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: 76

Date of Completion:
October 28, 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 67 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 A.2 and A.3. 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 A.2 on page 68 along with a functional specification in appendix A.3 on page 69.

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).

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 charge when not in use.
AC3:
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).
- Laser Power Supply: 5-24V (assuming a low-power solid-state laser for weeding, power requirements will depend on the specific laser selected).
- 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: 5V/3.3V DC-DC converters to manage voltage distribution efficiently across different subsystems.

These specifications are informed by reference data from "Dr Robot's" manual for the Jaguar Lite robot, which the prototype is modeled after, [21]. Similar robotic platforms such as autonomous lawnmowers and vacuum cleaners generally operate at 12V with battery capacities ranging from 2.8-8.8Ah, [19, 22]. 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, [23]. 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, [23].

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.

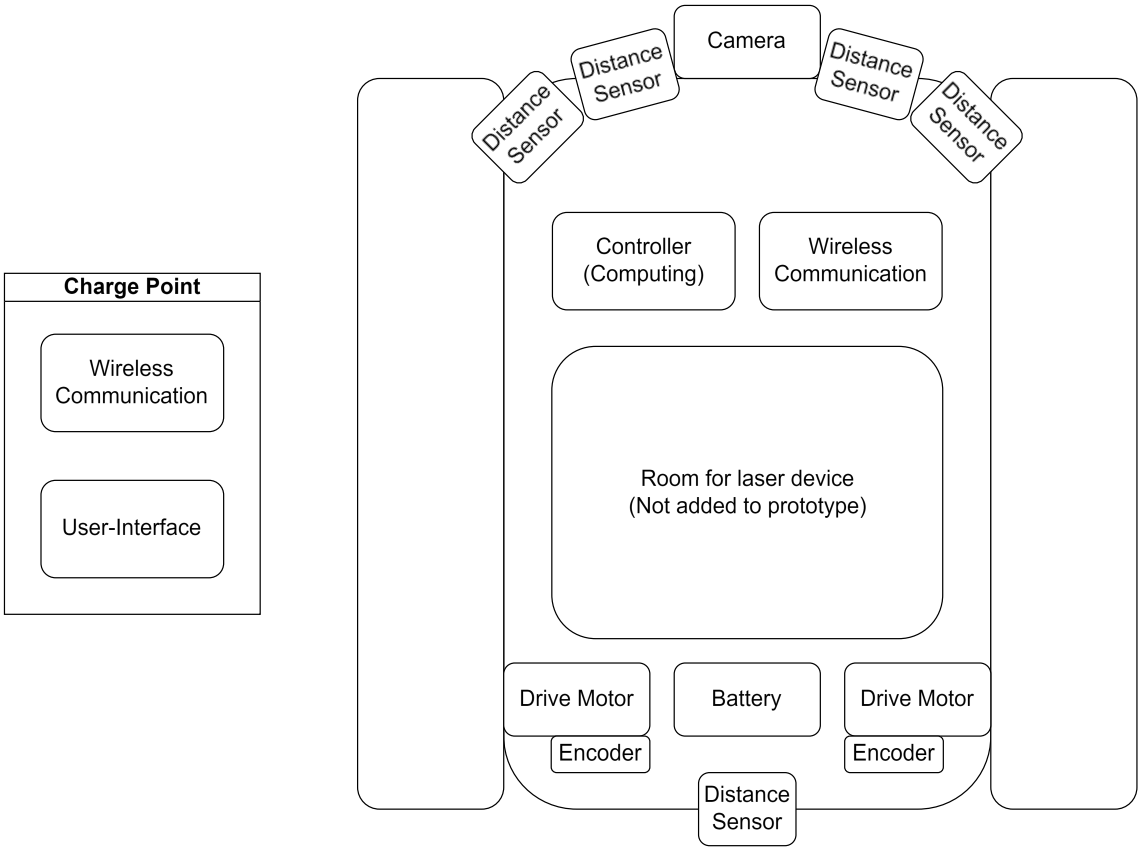


Figure 4.1: The preliminary solution, with the bare minimum of sensors/elements present. The camera will act as the robot's primary vision, while distance sensors will act as backup.

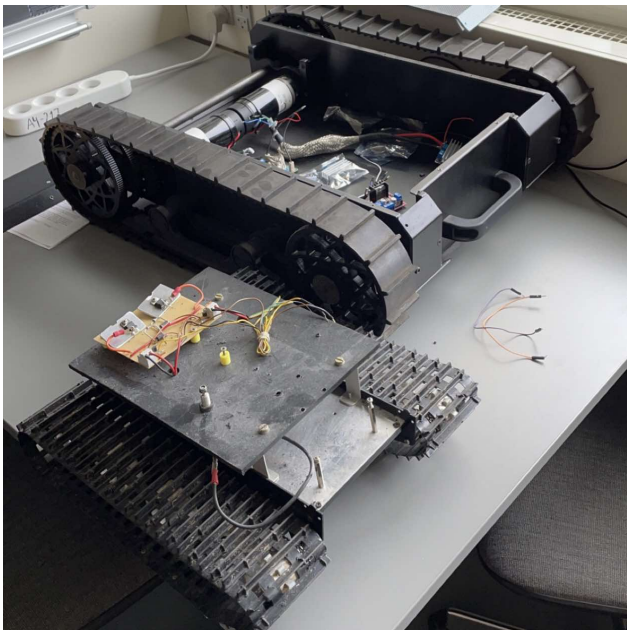


Figure 4.2: The physical tracked platform, which the system will be built upon.

In figure 4.2 the available tracked vehicle which the system will be built upon is shown. As this is only a prototype, the physical elements are not as important, and will therefore not be described in great detail, other than it is a tracked vehicle.

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.

4.2 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. Furthermore, this section will incorporate knowledge for creating stable and smooth systems. 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 in order 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.

A good basis for autonomy is made with the above 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 25. The above lists are derived from the first chapter of [23].

[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.

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, and the reasoning behind can be found in section 5.10.2 starting page 54. 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.

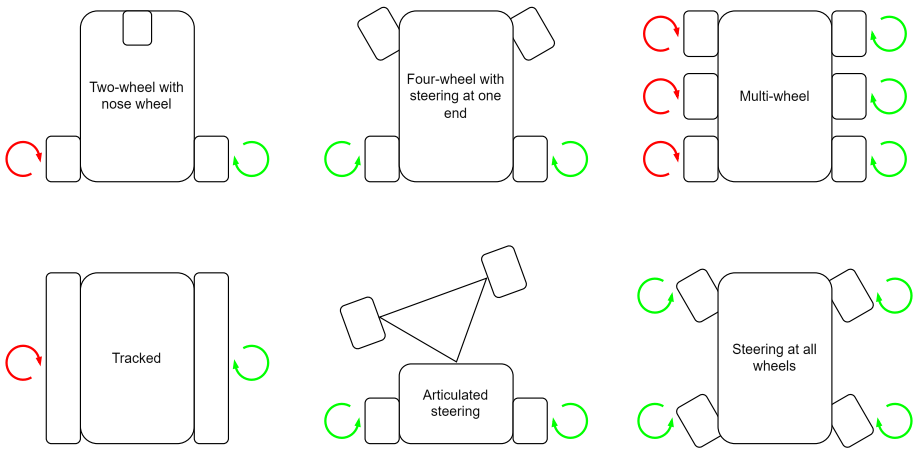


Figure 4.3: The six most common steering topologies used for vehicular robots. Arrows signify driving wheels, with green arrows signifying a forward movement and red signifying a backward movement. All six topologies are steering left in this drawing.

