NTNU
Norwegian University of
Science and Technology

Design of an Optimal Path-Planning System for a Trash-Collecting USV

Gulleik Lundtorp Olsen

01-06-2022

Master's Thesis
30 ECTS
Department of Engineering Cybernetics
Norwegian University of Science and Technology
Trondheim, Norway

Supervisor: Prof. Thor I. Fossen, Department of Engineering Cybernetics, NTNU

Contents

Preface

This thesis is a continuation of the authors' specialization project [1]. The motivation behind this thesis is to optimize the autonomy of the Clean Sea Solution cleaning drone V1 [2] to allow robots to be a more significant part of the green shift. I'm inspired by the work of The Ocean Cleanup project and hope to also play a vital role in keeping the oceans clean. As a student of cybernetics and robotics, it was also considered a rewarding challenge to work with an autonomous vehicle.

Acknowledgements

This report has been written with the aid of Per Elvestuen, Ragnar Eggen, Catharina Frostad, and Matthias Van Middendorp from Clean Sea Solutions. I want to thank them for letting me become a part of the Clean Sea team and for allowing me to work on this exciting project. I would also like to thank Ibrahim A. Hameed from the Department of ICT and Natural Sciences at NTNU in Ålesund for helping me find the suitable algorithms to use in this project. The employees at Maritime Robotics have also been invaluable in helping me get as far as I could and deserve special thanks. Lastly, I would like to give a special thanks to professor Thor I. Fossen at the Department of Engineering Cybernetics at NTNU in Trondheim for supervising me during this journey.

Abstract

This thesis further develops Clean Sea Solutions Cleaning Drone V1 (CDV1). The task was threefold. Firstly, the waypoint-generating algorithm created in the authors' specialization project will be used to control the drone through Robotic Operating System (ROS). The guidance system was run in ROS on a computer on land. A network module then transmitted the data to and from the isolated control system of the CDV1. The CDV1 was then run using the new guidance system in a small test area in Trondheim's harbor basin. Secondly, the current design of the drone will be tested and evaluated. To test the design of the current drone, the Cleaning Drone was manually steered into collections of trash to evaluate its trash-collecting capabilities. Lastly, a method for avoiding obstacles was added to the system and tested in a simulator. Object avoidance was done using the pathplanning algorithm D* Lite and tested in a simulator. D* Lite could easily steer the USV clear of land and unknown obstacles. The pathplanning was also improved by adding padding to obstacles and increasing the search direction.

Sammendrag

Denne rapporten omhandler videreutviklingen av Clean Sea Solutions Cleaning Drone V1 (CDV1). Oppgaven var tredelt. Det første punktet handler om å få den veipunktgenererende algoritmen utviklet i forfatternes prosjektoppgave til å styre dronen gjennom Robotic Operating System (ROS). Baneplanleggingssystemet ble kjørt i ROS på en datamaskin fra land. En nettverksmodul sendte deretter dataene til og fra det isolerte kontrollsystemet til CDV1. CDV1 ble deretter kjørt med det nye baneplanleggingssystemet i et lite testområde i Trondheims havnebasseng. Andre punkt innebærer å evaluere og teste dagens design av dronen. Dronens nåværende design ble testet ved at dronen ble manuelt styrt inn i søppelopphopninger på vannet for å evaluere dens søppeloppsamlingssevne. Til slutt ble en metode for å unngå hindringer lagt til systemet og testet i en simulator. Objektunngåelse ble gjort ved hjelp av baneplanleggingsalgoritmen D* Lite og testet i en simulator. D* Lite kunne enkelt styre dronen unna land og ukjente hindringer. Veiplanleggingen ble også forbedret ved å legge til virtuell polstring på hindringer og ved å øke antall søkeretninger.

List of Figures

List of Tables

1 Introduction

This chapter introduces the reader to the motivation and organization behind this thesis. The author has in this chapter also outlined the goals of this research, presented the case study, and given a short introduction to the unmanned surface vehicle (USV) used. The chapter will have the following structure:

- **Section 1.1:** Motivation
- **Section 1.2:** Problem definition
- **Section 1.3:** Literature review
- **Section 1.4:** Contributions
- **Section 1.5:** Organisation

1.1 Motivation

The Earth is about 71 percent water, and our oceans are an important resource for food production and a home for the Earth's aquatic organisms. The quality of life for these organisms is, however, threatened by human negligence and pollution of human debris in the oceans [7]. This negligence affects most parts of the oceanic wildlife with examples from larger organisms [8] [9] down to sea worms [10]. Most of this debris, mostly plastics, stems from land. Predictions estimate that this output will increase by order of magnitude within the decade [11]. Here are urban areas associated with most of the pollution. By focusing on collecting the trash before it leaves these urban areas, the negative impact of this pollution can hopefully be mitigated.

A study conducted in the coastal areas in Chicago, USA, indicates that direct littering in combination with the retention of trash by wind and waves close to the pier wall dominates the trash accumulation at piers [12]. This littering does not only pose an environmental challenge but also leaves a mark on the aesthetics of the coastal area.

The global effort to create autonomous vehicles for garbage collection has increased, resulting in multiple drones in the past decade being created. However, most of these drones are small and focus on closed pockets of water. The papers about them also focus on the design of the drone in question more than the guidance system used to collect the trash [13] [14] [15] [16] [17]. As a result, path optimization regarding autonomous cleaning of larger areas has received little attention. As per the author's findings, no papers focused on optimal ways of cleaning harbor basins using unmanned vehicles. This finding comes in contrast to other similar fields, such as autonomous vacuuming and floor-cleaning, where research on different path planning schemes is seen [18] [19] [20]. These methods give a nice platform to build upon, but in order to adapt the methods for larger, open, marine areas, one has to take into account that the trash is non-stationary and tends to pile up in hotspots [12]. The areas which need to be covered by a single unmanned surface vehicle (USV)

will also be significantly bigger. Thus some optimization and prioritization of where to clean will be needed.

This thesis is written in cooperation with a startup company called Clean Sea Solutions (CSS) [21]. The company has a goal of offering a clean waterfront as a service. CSS offers this service through multiple means. One of them is through the use of a USV. The USV or *Cleaning Drone V1* (CDV1) is a retrofitted version of Maritime Robotics *Otter* [22]. The *Otter* possesses autonomous capabilities, such as path following, but does not have a way of optimizing this path in regards to trash collection. This project aims to find an optimal path for an autonomous trash-collecting USV to maximize the amount of trash collected per battery charge.

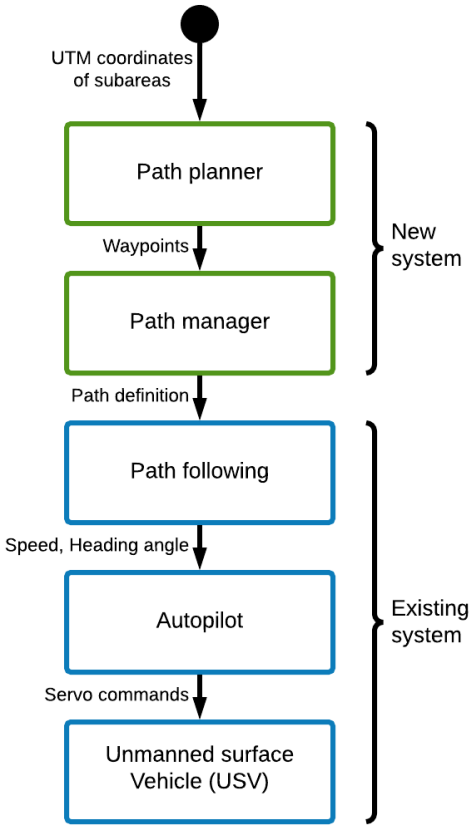


Figure 1: The relationship between different systems. The new system is designed in this thesis.

1.2 Problem definition

This thesis will aim to build upon existing autonomous covering/cleaning methods of an area and specialize them in harbor basin cleaning. In [1], a waypoint generating algorithm was created to maximize the amount of trash collected in one run by the CDV1. The algorithm was tested using a customized simulation environment and built upon multiple assumptions that will not hold in a real use case. The goal of this thesis is thus threefold. The first goal is to implement the waypoint generating algorithm from [1] onto the CDV1. Secondly, the design of both the CDV1 and its guidance system from [1] will be evaluated. Third, a framework and way of using a lidar mounted on the CDV1 to negate the following assumptions will be presented:

- There are no static obstacles for the CDV1 to collide with.
- The simulated battery usage is correct.

Lastly, a more optimized travel distance estimation method will be presented and implemented for the CDV1 to optimize the total range estimation of the CDV1.

Based on Figure 1, this report will describe the implementation and incremental development of a path planner, path manager, static object avoidance system, and a continuous remaining range prediction for the Clean Sea Solutions Cleaning drone V1. The USV has already got a working autopilot and path-following algorithm. Thus, the generation and updating of waypoints based on internal and external factors will be the main focus of this report. The testing of the USV will happen in the harbor outside of Maritime Robotics headquarters in Trondheim.

Research questions:

1. How can the waypoint generation algorithm from [1] be implemented in the Cleaning Drone V1?
2. How can static object avoidance be incorporated into the waypoint generating algorithm?
3. How can the remaining range of the USV be predicted more accurately than relying solely on the battery level?
4. Is the drone's design capable of collecting trash from hotspots?

1.2.1 The USV workarea

The area used for testing the implementation of the algorithm is the harbor outside of Maritime Robotics headquarters in Trondheim, as shown in Figure 2. This remote part of Trondheim harbor is considered well suited for testing the implementation of the waypoint generating algorithm as it is pretty sheltered from waves and there is little traffic in the area. Object avoidance can also be tested relatively easily here as the area consists of piers, corners, shallow areas, and boats. It is worth noting that this workarea is significantly smaller than the one used in the specialization project as it is only 216 meters times 240 meters. This size reduction is due to the unavailability of good test areas in Trondheim in combination with the fact that a vast area is not needed to test the desired implementation and object avoidance.

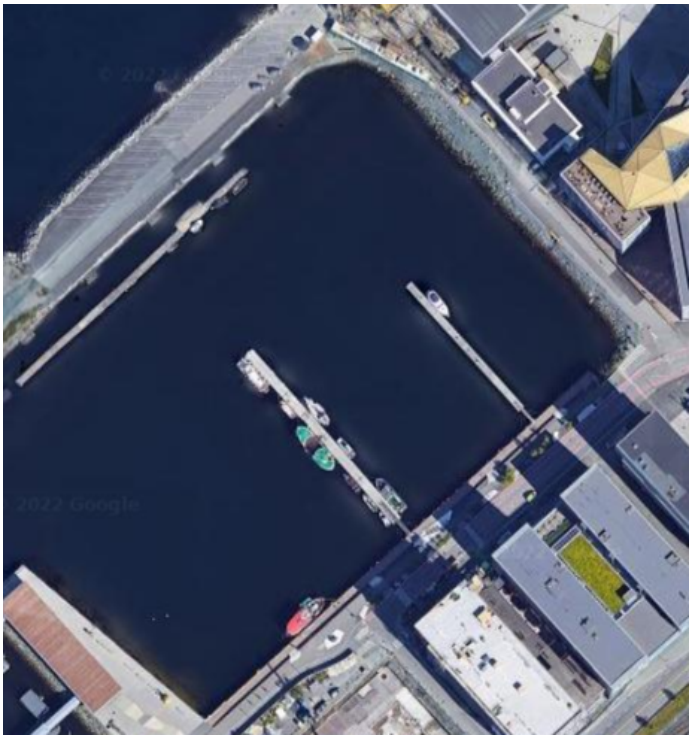


Figure 2: The workarea of the USV.

1.2.2 The Clean Sea Solution USV

The USV used by Clean Sea Solution is a retrofit of the Maritime Robotics *Otter* USV [22]. The specifications of the *Cleaning drone V1* (CDV1) can be found in Appendix A.

The CDV1 is a fully electric catamaran that uses two propellers to move. It does not have any rudders, so it has to adjust the speed of one of its propellers to turn while moving. However, turning on the spot is an included ability that makes the Otter significantly easier to operate and simulate.

Between the catamaran's pontoons is a box with an opening mounted at a slight angle in order for the USV to be able to collect trash it comes across. The backside of the box is filled with holes for water to flow through, meaning that the CDV1 is not capable of collecting small pieces of trash. The CDV1 can automatically empty its bin by pushing the backplate through the box, forcing the trash out in front. Emptying of the trashbin happens in a floating trashcan located next to the CDV1's charging station. In addition, a door is mounted on the box as shown in Figure 3 so that the drone can close its bin in order to prevent the trash from escaping whenever it has to reverse or do any other movement which can result in trash loss. A more detailed description of the CDV1 design can be found in Chapter 3.

(a) The CDV1 cleaning drone seen from the front with the backplate of the collection bin pushed out.

(b) The CDV1 cleaning drone seen from the side.

Figure 3: The CDV1 cleaning drone seen from the front and side.

1.3 Literature review

The literature review aims to present the existing methods and research within the report's scope. Different techniques, advantages and disadvantages, and relevant terminology will be presented to prepare the reader for the current state of research within this field.

1.3.1 Levels of autonomy

As autonomous systems are becoming more common in today's society, the research community is starting to implement more standards to explain the different levels of autonomy. The society for automotive engineers (SAE) has defined six levels of autonomy ranging from no autonomy to full autonomy for cars [23]. A similar scale has also been done for ships by Lloyds Register [6]. These levels can be used to explain the different levels of autonomy for the CDV1. Currently, the CDV1 is at autonomy level 3, as explained by Table 1. In order for CDV1 to do its job, its constantly in need of human input.

Autonomy level	Descripition
AL0	Manual navigation – no autonomous function.
AL1	On-ship decision support system
AL2	Off-ship decision support system
AL3	Semi-autonomous ship with active human in the loop.
AL4	Ship opperates autonomously, but with a human in the loop
AL5	Fully autonomous with room for human interferance
AL6	Fully autonomous with no need for human interferance

Table 1: Levels of vessel autonomy according to Lloyd's Register [6].

This thesis aims to lift the autonomy of the CDV1 to the autonomy level 4. The USV should then have a high level of autonomy in a known environment but cannot handle every situation it can encounter.

1.3.2 Path planning algorithms

In order for the CDV1 to operate more autonomously, it needs to include a path planning algorithm that generates a path for the CDV1 for it to be able to complete its mission. Path-planning algorithms have been separated into three distinct groups, namely the *classical approach*, the *advanced approach*, and the *hybrid group* by [24].

Furthermore, [24] described the classical approach as a two-step process consisting of (1) environment modeling to prepare for the search and (2) performing the search of the optimal path in this environment [24]. These methods are often used when the obstacles are static and there is no need for path replanning. This problem description matches the use case solved by the path planning algorithm developed in the authors' specialization project. Thus, the resulting path planning algorithm can be described as classical. Table 2 presents some of the advantages and disadvantages of the different algorithms discussed.

The A* path planning algorithm and its evolution's

The A* algorithm is one of the most common classical algorithms. The A* algorithm is popular due to it being correct, complete, and optimal [25]. Correctness in this context means that there will be no obstacles on the computational path. Completeness refers to finding a path if a feasible path exists. Optimality means that the algorithm can calculate the shortest path from the start point to the goal point [25].

[26] presents evolutions on the A* algorithm. Methods like these will be referred to as dynamic classical methods by the author, as they are classical methods that have evolved the ability to recalculate their trajectory. These methods include, among others, D*, D* lite, and Field D*. Both D*, D* lite, and Field D* are more optimized toward replanning the path as new obstacles are detected. However, they do not work in real-time as the current map must be updated, and the algorithm has to calculate a new path whenever an obstacle is detected. Of the presented A* iterations, Field D* is the most recent iteration of the algorithm. It has at least as good performance when finding an optimal path as D* and D* lite but can additionally smooth this path [27].

The advantage of the A* evolutions is that they are somewhat easy to implement. In addition, because of the algorithm's popularity and age, there are multiple articles and papers suggesting improvements to these types of algorithms. Examples of this are path-planning for a moving target [28], improved edge avoidance and increased search directions [4], and avoidance of complicated obstacles and sharp corners [29].

Advanced path planning methods

The advanced methods presented in [24] were methods more suited for real-time applications with the ability to replan and deal with dynamic obstacles. [24] separates these methods into four main categories, namely:

- Machine learning.
- Directional approaches.
  - Potential field methods
  - Velocity space methods
  - Vector field histogram methods
- Evolutionary algorithms.
- Sampling-based algorithms.

Machine learning methods include methods like convolutional neural networks as proposed in [25] and are a topic that is receiving much attention at the moment.

Directional approaches includes three different methods according to [24]. These are namely, *Potential field methods*, *Velocity space methods*, and *Vector field histogram methods*. Potential field methods work by assigning a positive field to the goal position and a negative field to any obstacles; in that way, the obstacles repel the agent, and the goal attracts the agent. This approach does, however, have the disadvantage of potentially trapping the agent in a local minima [30].

Vagale also separates velocity space methods into three subcategories: *Velocity obstacles*, *dynamic*

window, and *Vector field histogram methods*. *Velocity obstacles* is a method optimized toward avoiding collisions between moving obstacles [31]. *Dynamic window* is a method that uses A* post smoothing (A* PS) to calculate an optimal path. It uses a dynamic window to check for and avoid sudden obstacles by taking into account the surge and angular velocity of the agent [25] [32]. *Vector field histogram* methods calculate a polar histogram around the agent. This histogram represents the risk of collision [33].

Sampling-based algorithms sample the available area to find a suitable time with low computational cost. The probabilistic roadmap (PRM) and rapidly exploring random tree (RRT) are variations of sampling-based algorithms [34].

Hybrid path-planning methods

Methods that fall under the hybrid category often combine multiple other algorithms. Examples of this can be the ant colony algorithm or the particle swarm algorithm. Methods like this are more complex but able to overcome some of the shortcomings of the other algorithms. The author deemed algorithms in this category too complex for the scope of this thesis. No further study of these algorithms will thus be conducted.

Algorithm	Advantages	Disadvantages
Focused D*	Correct, complete and optimal	High memory cost
LPA* (Lifelong planning A*)	Able to handle changing edge costs	Always recalculates from same starting position
D* Lite	Simpler to understand and implement compared to D*. Runs at least as fast as focused D*. Able to work with changing starting positions	Does not work in "true" real-time
Field D*	Increment of D* lite, which can smooth the optimal path	Not true "real-time"
Potential field method	Able to operate in more "real-time" compared to the evolutions of D*	Can trap agent in local minima
Velocity obstacles	Able to operate in more "real-time" compared to the evolutions of D*	More specialized towards moving obstacles and faster vessels
Dynamic window	Able to operate in more "real-time" compared to the evolutions of D*	Requires knowledge of the kinematics of the system
Vector field histogram	Able to operate in more "real time" compared to the evolutions of D*	More specialized towards moving obstacles and faster vessels

Table 2: A comparison of the algorithms studied.

1.3.3 Obstacle avoidance

For the CDV1 to be autonomous, the drone has to be able to avoid obstacles in its path. The author will describe these obstacles as changing static obstacles. These obstacles will be static during a single run but might not be present in the next run.

The CDV1 is to be using a single lidar to detect obstacles in front of it. The lidar will then return

a point cloud of detections when in use. A lidar processing computer will then translate the cloud into an object that a pathplanning algorithm can interpret. This pathplanning algorithm will mainly be in use when the USV is traveling between areas, as the primary goal of the USV is not to dodge but instead to circumvent the object narrowly.

1.3.4 Extended object tracking

According to [3] extended object tracking is the method used when tracking multiple measurements at each time step structured around a single object. Figure 4 shows a visualization of this. This tracing method contrasts point object tracing, where each tracked object represents at most one measurement each timestep.

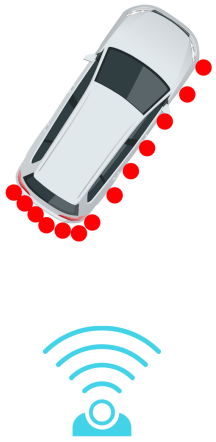


Figure 4: Visualization of extended object tracking.

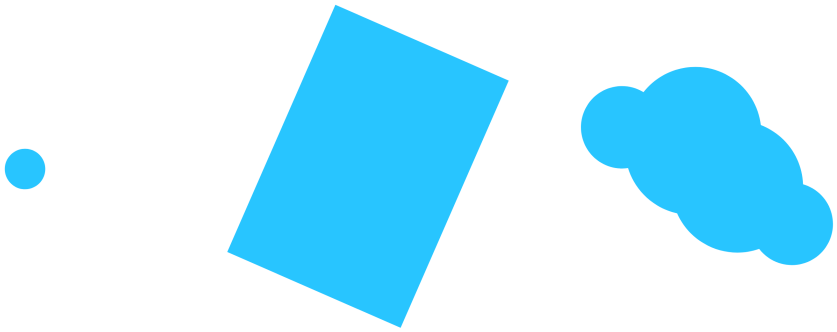
Lidars can return measurements in 2D and 3D, which can be used in extended object tracking. An engineer will then have to consider a trade-off between mapping the measured objects in 2D or 3D. 2D mapping is less complex but returns a less accurate representation of the measured object, while a 3D representation is more accurate and complex. Another option can be to map the 3D measurements onto a horizontal 2D plane. Such downsampling would simplify the modeling and lighten the computational load.

