ROYAL MILITARY ACADEMY

173th Promotion POL

Lieutenant Aviator Jan Olieslagers

Academic year 2022 – 2023

2nd Master

Development of a USV Platform and Navigation Method in a Dynamic Environment

Ensign Officer Cadet

Louis SOMERS

Royal Military Academy

Master Thesis of the department Mechanics
presented to obtain the academic degree
of Master in Engineering Science
under the supervision of Lieutenant Colonel Dr. Ir. Bart JANSSENS
Brussels, 2023

ROYAL MILITARY ACADEMY

Department Mechanics

Version 1.0

24/5/2023

Development of a USV Platform and Navigation Method in a Dynamic Environment

Louis SOMERS

Royal Military Academy

Abstract (English)

In recent years, unmanned (autonomous) surface vehicles or vessels (USVs) have gained a great deal of popularity. This is particularity highlighted in media for the aeronautics and automotive industry with the advancements in self-flying and driving drones. However, the maritime industry does not lag behind in this field. This can also be seen in Belgian Defence where the Belgian Navy will rely more and more on USVs to complete tasks. For example, the Minehunters will have access to USVs to aid in demining. For this advancement, new research platforms, as well as navigation software, are needed to further develop the capabilities.

In this thesis, two objectives for USVs were addressed. First, a navigation method for USVs in a dynamic environment was developed. The navigation method must dispose of a path generation and obstacle avoidance method. Second, a low-cost USV platform constructed from easily accessible materials was constructed. Additionally, the platform needed to be modular, such that components could quickly be interchanged or added.

The navigation method was implemented in a simulator using the open source Robot Operating System (ROS) and visualised with Gazebo 11. The implementation of the navigation method consisted of two main parts. The first was the implementation of a path generation method to navigate towards waypoints. This was done with simple Dubins path, which consists of an arc segment from the begin position that is used to give the USV the correct heading to then navigate in a straight line towards the end position. The second part of the navigation method focused on the detection and tracking of obstacles with a 2D LiDAR. The measurements of the LiDAR were grouped into clusters that represented objects. These clusters were compared over different LiDAR measurements and if the clusters corresponded to each other over the measurement, they were tracked. This track consisted of an object size and velocity estimation. With these tracks the trajectory of the object was predicted such that the USV could plan a path with it path generation method to avoid the object. The navigation method was tested with the three most frequent vessel-to-vessel interactions on sea. These are head-on on reciprocal courses, overtaking of a vessel and crossing of a vessel. In each situation, the USV was able to navigate towards the waypoint while avoiding the object in its respective situation.

The second objective, the development of a USV platform was done with hobby materials and affordable components. The USV is built with a boogie board and a watertight storage box on and in which different USV specific components like the thruster and autopilot are installed. With these components, a modular educational and research platform was created that could be used for different kinds of tasks, as sensors can be added, interchanged or removed. Furthermore, the USV was configured solely with open-source software to maximise the transparency and accessibility in hardware and software. Finally, the implementations done for the simulation were configured on the real-life USV. During the testing, the accuracy of the GNSS receiver that was used affected the reliability of the measurements, as the positioning was only accurate up to $2.5m$. This caused position jumps for the USV and the detected obstacles, such that safe navigation became difficult. This situation showed the difference between the^2 simulation, where there are little to no errors in the positioning, and the reality.

Abstract (Dutch)

De laatste jaren hebben onbemande en autonome voer- of vaartuigen (*Unmanned Surface Vehicle (USV)*) enorm aan populariteit gewonnen. Vooral aan de lucht en land applicaties werd veel aandacht besteed door de media. Dit werd vooral veroorzaakt door de vooruitgang op het gebied van zelfvliegende en -rijdende drones. De maritieme industrie blijft echter niet achter op dit gebied. Dit is ook te zien bij Defensie, waar de Belgische Marine meer en meer zal vertrouwen op USV's om taken uit te voeren. Zo zullen de mijnenjagers voorzien worden van USV's om te helpen bij het ontmijnen van zowel niet gedetoneerde munitie als mijnen op zee. Voor deze vooruitgang in technologie zijn nieuwe onderzoeksplatformen en navigatiesoftware nodig om de capaciteiten verder te ontwikkelen.

Dit proefschrift heeft twee vooropgestelde doelen met betrekking tot USV's besproken. Ten eerste werd een navigatiemethode voor USV's in een dynamische omgeving ontwikkeld. De beoogde navigatiemethode moest beschikken over een methode om een pad te genereren en obstakels te ontwijken. Ten tweede werd een goedkoop USV-platform uit gemakkelijk aankoopbare materialen gebouwd. Bovendien moest het platform modulair gebouwd worden, zodat verschillende onderdelen snel uitgewisseld,toegevoegd of verwijderd kunnen worden.

De navigatiemethode is geïmplementeerd in een simulator. De implementatie van de navigatiemethode bestond uit twee hoofdonderdelen. Het eerste was de implementatie van een padgeneratiemethode om te navigeren naar waypoints. Dit werd gedaan met een *simple Dubins path*, dat bestaat uit een boogsegment vanuit de begin positie die gebruikt wordt om de USV de juiste koers te geven zodat het vervolgens in een rechte lijn naar de eindpositie kan navigeren. Het tweede deel van de navigatiemethode was gericht op het opsporen en volgen van obstakels met een 2D LiDAR. De metingen van de LiDAR werden gegroepeerd in clusters die objecten voorstelden. Deze clusters werden vergeleken over verschillende LiDAR metingen heen en als de clusters van de verschillende metingen overeenkwamen, werden ze gevolgd in de tijd. Dit volgen bestond uit een schatting van de object grootte en een snelheids schatting. Met deze gegevens werd het traject van het object voorspeld zodat de USV een pad kon plannen om het object te vermijden. De navigatiemethode werd getest met de drie meest voorkomende interacties tussen schepen op zee. Deze zijn het frontaal naderen van een schip op tegengestelde koers, het inhalen van een schip en het kruisen van een schip. In elke situatie kon de USV naar het waypoint navigeren terwijl het object in de respectievelijke situatie vermeden werd.

De tweede doelstelling, de ontwikkeling van een USV-platform werd uitgevoerd met hobbymaterialen en betaalbare onderdelen. De USV is gebouwd met een boogie board en een waterdichte opberg doos waarop en -in verschillende USV-specifieke componenten zoals de voortstuwingsmotoren en de auto pilot zijn geïnstalleerd. Met deze componenten, werd een modulair onderwijs- en onderzoeksplatform gecreëerd dat voor verschillende soorten taken kan worden gebruikt. taken, aangezien sensoren kunnen worden toegevoegd, verwisseld of verwijderd. Bovendien werd de USV uitsluitend geconfigureerd met open-source software om de transparantie en toegankelijkheid van hard- en software te maximaliseren. Ten slotte werden de implementaties voor de simulatie geconfigureerd op de echte USV. Tijdens de tests beïnvloedde de nauwkeurigheid van de gebruikte GNSS-ontvanger de betrouwbaarheid van de metingen, aangezien de plaatsbepaling slechts tot $2.5m$ nauwkeurig was. Dit veroorzaakte positiesprongen voor zowel de USV als de gedetecteerde obstakels. Deze sprongen maakten veilige navigatie moeilijk. Deze situatie toonde aan dat er een groot verschil in simulatie, waar er weinig tot geen fouten in de plaatsbepaling zijn, en de werkelijkheid is.

Preface

This work would not have been possible with the aid and support of other persons. I am profoundly grateful to Lieutenant Colonel Dr. Ir Bart Janssens, my promoter, for his advice and feedback during the process. I also would like to express my deepest appreciation to Ir Tiên Thành Nguyên, the second reader of this work, for all the efforts and time he made to guide me. He was always available for discussions and gave his helpful insights when I came across issues during the research and writing of this work. Thanks should also go to the department of Mechanics for their feedback and input on the work. Finally, I also would like to thank my family and friends for supporting me along the way.

Contents

List of Figures

List of Tables

List of Abbreviations

2DRMS	Twice Distance Root Mean Squared
6DoF	Six Degrees of Freedom
AHRS	Attitude and Heading Reference System
AIS	Automatic Identification System
CEP	Circular Error Probable
DATMO	Detection And Tracking of Moving Objects
DRMS	Distance Root Mean Squared
ESC	Electronic Speed Controller
FOV	Field Of View
GNN	Global Nearest Neighbour
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HDMI	High Definition Multimedia Interface
IMU	Inertial Measurement Unit
LiDAR	Light Detection And Ranging
LiPo	Lithium Polymer
LOS	Line-Of-Sight
NUC	Next Unit of Computing
PID	Proportional-Integral-Derivative
PNT	Positioning, Navigation and Timing
PWM	Pulse Width Modulation
RC	Remote Control
RGB	Red, Green and Blue
ROS	Robot Operating System
RPM	Rounds Per Minute
RTK	Real Time Kinematic
SDK	Software Development Kit
SLAM	Simultaneous Localisation And Mapping
SPI	Serial Peripheral Interface

UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
USV	Unmanned Surface Vessel
UTM	Universal Transverse Mercator
VNC	Virtual Network Computing
VRX	Virtual RobotX

1. Introduction

1.1. Background

Unmanned Surface vessels (USV) are vessels that can act without a person on board. These vessels are often small in size and are deployed for research concerning the water quality and contents, seabed topography and structure or in other domains. With the development of unmanned vessels, the safety of operating personnel is increased because these vessels can be deployed in hazardous situations, for example, in demining, without endangering the life of its operators. These vessels are often still operated with a remote control (RC) connection, which means that there is an operator who steers and commands the USV. A large proportion of USV research focusses on the next step in the development of USV platforms, namely, the removal of the operator, so that the drone is unmanned and fully autonomous. When one speaks of an autonomous USV, it needs to posses a navigation method that can act under all circumstances and is able to handle obstacles when sailing towards its goals. To advance USV technology, easy access to USVs to develop software in real life is needed, as simulations alone are often insufficient. To meet the demand for easily accessible and affordable USVs , USVs made from widely available materials should be developed. With these affordable USVs research and education on robotics is made more accessible to students to train their skills.

Within Belgian Defence, USVs are also increasingly used by the Belgian Navy in its operations. These are particularly used on board of the current and future minehunters, as demining is a hazardous endeavour. When drones are used, these hazards are counteracted as the demining can be done from a larger distance. The addition of semi-autonomous or autonomous software also reduces the workload on the crew of the ships, as USVs can carry out certain tasks without human interaction.

1.2. Objective

This thesis has two main objectives. The first one is the development of a navigation method for a USV in a dynamic environment. A dynamic environment is an environment in which static as well as moving obstacles are present. The USV must be capable of detecting and predicting a short-term future path of these obstacles to avoid them while navigating towards a waypoint. The development of the navigation method is done in a simulation environment, using Robot Operating System (ROS) Noetic in which the method is tested on its behaviour in certain scenarios. The second objective is the construction of an affordable and highly customisable research and educational USV platform. The USV must be built with low-cost and easily accessible hardware and open-source software that allows for an easy expansion to tackle new tasks. The components used for this build will also be tested on their performance and compared with the performance in the simulation.

1.3. Outline

Each chapter in this thesis addresses a specific aspect of the development of the USV and its navigation method. First, a review of literature will be done to grasp the concepts of USV navigation and development. Secondly, the navigation method is developed, after which it is tested in simulation. Then, the construction and setup of the USV is explained. Finally, some experiments with the sensors and software will be performed.

- Chapter 2 - Literature review: gives background information and theory about the topic of the thesis. Here, methods for USV navigation and development used in the research domain are discussed.
- Chapter 3 - USV navigation: explains the navigation method that is developed. First, the interpretation of the environment around the USV is discussed. Then the detection,tracking and prediction of the trajectory of obstacles in vicinity of the USV is analysed. Finally, the path generation and navigation method with obstacle avoidance are explained.
- Chapter 4 - Simulation: explores different USV simulators, after which one is chosen and used to implement the USV navigation method. This method is then tested in different scenarios.
- Chapter 5 - USV development: describes the sensors and equipment used for the USV and documents the construction and configuration of all the components.
- Chapter 6 - Experiments: presents the results of the testing of the sensors and the real world experiments.
- Chapter 7 - Conclusion and outlook: gives a summary of the work and draws a conclusion from it. Furthermore, possibilities for further work are discussed.

2. Literature review

This chapter will explore which methods are used for navigation and object detection by USVs as well as methods to develop a low-cost autonomous USV. In Section 2.1, methods for navigation are discussed. Section 2.2 gives insight in the available USVs, the methods and materials to develop one from scratch and explains the purpose of the sensors used on USVs. Here, the low-cost aspect of the USV is emphasised. However, the inexpensiveness of the USV may not interfere with the ability to act autonomously.

Unmanned surface vehicles or vessels are vessels that can navigate without a person on board. These USVs can be steered with a controller by a person or can be fully autonomous. A completely autonomous USV is able to navigate towards its destination without the aid of a person. Autonomous and controlled USVs have been the subject of a lot of research in recent years as there appear more and more semi-autonomous or fully autonomous vessels in the marine environment [1]. As stated by Liu et al. [2], the further development of autonomous USVs will entail tremendous benefits as the range in which USVs can be deployed is extended and the safety of operators is increased as the USV can act autonomously and if necessary be controlled from a distance. A popular and important part of USV research is navigation and obstacle avoidance.

2.1. USV navigation

Autonomous navigation of a USV consists of multiple processes that work together to achieve a safe navigation towards a waypoint in the end. First, the USV must be aware of its own position and its surroundings. As stated by Specht et al. [3] GNSS receivers are commonly used for the positioning of USVs, while objects in the surroundings are detected with a LiDAR. LiDARs are widely used for object detection and avoidance, as discussed by Peng et al. [4]. The USV then needs to be able to process the measurements of the surroundings to detect the obstacles and it needs to track them over time to estimate their behaviour. When this is done, a suitable path towards the waypoint while avoiding the obstacles needs to be constructed. The different parts of navigation are split up and explored further on.

In Figure 2.1, a typical guidance, navigation and control loop for a USV is shown. This loop is a complete representation of what it takes for a USV to navigate autonomously.

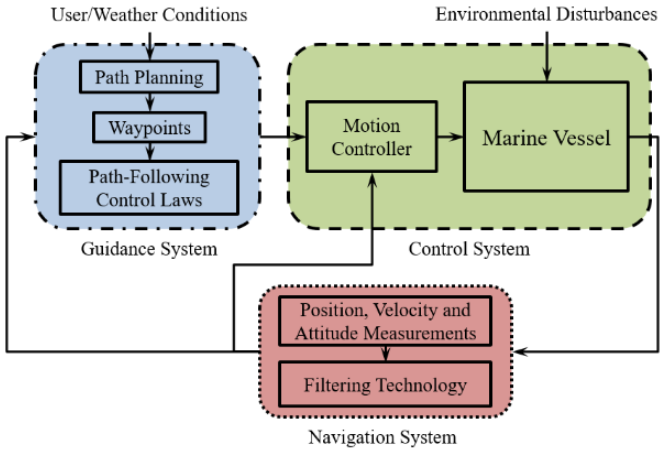


Figure 2.1. A guidance, navigation and control loop for a USV[5]

The guidance system depends on the input of the user and the weather conditions. It will generate the path that the USV should follow. The guidance system receives its input from the navigation system. The navigation system is responsible for collecting data using sensors. These sensors could for example be a GNSS receiver to locate the USV or a LiDAR to detect obstacles in the environment. The measurements of the sensors are filtered to get rid of as much noise as possible, for example, with Kalman filters. The output of the navigation system, together with the output from the guidance system is then used in the control system. The control system converts the received inputs into commands to the vessels actuators such that the objectives of the USV can be achieved. These objectives can be navigating towards a waypoint, hence following a generated path. The delivered commands are then performed by the USV and the loop start over again [5].

2.1.1. Occupancy grid mapping

The detected obstacles and the USV need to be mapped and located in an occupancy grid that can easily be processed by the USV. This grid can then be used in the path generation method to assign grid zones with obstacles and free zones. The processing of the occupancy grid should be as large as the range of the sensor with the farthest reach, which is in the case of this thesis the LiDAR. There are different methods to subdivide the world into smaller processable zones. With a grid-based occupancy mapping, the world is divided into a collection of uniform cells. These cells can be of different sizes and shapes. The size of such a cell can be determined by the size or the manoeuvrability of the USV. Examples of occupancy grids are with circular cells as proposed by Guo and Qu [6] or square cells as proposed by Xing et Qu [7]. In reality, the shape of the cells could be anything that fits the needs of the vehicle that needs to process it.

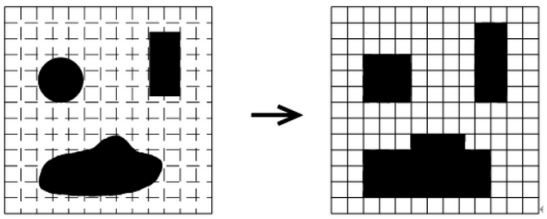


Figure 2.2. Rectangular occupancy grid[7]

2.1.2. Obstacle detection and tracking

When a USV needs to navigate autonomously, it is important that obstacles are detected and tracked to be able to successfully avoid a collision. In this process, an important distinction in difficulty between a static and dynamic environment is made. This is explained in more detail in this subsection.

Static environment

When a USV is in an environment known to be fully static, the obstacle detection is quite straightforward. It is not necessary to track the objects as they will never move and the output of the LiDAR can immediately be used as obstacle detection. The range measurements of the LiDAR are mapped on the occupancy grid, whereafter the cells that are occupied with objects are marked as blocked. In a static environment, it is also less important to identify and classify the objects as isolated and independent obstacles. It suffices to detect that something is on the path of the USV and that it needs to be avoided. Lenes [8] used the direct measurements of the LiDAR to detect the obstacles. Each individual measurement of the LiDAR is mapped to a geographical position and assumed to be an object. Each measurement is subsequently enlarged to account for a safety zone around the obstacle. A cell in which either measurements of the LiDAR or the enlarged part of a measurement is positioned is consequently blocked. In this manner, the obstacles are detected and mapped to the occupancy grid that is used.

Dynamic environment

In a dynamic environment, the detection and tracking of objects is not as straightforward. There are multiple possibilities for the USV to receive information about objects and obstacles in its surroundings.

Liu et al. [9] discuss dynamic obstacle detection and tracking on a global maritime basis by using data that the objects, ships in this case, send out themselves. This information is required by law to be sent out using the automatic identification system (AIS). In an AIS message, the dimensions, heading and velocity of the ship are included. With this information, the ships can be tracked and the trajectory can be predicted. The information of static objects and thus the environment, is loaded into the USV with electronic charts. This thus means that the operating environment must always be known and it is assumed that every dynamic obstacle can be identified with AIS data. This is absolutely not the case in this thesis and this is thus not a suitable solution to tackle obstacle avoidance.

When the obstacles do not send out information like in the case discussed above, own detection means such as a LiDAR need to be used. The LiDAR will receive reflections back from objects; these objects then need to be detected, categorised and tracked. This means that between successive laser pulses objects need to be associated with each other. Data from objects that are no longer in range need to be removed from the memory, while others still need to be tracked or added to the memory in case that they are new objects. Konstantinos [10] has developed a system capable of detecting and tracking moving vehicles for 2D LiDAR systems. The system relies on the clustering of LiDAR data in groups that each represents an object. After that, features like corners and L-shapes are extracted from these clusters. The clustering of objects is shown in Figure 2.3. These features are then fed to the tracking process which tracks the object based on an unscented Kalman and Kalman filter. An example of his developed tracking is shown in Figure 2.4. With the extracted L-shapes, a box is fitted to estimate the size of the objects. These boxes are shown in Figure 2.5. Additionally, the velocity is estimated through the movement of the tracked corner. The estimated velocity is shown by the arrow in Figure 2.5.

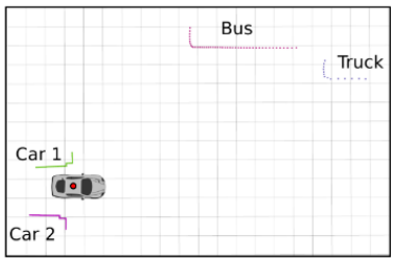


Figure 2.3. Clustering of the detected objects[10]

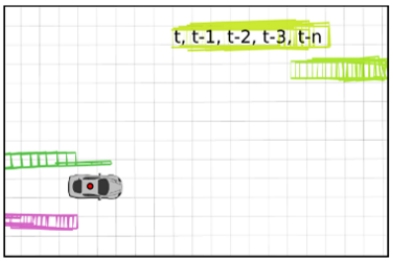


Figure 2.4. Tracking of the objects through time [10]

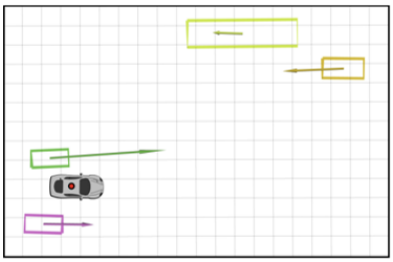


Figure 2.5. Estimated size and velocity of the objects[10]

A similar method to the method developed by Konstantinos is discussed by Thun and Puente Leon [11]. They also segment the data into clusters after which a probabilistic data association between measurements is made. This means that with a slightly different-shaped cluster, the clusters between measurement can still be linked to each other. If the change in shape is sufficiently slow between multiple measurements, the object will be tracked without interruption. Their implementation is aimed at detecting and tracking vehicles, so they also extract corners and L-shapes from the clusters. The methods of Thun and Puente Leon and Konstantinos offer the advantage that objects can be tracked continuously if the change of the detected shape is slow enough. This is the case for USVs under normal circumstances.