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 differently to the other set. The only true outlier is robots with the capability of steering at all wheels. In figure 4.3 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 4.3, 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. In figure 4.4 different operations on a joystick are paired with the steering of a tracked vehicle.

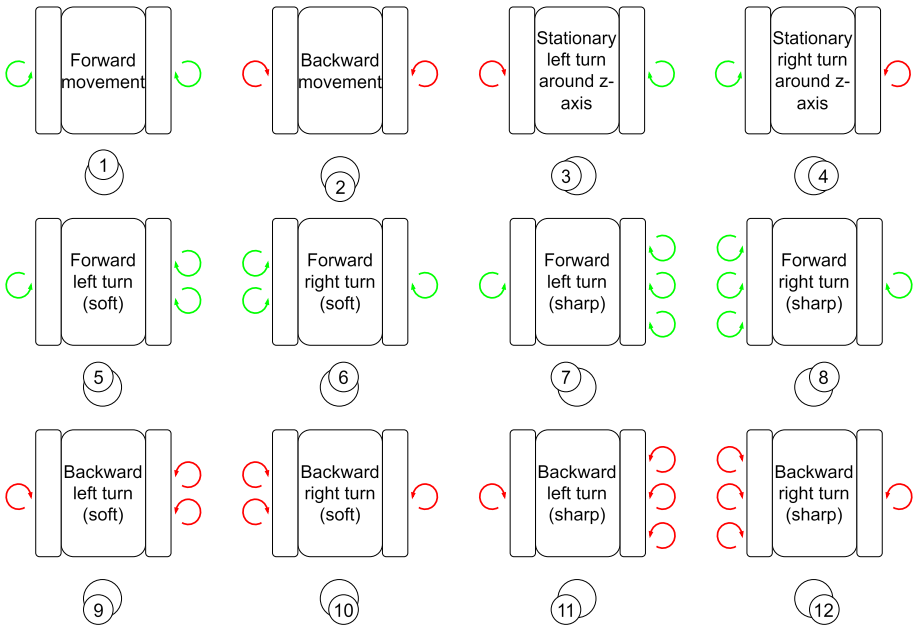


Figure 4.4: Twelve common operations for a tracked vehicle. Beneath each operation, a corresponding joystick placement is found. The number of rotating arrows signifies the speed of each track, and as in figure 4.3, red is backward and green forwards.

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 - machine vision and its corresponding benefits, disadvantages, and uses will be described in detail in section 4.4 starting page 25. 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. The first control loop worth mentioning is also the simplest, as it does not relate to any autonomy yet, but rather input made from a joystick.

In figure 4.5 a flowchart can be seen showing the steps used to adjust the movement of a tracked vehicle manually. In the flowchart, the user isn't directly controlling the speed at each track as a controller maps the joystick position to a signal that the motor at the left/right track can translate into movement unless the current speed matches the joystick position. What this flowchart does not contain, is assurance of the actual state of each track, nor any information of whether or not the tracked vehicle moves as instructed. That knowledge is reserved for the user, observing the vehicle. However, in figure 4.7 a control loop of the same system is found. In figure 4.6 a general control loop is introduced, to increase understanding of the control loop shown in figure 4.7.

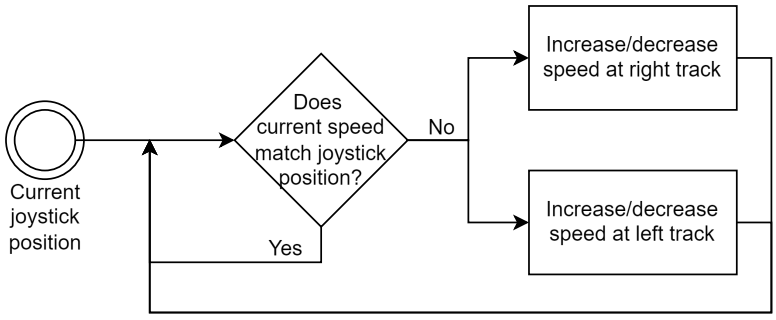


Figure 4.5: Flowchart for a tracked vehicle controlled by a joystick.

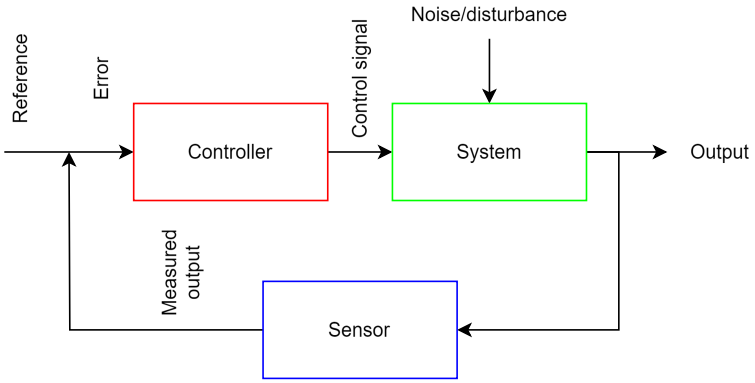


Figure 4.6: General control loop. A reference is introduced, whereafter a controller calculates which signal the system should receive. The system is affected by both the control signal and disturbances, yielding an output, which must be measured to compensate for errors. This is also known as a feedback loop.

Using the general control loop from figure 4.6 in joint with the knowledge from the flowchart shown in figure 4.5, it is possible to derive a more precise control loop, as shown in figure 4.7. In figure 4.7, system knowledge has been added in the form of: "H-Bridge", "DC motor", "Physical System", and "Encoder", as these are common mechanisms. From the control loop, it can be seen that joystick input is transformed into an adjustment in speed of the physical system, which can be disturbed the surface, payload, etc.. At its current state, the only feedback given to ensure the speed of the system is provided by the DC motor and its builtin encoder. The controller then uses this feedback to compare the desired speed (given by the joystick input) with the measured.

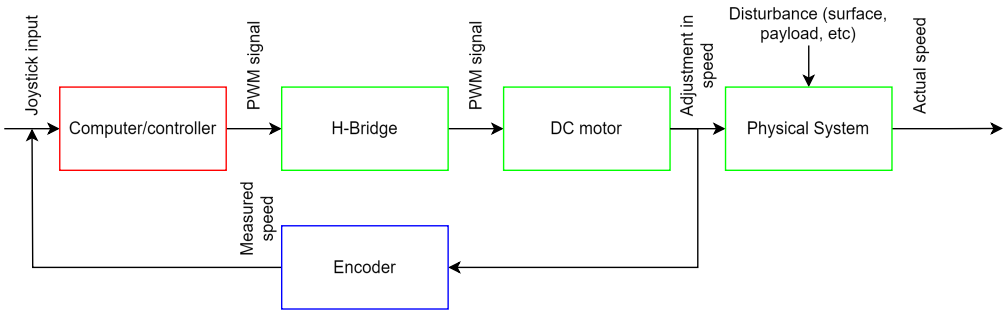


Figure 4.7: Control loop for the tracked vehicle, with the addition of a sensor in the form of an encoder, rather than user observation.

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. The choice of sensor, and thereby further development of this particular control loop will be made in the system design phase, chapter 5 page 44.

Now that common ground has been established for moving the tracked vehicle, it is possible to discuss the flow and control regarding actual obstacle avoidance and how to adjust movement based on sensor input other than encoded speed.

object avoidance

Encoder speed not matching actual speed (mud, sand, etc)

4.2.4 Stable and Unstable Systems

Obtaining a Stable System

Feedback Systems

Feedforward Systems

MIMO Systems

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 4.8. A general formula is derived in equation 4.1:

$$Distance = \frac{T_{Elapsed} * Propagation_{Speed}}{2} \tag{4.1}$$

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).

