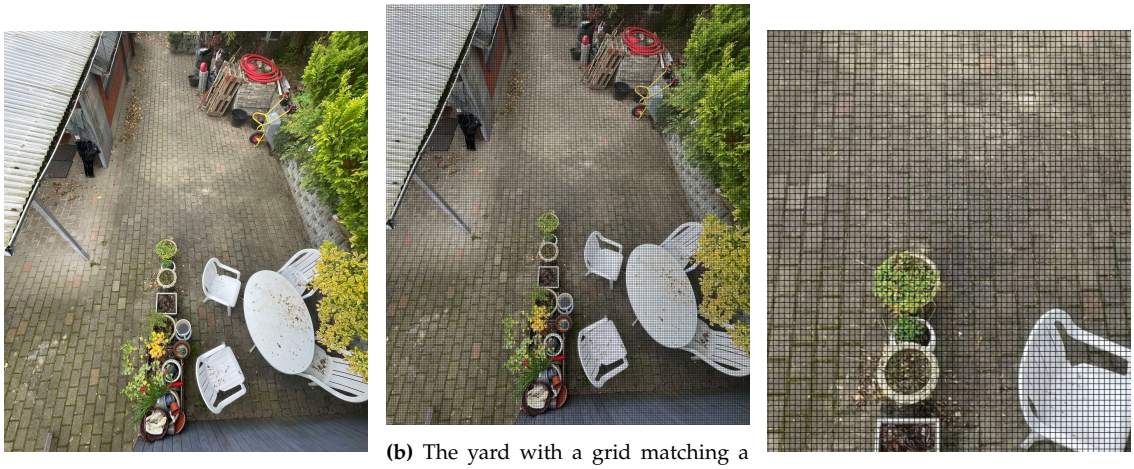
4.5.1 Instantiating a Coordinate System

Before any of the smart features related to having an area mapped out can be utilized, the first step is instantiating a coordinate system, by which the area can be described. As the system is not interested in the Z-axis, the coordinate system only has to be two-dimensional. Furthermore, the coordinate system should be cartesian to represent the surface realistically. The resolution of the coordinate system should represent the area in which the robot should operate. A suitable size depends on the area, which the coordinate system should represent.

According to "Boligsiden", most plots for regular housing in Denmark are below 1100 m^2, [43]. Ignoring that all plots are not square, the assumed coordinate system should cover 40x40 meters ($1600m^2$), to encompass most regular houses. The corresponding representation/image for the coordinate system will be 1000x1000 pixels, to match the scale used in "Vision Based Autonomous Robot Navigation", [24]. In the book, a 500x500 pixel coordinate system is used to describe an area of 20x20 meters. The resolution of these coordinate systems corresponds to 4 cm per pixel, which is decent enough to operate the robot for mapping purposes. In figure 4.33a a picture of the author's yard is shown, and in 4.33b the same yard with a grid approximately equivalent to a 1000x100 coordinate system. Figure 4.33c shows the same, but in a zoomed-in version. The figure showcases that even though 4 cm per pixel seems to leave a large error margin, it should be sufficient to map most areas and pavement styles.

Figure 4.33 is, of course, a little skewed, as it does not showcase the area in a direct

view from the top, which a final coordinate system should. It should be noted that the coordinate system and its representation of the area would be derived from pictures taken by the robot at a height of 15 cm above ground. Therefore, the representation in a coordinate system should resemble a direct bird's-eye view of the property in the end.



(a) A part of the author's yard.

(b) The yard with a grid matching a 1000x1000 coordinate system.

(c) A zoomed-in version of 4.33c.

Figure 4.33: The author's yard, with and without a corresponding coordinate system mapped onto it.

Conversion from Picture to Cartesian Coordinates

A model has to be made to convert the pictures taken by the robot to cartesian coordinates. Instead of calculating the relationship between pixels in the images and translating it to cartesian coordinates, the translation will be made under the following assumptions:

- The camera will at all times be at the same height
- The surface on which the system is traveling is flat

With those assumptions in place, creating a look-up table that translates pixel coordinates to cartesian coordinates is possible. This is done by taking a reference image shown in figure 4.34, and then translating each pixel address to a known cartesian coordinate. In figure 4.35 a top-down view of the image shown in 4.34 is seen. This trapezoid makes it possible to translate pixel coordinates to various distances relative to the camera. Some examples of this are shown in table 4.6 and figure 4.35. It should be noted that the cartesian coordinates are in cm, as they are based on reading the image in figure 4.34 and all coordinates are mapped from the top left corner of the trapezoid.

However, translating the image coordinates of a trapezoid to cartesian coordinates is a mess, as image coordinates start in the top left corner, which warps the coordinates. Therefore an adjusted coordinate system is used, with the origo in the center bottom of the trapezoid. With the origo centered in the trapezoid, coordinate warping along the x-axis is the same on both sides, while maintaining a coherent relationship with the y-axis (forward). However, it should be noted that each pixel's "resolution" is decreased as the y value increases. This is because the pixels represent an increasing amount of the surface, the further from the camera the surface is.

Image Coordinate	Cartesian Coordinate	Adjusted Origo Image Coordinate	Adjusted Origo Cartesian Coordinate
0,0	0,0	0,0	0,0
80,0	23,0	-80,0	-9,0
160,0	46,0	80,0	9,0
0,60	0,27	0,60	0,13
0,120	0,40	0,120	0,40
80,60	10,27	80,60	13,13
80,120	14,40	80,120	23,40
160,120	32,40	-80,120	-23,40

Table 4.6: Table of correlation between image coordinates (u,v) and cartesian coordinates (x,y). As can be seen, an adjusted version is easier to comprehend, than a strict translation.

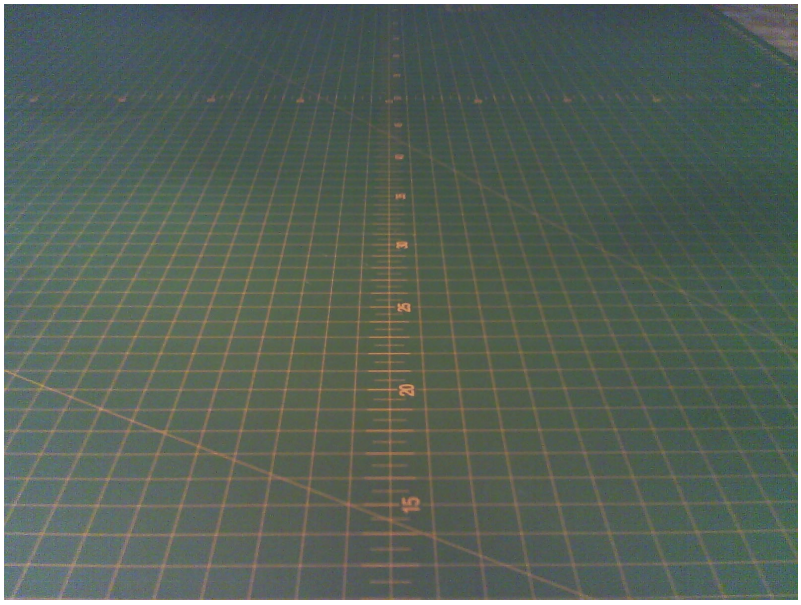


Figure 4.34: Reference image taken of a 90x60cm cutting mat. The image shown here is in 1600x1200 resolution. Each block in the image is 1x1 cm.

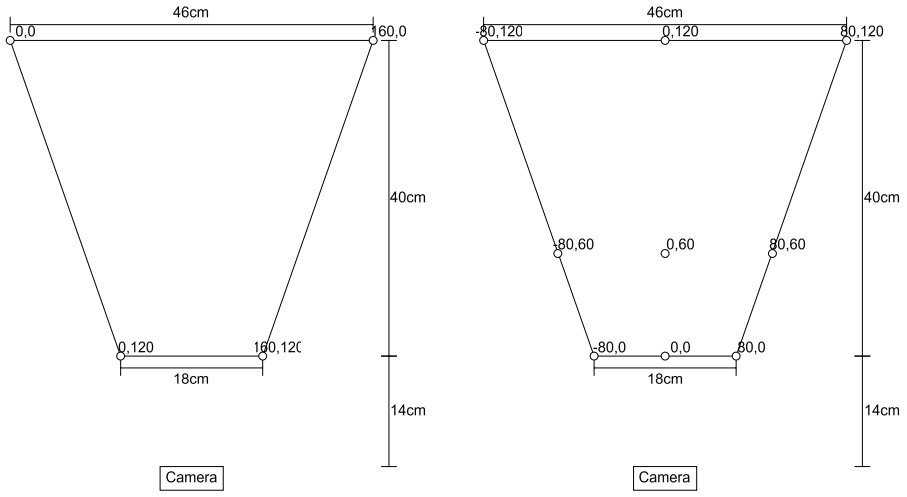


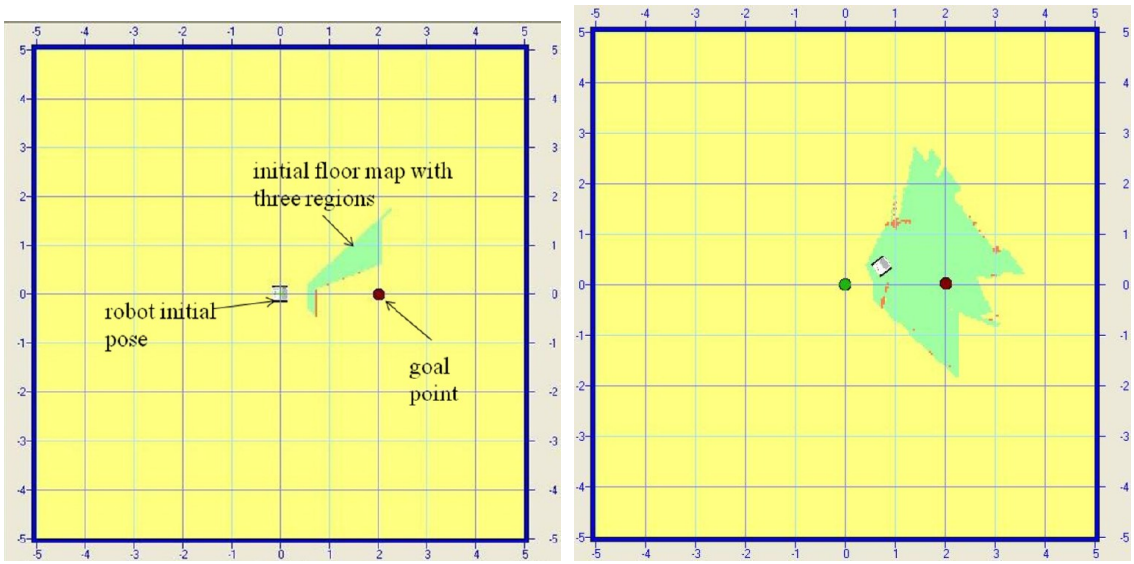
Figure 4.35: Top-down view of what area the camera sees, and how picture coordinates translate to cartesian coordinates. The resolution is reduced to 160x120 to minimize computing costs.

Now that the images can be translated into cartesian coordinates, with a precision of approximately 1mm close to the robot, it is possible to map out entire areas. This is done by differentiating between obstacles, grass, asphalt, and the lines in pavement, before translating them into coordinates. With this translation in place, the 1000x1000 pixel map of the area can be made.

Furthermore, the knowledge of how lines in the pavement orient up to 54 cm in front of the robot, will be used to control the speed and direction of the robot. By this, it is meant as the speed can be increased as long as the angle difference is sufficiently small, otherwise, the speed must be reduced to allow for correction of the angle.

4.5.2 A Systematical Approach

As a map can now be instantiated through images, it becomes possible to map out lines in the pavement and adjust the behavior of the robot. To create the map, images taken by the robot must continuously be added together, to create a coherent map of lines, obstacles, etc., as shown in figure 4.36 taken from [24]. In the figure, yellow signifies an undiscovered area, red is obstacles and green is a viable path. In this prototype, additional markings will be added to the lines in the pavement. With the knowledge presented, it is possible to create algorithms for moving systematically around. It should, however, be noted, that optimized systematical routes can first be established, when the entire area has been mapped.



(a) Initial mapping created by Chatterjee, [24].

(b) Continous mapping created by Chatterjee, [24].

Figure 4.36: The general idea of how areas will be mapped using Chatterjees model, [24].

4.6 Power Monitoring

Power monitoring is a necessary implementation for any battery-powered device. In this specific case, the primary goal of power monitoring is to guarantee sufficient energy remains, to return "home" to a charge point. The easiest implementation is continuously measuring the battery voltage, and corresponding it with a battery percentage, also known as "state of charge" (SoC). As some version of lithium battery will be used, this is the only case worth examining. Apart from "just" measuring the SoC, it has to be mapped to a distance that can be traveled. The resulting graph will be along the lines of the idealized graph in Figure 4.38, as it illustrates the relationship between battery percentage and the remaining distance the system can cover. Measuring the voltage will be done by the ESP32, utilizing the ADC ports, [44, 45].

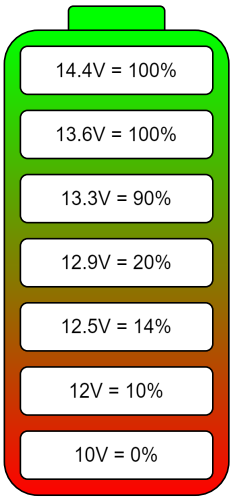


Figure 4.37: Various battery percentages for a lithium battery. Values are taken from [44, 45].

The state of charge (SoC) can be directly translated to the amount of Ah left in the battery, and can be measured by voltage or counting coulombs. Counting coulombs refers to monitoring the exact current used in the battery, and for how long, giving a value in Ah. Counting coulombs is by far the most precise method, but requires deep knowledge of current use in every aspect of the system. In an ideal world, the battery will not lose Ah over time, but in reality, the Ah capacity of a battery is reliant on charge cycles, temperature, quality of the cells, etc. and therefore voltage isn't the most precise method, as the voltage does not take this into account, whereas coulomb counting both charge and discharge, tells exactly how many Ah is left, [44, 45].

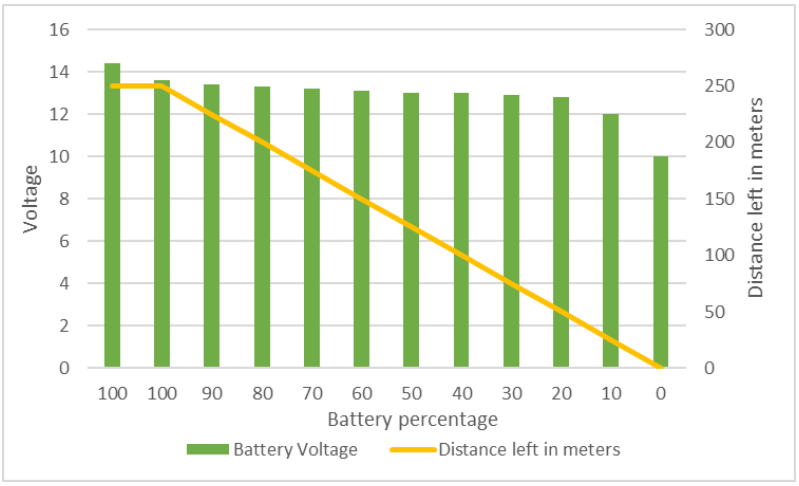


Figure 4.38: Idealized correlation between voltage and distance left, that the system can travel.

4.7 Wireless Communication

Wireless communication is desired in the project to transmit data across several processing units and microcontroller, enabling future iterations be much smarter than this prototype. Furthermore, the implementation of wireless communication makes room for a host of other functionalities, such as an user interface, Kalman localization, remote control capabilities, and much more. However, as this is still a prototype, the latter functionalitites are not prioritized yet. Instead, this sections strives to explain the ESPNOW protocol and its workings, as it will be used to transmit data between ESP boards, enabling the possibility of adding more than MCU's to the system. This design choice is further explained in 5.2 starting page 61.

4.7.1 ESPNOW

The ESPNOW protocol is developed by Espressif Systems as a dedicated wireless communication API for use on ESP-boards. The API is developed using the "WiFi.h" library for ESP and Arduino, and simplifies communication between several ESP's. Furthermore, ESPNOW is an ad-hoc communication protocol using WiFi physical layers and MAC addressing for direct device to device communication. This results in ESPNOW being fully functional independently of external routers or other internet infrastructure. The protocol can be compared to Bluetooth, as this is also a direct device to device communication protocol. Furthermore, the ESPNOW protocol has higher data rates, multiple peer support, and integration with existing WiFi systems implemented as well. In short, the ESPNOW's capabilities can be summarized to:

- **Low latency**
  ESPNOW is capable of low-latency communication (1-3ms) and supports transmission of up to 1 Mbps.
- **Independent communication**
  As the ESPNOW protocol is utilizing direct device to device communication, no external wireless infrastructure is needed.
- **Broadcast and unicast**
  A single board can act as "master" and broadcast to several other boards acting as "slaves", or several "masters" can send packets to a single "slave".
- **Multiple peer to peer**
  On ESP32, the ESPNOW protocol is capable of connection with up to 20 other devices.
- **Encryption**
  ESPNOW supports encryption of data with up to 17 different devices at a time, depending on firmware configurations. Standard is 7 devices.
- **Efficient power consumption**
  If ever needed, the entire API is designed to be useable for low-power applications.
- **Integrateable with existing WiFi protocols**
  As the ESPNOW isn't operating on standard WiFi systems, it can coexist with WiFi protocols, enabling ESP boards to switch seamlessly between ESPNOW and WiFi functionality.

- **Large payload size**
  An ESPNOW packet can carry up to 250 bytes, depending on encryption settings.

The ESPNOW Protocol

As mentioned earlier, the ESPNOW protocol is a lightweight communications protocol utilizing the physical WiFi layers and MAC layers. To expand on this, the protocol does not employ any higher-level networking layers such as TCP/UDP or IP. A complete walkthrough of how ESPNOW acts based on the OSI model is made in section 4.7.1. The ESPNOW protocol can be divided into the following steps:

Initialization

The master device initializes the ESPNOW protocol using "esp_now_init()", setting up the MAC communication layer. Remember that the MAC addresses have to be set before the code is uploaded.

Peer Registration

The master device registers any receiver MAC addresses with the "esp_now_add_peer()" function. This directly connects the two devices, and enables reconnection in case connection is lost. If encryption is enabled, encryption keys will also be provided at this point.

Packet Creation

The master device formats the packet to include payload, possible encryption etc..

Field	Value (HEX)	Description
Frame Control	`0x08`	Indicates a data frame.
Duration/ID	`0x00`	Duration field.
Destination Address	`FF:FF:FF:FF:FF:FF`	Broadcast address (for multicast).
Source Address	`24:6F:28:D9:8A:B0`	MAC address of the sender.
BSSID	`FF:FF:FF:FF:FF:FF`	Randomized value for ESP-NOW.
Sequence Control	`0x0010`	Sequence number for the packet.
Header	`0x02`	ESP-NOW packet type and length.
Payload	`48 65 6C 6C 6F 20 45 53 50 2D 4E`	Data ("Hello ESP-NOW") in ASCII.
Checksum	`0xDEAD`	Simple checksum for integrity validation

Table 4.7: Typical ESP-NOW packet structure.

Data Transmission

As a connection has been made, data transmission can be initiated. The master device sends packets using "esp_now_send()", which takes the peer address, the data to transmit, and the "sizeof()" the data as parameters. By doing so, the master can send packets to specific slaves, or broadcast to all registered peers by setting the value to 0 instead of a specific MAC address.

ACK Mechanism

As in most communication protocols, ACK and NACK signals exist in ESPNOW as well. In ESPNOW, the ACK process happens at the MAC layer using a callback function. If a callback has not been received by the master device within X timesteps (specified by the

user for each application), an automatic NACK is registered and the previous packet is resent.

Recieving Packets
When the slave device receives a packet it will first send a callback, acting as ACK to the master device. After that, packet decoding is initiated, where the slave device extracts the transmitted data, the senders' MAC address, and other metadata.

Encryption
If standard ESPNOW encryption is used, it will be implemented as AES-128 encryption, adding an additional Initialization Vector (IV) and authentication tag to the packet structure.

Data Integrity and Error Handling
Making use of the frame check sequence at the MAC layer (checksum) makes it possible to discard corrupted packets and calling a resend of the previous packet. While the checksum and callback functions can handle many errors, the ESPNOW protocol still lacks higher-level mechanisms to guarantee delivery.

Comparison with the OSI Model

If the OSI layer should be applied to ESPNOW, only the lower two layers, physical and data link, will be used. At the physical layer, ESPNOW transmits and receives data at 2.4GHz, which is also used for many WiFi and Bluetooth systems. At the data link layer, the MAC addressing is found. It is used for frame formatting, ACK, and NACK.

The ESPNOW omits higher-level protocols, eliminating the complexity of TCP handshakes, IP assignments etc.. Furthermore, the lack of routing mechanisms means that no session or connection establishment is initiated, only the direct device to device communication is instantiated.

5 System Design

5.1 System Design Overview for the Prototype

To create an intuitive overview of what elements the prototype will be composed of, it is divided into modules. The modules do not contain specific knowledge at this point. Still, they will mostly serve as an overview of what the different modules are supposed "to do", before the final system design of each module will begin, leading to grounds for designing a system capable of meeting the functional specification.

- Drive motors.
  Motors meant to operate the robot physically - will not be described again, as it is described in "Preliminary Solution" starting page 12.
- Power source.
  A combined power source capable of powering all parts of the robot.
- Power distribution/relay.
  DC-DC converters to match power from the power source to individual systems and relays for switching subsystems on/off.
- Microcontroller.
  The main operating processer, responsible for operating the robot and responding to input from sensors and the Computing module.
- Sensors.
  Sensors meant to enable "spatial awareness" of the robot.
- Camera.
  The main way of determining if the robot is on the correct course is through computer vision.
- Computing.
  The computing module will handle larger computations, such as machine vision/image processing, and possibly be placed externally from the robot.
- Wireless communication.
  The communication module will enable communication between the robot, the charge point, and the user.
- Physical platform.
  The physical platform on which the prototype will be developed - will not be described again, as it is described in "Preliminary Solution" starting page 12.

- Charge point.
  External housing which contains the wireless communication, user interface, and in future iterations, charging capabilities and increased processing power.

From the above list, a block diagram has been made to show interfaces between the modules. The block diagram can be seen in figure 5.1 on page 60.

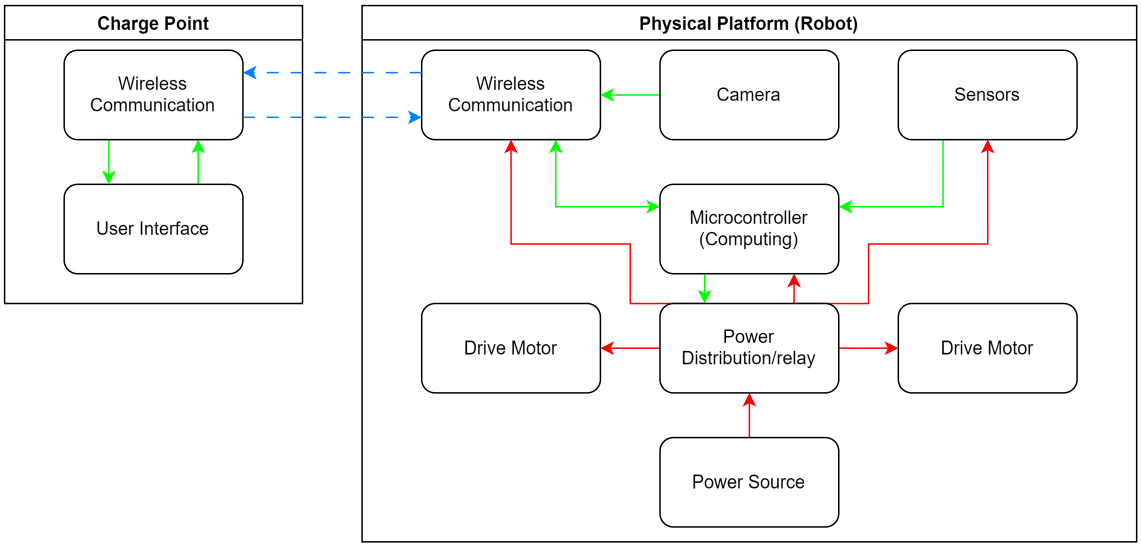


Figure 5.1: Block diagram showing relations between modules of the prototype. Red lines signify power, green signify data, and blue-dotted signify wireless data.

From the block diagram, a procedure can be established for determining each module and how they relate to each other. As power distribution requires knowledge of the voltages needed for each succeeding module, it will be the second-to-last module followed by the power source. The first modules to analyze are the microcontroller and camera, as they both determine the needs of connected modules. At the very end, the physical platform and charge point will be analyzed. This leaves the technical analysis sequence as follows:

1. Microcontroller.
2. Camera.
3. Wireless Communication.
4. Computing.
5. Sensors.
6. Drive Motors.
7. Power Distribution/Relay.
8. Power Source.
9. Physical Platform.
10. Charge Point/User Interface.

5.1.1 Time Constraints

Due to time constraints, several subsystems have to be discarded. Therefore the following elements will **NOT** be made physically: auxiliary sensor integration, power distribution/relay, power source, and charge point/user interface. Furthermore, localization measures such as kalman filters, landmark recognition, and encoders will not be integrated either.

5.2 Microcontroller Module

The microcontroller has to connect to all sensors utilized as well as power distribution and wireless communication. Therefore a large amount of I/O pins will be handy, along with integrated wireless communication, and multiple cores for added processing power. TO ensure stable and coherent operation, integration of RTOS protocols such as FreeRTOS will be beneficial.

5.2.1 Specification

To ensure the correct microcontroller is chosen for the system, the requirements for it are:

- Camera compatibility.
- Wireless communication.
- At least 4 PWM GPIO pins.
- I2C capability for communication with sensors.
- RTOS capabilities for task management.

5.2.2 Design

The first MCU board that comes to mind fitting these criteria would be the ESP32 or Raspberry Pi Pico W, as both employ dual-core processors, integrated WiFi/Bluetooth, and FreeRTOS compatibility. As the main sensor is an ESPCAM, it might be beneficial to use the ESP32, as clockrates and similar specifications are the same across the boards, as both boards utilize an ESP32 chip for computing. While the entire system might be able to run on the ESP32 chip embedded in the ESPCAM board, the board lacks GPIO pins, as the camera module inherent to the ESPCAM occupies more than half of the pins.
The additional ESP32 board will therefore be responsible for:

- PWM output to motors
- Communication with sensors
- Wireless communication with ESPCAM
- Task management
- Clock synchronization between development boards

The motor output will be determined by the image processing and computing element, designed in 5.3 and 5.5, before being interpreted by the MCU module and converted to PWM signals. While this is happening, the other core will be responsible for communication between the ESP boards, and auxiliary sensors, as well as task management and clock synchronization. The inner workings can be seen in figure 5.2, containing the flowchart for the MCU module.

5.2.3 Implementation

Implementation of the ESP board is done by connecting it to the auxiliary sensors and ESPCAM with I2C, utilizing the SDA and SCL lines. Furthermore, an emergency

cutoff button is added and connections are made to the ZK-BM1 H-Bridge responsible for translating PWM signals to voltage for the motors.

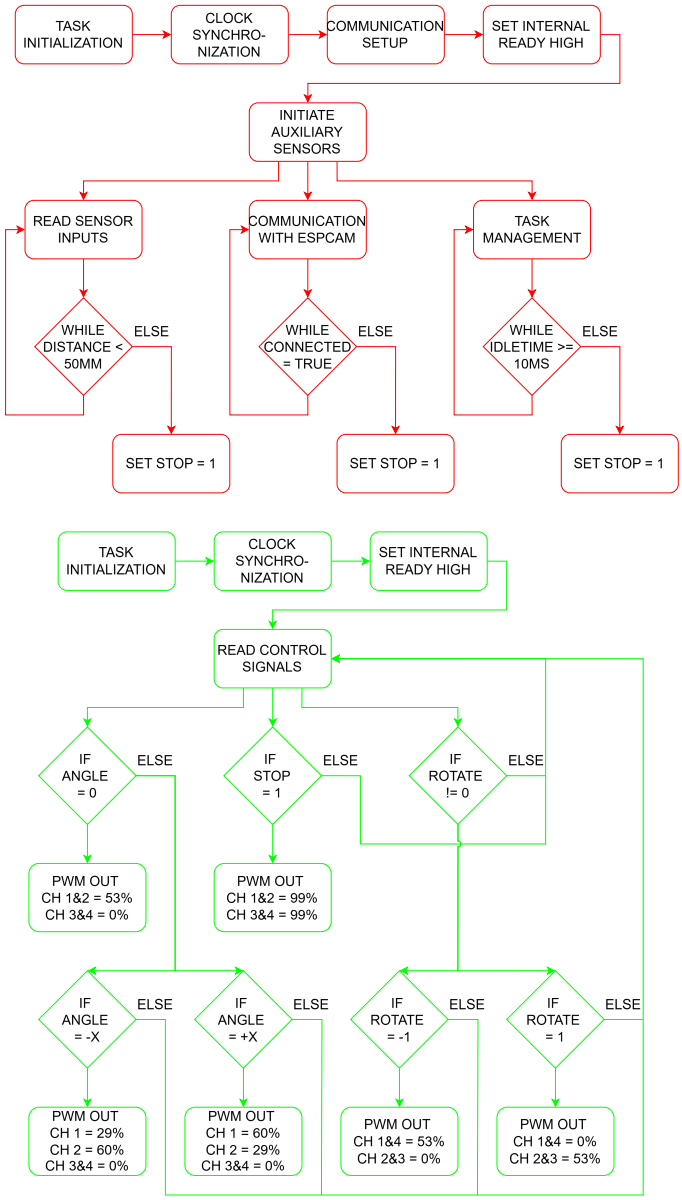


Figure 5.2: Flowchart to describe the procedures implemented on the MCU module. Red signifies operations on core 1 and green core 2.

In figure 5.3 the physical connections can be seen. Notice that the two ESP boards are connected wirelessly. The wireless communication transfers control signals along with synchronization protocols using ESPNOW. Furthermore, wireless communication can be used in future iterations for interpreting values to be used in an user interface. The I2C connection is only used to send serial data from the auxiliary sensors to the ESP32 board.

5.2.4 Test

The MCU module will be tested in the following operations:

(A) Task initialization and management.

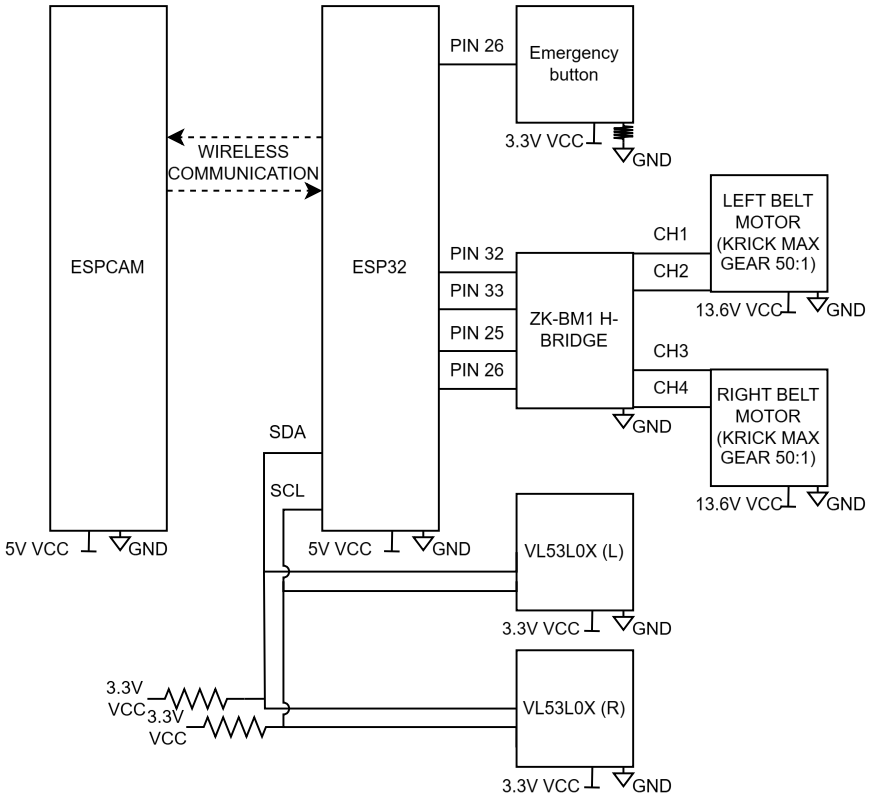


Figure 5.3: Physical connections used to implement the ESP with the ESPCAM and auxiliary systems.

(B) Communication with sensors.

(C) PWM output to motors.

(D) Clock synchronization.

Test Setup

To test the above, the setup seen in figure 5.3 is used with the following parts. Do note that not all tests require all the objects to be used.