Simplified geometric representations of measured objects can also lighten the computational load. Modeling the tracked object to a shape that can be represented with only a few parameters will drastically decrease the memory required to keep track of multiple objects. Here three different shape complexity levels are proposed by [3], namely modeling:

- No shape.
- Simple geometric shapes like rectangles and ellipses.
- A variety of complex shapes.

In the no-shape complexity model, only the kinematic properties of the tracked object are mod-

eled, and no shape is generated. In contrast, the simple geometric shape level assumes that the tracked object has a simple geometric shape described by a few parameters (e.g. a rectangle or an ellipse). Lastly, in the highest complexity level, we assume that the tracked object can have an arbitrary shape and fit the measured shape to the object. The three different complexity levels are displayed in Figure 5



(a) No shape model. (b) Simple geometric shape. (c) Arbitrary shape.

Figure 5: Visualization of different shape complexities presented in [3].

1.3.5 Travel distance estimation

Improved travel distance estimation is a topic with popularity increasing along with the electrification of cars. This increase in interest is primarily driven by the urge to reduce the range anxiety experienced by drivers of electric vehicles (EV) [35]. Range anxiety is the fear of fully depleting a battery-driven electric vehicle (BEV) battery in the middle of a trip, leaving the driver stranded [36]. This issue also needs to be addressed for the CDV1. Emptying the battery in a run could, in a worst-case scenario, result in loss of the USV.

The travel distance of a USV can essentially be broken down into two parts. First, how much energy can the USV extract from the battery, and second, how long can the USV travel per charge. A paper from the European journal of operations research proposed that the models predicting consumption again can be separated into three different categories; factor models, macroscopic and microscopic [37]. Factor models are the simplest ones. They focus on energy or fuel per distance traveled (e.g. liters/km or kWh/km). Macroscopic models on network-wide emission rates and microscopic models are detailed models measuring instantaneous fuel consumption, often used in high accuracy simulators [38].

To get an idea of how much energy a vessel can extract from its battery, it is worth looking at a battery's state of health (SOH). The capability to store energy and provide a specific power decreases over the battery life because of aging [39]. Therefore, the state of health says something about the maximum amount of energy that can be extracted from the battery. This estimation can, in turn, be compared to the energy capacity of a new battery of the same type to see how much the maximum travel range has shrunk.

Most research on continuous EV range estimation is performed on cars, but the methods used can be transferred onto electric USVs. A study done on 50 electric cars over 1100 trips in 2018 suggested that 94% of the variation travel distance per charge could be explained by four different parameters[40]. These were:

1. Initial state of charge (SOC)
2. Trip distance
3. Speed
4. Temperature

Therefore, a simple regression model could be a good tool to estimate the remaining travel range.

1.4 Contributions

This section sums up the main contributions of the thesis. The results are extensions of previous work [1].

1. A ROS (2.7) implementation of the waypoint generating algorithm onboard the CDV1.
2. Integration of static object avoidance into the waypoint generating algorithm.
3. A evolved simulation environment that better matches the implemented system better.
4. An improved method for predicting the remaining range of the USV.
5. A literature review on autonomy levels, path-planning methods, range estimation for EVs, and extended object tracking.

1.5 Organization

The report is organized as follows:

- **Section 2:** A section explaining preliminary work and knowledge.
- **Section 3:** An in-depth presentation of Cleaning Drone design.
- **Section 4:** Implementation of an advanced path planning algorithm and obstacle avoidance.
- **Section 5:** A presentation of the pathplanning system implemented onto the Cleaning Drone.
- **Section 6:** A section presenting and discussing the results.
- **Section 7:** A conclusion summing up the findings and presenting possible future work.

2 Preliminaries

This chapter aims to present valuable preliminary knowledge that the reader should be familiar with before reading this report. Additional elements included in this chapter are a summary of the waypoint generating guidance algorithm developed by the author in the specialization project and the relevant theory behind the systems and formats used in this project. Assumptions made will also be presented in this chapter. The structure of the chapter is as follows:

- **Section 2.1:** Trash accumulation in harbours.
- **Section 2.2:** Assumptions regarding the USV environment.
- **Section 2.3:** The simulation environment.
- **Section 2.4:** Splitting the USV's workarea into fields.
- **Section 2.5:** The optimal path for trash collection.
- **Section 2.6:** The NMEA Format.
- **Section 2.7:** Robotic operating system (ROS).
- **Section 2.8:** Travel range estimation.
- **Section 2.9:** Estimation of remaining trash-bin capacity.

2.1 Trash accumulation in harbours

As mentioned in the introduction is ocean trash becoming a significant threat to marine life. Most of this debris stems from land, with a majority coming from urban areas [11]. Environmental forces then carry the trash out to sea. When at sea, the trash is dispersed over large areas, vastly increasing the effort required to collect it. We can prevent this dispersion by collecting trash before it leaves our shorelines.

The trash in harbours tends to be more affected by wind, waves, and tides compared to currents [41]. As a result, this report assumes that currents have close to little effect on the displacement of trash in harbours. Trash dispersion due to currents will therefore be neglected in this report. The trash does, however, tend to end up in hotspots, especially close to land, as a result of environmental forces [42], [12]. These hotspots will vary from location to location based on the weather and can often be found with a biological litter binding the trash together into a more extensive, intertwined structure. Figure 6 displays a picture of this

2.2 Assumptions regarding the USV environment

This thesis mainly regards the implementation of the path planning algorithm generated by the author during the specialization project. Many of the assumptions made in the specialization project will not hold and needs to be circumvented. However, some assumptions will be kept to make this task manageable. These assumptions are:

Figure 6: A trash hotspot.

1. There are no moving obstacles in the workarea of the USV.
2. There is no ice in the workarea of the USV.
3. There are no waves or bad weather in the workarea of the USV.

The first assumption can be considered reasonable as there is little traffic in the workspace selected for this case study (Section 1.2.1). Since most traffic in the area is manoeuvrable, most boats can easily steer clear of the USV. The no ice assumption is added as this will increase the complexity of the task as ice results in more resistance and limited movement. Lastly, the author finds it reasonable to assume that the CDV1 will not operate under challenging weather conditions. Thus this system will be created for operation in nice weather with waves of no more than one meter.

2.3 The simulation environment

Not everything is possible to implement/test on the cleaning drone in its current state. A simulation environment will therefore be used to test some of the functionality of the guidance system. The author created this python-based simulation environment during the specialization project.

The simulation environment consists of the map seen in Figure 7. A custom drone object moves between coordinates input to the system and records its movement in this environment. White areas of the map mark the area the drone is allowed to move in, and the black pixels on the map are obstacles. Lastly, an area that the drone has visited will be coloured gray. It is worth noting that the USV will not be stopped from passing onto "illegal" areas in the simulation. However, there is an option to check if a given coordinate is allowed. If movement into an "illegal" area has occurred, it can be seen in the visualization displayed following the simulation.

The simulator is implemented to mimic the control system of the actual drone as closely as possible. The map created uses universal transverse Mercator (UTM) coordinates of the show area as its axes values. UTM axes allow the guidance system to set a coordinate as a goal position, and this position will be the same in the simulator and the real world.

The simulator is based on a couple of assumptions worth noting. Namely:

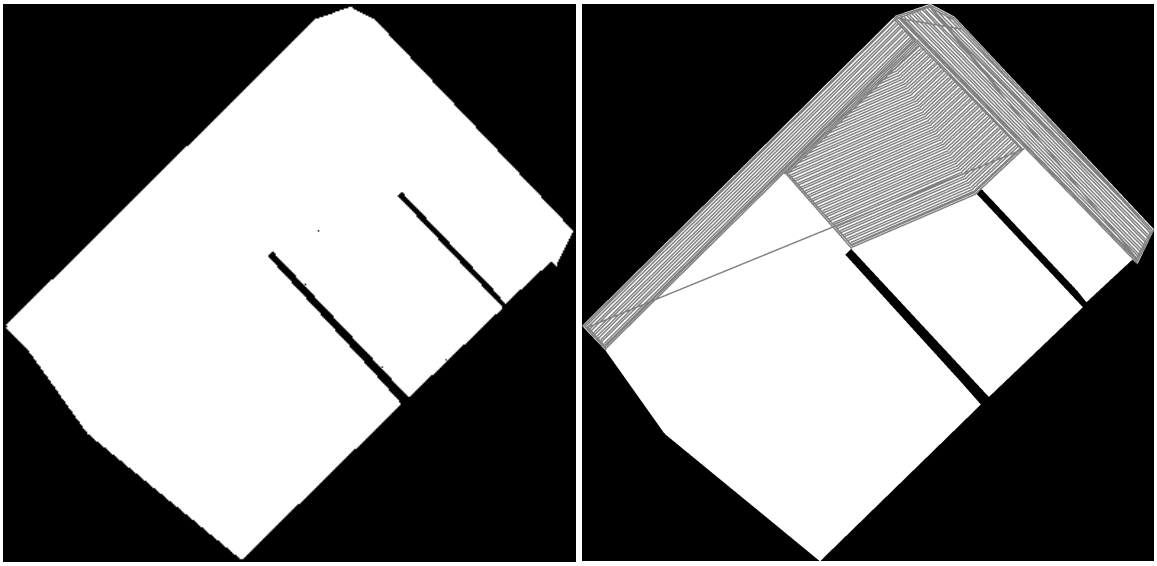
- The USV will not experience any disturbances from the environment
- The USV moves at a constant speed of 1.5 m/s
- The USV has a range of 100 km, and its battery level is an estimate of its remaining range.
- There are no moving obstacles in the area.
- There are no obstacles other than those already marked unless manually added.

These assumptions will not hold in an actual use case. However, the simulator helps predict calculation times and see if the different algorithms return the correct coordinates.

2.4 Splitting the USV's workarea into fields

Since the environment, human behaviour, and other factors create a nonuniform distribution of trash in the harbour, it is reasonable to split the USV's workarea into smaller subareas referred to by the author as fields. These fields are created based on the shape of the harbour, the common hotspot locations, inflowing rivers, and distance to land. The fields are also for the simplicity of cleaning, not too complex either, and preferably have a convex shape (Definition 1). This type of shape results in the CDV1 having to make fewer zigzag moves when it cleans a field.

It will be natural to generate unique fields for the different banks and areas with a river exit. It is also helpful to use the knowledge of the local harbour employees, as they will have a feeling of where trash hotspots tend to occur and how the trash flows in response to the weather. The size of these areas will vary, and it might be challenging to say what the proper resolution of a workarea should be, as this will vary from workarea to workarea. However, one should try not to create too small areas, as this will increase the complexity of the cleaning without increasing efficiency. Too large areas will also be sub-optimal as the difference in trash density between the areas will shrink, reducing the optimization value. In addition, the USV should not use more than 90% of the battery



(a) Edge map, marking movable area in the simulator. (b) The edge map partially covered.

Figure 7: The simulator input (left) and output (right).

for area cleaning. This energy retention ensures that the USV has enough charge to make the return trip to the charging station. Figure 8 shows the field creation for this case study.

Definition 1 *Convex (of a polygon): a polygon each of whose angles is less than a straight angle [43].*

2.5 The optimal path for trash collection

As part of the authors' specialization project, an algorithm was created to generate the optimal path for collecting trash in a given workspace. This algorithm works by taking in a set of areas, each with an associated cost and value. This can be seen in Table 3 and Figure 8 for the workarea described in Section 1.2.1. These areas were created based on the shape of the harbour and their proximity to land. The cost represents the amount of energy the USV uses to clean the given area, while the value represents the amount of trash the drone expects to collect in the same area. The current trash probabilities were picked at random as an example in this case. However, in a real use case, they will be set together with the harbour owner based on where trash historically has accumulated. The cost of covering an area was estimated by cleaning the given area in the simulator created during the specialization project.

Given the set of areas, the drone could then use a solver for the knapsack problem to select the set of areas which yield the highest amount of trash without exceeding the battery limit. When a selection of areas is picked, a solver for the travelling salesperson problem is then run to minimize the time travelling between the selected areas. The drone could then clean each area using a method

Area	Colorname (matplotlib)	Trash probability per pixel	Covering cost (battery %)
1	Cornflowerblue	0.1	24.94
2	Maroon	0.2	3.33
3	Yellow	0.1	12.33
4	Green	0.08	35.62
5	Hotpink	0.2	19.73
6	Cyan	0.1	12.58
7	Chocolate	0.1	11.03
8	Indigo	0.03	39.68
9	Greenyellow	0.08	23.44
10	Blueviolet	0.01	35.31
11	Darkorange	0.01	35.26

Table 3: The different areas, their colour, trash probability, and covering cost.

borrowed from agriculture.

The algorithm used when cleaning a field starts by performing a headland pass or pass around the edge of the desired area (see Figure 9 for visualization). Following the first headland pass, the USV performs two more to create some cleaned space on the edge of the area for the drone to turn on when covering the middle section. The first headland pass is often the most challenging part of the entire cleaning process, as the USV will have to circumvent potential parked boats or other obstacles while performing tricky turns in corners.

The USV covers the middle section of the area by making multiple zigzag routes along the longest curved edge (see Definition 2) until the entire area is cleaned. It then moves over to the next area. Figure 10 displays a simulated example of complete optimal coverage.

2.5.1 The first headland pass

This report assumes that there will be no moving obstacles in the way of the USV. This assumption, in turn, means that the only obstacles the USV will have to account for are static. In most scenarios the USV will encounter, these static obstacles will occur during the first headland pass as parked boats, stray rocks, Etc. In contrast to many standard pathplanning algorithms, the obstacles are not to be avoided with a clear margin but instead narrowly circumvented.

The first headland pass is also significant because it is when the USV is closest to land and at constant risk of colliding with the shoreline. There is also some added difficulty in turning this close to land, especially corners. All of these risks will have to be mitigated. Luckily the CDV1 is to be equipped with a *Velodyne Puck-16* Lidar [44], which can detect obstacles in 360 degrees around the USV. In combination with the GNSS, this sensor will be the primary tool to allow the USV to perform the first headland pass of a given area safely.

The first headland pass will mainly consist of the USV going in a straight line from its current position to its desired position and then using the lidar to keep a constant distance from walls and obstacles. A path planning algorithm will not be needed to a significant extent here as the

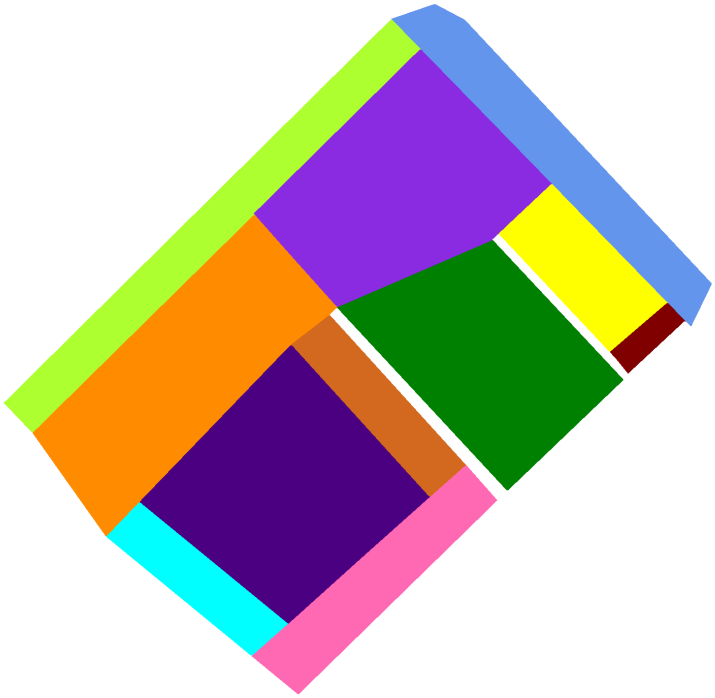


Figure 8: The current workarea split into areas.

primary goal is to follow the edge rather than reach the goal in the shortest distance possible. Since the drone will always perform the headland pass counter-clockwise, a simple solution could be to always follow the path to the right when in doubt after discovering an obstacle.

2.5.2 Consecutive headland passes

The algorithm creates waypoints for consecutive headland passes based on the area covered during the first headland pass. Here, it is assumed that the obstacles present during the first headland pass are still present during the consecutive once. The algorithm can therefore take the outline of the shape created in the previous headland pass and scale it down so that the new route is one "drone"-width from the previous headland pass (see Figure 9 or 11 for visual example). The waypoints for the consecutive passes are then created by scaling down the shape created by the first headland pass and creating a waypoint at each corner. The USV should hopefully not depend on the lidar for

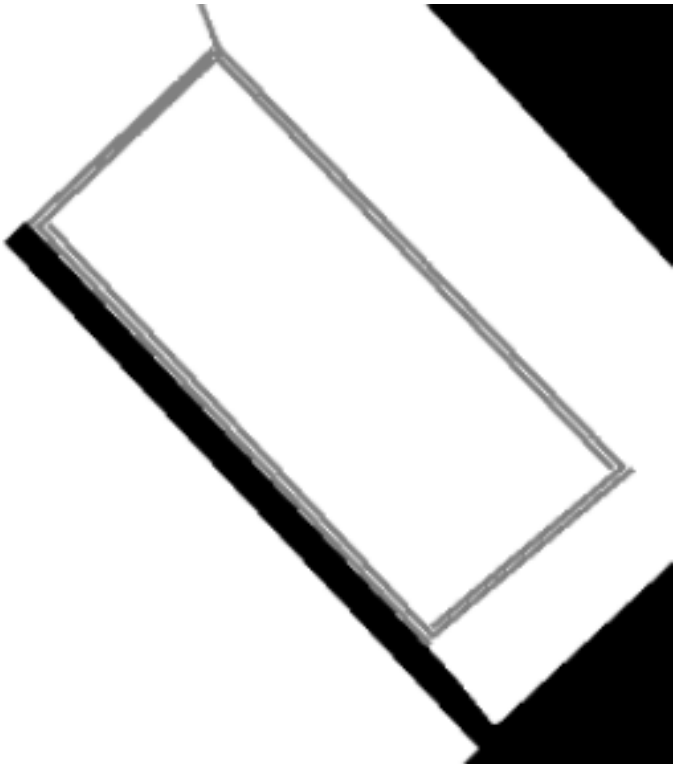


Figure 9: 3 headland passes displayed on area 3 in the simulator.

these passes as the area it passes has already been covered.

2.5.3 Zigzag movement

In order to clean the rest of the area, the USV will move in a zigzag pattern along the longest curved edge Definition 2. Here, the USV will rely mainly on the GNSS but continuously scan with the lidar if an obstacle occurs.

Definition 2 ***Curved edge****: A curved edge is a collection of sequential straight edges satisfying the criterion that the angle between two successive edges is less than or equal to a threshold angle [45].*

2.5.4 Movement between areas

The USV has to move over to a new area whenever an area is cleaned. The goal in this stage is to get as efficiently as possible to the next area without hitting any obstacles. At this stage, we no longer need to circumvent obstacles narrowly, and thus the different pathplanning algorithms presented in Section 1.3.2 would be more applicable.

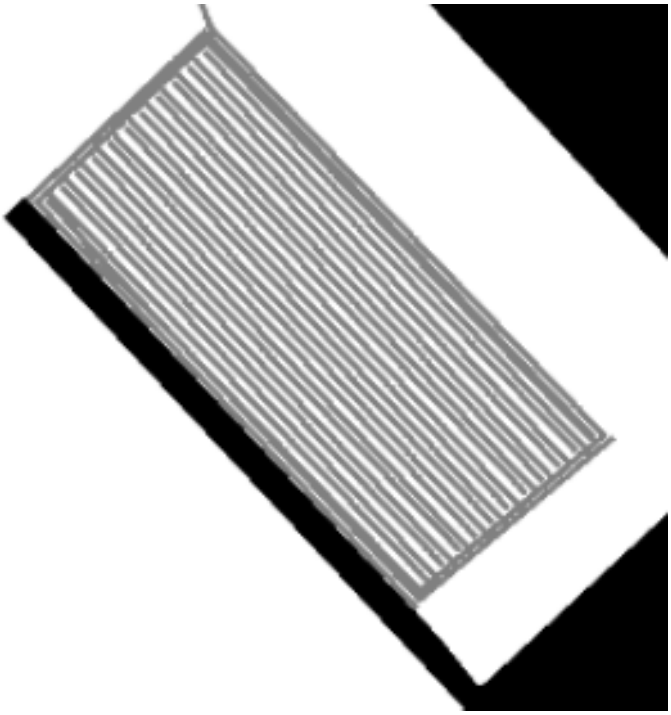


Figure 10: The optimal coverage of area 3 displayed in the simulator.

2.6 The NMEA Format

A standard format for shelf marine electronics to communicate with computers is the standardized NMEA (National Marine Electronics Association) 0183 format [46]. The NMEA 0183 standard uses a simple ASCII serial communications protocol and has a checksum included in each message for error detection. Absolute position, velocity and time information are included in this format. GNSSes send information over many lines, called sentences in the NMEA format. Each sentence is self-contained and independent of other sentences [47].