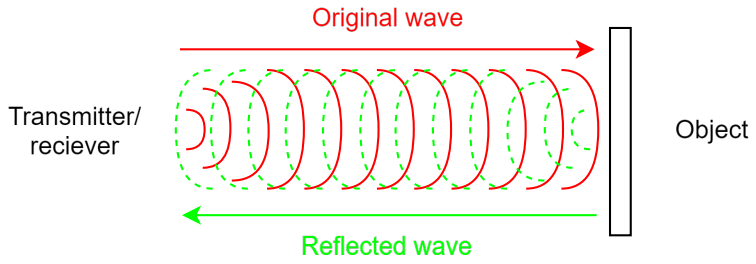


Figure 4.8: Illustration of how ultrasound and ToF sensors work.

Dividing by two yields the actual distance, as the pulse has to travel back and forth, making the total travel twice the distance.

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 4.2 can be set to find the maximum Pulse Repetition Frequency (PRF) at any distance (d):

$$PRF = \frac{0.5 * Propagation_{Speed}}{d} \tag{4.2}$$

As a failsafe, the PRF will be dependent on the maximum distance desired to read data, i.e. 1.5 meters. In figure 4.9 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.

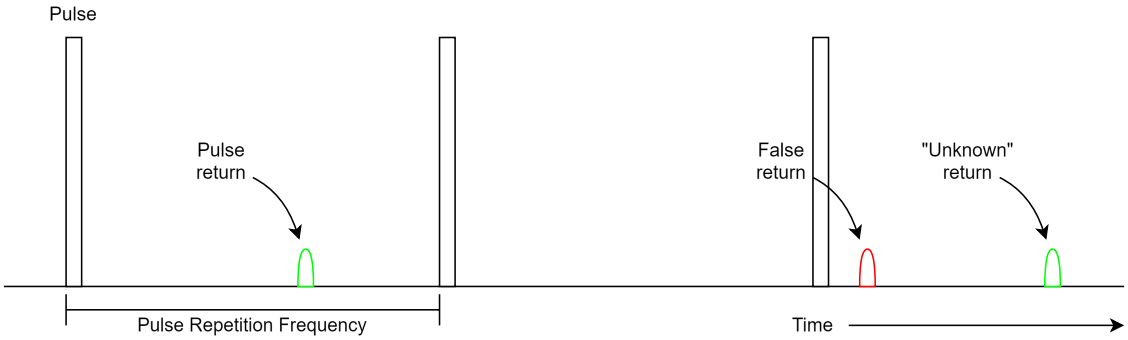


Figure 4.9: PRF correlation and how a too slow pulse return could result in a false positive.

4.3.3 Speed Sensor

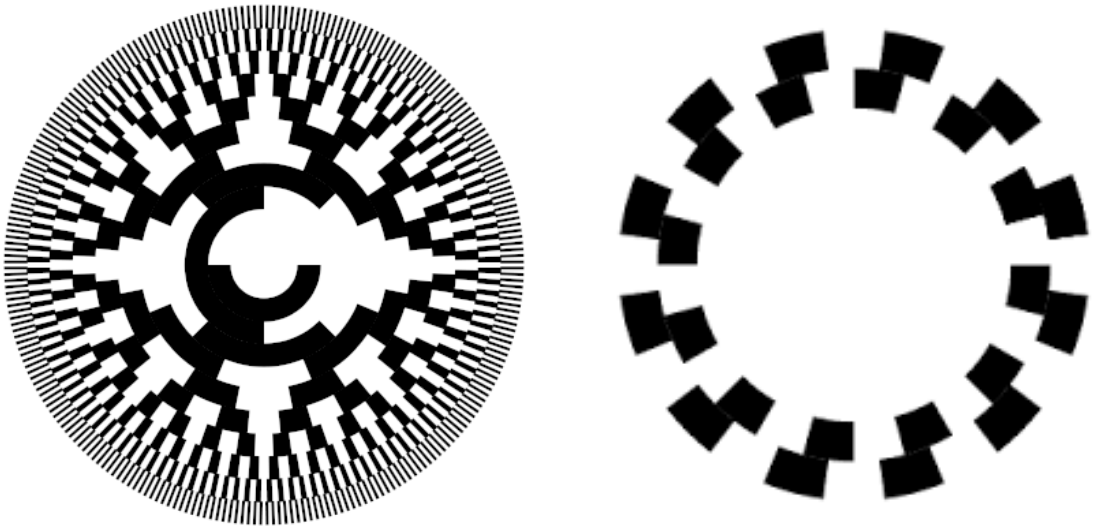
As discussed in section 4.2, it is detrimental to the operation of the robot that feedback is acquired regarding the actual speed of the robot. Without knowing the actual speed control will become largely impossible. To measure the speed several methods can be applied[2]:

[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.

- 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 4.10 both encoder versions are illustrated.



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

Figure 4.10: Encoder topologies.

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 4.3: 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:

$$GaussianExecutionTime = O(Width_{Kernel} * Height_{Kernel} * Width_{Image} * Height_{Image}) \quad (4.3)$$

[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 [23], 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, [24, 25, 26]:

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:

$$Grayscale = \frac{R + G + B}{3} \tag{4.4}$$

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:

$$Grayscale_{Weighted} = \frac{R * 0.299 + G * 0.587 + B * 0.144}{3} \tag{4.5}$$

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 4.11 of how a Gaussian filter processes an image.