- 1x ESPCAM
- 1x ESP32
- 1x VL53L0X ToF sensor
- 1x ZK-BM1 H-Bridge
- 1x Emergency button
- 1x Physical platform with motors
- 1x Oscilloscope, KeySight DSOX1102G
- 1x Triple Power Supply, HAMEG HM7042
- 4x 1m banana test leads (2 for GND and 2 for VCC)
- 1x 1m BNC coax cable
- 1x 0.1m BNC coax cable
- 1x BNC to banana plug adapter
- 1x PC for data logging

Actual Testing

To perform the tests, the following procedure should be used:

1. Connect the ESP to a computer
2. Load the code for each test respectively:
   (A) Code, link is also available in appendix A.3.9.
   (B) Code, link is also available in appendix A.3.12.
   (C) Code, link is also available in appendix A.3.2.
   (D) Code and code , link is also available in appendix A.3.10 and A.3.11.
3. Execute the code
4. Interpret the results shown in the serial monitor and/or the oscilloscope
5. Save results and power off

Test Results

Test A
A FreeRTOS routine has been implemented with dummy delays based on measured execution times +10% to test task scheduling. When the task scheduling is running with dummy delays, the idle task timer is allowed XX% runtime on core 1 and approximately XX% on core 2, leaving plenty of overhead in the RTOS.

Test B
The only sensor tested which will be tested with the ESP32 is the VL53L0X ToF sensor, which will be used as a safety feature if the vehicle comes too close to a wall or other objects. The sensor has been tested by placing it at the following ranges from a wall: 100 mm, 200 mm, 300 mm, 500 mm, 750 mm, 1000 mm, 1500 mm, 2000 mm, and 2200 mm with results visible in the appendix figure B.37 a-i on page 142. In general, the read value was off by 20-50 mm at more than 1.5 meters, but at distances relevant to this project (less than 300mm) the margin is only +-3mm.

Test C
The PWM output of the ESP32 has been tested with the H-Bridge, motors, and oscilloscope connected. During testing, the frequency was set at 2kHz to comply with the operating range of the H-Bridge. The entire span from 1-99% PWM has been tested both with manual code increments and dynamic joystick input. The ESP32 acts as expected and behavior is confirmed with the oscilloscope, with no visible error present when comparing PWM made by the oscilloscope or the ESP32.

Test D
As the wireless communication has not been instantiated at this point in time, testing of it can be found in section 5.4.4 on page 77.

5.2.5 Summary

The ESP32 is more than capable of acting as an additional board to the ESPCAM capable of image processing tasks. The addition of another board enables increases processing power and GPIO pins, enabling the system as a whole to comprehend more tasks. Furthermore, the entire system should be able to run on a single core, if strict task deadlines are met with a framerate of 2 FPS, making the system compliant with the 0.5 second sampling window introduced in 4.2. However, as multiple cores are available, this might be improved in chapter 6 starting page 85.

5.3 Camera Module

To enable the possibility of utilizing computer vision, a camera module has been added to the system. As mentioned in 5.2, an ESPCAM board has been incorporated.

5.3.1 Specification

- QQVGA resolution (160x120pixels)
- Flashlight
- 24 FPS capable
- Connectable to ESP
- Programmable setting
- Output in RGB565, RGB888, RAW, JPEG, or YUV422
- Preferably capable of grayscale photos
- 3.3V or 5V operable
- Flash memory

5.3.2 Design

As mentioned in the introduction to this section, an ESPCAM module has already been chosen. The functionality desired in the module is shown in the flowchart pictured in figure 5.4. In the flowchart it is visible that the camera is initialized at first, before pictures are taken at an, so far, unknown FPS, ensuring a sample rate of 1 sample every XX seconds. The proprietary libraries made for the ESPCAM enable the possibility of converting the image to grayscale with no apparent added compute time. The pictures are then encoded as RGB565 to keep memory requirements as low as possible. Encoding images as RGB565 instead of JPEG or similar formats also removes the compute time used to compress and decompress the image. Furthermore, by using RGB565 as the image format, the image can be represented by a 2-dimensional array, making image processing much easier.

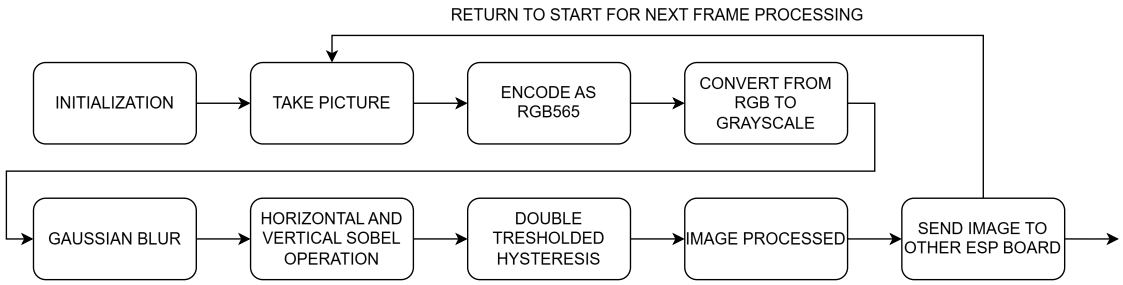


Figure 5.4: Flowchart of the camera modules operation.

With the image in RGB565 and already converted to grayscale, the next step is using image processing kernels introduced in section 4.4.1 starting page 38 to extract relevant data from the image. First, the image is Gaussian blurred with a 3x3 kernel before the Sobel operator is used to extract edges, using double thresholding along with connecting the edges by hysteresis. When the canny edge algorithm has processed the image, control signals will be computed (see sections 5.5 on page 78) and sent to the other ESP board.

5.3.3 Implementation

The camera module is connected to the common power and GND pin, enabling power to the module. Moreover, the ESPCAM will transmit control signals to the main ESP board by a wireless protocol, explained and introduced in section 5.4 on page 75.

5.3.4 Test

The camera module will be tested in the following operations:

(A) Take an image

(B) Grayscale conversion of image

(C) Image representation in a 2-D array

(D) Gaussian blurring

(E) Edge extraction using the Sobel operator

(F) Hysteresis to connect edges and remove noise

(G) Execution time

Test Setup

To perform all of the above tests, a test setup is shown in figure 5.5. The only needed elements to perform the test are:

- ESPCAM board
- USB-C cable
- PC for output interpretation and code upload

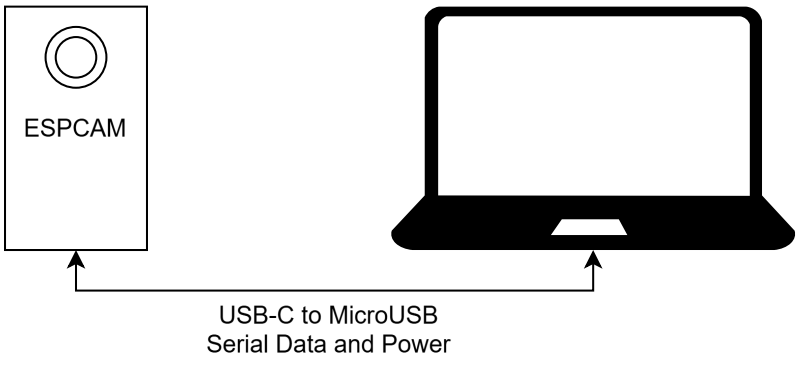


Figure 5.5: Overview of the test setup.

Actual Testing

To perform testing of the ESPCAM, the following procedure should be utilized:

1. Connect the ESPCAM to a computer
2. Load the respective code for each test respectively:

(A) Code, link is also available in appendix A.3.3.
(B) Code, link is also available in appendix A.3.4.
(C) Code, link is also available in appendix A.3.4.
(D) Code, link is also available in appendix A.3.5.
(E) Code, link is also available in appendix A.3.5.
(F) Code, link is also available in appendix A.3.8.
(G) Code, link is also available in appendix A.3.6.

3. Execute the code
4. Interpret the results either in serial monitor or by copying output to Excel
5. If the output is copied to excel, it should first be copied to a .txt document for easier handling. When inserted into Excel, a global rule must be implemented setting lowest value to colour the cell black, and the highest white. To get correct picture representation, all cells must be made equal in height and width as well, to represent a pixel.
6. Save results and power off the board

Test Notes

During testing it became apparent that the flash embedded in the ESPCAM board was initialized at every iteration of the loop running. Therefore its initialization was identified in the source code and disabled at all times. The reason is a timing mismatch between the flash going off and the shutter time of the lens, resulting in unstable images, see figure 5.6. Furthermore, with the flash disabled at all times, edge identification could possibly be easier in situations that have not yet been tested, as only natural shadows will be present.

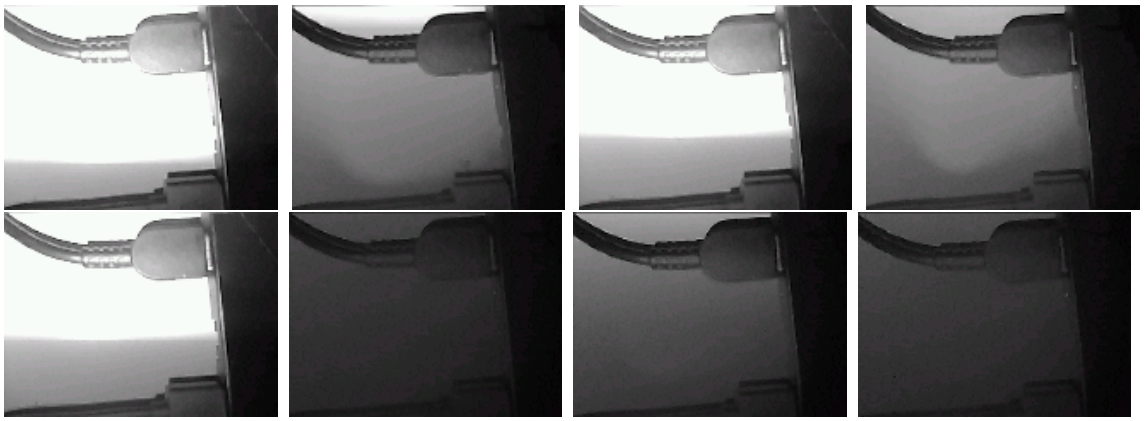


Figure 5.6: Images taken with a delay of 0.5 seconds, showcasing that the intermittent flash causes unstable image quality

Apart from the issues using the flash, it became apparent when trying to save the images as a 2D array, that the images were still RGB565 coded even after using the proprietary grayscale library, and not yet converted to a true 8-bit grayscale image. Because of this, each pixel was still represented by 16 bits. To make future image processing optimal, the code was then refactored so as not to include the proprietary grayscale command, and instead, it remained as 16-bit RGB565 with all colors. A function converting the image to 8-bit grayscale was added to achieve the desired grayscale representation.

Test Results

Test A

Testing the ESPCAM capability of taking an image was done outside and at different resolutions. Even though the resolution requirements are QQVGA (160x120 pixels), the ESPCAM is capable of every resolution up to UXGA (1600x1200 pixels), at the expense of processing time and increased memory usage. In figure 5.7 examples of QQVGA and UXGA resolutions are shown, picturing the same pavement.

(a) UXGA resolution image of pavement.

(b) QQVGA resolution image of pavement.

Figure 5.7: Images of the same piece of pavement at different resolution, taken with the ESPCAM.

By proof of the above pictures, it is visible that the ESPCAM can take an image.

Test B

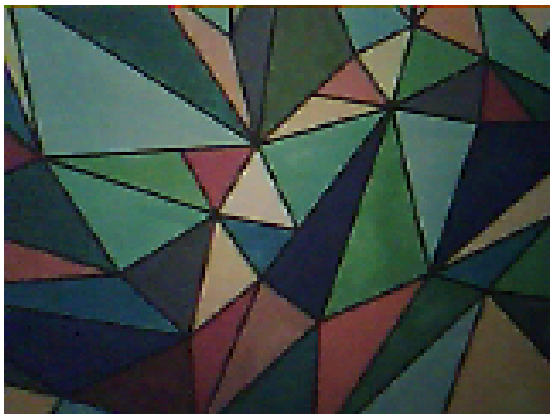
Using the proprietary libraries for the ESPCAM, it is possible to convert any image taken into grayscale. However, while it seems a true grayscale image, it is not. Therefore the proprietary grayscale function has been discarded in favor of a simple function to convert RGB565 to 8-bit grayscale:

```
// Read the pixel values row by row
  for (int y = 0; y < imgHeight; y++) {
    for (int x = 0; x < imgWidth; x++) {
      if (file.available()) {
        uint8_t byte1 = file.read();
        uint8_t byte2 = file.read();
        uint16_t pixel = (byte1 << 8) | byte2;  // RGB565 format
        // Extract RGB components
        uint8_t r = ((pixel >> 11) & 0x1F) * 255 / 31;
        uint8_t g = ((pixel >> 5) & 0x3F) * 255 / 63;
        uint8_t b = (pixel & 0x1F) * 255 / 31;
        // Convert to grayscale
        uint8_t gray = (0.299 * r) + (0.587 * g) + (0.114 * b);
      } else {
        Serial.println("Error: Unexpected end of file.");
        errorLine--;
        Serial.println(errorLine);
        break;
      }
    }
  }
```

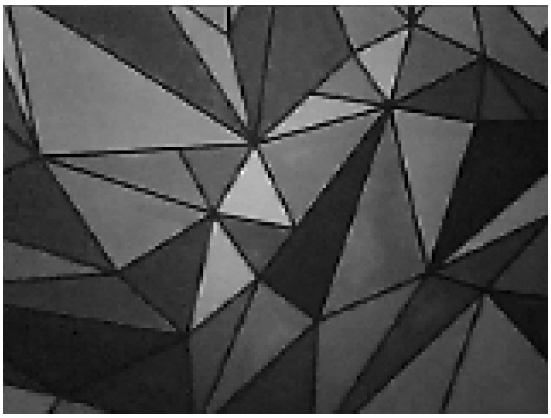
With the above code, it is possible to convert an RGB565 image into an 8-bit grayscale. Examples of this can be seen in figure 5.8.

(a) QQVGA image of outside pavement.

(b) Figure 5.8a grayscale converted.



(c) QQVGA image of a colorful background.



(d) Figure 5.8c grayscale converted.

Figure 5.8: Grayscale conversion examples using the function shown in 5.3.4.

Test C

With the images grayscale-converted, representing them in a 2D array becomes much easier. This is because each pixel is now represented by an 8-bit value where 0 is black, and 255 is white, instead of a mix of 3 values, representing red, green, and blue. To make human interpretation of the 2D array easier, the test-image shown in figure 5.9 is used. Furthermore, it is converted to a 256-bit bitmap, to ensure the image is readable row by row and column by column, just like an RGB565 image. By using a known image with clear distinctions in color, the 2D array result becomes easier to interpret.

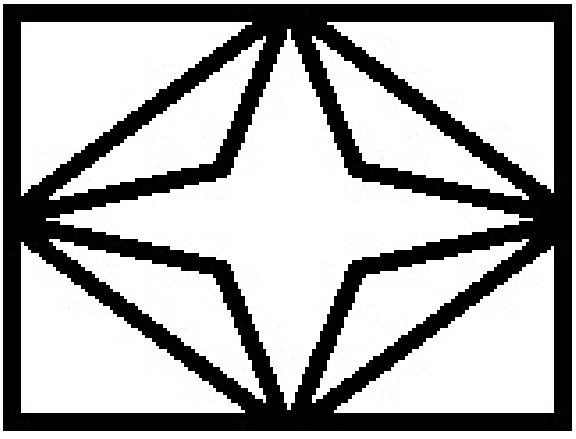


Figure 5.9: The test image is constructed as a bitmap image to test different functions when image processing, using the ESPCAM.

In figure 5.10 the initial 2D representation of the array is shown. As can be seen, the representation is less than ideal, as the first quarter of the image is placed at the end. Moreover, the last 8 rows seem to be missing. After reading up on image data formatting, it became apparent that the fault could be found in the header within the image, telling an operating system how to read the image. The header of a 256-bit

bitmap is 1078 bytes, displacing the first 8 rows of data before actual image values could be read. In figure 5.11 the correct representation can be seen. When observing the images, it should be noted that the image is only widened because the 2D arrays' data is displayed as values rather than pixels. In sections B.36 page 140 and 141 enlarged versions of the images can be seen.

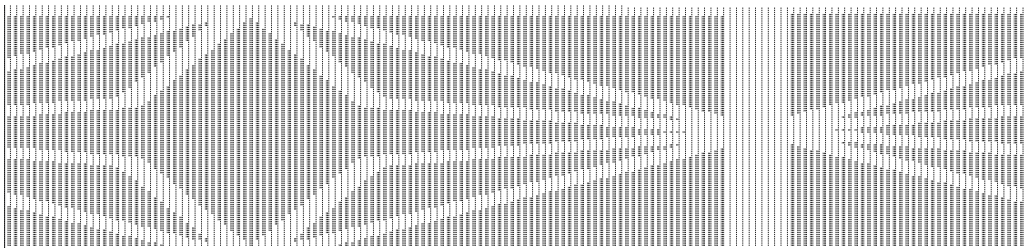


Figure 5.10: The test image converted into a 2D array. Notice that the first 1/4 of the image is placed at the end.

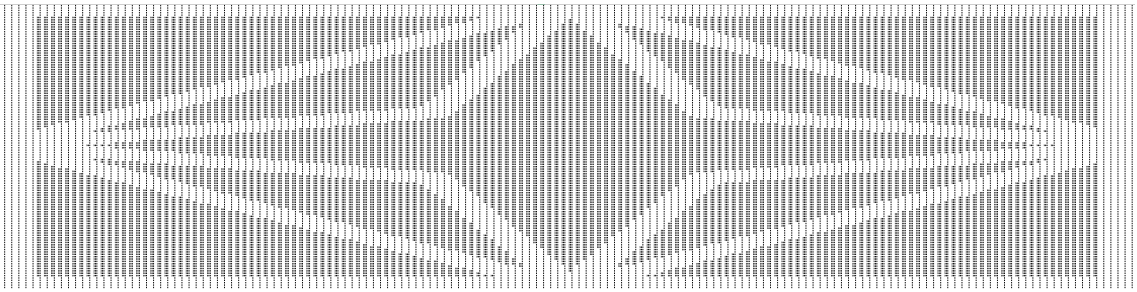
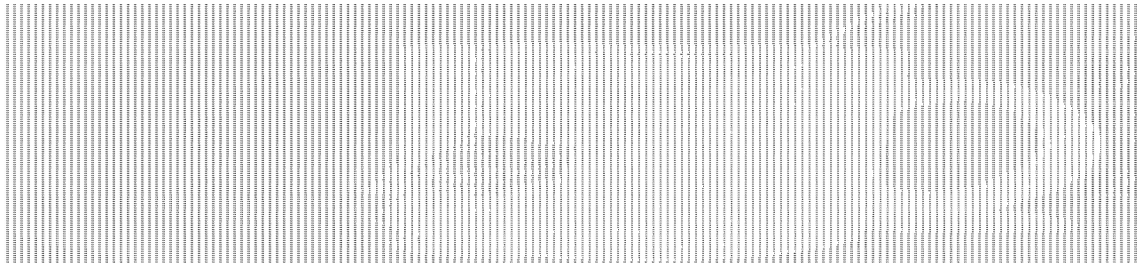


Figure 5.11: The test image converted into a 2D array. This time with the correct interpretation.

An example using an actual RGB565 image rather than an image was made as well. When reading a raw RGB565 image, there is no header, but trailing metadata instead. This only makes processing the image easier. Using the function in 5.3.4, the image is first grayscale converted, before it is plotted. In figure 5.12 an image of a tea mug grayscale converted by the proprietary grayscale function, along with a grayscaled image using the function from 5.3.4, and the corresponding 2D array is shown.

(a) Grayscale image of a tea mug, converted using the proprietary grayscale function.

(b) The 2D array put into excel, with equal sized cells. Furthermore, the cells have been grayscaled by their value.



(c) 2D array representation of the tea mug.

Figure 5.12: Grayscale conversion examples using the function shown in 5.3.4 with the 2D array representation shown as well.

Test D

With the image grayscaled, the next step is adding a Gaussian blur to remove the first layer of noise present, which is extraordinarily relevant for images in QQVGA resolution. The 3x3 gaussian kernel introduced in equation 4.58 on page 4.58 is used on the same tea-mug image as used so far, yielding an output shown in figure 5.13.

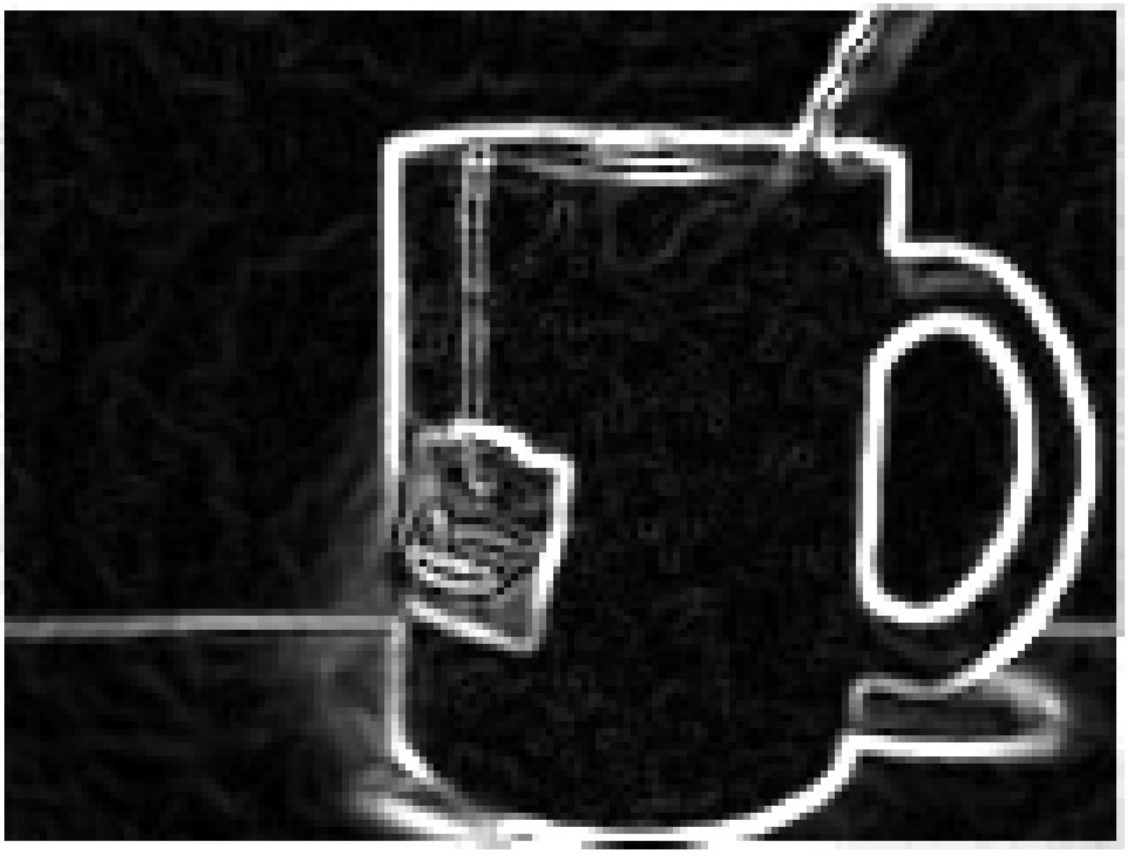
Figure 5.13: Gaussian blurred image of tea mug.

Test E

To begin edge extraction, the vertical and horizontal sobel operator, which can be seen in equation 4.85 on page 4.85. The result of doing so can be seen in figure 5.14a for the vertical, 5.14b for the horizontal, and 5.14c for the summed operation.

(a) Gaussian blurred image of a tea mug subjected to vertical Sobel operator.

(b) Gaussian blurred image of a tea mug subjected to horizontal Sobel operator.



(c) Gaussian blurred image of a tea mug subjected to vertical and horizontal Sobel operator summed into one image.

Figure 5.14: Sobel operations on Gaussian blurred image of a tea mug.

Test F

With most edges found using Sobel operators, the only step left before canny edge processing has been concluded is edge extraction using hysteresis/double thresholding. The thresholds used in this case are 150 and 50, where pixels with a value above 150 are converted to white pixels, pixels with values above 50 having an adjacent pixel with values above 150 are converted to white pixels, and all pixels below 150 with no adjacent pixels above 150 are converted to black pixels. In figure 5.15 the final result can be seen.

Test G

To test the execution time, the entire program has been set to run, from image extraction

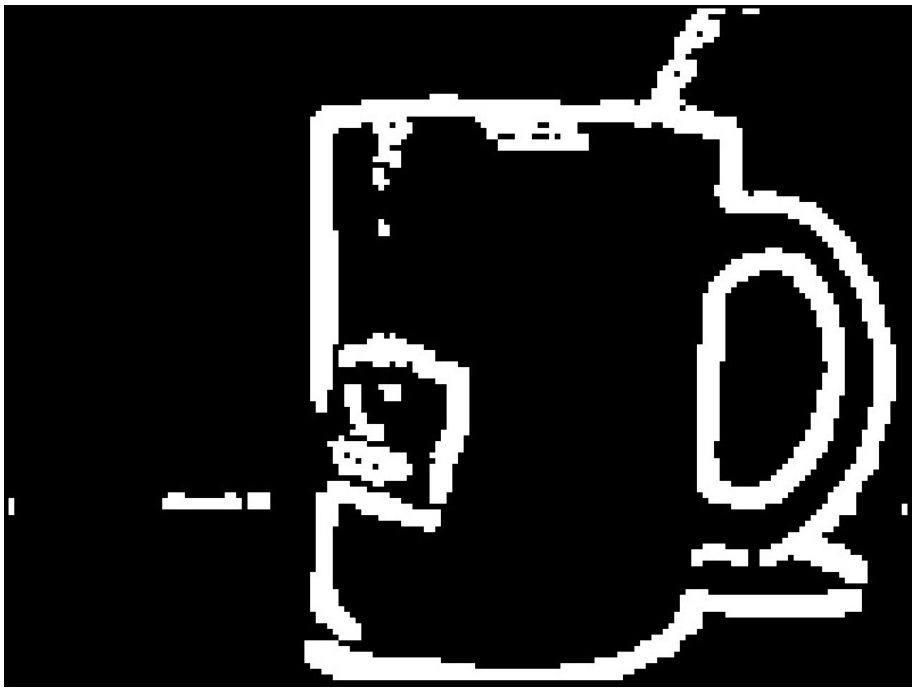


Figure 5.15: Double thresholded image of tea mug, with upper threshold of 150 and lower threshold of 50.

Iterations	Average Time (ms)
10	394.5
10	435.7
128	324.1
256	330.0
256	330.3

Table 5.1: Average execution time of the entire program at a clockrate of 20 MHz.

till double thresholding has been done. Several iterations have been done with the following execution times measured:

While the tests mostly went as expected, an issue began appearing. At an increased number of iterations, the ESPCAM began encountering an error called "cam_hal: EV-VSYNC-OVF". While this rarely occurred at 256 iterations, it was constant for a higher number of iterations, such as 512. This error occurs for some camera modules, as the camera is extraordinarily cheap and the hardware isn't top-of-the-line. The main cause of the error is connected to the data transfer rate between the camera and the ESP32 MCU, as data cannot be transferred fast enough between the two, resulting in an overflow error. The remedies for fixing it are either: decreasing clockrate, decreasing resolution, new hardware or simply adding delays. One way to add a delay would be to have the camera module compute the control signals as well, as this will increase execution time in total before a new image is taken.

5.3.5 Summary

The ESPCAM has proven capable of taking an image and processing it with a canny edge algorithm. Furthermore, it can do so at approximately 3 FPS. An issue has occurred regarding the clockrate when more than 256 images are taken in a row, leading to overflow errors and system reboots. The apparent solution is slowing the entire pro-

gram, task indexing, or acquiring new hardware, [46]. As new hardware is not an option for this specific project, the program's execution time will be throttled by making the ESPCAM responsible for computing control signals as well, which will be explained in section 5.5.

As a proof of concept, the camera has also been tested on pavement, yielding rather satisfactory results, as shown in figure 5.16 and 5.17

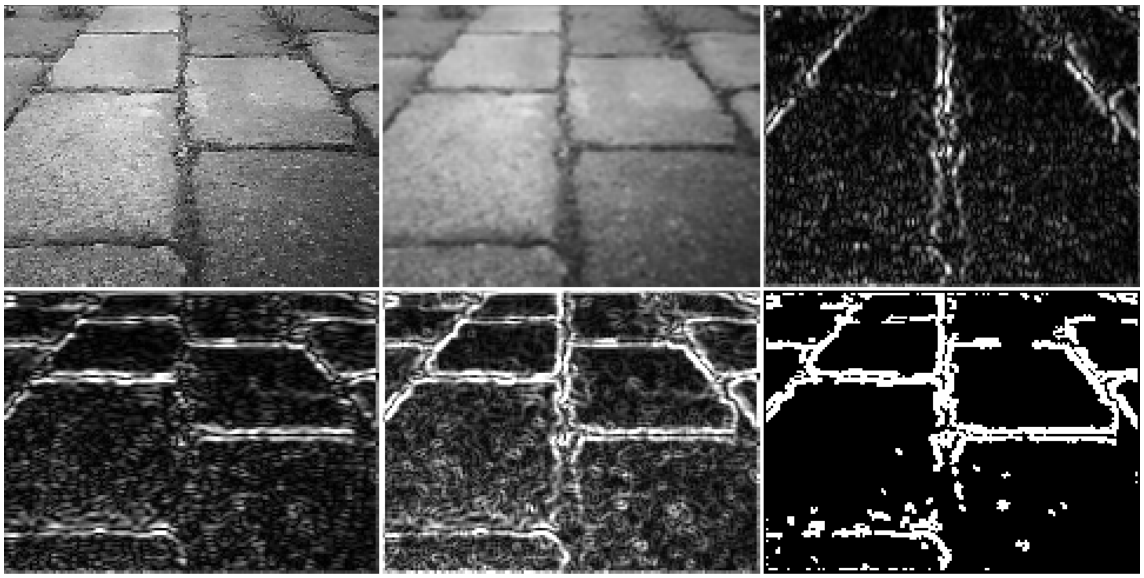


Figure 5.16: All steps used on an image of pavement.

Figure 5.17: Enlarged photo of pavement image with double thresholding set at 190 and 100.

5.4 Wireless Communication Module

The wireless communication to be established between the ESP32 and ESPCAM will be made using the ESPNOW protocol introduced in section 4.7.

5.4.1 Specification

- Low latency communication
- Data transfer of at least 4 bytes per packet, without frame
- ACK/NACK protocol
- Compatability with WiFi for future iterations

5.4.2 Design

The main purpose of introducing wireless communication to the system is to transfer control signals from the ESPCAM module to the ESP32. Moreover, the wireless communication is designated to synchronize the two modules. The specification can be made to:

- Establish a connection between ESPCAM and ESP32.
- Send clock synchronization timestamps from ESP32 to ESPCAM.
- Send packets containing control signals from the ESPCAM to the ESP32.
- Send callback signals between both boards, ensuring a stable connection.

To relieve the ESPCAM for as many processes as possible the ESP32 will be set as master, sending clock synchronization and establishing information to the ESPCAM, and the ESPCAM will only send status updates and control signals back to the ESP32.

5.4.3 Implementation

Implementation is easy, as it consists of adding a library to the main code segments, and thereafter adding wireless protocol to each.

5.4.4 Test

The following tests will be performed, to determine the capabilities of ESPNOW between the two boards:

- Execution time of connection instantiation
- Clock synchronization
- Data integrity and throughput capabilities
- Data packet loss
- Connection loss protocol

Test Setup

To perform tests, the setup shown in figure 5.18 is used. The necessary parts are:

- 1x ESPCAM
- 1x ESP32
- 1x PC for data logging and interpretation
- 2x USB-MicroUSB cables

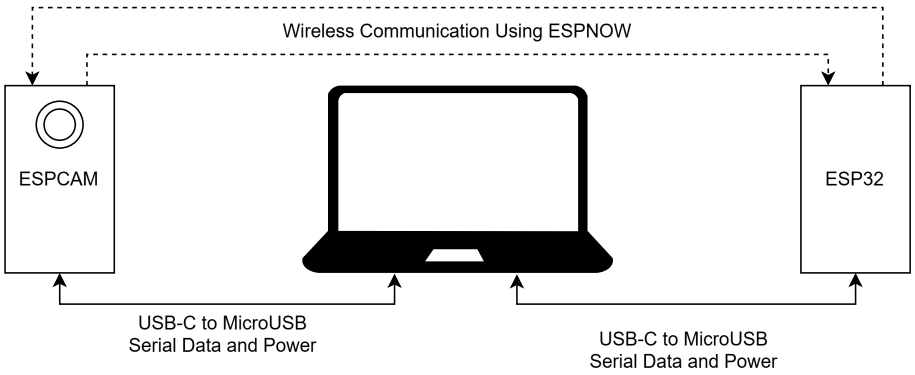


Figure 5.18: Test setup for wireless communication tests.