Mihálik et al. [12] also discussed the detection of dynamic objects with a 2D LiDAR. They implemented the tracking of objects by extracting three successive LiDAR measurements and performing scan matching. Each LiDAR measurement was segmented into point clouds after which groups of points were clustered. These clusters represent the objects. The clusters from the three different measurements were

then compared to each other to find matching clusters. When the clusters were matched to clusters from another measurement, one could say that those clusters are tracked. A major disadvantage of this method is that the clusters need to be almost completely the same between measurements; otherwise, the clusters cannot be matched.

2.1.3. Path generation methods

To navigate to the waypoints autonomously, a method to generate a feasible path must be defined. The generated path must be viable with the characteristics of the USV, but preferably it must also be an optimal method to navigate towards a waypoint. In this section, different path generation methods for autonomous vehicles are explored.

Bezier curves

A possible solution to the path generation is the use of Bezier curves as done by Choi et al. [13] in their path planning method. These curves are constructed with a set of control points for which only the first and last control points are on the defined curve. These points coincide with the starting and end position of the USV, respectively. The other control points will never coincide with the curve. An example of a curve is shown in Figure 2.6. Bezier curves are especially useful for their computational efficiency and simplicity. Another advantage is the curvature continuity, which means that a USV does not need to slow to a halt to change its heading and again needs to accelerate from nothing. The method, however, is seldom used due to the imposed curvature, jerking and lateral accelerations it causes [14].

A Bezier curve of the nth degree is calculated as follows [13]:

$$P_{[t_0,t_1]}(t) = \sum_{i=0}^{n} B_i^n(t) P_i \tag{2.1}$$

with P_i control points such that the equations $P(t_0) = P_0$ and $P(t_1) = P_n$ are satisfied and $B_i^n(t)$ is a Bernstein polynomial.

$$B_i^n(t) = \binom{n}{i} \left(\frac{t_1 - t}{t_1 - t_0}\right)^{n-i} \left(\frac{t - t_0}{t_1 - t_0}\right)^i \quad with\ i \in \{0, 1, ..., n\} \tag{2.2}$$

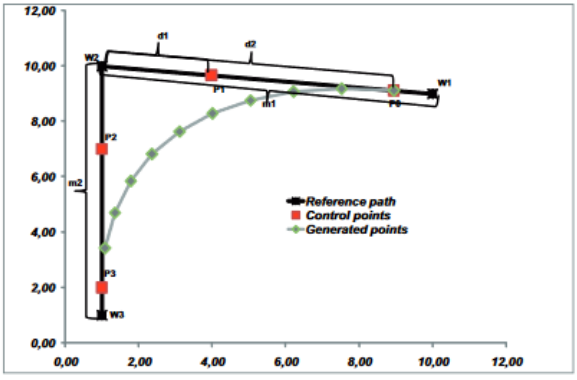


Figure 2.6. A generated path using Bezier curves [14]

Bezier curves are relatively easy to compute, but they have a complete lack of local control. When a control point of the Bezier curve is adapted, the whole curve will change. This is caused by the fact that the points on the curve are the weighted average of the control points and when the control

point is adapted, the whole curve will thus change [15]. This is to be avoided when one wants smooth navigation. This lack of local control could be partially solved by using piecewise Bezier curves, where the large trajectory is split into small parts. However, this will add complexity to the computation of the trajectory. Bezier curves are thus less interesting because of their hard relation between the amount of control points and the degree of the polynomial that is used [15].

Basis spline curves

Basis spline or B-spline curves are also a possibility to generate a path. They are generally considered as an extension to the Bezier curves [16]. B-spline curves are implemented by Tran et al. [17] for their USV. They used B-spline curves for a path planning method that includes obstacle avoidance. These curves consist of piecewise polynomials that are characterised by the convex combination of control points, which are weighted by a B-spline function. The use of piecewise polynomials allows to describe more complex paths while the polynomial degree is kept low. The piecewise curves are described by a polynomial of degree k, a set of $n+1$ control points and a set of $m+1$ knots, which define the point where the polynomial is changed. The vector of knots must satisfy $(m+1) > (k+1)$ and there must be a set of $(n+1) = m-k$ B-spline functions. In this way, the number of control points and knots can be computed for polynomials of degree k. An example of such a curve is given in Figure 2.7.

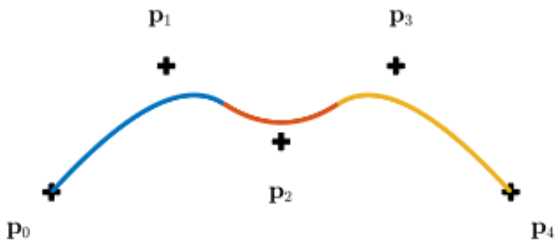


Figure 2.7. A B-spline curve [16]

The use of B-spline curves is also suggested by Rousseau et al. [16] for their quadrotor flight plan. In their work, they used these curves with multiple waypoints to generate a B-spline curve to navigate through corridors.

The advantages of B-spline curves are that they offer more flexibility for modification of control points over a Bezier curve and that they have more diversity in the way the total curve can be build. This makes B-spline curves thus more efficient than Bezier curves [15]. However, B-spline curves are more complex because each piecewise curve has its own polynomial.

Dubins path

The shortest curve that connects two points with a given heading and a given limit on the curvature is given by a Dubins path [18]. The Dubins path is often used in path planning and navigation when a maximal yaw rate is imposed [19]. This is the case for USVs. Furthermore, the calculation of the path is straightforward and simple. The path consists of at most three different segments, of which two are arc segments and one is a straight line. The path is generated with the initial and final positions and the tangents of these positions and a constraint on the turning radius of the USV. The method also assumes that the USV is only capable of moving forward to travel towards the final positions. An example of a Dubins path is shown in Figure 2.8. When a USV is also capable of reverse motions, it can generate a path with Reeds-Shepp's path [20].

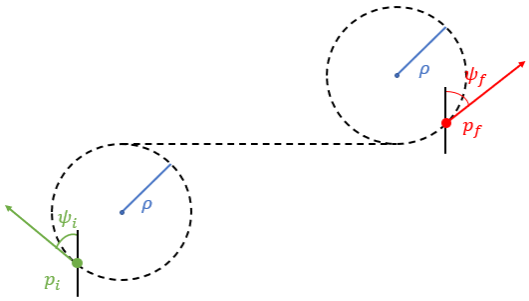


Figure 2.8. Dubins path geometry

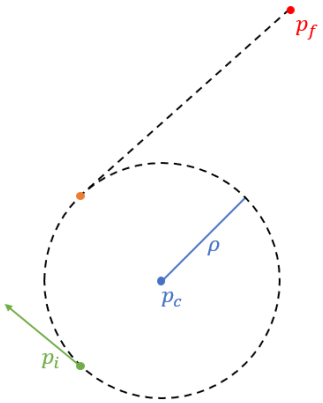


Figure 2.9. Simple Dubins path geometry

A variation of the method is the simple Dubins path [8]. In this method, the desired final heading is removed from the inputs, in order to simplify the method. This means that the second arc segment is no longer necessary because the final heading is determined by the heading on the straight line segment. The method also needs a minimal turning radius that is smaller than half the distance between the initial and final positions. The simple Dubins path is shown in Figure 2.9.

2.2. USV development

A USV platform on which the implemented navigation method will be tested will also be developed. There are various USVs readily available for purchase. These include the Otter USV [21], for example. However, the goal is to build and tune a USV from individual components. The aim of the construction of the USV is also to develop an affordable and highly modular research and educational platform that could be used for different tasks in the future. Finally, this section also explores the different sensors that a USV often has.

2.2.1. Otter USV

The Otter USV is a fully developed USV that is available for purchase. The USV comes with a fully configured guidance, navigation and control loop and is immediately ready for deployment. When operating this USV, one does not need to look at how the USV navigates but can focus completely on their research objective. The Otter USV can be equipped with multiple sensors and is often equipped with a SONAR to map the seabed. However, the USV is not a low-cost solution and certainly not a small and lightweight one. It has a footprint of $200x108x81.5cm$ and weighs $55kg$ [21]. The Otter USV is shown in Figure 2.10.

Figure 2.10. The Otter USV [21]

2.2.2. Self-developed USVs

As already said, the goal is to develop a USV from scratch. An example of this is the USV developed by Setiawan et al. [22]. They constructed a prototype USV with a Pixhawk autopilot, a compass and GNSS receiver. These sensors enable the USV to navigate autonomously from waypoint to waypoint. The thrust is delivered by two T200 thrusters from Blue Robotics [23]. The USV they have developed is shown in Figure 2.11. The USV is a catamaran-based design that minimises roll of the USV when navigating. The control of the USV is done with a proportional-integral-derivative (PID) control loop. They obtained the results for the control loop experimentally and put them into the autopilot program Mission Planner [24]. With this program autonomous navigation can be achieved when all the parameters are correctly configured for the control of the USV.

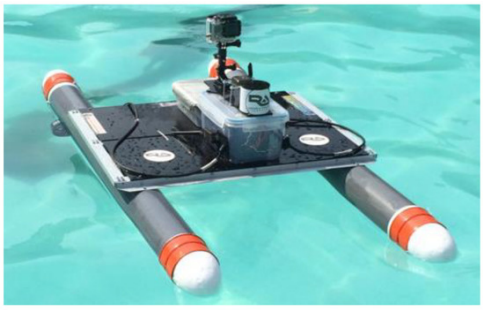
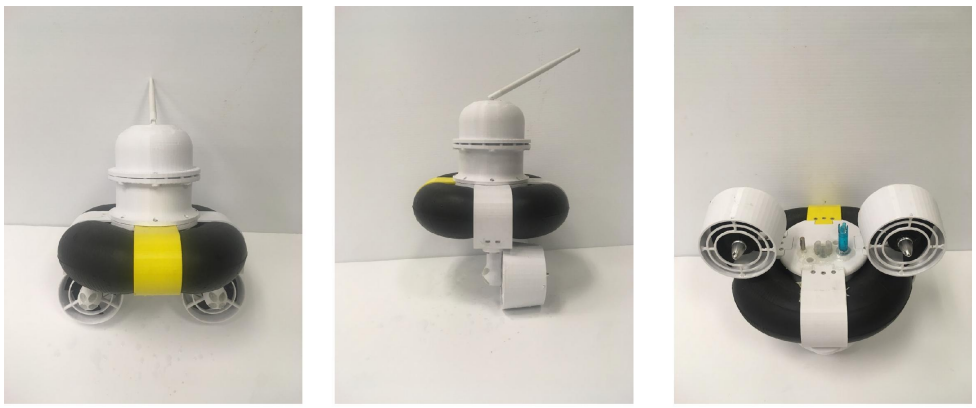


Figure 2.11. The self-developed USV from Setiawan et al [22]

The goal of this USV development was to obtain a low-cost method of constructing USVs. This is achieved because the total cost of the USV is only $524,97 [22].

Another low-cost self-developed USV is the USV designed and constructed by Jo et al. [25]. Their aim was to construct a small USV to monitor the quality of the water in lakes. They are the first to create a USV that is completely open-source and can thus be constructed by everybody. The USV is propelled by two fixed thrusters mounted underneath the USV and is kept afloat with a round blown-up tube. 90% of the components of the USV were 3D printed, which kept the cost of the USV down to $200, hence the goal of a low-cost solution is obtained. The USV is equipped with a GPS, water temperature and pH sensor. The GPS data is used to position itself and to navigate towards an objective with the software application Robot Operating System (ROS). The USV only uses GPS data to create the trajectory towards the waypoint, which means that the heading is calculated from the combination of different previous positions. This leads to a less precise navigation than with an IMU and GPS combined. The USV is shown in Figure 2.12



(a) (b) (c)

Figure 2.12. The self-developed USV from Jo et al [25]

Both of these USVs can navigate autonomously as long as there are no obstacles between the waypoints. They have a total lack of obstacle avoidance in the sense that it was not implemented and they also do not possess the sensors that could enable obstacle avoidance. In this thesis, obstacle avoidance will be implemented with a 2D LiDAR.

All these USVs, including the Otter USV, have two fixed thrusters, which means that the thrust must be different between both thrusters to allow for turning. This is done with skid-steer, which is explained in Chapter 5.

2.2.3. Sensors

The unmanned surface vessel will and needs to receive information about its surroundings and position through different sensors. The information that it receives will form the basis on which it will execute its tasks. The USV will be equipped with an IMU , LiDAR and GNSS receiver. With these sensors, the USV receives accurate and current information about the environment in which it operates. In this section, these sensors and their functioning are described in more detail.

IMU

An inertial measurement system (IMU) is a system that measures motion data of the object to which it is attached. It typically consists of gyroscopes, accelerometers and optionally magnetometers [26]. The gyroscope measures the angular velocity around the three axes (x, y, z), while the accelerometer measures the specific force across these axes. The optional magnetometer measures the Earth's magnetic field and provides a heading, this is a similar functioning as a compass. The data from the three devices are often fused to provide complete information on the orientation and heading of the system in an attitude and heading reference system (AHRS) . This is thus used to give information about the movement and rotation of an object in the six degrees of freedom (6DoF). These six degrees are movement in the three axes and pitch, yaw and roll rotations and are shown in Figure 2.13.

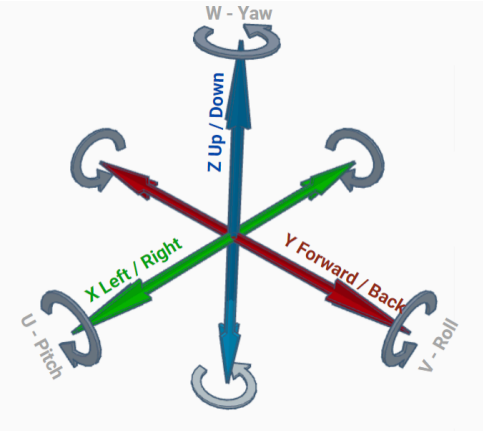


Figure 2.13. The six degrees of freedom[27]

LiDAR

Light detection and ranging is a remote sensing application that uses laser pulses to measure ranges to objects [28]. A LiDAR laser scanner sends out laser pulses that reflect on the surface of an object or the earth. Part of the reflection will be back in the direction of the LiDAR where it is received. The LiDAR gets an echo back at an angle in relation to the axis of the LiDAR and at a timestamp. Based on the timestamp of the sent-out pulse at that angle, a range to the object can be calculated. A LiDAR is characterised by its pulse repetition frequency and its ability to look in 2D or 3D. The pulse repetition

frequency also determines the maximal range because when an echo of a pulse is received after a new pulse is sent out, it is impossible to know that the echo corresponds to the previous pulse. The ability to look in 3D gives the LiDAR a much broader field of applications. A low cost 2D LiDAR is sufficient for this thesis, because the LiDAR is used to only detect an object and the USV can be assumed to move in a 2D plane when the velocity of the USV and sea state are not too high. Furthermore, the range of the LiDAR does not need to be large; the maximal 40 metre range of the S1 RPLIDAR suffices.

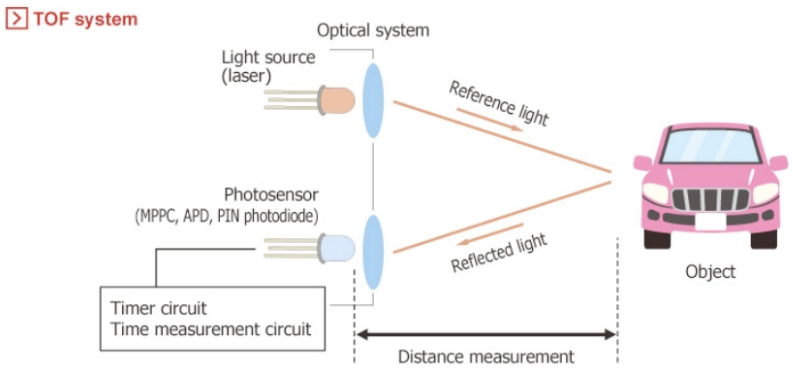


Figure 2.14. Functioning of a LiDAR[29]

GNSS systems

A global navigation satellite system or GNSS is any satellite constellation that provides positioning, navigation and timing (PNT) and this on a global or regional basis [30]. These satellites send out messages that can be captured by GNSS receivers. The receivers can, based on multiple received messages from different satellites, determine their position in the world. This position determination consists of a latitude, longitude and altitude on a high precision level of some metres. The precision can further be augmented by using real time kinematic (RTK) to achieve an accuracy level of 1cm. This is done by incorporating a correction stream that is received over an internet connection or long distance radio. There exist multiple GNSS systems from different countries. The most used systems are GPS from the United States of America, Galileo from the European Union, Glonass from the Russian Federation and BeiDou from the People's Republic of China. In this thesis, the position of the USV will be determined using the GPS and Galileo constellations.

3. USV navigation

In this chapter, a suitable navigation method for a dynamic environment will be developed. In Section 3.1, a manner to divide the surroundings will be discussed. Then in Section 3.2, a way of detecting and tracking dynamic obstacles will be explained. Finally, in Section 3.3, a path generation method to navigate to waypoints will be developed. This section will also deal with the obstacle avoidance.

3.1. Mapping of the surrounding

Before being able to handle navigation and obstacle avoidance, the USV should be able to position itself in the world and map the surroundings. This is achieved by using the LiDAR, GNSS receiver and IMU and will be discussed in this section.

3.1.1. Positioning

The USV is able to position itself in the world with the help of a GNSS receiver, compass and IMU. The GNSS receiver reports the position of the USV in the geographic coordinate system using latitude and longitude. These coordinates are then converted to map coordinates using the Cartesian coordinate system. The origin of the system is determined with the first position fix it receives from the GNSS receiver, hence the origin of the Cartesian system corresponds to the starting position of the USV. The orientation, angular velocity and linear acceleration of the USV are determined with the IMU. With these two sensors, it is possible to first locate the USV in the world and thus also in the map used and, secondly, to determine the correct heading and looking direction of the USV for path generation and detection of the surroundings. The USVs position and heading are shown in Figure 3.1 as an orientated origin of its local coordinate system. This local coordinate system is dependent on the global one of the map, to locate itself in the map.

3.1.2. Mapping for navigation

While navigating, the only new information about the surroundings that comes in is from the LiDAR. Therefore, it is more efficient to only take into account the cells that are in range of the LiDAR. This is achieved by applying a rolling window that moves together with the USV. The rolling window is a circle with the centre the LiDAR of the USV and a radius of the maximal range of the LiDAR. Because the LiDAR used is a 2D LiDAR, the moving window is restricted to the x and y-axes. A representation of the rolling window with a range of $25m$ is shown in Figure 3.1.

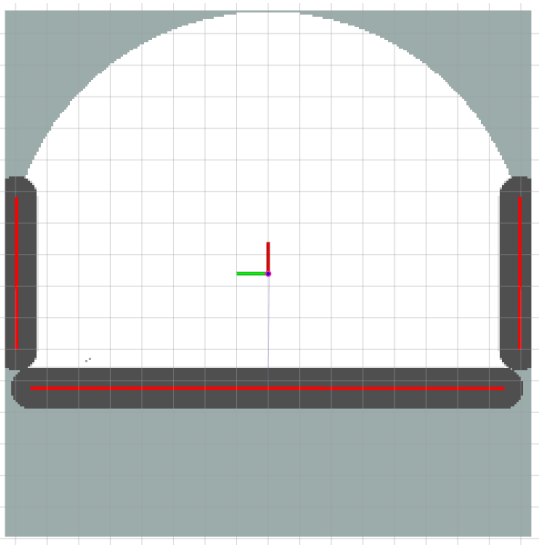


Figure 3.1. The rolling window with radius $25m$. The black cells are blocked, the white free, the grey unknown and the red points are LiDAR measurements.

As can be seen in the figure, the cells in the occupancy grid can have 3 different colours. These colours correspond to a probability value between 0 and 100. The grey coloured cells have a value of -1, because their status is unknown. This is caused by the fact that they are or not visible to the LiDAR because an obstacle is in front of them or that they are out of reach of the LiDAR. All cells start out as unknown. The white coloured cells have a probability value of 0 to 49 and are assigned as free. This means that the USV will consider them as navigable. The black coloured cells, finally, have an assigned value of 50 to 100 and are blocked. These cells need to be avoided by the USV. This thus corresponds to the following:

$$Cell = \begin{cases} \text{Free} & \text{if } 0 < value \leq 49 \\ \text{Blocked} & \text{if } 49 < value \leq 100 \\ \text{Unknown} & \text{if } value = -1 \end{cases} \tag{3.1}$$

When constructing the occupancy grid, a balance between processing and accuracy must be made. On the one hand, if a high resolution grid is built, each obstacle would be accurately presented in the grid, but the processing by the USV would take a long time and it is possible that the USV can not take decisions about the navigation in time. On the other hand, if the cells are large, the processing would be quick but the obstacles would not be positioned accurately. That is why a resolution of $20cm$ is chosen to have a sufficiently accurate position of the obstacle, while still being processable. The obstacle will also be inflated to account for a safety margin between the USV and the obstacle at all times during the navigation.