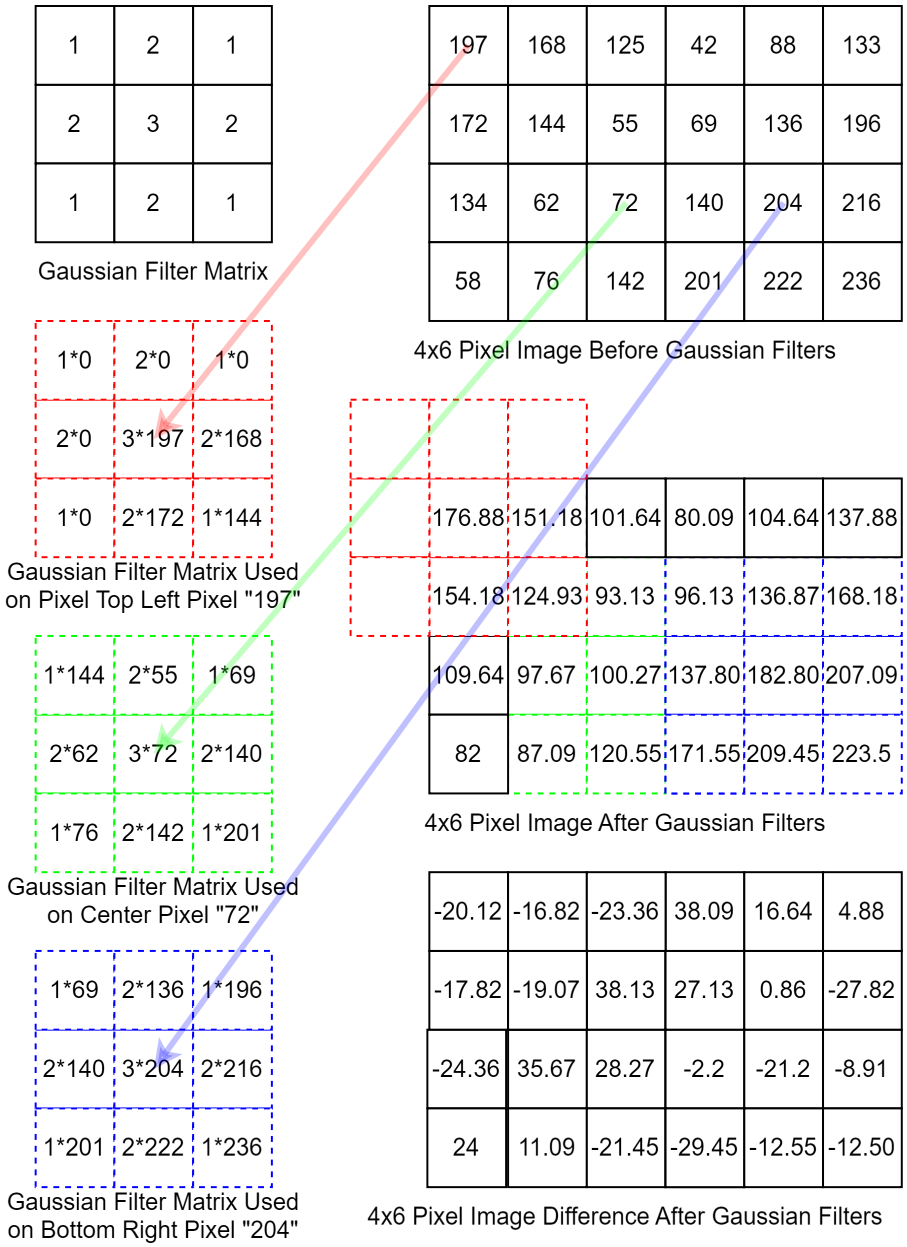


Figure 4.11: Caption

In figure 4.11, 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 4.11, the calculations are:

$$\textbf{RED/197} = \frac{1*0+2*0+1*0+2*0+3*197+2*168+1*0+2*172+1*144}{0+0+0+0+3+2+2+1} \tag{4.6}$$

$$\textbf{GREEN/72} = \frac{1*144+2*55+1*69+2*62+3*72+2*140+1*76+2*142+1*201}{1+2+1+2+3+2+1+2+1} \tag{4.7}$$

$$\textbf{BLUE/204} = \frac{1*69+2*136+1*196+2*140+3*204+2*216+1*201+2*222+1*236}{1+2+1+2+3+2+1+2+1} \tag{4.8}$$

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 4.6 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 4.11 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:

$$G(x,y) = \frac{1}{2*\pi*\sigma^2} * \exp -\frac{x^2+y^2}{2*\sigma^2} \tag{4.9}$$

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:

$$k = 2*ceil(3\sigma)+1 \tag{4.10}$$

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 4.12 through 4.36. 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 4.9, and using a sigma of 1 results in the following equation:

$$G(0,0) = \frac{1}{2 * \pi * 1^2} * \exp - \frac{0^2 + 0^2}{2 * 1^2} \tag{4.11}$$

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

$$\begin{bmatrix} \frac{1}{2*\pi*1^2} * \exp - \frac{-1^2+1^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{1^2+0^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{1^2+1^2}{2*1^2} \\ \frac{1}{2*\pi*1^2} * \exp - \frac{-1^2+0^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{0^2+0^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{1^2+0^2}{2*1^2} \\ \frac{1}{2*\pi*1^2} * \exp - \frac{-1^2+(-1^2)}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{-1^2+0^2}{2*1^2} & \frac{1}{2*\pi*1^2} * \exp - \frac{1^2+(-1^2)}{2*1^2} \end{bmatrix} \tag{4.12}$$

$$\downarrow \tag{4.13}$$

$$\begin{bmatrix} 0.059 & 0.097 & 0.059 \\ 0.097 & 0.159 & 0.097 \\ 0.059 & 0.097 & 0.059 \end{bmatrix} \tag{4.14}$$

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:

$$(0.059 + 0.097 + 0.058) + (0.097 + 0.159 + 0.097) + (0.059 + 0.097 + 0.058) = 0.779 \tag{4.15}$$

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:

$$\frac{0.058}{0.779} \approx 0.074 \tag{4.16}$$

$$\frac{0.097}{0.779} \approx 0.125 \tag{4.17}$$

$$\frac{0.159}{0.779} \approx 0.204 \tag{4.18}$$

Yielding a normalized kernel of:

$$\begin{bmatrix} 0.074 & 0.125 & 0.074 \\ 0.125 & 0.204 & 0.125 \\ 0.074 & 0.125 & 0.074 \end{bmatrix} \tag{4.19}$$

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

$$(0.074 + 0.125 + 0.074) + (0.125 + 0.204 + 0.125) + (0.074 + 0.125 + 0.074) = 1 \quad (4.20)$$

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:

$$\begin{bmatrix} 0.074 * 16 & 0.125 * 16 & 0.074 * 16 \\ 0.125 * 16 & 0.204 * 16 & 0.125 * 16 \\ 0.074 * 16 & 0.125 * 16 & 0.074 * 16 \end{bmatrix} \rightarrow \frac{1}{16} * \begin{bmatrix} 1.184 & 2 & 1.184 \\ 2 & 3.264 & 2 \\ 1.184 & 2 & 1.184 \end{bmatrix} \quad (4.21)$$

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:

$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (4.22)$$

Earlier it was mentioned that the sum should be preserved, and it has been, as both matrices 4.21 and 4.22 have a total sum of 16. This "coincidentally" matches the factor by which the normalized matrix is scaled, which is desired based on [27, 24, 25, 26]. 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:

$$\begin{bmatrix} \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+2^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+2^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+2^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+2^2}{2*1^2}} \\ \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+1^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-1^2+1^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+1^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+1^2}{2*1^2}} \\ \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-1^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{0^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+0^2}{2*1^2}} \\ \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+(-1^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-1^2+(-1^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-1^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+(-1^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+(-1^2)}{2*1^2}} \\ \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+(-2^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+(-2^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{-2^2+0^2}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{1^2+(-2^2)}{2*1^2}} & \frac{1}{2*\pi*1^2} * e^{-\frac{2^2+(-2^2)}{2*1^2}} \end{bmatrix} \quad (4.23)$$

$$\downarrow$$

$$\begin{bmatrix} 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \\ 0.013 & 0.059 & 0.097 & 0.059 & 0.013 \\ 0.022 & 0.097 & 0.159 & 0.097 & 0.022 \\ 0.013 & 0.059 & 0.097 & 0.059 & 0.013 \\ 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \end{bmatrix} \tag{4.24}$$

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:

$$(0.003 + 0.013 + 0.022 + 0.013 + 0.003) + (0.013 + 0.059 + 0.097 + 0.059 + 0.013) + (0.022 + 0.097 + 0.159 + 0.0 \tag{4.25}$$

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

$$\frac{0.003}{0.987} \approx 0.00304 \tag{4.26}$$

$$\frac{0.013}{0.987} \approx 0.01317 \tag{4.27}$$

$$\frac{0.022}{0.987} \approx 0.02229 \tag{4.28}$$

$$\frac{0.058}{0.987} \approx 0.05876 \tag{4.29}$$

$$\frac{0.097}{0.987} \approx 0.09828 \tag{4.30}$$

$$\frac{0.159}{0.987} \approx 0.16109 \tag{4.31}$$

Yielding a normalized kernel of:

$$\begin{bmatrix} 0.00304 & 0.01317 & 0.02229 & 0.01317 & 0.00304 \\ 0.01317 & 0.05876 & 0.09828 & 0.05876 & 0.01317 \\ 0.02229 & 0.09828 & 0.16109 & 0.09828 & 0.02229 \\ 0.01317 & 0.05876 & 0.09828 & 0.05876 & 0.01317 \\ 0.00304 & 0.01317 & 0.02229 & 0.01317 & 0.00304 \end{bmatrix} \tag{4.32}$$