Actual Testing

To perform the tests, the following procedure should be used:

1. Connect the ESP32 and ESPCAM to a computer
2. Load the code for each test respectively:
   (A) Code, link is also available in appendix A.3.14.
   (B) Code and code , link is also available in appendix A.3.10 and A.3.11.
   (C) Code, link is also available in appendix A.3.13.
   (D) Code, link is also available in appendix A.3.15.
   (E) Any code can be used, as the slave device will be powered off simulating a total connection loss.
3. Execute the code
4. Interpret the results shown in the serial monitor
5. Save results and power off

Test Results

Test A

For the test, the ESP32 was set as the master device, running the code entirely before the ESPCAM was powered on. Upon powering the ESPCAM on, it takes ≈75ms to make a connection. Consecutive packets being sent take ≈0.2ms and the

Test B

To ensure actual synchronization with the master clock, the ESPCAM was powered on at first and set idle, before the ESP32 was turned on, and broadcasted a synchronization signal. Within 1 packet transmission, the boards were synchronized, which can be seen from the serial output in figure 5.19. The average difference of 60ms between the local ESP32 time and corrected time on the ESPCAM is found in the serial monitor protocol. The first 50 ms are caused by the different timestamps in each serial monitor and the remaining 10ms are found by looking at "serial.print"'s execution time of approximately 1.6ms per line (bits/BAUDrate * 1000) and latency of up to 3ms for each transmission. Nevertheless, in the worst case, 20ms difference is manageable to work around.

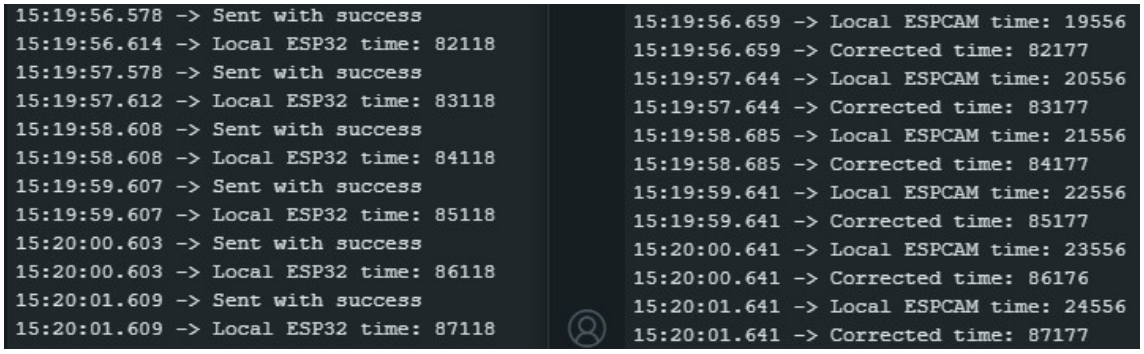


Figure 5.19: Serial monitor snap of clock synchronization between the ESP32 and ESPCAM. As can be seen, the two board clocks are now within 10-20ms of each other.

Test C

During a testing window of 10 minutes, no data was lost or corrupted. Furthermore, upon the next packet being sent to a disabled system, the callback received alarmed the sender. If a packet of the wrong length was sent, another alert was initiated, ensuring proper data integrity.

Test D

As can be read from the code, a deliberate packet loss was made to check whether or not the callback function operates as expected. When the packet was skipped/a null was sent, the correct callback requesting a resend was made, from the ESP32.

Test E

As with the lost packet, resend was prompted upon the first missing packet. After 10 missing packets, the ESP32 initiates an emergency stop, as the connection has been lost or at least lost for long enough for a restart to be prompted. The test was done by powering off the ESPCAM after 10 correct packets had been sent.

5.4.5 Summary

In general, the ESPNOW protocol lives up to all demands set for the prototypical De-Weeder. Furthermore, the protocol delivers consistent results and high data integrity with low power demands, making it ideal for further prototyping.

5.5 Computing Module

The computing modules' goal is to translate the canny-edged images yielded by the camera module into control signals for the tracked vehicle. This will be done by reading pixel values in specific pixel coordinates within the image, and then instantiating cases for high (255) or low (0) values at these coordinates. From these cases, the belt speeds are determined.

5.5.1 Specification

The computing module should be able to:

- Determine the direction of the vehicle based on image input
- Respond to the direction of the vehicle and reduce the angle difference to 0 degrees
- Locate a line in the image and place the line vertically in the middle of the frame
- If more lines are present in the image, locate the nearest

5.5.2 Design

To distinguish between lines in the pavement, noise, etc., a grid of point within each image has been made. The grid's purpose is to reveal whether or not edges has been found in their spots. In figure 5.20 an example of the grid can be seen. It should be noted that each spot is 4x4 pixels in size, with 4 pixels between them. Furthermore, the line down the middle represents a perfect straight.

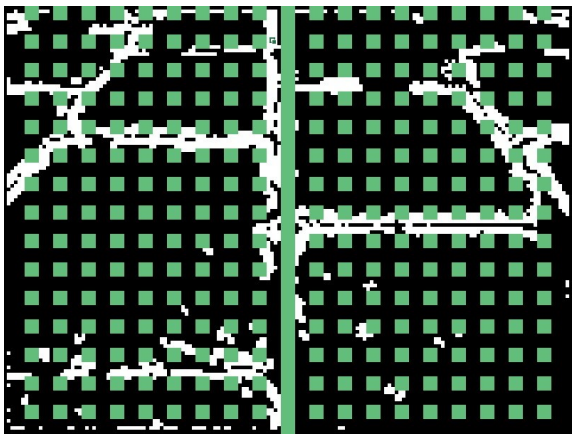


Figure 5.20: Grid example used on the canny-edged pavement image from section 5.3.5.

By using a grid as shown in 5.20 it is possible to determine where lines are present if any are present. Furthermore, it is easy to determine their orientation. The flowchart describing how data should be interpreted is shown in figure 5.21. When the computing algorithm receives a new image, it will first determine if the centerline is above or below 61200 (the combined value of 120x2x255), as this value represents half of the centerline being occupied by an edge. If the centerline is less than 61200, it is safe to assume that the line to follow, is no longer centered, leading to each (green) spot in the image being calculated to either signify a 0 or 1 value, if the value is above or below 2040 (4x2x255). If a spot has a value higher than or equal to 2040, it is set as 1 in the line matrix. The line matrix is a 19x15 matrix, wherein index (9,15) is set as origo, and column 9 in its

entirety is the centerline. When the entire image has been analyzed for edge values, the line matrix should have a set of values, which can be interpreted into a direction. In figure 5.22 the flowchart determines which way the tracked vehicle should turn if the centerline is less than 61200.

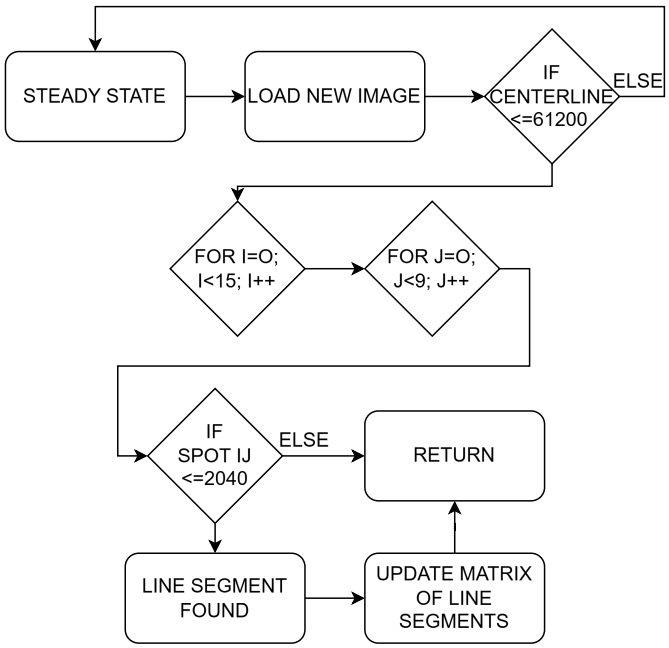


Figure 5.21: Flowchart of how the computing module translates high-value spots into a smaller matrix for further processing.

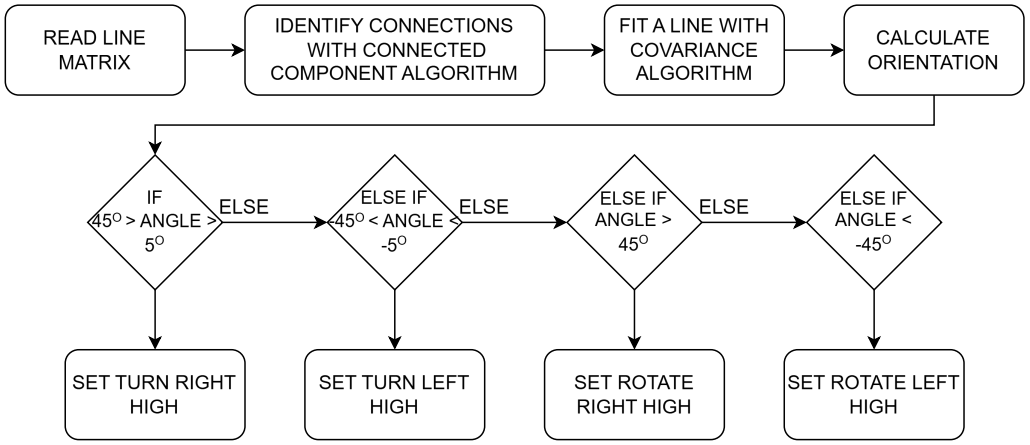


Figure 5.22: Flowchart of how the computing module translates the line matrix into a control signal for the tracked vehicle.

The connected component algorithm is meant to determine whether or not index "i1,j1" is connected to index "i2,j2", and so forth. The algorithm works by iterating a 2x2 raster (shown in red, green, and blue in figure 5.23) through the matrix row by row, labeling any high values (1 in this case) as 1 in another matrix. When the algorithm encounters a high value not connected to anything else, it labels it as 2, 3, 4, and so forth, until the entire image has been iterated. When two sets of high values happen to meet each other further down the picture, the higher label will be renamed to fit the lower value. This can be seen in figure 5.23 step by step. When the entire image has been iterated, leaving only segmented components, either the largest component or the

component matching earlier findings is set as the desired line to follow.

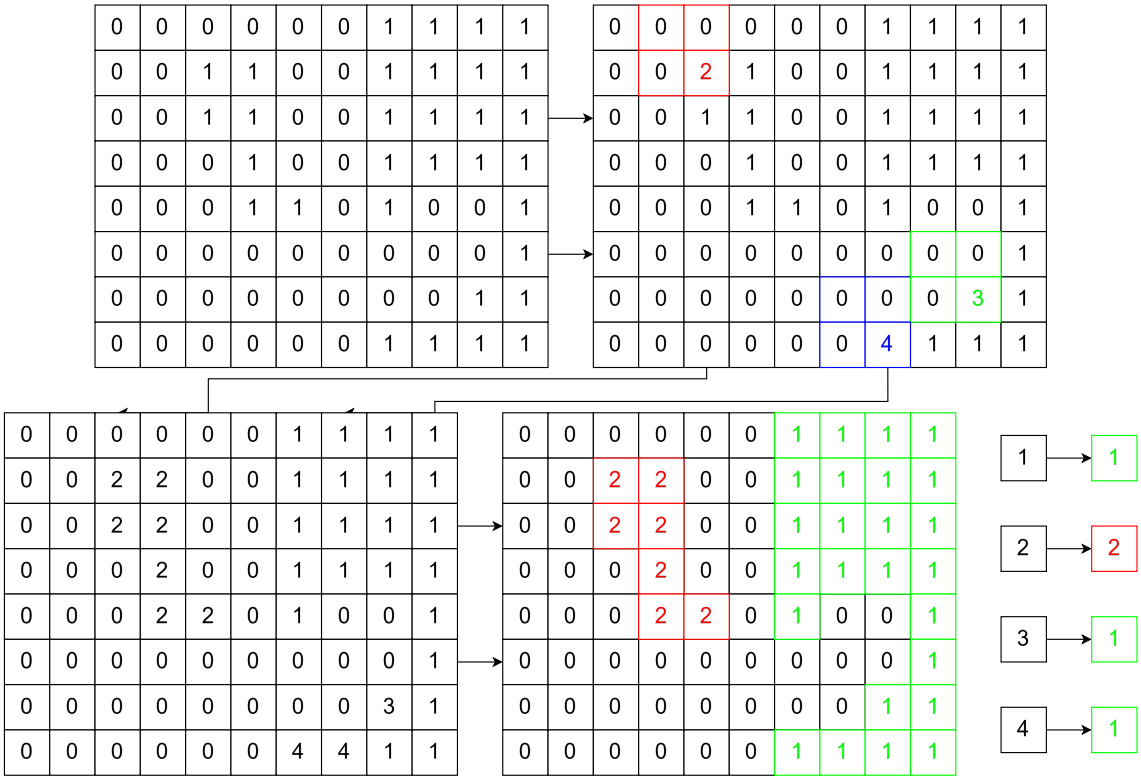


Figure 5.23: Connected component algorithm. Notice that at first 4 components are found, but reduced to 2, as the last two happen to be connected to the first.

After the connected component algorithm has done its work, the covariance algorithm is executed. The covariance algorithm is best explained using an example. In the matrix shown in equation 5.1, a 19x15 matrix can be seen, matching the spots shown in figure 5.20, with column 9 being the centerline. The high values match a possible line that the tracked vehicle could follow.

$$\begin{pmatrix}
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0
\end{pmatrix} \tag{5.1}$$

First, all points are extracted where the value is 1, treating column 9 as the origin. The transformed coordinates are:

$$(0,-5),(1,-5),(2,-5),(3,-4),(4,-4),(5,-4),(6,-3),(7,-3),$$
$$(8,-3),(9,-2),(10,-2),(11,-2),(12,-1),(13,-1),(14,-1)$$

The centroid (x_c, y_c) is given by:

$$x_c = \frac{\sum x_i}{n}, \quad y_c = \frac{\sum y_i}{n}$$

Where $n = 15$ is the total number of points.

$$\sum x_i = 0+1+2+3+4+5+6+7+8+9+10+11+12+13+14 = 105$$

$$\sum y_i = -5-5-5-4-4-4-3-3-3-2-2-2-1-1-1 = -50$$

$$x_c = \frac{105}{15} = 7, \quad y_c = \frac{-50}{15} \approx -3.33$$

The variance and covariance are calculated as:

$$\text{Variance}_x = \frac{\sum (x_i - x_c)^2}{n}, \quad \text{Variance}_y = \frac{\sum (y_i - y_c)^2}{n} \tag{5.2}$$

$$\text{Covariance} = \frac{\sum (x_i - x_c)(y_i - y_c)}{n}. \tag{5.3}$$

For each point, $(x_i - x_c)$, $(y_i - y_c)$ is computed together with the respective squared terms. The total summations are:

$$\sum (x_i - x_c)^2 = 280, \quad \sum (y_i - y_c)^2 = 30.78, \quad \sum (x_i - x_c)(y_i - y_c) = 100.68. \tag{5.4}$$

The slope m is given by:

$$m = \frac{\text{Covariance}}{\text{Variance}_x} = \frac{100.68}{280} \approx 0.36 \tag{5.5}$$

The angle of orientation θ is:

$$\theta = \arctan(m) = \arctan(0.36) \approx 19.8^\circ \quad (0.345\,\text{radians}). \tag{5.6}$$

The line has an orientation of approximately 19.8° relative to column 9. With the angle found, the if loops shown in figure 5.22 determine how the vehicle should operate.

Unfortunately, due to time constraints, control logic for making 90-degree turns and traveling to the next line of pavement in the opposite direction will not be made for this prototype, rendering the system a simple line-follower robot.

5.5.3 Implementation

As the entirety of the computing module can be implemented as a function, the only thing to do is import it into the main code. The computing control signals code as a singular function can be found here, the link is also available in appendix A.3.16.

5.5.4 Test

Testing the computing module will be done by feeding it preset matrices with known expected outputs. The known matrices can be seen as black/white images in figure 5.24.

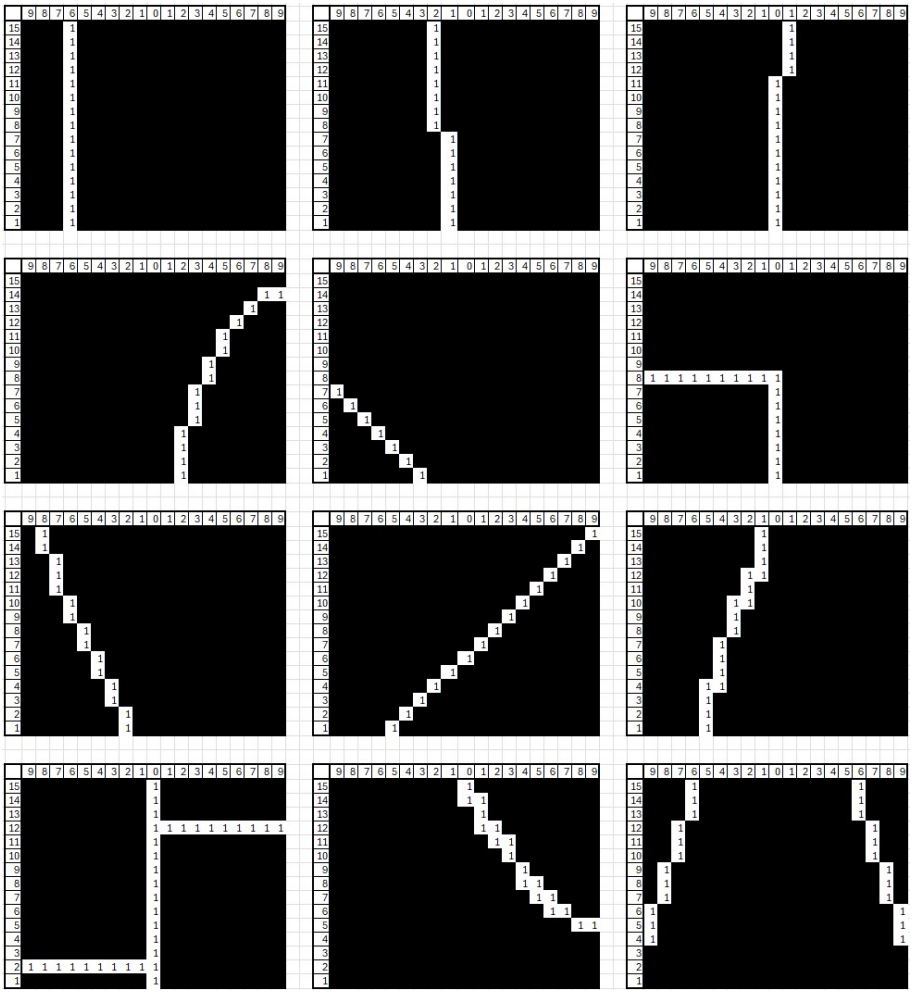


Figure 5.24: Preset matrices to be inserted into the compute control signal function.

The test is conducted by inserting each matrix into the compute control signal function and monitoring the output in the serial monitor to see if the output matches the expected output. Finally, execution times are monitored as well for FreeRTOS scheduling.

Test Setup

As the function is supposed to run on the ESPCAM, the test setup shown in figure 5.25 is used again. The only needed elements to perform the test are:

- ESPCAM board
- USB-C cable
- PC for output interpretation and code upload

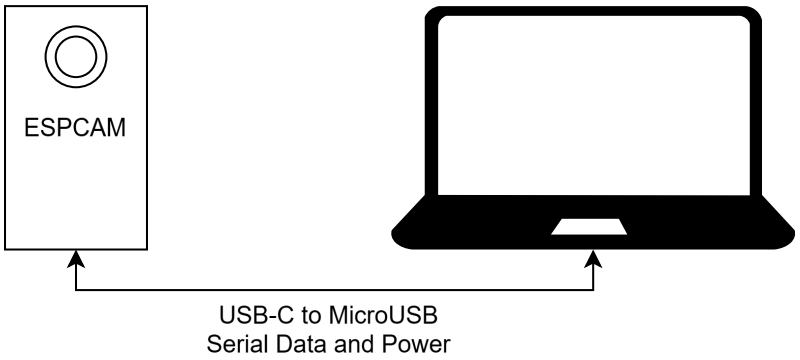


Figure 5.25: Overview of the test setup.

Test Results

Test results have been interpreted through the serial monitor, reading the output for each matrice as one of the options shown in figures 5.21 and 5.22. In table 5.2 the results can be seen.

Matrix	Expected Output	Actual Output	Execution Time
1	Turn left	Turn left[1]	0.317ms
2	Turn left	Turn left	0.267ms
3	Continue straight	Continue straight	10.397ms
4	Turn right	Turn right	0.240ms
5	Rotate left	Rotate left	0.245ms
6	Turn left	Turn left	0.258ms
7	Turn left	Turn left	0.254ms
8	Rotate right	Rotate right	0.238ms
9	Continue straight	Continue straight	0.239ms
10	Continue straight	Continue straight	10.416ms
11	Continue straight	Continue straight	0.241ms
12	Halt, as no line is larger than the other	Continue straight	10.594ms

Table 5.2: Expected and actual output for each matrice fed to the compute control signal algorithm. Do note the extraordinarily long execution times for 3, 10, 11, and 12.

5.5.5 Summary

Control signals can be derived from downsampled matrices and output the expected values in almost every case, except for when two equally large components/lines are present in the frame. Unfortunately, for some matrices, the execution time is ≈ 51 times larger, and the cause for this has yet to be found, as each matrice is subjected to the exact same algorithm. Nevertheless, the execution time is sufficiently low for the algorithm to be viable in the system. Furthermore, as the algorithm yields the wanted behavior robustly, the module as a whole should be implementable as is, into the main program.

[1] Best explained by looking at lines 405-413 in the code.

5.6 Physical Platform Module

GRUNDIG BESKRIVELSE AF HVORFOR PÅ LARVEFØDDER

5.6.1 Specification

-

5.6.2 Design

5.6.3 Implementation

5.6.4 Test

Test Setup

Actual Testing

Test Results

5.6.5 Summary

6 Integration

Blokdiagram over hele systemet, som er virkelig detaljeret

6.1 Physical Platform and Charge Point

6.2 MCU, Drive Motors, Power Source, and Power Distribution

6.3 Sensors

6.4 Camera, Computing, and Wireless Communication

7 Acceptance test

8 Discussion

9 Conclusion

Bibliography

[1] "Ukrudtsbekaempelse på belaegninger". In: ().

[2] *Biodiversitet - Miljøstyrelsen*. URL: https://mst.dk/borger/natur-og-fritid/natur-og-biodiversitet/biodiversitet.

[3] *Du må ikke længere bruge Roundup midler med glyphosat på veje, fortove, gårdspladser, terrasser, grusstier og lign. - Miljøstyrelsen*. URL: https://mst.dk/nyheder/2024/februar/du-maa-ikke-laengere-bruge-roundup-midler-med-glyphosat-paa-vej e-fortove-gaardspladser-terrasser-grusstier-og-lign.

[4] *Forbud mod at bruge Roundup i indkørsler og på terrasser*. URL: https://www.bolius.dk/slut-med-at-bruge-roundup-i-indkoersler-og-paa-terrasser-97376.

[5] *Sådan bekæmper du IKKE ukrudt | Brug fx ikke salt eller eller eddikesyre*. URL: https://www.bolius.dk/saadan-bekaemper-du-ikke-ukrudt-2959.

[6] *Populært ukrudtsmiddel gjort ulovligt: Her er alternativerne | Havehobby.dk*. URL: https://via.ritzau.dk/pressemeddelelse/13577322/populaert-ukrudtsmiddel-gjort-ulovligt-her-er-alternativerne?publisherId=13559506.

[7] *Naturlig ukrudstbekæmpelse: Derfor bør du undgå salt mod ukrudt | idenyt*. URL: https://idenyt.dk/haven/ukrudt/fa-styr-pa-ukrudtet-inden-det-far-fat/.

[8] *Må du bruge eddike eller eddikesyre til at bekæmpe ukrudt?* URL: https://www.bolius.dk/maa-du-bruge-eddikesyre-som-ukrudtsmiddel-35665.

[9] *Græsplæner - Miljøstyrelsen*. URL: https://mst.dk/erhverv/sikker-kemi/pesticider/anvendelse-af-pesticider/graesplaener.

[10] *Fugebørste med skraber - Freund | BAUHAUS*. URL: https://www.bauhaus.dk/fugeborste-med-skraber-freund.

[11] *News & Media — Carbon Robotics*. URL: https://carbonrobotics.com/news-media.

[12] *Technology — Carbon Robotics*. URL: https://carbonrobotics.com/laserweeding-technology.

[13] *WeedBot Products*. URL: https://weedbot.eu/weedbot-products/.

[14] *We Laser Project: Eco-Innovative weeding with laser*. URL: https://welaser-project.eu/.

[15] Christian Andreasen et al. "Side-effects of laser weeding: quantifying off-target risks to earthworms (Enchytraeids) and insects (Tenebrio molitor and Adalia bipunctata)". In: *Frontiers in Agronomy* 5 (Nov. 2023), p. 1198840. ISSN: 26733218. DOI: 10.3389/FAGRO.2023.1198840/BIBTEX. URL: https://welaser-project.eu/.

[16] Christian Andreasen, Karsten Scholle, and Mahin Saberi. "Laser Weeding With Small Autonomous Vehicles: Friends or Foes?" In: *Frontiers in Agronomy* 4 (Mar. 2022). ISSN: 26733218. DOI: 10.3389/FAGRO.2022.841086.

[17] Christian Andreasen et al. "Laser weed seed control: challenges and opportunities". In: *Frontiers in Agronomy* 6 (2024). ISSN: 26733218. DOI: 10.3389/FAGRO.2024.1342372/FULL.

[18] Christian Andreasen, Eleni Vlassi, and Najmeh Salehan. "Laser weeding of common weed species". In: *Frontiers in Plant Science* 15 (May 2024), p. 1375164. ISSN: 1664462X. DOI: 10.3389/FPLS.2024.1375164/BIBTEX. URL: https://welaser-project.eu/.

[19] *LUBA AWD 5000 : Robotplæneklipper uden tråd i perimeteren*. URL: https://mammotion.com/products/luba-awd-5000-perimeter-wire-free-robot-lawn-mower.

[20] *iRobot® Roomba® Combo 10 Max*. URL: https://www.irobot.dk/products/combo-10-max.

[21] *Dr Robot Inc.: WiFi 802.11 robot, Network-based Robot, robotic, robot kit, humanoid robot, OEM solution*. URL: http://jaguar.drrobot.com/specification_lite.asp.

[22] *Geared Motor for Modelboat - Max Gear 50:1 - KRICK*. URL: https://www.newcapmaquettes.com/Geared-Motor-50-1-KRICK.html.

[23] *iRobot Roomba robotstøvsuger | Opdateret 2024-sortiment*. URL: https://www.irobot.dk/collections/roomba-robotstoevsugere.

[24] Amitava. Chatterjee, Anjan. Rakshit, and N. NirmalSingh. *Vision Based Autonomous Robot Navigation : Algorithms and Implementations*. eng. Ed. by Amitava. Chatterjee et al. Elektronisk udgave. Studies in Computational Intelligence, 455. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 9783642339653.

[25] *control theory - Relative Gain Array of a singular matrix - Mathematics Stack Exchange*. URL: https://math.stackexchange.com/questions/1297659/relative-gain-array-of-a-singular-matrix.

[26] *(PDF) On the Relative Gain Array (RGA) with Singular and Rectangular Matrices*. URL: https://www.researchgate.net/publication/325413752_On_the_Relative_Gain_Array_RGA_with_Singular_and_Rectangular_Matrices.

[27] Jeffrey Uhlmann. "On the Relative Gain Array (RGA) with singular and rectangular matrices". In: *Applied Mathematics Letters* 93 (July 2019), pp. 52–57. ISSN: 0893-9659. DOI: 10.1016/J.AML.2019.01.031.

[28] *[1806.01776] A Generalized Matrix Inverse with Applications to Robotic Systems*. URL: https://arxiv.org/abs/1806.01776.

[29] *How Blurs & Filters Work - Computerphile - YouTube*. URL: https://www.youtube.com/watch?v=C_zFhWdM4ic.

[30] *Finding the Edges (Sobel Operator) - Computerphile - YouTube*. URL: https://www.youtube.com/watch?v=uihBwtPIBxM.

[31] *Canny Edge Detector - Computerphile - YouTube*. URL: https://www.youtube.com/watch?v=sRFM5IEqR2w.

[32] *python - Do I need to normalize image after blurring? (if the sum of gaussian filter is not 1) - Stack Overflow*. URL: https://stackoverflow.com/questions/73018712/do-i-need-to-normalize-image-after-blurring-if-the-sum-of-gaussian-filter-is-n.

[33] *Edge Detection in Image Processing: An Introduction*. URL: https://blog.roboflow.com/edge-detection/.

[34] *Canny Edge Detection Step by Step in Python — Computer Vision | by Sofiane Sahir | Towards Data Science*. URL: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123.

[35] *Hough Transform. A comprehensive guide to edge detection... | by Surya Teja Karri | Medium*. URL: https://medium.com/@st1739/hough-transform-287b2dac0c70.

[36] *Hough Transforms in Image Processing - Scaler Topics*. URL: https://www.scaler.com/topics/hough-transform-in-image-processing/.

[37] *Hough Transform - YouTube*. URL: https://www.youtube.com/watch?v=uJ_8byXUqGc.

[38] *Hough transform example in polar coordinates | Digital Image Processing | Image segmentation | - YouTube*. URL: https://www.youtube.com/watch?v=RoiY1_-b8pY.

[39] *Hough Transform | Boundary Detection - YouTube*. URL: https://www.youtube.com/watch?v=XRBc_xkZREg.

[40] *Hough transform in computer vision. - GeeksforGeeks*. URL: https://www.geeksforgeeks.org/hough-transform-in-computer-vision/.

[41] *Hough transform — skimage v0.6 docs*. URL: https://scikit-image.org/docs/0.6/auto_examples/plot_hough_transform.html.

[42] *Hough Transform. A comprehensive guide to edge detection... | by Surya Teja Karri | Medium*. URL: https://medium.com/@st1739/hough-transform-287b2dac0c70.

[43] *Pris og størrelse på byggegrunde i Danmark varierer voldsomt*. URL: https://www.boligsiden.dk/nyheder/boligpriser/pris-og-stoerrelse-paa-byggegrunde-i-danmark-varierer-voldsomt.

[44] *Battery Monitoring*. URL: https://www.12voltplanet.co.uk/battery-monitoring-explained.html.

[45] *Ultimate Guide to LiFePO4 Voltage Chart (3.2V, 12V, 24V, & 48V) - Jackery*. URL: https://www.jackery.com/blogs/knowledge/ultimate-guide-to-lifepo4-voltage-chart.

[46] *Camera Application - - — ESP-FAQ latest documentation*. URL: https://docs.espressif.com/projects/esp-faq/en/latest/application-solution/camera-application.html#what-triggers-cam-hal-ev-vsync-ovf-in-esp32-camera.

[47] *TMX1000 - Pris 1 stk. 4.599,- 10+ stk. på eget lager. - texas.dk*. URL: https://www.texas.dk/da/products/robotplaeneklipper/tmx1000/?partno=90070243.

Glossary

MVP Minimum Viable Product. 8, 150

A Appendix

A.1 Average Cleaning Time Driveway

The testing/timing has been done by cleaning the driveway every 3 weeks across the 2024 season with one of the mentioned methods and timing how long each method took. The driveway has had similar growth between each clean, however, do keep in mind that the driveway is being used and identical weed growth cannot be guaranteed, making these numbers a mere guideline rather than strict facts."Aftertreatment" refers to treatment such as refilling pavement with sand, sweeping dead shrubs away, etc.

Method	Area in m^2	Time Used	Aftertreatment	Time Per m^2
Manual Cleaning	106.8	166		1.554
Pressure Washing	106.8	204	351	5.197
Rotating Steel Brush	106.8	128	30	1.479
Chemicals	106.8	53		0.496
Sweeping	106.8	37		0.346
Weed Burning	106.8	117	30	1.376

Table A.1: Average cleaning time per square meter of herregårdssten in my own (Christians) driveway. All times are in minutes.