Obstacle inflation

While navigating, the USV needs to stay clear of the obstacles at all times. If only the obstacle is put in the occupancy grid, there is a risk that the USV will come too close to the obstacle and in the worst case due to some circumstance will collide with the obstacle. That is why each obstacle is enlarged with an inflation radius in the occupancy grid. Each cell that is inside this inflation radius will be assigned the blocked status. The inflation of an obstacle is shown in Figure 3.1 by the black cells around the red LiDAR measurements. The inflation radius is a value that can be tuned and is set by the user. In this thesis, the inflation radius is set to $2m$, as will be explained in Chapter 4.

The obstacle inflation and rolling window are constructed with the package *costmap_2D* in ROS [31].

3.1.3. Cell partitioning

The constructed occupancy grid is not suitable for navigation, due to the small cell size. That is why the map that will be used for navigation is partitioned into larger cells that are assigned a state calculated with information extracted from the occupancy grid. The map is partitioned into circular cells with a method proposed by Guo and Qu [6]. When using a map with larger cell sizes, the safety zone around an obstacle is enlarged, because a larger partition cell will be blocked if it contains a blocked occupancy grid cell. Furthermore, the implementation of a navigation method is simpler when the USV can navigate towards these larger cells instead of finding the small free occupancy grid cells to navigate to.

Circular cell partitioning

The method chosen for this thesis is thus the use of circles to partition the map used for navigation. This map must be covered by a minimal number of circles. A method to cover a rectangle, the navigation map, with this minimal number of circles is proposed by Guo and Qu [6] and is as follows. When a rectangle with lengths x_l and y_l along the x and y-axis is given, the rectangle can be covered by circles with a radius r. These circles are positioned along lines parallel to the x and y-axis and the distance between the centre of adjacent circles is equal to $\sqrt{3}r$. In this way m columns can be positioned inside the rectangle such that the centre of the circles between each column is $\frac{2}{3}r$. The amount of columns m with circles is then computed by:

$$m = \begin{cases} \lfloor \frac{x}{\frac{3}{2}r} \rfloor + 1 & \text{if } \frac{x}{\frac{3}{2}r} \leq \frac{2}{3} \\ \lfloor \frac{x}{\frac{3}{2}r} \rfloor + 2 & \text{otherwise} \end{cases} \tag{3.2}$$

While the amount of circles (n) in each row is calculated as follows:

$$m = \begin{cases} \lfloor \frac{y}{\sqrt{3}r} \rfloor + 1 & \text{if } \frac{y}{\sqrt{3}r} mod 1 \leq \frac{1}{2} \\ \lfloor \frac{x}{\sqrt{3}r} \rfloor + 1 + (\mathrm{l} \cdot \mathrm{mod} 2) & \text{otherwise} \end{cases} \tag{3.3}$$

Where $\lfloor x \rfloor$ is the floor function and mod is the modulo function. Now, the position of each circle has to be calculated. This is done by determining the centre position of each circle. The centre position of a circle somewhere from column $1 \leq l \leq m$ and row $1 \leq k \leq n$ is calculated as follows:

$$[x^{kl}, y^{kl}] = \begin{cases} [(\frac{3}{2}l - 1)r, (k-1)\sqrt{3}r] & \text{if } l \; mod 2 = 1 \\ [(\frac{3}{2}l - 1)r, (k - \frac{1}{2})\sqrt{3}r] & \text{otherwise} \end{cases} \tag{3.4}$$

An example of such a cell partitioning is shown in Figure 3.2.

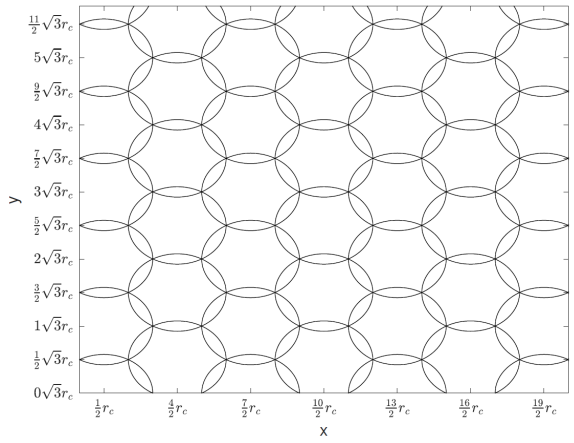


Figure 3.2. The map covered by a minimum number of circular cells [8]

With these equations, the map is divided into cells that can be used for navigation. Each cell is assigned a certain state, just like the cells in the occupancy grid. The possible states are *unknown, free, blocked* or *predicted*. The state is set to *unknown* when there is no information about the cells of the occupancy grid that fall into this partition cell. It is set to *free* when all the cells of the occupancy grid are free, hence there is not an obstacle inside the partition cell. The state *blocked* is assigned when as little as one occupancy grid cell has the blocked value in the partition cell. Finally, the cell has the state *predicted* when one of the prediction points of the moving obstacle falls within the partition cell. The prediction of the moving obstacle is explained in Subsection 3.2.3.

$$\text{Possible cell states} = \begin{cases} \text{Unknown} \\ \text{Free} \\ \text{Blocked} \\ \text{Predicted} \end{cases}$$

An example of all these states is shown in Figure 3.3. The red cells are or blocked cells or predicted cells. The grey cells are unknown, while the blue ones are free. The green cells are cells where the USV has already travelled. These cells have a different colour, but their state remains unchanged from their original state.

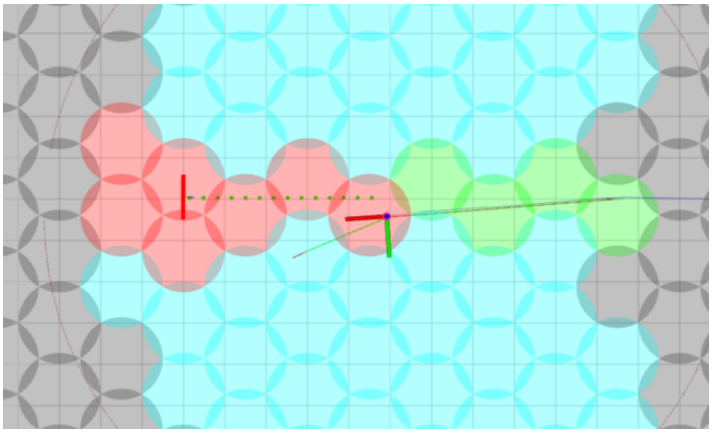


Figure 3.3. The partitioned map. The grey circles are unknown, red are blocked or predicted cells, blue are free cells and green are cells where the USV has passed through.

When choosing the size of the partition cells, different parameters have to be taken into account. First, the footprint of the USV must be able to fall completely inside one cell. In this way, it is possible to position the USV in a free cell surrounded by blocked cells. Secondly, the Path of the USV must be able to be generated in free cells at all times, hence the turning radius of the USV must be taken into account when selecting the cell size. The chosen size of the cell is explained in Chapter 4 for the simulation and in Chapter 6 for the real-life experiments.

3.2. Dynamic obstacle detection

In this section, a method for obstacle detection and subsequent tracking of these obstacles is explained. Furthermore, the prediction of a future trajectory of the tracked obstacle is made.

3.2.1. Object detection

The USV is capable of detecting objects with the aid of a 2D LiDAR. The LiDAR will send out a laser beam in a rotational motion and in a single horizontal plane and will capture the echos of this beam. The angle and time of arrival of the echo are registered and compared to the beam that was sent out. In this way, the distance and angle of an object relative to the LiDAR is registered and the object can be located in space. An example of LiDAR output is shown in Figure 3.4.

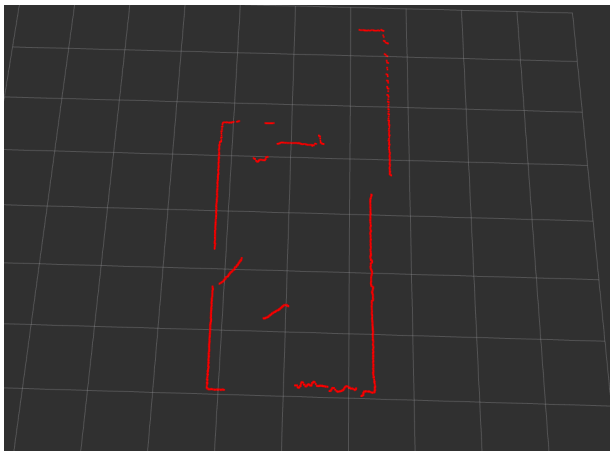


Figure 3.4. A LiDAR scan

The object is detected, but no velocity is given to these objects. This will be done with Detection and Tracking of Moving Objects created by Konstantinidis [10] and is described in the next section.

3.2.2. Detection and Tracking of Moving Objects

It is possible to detect objects in the surrounding with the aid of a LiDAR laser scanner. These objects, however, are only represented with the echo of the laser scan that was sent out. To be able to cluster these echos in objects and to track moving objects, other solutions are required. This is done with Detection And Tracking of Moving Objects (DATMO). DATMO for LiDAR applications can be divided into three main categories. These are the model, the grid and the traditional based approaches [10]. The model based approach starts from scan lines obtained from a LiDAR scanner. These lines are then converted to the appropriate coordinates and segmented. From these segments, corner and line features are extracted and associated with existing shape models [32]. The grid based approach will detect and track objects through the construction of an occupancy grid around the LiDAR. The cells of the grid are then tracked with a Bayesian filter [10]. Finally, the traditional approach will first divide the incoming laser scan measurement into clusters. These clusters will then be associated to previous time instances. The system described further will fit geometric shapes onto these clusters to estimate the dimensions of the tracked object [10].

Detection of moving object

The detection of an object is a multiple stage process that is shown in Figure 3.5. The first stage is the segmentation of the data that is received from the 2D LiDAR. During this stage, the data will be

divided into clusters where each cluster represents an object. The segmentation is followed by a feature extraction, during which rectangles with an estimate of the objects dimensions are fitted on the clustered data sets. During the last stage of the detection process, a L-shape from the closest corner of each rectangle is extracted. These extracted shapes are then used in the tracking process of the system [10].

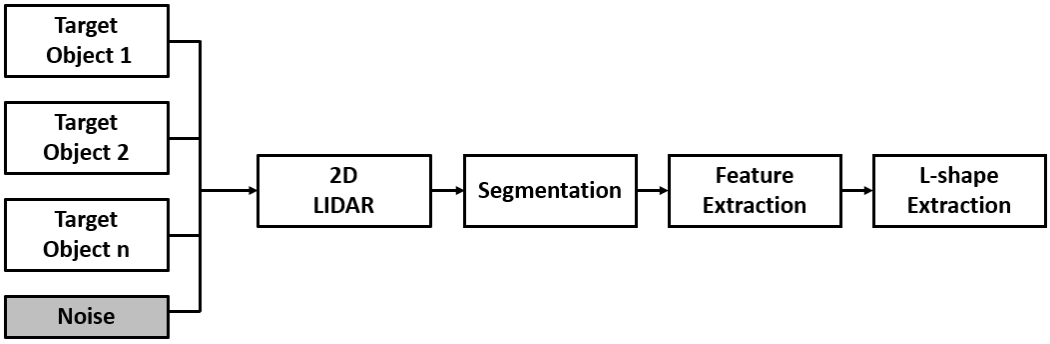


Figure 3.5. Detection process of an object [10]

Segmentation

The first stage is to split the LiDAR measurements into groups of objects. The algorithm used to find those groups is the Adaptive Breakpoint Detector algorithm [10], a visualisation of the algorithm is shown in Figure 3.6. It groups points together based on the Euclidean distance between consecutive measured points.

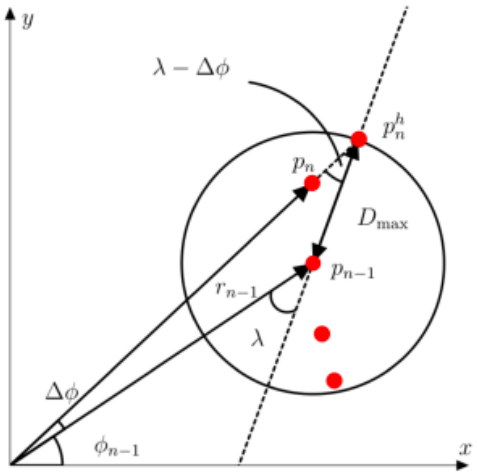


Figure 3.6. Visualisation of the Adaptive Breakpoint Detector Algorithm [10]

$$\|p_n - p_{n-1}\| > D_{max} \tag{3.5}$$

If the distance between point p_n and p_{n-1} is greater than a predefined maximum distance, D_{max}, the points belong to two different objects. D_{max} is, however, not a fixed distance. The maximum distance changes accordingly with the distance from the point to the LiDAR. This is done in the following way:

$$r_{n-1} \cdot sin(\lambda) = r_n^h \cdot sin(\lambda - \Delta\phi) \tag{3.6}$$

By rewriting the equation with trigonometry, the equation for D_{max} is obtained.

$$D_{max} = r_{n-1} \cdot \frac{sin(\Delta\phi)}{sin(\lambda - \Delta\phi)} \tag{3.7}$$

with these equations the measurement data of the LiDAR is grouped into clusters that correspond to objects in reality.

Feature extraction

The next step of the process is the fitting of a geometric shape over the clusters. The chosen geometric shape is a rectangle, which is a common extracted shape for vehicles [33]. This is done with a Search-Based Rectangle Fitting algorithm [10]. The algorithm starts by iterating through all possible directions while finding a rectangle that covers all the points in the cluster for each direction. These rectangles are then compared to each other with a performance score after which the highest scoring one is chosen. In Figure 3.7, the working of the algorithm is shown. The score of each rectangle is calculated with the point-to-edges closeness maximisation criterion, where a higher score is given to rectangles with more points close to the edge of the rectangle [10].

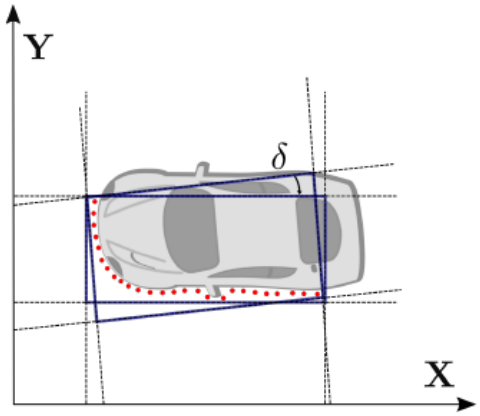


Figure 3.7. Working of the Search-Based Rectangle Fitting algorithm[10]

L-shape extraction

The system tracks the moving object with an L-shape, so this shape still needs to be defined. The last stage extracts the L-shape formed by the closest corner of the rectangle to the sensor. The two lines that form the angle are named L_1 and L_2 in counterclockwise notation. Furthermore, the orientation angle of L_1 is also extracted. The information that is fed to the tracking process of the system is the position of the corner point, the lengths of the lines and the orientation angle [10].

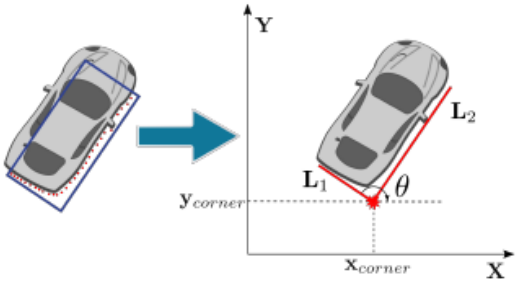


Figure 3.8. L-shape extraction[10]

Tracking of moving object

In this part of the system, the position, speed and dimensions of the vehicle are estimated as accurately as possible with only the current and previous LiDAR measurements. The different stages of the process are shown in Figure 3.9. During the data association stage, the extracted L-shapes from the detection process are associated with objects from the previous measurements. This enables the tracking of the same objects. In the following stage, the detected L-shapes are compared with the previous corresponding L-shape to detect any translation or rotation of the object. If a change is detected, the three trackers are updated [10].

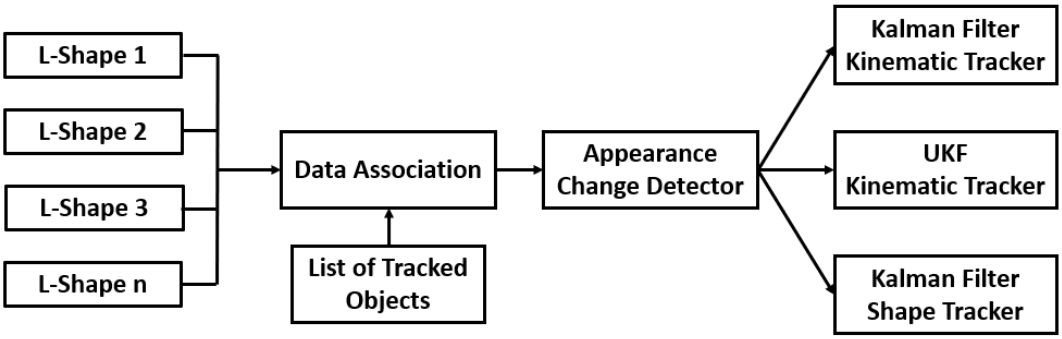


Figure 3.9. Tracking process of an object [10]

Data association

During the first stage, the newly received L-shapes are associated with the objects that were already tracked in the previous timesteps. The association method used here is the Global Nearest Neighbour (GNN) filter. This filter will associate the new L-shapes with previous ones based on Euclidean distance. To avoid filling up the memory with previously used L-shapes, each previous tracked object that is not associated with a new L-shape is removed from the memory, while new L-shapes are added to the memory as new objects [10].

Trackers

There are three trackers of which two are meant for the position and one for the shape of the object. The first two are a Kinematic Kalman filter tracker and a Kinematic Unscented Kalman filter tracker.

The kinematic Kalman filter tracker is not used in this thesis. More information about this tracker can be found in [10].