The NMEA format is also used by the GNSS and the onboard computer (OBC) on the cleaning drone V1. The GNSS returns time and position data as NMEA sentences, and the OBC also expects movement commands for the USV to come as NMEA sentences. The standardization of the messages sent to and from the OBC makes it easier to scale the system and add new and different components in the future.

2.7 Robotic operating system (ROS)

The author has chosen to create the guidance system in ROS [48] when implementing and expanding the waypoint generating algorithm onto the USV.

Robotic operating system, or ROS, is an open-source structured communication layer created to

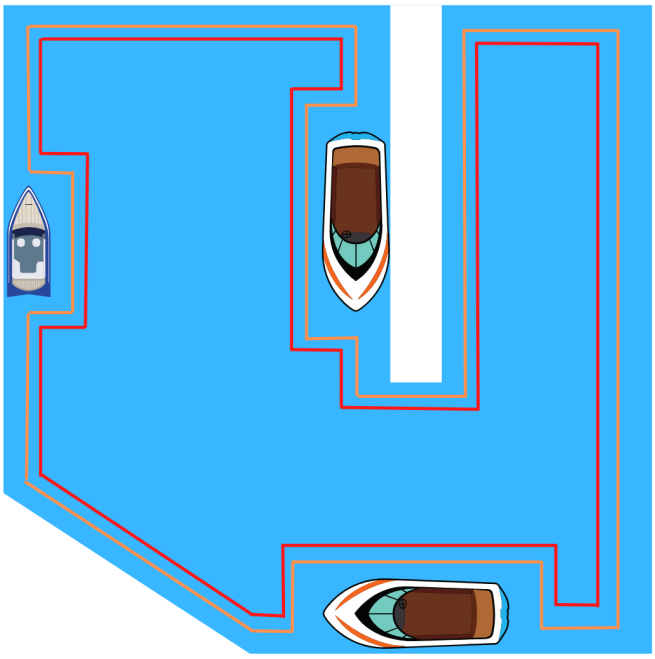


Figure 11: The path of the first headland pass (orange), and a consecutive one (red).

make modular, tools-based software development across different hardware more straightforward and robust [49]. By using ROS, the author can use preexisting and tested packages and drivers for the different hardware, which this project has to consider. ROS also facilitates the possibility of a more modular design through its nodes and packages. This modularity can again come in handy in terms of robustness and system scaling in the future.

ROS employs two different methods to communicate between the different ROS packages and modules, namely topics and services. According to the ROS wiki, Topics are named buses over which nodes exchange messages. Therefore, a node can publish and or listen to a given topic to add or fetch data. The wiki describes Services as a request/reply action. A node can send a request containing some data to a service. The service then performs its dedicated action on the data before sending a reply.

Being language agnostic, ROS enables the programmer to work with the desired programming language. This ability eases the transition of adapting the existing waypoint generation algorithm to the USV as it does not have to be rewritten in a new language.

ROS1 was chosen here over ROS2 [50] as the ROS1 framework is more mature. The ROS1 support community is larger, and it is the ROS version the engineers at Maritime Robotics were most familiar with. ROS2, on the other hand, tackles a lot of the issues with ROS1, mainly the ones regarding real-time, safety, certification, and security [50]. ROS2 is, in short, the next evolution of

ROS and will have to be adopted in the future as support for ROS1 ends in 2025. However, In terms of current support, it is more helpful to be found by using ROS1, and it was therefore chosen.

2.8 Travel range estimation

As for most other modern electric vehicles, good travel range estimation is also an issue for the CDV1. For the USV to work as efficiently as possible, the drone must accurately predict how far it can travel without being stranded. State of Charge (SOC) is, in these cases, insufficient for driving range estimation as battery-driven electric vehicles are complex combined systems [51]. There are many factors influencing the remaining driving range of the CDV1. Examples of these are:

- Thrash collected.
- Waves.
- Wind.
- Behavior of other vessels in the area.
- The path the USV takes.

An increase in the amount of trash collected will increase the drag of the CDV1 as its trash bin fills up. The introduction of waves will also change the motion of the CDV1. This change of motion will result in the CDV1 having to increase its power output to keep its current motion. The effect of wind is not that critical compared to current and waves because of the low density of air. For high-speed wind, however, it should be considered in ship manoeuvring [52].

The behaviour of other vessels in the area will also induce waves and force the drone to take action to avoid them, again leading to a reduction in range. Lastly, the path of the vehicle plays a role. This is because acceleration costs energy to perform, and numerous changes in acceleration drain the battery.

Most research on travel distance estimation performed on electric vehicles is done on cars. This trend is likely due to the relatively high interest in electric road vehicles compared to electric sea vessels. Another reason might be the relatively short range of electric ships compared to combustion engine ships. Limited travel distance and capacity are major restraints of full-electric ships [53]. The price of battery packs required for a ship size vessel also becomes infeasible expensive [54]. As per the author's findings, little research has been done on travel range estimation for smaller sea vessels.

2.9 Estimation of remaining trash-bin capacity

Currently, there is no way of directly measuring the amount of trash collected by CDV1 as the USV does not contain an appropriate sensor. The remaining bin capacity can, however, be estimated. By assuming that the drag in the surge direction will increase with increasing levels of trash collected, one can estimate the amount of trash collected by estimating this additional drag. The USV will always try to keep its commanded speed, and by measuring the motors' power output, this drag increase can be extrapolated.

A simple regression could be sufficient with enough data. By comparing known values of power usage when the bin is full and at different capacity levels against an empty bin, a continuous

estimation of the remaining capacity of the trash bin can be created.

3 The cleaning drone system

A part of implementing a new guidance system on a robot is to see how well the guidance system performs with the specific robot in question. The system can be pretty general or made explicitly for that given robot. As the system created in this report aims to optimize the trash collecting capabilities of the Cleaning Drone V1, it can be regarded as a specific system. Therefore, the author will strive to model the guidance system as best fit as possible for the given drone.

Another possibility that opens up due to the guidance system and drone being developed in parallel is to evaluate how well the current USV works with the created guidance system. Rather than changing the guidance system, it might be to change the drone's design. This development aims to create the most optimized system possible for collecting trash in harbours.

In this chapter, the design of the USV will be presented. Tests for checking the compatibility of the drone and the guidance system will also be presented. Lastly, some tests enabling the author to evaluate the current collection systems design will also be created in this chapter.

- **Section 3.1:** The current cleaning drone.
- **Section 3.2:** Further development of the Cleaning Drone V1.

3.1 The current cleaning drone

The USV used by Clean Sea Solutions (CSS) was briefly introduced in Section 1.2.2. The cleaning drone is a retrofit of the Maritime Robotics Otter platform. The platform was selected because the Otter is a well-tested framework with tested and working autonomy features. It includes both an operational control and navigation system. The use of this platform as a base framework enables Clean Sea Solution to focus entirely on the guidance system of the CDV1.

The Otter has a dimension of approximately one times two meters; exact measurements can be found in Appendix A. The drone's light weight and small size make it possible for two persons to lift and handle manually. Being a catamaran, the Otter also allows for 360° turns on the spot, thus increasing its mobility. The catamaran design also enabled CSS to mount a trash collection system between the pontoons, an image of this can be seen in Figure 13. Being all-electric and using multiple off the shelf components, the otter platform fits the green image Clean Sea Solutions wants to project while at the same time making part changes more straightforward. Its pontoons are made out of the plastic polyethylene, making them durable and able to take a hit. A rendering of the otter platform can be seen in Figure 12

As the Otter platform is a general-purpose platform, it was created with the intention of each customer being able to retrofit it following their needs. For CSS, this was to mount a trash bin between the pontoons so that the USV could collect trash it came across. The backplate of this trash bin is a metal plate with multiple holes borrowed through it to reduce the drag of the bin. This plate

Figure 12: The Maritime Robotics *Otter* platform. source:(www.maritimerobotics.com/otter).

is also connected to a linear actuator. The actuator can then push the backplate out the front of the bin to dispose of collected trash. The bin also comes with a door that can be opened and closed at command using another linear actuator. This door prevents trash from escaping whenever the USV has to move in reverse or do sharp turns.

Currently, the USV is included with the Hikvision bullet camera [55] and a dual antenna GNSS heading and position system for sensing the environment. There is also an option to mount a LIDAR, but the supplier has not been able to deliver it yet due to the ongoing global electronics shortage. These components are mounted on the *Otter* "Targa" or tower to get the most expansive field of view. An image of the Targa can be seen in Figure 14.

3.2 Further development of the Cleaning Drone V1

The CDV1, in its current stage, can autonomously sail around and collect trash in open areas. However, the drone cannot be trusted to circumvent obstacles narrowly. This distrust is due to inaccuracies in the GNSS and the maps given to the system. For the CDV1 to get these capabilities, a lidar will have to be used. This lidar, the Velodyne Puck, will continuously scan for obstacles in the USVs proximity. Its outputs will be transferred and processed in a local processing computer onboard the Otter before passing the data to the guidance system.

It is also worth noting that the cleaning drone used when testing the guidance system created in this report is a first generation product or a minimum value proposition (MVP). During the test phase of the guidance system, the design of the cleaning drone will also be extensively tested. This testing will be performed by running the USV over a more extended period, at different speeds, and in changing weather. The author will observe and log the USV behaviour., The data produced by the drone will also be logged by the author.

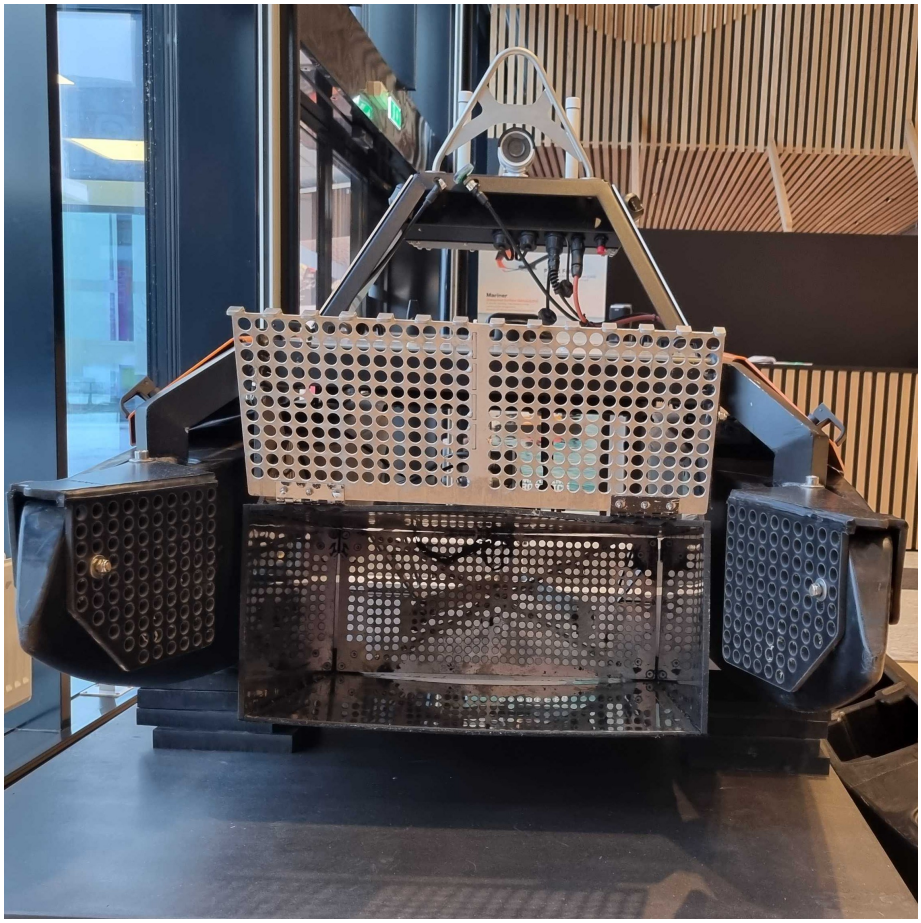


Figure 13: The CDV1 trash bin mounted between the pontoons of the USV.

The drone's ability to collect trash, both in general and from "hotspots" (Section 2.1) will also be evaluated. For hotspots or clustered trash, the author will manually operate the USV into the hotspot and see if it can collect a desirable amount of trash. For non-clustered trash, this evaluation will happen through regular area coverage and just seeing how the drone responds to trash that comes across its path.

Figure 14: The *Otter* targa.

4 Optimal path planning and obstacle avoidance

This chapter concerns the area to area movement and obstacle avoidance. The CDV1 is an advanced system in continuous development, resulting in some parts of the system having to be implemented and tested in the simulator described in Section 2.3. This chapter describes how area to area movement and obstacle avoidance was done for the previously described system, and the chapter is structured as follows:

- **Section 4.1:** Path planning between areas.
- **Section 4.2:** Obstacle avoidance.
- **Section 4.3:** Improving the original pathplanner.
- **Section 4.4:** Experimental setup.

4.1 Path planning between areas.

The D* Lite [5] algorithm by Sven Koenig, and Maxim Likhachev was selected as the best path-planning algorithm to handle area to area movement for our current use case. Its pseudocode can be seen in Appendix C. This choice of algorithm comes from the algorithm working well in closed environments as the algorithm keeps a map of the area containing the cost of moving between different cells. Another advantage is that the onboard computer can perform most of the required calculations before initiating movement. Only a couple of calculations are then required when an obstacle is detected and the map is updated. The algorithm does not suffer the drawbacks of being stuck in a local minimum. Thus the USV should always be able to reach its goal if a path exists.

When selecting an algorithm, it came down to a decision between D* lite and Field D*. D* Lite came out on top here as Field D* used 1.7 times as long to generate a solution as D* Lite [27], [56]. Field D* solutions were on average 96% as long as the solutions generated by D* Lite, but this small path optimization did not outweigh the increase in computation time.

4.1.1 The D* Lite algorithm

D* Lite is a path-planning algorithm that combines incremental and heuristic search to find a path from a dynamic starting node to a fixed goal node. The algorithm is an incrementation on lifelong planning A* (LPA*), also developed by Koenig and Likhachev. In contrast to A*, LPA* forwards previously calculated information about the path from search to search [5]. This feature enables the algorithm to recalculate a route faster if a blocking obstacle is detected.

D* lite builds upon LPA* but only considers the path between its current and goal position when calculating a new path. This calculated path reduction is done by revering the search direction. While traditional A* and LPA* calculate a path from the start to the goal, D* Lite calculates a path from the goal to the agent's current position. This optimization leads to fewer vertexes being

expanded, resulting in faster replanning compared to LPA*.

4.1.2 Creation of the initial path-planner

The D* Lite algorithm implemented is based on the pseudocode in [5]. To implement this code, the author first implemented and tested the pseudocode for LPA*, also found in [5]. When the LPA* code worked, it was further iterated to become D* Lite. An example of a calculated path can be seen in Figure 15

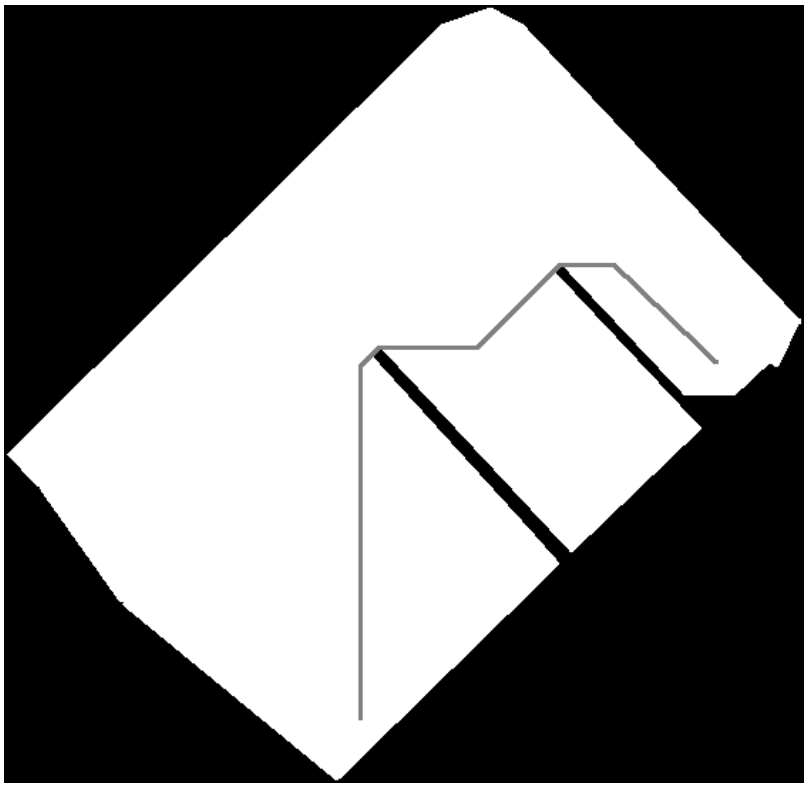


Figure 15: The path created by D* Lite from one end of the map to the other.

The initial implementation of the algorithm took a significant amount of time to run. Thus, a downsampled version of the workspace map was used to calculate the optimal path between two areas. This downsampled map enabled the algorithm to run faster compared with using the original resolution. Each cell in the map thus went from representing approximately 25cm times 25 cm in the real world to represent a 40 cm times 40 cm area. The reason behind this choice is that the map does not need to be entirely correct when planning the initial route, as the algorithm is optimized for handling potential obstacles it might encounter. Therefore, there is little say if these obstacles come from a new element blocking the path or from slight inaccuracies in the map. Reducing the number of tiles resulted in the algorithm expanding fewer vertexes to find its goal. This optimization

again leads to the algorithm finding a suitable path faster. A conversion algorithm can then be used to map the coordinates of the different maps up against each other to find the equivalent real-world position.

4.2 Obstacle avoidance

As described in sections 1.3.3 and 1.3.4, extended object tracking based on inputs from a lidar can be used to avoid obstacles while the USV is moving between areas. However, extended object tracking is most useful when the system intends to track a moving obstacle. In those cases, it is helpful to estimate the extent of the object in order to predict which cells it will occupy in the coming timesteps. Since it is assumed in the current case study that all obstacles are static, there is no need to perform extended object tracking.

While the USV is moving, the lidar will continuously scan for known and unknown obstacles. Discovered unknown obstacles will block the cell they are occupying. If these cells were to be a part of the current path of the drone, the path would be recalculated using the replanning step of the D* Lite algorithm described in the previous section.

A simulated lidar was added to the simulator since it was not possible to test with an actual lidar in the timeframe of this thesis. The simulated lidar works by scanning the area surrounding the drone in 360° field of view for each new cell the drone enters. The simulation scans the area by inspecting a line of tiles at a fixed angle originating at the drone and pointing outwards to mimic an authentic lidar as close as possible. The lidar then returns the tiles with detected obstacles. Pseudocode for the lidar scan can be seen in Algorithm 1. The current scanning range is set to 20 tiles on the down-scaled map or roughly 8 meters in real life.

4.3 Improving the original pathplanner

As mentioned in Section 1.3.2, there exist multiple different incremental improvements to the original D* lite algorithm. The algorithm can plan an initial route and replan if new obstacles are detected. Using this path planner, the USV can always find a path to its goal if one exists.

Multiple different improvements have been evaluated. However, the author decided to implement some of the improvements presented in [4]. These improvements were chosen based on their ability to negate some of the current implementation's drawbacks.

The algorithm in its current state does not consider the size of the USV. It also plans paths close to detected obstacles. When not cleaning the area, this should be avoided. There is, in addition, room for speeding up the time it takes to calculate a new path. Some smoothing of the path should also be added.

4.3.1 Adding padding to obstacles

[4] presents a solution to avoid sailing close to obstacles. In [4], the authors are tasked with sailing a USV carrying sensory equipment in an unknown area to collect water samples. The scenario made it risky for the USV to sail close to obstacles in fear of damaging the sensors. Therefore, the solution presented was to add a small amount of padding to the obstacles before calculating the route.