A.2 Introduction to TITO Systems

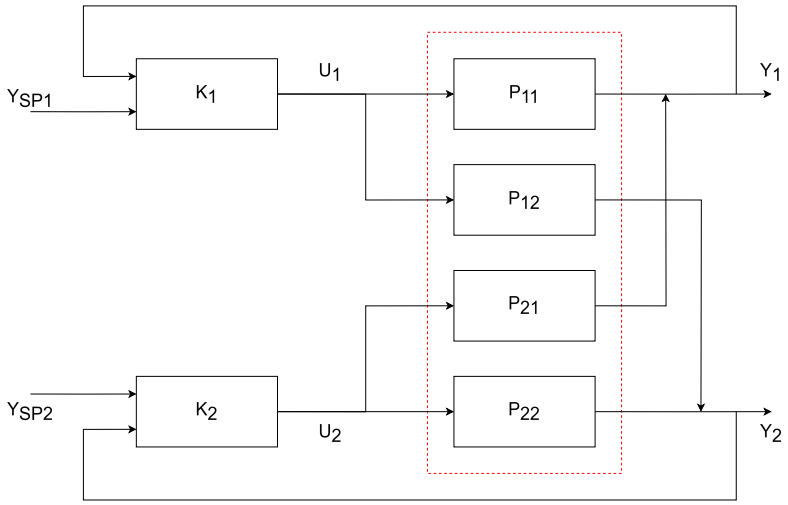


Figure A.1: General block diagram for a TITO system. The red square signifies the processes encompassed by the TITO system.

To make the TITO system more approachable it is introduced using variables, before it is adapted to fit the transfer functions describing the movement of the tracked vehicle, introduced in 4.1.2. For now, the open loop functions describing the relations between Ysp1-Y1 and Ysp2-Y2 are given in equations A.1 and A.2 respectively, and can also be described by the transfer matrix shown in equation A.3. In equations A.4 and A.5 the feedback equations are given in reverse to compute U(s).

$$Y_1 = P_{11} * U_1 + P_{12} * U_2 \tag{A.1}$$

$$Y_2 = P_{21} * U_1 + P_{22} * U_2 \tag{A.2}$$

$$P(s) = \begin{pmatrix} P_{11} * U_1 & P_{12} * U_2 \\ P_{21} * U_1 & P_{22} * U_2 \end{pmatrix} \tag{A.3}$$

$$U_1 = -K_1 * Y_1 \tag{A.4}$$

$$U_2 = -K_2 * Y_2 \tag{A.5}$$

Where all Y's, P's, and U's are functions of s. The next step is determining each system's interaction/influence on the other. An example could be made, with equations A.6 and A.7. Starting from the top, the influence of U2 is determined by equation A.10. Using A.10 in equation A.6 yields equation A.11, which can be used to describe the influence of U2/K2, without knowing either beforehand.

$$Y_{1Example} = \frac{1}{(s+1)^2} * U_1(s) + \frac{2}{(s+1)^2} * U_2(s) \tag{A.6}$$

$$Y_{2Example} = \frac{1}{(s+1)^2} * U_1(s) + \frac{1}{(s+1)^2} * U_2(s) \tag{A.7}$$

$$U_2(s) = -K_2 * Y_2(s) = Y_2(s) \Rightarrow \frac{-U_2(s)}{K_2} \tag{A.8}$$

Which can then be inserted into equation A.7, as seen in equation A.9

$$\frac{-U_2(s)}{K_2} = \frac{1}{(s+1)^2} * U_1(s) + \frac{1}{(s+1)^2} * U_2(s) \Rightarrow \frac{-U_2(s)}{K_2} - \frac{1}{(s+1)^2} * U_2(s) = \frac{1}{(s+1)^2} * U_1(s) \tag{A.9}$$

Which is simplified to:

$$\begin{aligned} U_2(s) * (\frac{-1}{K_2} - \frac{1}{(s+1)^2}) = \frac{1}{(s+1)^2} * U_1(s) \Rightarrow U_2(s) * (\frac{-(s+1)^2 - K_2}{K_2(s+1)^2}) = \frac{1}{(s+1)^2} * U_1 \\ \Downarrow \\ U_2(s) = \frac{-K_2}{(s+1)^2 + K_2} * U_1(s) \\ \Downarrow \\ U_2(s) = -\frac{K_2}{s^2 + 2s + K_2 + 1} * U_1(s) \end{aligned} \tag{A.10}$$

$$\begin{aligned} Y_{1Example}(s) = \frac{1}{(s+1)^2} * U_1(s) + \frac{2}{(s+1)^2} * (-\frac{K_2}{s^2 + s + K_2 + 1} * U_1(s)) \\ \Downarrow \\ Y_{1Example}(s) = \frac{s^2 + 2s + 1 - K_2}{(s+1)^2 * (s^2 + 2s + 1 + K_2)} * U_1(s) \end{aligned} \tag{A.11}$$

Which can be reduced to equation A.12 if s is set to 0:

$$Y_1(0) = \frac{1 - K_2}{1 + K_2} \tag{A.12}$$

By equation A.12 it becomes apparent that K2 will influence Y1. K2's influence will decrease as its own value increases, as equation A.12 will then yield smaller values until the gain becomes negative above K2=1. This implies that the system is coupled, as seen in figure A.1. Similar equations could be derived to find K1's influence on Y2. The question now is, which loop should determine the behavior of the other loop?

A scientific way would be to use a relative gain array (RGA). The RGA is a matrix, that can reveal the effect of pairing input and output variables in TITO systems. It is derived from the steady-state gain matrix of the system denoted "P". Mathematically, the RGA is defined as:

$$RGA = P(0) \circ (P(0)^{-1})^T \tag{A.13}$$

Where:
$P(0) = \begin{pmatrix} P_{11}(0) & P_{12}(0) \\ P_{21}(0) & P_{22}(0) \end{pmatrix}$ is the static gain of the system. $\circ$ is piecewise multiplication. $(P(0)^{-1})^T$ is the inverse transposed of $P(0)$.

The next steps are best explained using an example. A steady state matrix is given by $P(0) = \begin{pmatrix} 2 & 1 \\ 3 & 4 \end{pmatrix}$. Its inverse is given by equation A.14 and the transposed by A.15. Then

the piecewise multiplication is made in equation A.16. When done, a checksum would be that each row and column sums to 1, as that is an inherited property of relative gain arrays.

$$P(0)^{-1} = \frac{1}{2*4-1*3} * \begin{pmatrix} 4 & -1 \\ -3 & 2 \end{pmatrix} = \begin{pmatrix} 0.8 & -0.2 \\ -0.6 & 0.4 \end{pmatrix} \tag{A.14}$$

$$(P(0)^{-1})^T = \begin{pmatrix} 0.8 & -0.6 \\ -0.2 & 0.4 \end{pmatrix} \tag{A.15}$$

$$RGA = P(0) \circ (P(0)^{-1})^T = \begin{pmatrix} 2 & 1 \\ 3 & 4 \end{pmatrix} \circ \begin{pmatrix} 0.8 & -0.6 \\ -0.2 & 0.4 \end{pmatrix} = \begin{pmatrix} 1.6 & -0.6 \\ -0.6 & 1.6 \end{pmatrix} \tag{A.16}$$

Now that a final RGA has been achieved, it can be examined. To describe the correlation, the interaction index $R = \begin{pmatrix} \lambda & 1-\lambda \\ 1-\lambda & \lambda \end{pmatrix}$ is used. The idea of the interaction index is to analyze how big the impact between control loops is. A λ of 1 indicates no interaction is present, whereas $\lambda = 0$ also indicates no interaction, but good practice would be to interchange the loops. In the case of $\lambda < 0.5$ loops should be interchanged as well. In general, the closer to $\lambda = 1$ the result is, the better. If the gains are outside of an interval stretching $0.67 < \lambda < 1.5$, decoupling can improve controllability. To find λ, the RGA is used. Using the example RGA achieved in A.16, λ would be 1.6, as shown by equation A.17, wherein a lambda of 1.6 will result in a true interaction index, as it becomes equal to the one given by A.16. Going by the rules of thumb, a λ of 1.6 indicates that decoupling could benefit the example system.

$$\begin{pmatrix} 1.6 & -0.6 \\ -0.6 & 1.6 \end{pmatrix} == \begin{pmatrix} \lambda & 1-\lambda \\ 1-\lambda & \lambda \end{pmatrix} \tag{A.17}$$

Decoupling is meant to eliminate the effect of interaction between different control loops. The dynamic coupling factor is given in equation A.18. To actually begin decoupling the system, a new loop "identical" to the original one is introduced in figure A.2. To describe the decoupling mathematically, equations A.19 and A.20 is introduced. Using equation A.21 and A.22 also makes it possible to reduce the number of unknowns in the decoupling equations, as shown in A.23 and A.24.

$$Q = \frac{P_{21} * P_{12}}{P_{11} * P_{22}} \tag{A.18}$$

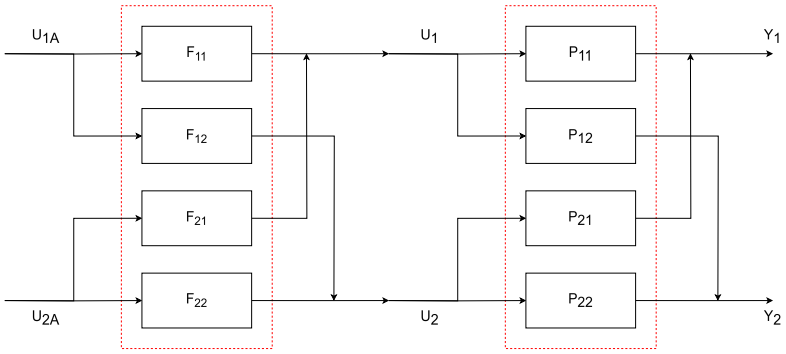


Figure A.2: A decoupled TITO system.

$$\frac{Y_1}{U_{1A}} = F_{12} * P_{22} + F_{11} * P_{12} = 0 \tag{A.19}$$

$$\frac{Y_2}{U_{1A}} = F_{21} * P_{11} + F_{22} * P_{21} = 0 \tag{A.20}$$

$$F_{12} = -F_{11} * \frac{P_{12}}{P_{22}} \tag{A.21}$$

$$F_{21} = -F_{22} * \frac{P_{21}}{P_{11}} \tag{A.22}$$

$$\frac{Y_1}{U_{1A}} = -F_{11} * \frac{P_{12}}{P_{22}} * P_{22} + F_{11} * P_{12} = 0 \tag{A.23}$$

$$\frac{Y_2}{U_{2A}} = -F_{22} * \frac{P_{21}}{P_{11}} * P_{11} + F_{22} * P_{21} = 0 \tag{A.24}$$

Equations A.25 and A.26 are used to continue the simplification. This simplification has left the equation system with 4 unknowns (F_{11}, F_{12}, F_{21}, *and* F_{22}) describable by 2 equations. Therefore, two of the unknowns can be chosen, and to simplify matters, they should be 1, as this makes it easier to find the remaining two, using equations A.21 and A.22 with F_{11} and F_{22} set to 1, yielding equations A.27 and A.28.

$$\frac{Y_1}{U_{1A}} = -F_{11} * \frac{P_{12}}{P_{22}} * P_{21} \Rightarrow (1 - \frac{P_{21} * P_{12}}{P_{11} * P_{22}}) * P_{11} * F_{11} \Rightarrow (1 - Q) * P_{11} * F_{11} \tag{A.25}$$

$$\frac{Y_2}{U_{2A}} = -F_{22} * \frac{P_{21}}{P_{11}} * P_{12} \Rightarrow (1 - \frac{P_{21} * P_{12}}{P_{11} * P_{22}}) * P_{22} * F_{22} \Rightarrow (1 - Q) * P_{22} * F_{22} \tag{A.26}$$

$$F_{12} = -1 * \frac{P_{12}}{P_{22}} \tag{A.27}$$

$$F_{21} = -1 * \frac{P_{21}}{P_{11}} \tag{A.28}$$

This leaves the final two expressions to be given in equation A.29 and A.30, enabling modeling using SISO methods rules from here on out, as shown in figure A.3.

$$Y_1 = U_{1A} * (1 - Q) * P_{11} \tag{A.29}$$

$$Y_2 = U_{2A} * (1 - Q) * P_{22} \tag{A.30}$$

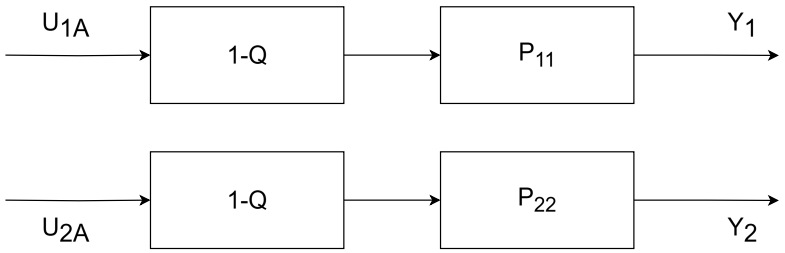


Figure A.3: Final interpretation of how a TITO system becomes a SISO system.

A.3 Links to Github Repositories

A.3.1 Complete Repository

https://github.com/Morl4d1m/ESD5Fall2024/tree/main

A.3.2 Movement

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/movement/movement.ino

A.3.3 Take Picture To SD

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/takePictureToSD/takePictureToSD.ino

A.3.4 ESP Read Image Grayscale Values

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/ESPReadImageGrayScaleValues/ESPReadImageGrayScaleValues.ino

A.3.5 Sobel Kernel On Grayscale Image

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/sobelKernelOnGrayscaleImage/sobelKernelOnGrayscaleImage.ino

A.3.6 Complete Canny Edge On ESPCAM

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/completeCannyEdgeOnESPCAM/completeCannyEdgeOnESPCAM.ino

A.3.7 DE-WEEDER Main

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/deWeederMain/deWeederMain.ino

A.3.8 Take Picture and Canny Edge It

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/takePictureAndCannyEdgeIt/takePictureAndCannyEdgeIt.ino

A.3.9 Dummy Testing FreeRTOS Routine

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/dummyTestingFreeRTOSRoutine/dummyTestingFreeRTOSRoutine.ino

A.3.10 Clock Synchronization ESP32

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/clockSynchronizationESPC32/clockSynchronizationESPC32.ino

A.3.11 Clock Synchronization ESPCAM

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/clockSynchronizationESPCAM/clockSynchronizationESPCAM.ino

A.3.12 Sensor Communication DE-WEEDER

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/sensorCommunicationDeWeeder/sensorCommunicationDeWeeder.ino

A.3.13 ESPNOW Data Integrity

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/ESPNOWDataIntegrity/ESPNOWDataIntegrity.ino

A.3.14 ESPNOW Execution Time Connection

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/ESPNOWExecutionTimeConnection/ESPNOWExecutionTimeConnection.ino

A.3.15 ESPNOW Loose a Packet

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/ESPNOWLooseAPacket/ESPNOWLooseAPacket.ino

A.3.16 Computing Control Signals

https://github.com/Morl4d1m/ESD5Fall2024/tree/main/Project/Kode/computeControlSignals/computeControlSignals.ino

B Test Reports

B.1 Test Report 1 - Forward Motion of Robot

The purpose of this test is to establish the correlation between forward movement and PWM input. The test will be performed for the robot as a whole, and each belt individually.

B.1.1 Test Setup

To perform the test the following is needed:

- 1x Robot
- 1x ZK-BM1 H-bridge
- 1x Oscilloscope, KeySight DSOX1102G
- 1x Triple Power Supply, HAMEG HM7042
- 4x 1m banana test leads (2 for GND and 2 for VCC)
- 1x 1m BNC coax cable
- 1x 0.1m BNC coax cable
- 1x BNC to banana plug adapter
- 1x Treadmill
- 1x Camera capable of 60fps
- 1x Camera-jig
- 1x Wire-jig
- 1x PC for data logging
- 1x Tether to robot to ensure straight travel

The test is performed on a Rodby treadmill capable of 0.1 km/h increments starting from 0.5 km/h. The test setup can be seen in figure B.1 and B.2.

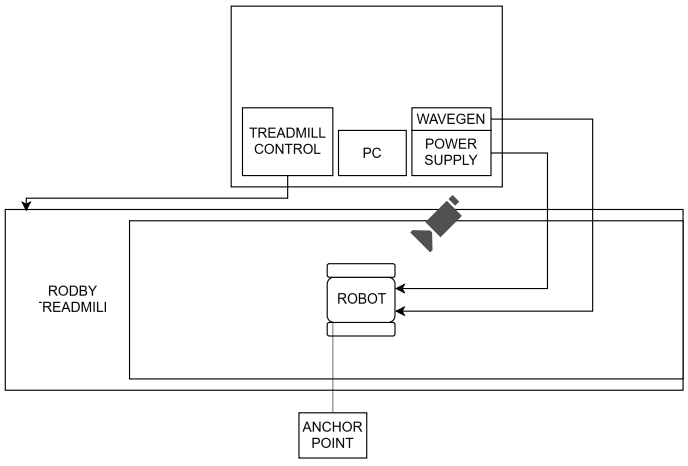


Figure B.1: Test setup for testing forward motion of the robot.

Figure B.2: Test setup for testing the forward motion of the robot in real life. The wooden board sticking from the table keeps cables out of the way.

To conduct the test, the following procedures must be followed:

1. Set the desired speed to test on the Rodby control panel, by first pressing "start", then "+" until the desired speed is met. The treadmill should not move until "start" is pressed again.
2. Set the voltage on the HAMEG power supply's third channel to 13.6V, and turn the current knob to the top. Furthermore, constant-current must be on, to ensure the power supply does not turn off when starting the motors.
3. Connect the power supply to the banana plugs on the vehicle.
4. Set the oscilloscope to wavegen, and output a square PWM signal with a peak-peak amplitude of 3.3V, frequency of 2kHz, offset of 0V, and PWM percentage at 1 to start.
5. Connect the oscilloscope wavegen with test probes on channels 2, 4, and ground on the ZK-BM1 H-bridge.
6. Connect the oscilloscope wavegen to channel 1 of the oscilloscope for visual interpretation.
7. Place the vehicle in the middle of the treadmill.
8. Turn channel 3 on at the power supply simultaneously with pressing "start" on the Rodby control panel.
9. Adjust PWM up or down to match the speed of the treadmill.
10. Estimate that the vehicle speed is matched when the vehicle does not change its position for a minute.
11. Log data such as: PWM, voltage, amperes, and speed.

B.1.2 Actual Testing

While testing both belts at once, it became apparent that the right belt didn't match the speed of the left belt, as the robot veered to the right a little at low speed, and as speed increased it veered increasingly. Therefore a tether to an anchor point was added to the setup (also shown in figure B.1) to keep the vehicle driving straight.

All PWM relations have been an estimate made by observing the vehicle at each speed set by the treadmill, observing if the vehicle advanced or fell back at varying PWM percentages. The rule of thumb used was that if the vehicle did not change its position on the treadmill more than 1cm back or forth over a minute, the chosen PWM matched the speed of the treadmill. While this may seem like a very unscientific method, it easily holds, as a PWM 1% below or above the matched, in most cases meant that the vehicle moved forwards or backward by a lot more than 1cm over a minute (up to 8cm in one instance).

It should also be noted that while testing the belts individually, the left belt could most often only match a given speed if the exact PWM was chosen, whereas the right belt could match the speed of the treadmill within +- 1% PWM at all speeds, and sometimes up to +- 2% PWM. This suggests that the left belt/motor is more precise/less sensitive to its surroundings than the right belt. This also meant that each PWM chosen for the right belt was based on an estimate of what PWM fit the speed the most, by listening to the motor and visually evaluating the system. By visual evaluation, the flex of the stationary belt is meant, as it flexed dependent on the driving belt being dragged up to speed, or slowed to match the speed.

B.1.3 Test Results

The resulting PWM to speed correlation is shown in table B.1 and the correlation for each belt individually is shown in table B.2. In figure B.3 a graphical representation of the correlations is shown for each belt and both belts in conjunction. For further modeling, the belts in combination will be used to describe the expected speeds of the vehicle. For further testing and actual operation of the vehicle, the respective PWM percentages will be used, to ensure as straight operation as possible.

Looking at figure B.4, it is clear a linear area in both the speed and amperes drawn correlation around 53% PWM. Therefore, this will be the steady-state speed of the vehicle. This steady state will be used to find turning capabilities in section B.3 starting page B.3. As of now, a linear response is prioritized above speed, further cementing the choice of 53% as steady-state.

PWM	Speed (km/h)	Speed (m/s)	Amperes Drawn	Peak Amperes	Voltage
11%	0.5	0.138	0.802	0.869	13.6
17%	0.6	0.167	0.877	0.955	13.6
24%	0.7	0.195	0.957	1.047	13.6
29%	0.8	0.222	0.987	1.118	13.6
37%	0.9	0.250	1.076	1.362	13.6
44%	1.0	0.278	1.143	1.336	13.6
50%	1.1	0.306	1.268	1.33	13.6
53%	1.2	0.333	1.331	1.42	13.6
56%	1.3	0.361	1.386	1.62	13.6
58%	1.4	0.389	1.392	1.72	13.6
60%	1.5	0.417	1.401	1.72	13.6
64%	1.6	0.444	1.426	2.11	13.6
71%	1.7	0.472	1.440	2.12	13.6
99%	1.8	0.500	1.451	2.12	13.6
99%	1.9	0.528	1.488	2.12	14.4

Table B.1: Correlation between PWM and speed for the whole robot. Only the relevant speeds are shown in this table, as the treadmill used for testing can only increment at 0.1 km/h intervals. It should be noted that the power supply used current protects above 2A.

PWM (Left Belt)	PWM (Right Belt	Speed (km/h)	Speed (m/s)	Voltage
10%	15%	0.5	0.138	13.6
16%	19%	0.6	0.167	13.6
21%	26%	0.7	0.195	13.6
26%	32%	0.8	0.222	13.6
36%	39%	0.9	0.250	13.6
44%	46%	1.0	0.278	13.6
48%	52%	1.1	0.306	13.6
51%	55%	1.2	0.333	13.6
53%	56%	1.3	0.361	13.6
55%	58%	1.4	0.389	13.6
57%	61%	1.5	0.417	13.6
62%	64%	1.6	0.444	13.6
70%	73%	1.7	0.472	13.6
98%	99% @ 14.4V	1.8	0.500	13.6

Table B.2: Correlation between PWM and speed for each belt. Only the relevant speeds have been shown in this table.

B.1.4 Summary

Testing the forward movement of the vehicle has shown that a linear area is present from 1.1-1.3 km/h, and in a broader perspective, from 0.8-1.7. Furthermore, it has shown that when moving, the vehicle draws less than 1.5A at 13.6V, and at startup more than 2.12A is drawn, setting boundaries for future battery selection. The test also showed that even though the motors are identical, they respond differently to the same PWM signal, implying that each motor/belt would benefit from its own PWM correlation. Enlarged photos are available in B.5 starting page 123.

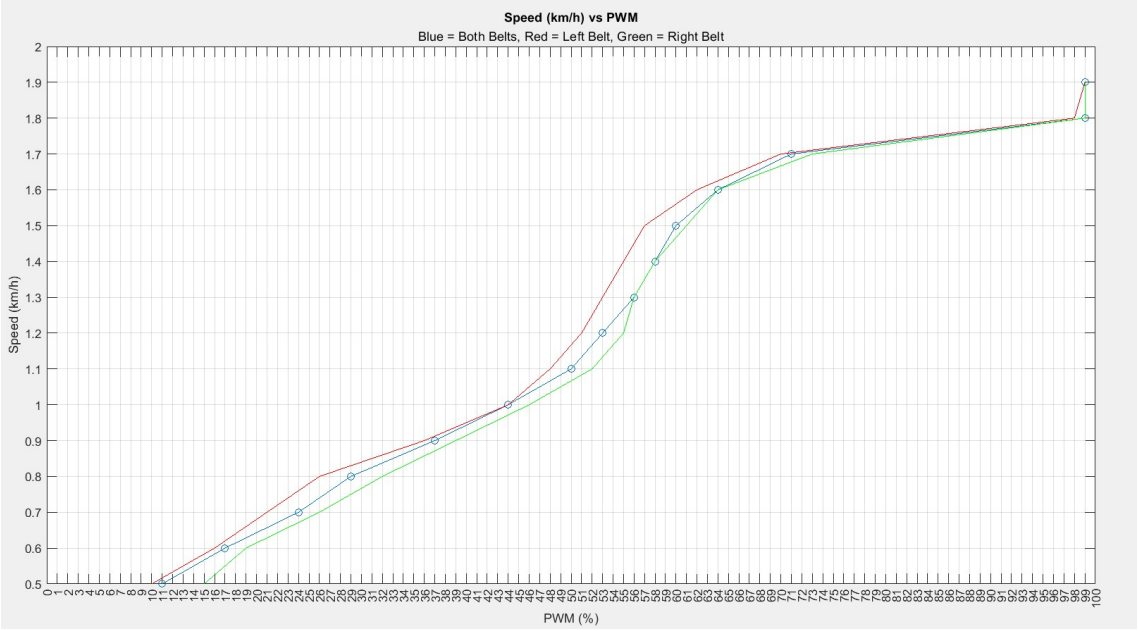


Figure B.3: PWM correlation between both belts together and individually.

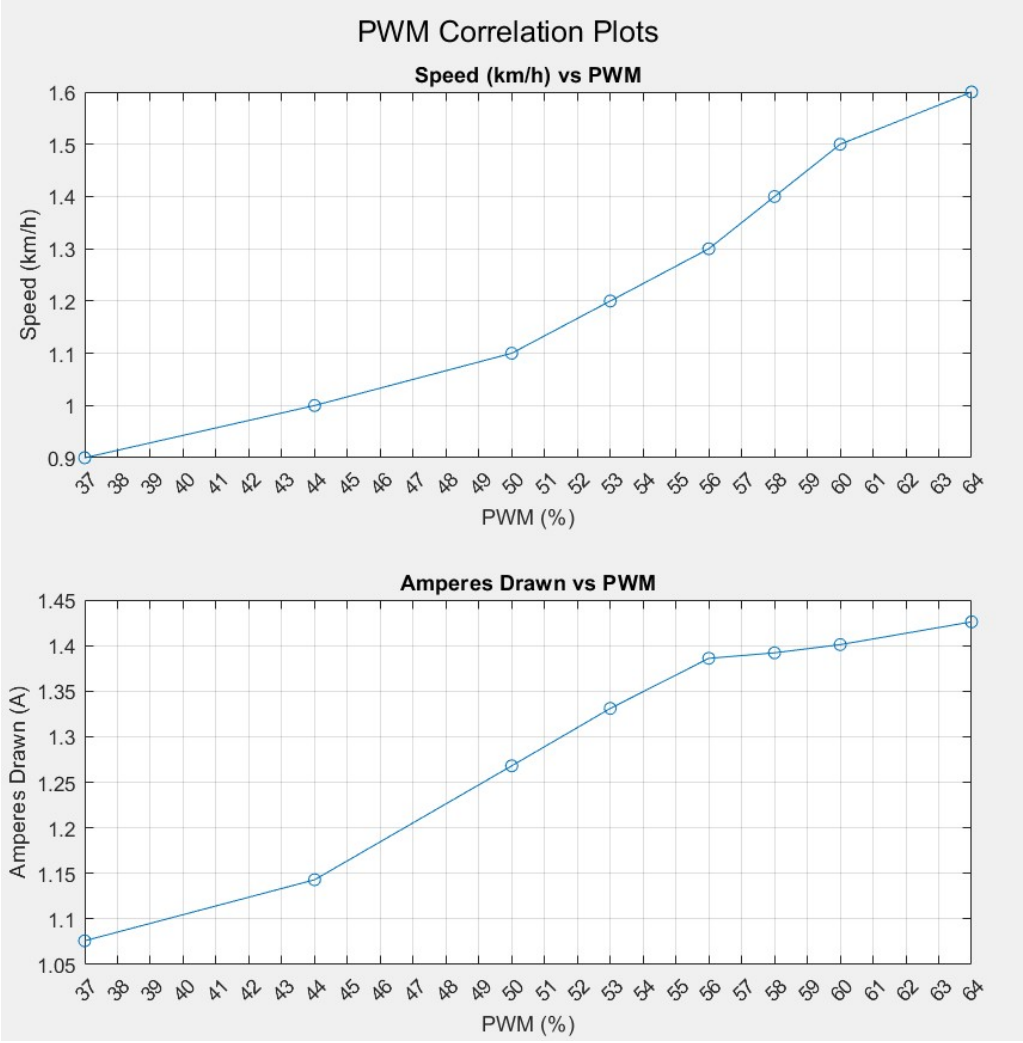


Figure B.4: Zoomed in version of 4.3 around 44-64% PWM, to showcase linearity present around 53%.

B.2 Test Report 2 - Starting the Robot From a Standstill

As a PWM correlation has been found for each belt and the vehicle as a whole in Test Report 1 - Forward Motion of Robot, it is possible to test behavior from a standstill. This means that now, the vehicle will be subject to a step-response at PWM equivalent to certain speeds, to establish transfer functions from 0 to the desired speed. This test is important as it gives an idea, of how quickly the system can change state. This is necessary knowledge as it impacts the stability of data obtained within the sampling time.

B.2.1 Test Setup

To perform the test the following is needed:

- 1x Robot
- 1x ZK-BM1 H-bridge
- 1x Oscilloscope, KeySight DSOX1102G
- 1x Triple Power Supply, HAMEG HM7042
- 4x 1m banana test leads (2 for GND and 2 for VCC)
- 1x 1m BNC coax cable
- 1x 0.1m BNC coax cable
- 1x BNC to banana plug adapter
- 1x Treadmill
- 1x Camera capable of 60fps
- 1x Camera-jig
- 1x Wire-jig
- 1x PC for data logging

The same test setup used in test report 1 can be used, see figure B.1 and B.2.
To conduct the test, the following procedure must be followed:

1. Set the voltage on the HAMEG power supply's third channel to 13.6V, and turn the current knob to the top. Furthermore, constant-current must be on, to ensure the power supply does not turn off when starting the motors.
2. Connect the power supply to the banana plugs on the vehicle.
3. Set the oscilloscope to wavegen, and output a square PWM signal with a peak-peak amplitude of 3.3V, frequency of 2kHz, offset of 0V, and PWM percentage at 1 to start.
4. Connect the oscilloscope wavegen with test probes on channels 2, 4, and ground on the ZK-BM1 H-bridge.
5. Connect the oscilloscope wavegen to channel 1 of the oscilloscope for visual interpretation.
6. Place the vehicle at the edge of the treadmill to use measuring marks.
7. Adjust PWM up or down to match the speed desired to test.
8. Turn on the camera and make sure that the power supply, robot, and measurement marks are visible.
9. Turn channel 3 on at the power supply.

10. Turn channel 3 off at the power supply when the nominal velocity has been reached.
11. Log data such as: PWM, voltage, amperes, and time/distance used to reach nominal speed.

B.2.2 Test Results

All possible speeds have been tested, but data to derive a transfer function has only been made for 0.5 and 1.2 km/h, as these are the most relevant speeds. The speeds to analyze are chosen as the slowest possible speed (to know what is required to even make the vehicle move) and the intended run speed. To verify the results and development of speed, the vehicle has been videotaped at 60 FPS in HD, and its movement in relation to the measuring marks has been noted down for every frame. This means that the current position of the vehicle has been noted at frame X1, and again at X2. While this method might seem vulnerable to faulty readings, the readings are only off of the expected velocity by 0.003 m/s for 0.5 km/h and by 0.001 m/s for 1.2 km/h. Therefore, the readings are accepted as true. Both speeds have been observed for 100 frames (1.67 seconds) after power is connected, and to find the mean velocity, only the speed within the last 40 frames is used.

$$Velocity_{Expected0.5} = 0.138\frac{m}{s} \quad Velocity_{Measured0.5} = 0.141\frac{m}{s}$$
$$Velocity_{Expected1.2} = 0.333\frac{m}{s} \quad Velocity_{Measured1.2} = 0.332\frac{m}{s}$$

The resulting graphs are shown in figure B.5 and B.6, with velocity along the Y-axis, time along X, and the transfer functions shown in green. The resulting transfer functions are given in equation B.1 and B.2. The transfer functions have been derived by looking at the graphs of velocities development over time.

$$G(s) = e^{-0.1*s} * 11 * \frac{0.0125}{0.15 * s + 1} \tag{B.1}$$

$$G(s) = e^{-0.2*s} * 53 * \frac{0.0062}{0.15 * s + 1} \tag{B.2}$$

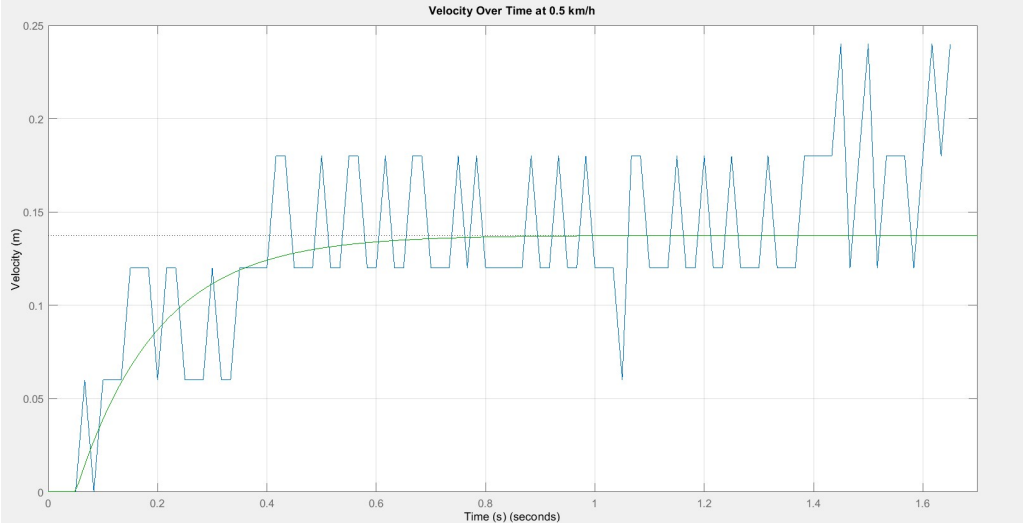


Figure B.5: Graph showing the relation between velocity and time for a step response equivalent to 0.5 km/h. The transfer function is shown in green.