The Kinematic Unscented Kalman filter tracker has a nonlinear motion model in which the position (x, y), speed (v_x, v_y) and turn rate (ω) are tracked. The motion model used here is the Coordinated Turn Model [10]. The state vector x_{CTM} and the kinematic function f_{CTM} are as follows, with T the sampling time:

$$x_{CTM} = [x \quad y \quad v_x \quad v_y \quad \omega]^T \tag{3.8}$$

$$f_{CTM} = \begin{bmatrix} x + \frac{v_x}{\omega} \cdot sin(\omega T) - \frac{v_y}{\omega} \cdot (1 - cos(\omega T)) \\ y + \frac{v_x}{\omega} \cdot (1 - cos(\omega T)) + \frac{v_y}{\omega} \cdot sin(\omega T) \\ v_x \cdot cos(\omega T) - v_y \cdot sin(\omega T) \\ v_x \cdot sin(\omega T) + v_y \cdot cos(\omega T) \\ \omega \end{bmatrix} \tag{3.9}$$

The measurement vector and matrix are as follows:

$$z_{CTM} = [x_{corner} \quad y_{corner}]^T \tag{3.10}$$

$$H_{CTM} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.11}$$

The final tracker is the shape tracker. This tracker is the most complicated one, because it needs to account for the fact that a L-shape, different from the previous one, of an object is measured due to the movement of the objects and LiDAR. The dimensions of the object are tracked using a Kalman filter and a state vector containing the lengths of the lines (L_1, L_2), the orientation of the first line $L_1(\theta)$ and the turn rate (ω) [10]. The contents of the state vector are visualised in Figure 3.10. The state vector is shown below:

$$x_S = [L_1 \quad L_2 \quad \theta \quad \omega]^T \tag{3.12}$$

To estimate the shape of the vehicle, two assumptions are applied. The first is that the size of the object does not change over time and the second is that the yaw rate of the L-shape does not change rapidly. This then means that a static model can be applied for the first assumption and a constant turn rate model for the second. The process matrix A_S, measurement vector z_s and model H_S are then as follows:

$$A_S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

$$z_s = [L_1 \quad L_2 \quad \theta]^T \tag{3.14}$$

$$H_S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.15}$$

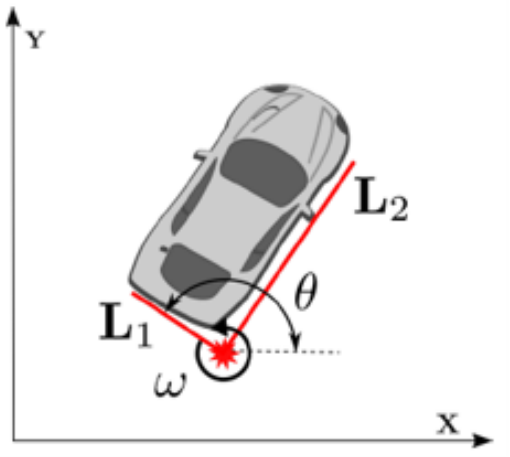


Figure 3.10. Shape filter estimators [10]

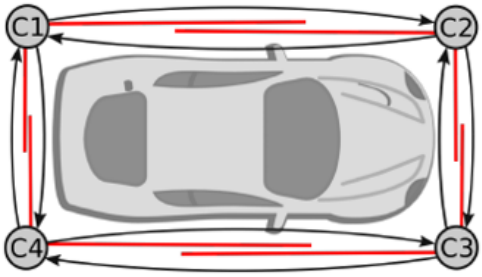


Figure 3.11. The corner points of an object [10]

The second part of the shape tracker is the corner point switching. This part solves the problem of a different extracted L-shape of the same object compared to the previous timestep. For example, at timestep $t-1$ the corner $C3$ was extracted from Figure 3.11, while at timestep t the corner $C4$ was extracted. This would result in the detection of two different objects, while they are, in fact, the same object. These corner changes are detected using the Mahalanobis distances of the new data of the

previously tracked corner point and their two adjacent ones. In the case that $C3$ in Figure 3.11 would be tracked the adjacent points would be $C2$ and $C4$. The state vector of the adjacent points is calculated and compared with the state vector of the new L-shape. The Mahalanobis distance D_M is computed as follows [10]:

$$D_M^2 = (c - m)^T \cdot P^{-1} \cdot (c - m) \tag{3.16}$$

with P the covariance of the tracked corner point and c and m as follows:

$$\begin{aligned} m &= [x_{corner}, y_{corner}, \theta, L_1, L_2] \\ c &= m^T \end{aligned} \tag{3.17}$$

If the shortest Mahalanobis distance is between m and the already tracked corner point, then there was no corner switching. But if the shortest distance is from a neighbouring point, a change of corner is detected and the Kalman filter needs to be adapted. The process also makes a difference in clockwise and counterclockwise switching. In the example with $C3$, a clockwise switch is to $C4$ and a counterclockwise switch is with $C2$. The filter change is computed as follows for the clockwise shifting [10]:

$$\begin{aligned} x_{corner}^{C_j} &= x_{corner}^{C_i} + L_1^{C_i} \cdot cos(\theta^{C_i}) \\ y_{corner}^{C_j} &= y_{corner}^{C_i} + y_1^{C_i} \cdot sin(\theta^{C_i}) \\ v_x^{C_j} &= v_x^{C_i} + L_1^{C_i}\omega \cdot sin(\theta^{C_i}) \\ v_y^{C_j} &= v_y^{C_i} + L_1^{C_i}\omega \cdot cos(\theta^{C_i}) \\ \theta^{C_j} &= \theta^{C_i} - \pi/2 \\ L_1^{C_j} &= L_2^{C_i} \\ L_2^{C_j} &= L_1^{C_i} \\ where &\begin{cases} j = i + 1 i < 4 \\ j = 1 i = 4 \end{cases} \end{aligned} \tag{3.18}$$

The counterclockwise change is calculated as follows:

$$\begin{aligned} x_{corner}^{C_j} &= x_{corner}^{C_i} + L_1^{C_i} \cdot sin(\theta^{C_i}) \\ y_{corner}^{C_j} &= y_{corner}^{C_i} + y_1^{C_i} \cdot cos(\theta^{C_i}) \\ v_x^{C_j} &= v_x^{C_i} + L_1^{C_i}\omega \cdot cos(\theta^{C_i}) \\ v_y^{C_j} &= v_y^{C_i} + L_1^{C_i}\omega \cdot sin(\theta^{C_i}) \\ \theta^{C_j} &= \theta^{C_i} + \pi/2 \\ L_1^{C_j} &= L_2^{C_i} \\ L_2^{C_j} &= L_1^{C_i} \\ where &\begin{cases} j = i - 1 i > 1 \\ j = 1 i = 1 \end{cases} \end{aligned} \tag{3.19}$$

In both equation, C_i and C_j are the corner number before and after the model changes, respectively.

In the final stage, the L-shape needs to be converted to a box model to account for the motion of the whole object. First, the centre of the box is calculated.

$$\begin{aligned} x_{center} &= x_{corner} + (L_1 \cdot cos(\theta) + L_2 \cdot sin(\theta))/2 \\ y_{center} &= y_{corner} + (L_1 \cdot sin(\theta) - L_2 \cdot cos(\theta))/2 \end{aligned} \tag{3.20}$$

Next, the velocity is calculated. The velocity of the corner point is the sum of the velocity of the object and the tangential velocity, where the rotational motion of the corner point includes both translational velocity and rotational velocity. The rotational motion, however, is assumed as a uniform circular motion since a constant turn rate model was already assumed. The tangential velocity can then be computed [10]:

$$\begin{aligned} v_{x,center} &= v_{x,corner} - r\omega \cdot cos[atan(\frac{(L_1 \cdot sin(\theta) - L_2 \cdot cos(\theta))/2}{(L_1 \cdot cos(\theta) + L_2 \cdot sin(\theta))/2}) - \pi/2] \\ v_{y,center} &= v_{y,corner} - r\omega \cdot sin[atan(\frac{(L_1 \cdot sin(\theta) - L_2 \cdot cos(\theta))/2}{(L_1 \cdot cos(\theta) + L_2 \cdot sin(\theta))/2}) - \pi/2] \end{aligned} \tag{3.21}$$

Where r is the distance from the centre to the corner point.

The detected objects are thus tracked and the results are sent out in a ROS message under the name */datmo/marker_array* for the visualisation and under the name */datmo/TrackArray* for further processing. Figure 3.12 shows a visualisation of the process.

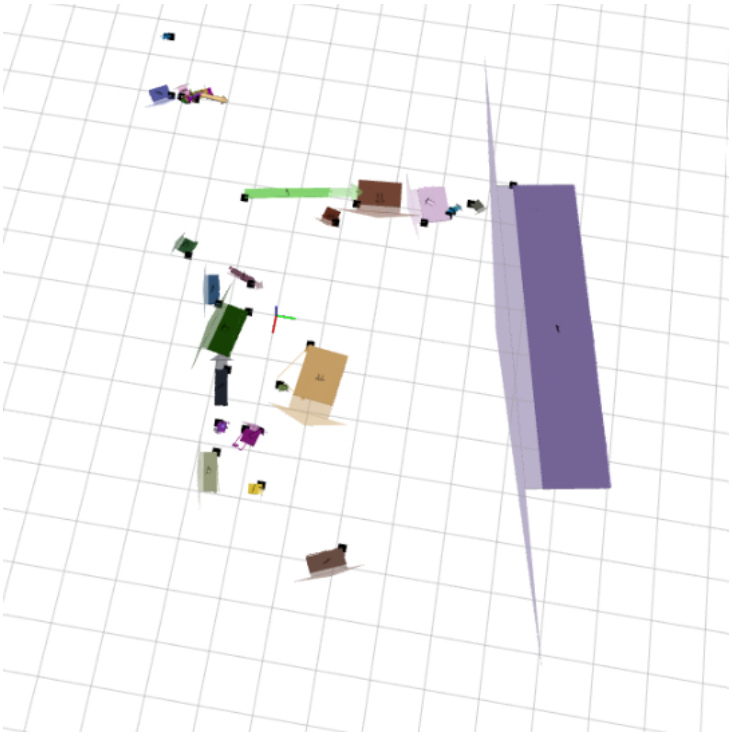


Figure 3.12. Detection and tracking of objects

3.2.3. Prediction of moving object

The object is thus detected, the size is estimated and it is tracked through the different LiDAR measurements, but the future positions of the object are not considered. This is done by tracking the centre of the box and based on its past positions, a short-term prediction is made.

Linear regression prediction

A first method to predict the path of an object with a straight trajectory is with linear regression [34]. With this method, an equation of a line is calculated with the use of the available points with x and y-coordinates. The equation can then be used together with the estimated velocities to predict future positions of the obstacle. The equation is calculated as follows:

$$\hat{y} = \alpha + \beta\hat{x} \tag{3.22}$$

With α and β, estimators that minimises the sum of squared residuals. they are calculated as follows:

$$\begin{aligned} \beta &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\ \alpha &= \bar{y} - (\beta\bar{x}) \end{aligned} \tag{3.23}$$

Where $\bar{x}$ and $\bar{y}$ are means of the x and y-coordinates, respectively.

With the equation of the line, future positions can be calculated. The distance at which these points should lie can be found with the velocity that is calculated through the difference in positions between different measurements.

As will be explained in Chapter 4, this method combined with the tracking algorithm is not a suitable solution to predict the path due to the movement of the centre of the calculated box. The algorithm predicts the size of the object with the received LiDAR laser scan measurements. When the size is estimated, the centre of the box is also estimated. This centre is used in the linear regression model to predict the trajectory. The position of the centre of the box is strongly dependent on what the LiDAR is able to detect. When the obstacle is in front of the USV, only one side of the whole obstacle can be seen and the size of the obstacle will not be estimated correctly. In this case, the centre of the box will almost coincide with the LiDAR measurements and an error in position of up-to half the size of the obstacle is made.

Mean prediction

Another possibility to predict the trajectory, if it is straight, is by using the mean of multiple measurements and this then combined with the predicted velocity of the obstacle. The LiDAR has a measurement rate of 15Hz, which means that the measurements are renewed 15 times each second and the prediction of the trajectory can thus be done on these measurements. The obstacles will not have moved over a great distance when the mean of ten of those measurements is taken. This is with the assumption that the obstacles have a low velocity. In Chapter 4, the velocity of the object is taken at $0.5m/s$ and $1m/s$. The mean of the tracked velocity over these ten measurements is calculated and based on these calculations, a prediction of the future positions is made. The velocity can be split in two parts, which are the velocities along the x and y-axis. This enables to calculate the direction of the trajectory and how quickly the object will move along this trajectory. The position from which these predictions start can be set to the latest calculated box centre position or they can also be a mean of the previous box centre positions. When it is the mean value, there is a certain delay on the prediction and the object will have moved slightly further already, but this is acceptable because the obstacles are assumed to have low velocity profiles. The amount of future position estimations that are made can be tuned and set by the user. The results of this prediction method are discussed in Chapter 4.

3.3. Path and waypoint generation

To navigate from waypoint to waypoint, a path generation method must be defined. The Dubins path is used in this thesis. This path generation method is the shortest path possible to a final position for a USV with a restricted turning radius when a heading and position is given [35]. That is why this method is used for the path generation. Aside from path generation, a method to remain or to navigate towards the path also needs to be defined. This will be a line-of-sight guidance law that is explained in Subsection 3.3.2. Furthermore, when the USV is avoiding obstacles, intermediate waypoints to which it can navigate to avoid the obstacle have to be selected. These obstacles can either be static or dynamic. The obstacle avoidance method will also be described in this section.

3.3.1. Dubins path

Dubins proved that the shortest path consist of three parts which are or arcs of a minimal radius or straight lines [35]. There are four different configurations possible for a Dubins path. These configurations are shown in Figure 3.13, where L stands for left, S stands for straight, and R stands for right.

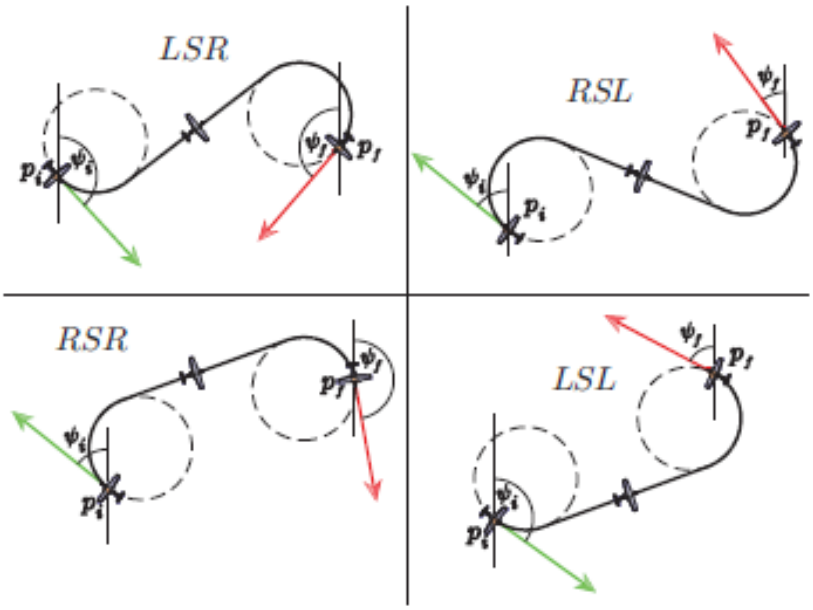


Figure 3.13. Possible configurations for Dubins path [35]

Turning circle

To be able to generate the turning circles, the initial position (p_i) and heading (ψ_i), the final position (p_f) and heading (ψ_f) as well as the minimum turning radius of the USV ρ are needed. The equations for the centre of each circle is then:

$$\begin{aligned} p_{c\ Ri} &= (x_{Ri}, y_{Ri}) = (x_i + \rho \cdot cos(\psi_i), y_i - \rho \cdot sin(\psi_i)) \\ p_{c\ Li} &= (x_{Li}, y_{Li}) = (x_i - \rho \cdot cos(\psi_i), y_i + \rho \cdot sin(\psi_i)) \\ p_{c\ Rf} &= (x_{Rf}, y_{Rf}) = (x_f + \rho \cdot cos(\psi_f), y_f - \rho \cdot sin(\psi_f)) \\ p_{c\ Lf} &= (x_{Lf}, y_{Lf}) = (x_f - \rho \cdot cos(\psi_f), y_f + \rho \cdot sin(\psi_f)) \end{aligned} \tag{3.24}$$

If the location of the final position is inside the circle of minimum turning radius, the Dubins path cannot be constructed. That is why the next waypoint always has to be chosen such that $\frac{|p_i - p_f|}{2} > \rho$.

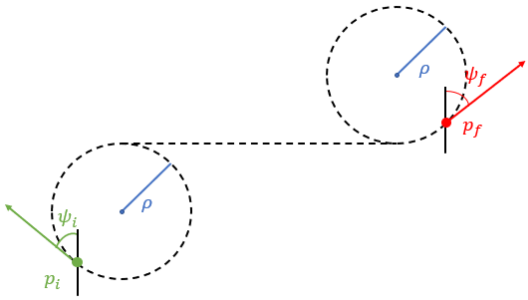


Figure 3.14. Dubins path geometry

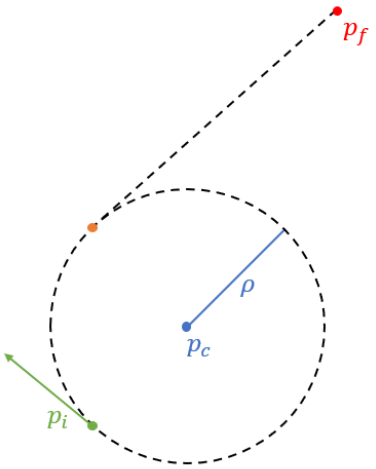


Figure 3.15. Simple Dubins path geometry

For the application of the thesis, only the centre of the initial circle needs to be calculated because the final position can be put on the straight part of the Dubins path. An illustration of such a Dubins path is shown in Figure 3.15.

Turning Direction

The direction in which to turn is calculated as follows:

$$-(x_i - x_f) \cdot sin(\psi_i) + (y_i - y_f) \cdot cos(\psi_f) \tag{3.25}$$

If the result is positive, a left turn will be made. Otherwise, a right turn will be made.

Straight path

The start of the straight trajectory is determined by the tangent line of the final position to the turning circle of the starting position. These two tangent lines can be found with the second-order trigonometric equation [8].

$$((x_c - x_f) \cdot sin(\beta) + (y_f - y_c) \cdot cos(\beta))^2 = \rho^2 \tag{3.26}$$

Here β is the angle of the tangent line. With this equation, two points on the circle are found. The end of the arc will thus be on the first point that the USV crosses, there the straight line commences.

3.3.2. Line-of-sight guidance law

As the path is defined, a method to remain on the generated path or to go towards the path if there is a error needs to be defined. The method used here is based on the work of Lekkas and Fossen [36] and is a curved path line-of-sight guidance law with a time varying look-ahead distance. The method is illustrated in Figure 3.18. The guidance law accepts the generated path as input and gives a velocity and heading as output.

Cross-track and along-track error

The cross-track and along-track errors are both errors that need to be minimised during the path following. The cross-track error is defined as the orthogonal distance between the position of the USV (x, y) and the closest point (x_t, y_t) on the track projected onto a vector perpendicular to the track, while

the along-track error is a measure of the difference between the predicted location and the actual location projected onto the track [37]. Illustrations of the errors are shown in Figures 3.16 and 3.17. The cross-track error y_e is calculated by:

$$y_e = -(x - x_t) \cdot sin(\gamma_p) + (y - y_t) \cdot cos(\gamma_p) \tag{3.27}$$

The along-track error x_e is computed as follows:

$$x_e = (x - x_t) \cdot cos(\gamma_p) + (y - y_t) \cdot sin(\gamma_p) \tag{3.28}$$

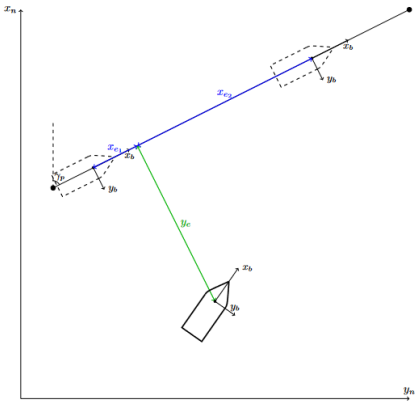


Figure 3.16. Cross-track error [36]

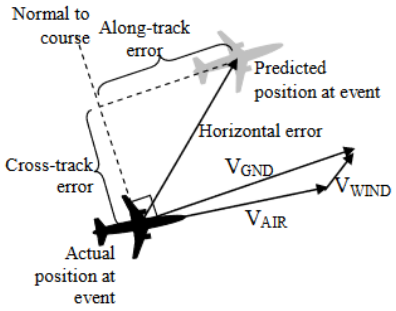


Figure 3.17. Type of errors [37]

The aim is then to minimise these two errors in time. The control objective is thus as follows:

$$\begin{aligned} \lim_{t \to \infty} x_e(t) &= 0 \\ \lim_{t \to \infty} y_e(t) &= 0 \end{aligned} \tag{3.29}$$

Time varying look-ahead distance guidance law

The guidance law is a line-of-sight (LOS) guidance law for maritime applications, which is different from a LOS guidance law in aerial applications. There the line-of-sight starts at the reference point and passes through the objective of guidance, while in a maritime application the LOS vector starts at the USV and goes through a point (x_{los}, y_{los}), which is found on the path-tangential line at a look-ahead distance $\delta(t) > 0$ in front of the projection of the USVs position (x, y) on to the path [38]. This is shown in Figure 3.18. The guidance law is described as follows:

$$\chi_d = \gamma_p + atan(\frac{-y_e}{\Delta}) \tag{3.30}$$

with Δ the look-ahead distance and χ_d the desired course angle of the vehicle.

$$\chi_d = \psi_d + \beta \tag{3.31}$$

In this equation ψ_d is the yaw angle of the USV and β the sideslip angle of the USV. A side-slip angle will occur with a lateral acceleration while turning. It can be seen as the difference in angle of the heading of the USV and the actual direction of movement.

The time-varying look-ahead distance Δ is used to find a balance between a rapid correction to the desired path, but with many oscillations, and a slow but smooth transition to the path. The following formulation is used for the look-ahead distance [38]:

$$\Delta(y_e) = (\Delta_{max} - \Delta_{min})e^{-K_\Delta y_e^2} + \Delta_{min} \tag{3.32}$$

Δ_{min} and Δ_{max} are the minimal and maximal values,respectively, for the look-ahead distance that are permitted and $K_\Delta > 0$ is the convergence rate. The philosophy behind the equation is that a small Δ will be used when the cross-track error is large. This then allows for a rapid return to the path. A large Δ will be computed when the cross-track error is small. This large look-ahead distance results in a smooth transition to the path with smaller overshoots. This guidance law thus allows some tuning by the user.