Algorithm 1 Lidar Scan

Input $drone\_position$, map, max_radius, $beams$, $cell\_size\_irl$
Output $blocked\_tiles\_absolute$, $blocked\_tiles\_relative$

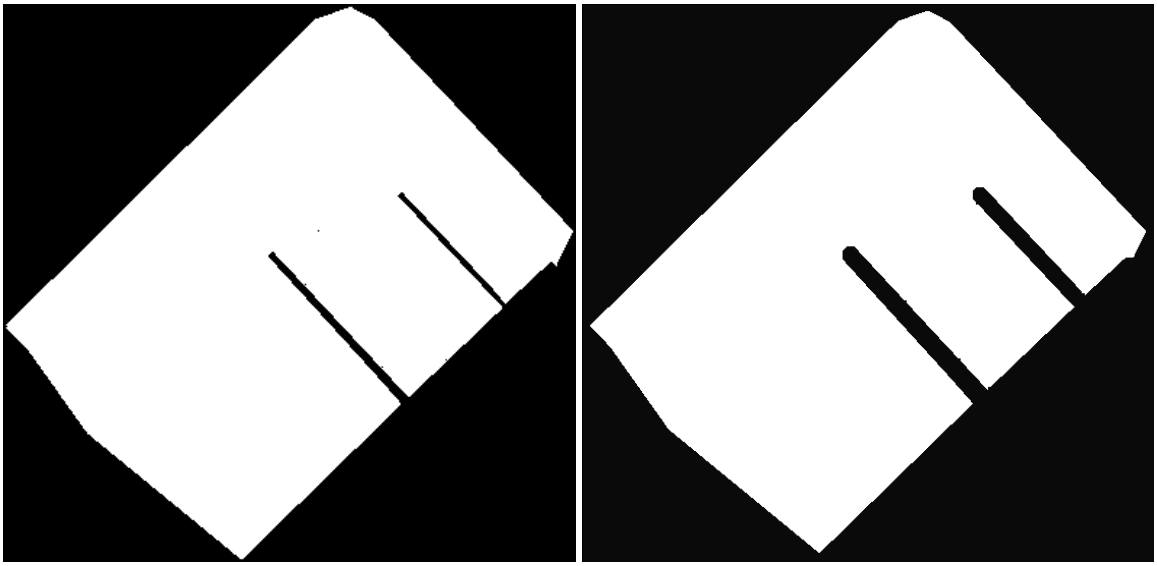
```
function LIDAR SCAN
    Blocked_tiles_absolute = []
    Blocked_tiles_relative = []
    Dist_between_beams ← 2 * π/beams
    X ← drone_position[0]
    Y ← drone_position[1]
    for beam_nr = 0; beam_nr ≤ beams; beam_nr++ do
        angle = dist_between_beams * beam_nr
        opposite_multiplyer = sin(angle)
        adjacent_multiplyer = cos(angle)
        for radius = 1; radius ≤ max_radius; radius++ do
            position = (X + int(rad * adjacent_multiplyer), Y + int(rad * opposite_multiplyer))
            if map[position] = 0 then /*tile blocked*/
                blocked_tiles_absolute ← position
                blocked_tiles_relative ← (int(rad*adjacent_multiplyer*cell_size_irl),
                                          int(rad*opposite_multiplyer*cell_size_irl)
                break
            end if
        end for
    end for
    blocked_tiles_absolute = remove_duplicates(blocked_tiles_absolute)
    blocked_tiles_relative = remove_duplicates(blocked_tiles_relative)
    Return blocked_tiles_absolute, blocked_tiles_relative
end function
```

Adding padding implies making the obstacle virtually larger and making the path planner plan a path that keeps a minimum distance to known objects. The padding will keep the USV from sailing close to obstacles and risk unnecessary collisions. The method will be implemented in the D* Lite version used in this thesis. The pseudocode presented in Section 3 in [4] will be used to add the padding to the map used in the authors' implementation of D* Lite. A comparison of the known areas with and without padding can be seen in Figure 16

For the current case study, padding also needs to be added to unknown obstacles. This padding is added as the obstacles are discovered. Here, the author used the same method as when padding known obstacles. However, an assertion was added to the function to reduce the padding size in case the USV was within the padding range / safe distance of an obstacle. This padding reduction prevents the USV from being trapped in virtual padding.

4.3.2 Expanding more adjacent nodes

When searching for the optimal path, the USV evaluates the eight nodes adjacent to it before moving into the node with the lowest cost. Only evaluating eight nodes means that the USV has a minimum



(a) The map given to the USV without padding. (b) The map given to the USV with padding.

Figure 16: The map the USV uses to navigate with and without padding.

steering angle of 45° when it sails between areas. A high minimum steering angle can lead to a rougher path than necessary as the USV has to perform large turns. [4] suggests increasing the number of nodes evaluated when selecting the next node to visit.

By increasing the number of evaluated nodes to 24 or 48, as depicted in Figure 17, the number of search directions also increases. This increase in search direction, in turn, leads the USV to have more steering choices when planning a path. As seen in Table 4, increasing the number of search directions also decreases the minimum steering angle, which in turn can lead to a smoother path.

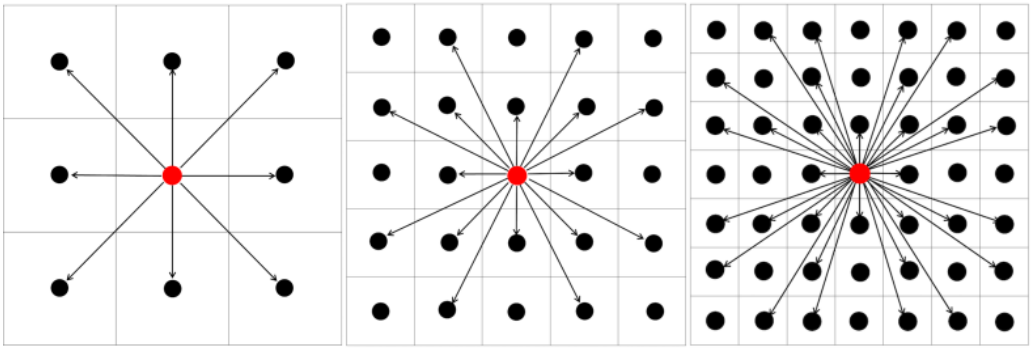
Number of Adjacent Nodes	Number of Search Directions	Minimum Steering Angle
8	8	45°
24	16	22.5°
48	32	11.25°

Table 4: Relationship between adjacent nodes, search directions and minimum steering angle. Source: [4].

4.3.3 Reducing computation time

Multiple different steps can be taken to reduce the algorithm's computation time. One example, as previously presented, is reducing the resolution of the map given to the algorithm. However, this section will focus on changes that can be made to the code to speed up the algorithm's runtime.

One suggestion by [4] is to switch out the conventional priority queue (array-based) used in



(a) 8 adjacent nodes (b) 24 adjacent nodes (c) 48 adjacent nodes

Figure 17: Adjacent nodes for search. Source: [4].

D* Lite with a min binary heap. A visual example of a binary heap can be seen in Figure 18. This binary heap should decrease the algorithm's worst-time time-complexity for the pop and the search operation. However, the time-complexity of inserting (pushing) new nodes into the queue increases. [4] argues that this is of no issue since the time complexity, on the whole, is reduced. The time complexities for some common actions for a conventional (array-based) priority queue and a binary-heap-based priority queue is listed in Table 5. It is also worth mentioning that the algorithm focuses on expanding nodes with high priority, meaning that most elements inserted in the queue will have a higher priority (lower value) compared to the elements already in the queue. Thus, moving the time complexity from the pop action to the push action could be advantageous in this use case.

Operation	Time Complexity conventional priority queue (s)	Time Complexity binary heap (s)
Pop up	O(n)	O(1)
Push in	O(1)	O(log_n)
Search	O(n)	O(log_n)

Table 5: Time complexities using priority queue vs using binary heap according to [4].

4.4 Experimental setup

To test if the replanning step works in the simulator, the lidar signals will be simulated using the function described in Section 4.2. The first map, as shown in Figure 19 is the "forbidden area" that the system is aware of, while the second map is the "unknown parts of the environment" only the lidar will be able to detect. As the lidar detects obstacles, the path will, in turn, have to be recalculated. In order to achieve this, the simulator had to be redesigned.

Previously the simulator had been given a set of waypoints created by the path planner and moved the USV between these given waypoints. In order to mimic the guidance system working

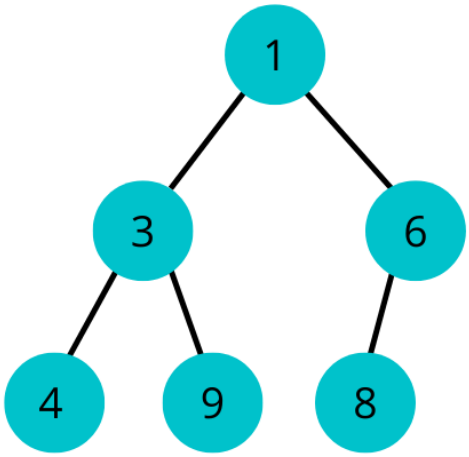
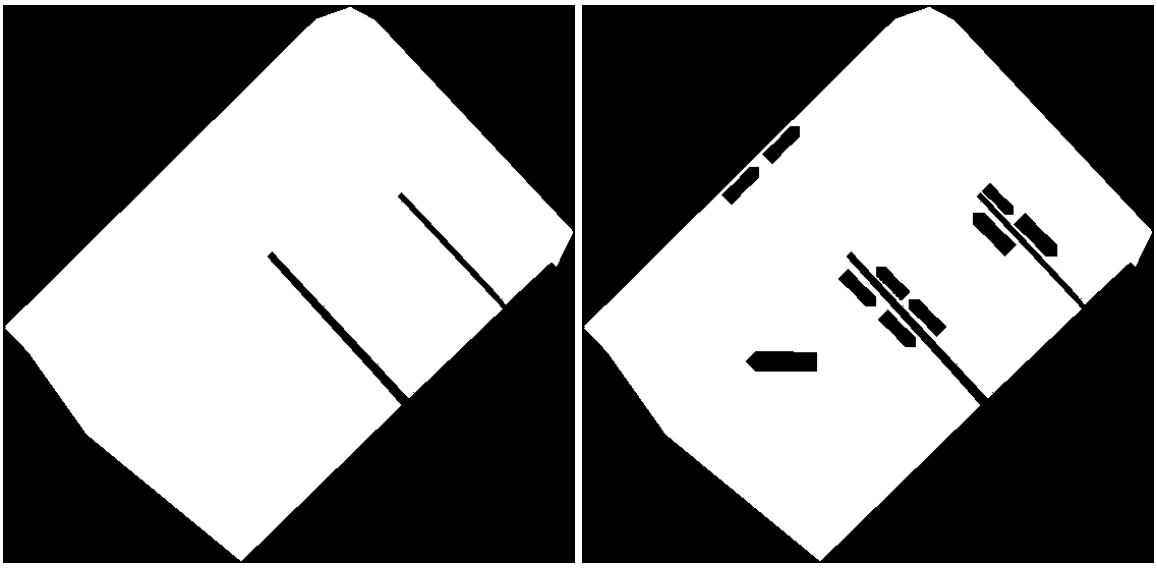


Figure 18: Binary heap example.

in parallel with the control of the USV, the simulator was extended to also scan for changes in the environment as it moved (e.g. cells suddenly being blocked). If a change were discovered, the movement would stop, and the path planner would calculate a new route.

It is worth noting that the D* Lite algorithm at this stage only is used when traveling between areas. For the remainder of the time, e.g. area cleaning, the USV moves directly between coordinates.

Changing the simulator to fit the execution of this algorithm also changes the way the simulated drone is controlled. This new way of controlling the simulated drone has the advantage of being more similar to how the real CDV1 is controlled. In the new simulator, all functions controlling the movement of the drone will be moved into the drone-movement class seen in Appendix D. The path planner will work solely with UTM coordinates instead of array indexes that it previously relied on. In this way, by having the coordinates returned by the path planner and the D* Lite algorithm in the UTM format, fewer changes have to be done to the system when it is implemented in ROS.



(a) The fixed features of the harbour which the system is aware of.

(b) The harbour with unknown features (boats) added. Only the lidar is able to detect the boats.

Figure 19: The harbour used in the simulation with and without boats (unknown obstacles).

5 Implementation of the path planning system onboard the USV

This chapter will explain the steps taken by the author to implement the optimal waypoint algorithm on the Cleaning Drone V1. The system is created in ROS (robotic operating system), using mainly python. Therefore, all the existing code was moved into ROS and divided into different packages and nodes before it was tested on the CDV1. The chapter is structured as follows:

- **Section 5.1:** The USV interface.
- **Section 5.2:** The main state machine.
- **Section 5.3:** Execution of the main cleaning pattern.
- **Section 5.4:** Automatic data logging.

5.1 The USV interface

To ensure stability and robustness, Maritime Robotics has shielded the control system of the USV from the rest of the code. The CDV1s control system is thus only accessible through a backseat driver. A backseat driver refers to a system that sends control inputs to the OBC (onboard computer). These control inputs can be one of the following:

- Manual control.
- Course following.
- Stationkeeping.
- leg-mode.

Manual control refers to directly inputting the torque and linear forces. Course-following mode makes the USV follow a specific course. Stationkeeping makes the USV take the shortest route to the given coordinate and keeps the USV at that position. Leg-mode has the USV follow a leg between two points. Examples of the control modes visualized can be seen in Figure 20. Since the goal of the optimal waypoint generation algorithm is to make the USV move in multiple straight lines, leg mode was the most logical control mode to use when commanding the USV back and forth.

It should also be mentioned that the only way for the backseat driver to communicate with the OBC is through the NMEA format explained in Section 2.6. Therefore, a package for handling all the communication was created to parse and translate NMEA messages before sending them to the OBC. The OBC will enter drift mode (no actuator movement) if no message has been received for 5 seconds as a safety measure. As a result, the guidance system must continuously repeat the current commando to the OBC to avoid this.

Connection to the OBC is set up through a network socket. The author decided to go with the user datagram protocol (UDP) [57] in this case. This protocol was selected to avoid the congestion that would have ensued if the transmission control protocol (TCP) [58] had been used. The TCP

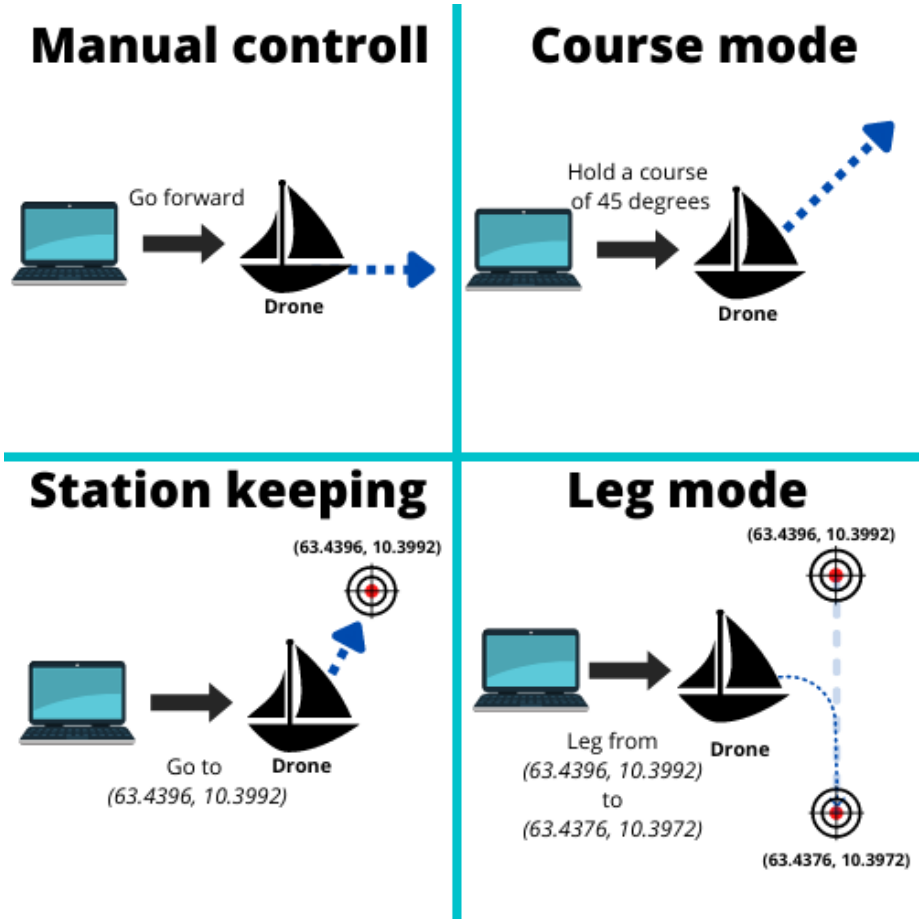


Figure 20: Examples of the different control modes.

protocol requires all sent messages to be acknowledged, which requires the guidance system code to run at least as fast as the OBC. The computer running the guidance system would then waste a lot of computing power acknowledging messages from the OBC, which can reduce its real-time computing capabilities. Using UDP, no messages need to be acknowledged, and missed messages are lost. Loss of messages is of no concern as the command messages are repeated and since the most crucial GNSS messages are the current ones.

Ideally, the guidance system would run on a separate computer on the USV, but at the current development stage, it runs on a computer on land and broadcast to the USV using radio. The messages received from the onboard computer will also be transferred over this same radio link.

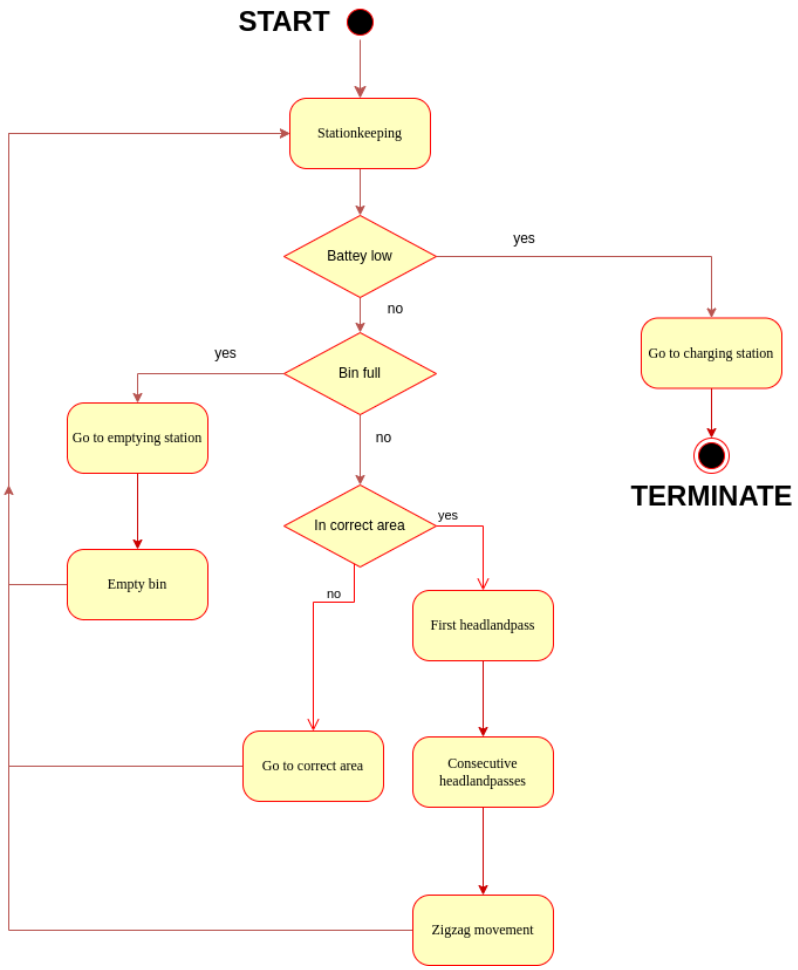


Figure 21: The state diagram for USV path-planning algorithm.

5.2 The state machine

A safe and straightforward way of implementing the USV behaviour was through a finite state machine. The behaviour of the USV could then be divided into different states that could be executed independently. This modularity in the form of states also made the code easier to debug, as errors could be identified within a given state. The structure of the central state machine for the created guidance system can be seen in Figure 21.

The initial and idle mode of the system will be stationkeeping. This mode ensures that the USV does not drift off when no new commands are given. The process is only terminated when the battery is low, and the USV has returned to its charging station. The behaviour of the USV can, as seen in the state diagram, be described by four if-statements. namely:

- If the battery is low, go to the charging station.
- If the bin is full, go empty it.
- If the USV is in the correct area, clean it.
- If the USV is not in the correct area, go to it.

All the different states will have to be again divided into their own state machines, but Figure 21 should show the overall behaviour of the desired system in the end. Using a state machine to control the overall system was picked over, e.g. threading. This choice was due to the simplicity of implementing a state machine and the relatively low speed of the system at this level. There is room for some delay when switching between the displayed states.

5.3 Execution of the cleaning pattern

The cleaning pattern starts execution by requesting the set of coordinates for a given area from the pathplanner. The pathplanner looks up the areas dimensions using the stored coordinates and creates a set of coordinates that need to be visited for an area to be considered clean.

Since the advanced in-between area movement pathplanning only is implemented in the simulator, the guidance system initially tells the USV to take a direct path to the first coordinate returned by the pathplanner. When this first coordinate is reached, the code considers the USV to be in the correct area. The guidance system then uses repeating leg-mode commands to tell the USV to move in a straigth line between the current goal coordinate and the previos one. This way the USV should keep a straigth line between its current and the next coordinate. This leg-mode pattern will continue untill all of the required coordinates has been visited.

When the USV is finished with cleaning an area, the state machine sends a new request to the pathplanner to get the coordinates for the next area which needs to be cleaned. This pattern continues. The algorithm only terminates when all of the planned areas have been cleaned, or the USVs battery status has dipped below a threshold value. A visualized example of this can be seen in Figure 22.

5.4 Automatic data logging

In order to create a data-driven distance estimation model, data collection for the USV has to be carried out. For this, a data logger module was created in ROS. The logger tracks the following metrics with an interval of 1 Hz:

- date.
- time.
- battery level.
- rpm port motor.
- rpm starboard motor.
- temp port motor.
- temp starboard motor.
- power consumption port motor.