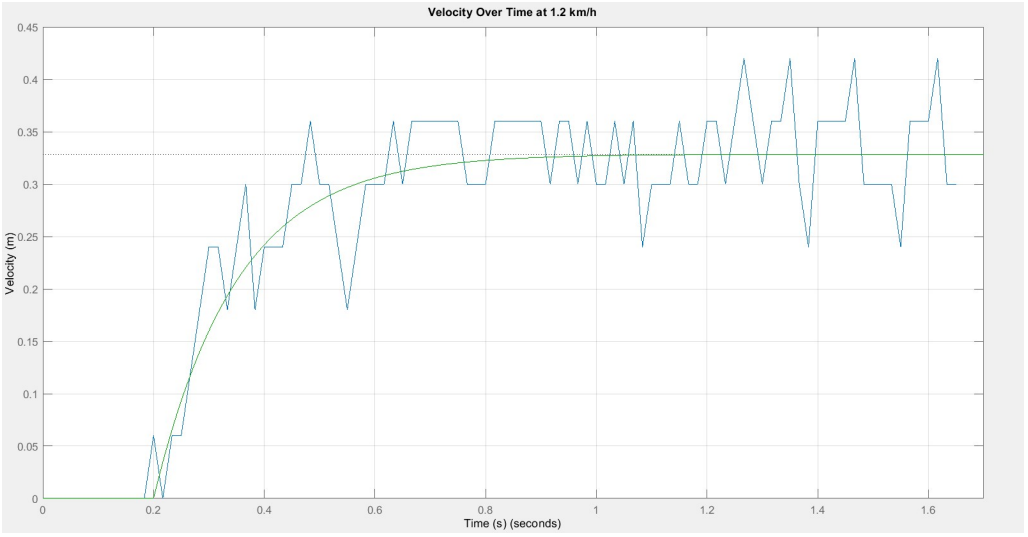


Figure B.6: Graph showing the relation between velocity and time for a step response equivalent to 1.2 km/h. The transfer function is shown in green.

From these graphs and the transfer functions it is evident that some delay is to be expected in both cases. Other than that, is is evident that the time constants are identical and that both systems are settled within 1 second.

B.2.3 Summary

The standstill start testing has concluded that the system reaches nominal velocity within approximately 1 second at relevant speeds (0.5 km/h and 1.2 km/h). Furthermore, it has resulted in equation B.1 and B.2 which describes the behavior of the system when starting from a standstill. The knowledge obtained is relevant for further system development, as it can now be designed to account for startup procedures, and will wait for X samples before checking if the nominal speed has been reached. As in the previous test report, enlarged photos are available in B.5 starting page 123.

B.3 Test Report 3 - Turning the Robot

Test 3 is designated to derive information on how fast the vehicle is capable of turning and how fast it turns at varying differences in speed among the belts, to determine if it has any impact on system operation and sample time. It should be noted that if any difference in speed is present between the two belts, then the vehicle will drive in circles, as shown in figure 4.14.

Test Setup

To perform the test the following is needed:

- 1x Robot
- 1x ESP32 WROOM with a joystick or similar button
- 1x ZK-BM1 H-bridge
- 1x Triple Power Supply, HAMEG HM7042
- 4x 1m banana test leads (2 for GND and 2 for VCC)
- 1x Treadmill
- 1x Camera capable of 60fps
- 1x Camera-jig
- 1x Wire-jig
- 1x PC for data logging and code manipulation

Most of the same test setup used in test report 1 can be used, see figure B.1 and B.2. The main difference is the placement of the camera in another jig, to record the vehicle from above. Recording the vehicle from above makes it easier to determine angle differences between timestamps. The new test setup is shown in figure B.7, where the new camera jig is visible above the treadmill and lines have been added every 2.5 cm on the treadmills band.

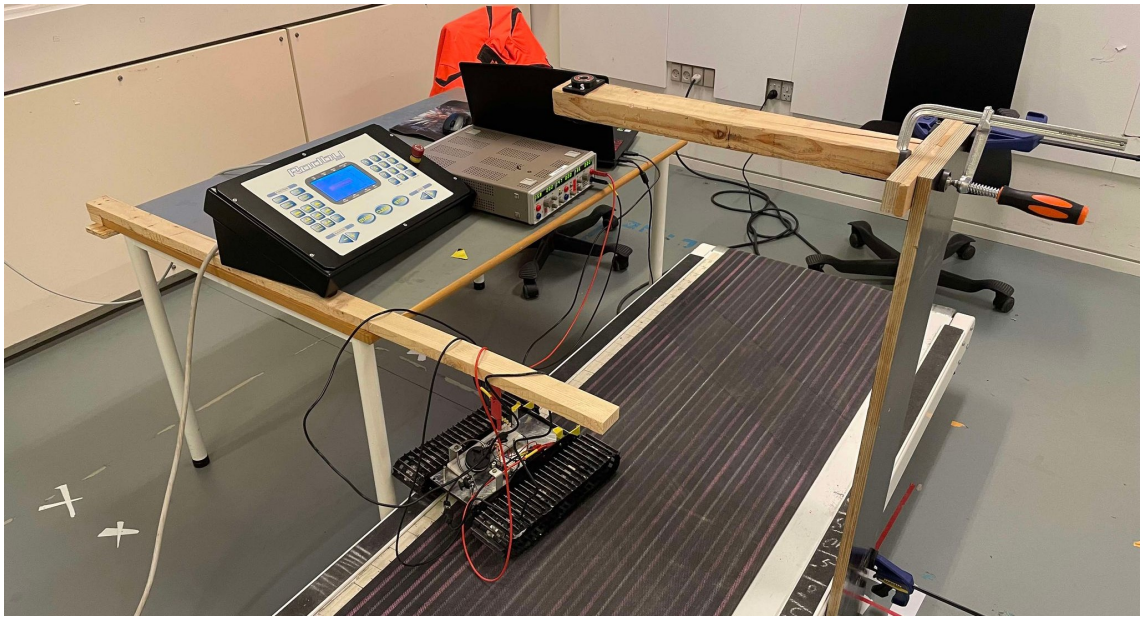


Figure B.7: Test setup used to record the vehicle turning. The main difference is the camera jig now placed above the treadmill and the addition of lines along the treadmill.

To conduct the test, the following procedure must be followed:

1. Set the desired speed (1.2 km/h) to test on the Rodby control panel, by first pressing "start", then "+" until the desired speed is met. The treadmill should not move until "start" is pressed again.
2. Set the voltage on the HAMEG power supply's third channel to 13.6V, and turn the current knob to the top. Furthermore, constant-current must be on, to ensure the power supply does not turn off when starting the motors.
3. Connect the power supply to the banana plugs on the vehicle.
4. Place the vehicle on the left of the treadmill to leave the most possible room for turning.
5. Connect the ESP32 to the H-bridge, with the following pins: H-bridge input 1=32, H-bridge input 2=33, H-bridge input 3=25, H-bridge input 4=26.
6. Upload code to the ESP32 matching the desired difference in speed.
7. Turn channel 3 on at the power supply and press start on the treadmill set at 1.2 km/h.
8. Turn on the camera and make sure that the power supply, robot, and measurement marks are visible.
9. With the vehicle moving forward on the treadmill, press the button integrated into the joystick to enable turning. Remember there is a delay of 5 seconds in the code, to ensure the testee can remove themselves from the vehicle.
10. Turn channel 3 off at the power supply and the treadmill should be stopped when the vehicle reaches the other side of the treadmill.
11. Reset the ESP32.

Actual Testing

While testing, it became evident that even though the PWM signals for each belt were adjusted to match their individually measured values, the right belt continued to lag behind. While this did not pose any significant turning, it is still worth noticing and implies that the PWM of the right belt possibly should be increased even more. Furthermore, the joystick was not optimal to use, as its ground pin often fell out, rendering the test obsolete, as it did nothing.

Test Results

To analyze the angle difference/development when turning, the MATLAB function "vision.pointtracker" function from the computer vision toolbox is used. It functions by specifying two points on the vehicle and then measuring a vector from point a to point b's development over time, in relation to a set vector, such as an X-axis. In this case, either the rear edge or two parallel holes in the vehicle have served as tracking points, as shown in figure B.8 and B.9. The vision tool then measures the green vector angle in comparison with an x-axis. As the video recorded is at 60 FPS, the resulting measurements are then updated every 0.167 seconds. A disadvantage of measuring the angle like this is, that a lot of noise is introduced as a result of non-optimal lighting conditions and the angle might be skewed because of the camera's focal point.

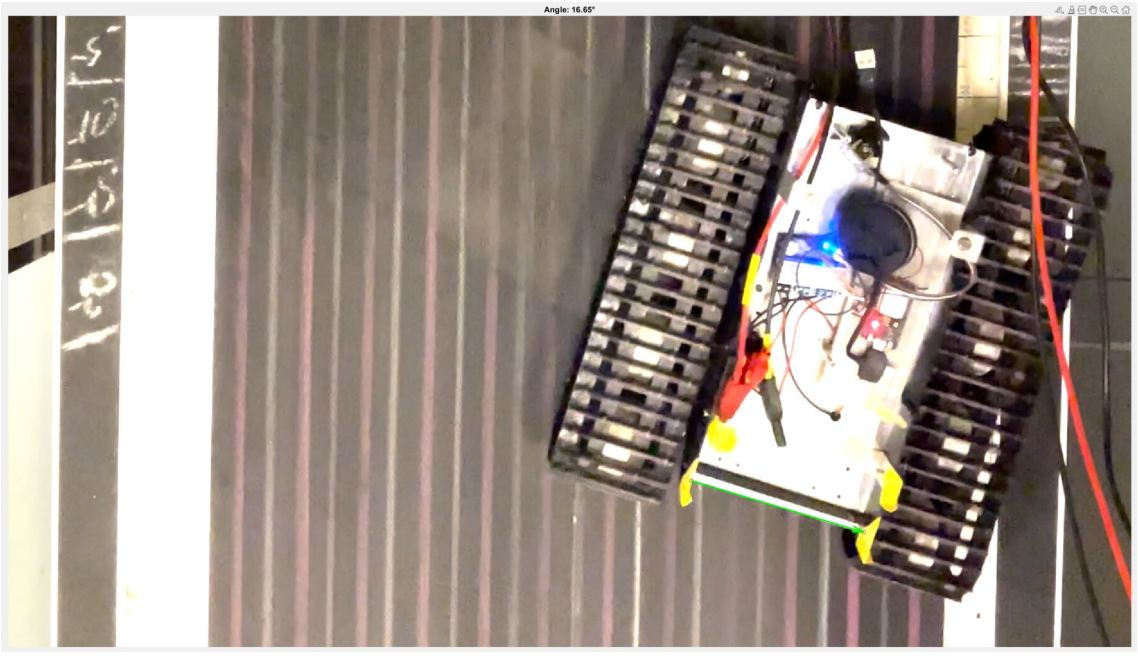


Figure B.8: End position and angle of the vehicle when subjecting it to a speed difference of 0.7 km/h. Note the green vector placed along the rear edge of the vehicle.

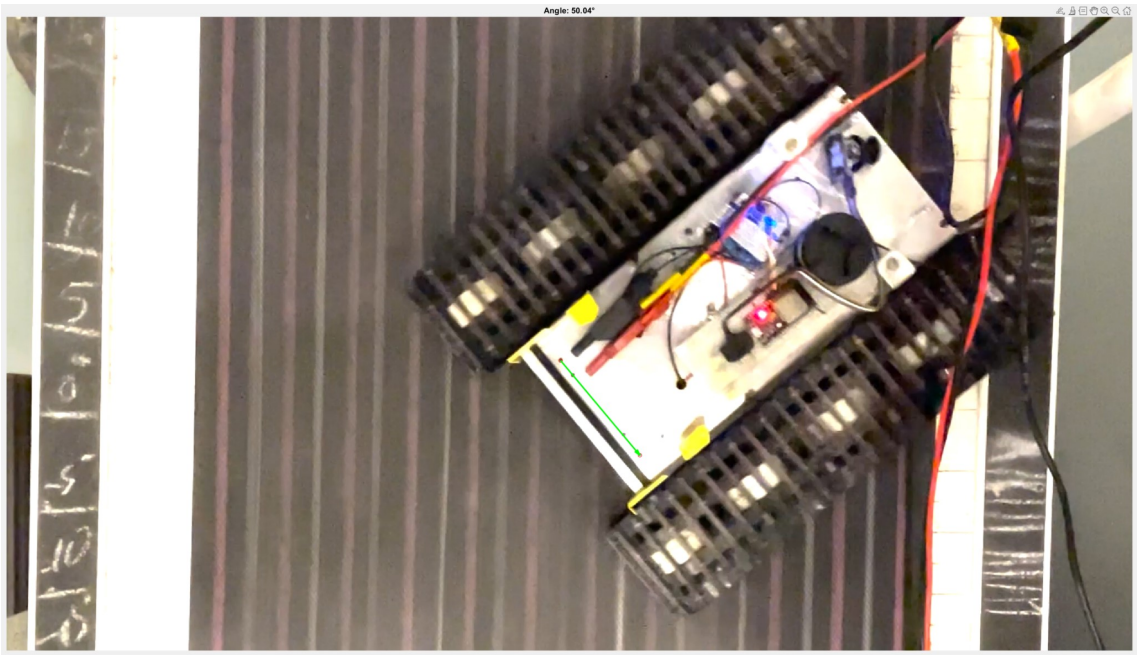


Figure B.9: End position and angle of the vehicle when subjecting it to a speed difference of 1.3 km/h. Note the green vector placed between two parallel holes on the vehicle.

Just as with "Test Report 2 - Starting the Robot From a Standstill", only a few speeds are analyzed. In this case, it is a speed difference of 0.7 km/h and 1.3 km/h between the belts, specifically setting the speeds at 0.8/1.5 km/h to obtain 0.7 km/h difference, and 0.5/1.8km/h to obtain the 1.3 km/h difference. These speeds are chosen, as 1.3 km/h is the maximum speed difference possible, while the vehicle is moving, and 0.7 is chosen subjectively, as the vehicle turns without exaggerating with this difference. Furthermore, by remaining at 0.8 km/h and 1.5 km/h, the vehicle remains within the mostly linear area shown in figure B.3.

The resulting angle development for each of these speed differences are shown in figure B.10 and B.11.

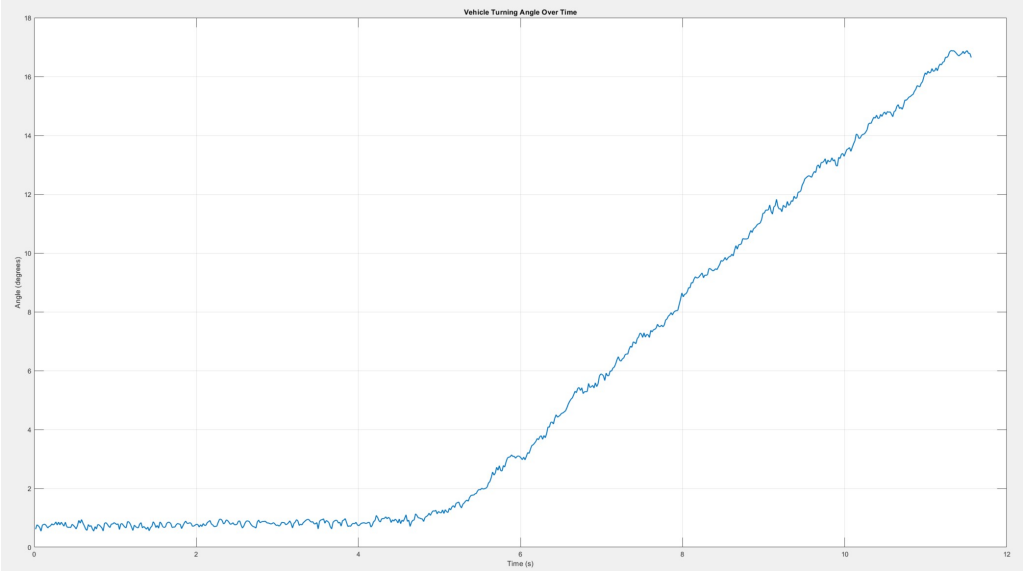


Figure B.10: Angle development/change over time when the vehicle is subjected to a speed difference of 0.7 km/h. Note that the first 5 seconds are driving straight ahead, as there is a 5-second delay, between pressing the button and the PWM being altered.

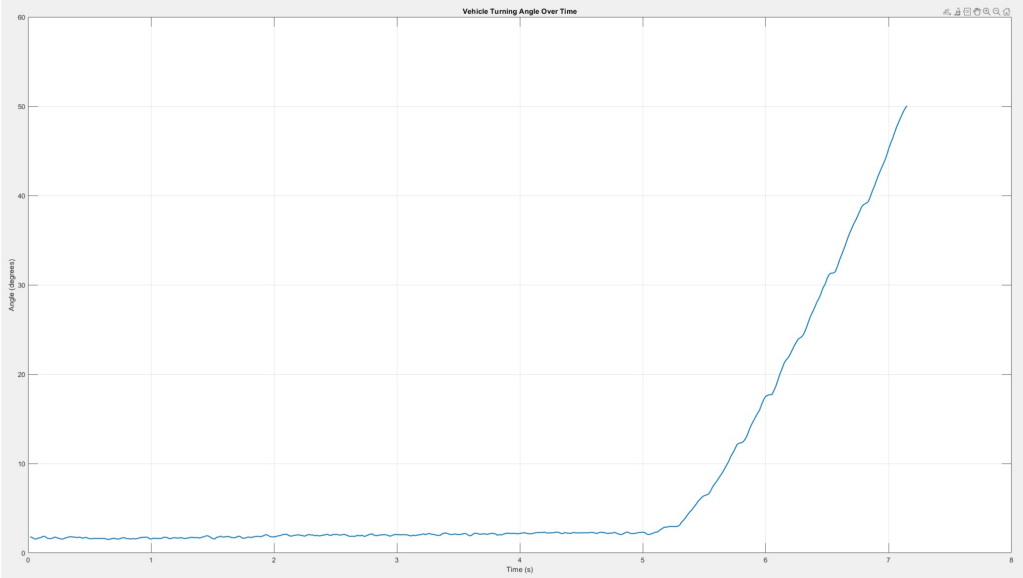


Figure B.11: Angle development/change over time when the vehicle is subjected to a speed difference of 1.3 km/h. Note that the first 5 seconds are driving straight ahead, as there is a 5-second delay, between pressing the button and the PWM being altered.

The noteworthy aspect of these graphs is the change of angle between each timestep. If the timestep is set to 1 second, the average change of angle for a difference of 0.7 km/h is 2.48 degrees or 0.04 degrees for each frame. The answer is a little more ambiguous for a difference of 1.3 km/h. For the first half-second the average change is only 21.69 degrees, while for the next half-second, the change is 27.37 degrees. Furthermore, when looking at the average angle difference for the last half-second being measured, the angle difference is 27.48 degrees. If the last 100 frames are used, the average change between

frames is 0.437 degrees. This hints at a first-order system capable of describing how long it takes to go from no steering to full steering. Therefore, the transfer function shown in equation B.6 has been derived. The transfer function has been derived by observing the angle difference between each frame, which somewhat resembles a first-order transfer function when observing the graph shown in figure B.12 with the approximated transfer function in green. The negative spikes shown in the graph are a result of the physical properties of the belts on the vehicle. When turning hard, the belt has to overcome the friction between the treadmill and the belt, before it can continue turning. In the video "difference13TopViewForMatlab.mp4", it is clear to see that the vehicle "skips" now and then, before continuing to turn, resulting in the negative spikes, as the angle difference is negligible to the previous frame. The estimated variables for a transfer function describing the system are:

$$Gain = 0.437 \quad - Found\ by\ calculating\ the\ average\ value\ across\ the\ last\ 100\ frames\ of\ video \tag{B.3}$$

$$\tau = 0.35[s] \tag{B.4}$$

$$Delay = 0.0166 * 300 = 4.988[s] \tag{B.5}$$

$$G(s) = e^{-4.988*s} * \frac{0.437}{0.35 * s + 1} \tag{B.6}$$

The gain was found by taking the average angle difference between the last 100 frames, and the time constant was estimated by trying different values until a proper fit was found. The delay is derived from the video, as the last frame before the vehicle begins turning. The largest fault could be the gain, as the vehicle might not have reached its maximum turning speed before driving off the treadmill. This is supported by the fact that the average angle difference for the last 50 frames is 0.506 degrees and 0.510 degrees for the last 25 frames. Looking at the graph, it could also seem as if it continues to rise beyond the measured.

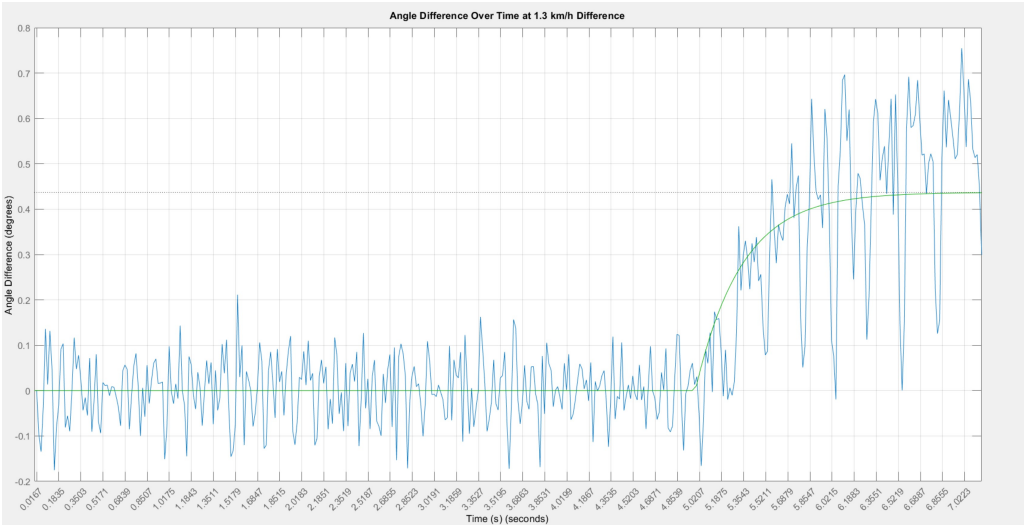


Figure B.12: Angle difference over time (blue) with estimated transfer function (green).

A similar transfer function and graph can be made for the difference of 0.7 km/h, although the first-order system is not nearly as apparent because of noise, as shown in figure B.13. As with figure B.12, the transfer function has mostly been found by trying qualified values when looking at the raw data in Excel. Such qualified values are the average angle difference found to be 0.04 between each frame, and a time constant of 0.075 derived from the average across 10 frames changing from approximately 0.025 or lower to more than 0.04 approximately 0.3 seconds after the turning has begun. A snippet of the Excel sheet can be seen in table B.3. This yields the transfer function given in equation B.10, where the delay is the same as for B.6, since both steps are applied 300 frames into the video. The estimated variables for a transfer function describing the system are:

$$Gain = 0.040 \quad - Found\ by\ calculating\ the\ average\ value\ across\ the\ last\ 386\ frames\ of\ video \tag{B.7}$$

$$\tau = 0.075[s] \tag{B.8}$$

$$Delay = 0.0166 * 300 = 4.988[s] \tag{B.9}$$

$$G(s) = e^{-4.988*s} * \frac{0.04}{0.075 * s + 1} \tag{B.10}$$

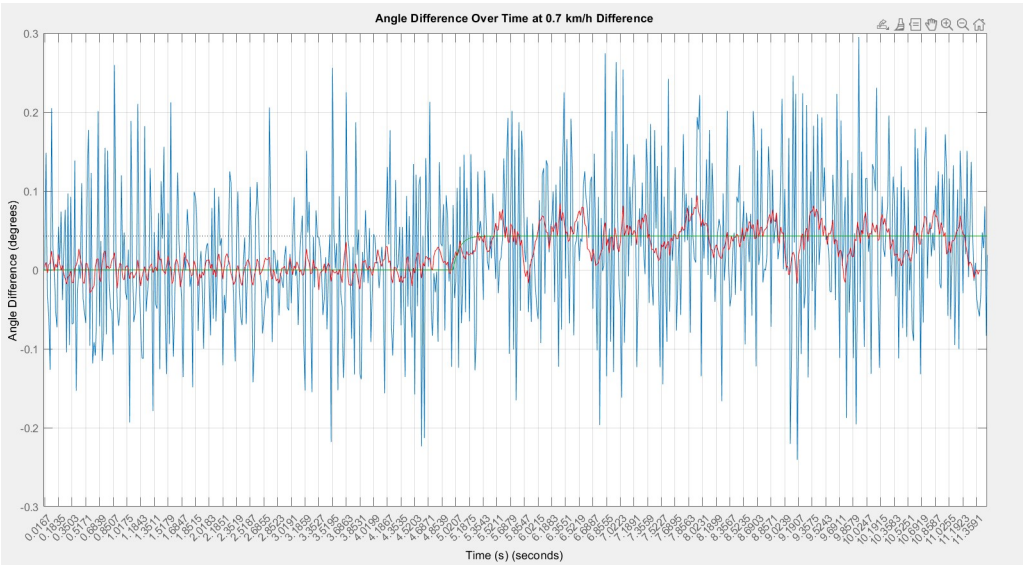


Figure B.13: Angle difference over time (blue) with estimated transfer function (green) and average angle differences across the next 10 samples (red).

Time (s)	Current Angle	Angle Difference	Average Angle Difference Across Next 10
4,988316667	1,214799556	0,0820665	0,013667
5,005	1,179046752	-0,035752804	0,015863
5,021683333	1,164486686	-0,014560067	0,023333
5,038366667	1,268226393	0,103739707	0,01826
5,05505	1,144213249	-0,124013144	0,022541
5,071733333	1,275002296	0,130789047	0,036891
5,088416667	1,207490816	-0,06751148	0,026303
5,1051	1,177060877	-0,030429938	0,020322
5,121783333	1,323033068	0,145972191	0,015676
5,138466667	1,269403599	-0,05362947	0,01354
5,15515	1,373431889	0,10402829	0,023405
5,171833333	1,412373166	0,038941277	0,019206
5,188516667	1,347083708	-0,065289458	0,018928
5,2052	1,493634275	0,146550567	0,021654
5,221883333	1,513126882	0,019492606	0,019594
5,238566667	1,538029811	0,024902929	0,024535
5,25525	1,410713332	-0,127316479	0,023221
5,271933333	1,333819892	-0,076893439	0,035929
5,288616667	1,458437772	0,124617879	0,047923

Table B.3: Snippet of the Excel sheet containing angle development with a difference of 0.7 km/h. note that between the last 3 timestamps, the angle difference across the next 10 timesteps varies quite a lot.

B.3.1 Summary

Testing the turning capabilities of the vehicle has given valuable insight into how fast the vehicle can turn, and how long it takes to achieve it. In the product, the chosen speed difference/turn rate is set to 0.7 km/h as this is fast enough to provide a turning effect while being slow enough to not exaggerate the movement of the vehicle. Using this value, the vehicle is at full turning speed approximately 0.3 seconds after being told to turn and turns by 2.48 degrees every second.

The other value tested (1.3 km/h difference) yielded turning capabilities of approximately 27.5 degrees every 0.5 seconds, or 55 degrees every second. While this capability will be useable for driving around corners, it is too big a difference to control small errors when driving forward. While it may not have reached full turning speed on the treadmill, the settling time is estimated to be 1.4 seconds, with the data available.

Having obtained the transfer functions for a step equalling a difference in speed, it becomes apparent that the delay and time constant will first influence operation at sample times below 1.4 seconds for the largest possible speed difference while moving forward. Assuming a sample time of 0.5 seconds, this is catastrophic, as the programming then might increase the gain, until the final angle difference of 27.5 degrees every 0.5 seconds is met. For the expected working difference of 0.7 km/h, the sample time has to be lower than 0.3 seconds to be influenced by the settling time. As in the previous test reports, enlarged photos are available in B.5 starting page 123.

B.4 Test Report 4 - Rotating the Robot

The final test of the preliminary system is destined to describe how fast it can rotate around its own Z-axis, and if it is possible while "stationary", or if the robot moves along the x and/or y direction. The demand specification requires this functionality "Operate the Robot" on page 9.

Test Setup

To perform the test the following is needed:

- 1x Robot
- 1x ESP32 WROOM with a joystick or similar button
- 1x ZK-BM1 H-bridge
- 1x Triple Power Supply, HAMEG HM7042
- 4x 1m banana test leads (2 for GND and 2 for VCC)
- 1x Treadmill
- 1x Camera capable of 60fps
- 1x Camera-jig
- 1x Wire-jig
- 1x PC for data logging and code manipulation

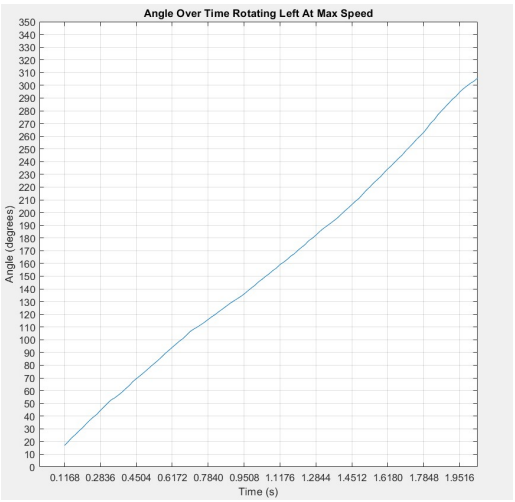
To conduct the test, the following procedure must be followed:

1. Set the voltage on the HAMEG power supply's third channel to 13.6V, and turn the current knob to the top. Furthermore, constant-current must be on, to ensure the power supply does not turn off when starting the motors.
2. Connect the power supply to the banana plugs on the vehicle.
3. Place the vehicle in the middle of the treadmill to leave the most possible room for rotating.
4. Connect the ESP32 to the H-bridge, with the following pins: H-bridge input 1=32, H-bridge input 2=33, H-bridge input 3=25, H-bridge input 4=26.
5. Upload code to the ESP32 matching the desired difference in speed - in this case maximum in each direction.
6. Turn channel 3 on at the power supply.
7. Turn on the camera and make sure that the power supply, robot, and measurement marks are visible.
8. Press the button integrated into the joystick to enable rotation. Remember there is a delay of 5 seconds in the code, to ensure the testee can remove themselves from the vehicle.
9. Turn channel 3 off at the power supply after a few rotations.
10. Reset the ESP32.

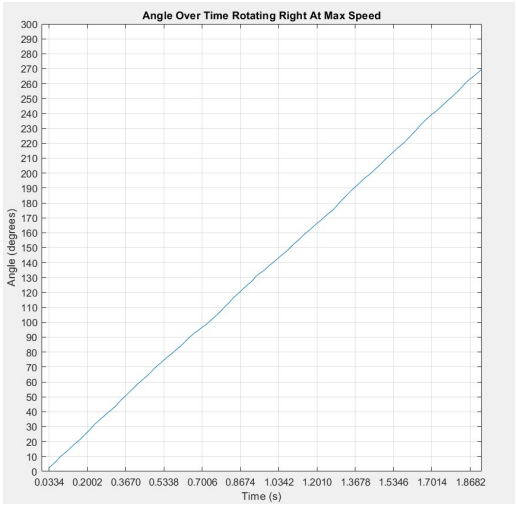
Actual Testing

While this was a relatively simple test physically, the data processing was greatly hindered by the data and power lines running to the vehicle. As in "Test Report 3 - Turning the Robot", MATLABs computer vision toolbox and specifically the "vision.PointTracker" tool was used to derive angle development over time. The problem arises beyond 260-279 degrees of rotation, as the points being tracked on the vehicle, become obstructed by the wires, resulting in an automatic shutdown of the process.