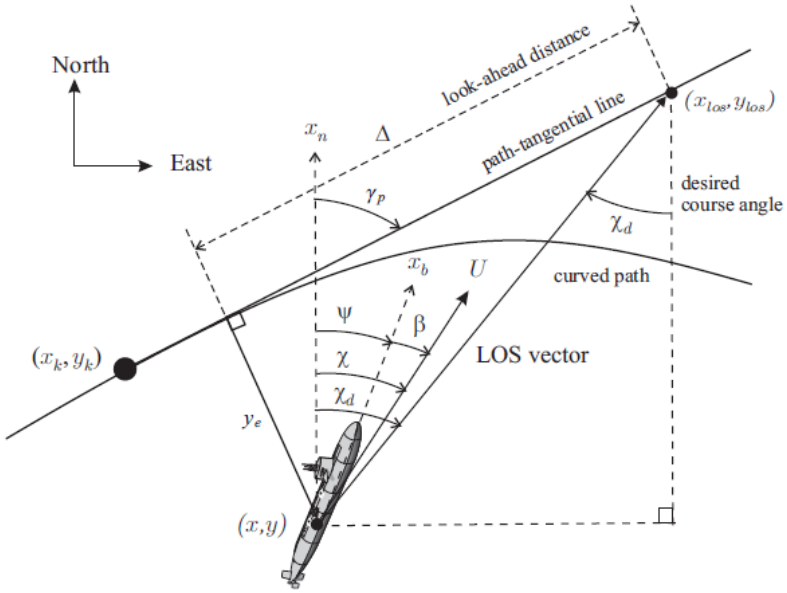


Figure 3.18. The guidance law[38]

Velocity determination

Now that the guidance law and path generation method are defined, a method to assign a velocity in each situation needs to be defined. This is done as follows [38]:

$$v_d = max(v_max \cdot (1 - \frac{|y_e|}{y_{max}} - \frac{|\chi_e}{\chi_{max}}) + v_{min}, v_{min}) \tag{3.33}$$

where v_d, v_{max}, v_{min} are the desired, maximum and minimum velocities, respectively, y_e is the cross-track error, $\chi_e = \chi - \chi_d$ is the error in the course angle. χ_{max} and y_{max} can be tuned to assign a maximal cross-track error and course angle.

The adaptation of the velocity is important because the turning radius of the USV is related to its velocity, when the velocity is lowered the turning radius will become smaller and the USV will be able to more rapidly regain the desired path. This reduction is done up to a minimal velocity at which the turning radius will also be minimal. Additionally, the velocity will be increased when the cross-track error is small and the path is thus followed.

3.3.3. Obstacle avoidance

When an obstacle is encountered, the USV plans its path to avoid the obstacle. This can be done by selecting intermediate waypoints to where it will navigate as long as the direct path towards the waypoint is blocked, after which it heads straight back towards the final waypoint or by avoiding the obstacle, after which it will regain the track on which it was sailing before the obstacle was detected. The encountering of an obstacle is a likely scenario that can happen multiple times while navigating towards a waypoint because of the limited field of view (FOV) of the sensors.

Avoidance with intermediate waypoints

The first method discussed is the method where the obstacle is avoided after which the USV will continue to sail straight towards the final waypoint.

The environment around the USV is partitioned in free and blocked cells as described in Section 3.1. The information of these cells is used to detect an obstacle on the planned trajectory of the USV and to select intermediate waypoints.

As the position of the USV (p_{USV}) and the coordinates of the waypoint (p_f) are known, the angle (θ) of the line through both coordinates with the x-axis is calculated. This is done as follows:

$$
\begin{aligned}
&if x_f \neq x_{USV} \\
&\theta = atan(\frac{y_f - y_{USV}}{x_f - y_{USV}}) \quad for\ y_f > y_{USV} \\
&\theta = atan(\frac{y_f - y_{USV}}{x_f - y_{USV}}) + \pi \quad for\ y_f < y_{USV} \\
&if x_f = x_{USV} \\
&\theta = \pi \quad for\ y_f > y_{USV} \\
&\theta = \frac{3}{2}\pi \quad for\ y_f < y_{USV}
\end{aligned}
\tag{3.34}
$$

With these equations, the angle towards the waypoint is found and with some trigonometry and a user-specified distance the next cells can be checked if they are blocked or free. The user-specified distance is then the distance to which the USV checks on the map in front of it that those cells are free. The USV goes through a while loop just as the condition that the cells along a heading and at the determined distance are free.

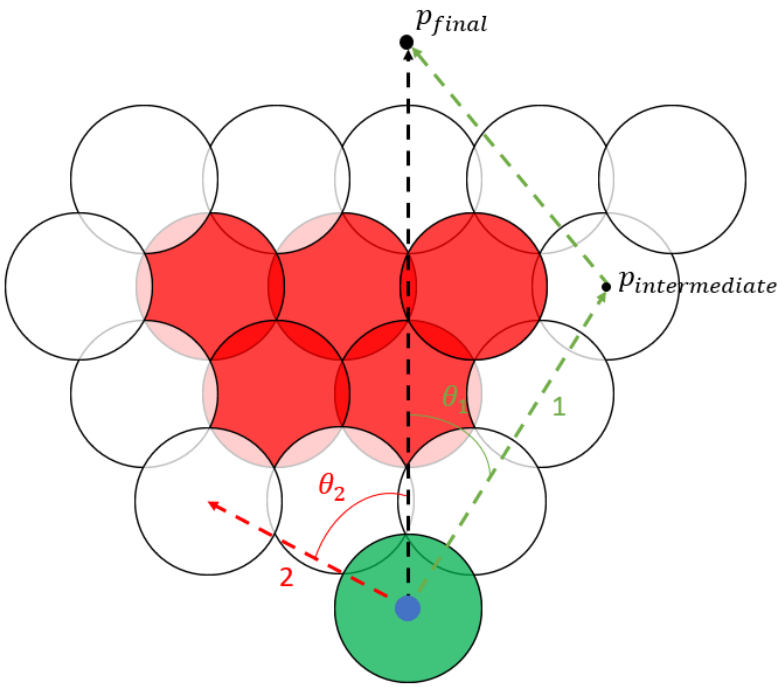


Figure 3.19. Generating of path to avoid an obstacle

If a blocked cell is encountered, which means an object is detected and in front of the USV, the USV must manoeuvre to avoid the obstacle. This is shown in Figure 3.19. The USV can manoeuvre to the left or the right. The direction to manoeuvre to is decided based on the size of the angle between the current heading of the USV and the heading needed to avoid the obstacle. These right and left angles (respectively θ_1 and θ_2) are calculated iteratively with a while loop. As long as the sum of the current heading and angle go through a blocked cell the loop repeats with a new and higher angle. The loop is interrupted when the cells are free. The right and left angles are then compared to each other where the smallest change in angle to obtain the shortest manoeuvre is taken. In the example shown in Figure 3.19, θ_1 is smaller and this angle will thus be chosen to calculate the new heading and path to avoid the obstacle.

While avoiding the obstacle, the USV will continually check if the straight path towards the final waypoint is free of blocked cells. If that is the case, the USV will sail towards the final waypoint using a Dubins path as described in Subsection 3.3.1

Avoidance with track regeneration

The other possibility to avoid an obstacle and navigate towards the final waypoint is by avoiding the obstacle in the same manner as here above, but instead of directly going to the final waypoint the original track is regained and continued. This is shown in Figure 3.20.

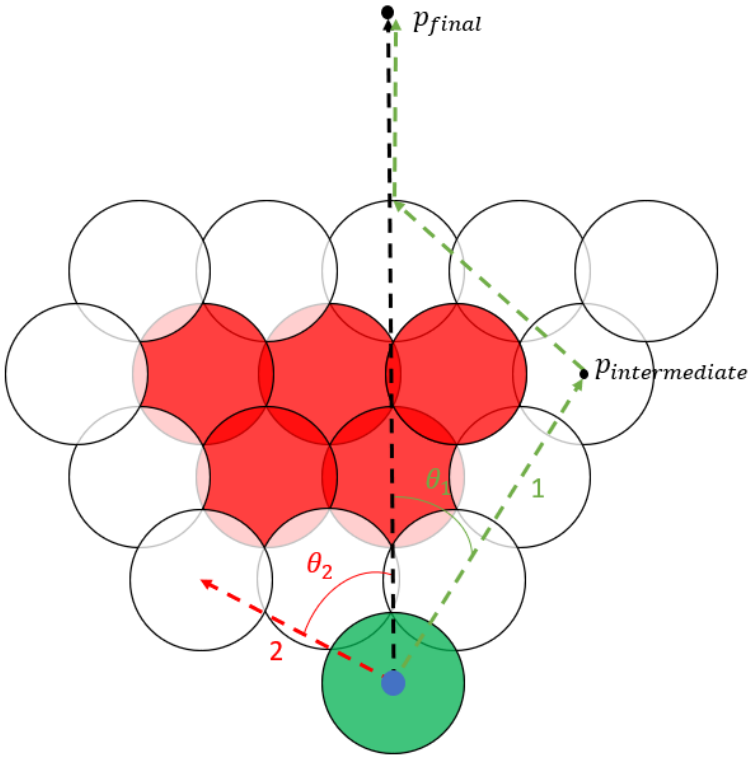


Figure 3.20. Obstacle avoidance with track regeneration

The equation of a straight line through two points gives the possibility to calculate each point on the line. Here the starting and final point are known and hence also the equation of the trajectory that should be followed. When the obstacle is encountered, it is avoided in the same manner as above, but as soon as it is clear to return to this trajectory, this is done. This method is also used on board of ships when the navigation is done based on visual marks.

Dynamic obstacle avoidance

When the USV must avoid moving obstacles, there is an additional difficulty for the obstacle avoidance. The vessel must predict where the obstacle will be in the short-term future. This is done with the dynamic obstacle detection and tracking together with the prediction method that is explained in Section 3.2.

The obstacle avoidance uses the same principle as discussed in the first case of the static avoidance, but the USV will start to manoeuvre earlier. When the USV starts manoeuvring is decided by the predicted future positions of the dynamic obstacle, which is explained in Subsection 3.2.3. These positions are calculated and the cells in which they are positioned are already blocked before the obstacle is in these cells. This means that the USV will already avoid these cells as if they were occupied by a static obstacle. The range of the prediction is dependent of the tracked velocity of the obstacle and to where the obstacle is moving relative to the USV. If the obstacle is moving towards the USV, while the USV is moving towards the obstacle. The velocity of the USV itself must be taken into account because the distance between the obstacle and the USV will shorten more rapidly. This is taken into account by calculating the angle between the connecting line of the box centre position and the USVs position and the heading of the USV. The basic range of the position prediction is set by the user and is done in Chapter 4. An illustration of the dynamic obstacle avoidance is shown in Figure 3.21. In Figure 3.21, the cells that are blocked with an obstacle are coloured red, the cells that need to be avoided due to the predicted trajectory are coloured orange while the USV is positioned in the green cell. The USV sees the orange cells in front of itself and starts avoiding by checking the free cells left and right of it. The path with the lowest heading change is then chosen, just like in the static avoidance case.

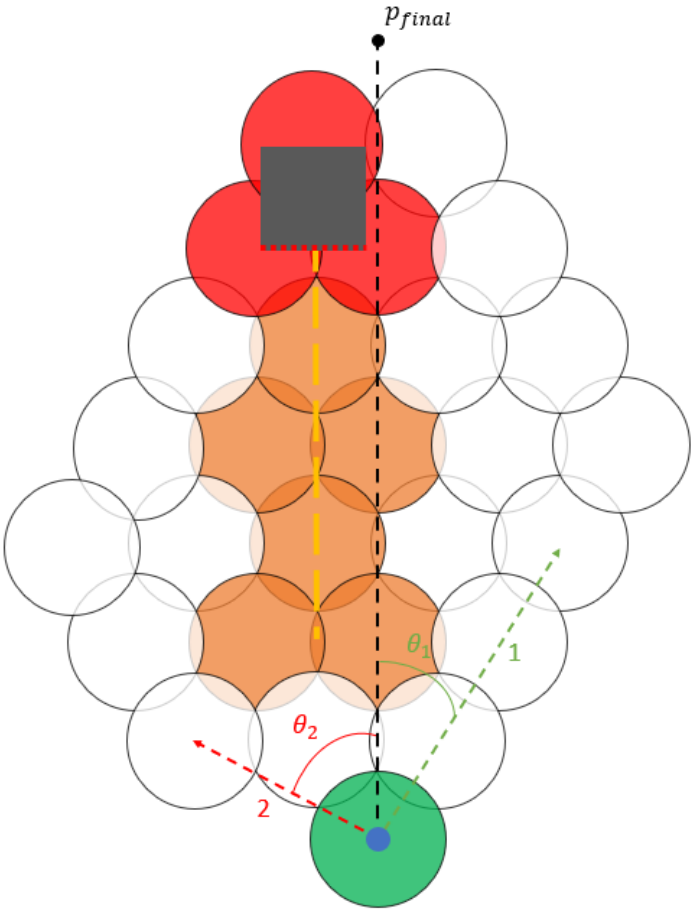


Figure 3.21. Dynamic obstacle avoidance

4. Simulation

In this chapter, the simulation of the navigation method is discussed. In Section 4.1, different simulators that are available are explored and discussed. The simulator that is chosen is discussed in Section 4.2. This discussion sets out the capabilities and parameters of the simulator. The implementation of Dubins path and the path generation is discussed in Section 4.3. In Section 4.4, different tests are performed with static and dynamic obstacles. For the dynamic obstacles, the future trajectory of the obstacle needs to be predicted. A manner to do this and its results are also discussed in this section.

4.1. Simulator

This chapter discusses a USV simulator that is required to develop and test the navigation method in a simulated environment before it is implemented on a real vessel. There are several simulators available that deal with the topic of USVs and autonomous navigation for different goals. These simulators often use a version of Robot Operating System (ROS) and Gazebo to visualise the simulation. ROS is an open source library of software and tools that can be used to build robot applications [39]. In the following subsections different simulators are explored, before one is chosen in which the method is implemented and tested.

4.1.1. Virtual RobotX

Virtual RobotX (VRX) from Bingham et al. [40] is a USV simulation environment in which USVs can be simulated. It is an extension of the Gazebo robot simulator and specifically targets USVs. The simulation environment has the added features of waves that interact with the objects, the representation of a water surface, the addition of wind, a USV with an adaptable propulsion system, the addition of buoyancy forces on objects and a LiDAR simulation that can interact with the water representation if it is set in 3D mode. These new features transform the Gazebo robot simulator in a simulator for USVs. The simulator was used as a basis for the Virtual RobotX competition of 2022 in which participants needed to implement capabilities for the USV and let the USV complete certain tasks. A submission for this competition is discussed below. RobotX is a program which organises a competition every year around the development of unmanned and autonomous systems across all the domains [41]. This includes unmanned aerial vehicles (UAV) as well as USVs. The USV that is created in the simulator is a catamaran-based design USV. Such a type of USV is good to minimise roll because it has two separate points of contact with the water surface. The USV that is used in the simulation is based on the WAM-V designed by Fish et al. [42]. The design has the possibility of two types of propulsion. The first type is with two fixed thrusters that allows USV turning with the implementation of a skid-steer drive. Skid-steer driving is explained in Chapter 5. The second type is with vectored thrusting, where the thrusters can turn independently of the USV around their axis. This allows a sideways thrust. When vector thrusting is used, the two thrusters can still have a differential thrust. This simulator produces a USV and a well-constructed environment, where a lot of different scenarios and functions can be implemented and tested but it does not deliver any capabilities for the USV, these need to be implemented. This is why it is often a basis for more advanced simulators. The USV and a simulated environment is shown in Figure 4.1

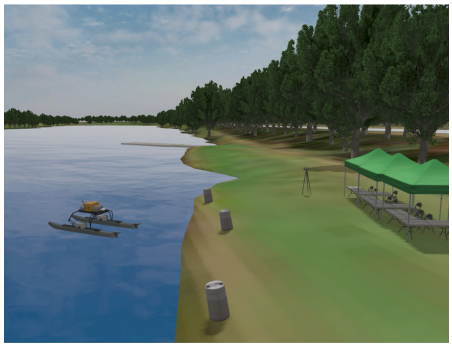


Figure 4.1. The Virtual RobotX simulator with a simulation of the WAM-V USV [40]

Figure 4.2. The WAM-V USV, the real USV used in the simulator [42]

Singaboat-VRX

The Singaboat-VRX was built by Samak et al. [43] for the Virtual RobotX (VRX) competition in the Virtual RobotX simulator. In the competition, the USV needed to be capable of performing six different tasks autonomously. These tasks were

- Station-keeping: In this task, the USV is spawned in a random location inside the map and it must sail towards a defined position. When it has reached this position, it needs to remain static there.
- Wayfinding: Here, the USV must navigate towards waypoints or achieve heading values that are published to the USV.
- Scene perception: The USV must be able to detect and recognise certain objects, like buoys, that are spawned in its field of view. This can be done with cameras, LiDARs or other sensors.
- Semantic navigation: Here, The USV navigates towards a waypoint while avoiding certain objects and performing tasks at others. These tasks include making a full circle around an object in clock or counterclockwise rotation.
- Localisation of an acoustic beacon, navigation through a channel and obstacle avoidance: In this task, the vessel must find an acoustic beacon after it has passed through a channel. The acoustic beacon is located in an area that is filled with static obstacles that must be avoided.
- Dock and deliver: The USV must be able to select a dock where it can dock. This is followed by the docking manoeuvre and subsequently the exiting of the dock.

The Singaboat team has implemented these tasks in varying degrees of completeness, but scored overall good in the competition. They have thus implemented capabilities in the Virtual RobotX simulator. An example of their environment can be seen in Figure 4.3. This simulator uses ROS2 and Gazebo 11 to run.

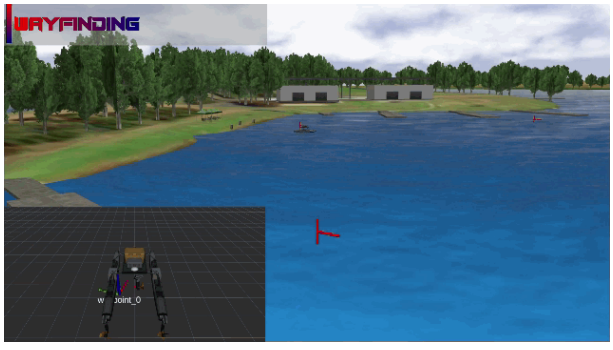


Figure 4.3. The Singaboat simulation [43]

4.1.2. Kingfisher/Heron USV

Another simulator is the Kingfisher/Heron USV simulator that supplies a simulator for the Heron USV from Clearpath Robotics [44]. The Heron USV is a portable and modular USV that can be handled by one person, making it a popular solution for the testing of navigation methods and mapping applications. The simulator is developed by Clearpath Robotics [45] The USV can be controlled in the simulation with keyboard inputs, but has no implemented navigation method. The simulator supports the use of an IMU and GPS to position itself in the generated world frame. This simulator is thus a very basic one with little implemented capabilities. The simulator can be run using ROS Indigo and Gazebo 7.

4.1.3. Otter USV

The Otter USV is a simulator developed by Lenes [46] and is based on the Virtual RobotX from Bingham et al. [40]. This is an implementation of the Otter USV in a simulated environment. The Otter USV is a lightweight drone made by Maritime Robotics and is meant for seabed mapping and the monitoring of sheltered waters [21]. Lenes has implemented the Otter USV in ROS to simulate autonomous path planning and path-following to scan the seabed in a port. This simulator is for static environments, just as the Singaboat simulator. He has implemented the detection of obstacles, but no avoidance to reach a waypoint because his implementation was for the complete mapping of the seabed of a port. The aim was to pass over every place in the port and not to navigate towards positions. The Otter USV that is implemented uses a fixed thruster design, just as the USV used in this thesis. This means that the thrusters are commanded in the same manner to navigate. The otter USV is developed to be used on ROS Melodic and Gazebo 9.

4.2. Used Simulator

The simulator used as a basis in the thesis is the Otter USV simulator and it is run in ROS Noetic and Gazebo 11. The USV in the simulator is one with two fixed thrusters that are operated using a skid-steer drive just like the discussed simulators above. The USV is equipped with a LiDAR, IMU, compass and GNSS receiver to locate itself in the world and to detect its surroundings. In the simulated world, a lake or the sea can be simulated. There is a possibility to enable waves that then interact with the USV. Furthermore, wind can be enabled such that the USV drifts from its path. In the following sections, the results of the implementation are discussed. These results are gathered on a flat sea surface, there were no waves and it is a windless environment.

4.2.1. Parameters

In the simulation, the USV has some parameters that need to be set. These parameters allow tuning of the USV for different purposes. The maximal velocity of the USV is set at $1.5m/s$, while its minimal velocity is $0.4m/s$. If the USV makes a turn, the velocity is further limited to $1m/s$ to allow full control over the turning. To calculate and execute the Dubins path, a circle with a minimal radius must be defined. This minimal radius is set at one metre. Furthermore, the LiDAR also has some parameters. The maximal range of the LiDAR is set at $25m$. This range is less than the maximal range of $40m$ of the RPLIDAR S1, which is described in Chapter 5, that is used on the USV platform. However, the $40m$ range is when the detected obstacle is a perfect white obstacle. As this is almost never the case, the actual range is lower than $40m$, hence $25m$ is chosen for the simulation. The pulse repetition frequency of the LiDAR is set at $15Hz$, which corresponds to the repetition frequency of the RPLIDAR. The range resolution and angular resolution are set the same as for the RPLIDAR, these are $3cm$ and $0.391°$, respectively. The only parameter for the LiDAR that differs from the actual LiDAR is the sample rate. In the simulation, the sample rate is set at $2kHZ$ instead of $9.2kHz$. This is to limit the calculation time in the simulation.