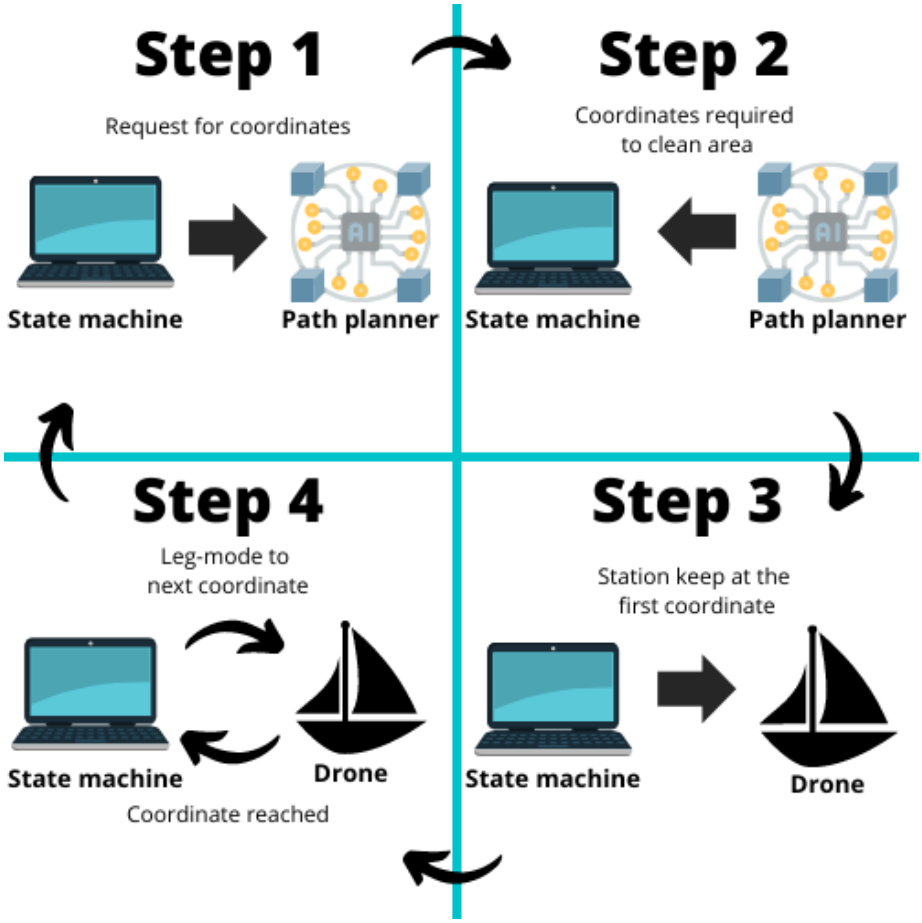


Figure 22: The main cleaning pattern visualized.

- power consumption starboard motor.
- power consumption in total.
- batteries count.
- distance since the previous step.

Most of these metrics are extracted directly from the OBC. However, distance since the previous step is calculated by measuring the Euclidean distance between the last GPS coordinate and the current one. Date and time are also measured using standard python packages.

6 Results

The results from the three different parts of this thesis are highlighted and discussed in sections 6.1-6.5. The results will reflect how well the implementation and design of new and old systems have worked both in the simulator and in practice. Both the implemented guidance system and the simulations were run on a computer with specs given in Appendix F. The structure of the chapter is as follows:

- **Section 6.1:** Results from implementing a custom guidance system on a USV using ROS.
- **Section 6.2:** Guidance system performance.
- **Section 6.3:** Remaining range estimation.
- **Section 6.4:** Path planning between areas using D* Lite.
- **Section 6.5:** Trash collection using the CDV1.

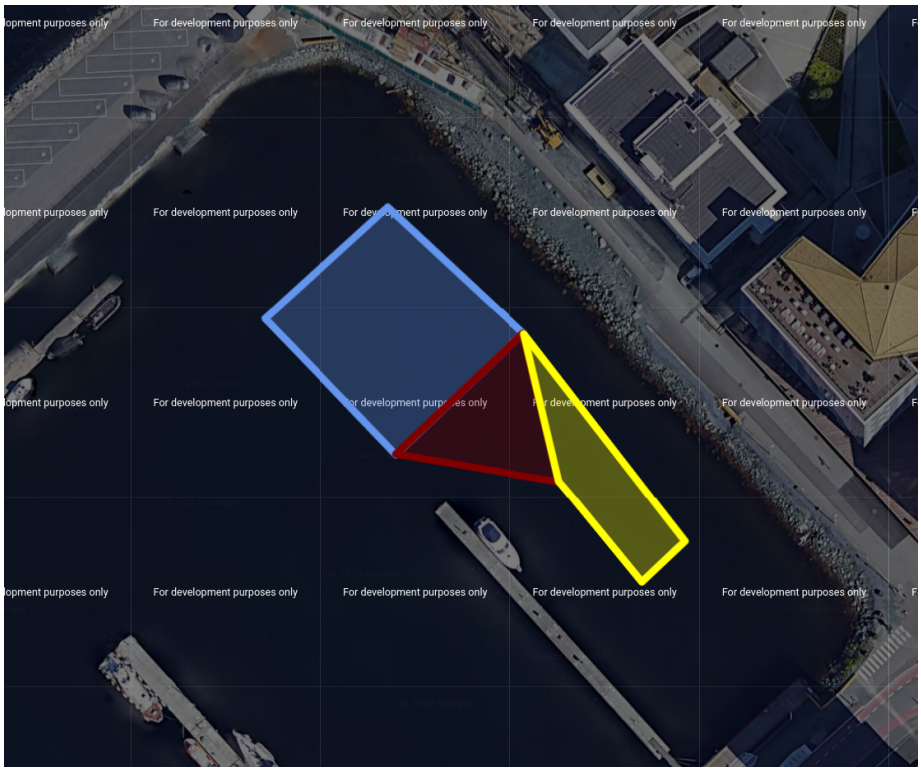


Figure 23: Area spilt for testing without lidar.

6.1 Results from implementing a custom guidance system onboard a USV using ROS

The system created in ROS worked to a satisfying degree when implemented onto the drone. The system's modularity made it easy to debug and add features. From a system safety point of view, the modularity prevented a crash of one packet from bringing down the entire system. This failsafe was proven a couple of times as the logger would crash while the rest of the system would remain operational.

There was a slight delay between the guidance system running on the local computer and the actions performed by the onboard computer on the USV. This delay caused no critical safety issues but made the USV overshoot its goal a couple of times as it was not told fast enough to slow down when needed. A random error causing the GPS signal not to come through correctly also occurred a couple of times. Otherwise, the system worked as expected and guided the USV to the correct position within a satisfying timeframe.

Another realized advantage of using ROS is that it simplifies creating parallel running code, as each ROS node runs independently of the other nodes. This parallelization allowed for a much smoother execution of the guidance system as planning, communication, and logging does not have to run serially. By also using the ROS-topics to communicate between different nodes and packages, the same message could be sent to multiple sources. The final system created can be seen in Appendix B and is created to be scalable in the future.

6.2 Guidance system performance

The waypoint generating algorithm created in [1] was tested in the area shown in Figure 23. The goal of the testing was to see if CDV1 could follow the pattern mapped out by the waypoint generating algorithm.

The USV was able to get to all of the desired waypoints within an expected timeframe, meaning that the ROS implementation of the waypoint generating guidance algorithm onto CDV1 worked. A waypoint was considered reached when the CDV1 was within two meters. This two-meter limit was deemed necessary due to inaccuracies in the GPS and delays in the system. It was picket through trial and error.

In the initial testing, the speed of the CDV1 was set to 1 m/s. However, this speed did lead to the USV overshooting its goals. The vehicle was also perceived as faster than expected and made the USV seem less friendly. This overshooting of its goals also resulted in the USV deviating from its planned path. The speed was then reduced to 0.5 m/s. At these speeds, wind and small currents had a more significant impact on the CDV1, and the USV started to wobble around its desired trajectory instead of keeping a straight line. This speed reduction decreased the overshoot but did not remove it.

To counteract the overshooting of the targets, the guidance system was adjusted so that the reference speed of the USV became linearly dependent on the distance to the goal. This speed was capped on top at 0.6 m/s as described by equation 6.1. This adjustment removed the overshoot and

kept the USV at a relatively low speed.

$$r_{speed} = \min(e_{goal}/20, 0.6) \tag{6.1}$$

Since the USV was not moving in the precise lines mapped out for it by the waypoint generating algorithm, not all of the planned area was cleaned. In addition, because a point is accepted with a margin of two meters, all of the corners in the area were cut. The cutting of corners is also a result of the algorithm's flawed assumption that the USV could stop precisely at the desired point and then turn on the spot.

These results show that it is challenging to achieve precise movement when working with small vessels on water. Even with an algorithm created for covering a whole area, many patches were still left out. There was much overlap during the headland passes as small offsets had the CDV1 deviate a little from its path. The accuracy of the USV's GPS might be in the cm span, but the deviation might be considered a bit more in these conditions for the system as a whole. The algorithm created in the authors' specialization project can still be used to cover most of the desired area. It would also be wrong to assume that all trash gets collected by the described coverage. This assumption is flawed because the trash moves about in a random pattern, and the USV does not move as precisely as we would like.

During testing, it was also noticed that most trash in this test area tends to pile up bounded in organic matter in the different corners of the harbor. This discovery coincides with the theory previously discussed in Section 2.1 about trash hotspots. However, the accumulation of trash in such hotspots was more than expected. Therefore, two to three headland passes might be sufficient for collecting trash. For these headland passes, a lidar will be crucial to avoid collisions and use the distance to land to keep a stable course. Following tracks using only the GPS will be used only when cleaning areas not close to land. These areas will probably have trash sparsely spread, so a centimeter accuracy when following the tracks might not be necessary.

6.3 Remaining range estimation

In order to estimate the remaining travel range, the data logged by the USV throughout the testing was examined. However, this data did bear the mark of a new system continuously being implemented and tested. The data was fractioned, and there was little of it. There was not much information that could be extracted from it as the data consisted of multiple smaller trips.

The lack of good quality data rendered the author unable to create a representative regression. However, it could be extracted that the CDV1 could travel approximately 250 meters per percentage of charge with its current setup from the existing data. This range is compared against the standard otter platform with both two and four batteries in Figure 24. The discovery means that with a 10% battery retention when cleaning, the CDV1 should be able to get home with a reasonable margin of error if it is within approximately two kilometers of its charging station.

These values are a huge deviation from what can be seen in Appendix A. This deviation can be explained by the fact that the values in Appendix A are for an *otter* with four batteries attached.

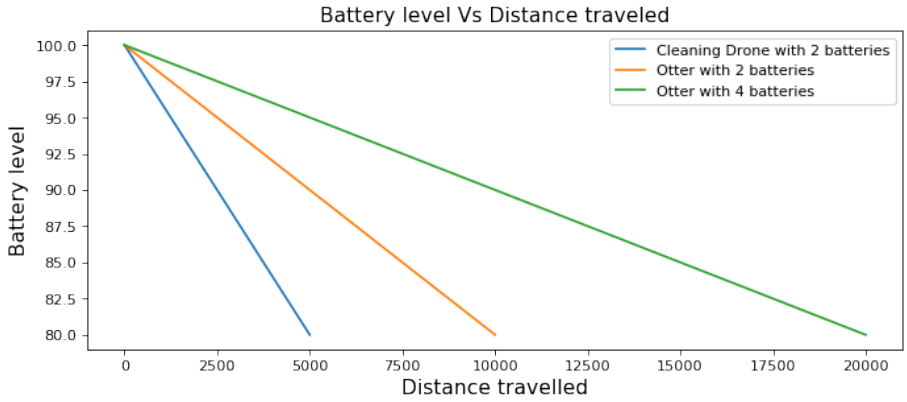


Figure 24: The predicted travel range for the CDV1 and an *otter* with two and four batteries.

For simplicity of the testing, only two batteries were used on the CDV1. This halving of batteries should, in theory, half both the operating time and operating range of the CDV1, resulting in the drone being able to travel five kilometers on 10% of the battery.

The drop from five kilometers to 2.5 kilometers can presumably be linked to the increase in drag due to the trashbin and the fact that the CDV1 movement is more variable compared to the usual missions of the *otter* platform. The *otter* is mainly used for surveying missions. These missions mainly consist of moving in straight lines while having few and large turns. The Cleaning Drones pattern consists of more and sharper turns, thus decreasing the expected maximum distance.

Data collection should have a high priority in the future, as the travel range estimate made might vary based on multiple factors. The weather, amount of trash collected, behavior, and distance traveled might all play a role in determining the remaining range of the CDV1. Therefore, data collection in all of these scenarios should be conducted in the future.

The data these assumptions were made on were bad, and further testing will have to be conducted before it can be determined that the current max distance of the CDV1 with two batteries is 25 kilometers. However, the current estimate is a rough estimate that might not be too far from the truth.

6.4 Path planning between areas using D* Lite

Before implementing a pathplanning algorithm to plan between area movement, the code always took the fastest path from one area to the next. This basic pathplanning often resulted in many illegal movements as obstacles such as floating jetties were crossed. An example of this can be seen in Figure 25. In addition to the guidance system not taking into account known obstacles, unknown obstacles would not be avoided either.

6.4.1 Results from implementing D* Lite into the pathplanner

With the implementation of D* Lite as a pathplanning algorithm, the USV avoided known obstacles while still finding a good path to the goal position. This new movement can be seen in Figure 26.

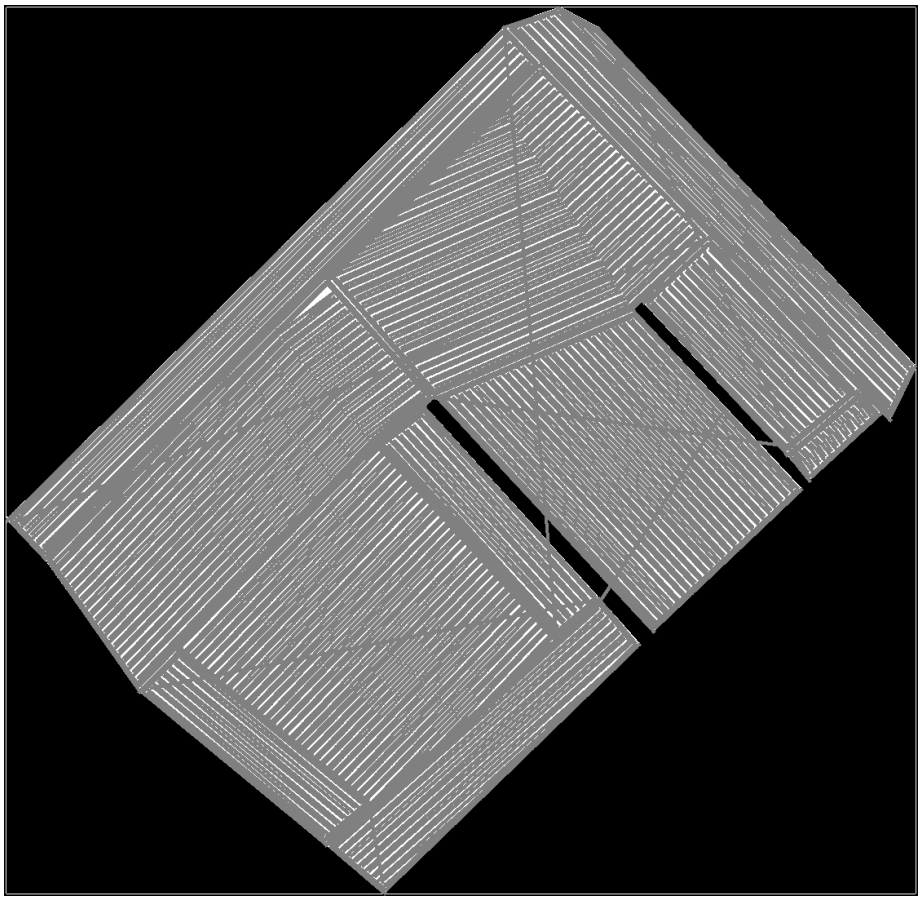


Figure 25: Covering of all areas prior to D* Lite being implemented

Initially, it was thought that since the CDV1 can go in a straight line to a given waypoint, additional code could be added to the D* Lite algorithm so that the path the code returns only contains the coordinates of the path where the heading changes. An example of this can be seen in Figure 27.

The guidance system can get the USV to move forward continuously and then hold if any time-consuming recalculation has to be done. However, only following the corners of the generated path was not sufficient. The USV needed to continuously scan for possible obstacles and be able to update its path, given that a blocking object was discovered. It made more sense to take many smaller steps as the code responsible for the movement needs to check if any changes to the environment have been detected at a fixed interval. The USV could then be moved from one point to another by

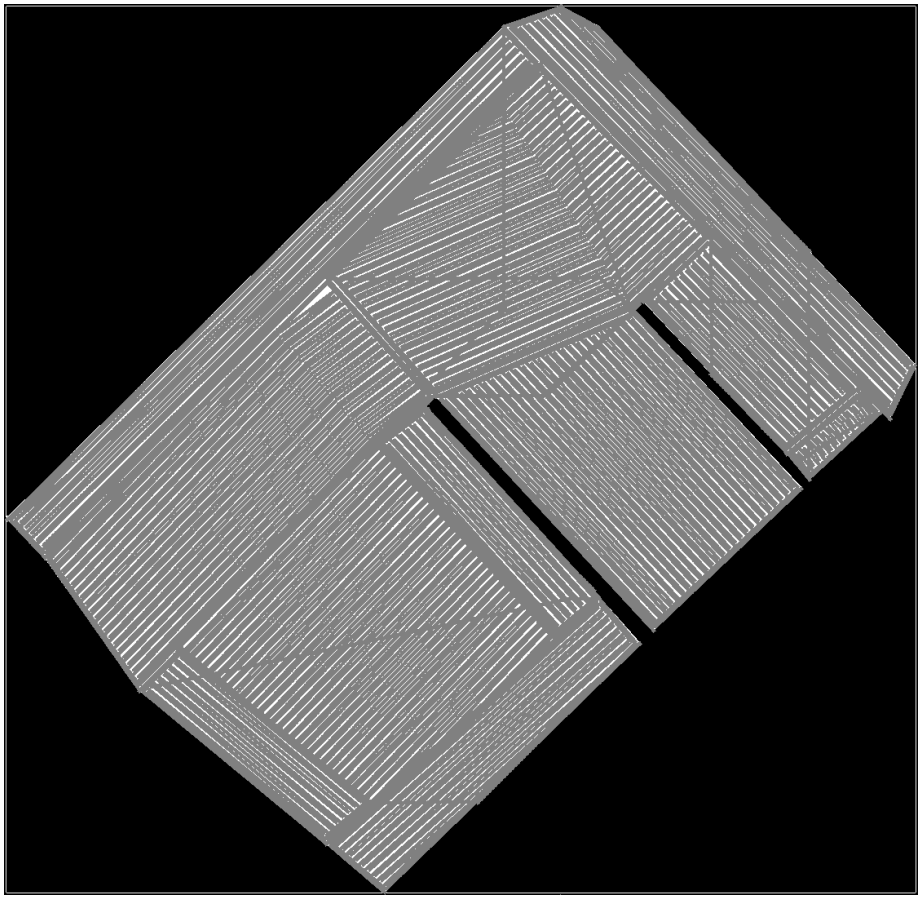


Figure 26: Covering of all areas after D* Lite was implemented.

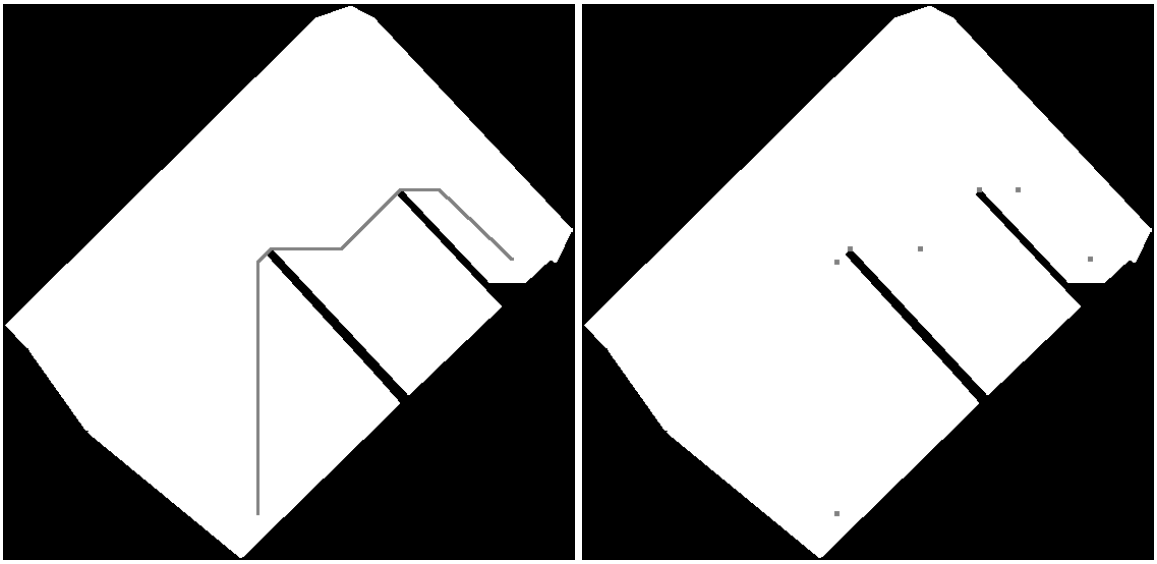
continuously moving a station-keeping coordinate.