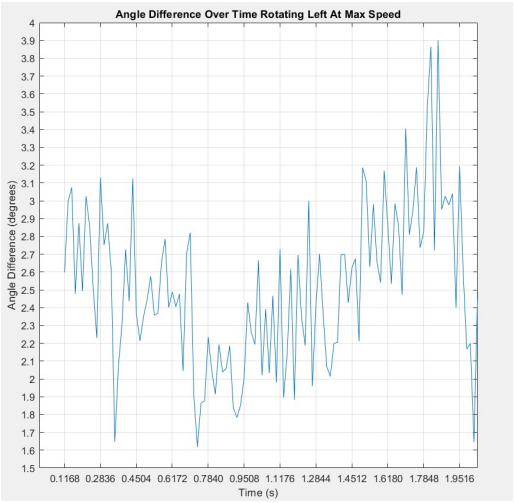
Another set of data has been made in each direction, when the vehicle is doing its third rotation (and is at maximum rotational speed), to determine the maximum rotational velocity. The average angle difference between each frame is found to be 2.38 degrees when rotating right, and 2.54 degrees when rotating left. In figure B.14a and B.14b the angle development can be seen, when the vehicle is at maximum rotational speed, along with the angle differences in figure B.14c and B.14d.



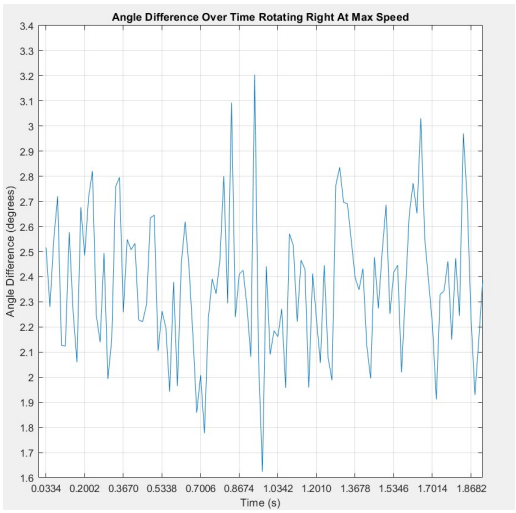
(a) As can be seen, the function is practically linear, witnessing maximum of 153 degrees/second velocity has been reached.



(b) As can be seen, the function is practically linear, witnessing maximum of 143 degrees/second velocity has been reached.



(c) While it may not look as such, an average of 2.54 degrees in change is made between each frame.



(d) While it may not look as such, an average of 2.38 degrees in change is made between each frame.

Figure B.14: Various graphs describing the systems behavior at maximum rotational velocity.

Furthermore, when looking at the video material for turning both left and right, it is evident that for the first rotation, the vehicle is mostly stationary, whereas for the following rotations, it begins to move forward in relation to its initiate position and orientation. After 4 rotations, it has moved approximately 9 cm forward, a little to the left when rotating left around itself, and a little to the right when turning right around itself.

Test Results

The test results available are a little muddled. As explained previously, the wires supplying data to the ESP and power to the motors, are hindering optimal computer vision and thereby effective data analysis. Because of the wires, the maximum angle change measured for turning left is 260 degrees, and for turning right it is 279 degrees. In figure B.15 it is visible that the change in degrees are similar, no matter which direction the vehicle is turning.

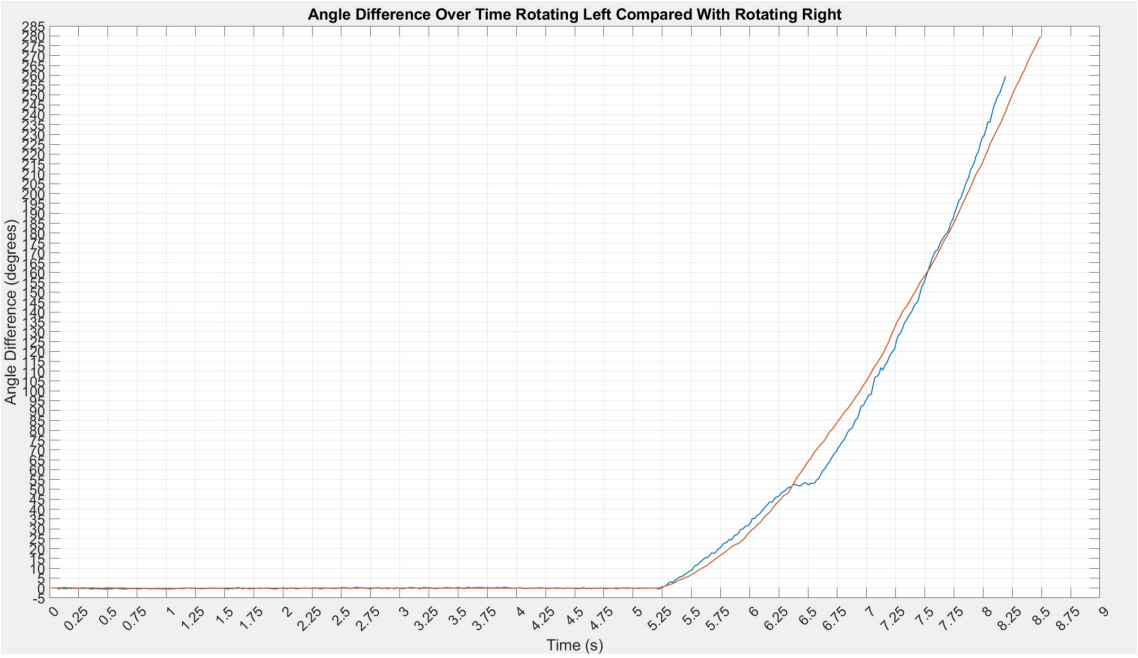


Figure B.15: Comparison of how the vehicle rotates in both directions. Orange is rotating right, and blue is rotating left.

As in "Test Report 3 - Turning the Robot", the angle difference between each frame will be used to determine a transfer function of how fast the vehicle is rotating and how fast it reaches maximum velocity. In figure B.16 an approximated transfer function for rotating left is shown, along with its corresponding velocity plot. In figure B.17 an approximated transfer function for rotating right is shown, along with its corresponding velocity plot. As in "Test Report 3 - Turning the Robot", the transfer functions are derived by a combination of looking at the plots describing angle differences over time, and the Excel sheets with average angle difference across the next 10 samples calculated in a separate column. As the vehicle doesn't reach maximum rotational speed in any of the tests that could be analyzed, the rise time has been estimated instead of the settling time, to estimate a time constant in both directions. For rotating left, the variables are:

$$Gain = 2.54 \tag{B.11}$$

$$Delay_{Button} = e^{-4.988*s} \tag{B.12}$$

$$Delay_{System} = e^{-0.12*s} \tag{B.13}$$

$$\tau = 1.2 \tag{B.14}$$

$$G(s) = e^{-4.988*s} * e^{-0.12*s} * \frac{2.54}{1.2*s+1} \tag{B.15}$$

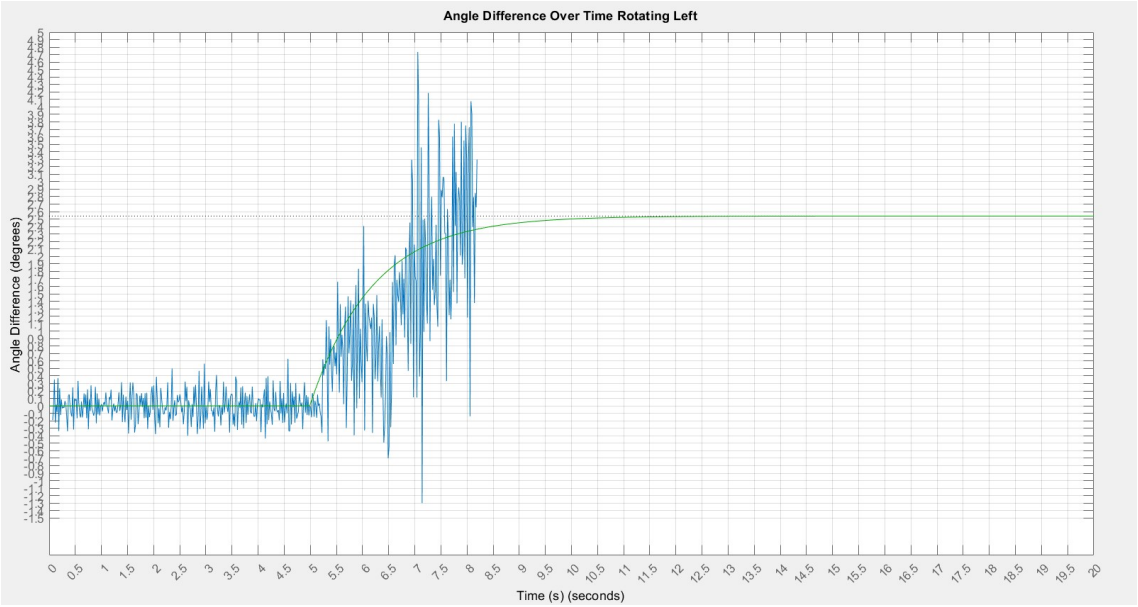


Figure B.16: Estimated transfer function of rotating left, with known maximum rotational velocity.

And for rotating right:

$$Gain = 2.38 \tag{B.16}$$

$$Delay_{Button} = e^{-4.988*s} \tag{B.17}$$

$$Delay_{System} = e^{-0.13*s} \tag{B.18}$$

$$\tau = 1.5 \tag{B.19}$$

$$G(s) = e^{-4.988*s} * e^{-0.13*s} * \frac{2.38}{1.5*s+1} \tag{B.20}$$

The maximum rotational velocity is in the worst case first achieved after 6 seconds. However, this is completely irrelevant, as the finished product most likely will never experience a situation, wherein the vehicle has to rotate around itself as fast as possible. Therefore, the valuable insight provided by this experiment is given by how fast it can rotate any number of degrees around itself. From the graph shown in figure B.15, a general function can be described by equation B.21, which is derived by observing the

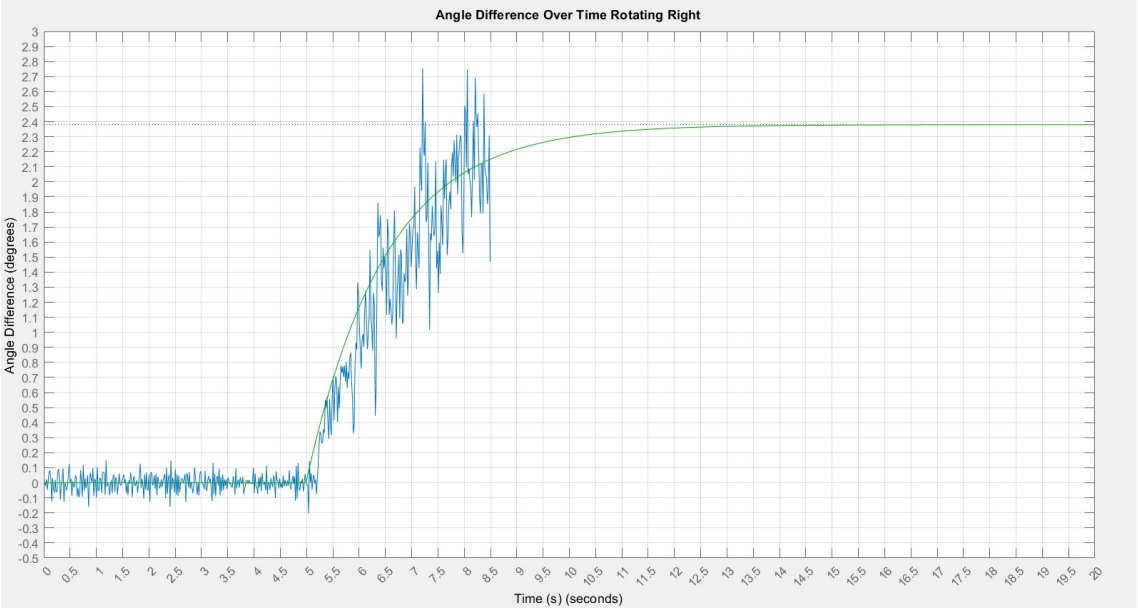


Figure B.17: Estimated transfer function of rotating right, with known maximum rotational velocity.

intersections for rotating right (7.75,185), (8.25,250) and for rotating left in (7.75,190), (8.0,225). Both intersection sets describe a difference in degrees of approximately 35 degrees every 0.5 seconds. To compensate a little for the slower rotation at the beginning and for computational efficiency, a value of 60 degrees every second is chosen to describe how the vehicle rotates. A larger version of the comparison plot is shown in B.32 on page 137. If the graph is closely examined, it becomes apparent the function of 60 degrees every 1 second, fits decently for rotations of 90 and 180 degrees.

$$f(x) = 60_{degrees} * 1_{Second} \tag{B.21}$$

B.4.1 Summary

In summary, transfer functions describing how the vehicle rotates have been made for both directions, and a generalized linear function has been estimated for rotating up to 180 degrees with little error. Furthermore, the test has yielded insight into how the vehicle behaves when rotating around its Z-axis. As in the previous test reports, enlarged photos are available in B.5 starting page 123.

B.5 Test Reports - Extra Photos

This section serves as a lookup for all the extra photos needed to describe behavior in each test report and for large format of the most relevant photos. The photos will not be accompanied by any text, except for that in the caption.

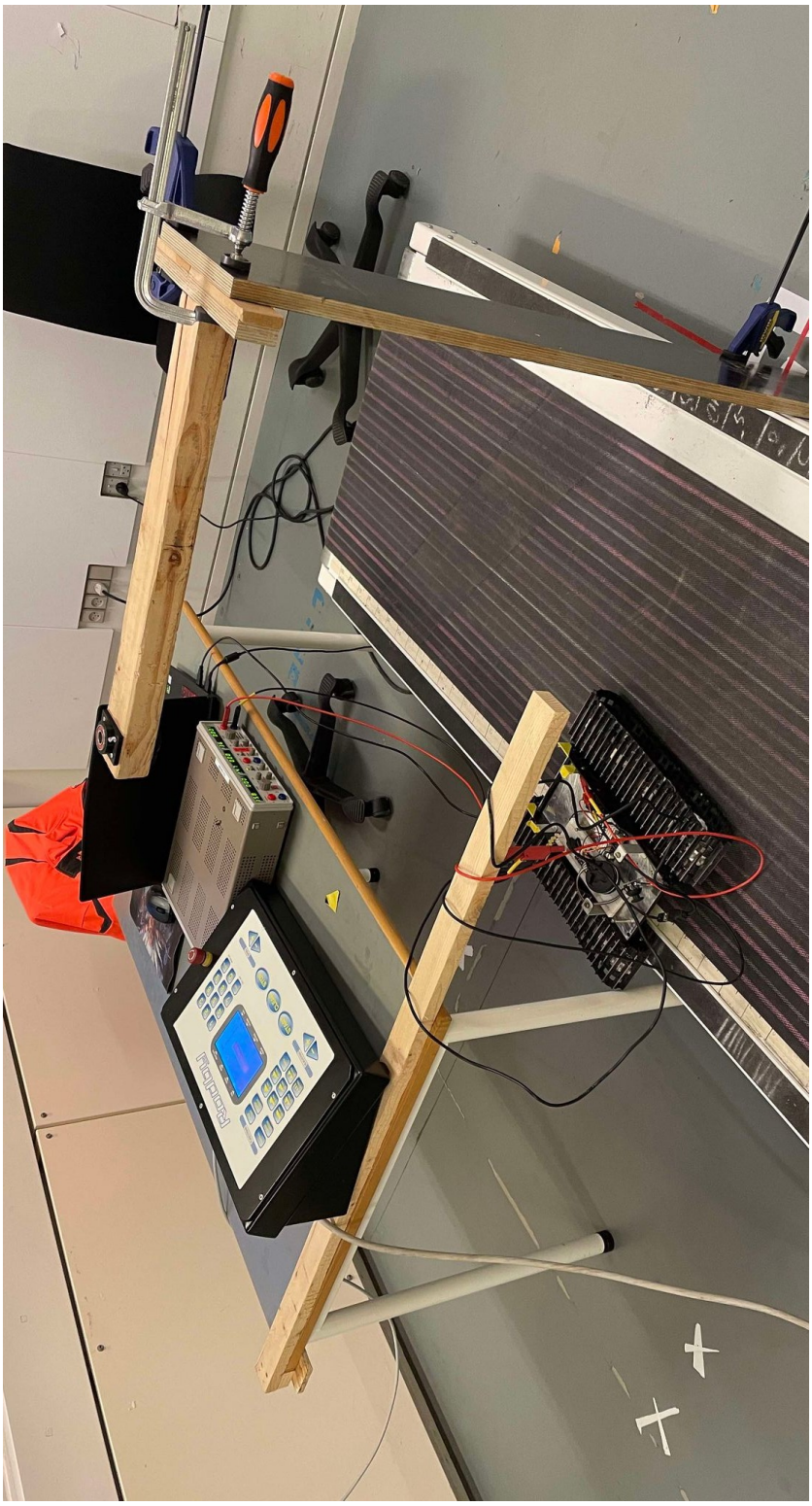


Figure B.18: Test setup used to record the vehicle turning. The main difference is the camera jig now placed above the treadmill and the addition of lines along the treadmill.

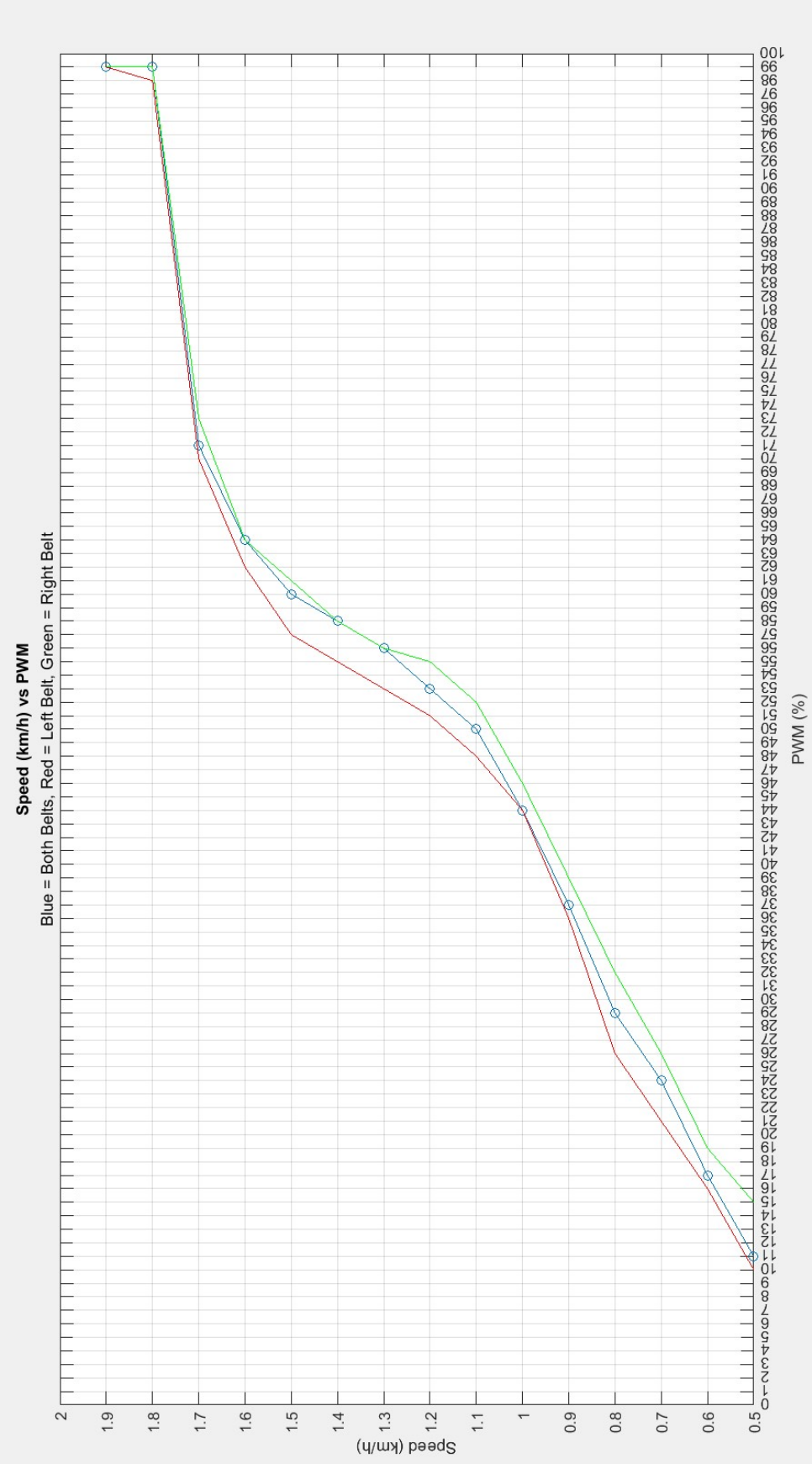


Figure B.19: PWM correlation between both belts together and individually

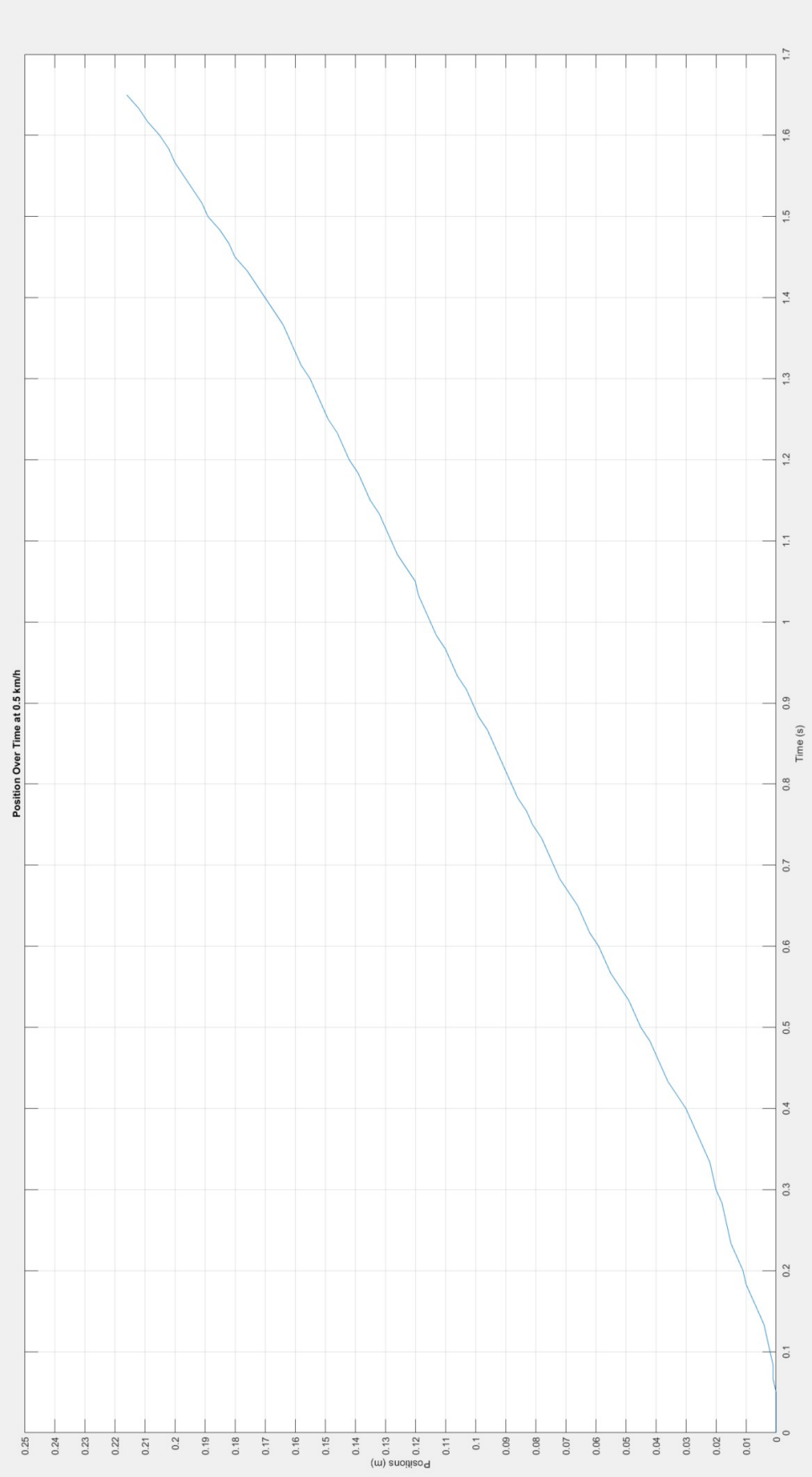


Figure B.20: Graph showing current position over time at 0.5 km/h.

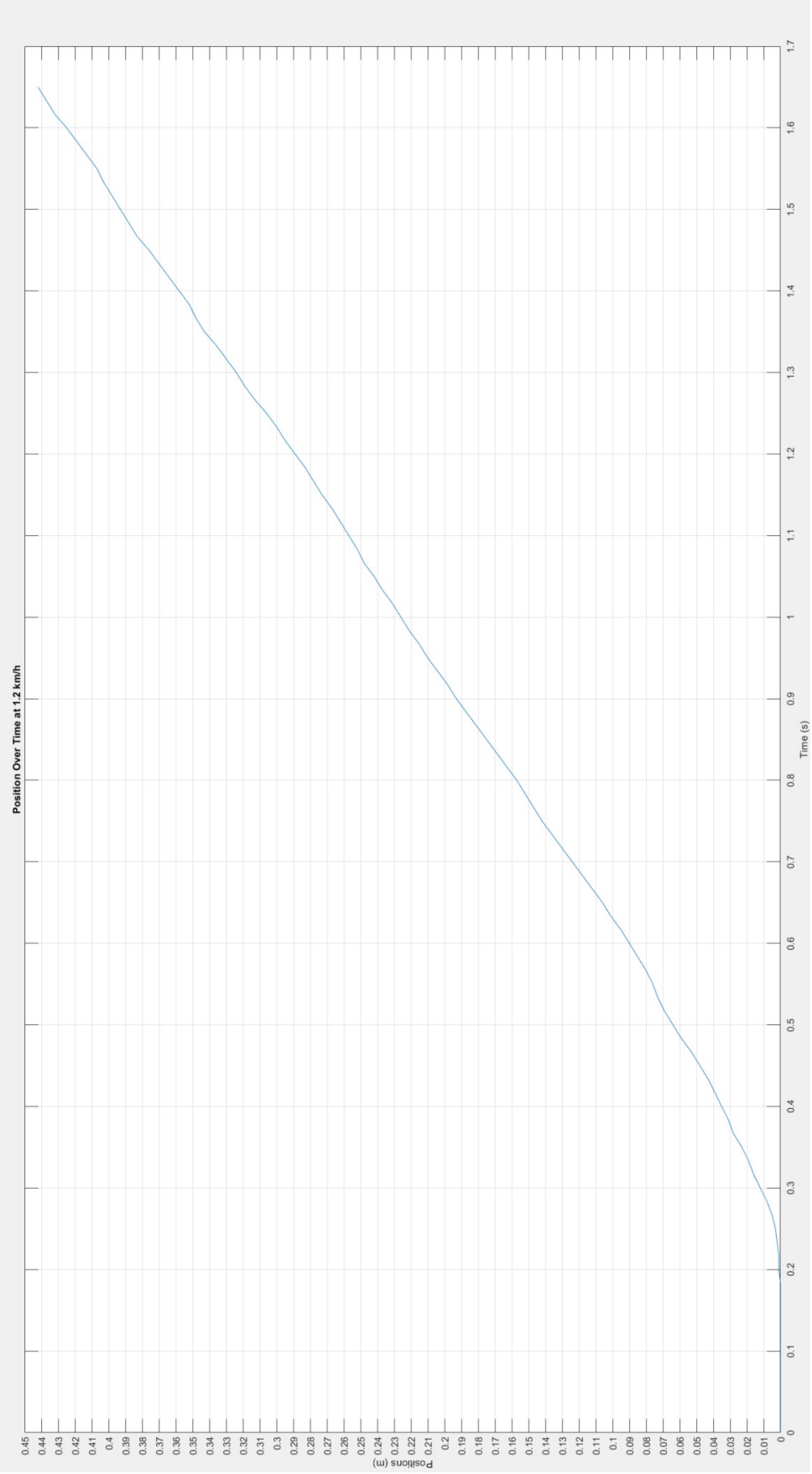


Figure B.21: Graph showing current position over time at 1.2 km/h.

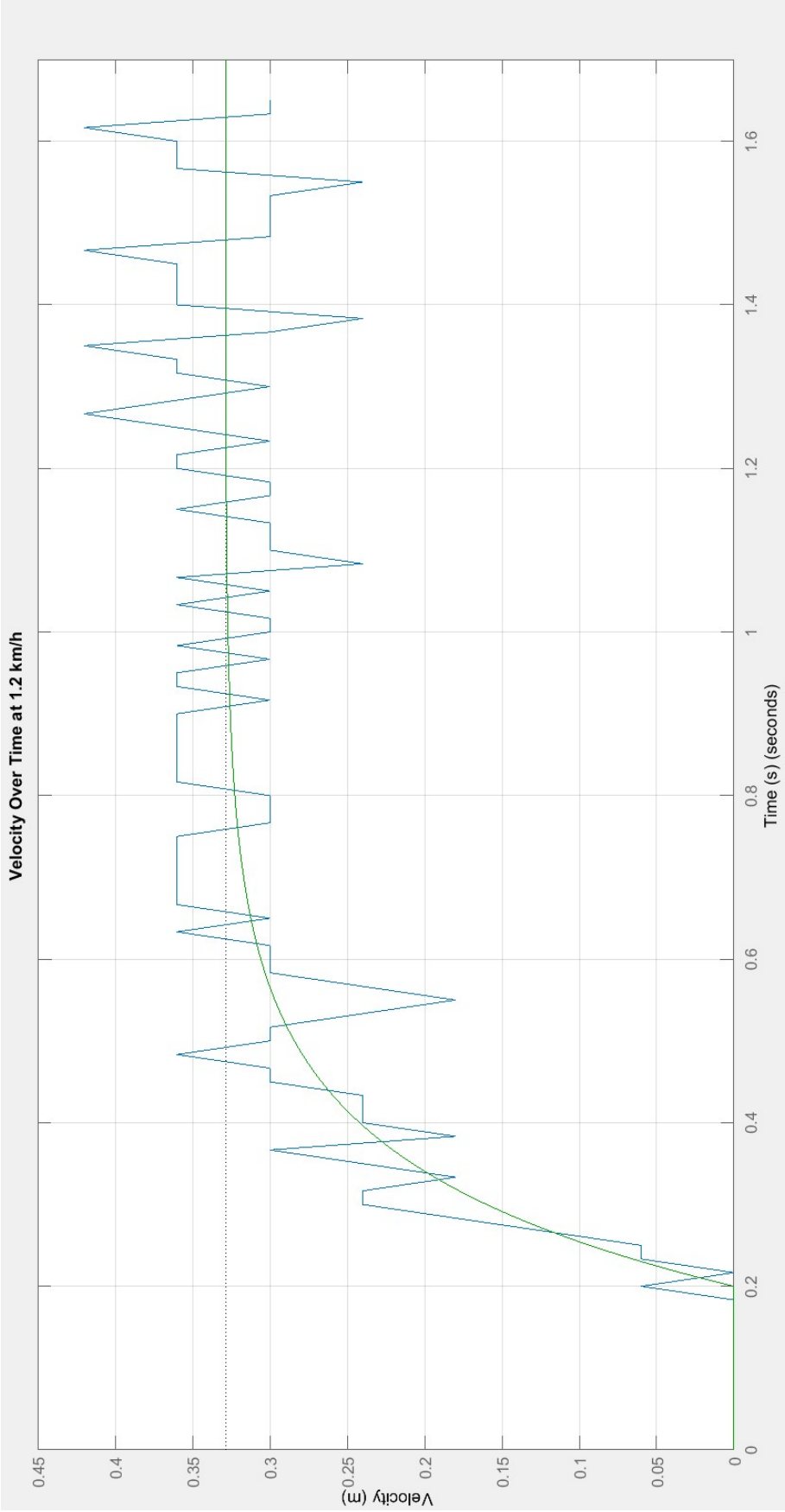


Figure B.22: Graph showing the relation between velocity and time for a step response equivalent to 1.2 km/h. The transfer function is shown in green.