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

$$\begin{aligned} &(0.00304 + 0.01317 + 0.02229 + 0.01317 + 0.00304) + \\ &(0.01317 + 0.05876 + 0.09828 + 0.05876 + 0.01317) + \\ &(0.02229 + 0.09828 + 0.16109 + 0.09828 + 0.02229) + \\ &(0.01317 + 0.05876 + 0.09828 + 0.05876 + 0.01317) + \\ &(0.00304 + 0.01317 + 0.02229 + 0.01317 + 0.00304) \approx 1 \end{aligned} \tag{4.33}$$

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

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

$$\begin{bmatrix} 0.00304*16 & 0.01317*16 & 0.02229*16 & 0.01317*16 & 0.00304*16 \\ 0.01317*16 & 0.05876*16 & 0.09828*16 & 0.05876*16 & 0.01317*16 \\ 0.02229*16 & 0.09828*16 & 0.16109*16 & 0.09828*16 & 0.02229*16 \\ 0.01317*16 & 0.05876*16 & 0.09828*16 & 0.05876*16 & 0.01317*16 \\ 0.00304*16 & 0.01317*16 & 0.02229*16 & 0.01317*16 & 0.00304*16 \end{bmatrix} \tag{4.34}$$

$$\downarrow$$

$$\frac{1}{16} * \begin{bmatrix} 0.048 & 0.210 & 0.356 & 0.210 & 0.048 \\ 0.210 & 0.956 & 1.57 & 0.956 & 0.210 \\ 0.356 & 1.57 & 2.58 & 1.57 & 0.356 \\ 0.210 & 0.956 & 1.57 & 0.956 & 0.210 \\ 0.048 & 0.210 & 0.356 & 0.210 & 0.048 \end{bmatrix} \tag{4.35}$$

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 4.36:

$$\frac{1}{160} * \begin{bmatrix} 0 & 2 & 4 & 2 & 0 \\ 2 & 9 & 16 & 9 & 2 \\ 4 & 16 & 26 & 16 & 4 \\ 2 & 9 & 16 & 9 & 2 \\ 0 & 2 & 4 & 2 & 0 \end{bmatrix} \tag{4.36}$$

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:

$$G_{(x)} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} and G_{(Y)} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{4.37}$$

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 4.12:

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

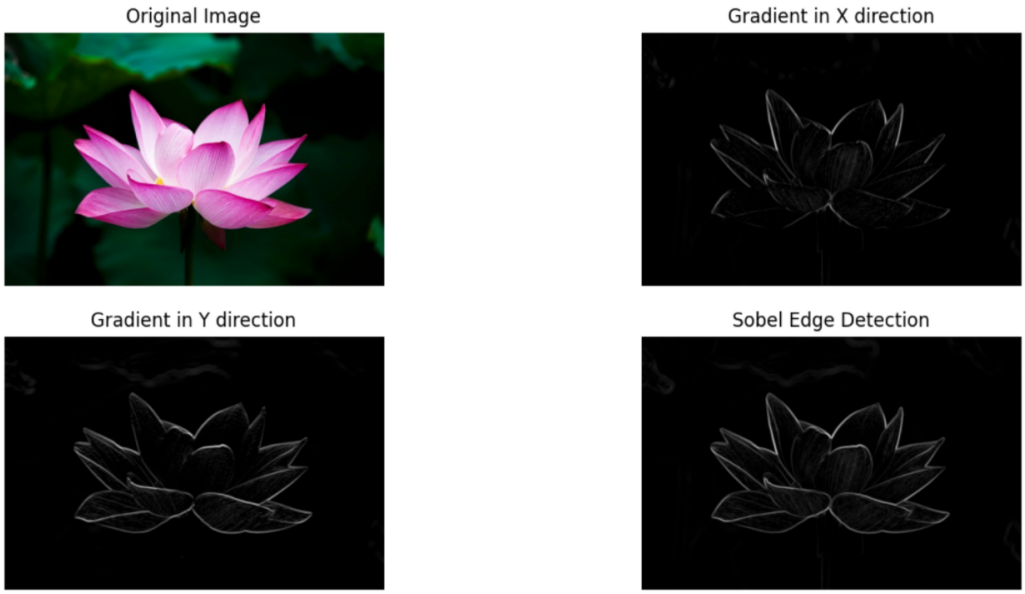


Figure 4.12: Comparison of images exposed to the Sobel, [28].

$$G = \sqrt{G_{(X)}^2 + G_{(Y)}^2} \tag{4.38}$$

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:

$$O = \arctan(\frac{G_{(X)}}{G_{(Y)}} \tag{4.39}$$

The result of using inverse tangent on all edges is shown visually in figure 4.13. While this information might not seem useful, it becomes indispensable in the next step, found at page 34.

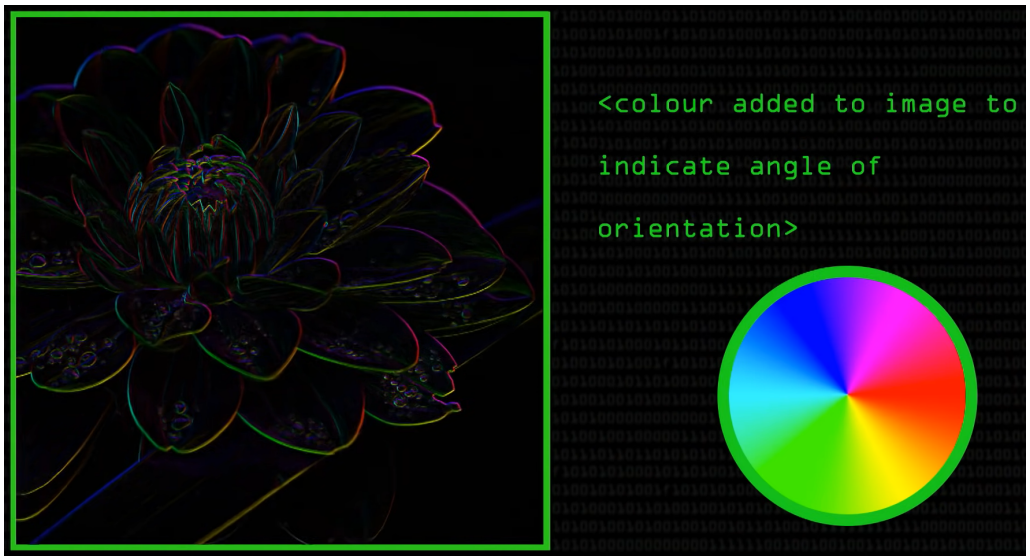


Figure 4.13: The circular RGB disc is meant to represent which orientation each edge has. The actual image is in grayscale, the colors are only added to show orientation. Image courtesy of [25].

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 4.14 an illustration of how each pixel is compared to its adjacent pixels in its respective orientation is shown.

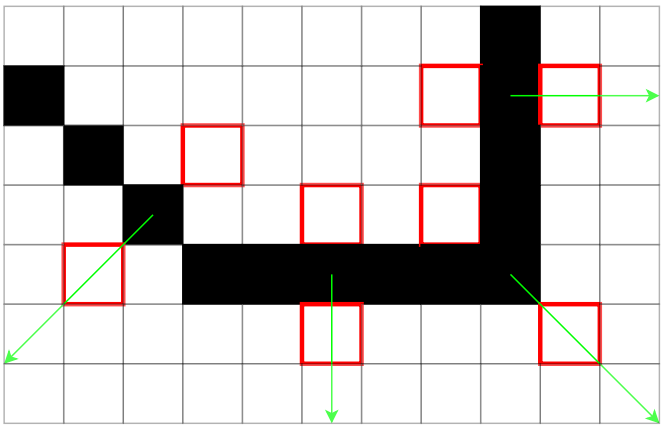


Figure 4.14: The black line has been reduced to 1 pixel wide, by comparing all the adjacent pixels (red border) in the correct orientation (green arrow) with each pixel along the line.