In this simulation, the resolution of the pixels used in the simulator was approximately 40 cm in the real world. However, this can be both increased or decreased in the future to find a good computation-time / real-world accuracy trade-off.

6.4.2 Results regarding obstacle avoidance

The goal of using the D* Lite algorithm was not only to avoid land but also to avoid unknown obstacles. As described in Section 4.4, the system's capabilities were using a simulated lidar and a lot of "hidden" obstacles (boats). The algorithm would initially plan a route, follow it, and then re-plan as the obstacles were discovered.

The algorithm was, in all cases, able to re-plan a route with little effort and only used a fraction of the time required to calculate a new route. The results from the simulator can be seen in Figure 28 and Figure 29. The results show the parts of the boats detected and the system's path to avoid them.



(a) The path created by D* Lite from one end of the map to the other

(b) The coordinates where the heading of the path changes, as well as start and goal position

Figure 27: The path created by D* Lite, and the resulting coordinates given to the drone.

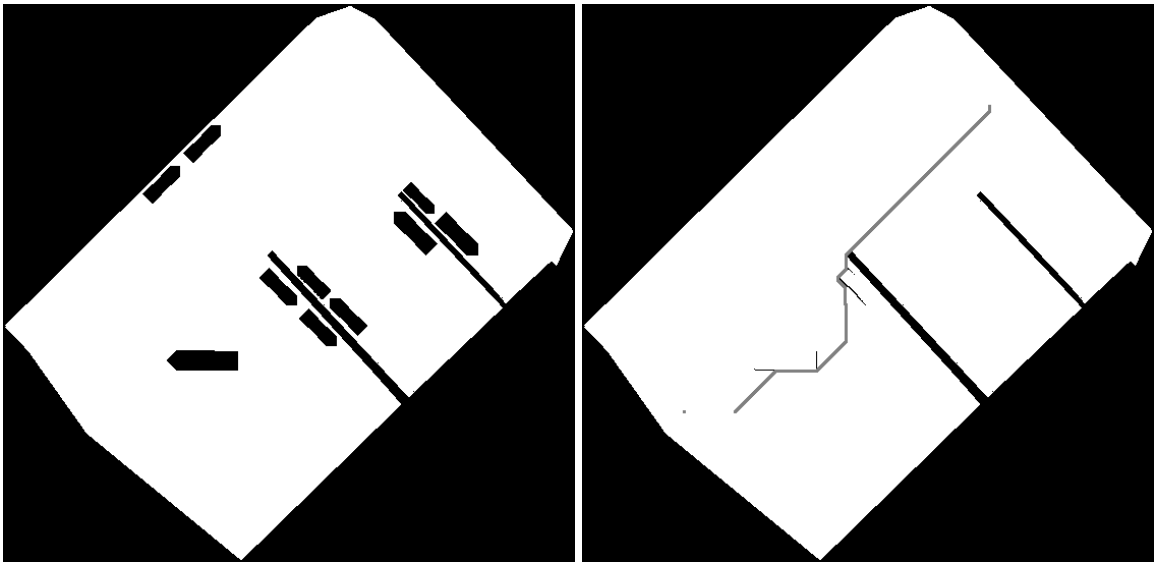
This new pathplanning algorithm should make the guidance system more resilient to handling obstacles in its path. With sufficient integration onto the CDV1, the guidance system should be ready to navigate around obstacles detected by the lidar when it is mounted onto the drone.

Figure 30 Displays one of the early attempts at adding obstacles to the path of the USV. In that scenario, the obstacle spawned with a 10% probability for each step the drone took. The USV noticed the changed tiles when the obstacle spawned and recalculated the route. This method of detecting obstacles was later dropped for the current one. The old method led to the system expanding more vertexes than necessary, and it was not close to how the system would detect obstacles in the real world. Even though this method of adding obstacles was dropped, it proved that the system could handle the detection of obstacles in multiple different ways.

6.4.3 Results following the addition of padding

In order to avoid sailing too close to obstacles when traveling between areas, padding was added to all obstacles. This padding or virtual obstacle made the USV sail with a minimum distance to excising and discovered obstacles. Some simulated examples of the USV traversing a path with and without padding can be seen in Figure 31.

Adding padding increased the initial runtime of the simulation depending on the amount of discovered objects. All known obstacles are padded during the calculation of the initial path and only contributed to a little increase in runtime prior to traversing the path. The padding to the unknown objects was added as the objects were discovered.



(a) The harbour with dynamic features (boats). (b) A path taken by the USV in the simulator.

Figure 28: The simulated harbour with boats, and the path taken by the USV.

It is worth noting that the current implementation assumes that all discovered obstacles are static. The padding is thus just blocking the USV from entering a specific cell on the map. If the blocking object is moving, it might collide with the USV or render many cells blocked since padding is currently added and not removed. The padding is, either way, a step toward safer movement between areas.

6.4.4 Increasing the number of search directions

As seen in Figure 32 increasing the number of search directions reduces the number of sharp corners taken by the USV and allows for a smoother path. This smoothening of the path also happened without a noticeable increase in runtime.

Initially, the increase in search directions was added to the expanding of map vertexes and when traversing the map. However, that lead to a significant increase in runtime. The author then decided not to include the additional search directions when expanding vertexes as this led to much unnecessary overlap in the vertexes expanded without increasing the system's performance. By only using the additional search directions when selecting which node to move into next, the path still got smoothed, but the simulation runtime was not significantly impacted.

The increase in evaluated vertexes when moving did, however, lead to the USV taking longer steps (up to three tiles) before finding a new cell to move into. This increase in step length could increase the time it takes for the system to respond to obstacles and have the USV move closer than desired to discovered obstacles. Therefore, this improvement should be implemented with the padding to maintain a reasonable distance to discovered obstacles.

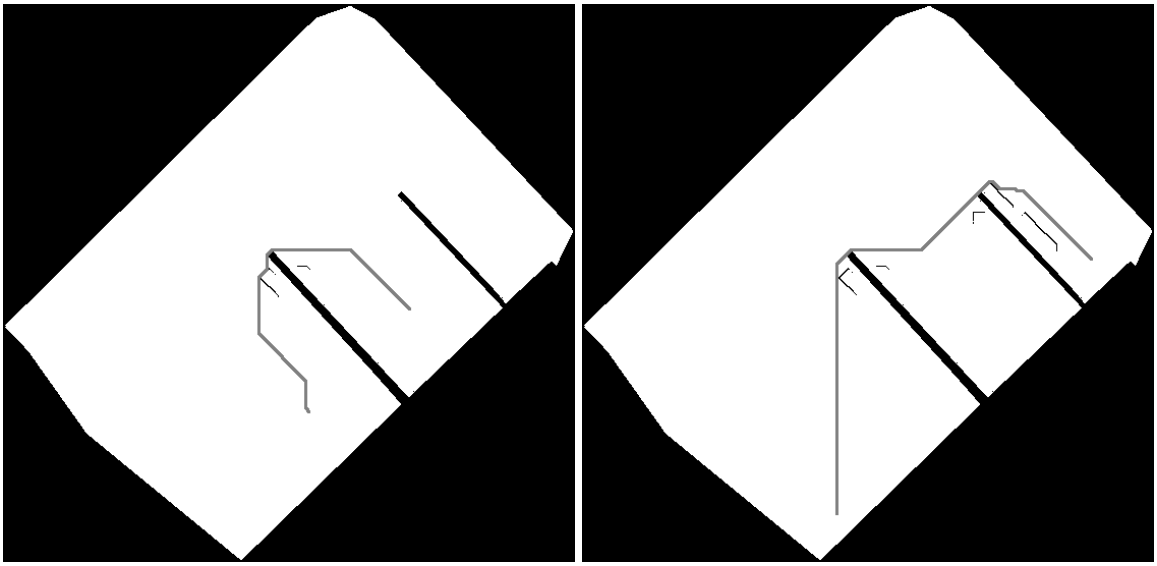


Figure 29: More paths taken by the USV.

6.4.5 Using a min binary heap compared to a conventional priority queue

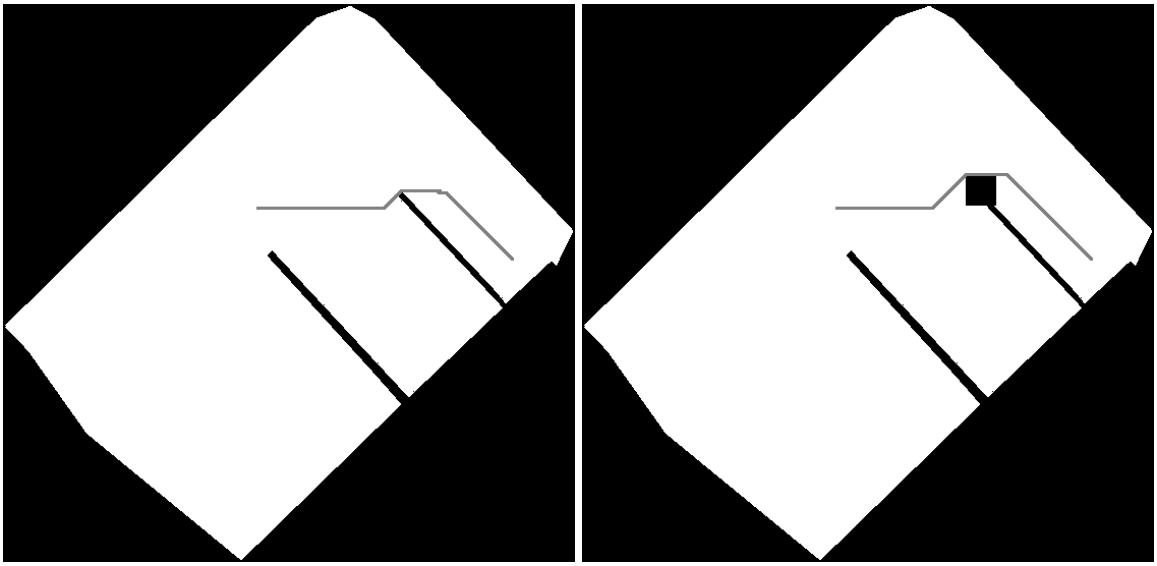
As suggested by [4] the conventional priority queue initially used was changed to a minimum binary heap. This change resulted in a somewhat slower runtime as seen in Table 6. The results presented in Table 6 is a rounded average from multiple runs. There were some variations, but the runs using an array-based priority queue took consistently lower time compared to the minimum binary heap.

Path	Simulation time using priority queue (s)	Simulation time using binary heap (s)
1	20	23
2	27	31
3	24	28

Table 6: Simulation time using priority queue vs using binary heap.

The poor performance of the minimum binary heap is probably because the implementation of D* Lite sometimes has to delete or update nodes that are not at the top of the heap. These functions require the algorithm to search through the heap, find the right element, change it, and recreate the heap again. For a conventional (array-based priority queue), the element still has to be found and changed, but there is no need to recreate the heap, thus saving time.

A time complexity of O(log_n) for the search operation for a minimum binary heap as presented by [4] might not be entirely correct either. In the authors' implementation, all elements must be visited during a search, resulting in time complexity of O(n). Wikipedia also states that the time complexity of the search is O(n). With this in mind, the minimum binary heap is only faster than an array-based priority queue during the pop operation. A time complexity of O(n) for the binary



(a) The path created by D* Lite prior to obstacle being added (the ideal path).

(b) Another example of where the algorithm avoids a larger obstacle.

Figure 30: The desired path, and the new path following the addition of an obstacle.

heaps search operation makes the array-based priority queue faster during the push, delete, and update operation. It might be why the array-based implementation of the priority queue is a little bit faster.

6.5 Trash collection using the CDV1

Another significant part of the cleaning drones performance is its ability to collect trash using its current design. As mentioned previously, most of the trash encountered is in for of "hotspots" Section 2.1. To test the design of the CDV1, the USV was manually operated into one of these hotspots found at "Brattørakaia" (test harbor located in Trondheim).

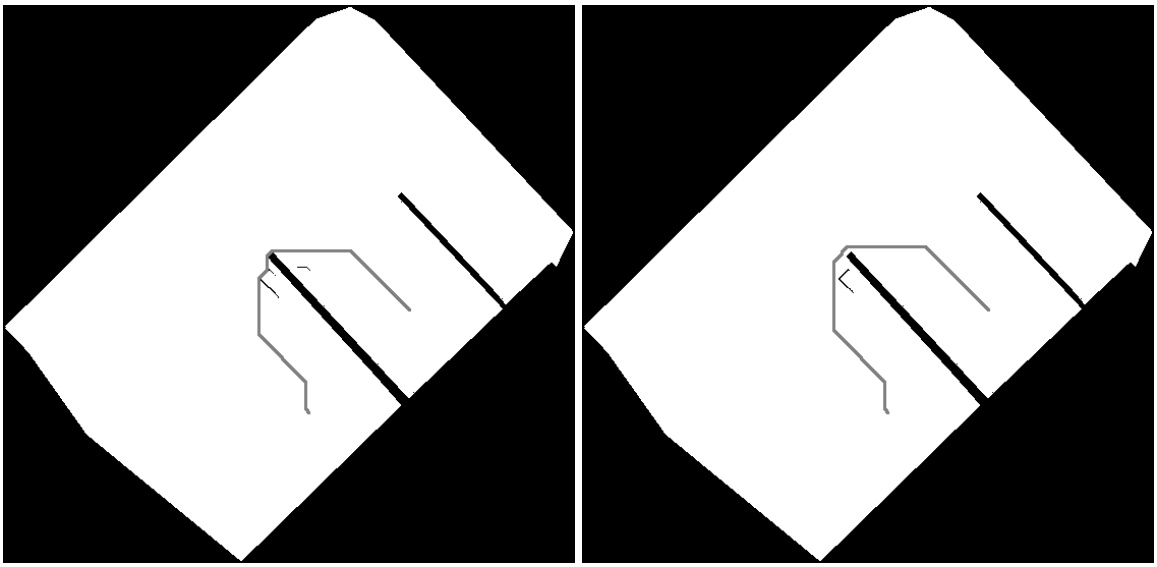
When the CDV1 was steered into the hotspot, the trashbin filled immediately, and the USV started to push trash around. An image of this can be seen in Figure 33. It was discovered that the flaw in the design is that since the trash floats, it can be considered to be in a 2D plane. The 2D plane or area of the current trash collecting system is tiny, only approximately 60 cm x 30 cm. This area fills up fast without the drone having collected a significant amount of trash.

The CDV1 did not excel at collecting trash close to the pier. As the drone got close and moved parallel to the pier, much trash was squeezed between the pontoons and land. This squeeze resulted in the trash not being collected. If the drone tried to ram directly into the pier to collect trash, the trash close to land could be retrieved by closing the trash bin door. The downside of this is that this technique is inefficient and requires a lot of ramming.

Another disadvantage of the current connection system is that the front of the CDV1 dips into the

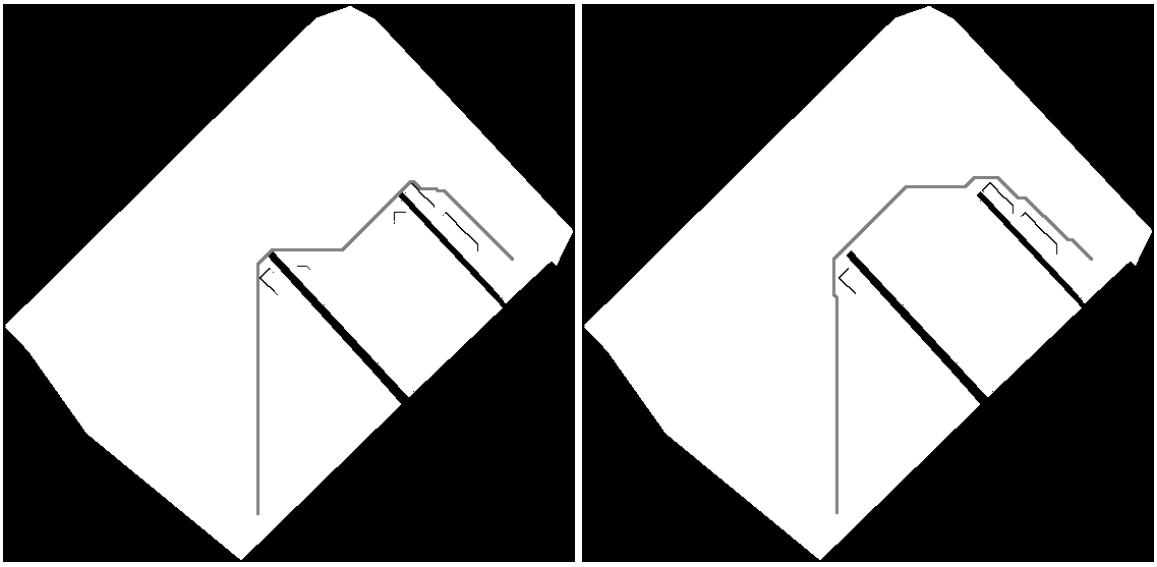
water at high speeds. This dip is due to the drag induced by the trashbin. In the long run, this dip can lead to issues if a component of the USV that generally should be overwater is put underwater more than necessary. It also has the disadvantage of making the CDV1s sailing look less smooth. A counterargument to this could be that the CDV1 is designed to only operate at low speeds, and thus its performance at high speeds is unnecessary to look at.

Most of the issues mentioned can probably be solved by changing out the box with a net that spans the entire length of the *otter*. The net could reduce drag and give a larger and more flexible collection area. The net could also be stretched to have a collection width of the entire drone, allowing for simpler collection close to land. This is, however, a path that will have to be pursued in the future.



(a) Path 1 without padding.

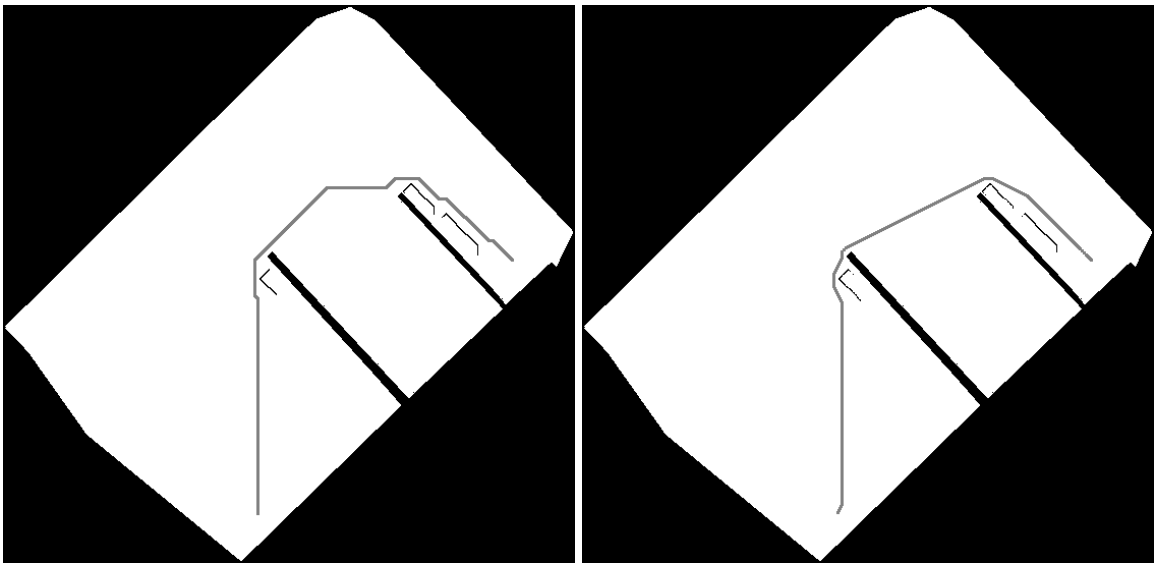
(b) Path 1 with padding.



(c) Path 2 without padding.