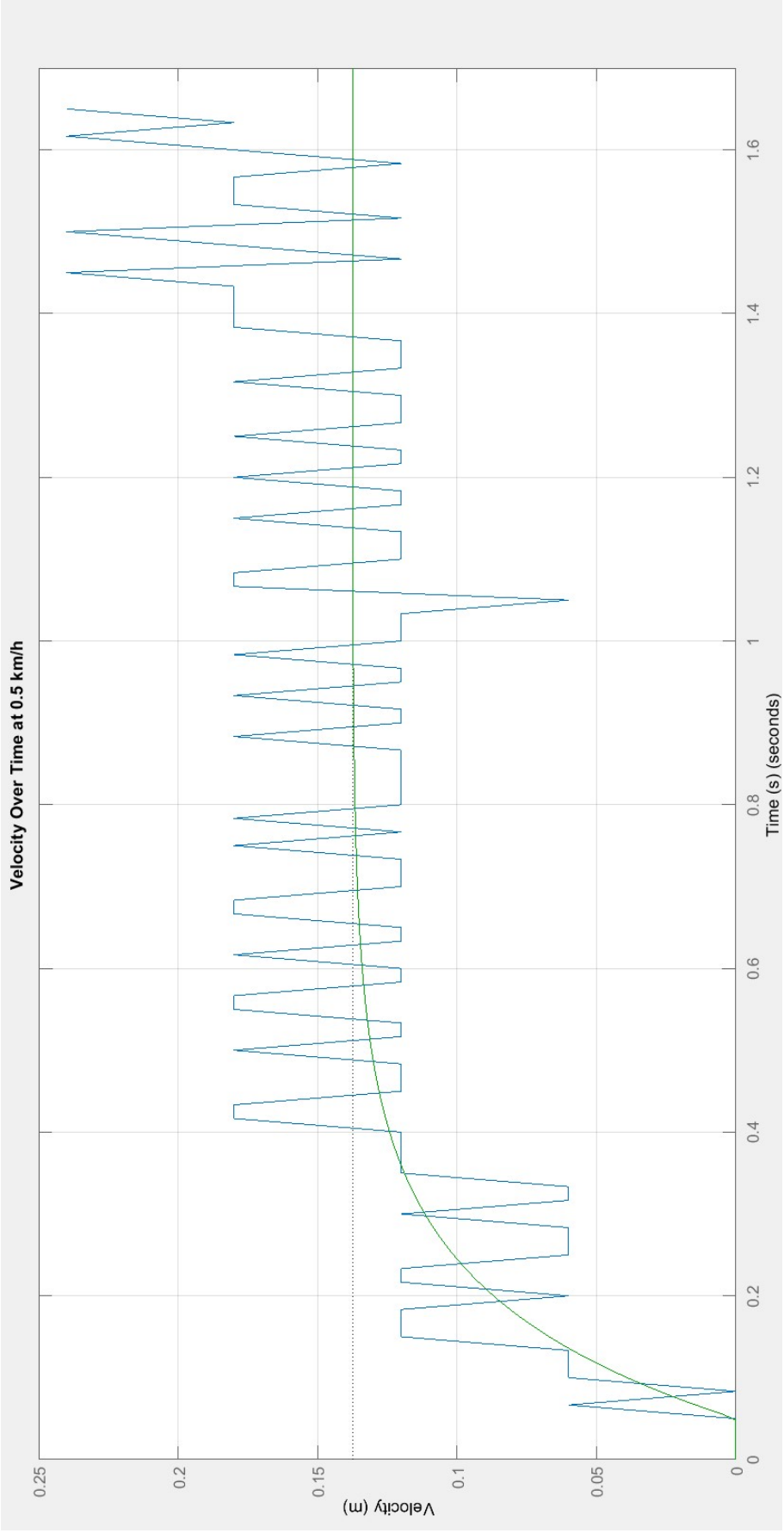


Figure B.23: Graph showing the relation between velocity and time for a step response equivalent to 0.7 km/h. The transfer function is shown in green.

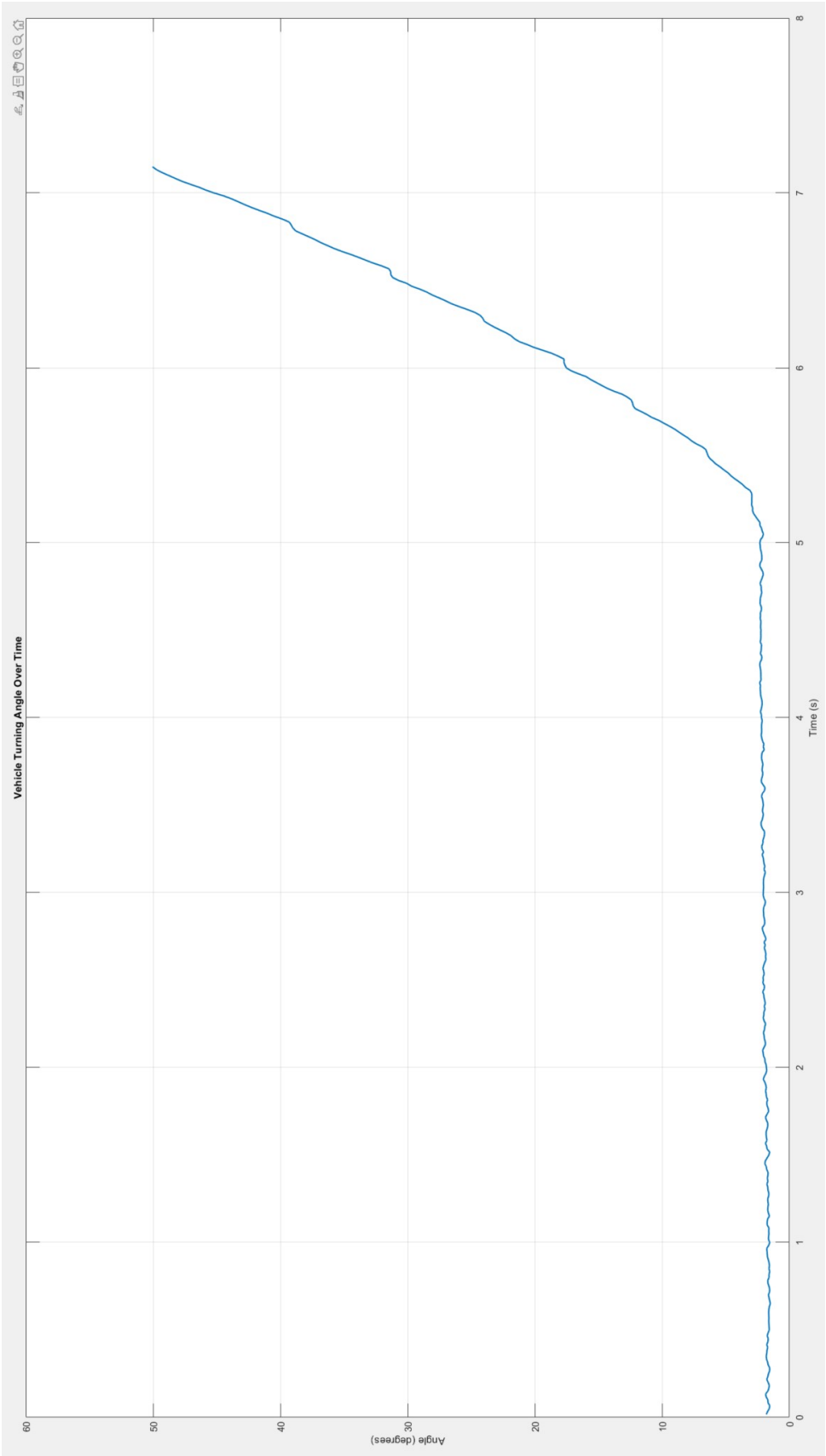


Figure B.24: Angle development/change over time when the vehicle is subjected to a speed difference of 1.3 km/h. Note that the first 5 seconds are driving straight ahead, as there is a 5-second delay, between pressing the button and the PWM being altered

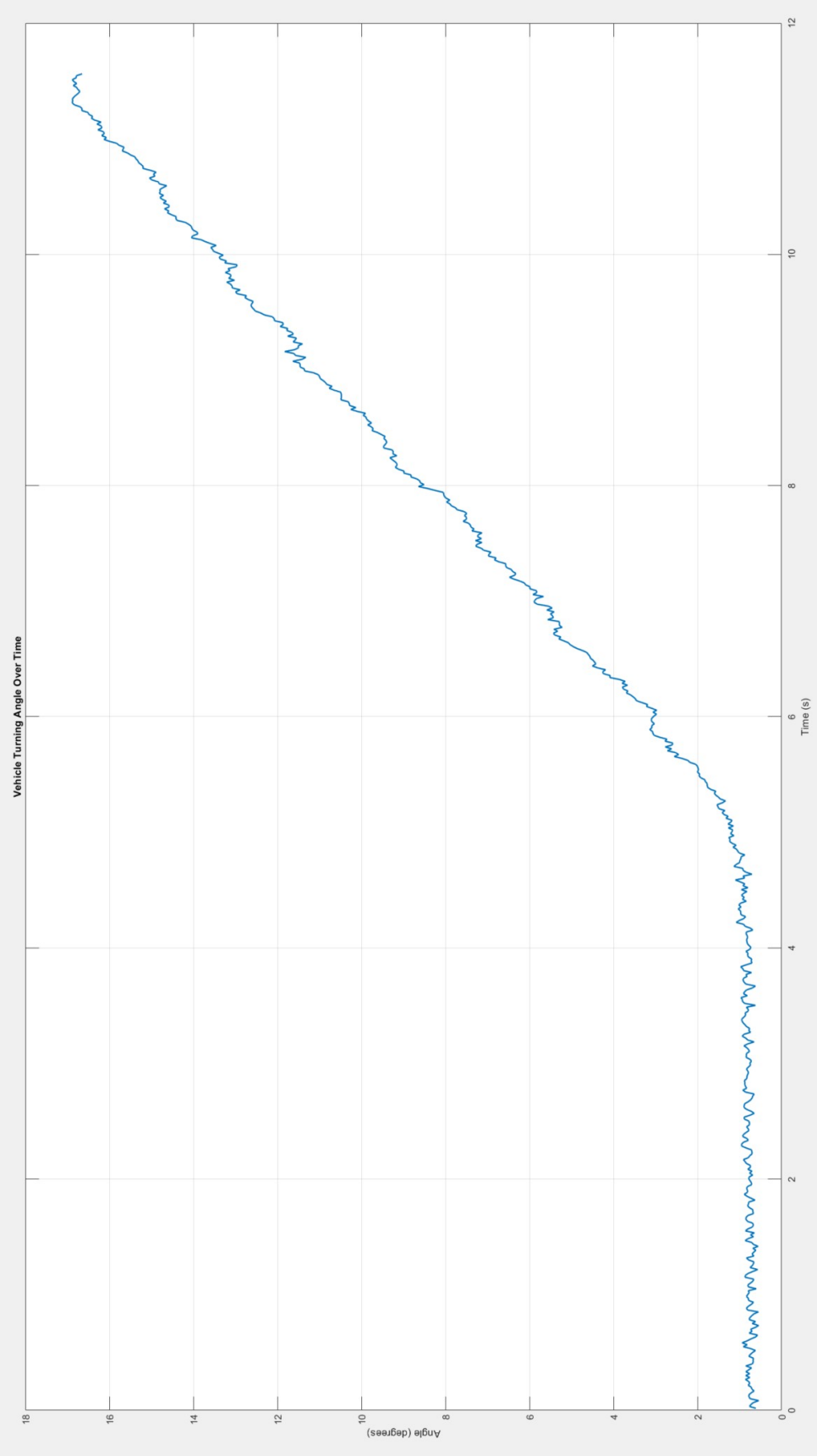


Figure B.25: Angle development/change over time when the vehicle is subjected to a speed difference of 0.7 km/h. Note that the first 5 seconds are driving straight ahead, as there is a 5-second delay, between pressing the button and the PWM being altered

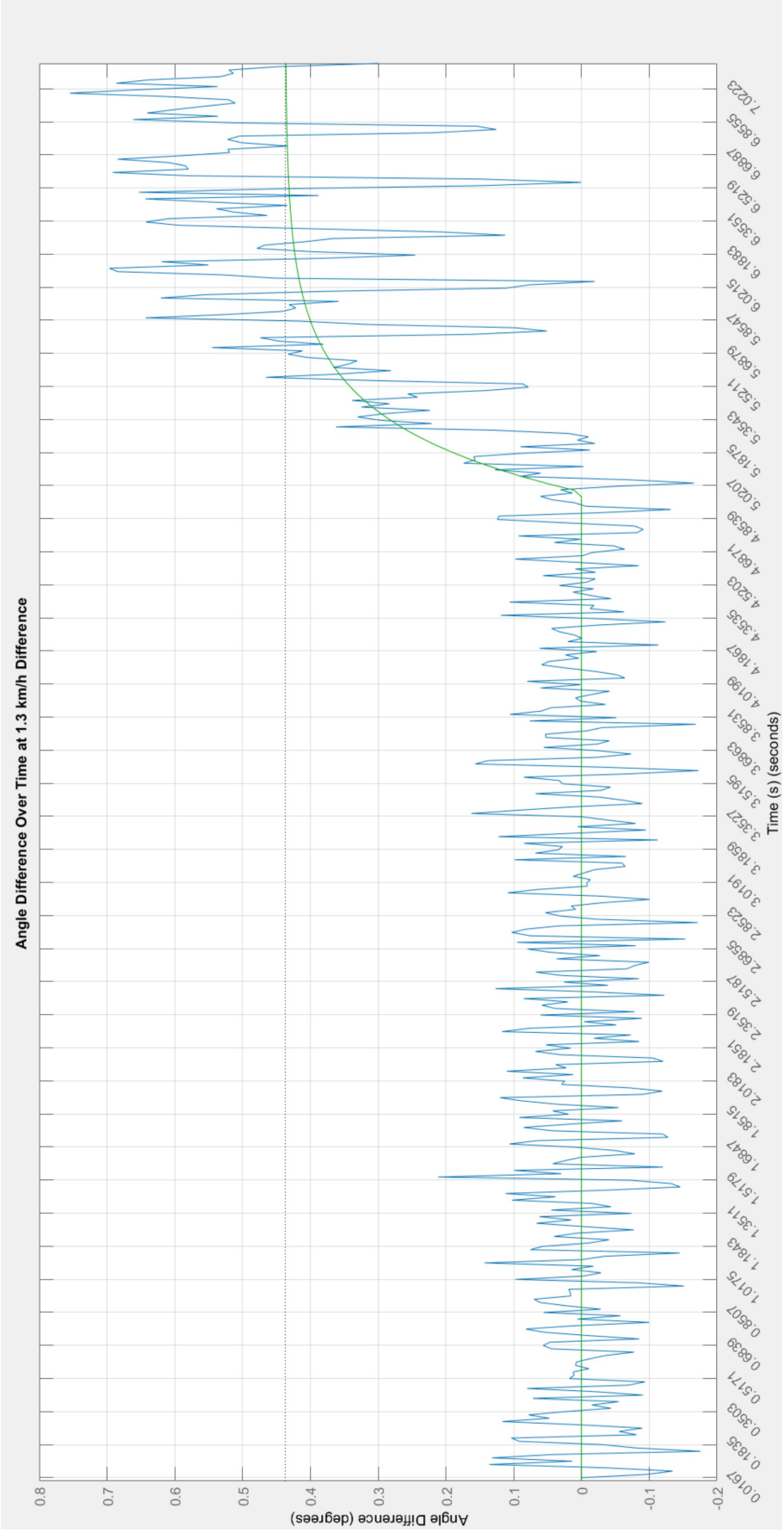


Figure B.26: Angle difference over time (blue) with estimated transfer function (green).

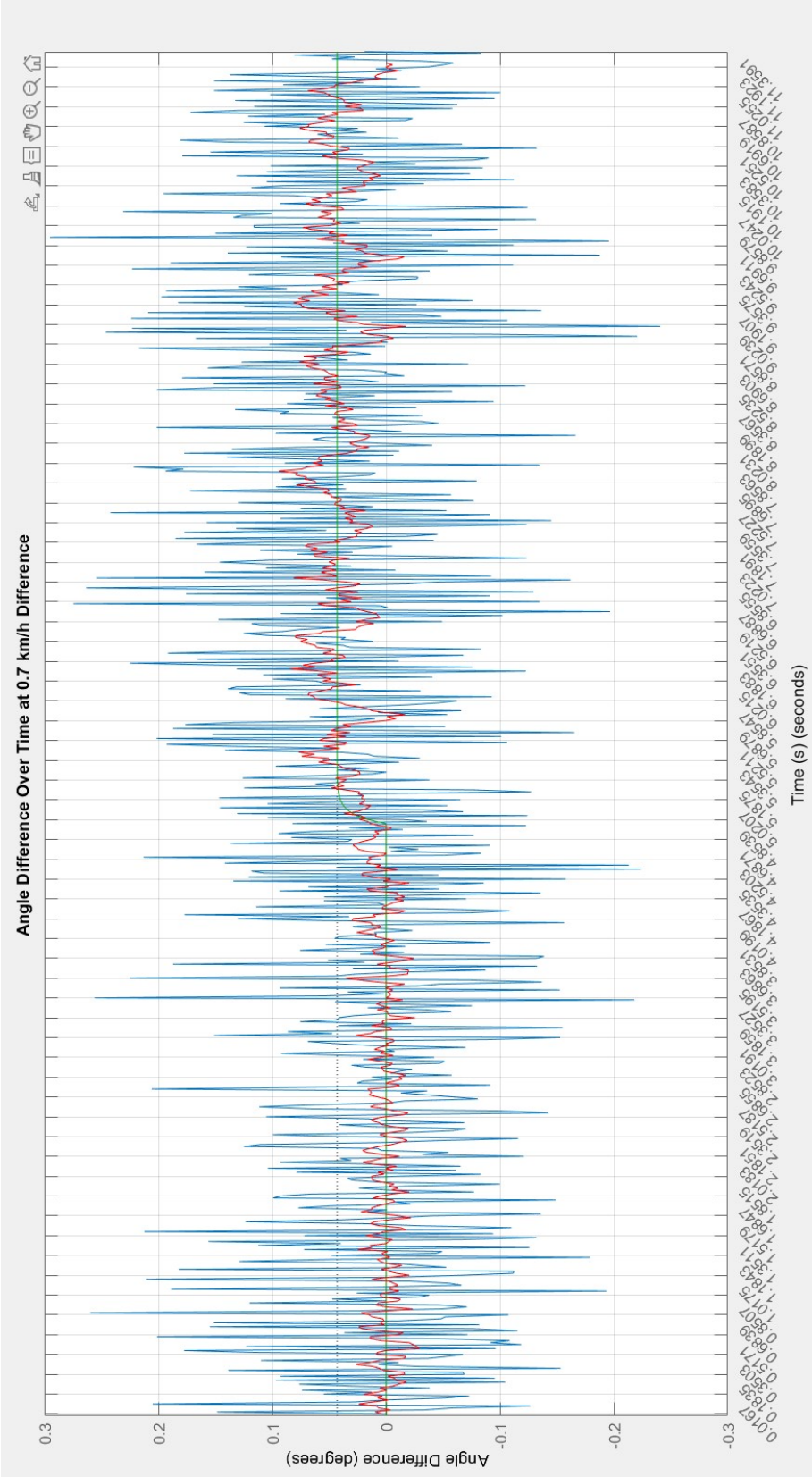


Figure B.27: Angle difference over time (blue) with estimated transfer function (green) and average angle differences across the next 10 samples (red).

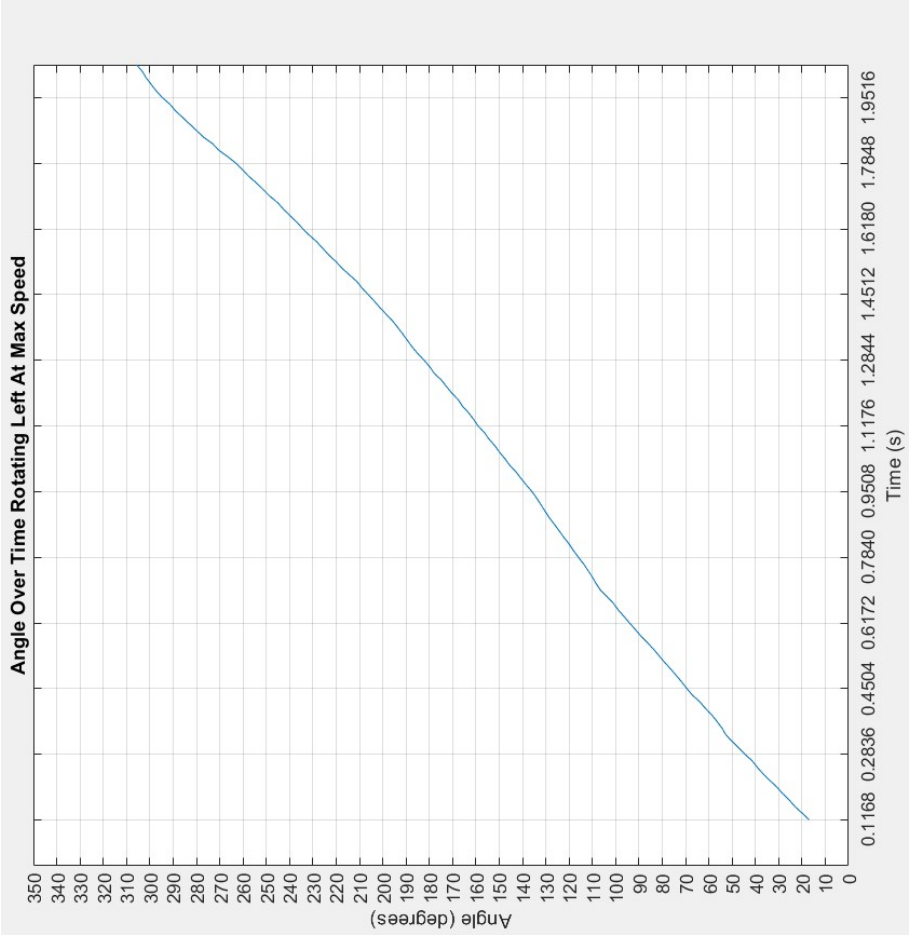


Figure B.28: As can be seen, the function is practically linear, witnessing maximum velocity of 153 degrees/second has been reached.

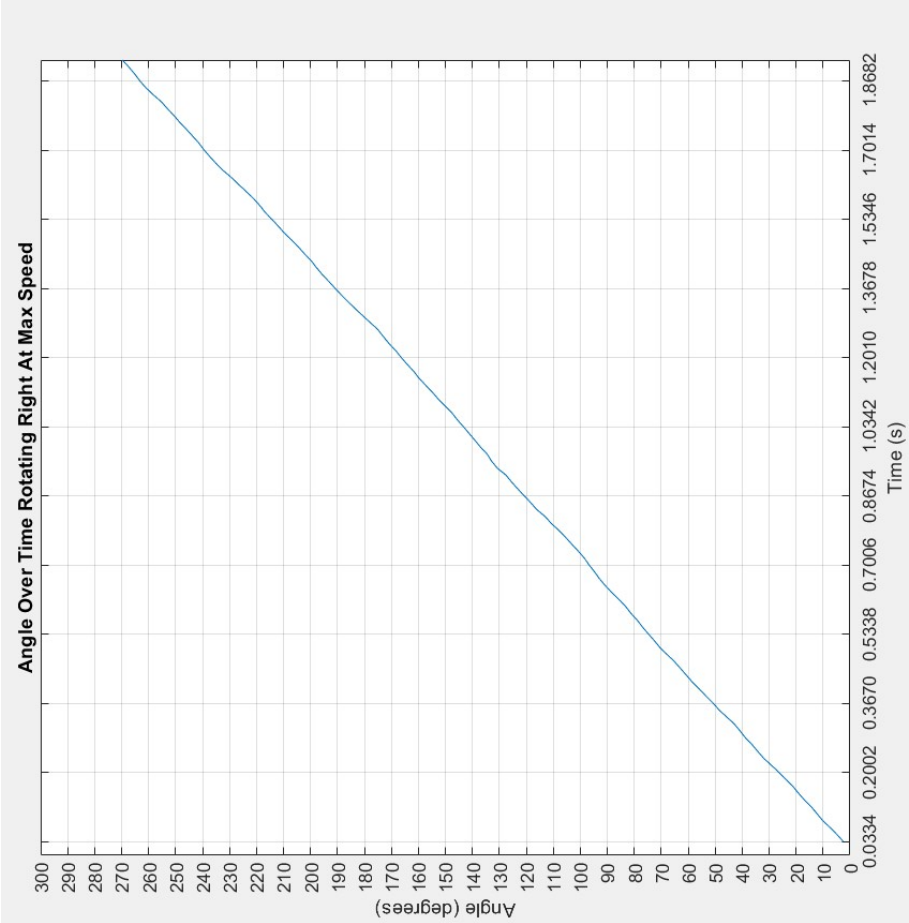


Figure B.29: As can be seen, the function is practically linear, witnessing maximum velocity of 143 degrees/second has been reached.

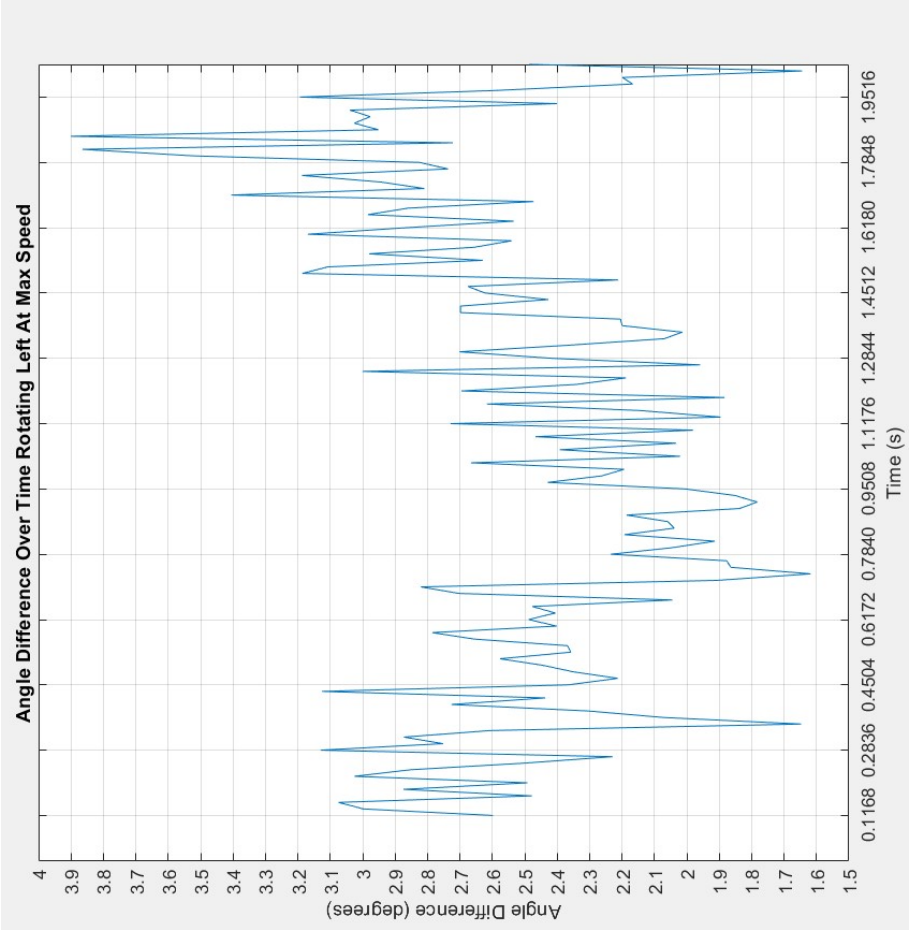


Figure B.30: While it may not look as such, an average of 2.54 degrees in change is made between each frame.

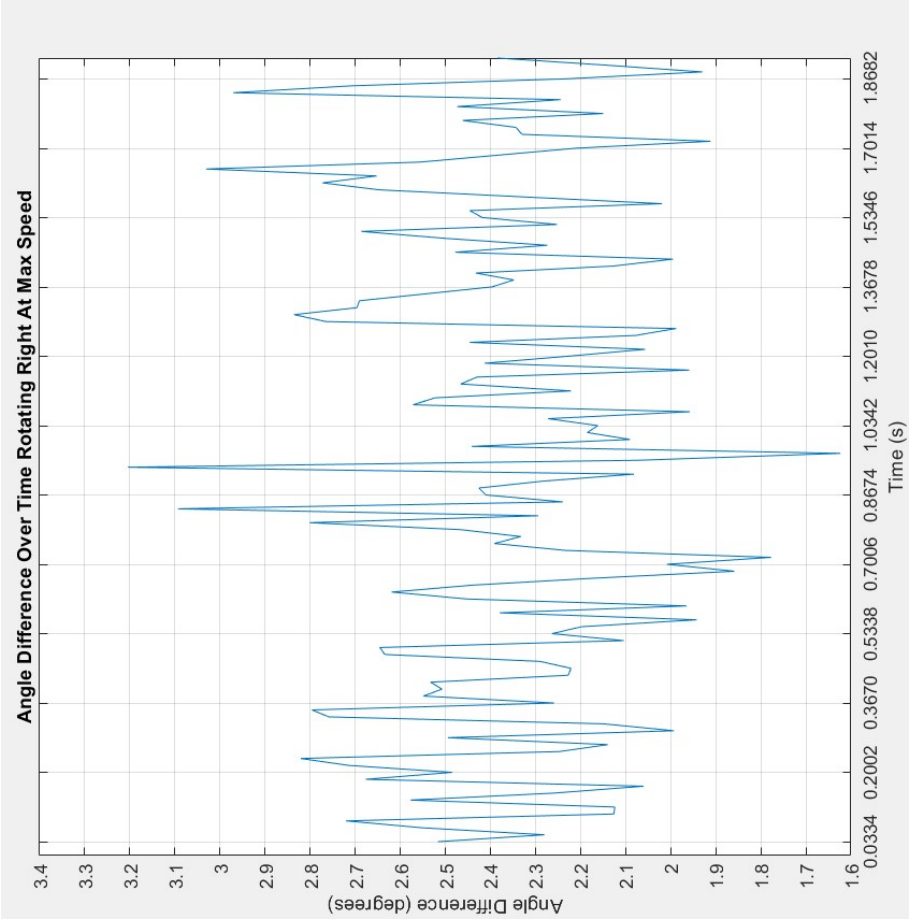


Figure B.31: While it may not look as such, an average of 2.38 degrees in change is made between each frame.

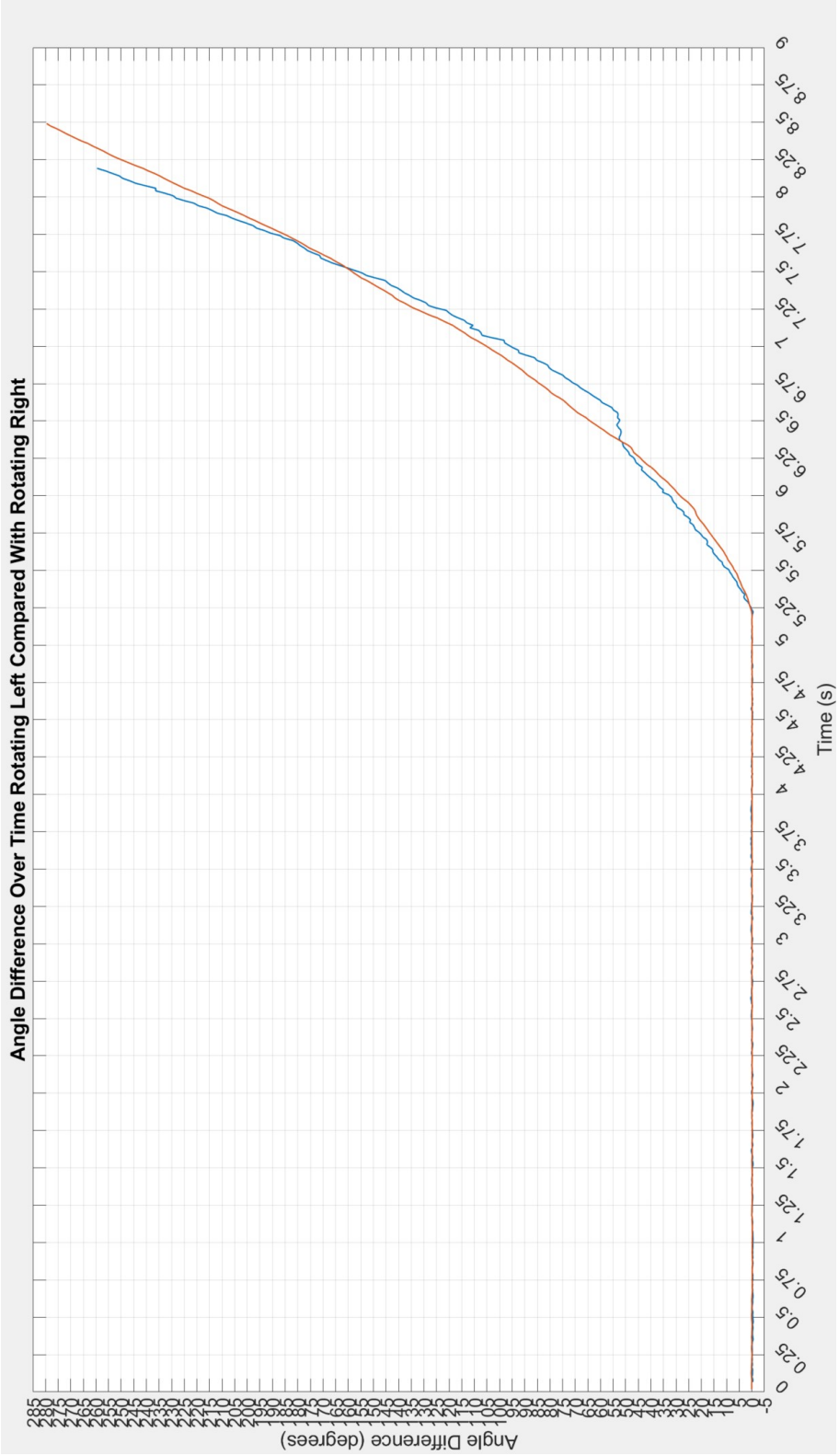


Figure B.32: Comparison of how the vehicle rotates in both directions. Orange is rotating right, and blue is rotating left.