4.2.2. Definition coordinate system

The coordinate system used in the simulation is a Cartesian system where the z-axis is always pointed upwards. There is one absolute coordinate system to which the position of all objects and the USV is reported to. Each object has its own relative system that turns together with the object. This can be used to report the heading with respect to the absolute system. The relative axis system of the USV is shown in Figure 4.6. The red line is the x-axis, while the green one is the y-axis. The z-axis is pointed upwards, but because the objects are only detected and the USV only moves in the 2D XY-plane this one is of lesser concern. These coordinate systems remain the same for the rest of the thesis.

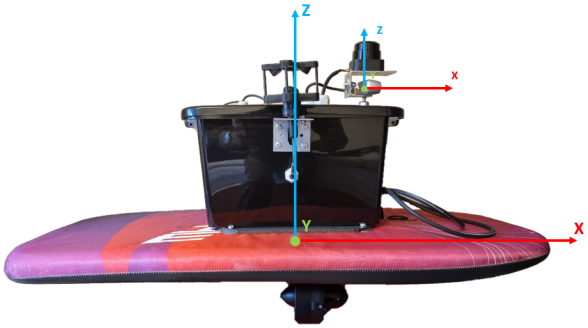


Figure 4.4. The axis definition in the simulation

4.2.3. Obstacle detection and enlargement

In Chapter 3, the detection and subsequent inflation of the obstacles to create a safety zone were discussed. The enlargement and detection of the obstacle is shown in Figure 4.5. It is implemented using the costmap_2D package in ROS [31]. This package receives data from the sensor to which it is subscribed, which is in this case the LiDAR sensor, and constructs an occupancy grid with that received data. The occupancy grid is built up of small cells with one of three attributes, which are *occupied*, *free* or *unknown*. Furthermore, the package also enlarges or inflates the cells that are occupied. The enlargement is done with a user-specified inflation radius. In the case of the figure, the inflation radius is set to $2m$. This is because the largest dimension of the Otter USV, which is used in the simulation, is $2m$. In this way there is always at least the maximal length of one USV between the USV and an obstacle. In the figure, the white cells are free, the grey are unknown and the black are occupied cells, with LiDAR scanning data in red. A cell is unknown when it is not in range of the LiDAR or when the cells are blocked by an obstacle, as can be seen in the figure.

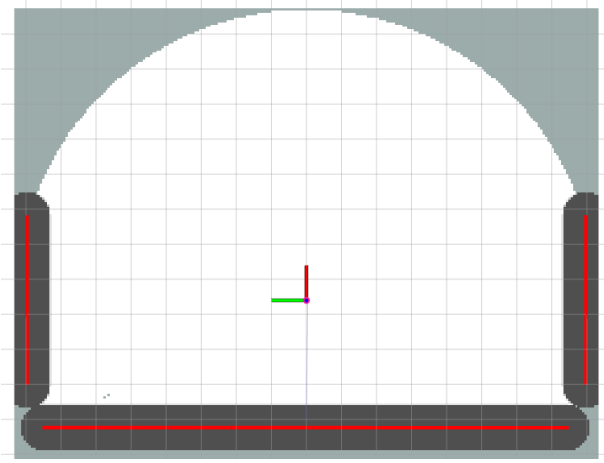


Figure 4.5. The detected and inflated obstacle

For the navigation and obstacle avoidance of the USV, another type of occupation cells other than the costmap cells is used. These cells are shown in Figure 4.6. The cells are round with a user-specified radius, in the figure the radius is set to $2m$. Just as with the cells of the costmap, these cells have four states that can be assigned, namely *free*, *unknown*, *blocked* or *predicted*. A cell is *blocked* whenever occupied cells from the 2D costmap fall within its surface. They are set to *predicted* when the predicted trajectory of the dynamic obstacle falls in the cell, as explained in Chapter 3. They are *unknown* when there is no information available of the 2D costmap cells inside the larger cells, hence when the cells are situated out of range of the LiDAR or are blocked by an obstacle in front of them. They are marked *free* when there are no detected obstacles inside the cell.

4.3. Implementation of Dubins path and path generation

The generation of a path towards a waypoint using a Dubins path, as explained in Section 3.3, is tested in the simulation. The USV navigates towards a waypoint that is situated to its right. The USV starts at position $x = 0, y = 0$ and the waypoint is located at $x = 40, y = -15$. The generated path is shown in Figure 4.6.

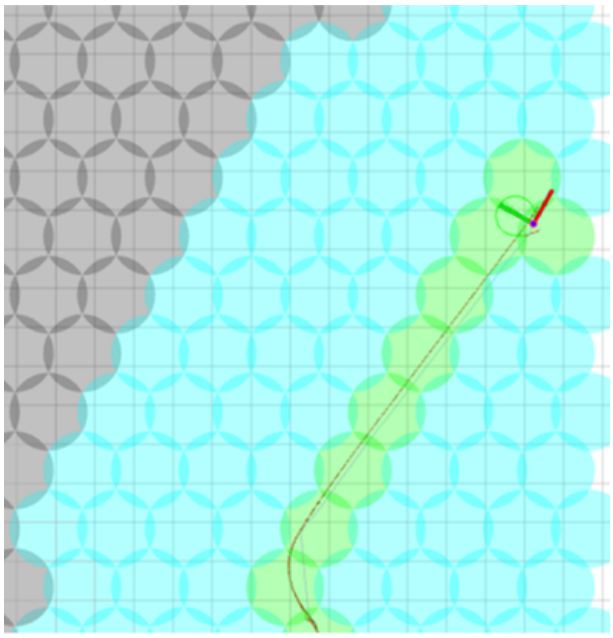


Figure 4.6. Dubins path and path generation

As can be seen in Figure 4.6, the USV starts with following a circle to adapt the heading to the heading that corresponds to the waypoint. After the correct heading is obtained, it starts the straight line on which it travels until the waypoint is reached. This test shows that the generation of the Dubins path works to get the USV to a waypoint.

The cells through which the USV travelled are marked in a green colour, the unblocked cells are blue, the grey cells are cells where there is no information of, while the red cells are cells that are or covered by an obstacle or included in the safety zone around an obstacle. The path that is covered by the USV is shown by a red line, whereas the heading and point to where the USV needs to move are marked by a green line. This green line can be seen in Figure 4.9.

4.4. Obstacle avoidance

In this section, the developed navigation method is tested with obstacles. Firstly, it is tested with static obstacles in Subsection 4.4.1 and then with dynamic obstacles in Subsection 4.4.2.

4.4.1. Static avoidance

The performed tests, for the static avoidance, include the avoidance of some isolated obstacles and the navigation through a narrow corridor. In the case of navigating through a narrow corridor, the USV has obstacles from three sides and it must navigate towards a waypoint at the end of the corridor. When the obstacles are static, it is not needed to track them, hence the detection and tracking of dynamic obstacles is not yet used. Before the results of the tests are explained, the way the USV sees the obstacles is explained. Secondly, the path towards the waypoint that is set in the tests is performed without any obstacles.

Static obstacles

An obstacle must thus be spawned in the simulator. This is done inside Gazebo by defining an object with a certain pose and dimensions. To keep the simulated objects simple, all the objects are cubes or cuboids. This shape also has the advantage that this works the best with the detection and tracking algorithm that is used. An example of an obstacle is shown in Figure 4.7. The spawned obstacles have dimensions of $3m$ in width, $3m$ in length and $1m$ in height. They are positioned at $x = 15, y = 0$ and $x = 15, y = 10$. The laser scan from the LiDAR for the two obstacles is shown in Figure 4.8. The right obstacle is directly in front of the USV, which leads to only the visibility of the front side. The other obstacle is seen from the side and a corner of the obstacle is thus seen from the LiDAR scan.

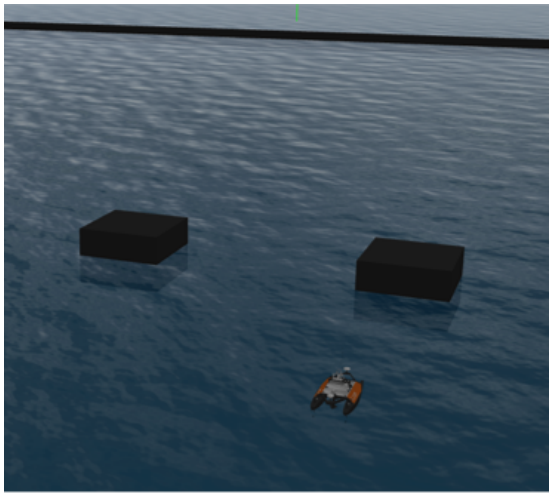


Figure 4.7. Two static obstacles

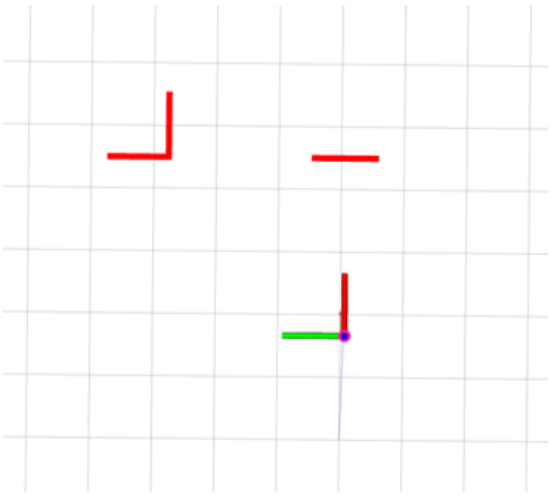


Figure 4.8. Laser scan of the 2 obstacles

Navigation without obstacles

The USV is spawned at position $x = 0, y = 0$ and needs to navigate towards $x = 80, y = 40$. In Figure 4.9, the trajectory towards the waypoint is shown when there are no obstacles. As can be seen, the USV uses the Dubins path to navigate towards the waypoint. It starts with an arc segment to have the correct heading before it begins with its straight line. As explained in Chapter 3, the waypoint is at the end of the straight trajectory. When there are no obstacles, the USV will travel along a straight line towards the waypoint after the heading is obtained.

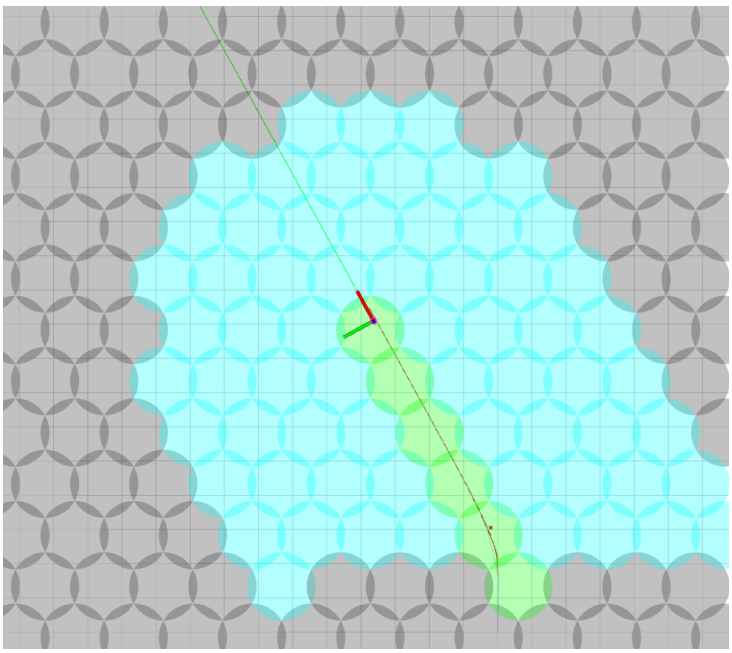


Figure 4.9. Navigation without any obstacles

This scenario was also tested with a calm sea state and thus with small waves. The USV could still attain its waypoint but drifted off from its starting position in the beginning. This same reaction was seen when a light wind breeze was added.

Single obstacle

The Obstacle avoidance is first tested with an obstacle that is on the path towards the waypoint. The setup is shown in Figure 4.10. The obstacle, with dimensions $5m$ in width, $5m$ in length and $1m$ in height is placed in the middle of the path from the starting point of the USV and its final waypoint. The USV must manoeuvre around the object to attain the waypoint.

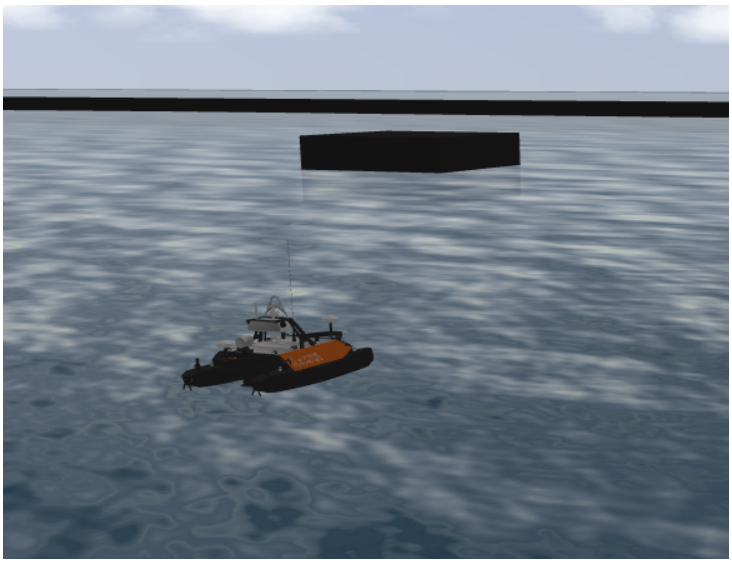


Figure 4.10. Setup single obstacle

In Figures 4.11 until 4.16, the navigating towards the waypoint while avoiding the static obstacle is shown.

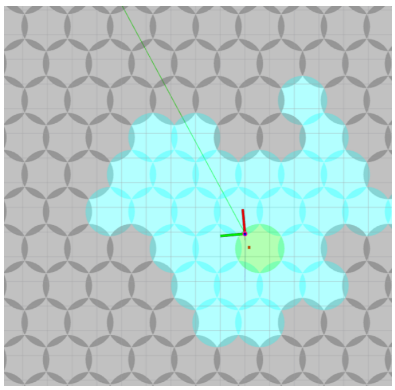


Figure 4.11. Begin navigating

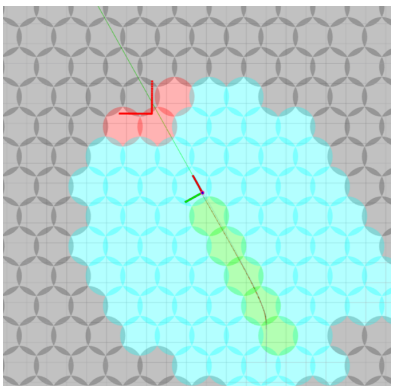


Figure 4.12. Obstacle detected

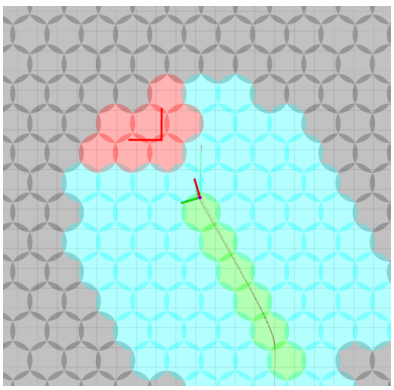


Figure 4.13. Start of avoidance

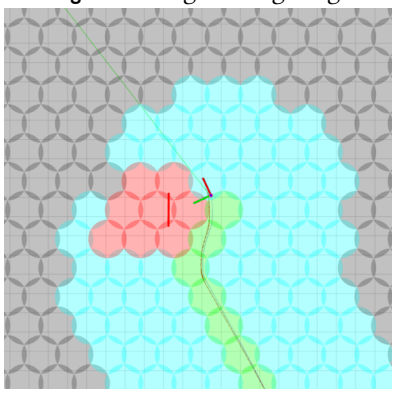


Figure 4.14. Continue of avoidance

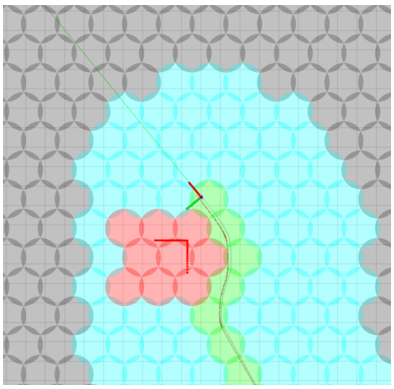


Figure 4.15. Obstacle avoided

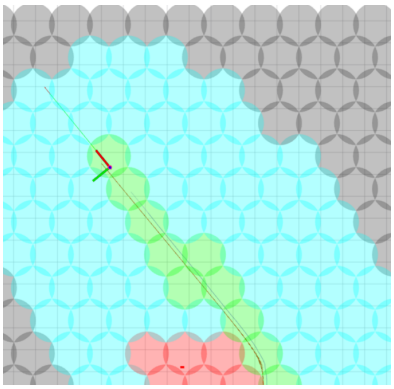


Figure 4.16. Waypoint reached

In Figure 4.11, the USV is in its starting position and starts its path towards the waypoint. With a Dubins path, the USV is put in the correct heading and it starts navigating towards the waypoint. In Figure 4.12, the obstacle is detected on the path of the USV. The detected part of the obstacle is shown as the red corner in the figure. This corner is what the LiDAR is able to pickup from the obstacle with its laser scans. The red cells around the obstacle are a safety zone in which it is preferred that the USV does not enter. The vessel is still travelling along the straight path because the set distance to start avoiding is still too large. The start of the avoidance is shown in Figure 4.13. Here, the USV calculates which direction is best to avoid the obstacle. The heading change to turn to the right is smaller than the heading change needed to turn left, so the USV will turn to the right to avoid the obstacle. In Figure 4.14, the vessel continues its avoidance because the straight line towards the waypoint is not yet clear. The cells that are free with the smallest heading change are chosen to continue avoiding the obstacle. In this case, it are the cells in front of the USV. The obstacle is cleared in Figure 4.15 and the USV continues its way straight towards the waypoint. In the final figure, Figure 4.16, the USV is at the waypoint and the obstacle is thus successfully avoided.

Obstacle path

The behaviour of the USV is further tested in a generated map shown in Figure 4.17. This map consists of a corridor in which the USV needs to navigate from its starting position $x = 0, y = 0$ towards $x = 80, y = 40$. The navigation without any obstacles is the same as for the simple obstacles. To sail towards the waypoint, the USV is not able to generate a straight path, hence it needs to move in the direction of the x-axis. Then it will need to take a turn and sail along the direction of the y-axis, before taking another turn that will lead towards the waypoint.

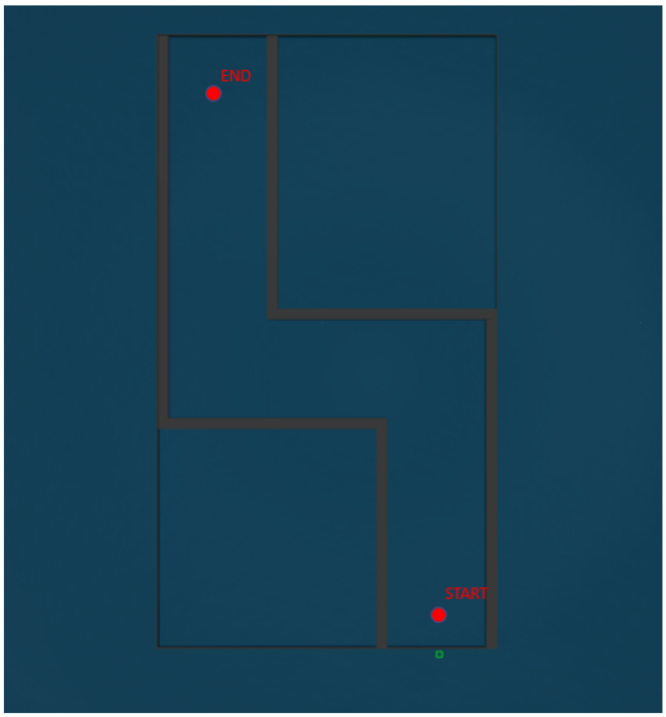


Figure 4.17. The generated obstacle path. The red dots are the starting and end point of the USV

In Figures 4.18 until 4.23, the navigation inside the corridor is shown at different moments.