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, [26].

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, [29]. 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.

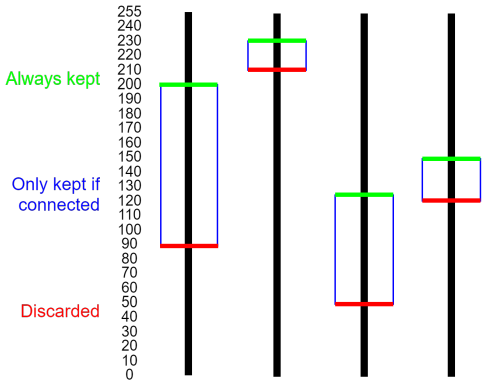


Figure 4.15: Various double threshold scenarios. The pixel values are along the Y-axis, ranging from 0-255 (8-bit).

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 4.15.

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 47, 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 4.5.6 starting page 41. The entirety of this section is based upon: [30, 31, 32, 33, 34, 35, 36, 37].

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 4.16 the correlation between image space and parameter space is visualized.

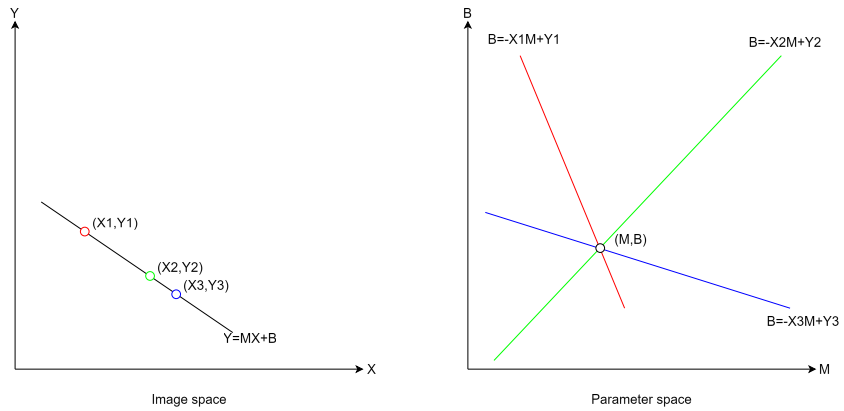


Figure 4.16: Correlation between image space and parameter space when the Hough transform is used.

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 4.17.

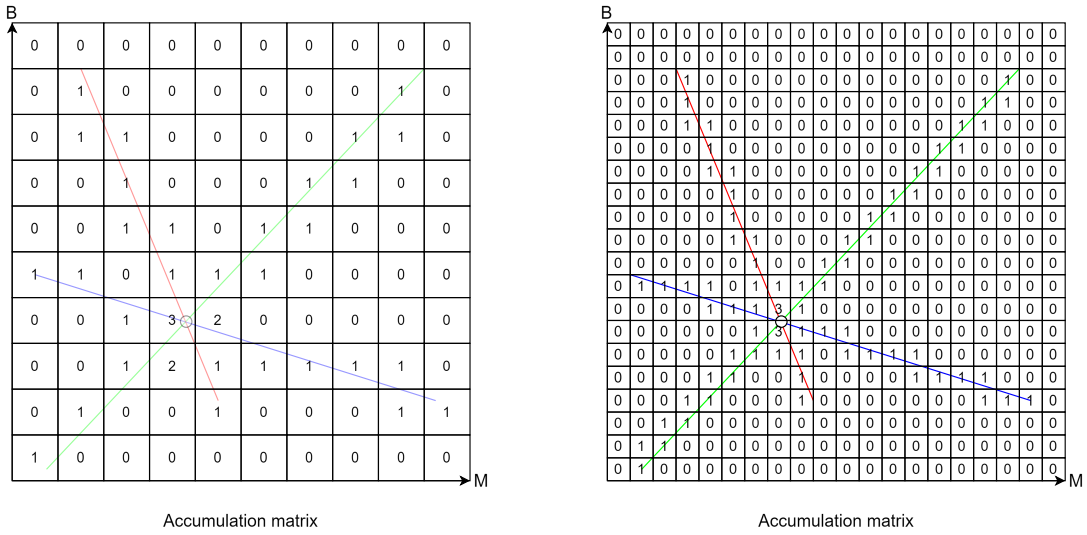


Figure 4.17: As increasing amounts of lines intersect, the common intersection weight also increases, revealing a maximum. To the right is an accumulation matrix with 4 times the resolution, but the maxima have been spread now, leaving ambiguity for which value is the correct.

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:

$$y = mx + b \tag{4.40}$$

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

$$x * sin(\theta) - y * cos(\theta) = \rho \tag{4.41}$$

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 4.41 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 4.17 shows possible complications. In figure 4.18, sinusoids derived from an edge can be seen, along with corresponding maxima.

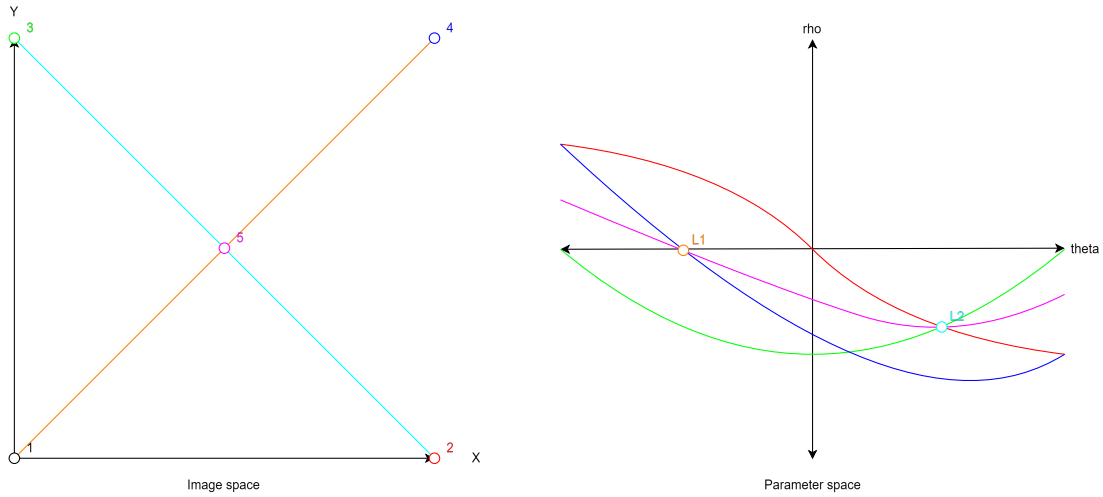


Figure 4.18: Sinusiodal representation in parameter space. The intersections (maxima) in parameter space translate to lines in image space, as shown by L1 and L2.

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.