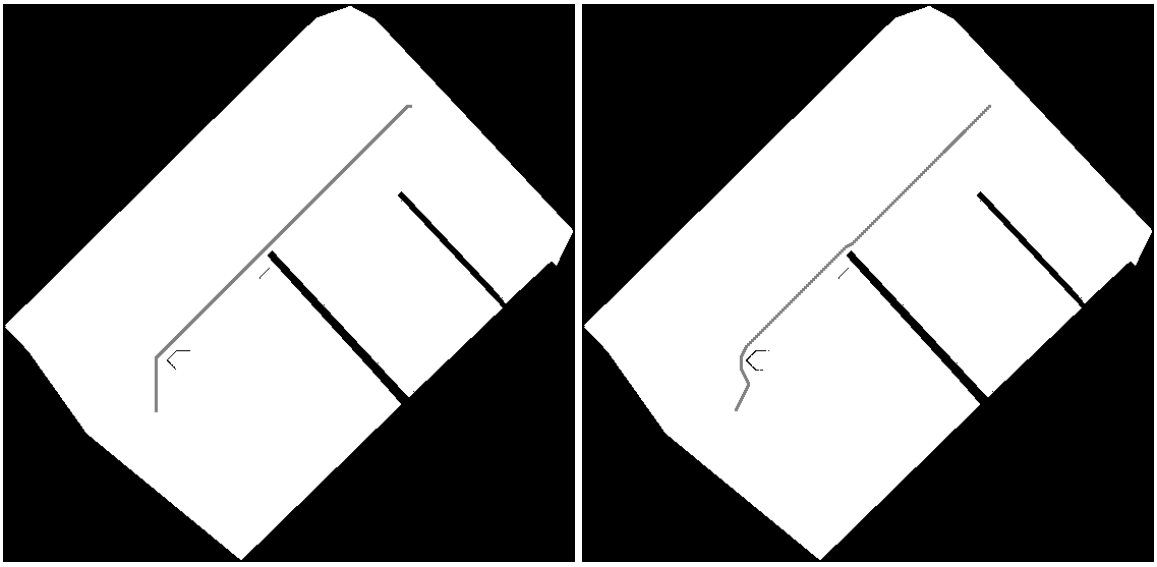
(d) Path 2 with padding.

Figure 31: Results following the addition of padding to obstacles.



(a) Path 1 with 8 search directions.

(b) Path 1 with 16 search directions.



(c) Path 2 with 8 search directions.

(d) Path 2 with 16 search directions.

Figure 32: Results following the addition of more search directions.

Figure 33: The CDV1 manually steered into a hotspot.

7 Conclusion and further work

In this final chapter, conclusions from the results will be presented. The research questions will be restated and answered. The author will also present possible further work and improvements which can be made to this system. The chapter is structured as follows:

- **Section 7.1:** Conclusion.
- **Section 7.2:** Further work.

Research questions:

1. How can the waypoint generation algorithm from [1] be implemented in the Cleaning Drone V1?
2. How can static object avoidance be incorporated into the waypoint generating algorithm?
3. How can the remaining range of the USV be predicted more accurately than relying solely on the battery level?
4. Is the drone's design capable of collecting trash from hotspots?

7.1 Conclusion

This report was aimed at answering multiple sub-problems and research questions in regards to cleaning a harbor basin using the *Cleaning Drone V1*. A guidance system was created and tested on the *Cleaning Drone V1*, and pathplanning using D* lite was tested in a simulator.

The guidance system created during the authors' specialization project was implemented in ROS and communicated from a local computer through NMEA messages over to the *Cleaning Drone V1*. The ROS implementation proved robust, easily scalable, and allowed code to run smoothly in parallel. These features of ROS made the system easy to implement and debug.

When running the created guidance system on the *Cleaning Drone V1* at Brattørakaia, the system worked as expected. However, it was quickly noticed that moving a small vessel in straight lines on water is difficult. Inaccuracies in the GNSS measurement and environmental disturbances caused many areas to be covered multiple times, while some parts of the area were left out. Therefore, creating an algorithm that makes a USV cover an area by moving in a fixed pattern seems not to work.

In order to evolve the guidance system, and prepare it for when a lidar is mounted onto the USV, the D* Lite algorithm was added to the guidance system to handle area to area movement. The algorithm was quickly able to direct the USV from one area to another without hitting any obstacles. A simulated lidar was also implemented to test the system's capabilities when discovering unknown obstacles. The system was, in all cases, able to detect and avoid the obstacles with reasonable clearance. The initial D* Lite algorithm was also improved by adding padding to obstacles and

increasing the number of search directions to decrease the minimum turning angle.

Data was tried collected from the USV during the testing to be used for a more accurate travel range estimation for the CDV1. Only a little useful data was collected, and this data consisted of multiple smaller trips. However, it could be extrapolated that the Cleaning Drone had a travel range of approximately 25 kilometers. This estimate is rough and more data has to be collected in the future in order to confirm this.

The *Cleaning Drone V1* current design has been evaluated to have an insufficient trash capacity compared to what was expected. As backed by the theory, most trash could be found in hotspots close to land. However, the lack of a lidar resulted in the CDV1 being unable to sail close to land and collect it autonomously. Without a lidar, inaccuracies in the system could lead to a collision, and this collection was hence done manually. By manually sailing the USV into the hotspot, the author evaluated the USVs trash-collecting capabilities. The results were quite bad, as the drone quickly filled up and could barely collect any thrash. The current design also made it difficult for the USV to collect trash close to land.

In conclusion, the author has synthesized research regarding extended object tracking, compared different path planning algorithms, and selected a suitable one for our use case. The selected path planning algorithm was integrated into the simulator and tested. The guidance system created in the authors' specialization project was implemented onto the CDV1. Its effectiveness was also tested in a local harbor. Lastly, the current design of the trash collecting system on the CDV1 was evaluated and was deemed insufficient regarding trash-collecting capabilities.

7.2 Further work

Many things need to be addressed to develop the drone and its guidance system further. The next step in this development will be to redesign the trash collection system of the *Cleaning Drone*. This step has to be done in combination with adding a lidar to the system, thus enabling object detection. The *Cleaning Drone* should then be able to collect trash closer to land. All functionality implemented and tested in the simulator must also be tested on the real USV after the lidar has been mounted.

An improved method of cleaning an area with the help of the lidar is also needed. Using the lidar, the USV should be able to sail close to land without colliding with boats and other objects. A way of handling the issue of a goal position being unreachable must also be created. This could be either because an obstacle blocks the goal, or because the goal is placed on land due to inaccuracies.

The guidance system also needs to be implemented on the CDV1 along with a dedicated computer to handle its processing. A dedicated computer will enable the guidance system to run on the drone instead of having to be transmitted over radio from a computer on land.

This development aims to lift the system in the autonomy ranking and limit the need for human interaction. For this to happen, wireless charging of the drone has to be added. The USV also needs a system for handling dynamic obstacles.

More data from the system also needs to be collected. Both visual data, i.e., the camera feed, and system data, i.e., power consumption, need to be collected. More visual data will allow the engineers to train a computer vision model for the CDV1. At the same time, more system data is

necessary in order to get a more accurate travel range estimation.

It is also worth looking at the possibility of extending this algorithm to handle multiple drones in cooperation over a larger area (swarm behavior). The otter platform also includes a front-facing camera which can be used to detect trash or obstacles to build even more on the efficiency of the drone. Reinforcement learning can also be implemented as a tool for the USV to optimize its cleaning further or possibly create better areas based on where it finds trash.

Bibliography

[1] Olsen, G. L. 2021. Optimal path planning for autonomous trash collection in harbor basins. *Report in TTK4550 Engineering Cybernetics, Specialization Project*, Norwegian University of Science and Technology.

[2] Solutions, C. S. 2022. Clean sea solutions cleaning drone. https://www.cleanseasolutions.no/product-aquadrone. Accessed: 2022-05-16.

[3] Granstrom, K., Baum, M., & Reuter, S. 2016. Extended object tracking: Introduction, overview and applications. *arXiv preprint arXiv:1604.00970*. doi:https://doi.org/10.48550/arXiv.1604.00970.

[4] Zhu, X., Yan, B., & Yue, Y. 2021. Path planning and collision avoidance in unknown environments for usvs based on an improved d* lite. *Applied Sciences*, 11(17). URL: https://www.mdpi.com/2076-3417/11/17/7863, doi:10.3390/app11177863.

[5] Koenig, S. & Likhachev, M. 2002. D* lite. *Aaai/iaai*, 15, 476–483. URL: https://dl.acm.org/doi/10.5555/777092.777167.

[6] ShipRight procedure – autonomous ships, . 2016. Lloyd's register (2016) cyber-enabled ships. https://mymaritimeblog.files.wordpress.com/2016/07/lr_cyber_enabled_ships_shipright_procedure_autonomous_ships_version_1-0_july_2016.pdf? Accessed: 2022-02-01.

[7] Gall, S. C. & Thompson, R. C. 2015. The impact of debris on marine life. *Marine pollution bulletin*, 92(1-2), 170–179. doi:https://doi.org/10.1016/j.marpolbul.2014.12.041.

[8] Derraik, J. G. 2002. The pollution of the marine environment by plastic debris: a review. *Marine pollution bulletin*, 44(9), 842–852. doi:https://doi.org/10.1016/S0025-326X(02)00220-5.

[9] Wabnitz, C. & Nichols, W. J. 2010. Plastic pollution: An ocean emergency. *Marine Turtle Newsletter*, 1–5.

[10] Van Cauwenberghe, L., Claessens, M., Vandegehuchte, M. B., & Janssen, C. R. 2015. Microplastics are taken up by mussels (mytilus edulis) and lugworms (arenicola marina) living in natural habitats. *Environmental pollution*, 199, 10–17. doi:https://doi.org/10.1016/j.envpol.2015.01.008.

[11] Jambeck, J. R., Geyer, R., Wilcox, C., Siegler, T. R., Perryman, M., Andrady, A., Narayan, R., & Law, K. L. 2015. Plastic waste inputs from land into the ocean. *Science*, 347(6223), 768–771. doi:10.1126/science.1260352.

[12] Vincent, A. E. & Hoellein, T. J. 2017. Anthropogenic litter abundance and accumulation rates point to seasonal litter sources on a great lakes beach. *Journal of Contemporary Water Research & Education*, 160(1), 72–84. doi:https://doi.org/10.1111/j.1936-704X.2017.03241.x.

[13] Kong, S., Tian, M., Qiu, C., Wu, Z., & Yu, J. 2021. Iwscr: An intelligent water surface cleaner robot for collecting floating garbage. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(10), 6358–6368. doi:10.1109/TSMC.2019.2961687.

[14] Gao, X. & Fu, X. 2020. Miniature water surface garbage cleaning robot. In *2020 International Conference on Computer Engineering and Application (ICCEA)*, 806–810. doi:10.1109/ICCEA50009.2020.00176.

[15] Chang, H.-C., Hsu, Y.-L., Hung, S.-S., Ou, G.-R., Wu, J.-R., & Hsu, C. 2021. Autonomous water quality monitoring and water surface cleaning for unmanned surface vehicle. *Sensors*, 21(4), 1–21. doi:https://doi.org/10.3390/s21041102.

[16] Wang, Y., Zhao, Y., Wu, Y., Zhang, S., & Wang, J. 2021. A multi-sensor intelligent surface garbage cleaning robot. In *2021 IEEE International Conference on Mechatronics and Automation (ICMA)*, 797–801. doi:10.1109/ICMA52036.2021.9512614.

[17] Wang, Z., Liu, Y., Yip, H. W., Peng, B., Qiao, S., & He, S. 2008. Design and hydrodynamic modeling of a lake surface cleaning robot. In *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 1343–1348. IEEE. doi:10.1109/AIM.2008.4601857.

[18] Hasan, K. M., Reza, K. J., et al. 2014. Path planning algorithm development for autonomous vacuum cleaner robots. In *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, 1–6. IEEE. doi:10.1109/ICIEV.2014.6850799.

[19] Liu, Y., Lin, X., & Zhu, S. 2008. Combined coverage path planning for autonomous cleaning robots in unstructured environments. In *2008 7th World Congress on Intelligent Control and Automation*, 8271–8276. IEEE. doi:10.1109/WCICA.2008.4594223.

[20] Lakshmanan, A. K., Mohan, R. E., Ramalingam, B., Le, A. V., Veerajagadeshwar, P., Tiwari, K., & Ilyas, M. 2020. Complete coverage path planning using reinforcement learning for tetromino based cleaning and maintenance robot. *Automation in Construction*, 112, 103078. doi:https://doi.org/10.1016/j.autcon.2020.103078.

[21] Solutions, C. S. 2021. Clean sea solutions website. https://www.cleanseasolutions.no/. Accessed: 2021-11-24.

[22] Robotics, M. 2021. Otter website. https://www.maritimerobotics.com/otter. Accessed: 2021-11-23.

[23] forautomotive engineers, S. 2021. Sae levels of driving automation™ refined for clarity and international audience. https://www.sae.org/blog/sae-j3016-update. Accessed: 2022-05-16.

[24] Vagale, A., Oucheikh, R., Bye, R. T., Osen, O. L., & Fossen, T. I. 2021. Path planning and collision avoidance for autonomous surface vehicles i: a review. *Journal of Marine Science and Technology*, 1–15. doi:https://doi.org/10.1007/s00773-020-00787-6.

[25] Wang, N., Gao, Y., Zheng, Z., Zhao, H., & Yin, J. 2018. A hybrid path-planning scheme for an unmanned surface vehicle. In *2018 Eighth International Conference on Information Science and Technology (ICIST)*, 231–236. IEEE. doi:10.1109/ICIST.2018.8426161.

[26] Souissi, O., Benatitallah, R., Duvivier, D., Artiba, A., Belanger, N., & Feyzeau, P. 2013. Path planning: A 2013 survey. In *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)*, 1–8. IEEE.

[27] Ferguson, D. & Stentz, A. 2007. Field d*: An interpolation-based path planner and replanner. In *Robotics research*, 239–253. Springer. doi:https://doi.org/10.1007/978-3-540-48113-3_22.

[28] Sun, X., Yeoh, W., & Koenig, S. 2010. Moving target d* lite. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 67–74.

[29] Yun, S. C., Ganapathy, V., & Chien, T. W. 2010. Enhanced d* lite algorithm for mobile robot navigation. In *2010 IEEE Symposium on Industrial Electronics and Applications (ISIEA)*, 545–550. doi:10.1109/ISIEA.2010.5679403.

[30] Shi, C., Zhang, M., & Peng, J. 2007. Harmonic potential field method for autonomous ship navigation. In *2007 7th International Conference on ITS Telecommunications*, 1–6. IEEE. doi:10.1109/ITST.2007.4295916.

[31] Kuwata, Y., Wolf, M. T., Zarzhitsky, D., & Huntsberger, T. L. 2013. Safe maritime autonomous navigation with colregs, using velocity obstacles. *IEEE Journal of Oceanic Engineering*, 39(1), 110–119. doi:10.1109/JOE.2013.2254214.

[32] Fox, D., Burgard, W., & Thrun, S. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1), 23–33. doi:10.1109/100.580977.

[33] Tam, C., Bucknall, R., & Greig, A. 2009. Review of collision avoidance and path planning methods for ships in close range encounters. *The Journal of Navigation*, 62(3), 455–476. doi:https://doi.org/10.1017/S0373463308005134.

[34] Chen, X., Liu, Y., Hong, X., Wei, X., & Huang, Y. 2018. Unmanned ship path planning based on rrt. In *International Conference on Intelligent Computing*, 102–110. Springer. doi:https://doi.org/10.1007/978-3-319-95930-6_11.

[35] Zhang, Y., Wang, W., Kobayashi, Y., & Shirai, K. 2012. Remaining driving range estimation of electric vehicle. In *2012 IEEE International Electric Vehicle Conference*, 1–7. IEEE. doi:10.1109/IEVC.2012.6183172.

[36] Neubauer, J. & Wood, E. 2014. The impact of range anxiety and home, workplace, and public charging infrastructure on simulated battery electric vehicle lifetime utility. *Journal of power sources*, 257, 12–20. doi:https://doi.org/10.1016/j.jpowsour.2014.01.075.

[37] Demir, E., Bektaş, T., & Laporte, G. 2014. A review of recent research on green road freight transportation. *European journal of operational research*, 237(3), 775–793. doi:https://doi.org/10.1016/j.ejor.2013.12.033.

[38] Basso, R., Kulcsár, B., Egardt, B., Lindroth, P., & Sanchez-Diaz, I. 2019. Energy consumption estimation integrated into the electric vehicle routing problem. *Transportation Research Part D: Transport and Environment*, 69, 141–167. doi:https://doi.org/10.1016/j.trd.2019.01.006.

[39] Xiong, R., Li, L., & Tian, J. 2018. Towards a smarter battery management system: A critical review on battery state of health monitoring methods. *Journal of Power Sources*, 405, 18–29. doi:https://doi.org/10.1016/j.jpowsour.2018.10.019.

[40] Qi, Z., Yang, J., Jia, R., & Wang, F. 2018. Investigating real-world energy consumption of electric vehicles: A case study of shanghai. *Procedia computer science*, 131, 367–376. doi:https://doi.org/10.1016/j.procs.2018.04.176.

[41] Van Sebille, E., Aliani, S., Law, K. L., Maximenko, N., Alsina, J. M., Bagaev, A., Bergmann, M., Chapron, B., Chubarenko, I., Cózar, A., et al. 2020. The physical oceanography of the transport of floating marine debris. *Environmental Research Letters*, 15(2), 1–33. doi:10.1088/1748-9326/ab6d7d.

[42] Lazcano, R. F., Vincent, A. E., & Hoellein, T. J. 2020. Trash dance: Anthropogenic litter and organic matter co-accumulate on urban beaches. *Geosciences*, 10(9), 1–15. doi:https://doi.org/10.3390/geosciences10090335.

[43] merriam webster. 2021. merriam-webster convex definition. https://www.merriam-webster.com/dictionary/convex%20polygon. Accessed: 2021-11-25.

[44] Velodyne. 2022. Velodyne puck-16 lidar website. https://velodynelidar.com/products/puck/. Accessed: 2022-03-07.

[45] Hameed, I., Bochtis, D., Sørensen, C., & Nøremark, M. 2010. Automated generation of guidance lines for operational field planning. *Biosystems engineering*, 107(4), 294–306. doi:https://doi.org/10.1016/j.biosystemseng.2010.09.001.

[46] Wikipedia. 2022. Nmea wikipedia page. https://en.wikipedia.org/wiki/NMEA_0183. Accessed: 2022-05-16.

[47] Shoab, M., Jain, K., Anulhaq, M., & Shashi, M. 2013. Development and implementation of nmea interpreter for real time gps data logging. In *2013 3rd IEEE International Advance Computing Conference (IACC)*, 143–146. IEEE. doi:10.1109/IAdCC.2013.6514210.

[48] Wikipedia. 2022. Ros wikipedia page. https://en.wikipedia.org/wiki/Robot_Operating_System. Accessed: 2022-05-16.

[49] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. 2009. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5. Kobe, Japan.

[50] Back-End, T. R. 2020. Ros1 vs ros2, practical overview for ros developers. https://roboticsbackend.com/ros1-vs-ros2-practical-overview/. Accessed: 2022-03-06.

[51] Bi, J., Wang, Y., Sai, Q., & Ding, C. 2019. Estimating remaining driving range of battery electric vehicles based on real-world data: A case study of beijing, china. *Energy*, 169, 833–843. doi:https://doi.org/10.1016/j.energy.2018.12.061.

[52] Lee, T., Chung, H., & Myung, H. 2011. Multi-resolution path planning for marine surface vehicle considering environmental effects. In *OCEANS 2011 IEEE-Spain*, 1–9. IEEE. doi:10.1109/Oceans-Spain.2011.6003669.

[53] Markets & Markets. 2021. Electric ship market by type. https://www.marketsandmarkets.com/Market-Reports/electric-ships-market-167955093.html. Accessed: 2022-03-15.

[54] Anwar, S., Zia, M. Y. I., Rashid, M., Rubens, G. Z. d., & Enevoldsen, P. 2020. Towards ferry electrification in the maritime sector. *Energies*, 13(24), 6506. doi:https://doi.org/10.3390/en13246506.

[55] Hikvision. 2022. Hikvision 2 mp outdoor ultra-low light bullet camera website. https://us.hikvision.com/en/products/cameras/turbohd-analog-camera/2mp/outdoor-bullet/2-mp-outdoor-ultra-low-light-bullet-camera. Accessed: 2022-05-09.

[56] Ferguson, D. & Stentz, A. 2005. The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments. *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-19*.

[57] Wikipedia. 2022. Udp wikipedia page. https://nn.wikipedia.org/wiki/User_Datagram_Protocol. Accessed: 2022-05-16.

[58] Wikipedia. 2022. Tcp wikipedia page. https://nn.wikipedia.org/wiki/Transmission_Control_Protocol. Accessed: 2022-05-16.

[59] Robotics, O. 2022. Ros homepage. https://www.ros.org/. Accessed: 2022-05-19.

A Cleaning Drone V1 spesifications

The specifications for Clean Sea Solutions Cleaning Drone V1. Note that some of these numbers are taken from the general *otter* platform and is might not be true for the CDV1. These values include:

- Top Speed.
- Operating time per charge.
- Max distance per charge.
- Recharge time.

The specs listed above might not be correct, but they do contain a rough estimate of the what we can expect until the real values are found.

Spec	Value
Height	81.5 cm
Length	200 cm
Width	108 cm
Weight	55 kg
Top Speed	1.5 m/s
Operating time per charge	18 hours
Max distance per charge	100 km
Recharge time	10 hours
Trash can width	60 cm
Trash can heigth	40 cm
Trash can volume	65L
Positioning system	Dual antenna GNSS
Positioning system accuracy	+-2cm

Table 7: The Cleaning drone V1 specifications.