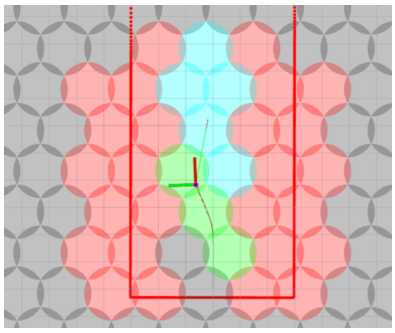


Figure 4.18. Begin corridor

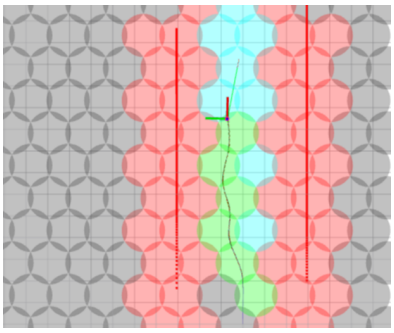


Figure 4.19. Halfway first corridor

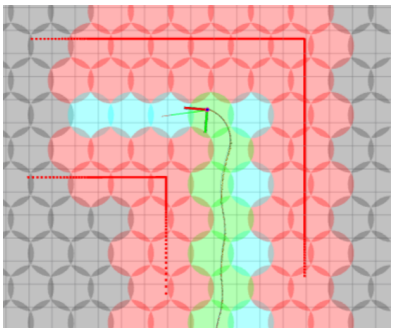


Figure 4.20. The first turn

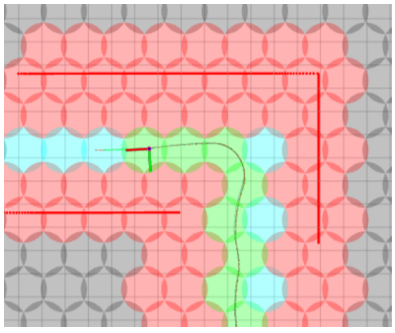


Figure 4.21. After the first turn

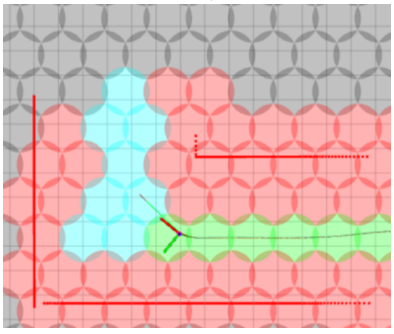


Figure 4.22. Second turn

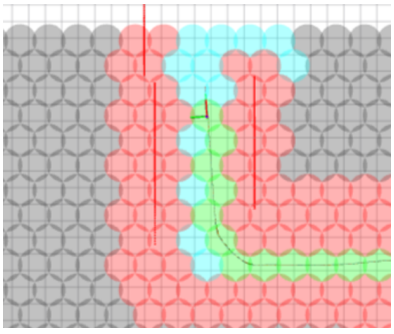


Figure 4.23. At waypoint

In this situation, the USV is continually surrounded by obstacles, which differs from the previous test. The setup of this test can be seen in Figure 4.17. In Figure 4.24, the map of the whole simulation environment is seen after the USV has reached its waypoint. The amount of free cells to navigate in is

very limited due to the corridor and the safety distance set up for the USV. In Figure 4.18, the USV is spawned in the position $x = 0, y = 0$ and is, as can be seen, directly surrounded by obstacles. In the beginning, the track towards the waypoint is determined, as can be seen by the green line in the figure. However, this track is not suitable and the USV must find its way. In Figure 4.19, the vessel has found a direction that is available to travel along. This is also the only direction that is possible. The first turn to the left must be made in Figure 4.20. As already explained, the USV will determine which direction to turn. As turning to the right is not possible because of the obstacles present, the USV must turn to the left towards the only possible path towards the waypoint. The situation after the turn is shown in Figure 4.21. Here, it can be seen that the USV has successfully made its first turn while keeping a safe distance from the obstacles. In Figure 4.22, the second and final turn to reach the waypoint is made. Here, the USV turns to the right and because the path towards the waypoint is clear immediately after the turn the USV navigates directly to the waypoint as shown in Figure 4.23.

The complete trajectory that the USV travelled is shown in Figure 4.24. Here, it is clearly seen that the trajectory, seen in red in the green cells, of the USV always stays clear of the obstacles and its safety zone. The avoidance of the obstacles has thus been successful.

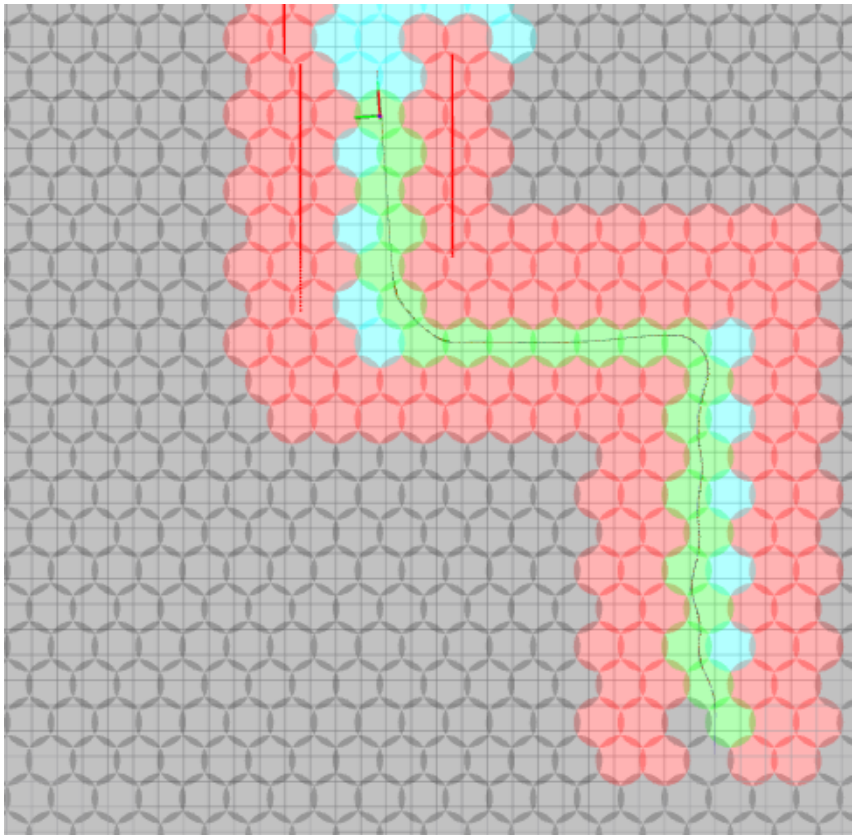


Figure 4.24. The full track that the USV travelled to atain its waypoint. The path of the USV is shown by the red line.

4.4.2. Dynamic avoidance

The developed navigation method has been proven to work with static obstacles in Subsection 4.4.1. In this section, the method is tested against dynamic obstacles that move at different velocities. The performed tests are an obstacle that moves towards the USV on a straight trajectory, hence a head-on obstacle, and an obstacle that performs a straight trajectory perpendicular to the USV, hence a sideways obstacle. These tests correspond to the most seen situations at sea. Most vessel-to-vessel encounters can be subdivided into three categories, namely overtaking, head-on and crossing situations, as shown in Figure 4.25 Overtaking can be considered as a special case of the head-on situation, as the object is also in front of the USV but it moves slower than the USV, additionally it follows the same direction of travel as the USV. Three testing scenarios will thus be performed. These scenarios are a head-on, overtaking and sideways crossing situation. However, first, the dynamic obstacles and the implementation of detection and tracking of dynamic obstacles are explained.

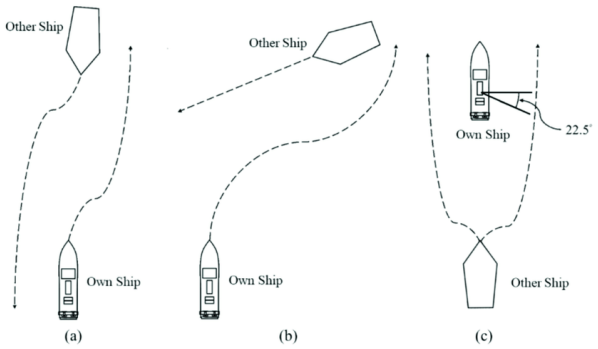


Figure 4.25. The different vessel-to-vessel interactions: (a) head-on, (b) crossing, (c) overtaking [47]

Dynamic obstacles

To test the dynamic avoidance, a moving obstacle needs to be implemented in Gazebo. This is achieved by using a dynamic model plugin [48] that is added to the statically generated obstacle from Subsection 4.4.1. The plugin enables the box to follow a predefined trajectory. The trajectory is defined as a list of positions where the object must pass through. Furthermore, the roll, pitch and yaw of the object at each position and, lastly, the velocity of the object at each point are also defined. The trajectory can be looped such that it is continually repeated. In Figure 4.26, two obstacles are placed at 10 and 20 metres from the USV, respectively. They both follow a straight trajectory that spans from $y = -10$ to $y = 10$ and this at a distance of $x = 10$ and $x = 20$ from the USV, respectively. Their velocity is $1.5m/s$. In Figure 4.27, the detected obstacles are tracked, as can be seen by the blue box that estimates the size of one of the obstacles and the arrows that estimate the velocity of the obstacles.

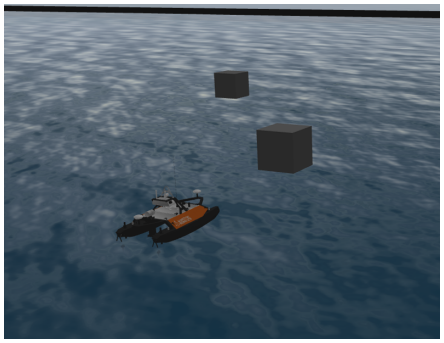


Figure 4.26. Two dynamic obstacles

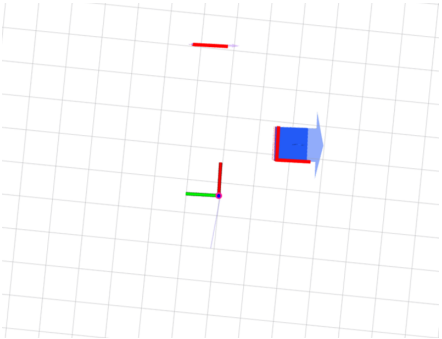


Figure 4.27. The detected and tracked dynamic obstacles

It is possible to generate obstacles that move with all kinds of trajectories. In the following subsections, the avoidance of a dynamic obstacle is developed.

Trajectory prediction

Here, detection, tracking and prediction of the trajectory of an obstacle is discussed. This prediction is made with the data that is gathered by the DATMO package, as discussed in Chapter 3. The two trajectory predictions that are discussed are linear regression and prediction based on the mean of previous positions. The positioning will be done with both the box centre and box corner, such that the accuracy of the DATMO package can be assessed.

Different trajectory prediction methods based on the estimated box centre position were developed in Chapter 3. One of those was the linear regression prediction. With this method, a linear equation was established based on the previous positions of the obstacle. However, this is not a suitable solution to predict the movement of the obstacle. This is due to the fact that it is based on individual positions of the estimated centre of the obstacle. In this prediction, a significant amount of error is possible, as shown in Figure 4.27. In the figure, the farthest obstacle from the USV, at a distance of 20 metres, is estimated to be around $3m$ in length, but only around $0.1m$ wide. In reality, the obstacle is a cube of $3m$. This gives thus rise to a positioning error of the centre of the obstacle of as much as half the width of the obstacle. The error will be the largest when the obstacle is right in front of the USV, because the other obstacle that is seen from its side in Figure 4.27 is estimated much more accurately. In Figures 4.28 and 4.29, a part of the estimated trajectory of the centre of an obstacle that moves on a straight line at $x = 20$ from $y = -10$ to 10 is shown against the ground truth of the real centre of the obstacle. The USV was in the position $x = 5$ and $y = 0$ looking straight at the trajectory of the obstacle for these measurements.

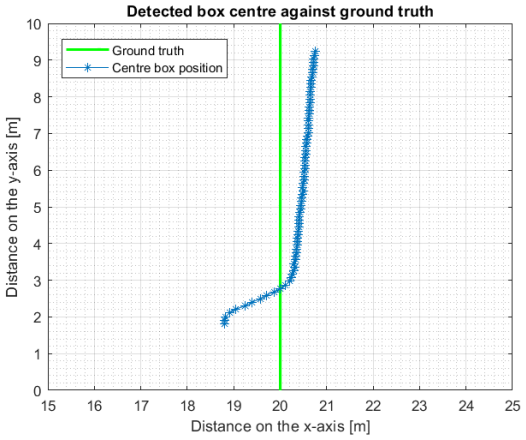


Figure 4.28. Estimation of the box centre position of the obstacle against the ground truth zoomed

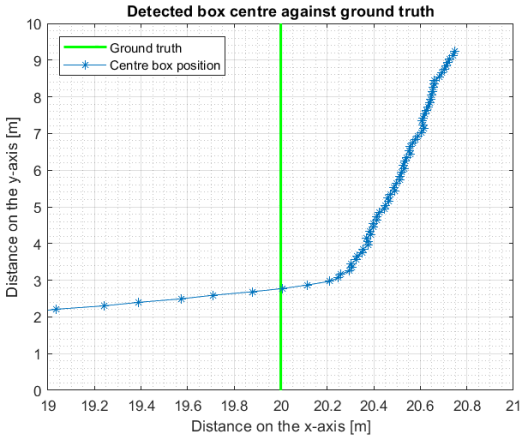


Figure 4.29. Estimation of the box centre position of the obstacle against the ground truth zoomed

As can be seen in Figure 4.29, the estimated centre position never coincides with the real position of the centre. This can be explained with the way a LiDAR obtains its measurements. This is illustrated in Figure 4.30. The LiDAR sends out a laser pulse that is reflected back on a surface. When a part of the box cannot be reached by a sent-out pulse due to the fact that it is not visible, the obstacle will be estimated to be smaller. This explains the wrongly determined box centres. The position of the centre has the largest error when the obstacle is in front of the USV. The error then diminishes to grow larger and to overestimate the position of the centre. When the linear regression is performed on these centres, it will render an equation with a large error. The estimated trajectory differs a lot from the ground truth and even has a slope. The linear regression model based on the centre of the obstacle is thus not a suitable solution to predict the trajectory of an obstacle.

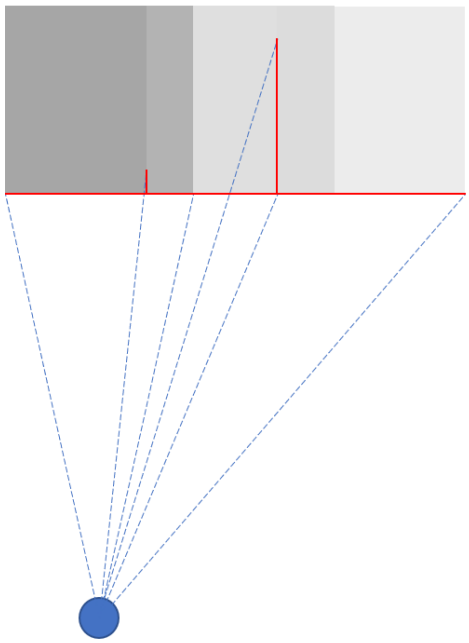


Figure 4.30. LiDAR measurements on a moving obstacle

The centre of the obstacle is calculated by the algorithm based on the detected corner and this can explain the large error on the position of the centre at the low and high values of y. At the low values the obstacle is directly or almost directly in front of the USV and no corner is detected. While at high distances the USV will only see the side of the obstacle and not the part that is normally facing the USV. When this is taken into account and the position of the corner is calculated instead of the centre, the error is less as shown in Figures 4.31 and 4.32. In reality, the corner lies at a position of $x = 18.5$. At low y-values, the error is not negligible, whereas at mid to higher values the position error in x is around $10cm$. This error is acceptable because there is a safety zone around an obstacle and the obstacle is still being detected with the LiDAR.

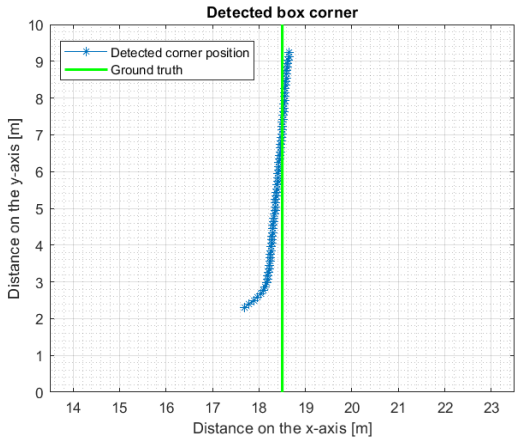


Figure 4.31. Detected distance of the corner of the obstacle

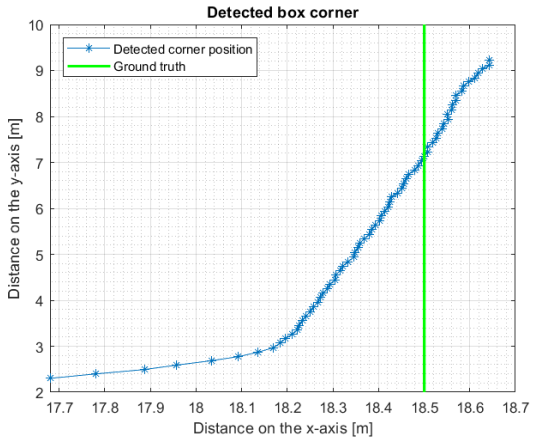


Figure 4.32. Detected distance of the corner of the obstacle zoomed

In Figure 4.33, the errors in the estimated position of the box centre and box corner are compared with each other. As can be seen in the figure, the estimations of the centre and corner point positions have a similar evolution, but the error on the box centre will be bigger than on the box corner. This can be explained by the fact that the box corner is detected more accurately as soon as two sides are visible. The two sides do not need to be visible over their whole length, but the laser scan measurements only need to make a corner. To estimate the box centre accurately in position, however, two sides need to

be visible over their whole length for the LiDAR. This is almost never the case, which causes a greater error in positioning. To quantify the difference between both curves, the mean difference of the errors on the positions is calculated. The mean difference between both measurements is equal to $0.306m$ and the box centre makes the biggest error in the estimation of the position. The box corner is thus the best to obtain measurements as accurately as possible.

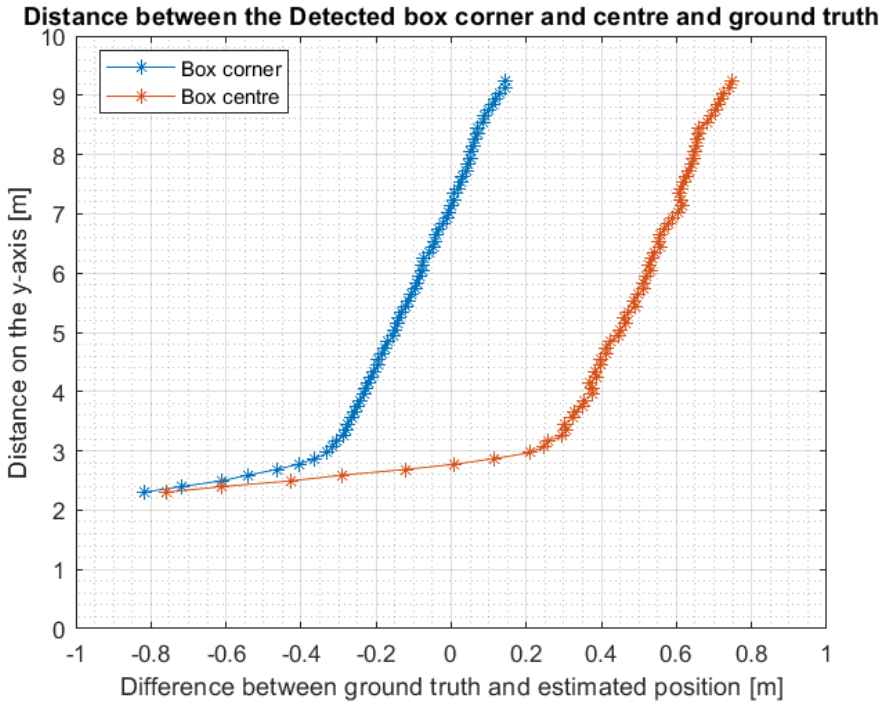


Figure 4.33. The difference of the estimation of the box centre or box corner position and the ground truth in x

The linear regression is not an appropriate solution for the prediction of the trajectory of the obstacle because there is a significant margin for error and the predicted trajectory does not correspond to the real trajectory.

As already stated the obstacle that is used moves with a velocity of $1.5m/s$ along a straight line parallel to the y-axis at $x = 20$. In Figure 4.34, the estimated velocity of the object is shown. The velocity is estimated by measuring the distance that the estimated box, based on the detected corner, travelled between two LiDAR measurements. As can be seen in the figure, the velocity along the x-axis is low and has a mean of $0.0869m/s$, while the velocity along the y-axis has a mean of $1.2681m/s$. The mean of the total velocity is $1.2711m/s$. This means that there is an error between the velocity that is estimated and the real velocity. This can be explained by the fact that the velocity is estimated based on information from LiDAR measurements. The velocity is estimated with the detected corner and its displacement between measurements. The error on the velocity is an average $8.69cm/s$ in the x-direction and $23.19cm/s$ in the y-direction. Overall, the estimation of the velocity and its direction are sufficient for its use in this thesis. When the future positions of the box are predicted with these velocity measurements there will thus be an error on the position, but because the prediction is continually renewed with new velocity estimations the prediction of the positions will also be slightly changed. For the obstacle avoidance, a zone to where the obstacle moves must be known and not its exact positions in the future. This zone is sufficient because there is a safety zone around the obstacle which also is to be avoided and the obstacle is still being detected with the LiDAR measurements as was used in the static avoidance. These LiDAR measurements can then be used to adjust the avoidance when in vicinity of the obstacle.