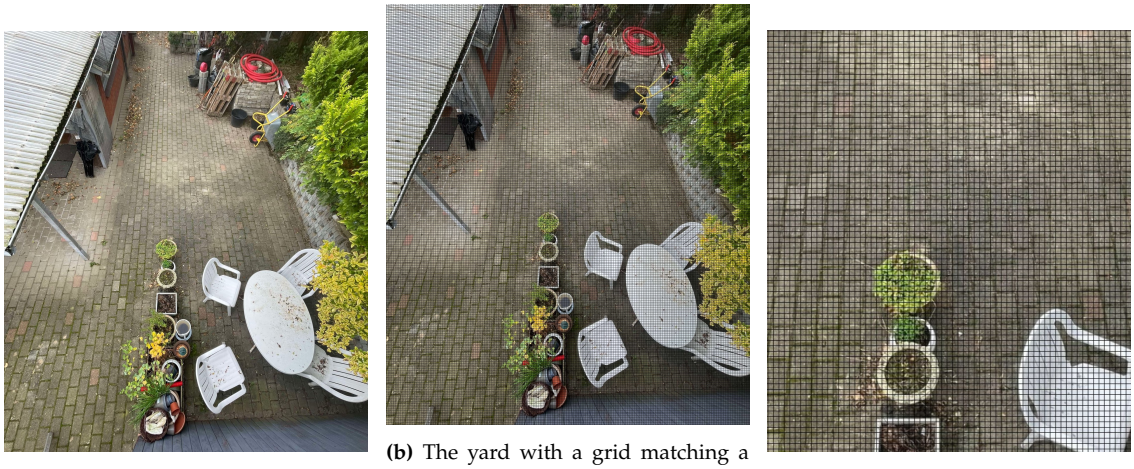
4.5.1 Instantiating a Coordinate System

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 realistically represent the surface. 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, [38]. 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", [23]. 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 4.19a a picture of the author's yard is shown, and in 4.19b the same yard with a grid approximately equivalent to a 1000x100 coordinate system. Figure 4.19c 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 4.19 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 XX 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 4.19c.

Figure 4.19: The author's yard, both 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. An interpretation can be made using the model and equations provided in [23]. In figure 4.20 the relationship between the image taken and the surface plane can be seen, wherein any image taken at an angle will develop a trapezoid floor plane.

The math accompanying their model is:

$$X_c = \frac{uh}{(f * sin(\theta_{EL}) + v * cos(\theta_{EL}))} \tag{4.42}$$

$$Y_c = \frac{h * (v * sin(\theta_{EL}) - f * cos(\theta_{EL}))}{(f * sin(\theta_{EL}) + v * cos(\theta_{EL}))} \tag{4.43}$$

Where:
h = height to optical center in the camera from the ground plane
f = focal length of the camera
θ_{EL} = Elevation angle of the camera with respect to the ground plane
(u,v) = coordinates in the image plane
X_c = X coordinate in cartesian ground plane
Y_c = Y coordinate in cartesian ground plane
$Y_{c_{min}}$ = A (non-observed zone)
$Y_{c_{max}}$ = B
$X_{c_{min}} = \frac{C}{2}$
$X_{c_{max}} = \frac{D}{2}$

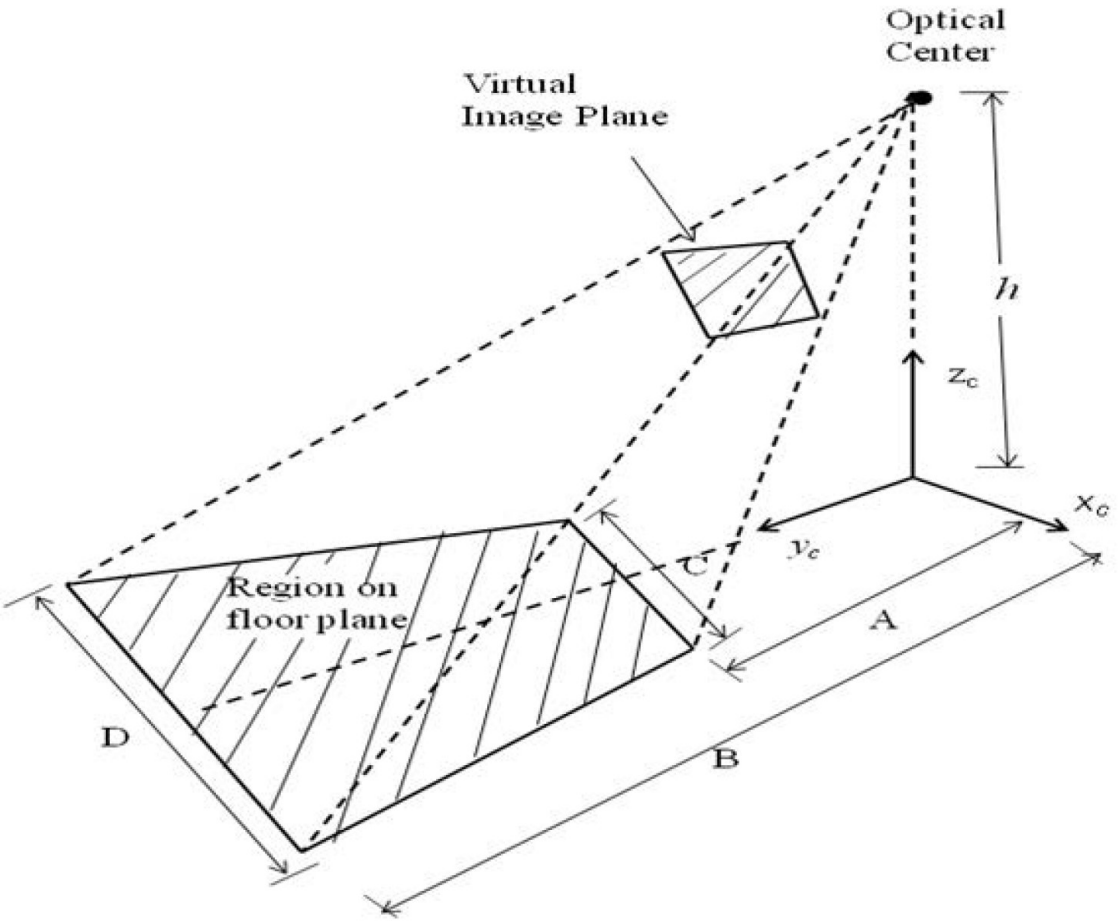


Figure 4.20: Correspondance between the image taken and actual surface plane captured by the image, [23].

4.5.2 Mapping Boundaries

4.5.3 Localizing a "Home"

4.5.4 Remembering Obstacles

4.5.5 Dividing Into Zones

4.5.6 A Systematical Approach

4.5.7 Kalman Filter and How To Use It

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. This is done through testing in section 5.9.4 on page 5.9.4. The resulting graph will be along the lines of the idealized graph in Figure 4.22, 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, [39, 40].

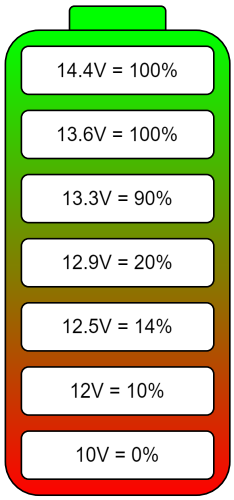


Figure 4.21: Various battery percentages for a lithium battery. Values are taken from [39, 40].

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, [39, 40].