Figure 34: The Cleaning Drone V1.

B Cleaning drone guidance system diagram

An overview of the system implemented as the guidance system in the Cleaning Drone V1. Each separate box is an independent package with a given purpose. In the current system there are four packages, namely; The main state machine, the logger, the network module, and the pathplanner. The network module is tasked with connecting to the USVs onboard computer, and the pathplanner calculates where the USV should move next.

Inter-package communication happens trough topics and services. Most of the topics are to and from the network module, as the module facilitates the communication with the OBC. These topics are namely, GPS signals, otter status, battery status and OBC commands, i.e commands that are being sent to the OBC.

There is also one service present. This service is offered by the pathplanner and allows other nodes to pass in an area number. The pathplanner will then return the coordinates that the USV will have to visit in order to clean the given area.

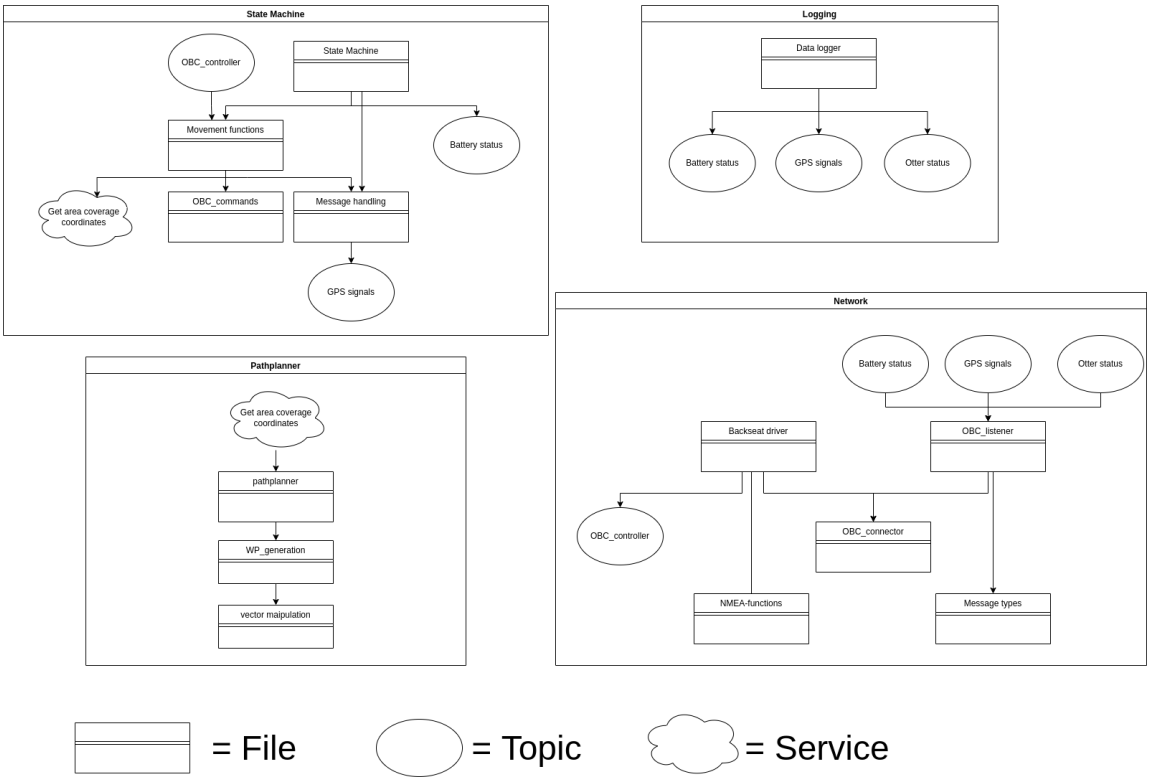


Figure 35: Simplified class diagram of the implemented guidance system.

C D* Lite optimized

The pseudocode used when creating the optimized D* Lite pathplanning algorithm used in this project. U is a priority queue with the open vertexes sorted by key, the vertex with the smallest key will be on top, and will be returned by $U.Top$. h is the heuristic function.

The rhs value is a one step lookahead estimate based on the g value, and replaces the f value of the traditional D* algorithm, its definition can be seen in the equation below.

$$\text{rhs}(s) = \begin{cases} 0 & \text{if } s = s_{start} \\ \min_{s' \in Pred(s)}(g(s') + c(s', s)) & \text{otherwise} \end{cases} \tag{C.1}$$

```
procedure CalculateKey(s)
{01"} return [min(g(s), rhs(s)) + h(s_start, s) + k_m; min(g(s), rhs(s))];

procedure Initialize()
{02"} U = ∅;
{03"} k_m = 0;
{04"} for all s ∈ S rhs(s) = g(s) = ∞;
{05"} rhs(s_goal) = 0;
{06"} U.Insert(s_goal, [h(s_start, s_goal); 0]);

procedure UpdateVertex(u)
{07"} if (g(u) ≠ rhs(u) AND u ∈ U) U.Update(u, CalculateKey(u));
{08"} else if (g(u) ≠ rhs(u) AND u ∉ U) U.Insert(u, CalculateKey(u));
{09"} else if (g(u) = rhs(u) AND u ∈ U) U.Remove(u);

procedure ComputeShortestPath()
{10"} while (U.TopKey() < CalculateKey(s_start) OR rhs(s_start) > g(s_start))
{11"}   u = U.Top();
{12"}   k_old = U.TopKey();
{13"}   k_new = CalculateKey(u));
{14"}   if(k_old < k_new)
{15"}     U.Update(u, k_new);
{16"}   else if (g(u) > rhs(u))
{17"}     g(u) = rhs(u);
{18"}     U.Remove(u);
{19"}     for all s ∈ Pred(u)
{20"}       if (s ≠ s_goal) rhs(s) = min(rhs(s), c(s, u) + g(u));
{21"}       UpdateVertex(s);
{22"}   else
{23"}     g_old = g(u);
{24"}     g(u) = ∞;
{25"}     for all s ∈ Pred(u) ∪ {u}
{26"}       if (rhs(s) = c(s, u) + g_old)
{27"}         if (s ≠ s_goal) rhs(s) = min_{s'∈Succ(s)} (c(s, s') + g(s'));
{28"}       UpdateVertex(s);

procedure Main()
{29"} s_last = s_start;
{30"} Initialize();
{31"} ComputeShortestPath();
{32"} while (s_start ≠ s_goal)
{33"}   /* if (rhs(s_start) = ∞) then there is no known path */
{34"}   s_start = arg min_{s'∈Succ(s_start)} (c(s_start, s') + g(s'));
{35"}   Move to s_start;
{36"}   Scan graph for changed edge costs;
{37"}   if any edge costs changed
{38"}     k_m = k_m + h(s_last, s_start);
{39"}     s_last = s_start;
{40"}     for all directed edges (u, v) with changed edge costs
{41"}       c_old = c(u, v);
{42"}       Update the edge cost c(u, v);
{43"}       if (c_old > c(u, v))
{44"}         if (u ≠ s_goal) rhs(u) = min(rhs(u), c(u, v) + g(v));
{45"}       else if (rhs(u) = c_old + g(v))
{46"}         if (u ≠ s_goal) rhs(u) = min_{s'∈Succ(u)} (c(u, s') + g(s'));
{47"}       UpdateVertex(u);
{48"}     ComputeShortestPath();
```

Figure 36: The pseudocode for the optimized version of D* Lite as presented in [5].

D Simulator code architecture

An overview of the different classes, their relationship and the functions used in the simulator can be found below. The two initial figures display the simulator as it was created in [1], while the last two figures display the final code architecture follow the updates made in the simulator.

D.1 Class diagram of simulator code

The class diagram of the initial simulator can be seen in Figure 37. The cleaning drone simulator was structured in the following way:

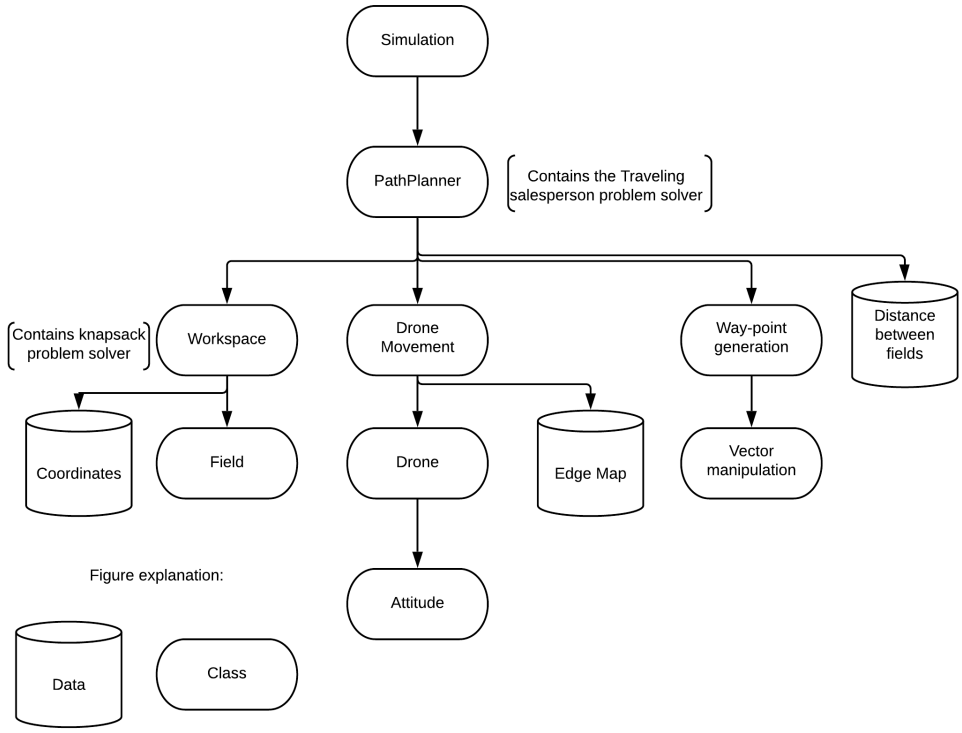


Figure 37: Class diagram of the code.

This can be regarded as three main branches which support the path planner, namely:

- The Areas branch, responsible for keeping track of the different areas.
- The Drone branch, controlling the drone movement

- The Waypoint generation branch, containing functions for waypoint generation.

D.2 Extended initial simulator class diagram

The classes found in Figure 37 with included variables and functions can be found below:

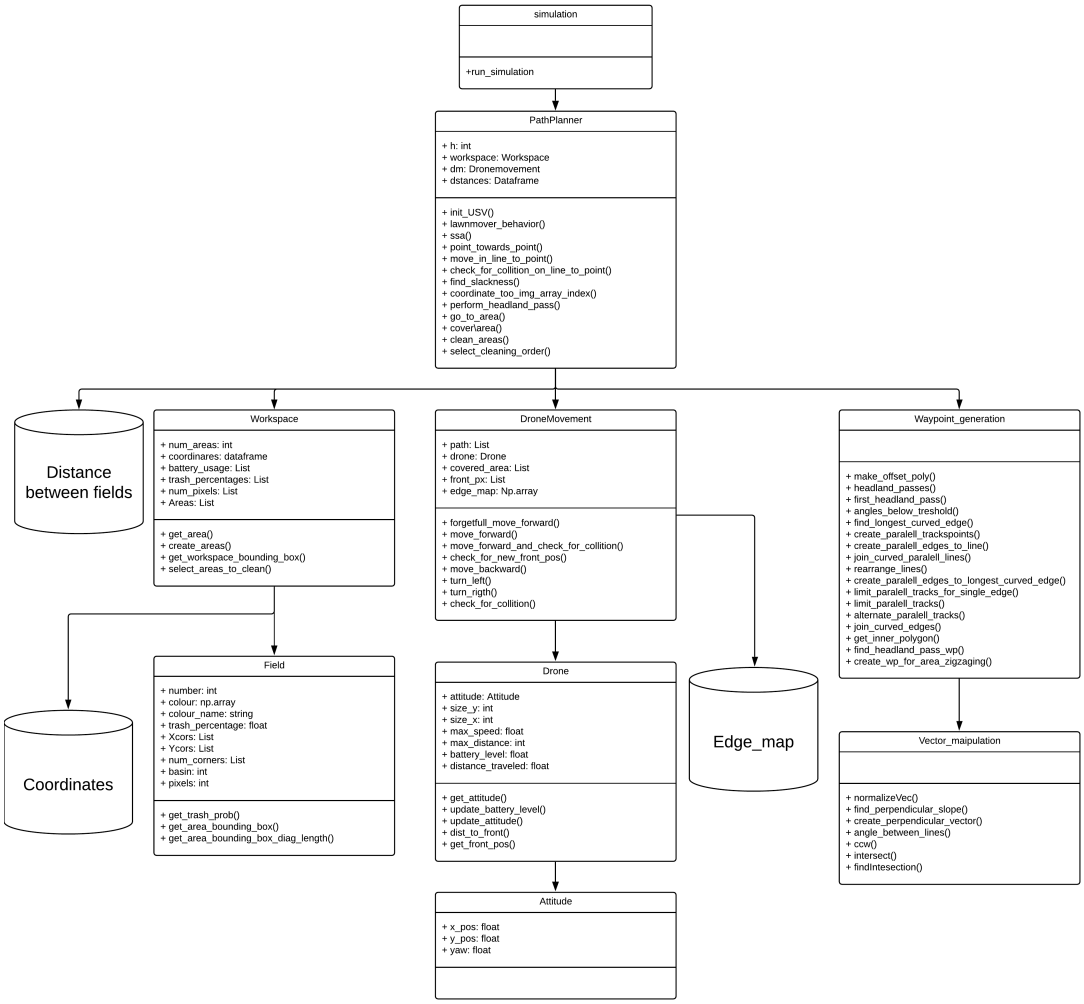


Figure 38: Extended class diagram of the code.

D.3 Class diagram of improved simulator

Below is the new structure of the code following the changes made in this thesis:

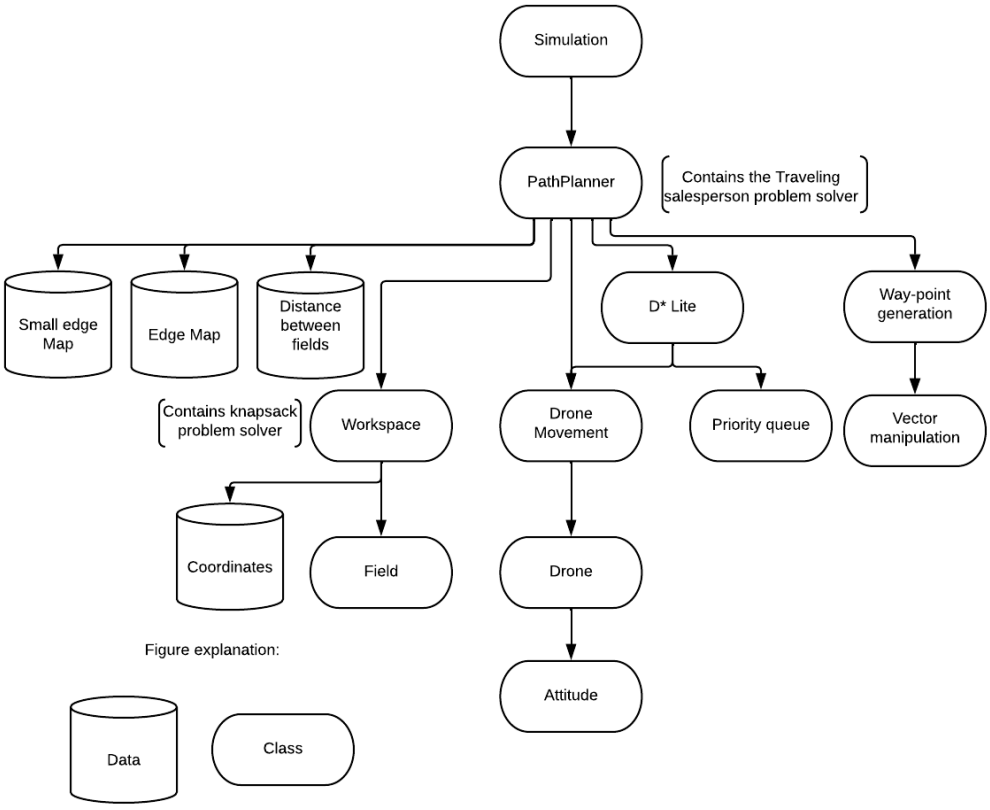


Figure 39: Improved simulator class diagram of the code.

D.4 Extended class diagram of improved simulator

Below is the new structure of the code following the changes made:

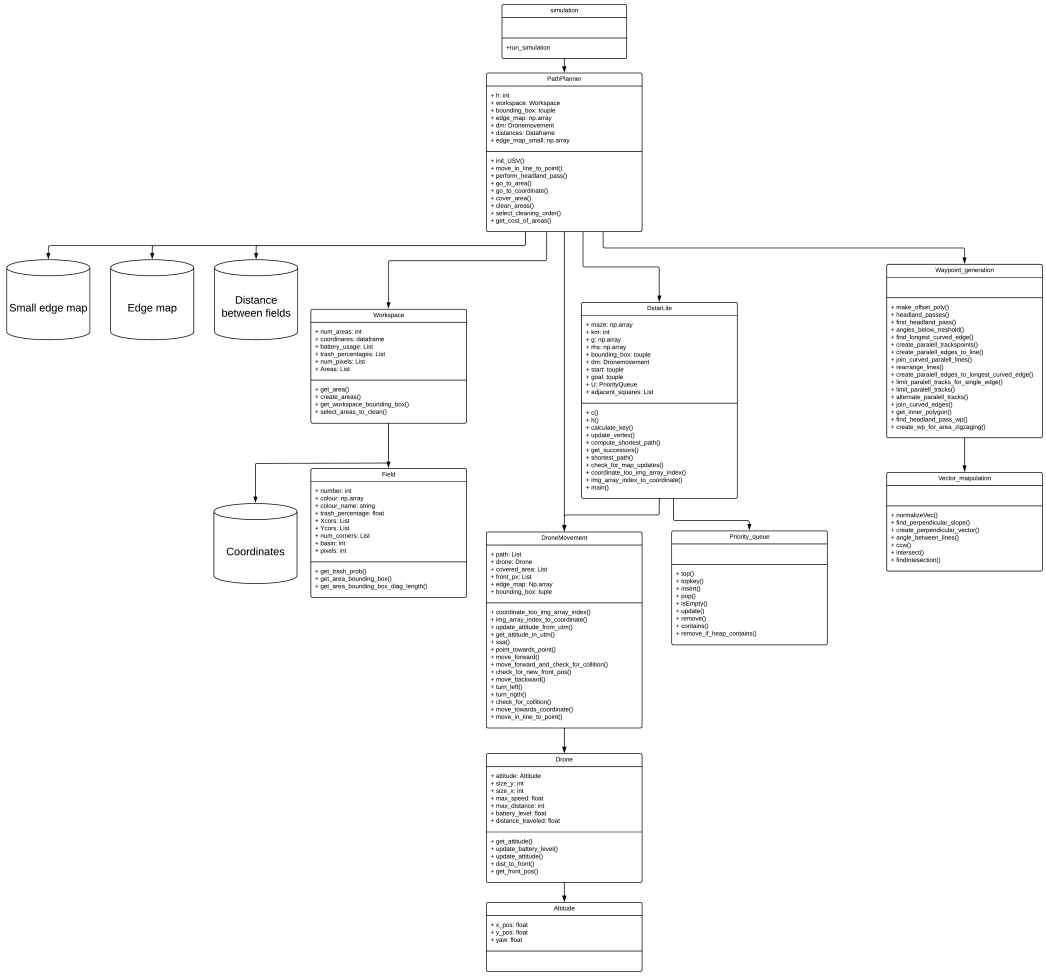


Figure 40: Extended class diagram of improved simulator.

E Software documentation

An overview of the different external packages used in this project and their version number will be presented in this chapter:

Packages used during the thesis

Package	Version
Python	3.9.7
Numpy	1.20.3
Pandas	1.3.3
Matplotlib	3.4.2
Tqdm	4.62.2
Pillow	8.3.1
Utm	0.7.0
Gmplot	1.4.1
Ubuntu	20.04.4 LTS
ROS	Noetic Ninjemys

Table 8: Packages used in the simulation.

The simulator is created to run cross platform, while the ROS implementation requires a linux OS. The linux version should preferably be Ubuntu 20.04 or newer, as ROS Noetic Ninhemys is targeted at this OS [59].

F Computer specifications

Both the simulation and the implemented guidance system was run on the authors computer, the specs of the computer used can be found below.

Specification	Value
OS	64-bit Ubuntu 20.04.4 LTS
Processor	Intel® Core™ i5-8250U CPU @ 1.60GHz × 8
RAM	8 GB
GPU	NV138 / Mesa Intel® UHD Graphics 620

Table 9: Hardware specifications.