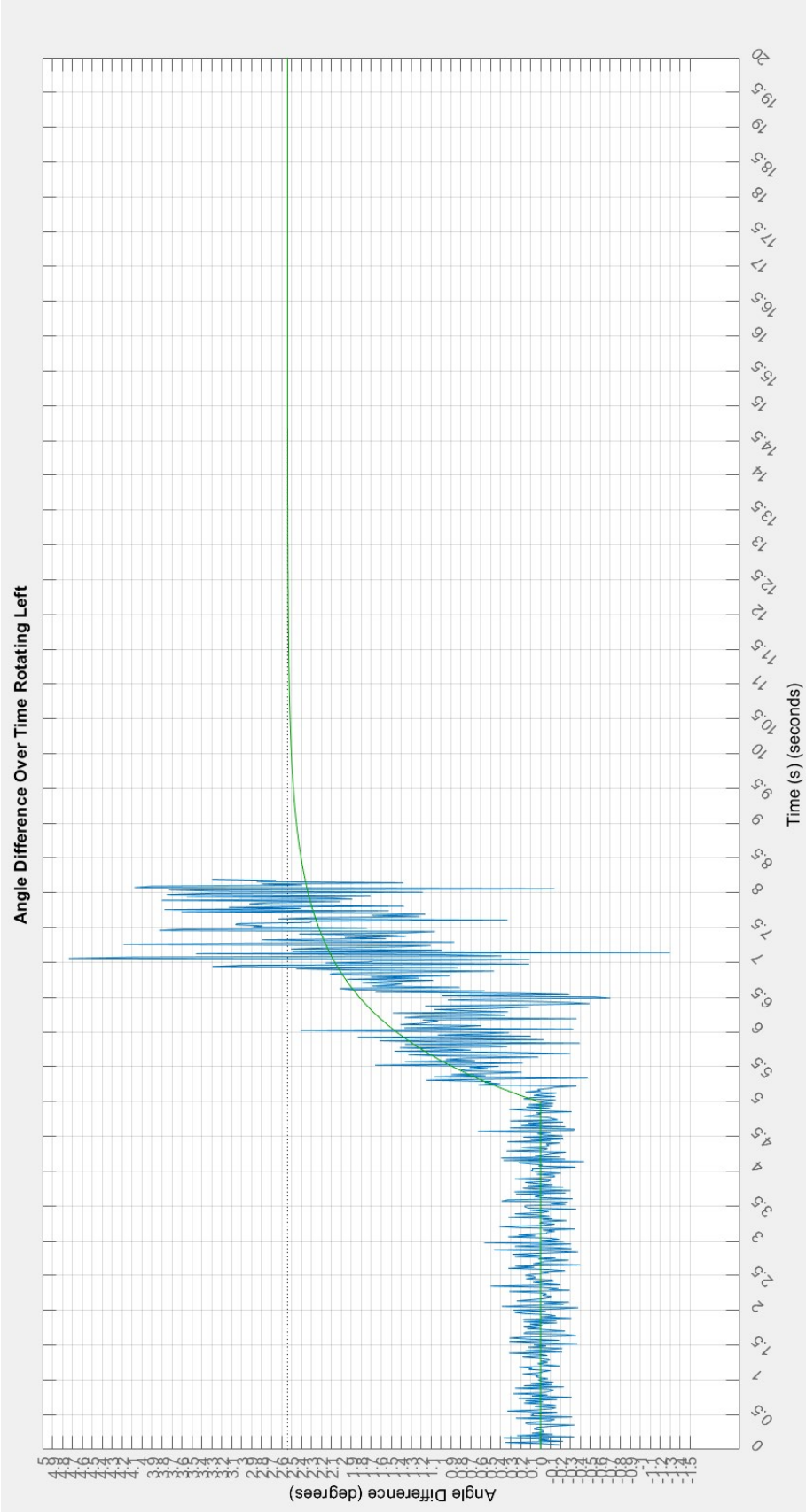


Figure B.33: Estimated transfer function of rotating left, with known maximum rotational velocity.

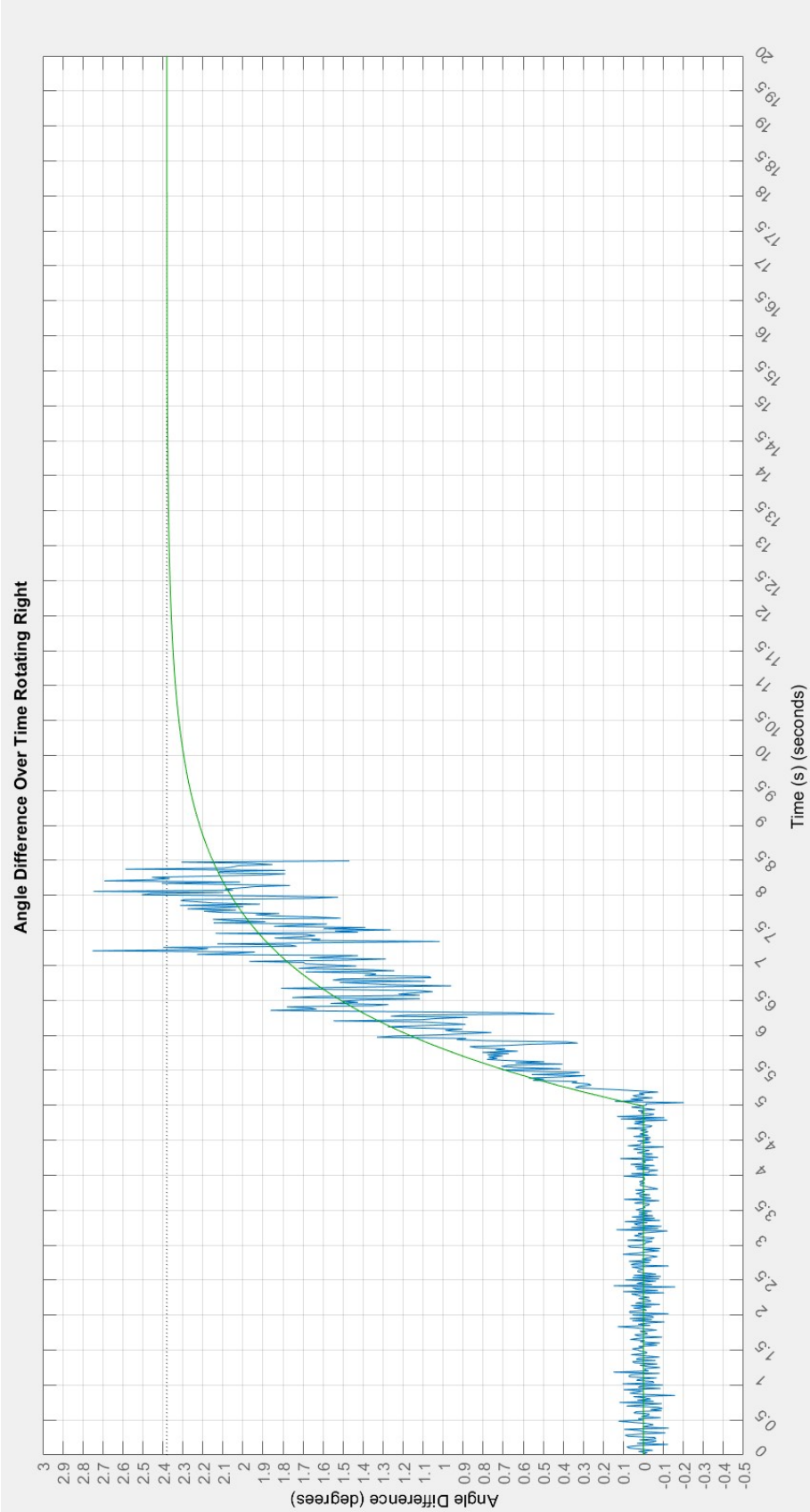


Figure B.34: Estimated transfer function of rotating right, with known maximum rotational velocity.

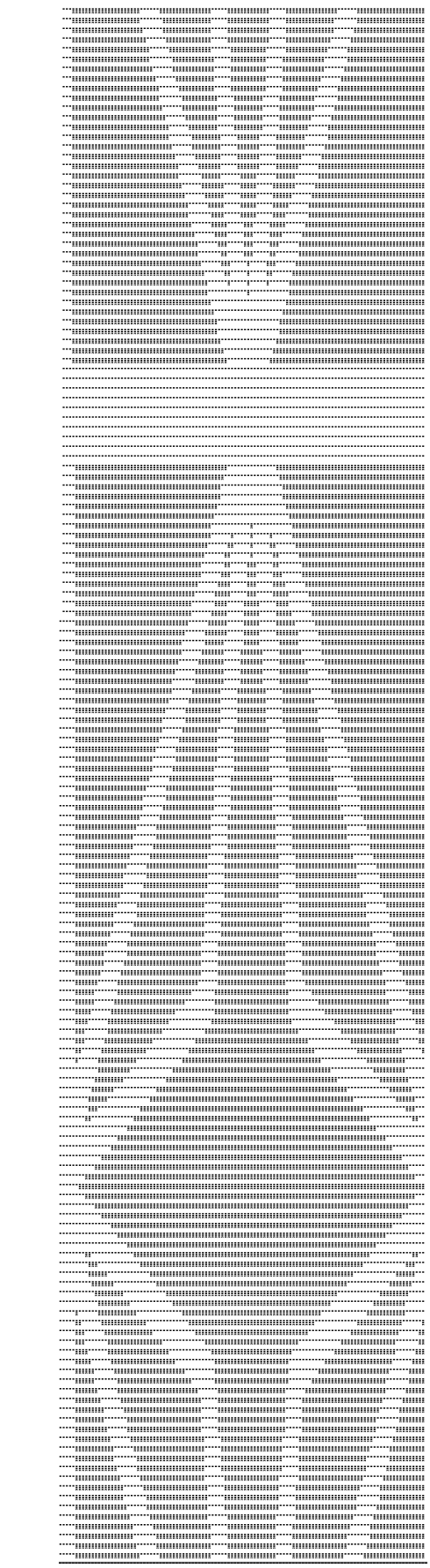


Figure B.35: The test image converted into a 2D array. Notice that the first 1/4 of the image is placed at the end.

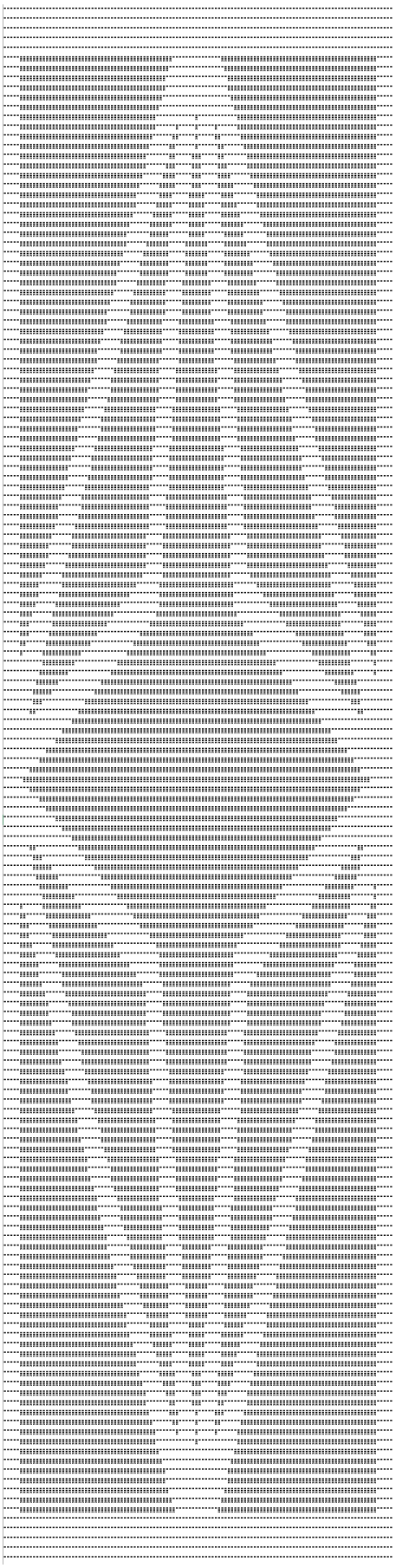


Figure B.36: The test image converted into a 2D array. This time with the correct interpretation.

B.6 VL53L0X Serial Outputs

```
15:57:39.917 -> Reading a measurement... Distance (mm): 119
15:57:39.949 -> Reading a measurement... Distance (mm): 117
15:57:39.983 -> Reading a measurement... Distance (mm): 116
15:57:40.030 -> Reading a measurement... Distance (mm): 117
15:57:40.074 -> Reading a measurement... Distance (mm): 117
15:57:40.116 -> Reading a measurement... Distance (mm): 117
15:57:40.150 -> Reading a measurement... Distance (mm): 117
15:57:40.195 -> Reading a measurement... Distance (mm): 118
15:57:40.235 -> Reading a measurement... Distance (mm): 118
15:57:40.268 -> Reading a measurement... Distance (mm): 116
```

(a) Serial monitor reading at 10 cm.

```
15:58:39.796 -> Reading a measurement... Distance (mm): 227
15:58:39.839 -> Reading a measurement... Distance (mm): 224
15:58:39.880 -> Reading a measurement... Distance (mm): 223
15:58:39.913 -> Reading a measurement... Distance (mm): 225
15:58:39.958 -> Reading a measurement... Distance (mm): 226
15:58:39.999 -> Reading a measurement... Distance (mm): 227
15:58:40.034 -> Reading a measurement... Distance (mm): 225
15:58:40.078 -> Reading a measurement... Distance (mm): 226
15:58:40.113 -> Reading a measurement... Distance (mm): 223
```

(b) Serial monitor reading at 20 cm.

```
15:59:17.588 -> Reading a measurement... Distance (mm): 323
15:59:17.629 -> Reading a measurement... Distance (mm): 319
15:59:17.672 -> Reading a measurement... Distance (mm): 318
15:59:17.712 -> Reading a measurement... Distance (mm): 322
15:59:17.752 -> Reading a measurement... Distance (mm): 316
15:59:17.791 -> Reading a measurement... Distance (mm): 318
15:59:17.824 -> Reading a measurement... Distance (mm): 321
15:59:17.857 -> Reading a measurement... Distance (mm): 321
15:59:17.890 -> Reading a measurement... Distance (mm): 320
15:59:17.931 -> Reading a measurement... Distance (mm): 315
```

(c) Serial monitor reading at 30 cm.

```
15:59:42.669 -> Reading a measurement... Distance (mm): 523
15:59:42.703 -> Reading a measurement... Distance (mm): 521
15:59:42.745 -> Reading a measurement... Distance (mm): 518
15:59:42.789 -> Reading a measurement... Distance (mm): 522
15:59:42.821 -> Reading a measurement... Distance (mm): 524
15:59:42.867 -> Reading a measurement... Distance (mm): 529
15:59:42.907 -> Reading a measurement... Distance (mm): 516
15:59:42.946 -> Reading a measurement... Distance (mm): 524
15:59:42.986 -> Reading a measurement... Distance (mm): 530
15:59:43.021 -> Reading a measurement... Distance (mm): 521
```

(d) Serial monitor reading at 10 cm.

```
16:01:32.111 -> Reading a measurement... Distance (mm): 736
16:01:32.151 -> Reading a measurement... Distance (mm): 727
16:01:32.183 -> Reading a measurement... Distance (mm): 737
16:01:32.225 -> Reading a measurement... Distance (mm): 727
16:01:32.270 -> Reading a measurement... Distance (mm): 740
16:01:32.309 -> Reading a measurement... Distance (mm): 748
16:01:32.344 -> Reading a measurement... Distance (mm): 740
16:01:32.389 -> Reading a measurement... Distance (mm): 737
16:01:32.427 -> Reading a measurement... Distance (mm): 745
```

(e) Serial monitor reading at 75 cm.

```
16:04:47.748 -> Reading a measurement... Distance (mm): 888
16:04:47.787 -> Reading a measurement... Distance (mm): 888
16:04:47.827 -> Reading a measurement... Distance (mm): 899
16:04:47.864 -> Reading a measurement... Distance (mm): 900
16:04:47.906 -> Reading a measurement... Distance (mm): 925
16:04:47.942 -> Reading a measurement... Distance (mm): 925
16:04:47.985 -> Reading a measurement... Distance (mm): 897
16:04:48.025 -> Reading a measurement... Distance (mm): 911
16:04:48.061 -> Reading a measurement... Distance (mm): 888
16:04:48.093 -> Reading a measurement... Distance (mm): 927
```

(f) Serial monitor reading at 100 cm.

```
16:06:57.037 -> Reading a measurement... Distance (mm): 1556
16:06:57.082 -> Reading a measurement... Distance (mm): 1540
16:06:57.122 -> Reading a measurement... Distance (mm): 1528
16:06:57.163 -> Reading a measurement... Distance (mm): 1502
16:06:57.203 -> Reading a measurement... Distance (mm): 1545
16:06:57.242 -> Reading a measurement... Distance (mm): 1530
16:06:57.275 -> Reading a measurement... Distance (mm): 1535
16:06:57.308 -> Reading a measurement... Distance (mm): 1568
16:06:57.341 -> Reading a measurement... Distance (mm): 1578
16:06:57.385 -> Reading a measurement... Distance (mm): 1557
```

(g) Serial monitor reading at 150 cm.

```
16:08:33.810 -> Reading a measurement... Distance (mm): 1972
16:08:33.854 -> Reading a measurement... Distance (mm): 2063
16:08:33.887 -> Reading a measurement... Distance (mm): 2024
16:08:33.921 -> Reading a measurement... Distance (mm): 2002
16:08:33.964 -> Reading a measurement... Distance (mm): 2041
16:08:34.008 -> Reading a measurement... Distance (mm): 1965
16:08:34.054 -> Reading a measurement... Distance (mm): 1965
16:08:34.100 -> Reading a measurement... Distance (mm): 2025
16:08:34.135 -> Reading a measurement... Distance (mm): 2033
16:08:34.179 -> Reading a measurement... Distance (mm): 1960
```

(h) Serial monitor reading at 200 cm.

```
16:09:29.293 -> Reading a measurement...  out of range
16:09:29.336 -> Reading a measurement... Distance (mm): 2197
16:09:29.377 -> Reading a measurement... Distance (mm): 2230
16:09:29.417 -> Reading a measurement... Distance (mm): 2208
16:09:29.456 -> Reading a measurement... Distance (mm): 2197
16:09:29.496 -> Reading a measurement... Distance (mm): 2186
16:09:29.531 -> Reading a measurement... Distance (mm): 2201
16:09:29.573 -> Reading a measurement... Distance (mm): 2236
16:09:29.615 -> Reading a measurement... Distance (mm): 2176
16:09:29.650 -> Reading a measurement...  out of range
```

(i) Serial monitor reading at 220 cm.

Figure B.37: The collected height sensor readings.

B.7 High Level Specification - Full System

This section describes a high-level overview of the functionality desired for the pavement cleansing robot. It is written as seen by a private person wanting to ease cleaning their pavement. Detailed specification can be found in section B.8 starting page 144.
As a house owner looking to ease removing weeds from my pavement, I want:

- *An autonomous robot capable of removing dandelions, moss, grass, and other common weeds found in the pavement.*
- *An autonomous robot capable of moving around.*
- *A charge point/home at which the robot can dock when not in use/when it has to charge.*
- *An autonomous robot that updates me of its whereabouts and I can call "home" in case it is in my way.*
- *An autonomous robot capable of climbing my driveway (37%/20°rise).*
- *An autonomous robot that returns "home" before the battery dies.*
- *A fully autonomous robot, meaning that I will only need to do one setup procedure, and then it will work forever.*
- *An autonomous robot only removing weeds within my land.*
- *An autonomous robot that can distinguish between pavement styles, and act accordingly.*
- *An autonomous robot that systematically cleans my driveway and other pavement, and therefore does not just bump around like a Roomba.*
- *An autonomous robot capable of passing an IP56 test.*
- *An autonomous robot that can detect when a puddle is nearby, and go around it.*
- *An autonomous robot that detects cars, outdoor furniture, and the like, so that it will only operate around objects where it is safe.*
- *An autonomous robot that adjusts its pattern, based both on the weather and on previous passes. I.e. if there were a lot of weeds on the previous pass around the driveway, it would schedule a new pass earlier.*
- *An autonomous robot capable of mapping what parts of the driveway that has been cleaned.*
- *An autonomous robot avoiding obstacles without bumping into them.*
- *An autonomous robot making as little noise as possible.*
- *An autonomous robot as small in size as possible, ideally no larger than a Roomba (approximately 45cm), [23].*
- *An autonomous robot that does not get stuck at random places around my land; if it does get stuck, I want it to attempt to get unstuck.*
- *An autonomous robot capable of cleaning a minimum of 500m^2.*

B.8 Functional Specification - Full System

This section describes the functional criteria of the product. The criteria are seen from an end-user perspective and made as user stories, where accept criteria (AC1 & AC2 e.g.) must be fulfilled. A test of the functional specifications are made in section 7.

B.8.1 Weed Removal

As a house owner, I want an autonomous robot capable of removing dandelions, moss, groundsel, grass, and other common weeds found in the pavement.
Accept Criteria:
AC1:
The robot should be capable of burning off several types of weeds, including but not limited to: dandelion, grass, groundsel, moss, thistle, cleavers, and horsetail.
AC2:
The robot should be capable of only burning the necessary intensity to stress the plants, without setting them aflame.
AC3:
All plants should perish within a season of continuous burning.
AC4:
The robot should be able to distinguish between weeds and non-organic items, such as a gardenhose.

B.8.2 Operate the Robot

As a house owner, I want an autonomous robot capable of moving around.
Accept Criteria:
AC1:
The robot should be able to drive forward.
AC2:
The robot should be able to drive backward.
AC3:
The robot should be able to turn around its own axis (Z-axis).

B.8.3 A Home for the Robot

As a house owner, I want a charge point/home at which the robot can dock when not in use/when it has to charge.
Accept Criteria:
AC1:
The charge point should be able to contain the robot.
AC2:
The robot should be able to charge when not in use.
AC3:
The robot should be able to communicate with the charge point.

B.8.4 User-interface

As a house owner, I want an autonomous robot that updates me of its whereabouts and I can call "home" in case it is in my way.

Accept Criteria:
AC1:
At all times the robot should broadcast its whereabouts.
AC2:
If the robot is in the way, a user-interface should enable me to send it "home" or to another area. **AC3:** The interface should enable me to take control of the robot, effectively making it remote-controlled.

B.8.5 Go Anywhere

As a house owner, I want an autonomous robot capable of climbing my driveway (20% rise).
Accept Criteria:
AC1:
The robot should be able to climb a 20% rise.
AC2:
The robot should be capable of traversing poorly laid pavement (up to 30mm height difference between tiles).
AC3:
The robot should be capable of traversing small obstacles, such as a garden hose.

B.8.6 Robot Go Home

As a house owner, I want an autonomous robot that returns ”home” before the battery dies.
Accept Criteria:
AC1:
At all times enough battery power is left to drive "home" to a charging point + 10% extra distance, meaning that a 20-meter travel home, requires power for at least 22 meters.
AC2:
The robot should be capable of mapping a route "home" with obstacles such as corners, cars, and lawnchairs added, to accommodate non-direct routes.

B.8.7 Easy Setup

As a house owner, I want a fully autonomous robot, meaning that I will only need to do one setup procedure, and then it will work forever.
Accept Criteria:
AC1:
The robot has to be capable of adjusting its trajectory and routing to changing environments, as long as the outer bounds do not change.
AC2:
The robot "home" has to be connectable to a standard wall socket.
AC3:
The robot has to auto-charge between passes.
AC4:
The robot should be updateable and automatically update at convenient times.
AC5:
The robot should be easy to connect to secondary devices, such as phones, routers, etc..

B.8.8 Stay On My Turf

As a house owner, I want an autonomous robot only removing weeds within my land.
Accept Criteria:
AC1:
The robot should not wander off from assigned bounds.
AC2:
The robot has to stay within its registered land.
AC3:
If placed off of its registered land, it should first ping its "home" to plan a possible route home, or simply send a message to the owner before turning off.

B.8.9 Pattern Recognition

As a house owner, I want an autonomous robot that can distinguish between pavement styles, and act accordingly.
Accept Criteria:
AC1:
The robot should be capable of recognizing the following (danish) pavement styles: herregårdssten (14x21cm), modul fliser (30x30cm, 30x60cm, 60x60cm, 40x40cm, 50x50cm, 25x50cm, 25x25cm, 20x20cm, 20x40cm, 15x30cm, 15x15cm), soldaterfliser (60x90cm, 90x90cm, 60x120cm, 90x120, 30x90cm, 45x90cm), sekskantede fliser (32x32cm), chaussesten (9x9cm) and SF-sten (10.5x19cm).
AC2:
The robot has to adjust its route based on which pavement style it is currently cleaning.

B.8.10 Systematical Procedure

As a house owner, I want an autonomous robot that systematically cleans my driveway and other pavement, and therefore does not just bump around like a Roomba.
Accept Criteria:
AC1:
The robot has to map out all paved areas.
AC2:
A systematic approach following lines in the pavement has to be used.
AC3:
If several types of pavement are present, different areas must be mapped to distinguish and optimize routes for each area.

B.8.11 Weather Endurant

As a house owner, I want an autonomous robot capable of passing an IP56 test.
Accept Criteria:
AC1:
The shell of the robot has to pass an IP56 test.
AC2:
The charging/home point has to pass an IP56 test.
AC3:
The robot should notify the owner if temperatures become low enough to damage the battery or other electronics.

AC4:
The robot should be able to drive "home" in severe rainfall i.e. more than 30mm over an hour.
AC3:
The robot should be capable of operation in sub-zero temperatures[1].

B.8.12 Water Avoidance

As a house owner, I want an autonomous robot that can detect when a puddle is nearby, and go around it.
Accept Criteria:
AC1:
The robot must recognize puddles, pits, and other flooded areas.
AC2:
The robot must reroute to navigate around water.
AC3:
If a reroute is unavailable, the robot must turn around and follow its previous route back to "home".

B.8.13 Spatial Awareness

As a house owner, I want an autonomous robot that detects cars, outdoor furniture, and the like, so that it will only operate around objects where it is safe.
Accept Criteria:
AC1:
The robot must be capable of determining whether or not, it can fit within a space.
AC2:
The robot must keep a minimum clearance of 10cm to anything above, so as to not wedge itself beneath anything.

B.8.14 Self-adjusting Scheduling

As a house owner, I want an autonomous robot adjusts its pattern, based both on the weather and on previous passes. I.e. if there were a lot of weeds on the previous pass around the driveway, it would schedule a new pass earlier.
Accept Criteria:
AC1:
The robot must map out every time a plant is hit, and keep that map in memory for the next 5 cycles.
AC2:
The robot must schedule passes based upon the amount of remaining greenery in an area.
AC3:
The robot must be capable of accessing weather information, so as to not plan its next pass when rain is predicted.
AC4:
The robot must be capable of intensifying passes if the greenery percentage does not go down in an area.

[1] In this instance, longevity of the battery is de-prioritized.

AC5:
The robot must be capable of adjusting its pattern to only focus on areas previously containing weeds. I.e. scheduling every second or third pass to only go for coordinates with "known" weeds, rather than traversing the full area.

B.8.15 Memory Capabilities

As a house owner, I want an autonomous robot capable of remembering what parts of the driveway have been cleaned.
Accept Criteria:
AC1:
The robot must map out which areas have been cleaned at which point, to rotate between areas.
AC2:
The robot must be capable of increasing its speed when no greenery is present.

B.8.16 Obstacle Avoidance

As a house owner, I want an autonomous robot avoiding obstacles without bumping into them.
Accept Criteria:
AC1:
The robot must not hit anything to change its course.
AC2:
The robot must not be closer than 5cm to anything in any direction, other than the pavement below it.
AC3:
The robot must not drive off of a ledge and tumble down.
AC4:
If the robot cannot turn around its own axis, it must reverse out from its current spot.
AC5:
If presented in a corner with no way out, the robot must turn off and notify the owner.

B.8.17 Low Noise

An autonomous robot making as little noise as possible.
Accept Criteria:
AC1:
The robot must not make any more noise than comparable lawn mowing robots, i.e. 58dB for a Texas TMX1000, [47].
AC2:
Under extraordinary circumstances, such as traversing small obstacles, volume may be increased to 65dB.
AC3:
Cooling of the robot must not surpass 55dB.

B.8.18 Small Size

An autonomous robot as small in size as possible, ideally no larger than a Roomba (approximately 45cm), [23].
Accept Criteria:

AC1:
The length and width of the robot must be smaller than 45cm.
AC2:
The height of the robot must be lower than 12cm.

B.8.19 Selfrecovery

An autonomous robot that does not get stuck at random places around my land; if it does get stuck, I want it to attempt to get unstuck.
Accept Criteria:
AC1:
If the cameras do not change pictures for 25 cycles, the robot should recognize it is stuck.
AC2:
If stuck and no obstacles are nearby, the robot should try to first reverse, then turn clockwise, then counterclockwise - all operations must be made at half speed to minimize the risk of loosing grip due to speed.
AC3:
If the robot cannot get unstuck on its own, it should notify the owner and turn off.

B.8.20 Area

An autonomous robot capable of cleaning a minimum of 500m^2.
Accept Criteria:
AC1:
By cleaning 500m^2, it is meant as mapping out 500m^2 and cleaning continuously, rather than in one pass.
AC2:
The robot should be capable of prioritizing some areas over others, so a customer can prioritize the terrace higher than the driveway i.e..

B.9 Limitating the Project - Full System

As this project only stretches for a single semester and is being done by a single student, some limitations have to be made. The project will instead of focusing on developing a full system meeting all demand specifications from the start, be divided into iterations. Therefore each iteration will have its own distinct goal(s) and specifications that it should meet. The first iteration will be the minimum viable product (MVP), and further iterations will contain increasingly advanced features.

B.9.1 1st Iteration - Minimum Viable Product/Focus of This Project

The minimum viable product is a version, which only really is an autonomous robot driving around pavement. The 1st iteration will therefore be based on a tracked vehicle available from AAU, and will mostly regard the steering and operation of an autonomous robot, rather than actual weed-removing capabilities. The goals for the first iteration are oriented at being able to follow a line in the pavement, drive around on flat pavement, and avoid hitting objects.
Functional specifications to meet:

- B.8.2 - Operate the Robot
- B.8.3 - A Home for the Robot
- B.8.4 - User-interface
- B.8.6 - Robot Go Home
- B.8.9 - Pattern Recognition, AC2
- B.8.12 - Water Avoidance
- B.8.13 - Spatial Awareness
- B.8.15 - Memory Capabilities
- B.8.16 - Obstacle Avoidance

B.9.2 2nd Iteration

The 2nd iteration will be the first to include actual weed-removing capabilities. At this iteration, a laser[2] and its power source will be added. Furthermore, image processing and recognition will be integrated, enabling the laser to be aimed correctly. On a software-level, systematical procedures for optimising routes and pattern recognition for more advanced kinds of pavement will be added.

Functional specifications to meet:

- B.8.1 - Weed Removal
- B.8.9 - Pattern Recognition, AC1
- B.8.10 - Systematical Procedure

[2]To stay within budget, the first laser will most likely be a laser-pointer, rather than an actual laser capable of burning weeds.

B.9.3 3rd Iteration

At its 3rd iteration, the autonomous robot will be configurable to stay within a designated area, ensuring the robot does not wander off into the wild. Furthermore, the setup/installation procedure should be made easier for the common man/woman to do. A self-adjusting schedule will also be integrated so that the robot optimizes its routing and number of passes to correspond with greenery growth, weather, and personalized preferences.

Functional specifications to meet:

- B.8.5 - Go Anywhere
- B.8.7 - Easy Setup
- B.8.8 - Stay On My Turf
- B.8.14 - Self-adjusting Scheduling
- B.8.19 - Selfrecovery
- B.8.20 - Area

B.9.4 4th Iteration

The 4th iteration will be the first to require a new vehicle, as this iteration will focus on making the robot more comfortable to be around, while also making it weather endurant and capable of being outside for extended periods of time.

Functional specifications to meet:

- B.8.11 - Weather Endurant
- B.8.17 - Low Noise
- B.8.18 - Small Size

B.9.5 5th Iteration

When a 5th iteration is developed, it will focus more on easing use for the consumer by implementing an app, where the user can set up prioritizing for different areas, divide their land into zones, and generally customize the operation of the robot. This iteration will have no functional specifications, as these are not designed within the scope of this project.

List of Figures

List of Tables