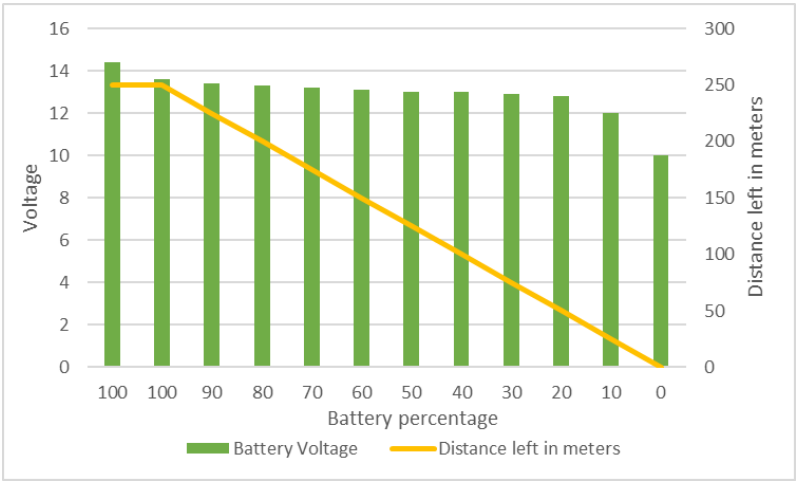


Figure 4.22: Idealized correlation between voltage and distance left, that the system can travel.

4.7 Wireless Communication

4.7.1 Bluetooth

Usage

Protocols

4.7.2 Wi-Fi

Usage

Hosting

Protocols

4.7.3 Connecting Several ESP32's Together

5 System Design

5.1 System Design Overview for the Prototype

To create an intuitive overview of what elements 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.
- 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 through machine vision or other similar protocols.
- 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 (Dr Robot Jaguar Lite).
  The physical platform on which the prototype will be developed.
- Charge point.
  External housing which contains the computing module, wireless communication, user interface, and in future iterations, charging capabilities

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 45.

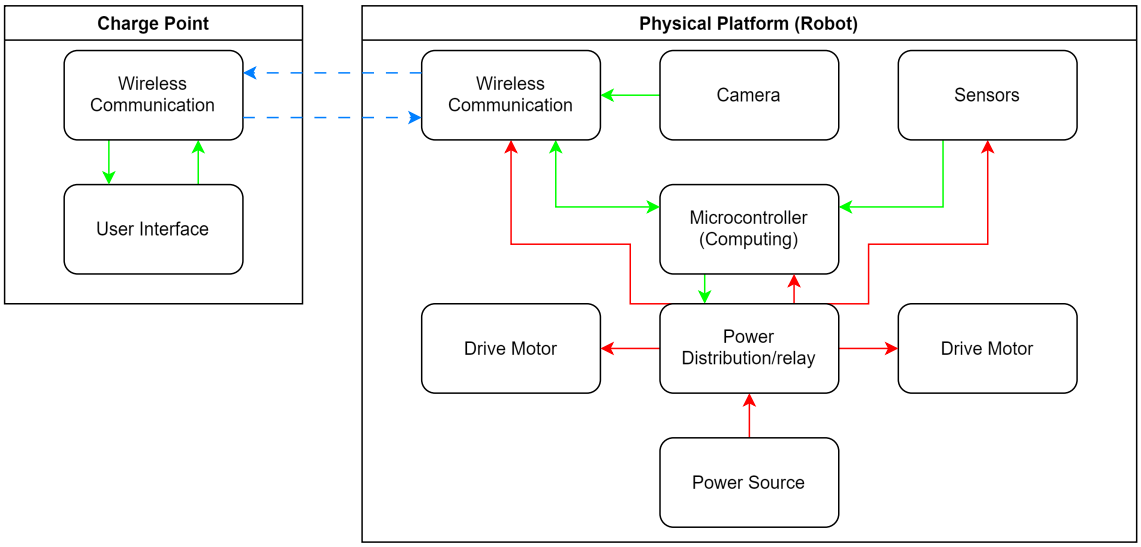


Figure 5.1: Block diagram showing relations between modules of the 1st iteration. Red lines signify power, green signify data, and blue dotted signify wireless data.

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.

5.2 Microcontroller Module

The microcontroller has to connect to a supposedly large amount of sensors (at least 5 along the front of the robot) as well as control power distribution and wireless communication, a large amount of I/O pins will be handy, along with integrated wireless communication, and multiple cores for added processing power.

5.2.1 Specification

-

5.2.2 Design

5.2.3 Implementation

5.2.4 Test

Test Setup

Actual Testing

Test Results

5.2.5 Summary

5.3 Camera Module

5.3.1 Specification

-

5.3.2 Design

5.3.3 Implementation

5.3.4 Test

Test Setup

Actual Testing

Test Results

5.3.5 Summary

5.4 Wireless Communication Module

5.4.1 Specification

-

5.4.2 Design

5.4.3 Implementation

5.4.4 Test

Test Setup

Actual Testing

Test Results

5.4.5 Summary

5.5 Computing Module

5.5.1 Specification

-

5.5.2 Design

5.5.3 Implementation

5.5.4 Test

Test Setup

Actual Testing

Test Results

5.5.5 Summary

5.6 Sensors Module

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

5.7 Drive Motors Module

5.7.1 Specification

-

5.7.2 Design

5.7.3 Implementation

5.7.4 Test

Test Setup

Actual Testing

Test Results

5.7.5 Summary

5.8 Power Distribution Module

5.8.1 Specification

-

5.8.2 Design

5.8.3 Implementation

5.8.4 Test

Test Setup

Actual Testing

Test Results

5.8.5 Summary

5.9 Power Source Module

5.9.1 Specification

-

5.9.2 Design

5.9.3 Implementation

5.9.4 Test

Test Setup

Actual Testing

Test Results

5.9.5 Summary

5.10 Physical Platform Module

GRUNDIG BESKRIVELSE AF HVORFOR PÅ LARVEFØDDER

5.10.1 Specification

-

5.10.2 Design

5.10.3 Implementation

5.10.4 Test

Test Setup

Actual Testing

Test Results

5.10.5 Summary

5.11 Charge Point Module

5.11.1 Specification

-

5.11.2 Design

5.11.3 Implementation

5.11.4 Test

Test Setup

Actual Testing

Test Results

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

[23] 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.

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

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

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

[27] *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.

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

[29] *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.

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

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

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

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

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

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

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

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

[38] *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.

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

[40] *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.

[41] *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, 75

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 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 A.3 starting page 69.
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), [22].*
- *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.*

A.3 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.

A.3.1 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.

A.3.2 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).

A.3.3 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.

A.3.4 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.

A.3.5 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.

A.3.6 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.

A.3.7 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..

A.3.8 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.

A.3.9 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.

A.3.10 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.

A.3.11 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].

A.3.12 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".

A.3.13 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.

A.3.14 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.

A.3.15 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.

A.3.16 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.

A.3.17 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, [41].
AC2:
Under extraordinary circumstances, such as traversing small obstacles, volume may be increased to 65dB.
AC3:
Cooling of the robot must not surpass 55dB.

A.3.18 Small Size

An autonomous robot as small in size as possible, ideally no larger than a Roomba (approximately 45cm), [22].
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.

A.3.19 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.

A.3.20 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..

A.4 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.

A.4.1 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:

- A.3.2 - Operate the Robot
- A.3.3 - A Home for the Robot
- A.3.4 - User-interface
- A.3.6 - Robot Go Home
- A.3.9 - Pattern Recognition, AC2
- A.3.12 - Water Avoidance
- A.3.13 - Spatial Awareness
- A.3.15 - Memory Capabilities
- A.3.16 - Obstacle Avoidance

A.4.2 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:

- A.3.1 - Weed Removal
- A.3.9 - Pattern Recognition, AC1
- A.3.10 - 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.

A.4.3 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:

- A.3.5 - Go Anywhere
- A.3.7 - Easy Setup
- A.3.8 - Stay On My Turf
- A.3.14 - Self-adjusting Scheduling
- A.3.19 - Selfrecovery
- A.3.20 - Area

A.4.4 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:

- A.3.11 - Weather Endurant
- A.3.17 - Low Noise
- A.3.18 - Small Size

A.4.5 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.