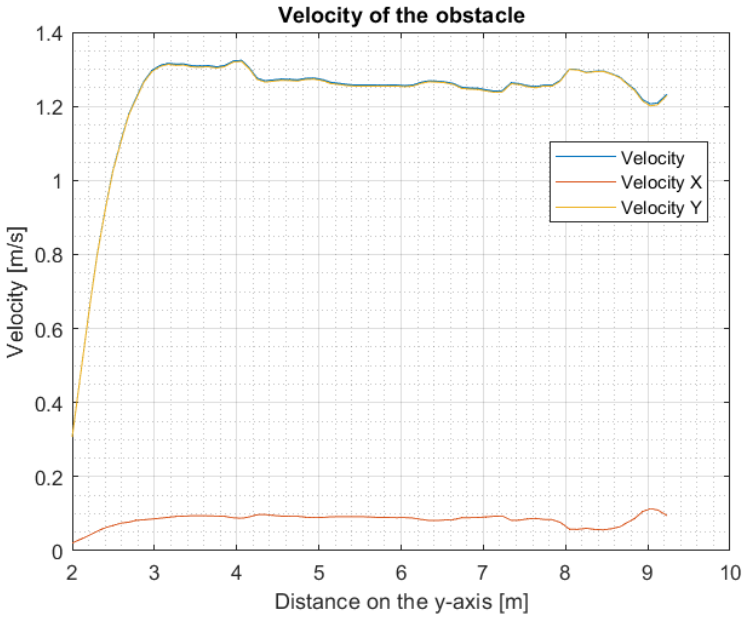


Figure 4.34. Estimated velocity of the obstacle

The trajectory of the obstacle can thus be predicted with the aid of the estimated velocity. A second method to estimate the future position of the obstacle is based on a mean of different positions and a mean of the calculated velocities.

Figure 4.35. The prediction when two sides are detected

Figure 4.36. The prediction when there is only one side detected

As can be seen in Figures 4.35 and 4.36, the heading of the obstacle is estimated correctly as well as the future positions. Each dot represents the position of the obstacle's centre in the future in intervals of one second. The future positions are calculated based on the mean of 10 previous positions and on the mean velocity between the 10 previous points. The LiDAR has a repetition frequency of $15Hz$, so the mean of the positions and velocities is a viable solution because the object has not moved a lot. There is, however, a difference when two sides of the obstacle are detected compared to when one side is detected. This difference in position estimates is acceptable because a safety zone around the obstacle is constructed, thus even with a small error in the y-direction in this case the prediction is sufficient to evade the obstacle. The obstacle is also still being detected with the LiDAR and the real-time laser scan data of the obstacle is also used for the avoidance, just as in the static environment.

Avoidance with a head-on obstacle

The first scenario is one in which the USV navigates from $x = 0, y = 0$ towards $x = 60, y = 5$. While the vessel travels towards the waypoint, it encounters an obstacle that is travelling along a trajectory of $x = 60, y = 0$ towards $x = 0, y = 0$, hence in the opposite direction of the USV. This obstacle moves with a velocity of $1m/s$, which is two thirds of the maximal velocity of the USV. The avoidance is shown in Figures 4.37 to 4.41.

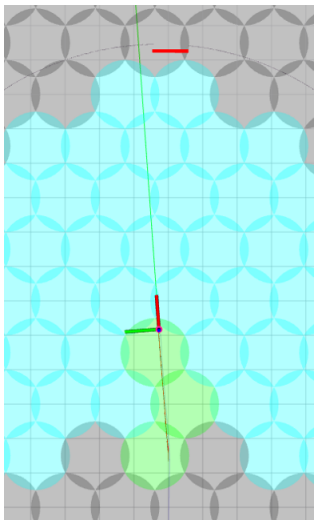


Figure 4.37. Obstacle detected

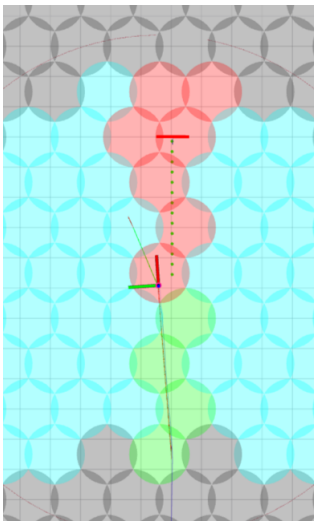


Figure 4.38. Obstacle tracked

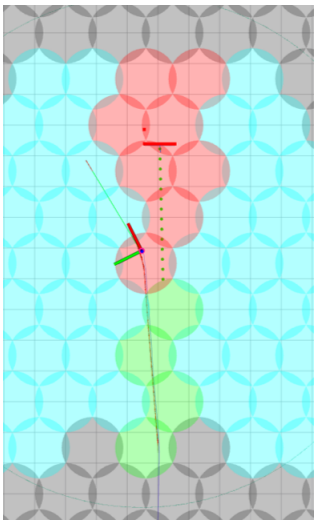


Figure 4.39. Continue avoiding

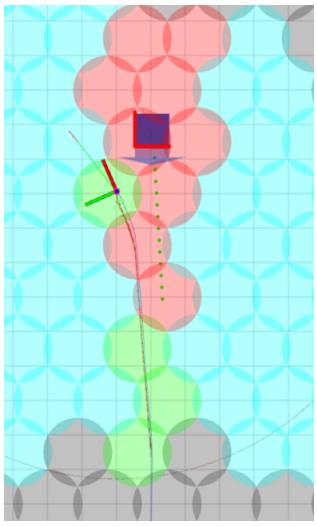


Figure 4.40. Obstacle at the same x as the USV

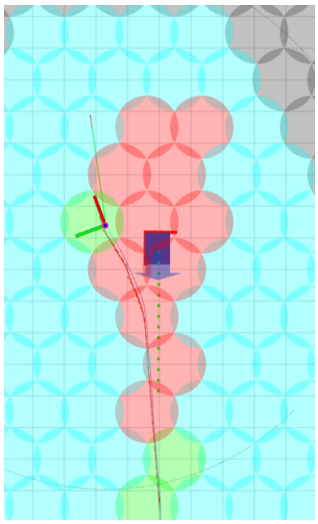


Figure 4.41. The obstacle is avoided

In Figure 4.37, the head-on obstacle is detected at the limit of the LiDAR range. The obstacle is not yet tracked, because there is not enough data available. Hence, the USV still keeps navigating towards its final waypoint. Figure 4.38 shows that the obstacle is now tracked and its future path is predicted. The future positions are shown with the green dots in the figure. The cells in which these dots are located are coloured red and their status is changed to blocked. In this way, the USV knows that it needs to start avoiding these cells. The final waypoint is also changed to an intermediate one to be able to avoid the obstacle. The path to this waypoint is shown as the green line with a red arrow that signifies the heading at the end. The obstacle is tracked, but its dimensions are not estimated correctly. This is because the USV cannot know how long the obstacle is based on the received measurement. This can be done as soon as the USV is able to see at least two sides of the obstacle, as depicted in Figure 4.40. The tracking of the obstacle, however, is still good because the velocity and position of the obstacle

can still be estimated with only the side that is seen. The estimated position is then the position of the front side of the obstacle. This position is just as useful as the centre position because the USV must avoid the whole obstacle and not just the centre position. In Figure 4.39, the USV plans its path to avoid the obstacle, this must be sufficiently in advance because the maximal velocity of the USV is chosen at $1.5m/s$. The planned path is two cells in advance in the figure. In Figure 4.40, The USV is almost at the same level as the obstacle, but must continue its local path to leave sufficient space between the USV and the obstacle. In the last figure, Figure 4.41, the USV has made enough space for the obstacle and the obstacle has passed. When the obstacle has passed, the red cells are set back to free, which are then coloured blue and the USV can recalculate a direct path towards its final waypoint.

Avoidance with an overtaking obstacle

The second scenario is also with a head-on obstacle, but the obstacle travels in the same direction as the USV. The obstacle moves at a velocity of $0.5m/s$, which is three times slower than the maximum velocity of the USV. The USV will thus overtake the obstacle. Just as in the previous scenario, the USV sails from $x = 0$, $y = 0$ to $x = 60$,$y = 5$, while the obstacle moves in a straight line from $x = 5$,$y = 0$ to $x = 60$,$y = 0$. The obstacle avoidance of the USV is shown in Figures 4.42 to 4.46.

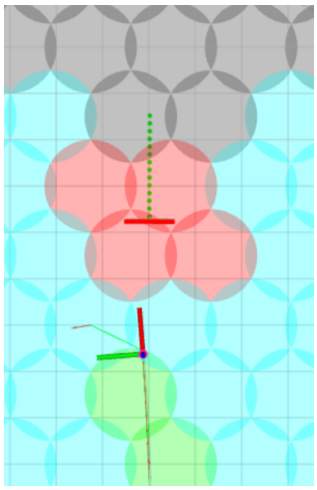


Figure 4.42. Obstacle detected and tracked

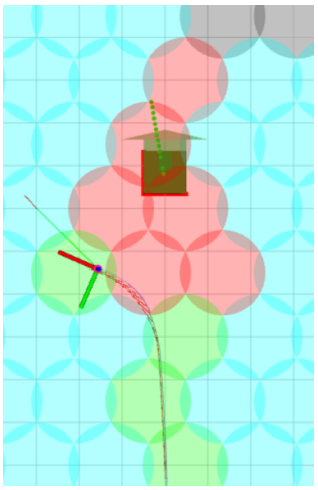


Figure 4.43. Avoidance to overtake the obstacle

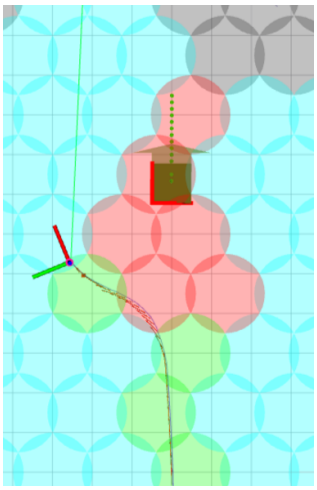


Figure 4.44. Avoidance complete, start overtaking

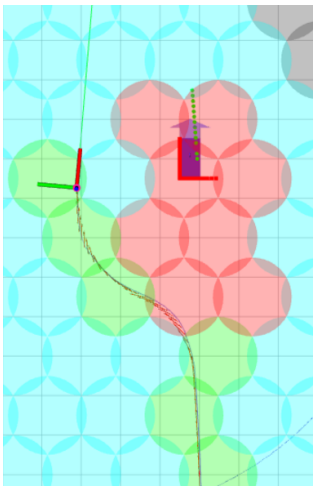


Figure 4.45. Obstacle at the same x as the USV

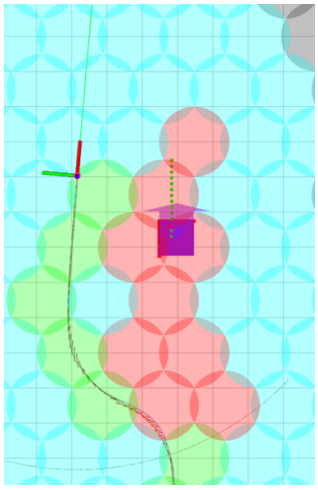


Figure 4.46. The obstacle is overtaken

In Figure 4.42, the obstacle is already detected and tracked by the USV, as can be seen from the green dots that represent the future trajectory, the red line that represents the LiDAR measurements, and the red cells that signify that they are blocked. The USV starts to avoid and overtake the obstacle by turning left as can be seen by the small green line. In Figure 4.43, the USV has already done a large part of the avoidance as there is already room between the USV and the obstacle to pass, but it still continues avoiding because the cells toward the waypoint are not yet all free. This can be seen by the red cells. The USV continues navigating toward the first free cell, as shown by the green line. In Figure 4.44, the USV has sufficiently moved out of the way to sail directly back towards the waypoint while still overtaking the obstacle. The USV is at the same level as the obstacle in Figure 4.45, hence, it can clearly be seen that the USV travels faster than the obstacle. Finally, in Figure 4.46, the USV has overtaken the obstacle and it can navigate towards the waypoint. The tracked obstacle changes colour between Figure 4.44 and Figure 4.45, which means that the USV lost its track of the obstacle for a short time and could not associate the extracted L-shapes from the different measurements with each other. However, this tracking loss did not interfere with the ability to safely overtake the obstacle.

Avoidance with a sideways obstacle

Here, the USV navigates from $x = 0, y = 0$ to $x = 80, y = 5$. While the USV navigates towards the waypoint a dynamic obstacle with a velocity of $1m/s$ that follows a straight trajectory from $x = 20, y = -10$ to $x = 20, y = 10$ is encountered. This trajectory is perpendicular to the path that the USV has planned. Therefore, the vessel must avoid an obstacle coming from its side. The avoidance of a sideways obstacle coming from the right, as seen by the USV is shown in Figures 4.47 to 4.52.

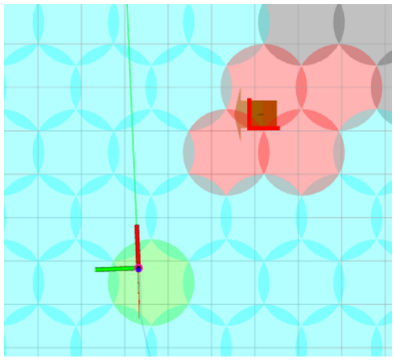


Figure 4.47. Navigating towards the waypoint

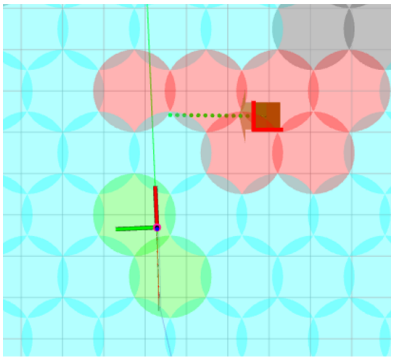


Figure 4.48. The tracked obstacle and predicted trajectory

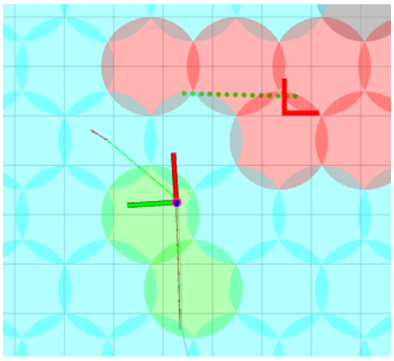


Figure 4.49. Start of the avoidance of the obstacle

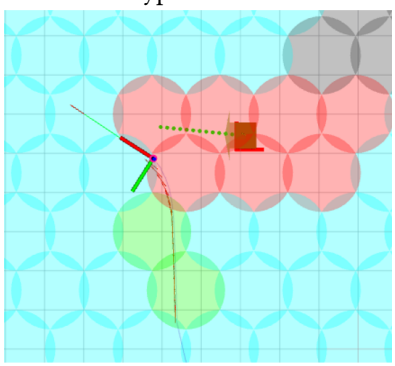


Figure 4.50. Further avoidance of the obstacle

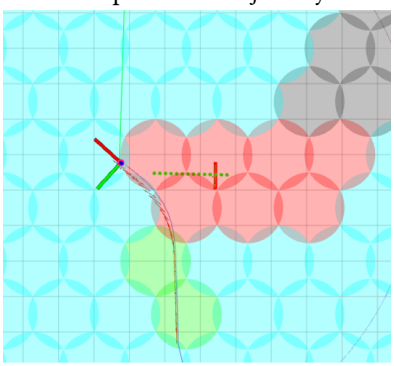


Figure 4.51. Obstacle avoided and USV navigates to waypoint

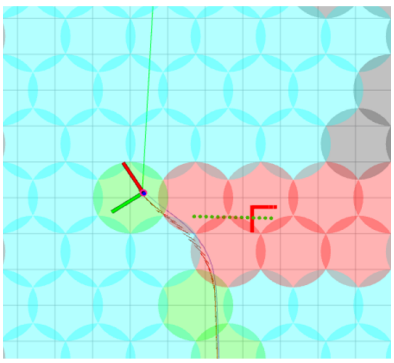


Figure 4.52. Further navigation to final waypoint

In Figure 4.47, the USV is travelling towards the final waypoint at $x = 80, y = 5$, while an obstacle in front of it moves perpendicularly to the direction of the USV. This obstacle moves at a velocity of $1m/s$ and is already tracked by the USV, but a future trajectory is not yet predicted. In Figure 4.48, the trajectory of the obstacle is predicted and the cells in front of it are blocked, hence their red colour. The

USV starts to avoid in Figure 4.49. The USV sets an intermediate waypoint to a free cell to avoid the obstacle. The direction to where the USV wants to sail is shown by the green line with the red arrow at the end. The waypoint from Figure 4.49 is corrected to another waypoint in a free cell in Figure 4.50. The waypoint is changed to a free cell that is as close to the direction in which the final waypoint at $x = 80, y = 5$ lies. In Figure 4.51, the obstacle is already sufficiently avoided to avoid a collision and the cells towards the final waypoint are free, thus the USV moves toward the final waypoint.

As established in the three scenarios, the USV can successfully avoid and overtake the head-on obstacles, as well as avoid a sideways obstacle. However, one has to take into account that the velocity of the USV is limited to $1.5m/s$ on a straight line and $1m/s$ while turning, additionally, the range of the LiDAR is constraint to $25m$. These limitations directly affect the obstacles that the USV can avoid. If the obstacles approach the USV too quickly, the USV will not be able to avoid them. The reason for this is two-fold. The obstacles will not be detected by the LiDAR in time to avoid them and the USV will not be able to act quickly enough to manoeuvre out of the way. For example, a head-on obstacle that moves with a velocity of $5m/s$ will collide with the USV in only five seconds after it was first detected by the LiDAR if the USV is static at that moment. If the USV travels in the general direction of the obstacle, this time decreases further. There is thus not enough time to detect, track, predict and avoid the obstacle at high velocities. That is why the three main situations encountered at sea were tested with obstacles that always had a lower velocity than the USV.

5. USV Development

In this chapter, the development and construction of the USV platform is described. In Section 5.1, the different hardware components used to build the platform are described in detail. The actual construction of the drone is explained in Section 5.2. Then, in Section 5.3, the configuration of the USV platform is described. Finally, in Section 5.4, the total moment of inertia for the whole USV is calculated.

The purpose of the USV is to be used as a research platform for the demonstration of an algorithm for autonomous navigation in a dynamic environment and can be further developed for different research demonstrations in the future. For autonomous navigation, the USV must be capable of determining its position and navigate towards given waypoints. Furthermore, the USV must be able to detect obstacles, predict their movement and subsequently avoid them while navigating towards the next waypoint. The choice of the systems architecture, hardware and software is aimed for a flexible and modular design so that for any future development, all the components can be easily upgraded, extended or replaced.

The development of the USV is based on the Maker Boat Basic [49]. The Maker Boat is a self-built unmanned vessel developed by Dr. Daniel Carlson, a research scientist at Helmholtz-Zentrum hereon GmbH, Germany. It is based on ArduPilot, an open-source autopilot system. The basic version of the Maker Boat can be controlled manually with a controller. The boat consists of a small surfing boogie board under which two thrusters are mounted. These thrusters are used to manoeuvre the boat forward, backward, and to turn. On the board, a watertight box is placed to accommodate the autopilot module, an onboard computer and provide place for a wide range of sensor systems. The autopilot is used to control the USV and can communicate with GPS to execute basic navigation tasks. For more advanced tasks like detection, tracking and avoiding dynamic obstacles, an onboard computer is used to process data from different sensor systems, and calculate a trajectory for the autopilot module. With this simple design and widely available components, it is aimed to build a low-cost, easy to maintain and extendable research platform with which different research developments can be performed.

In this thesis, the application is navigating in a dynamic environment. To achieve this objective, the USV is equipped with a 2D LiDAR to detect static and dynamic obstacles, a GNSS receiver to position itself in the world, an IMU to detect the heading of the USV and a stereo camera to support the navigation and monitor the situation. These sensors send their data to the onboard computer that processes the data and issues commands to the autopilot. In this research, the scope of development is limited to a calm sea state for a small-size model boat, with the target testing condition in a small lake since the focus here is rather about navigation of the USV in a dynamic environment than robust control in harsh conditions.

5.1. Components

In this section, the hardware components of the USV platform are presented. First, a bill of materials is given. Then the components and sensors that are used are described in more detail.

5.1.1. Bill of materials

The USV has the following components shown in Table 5.1.

Component	Quantity	Image	Description
Boogie board	1		This is the platform on which everything is mounted. It is the floating surface.
Watertight box	1		This is the box in which all the sensitive components that need protection from water are placed.
Cable gland	8		These glands provide a watertight connection in the box.
Switch	2		The switches ease the powering of the USV and computer.
T200 Thruster	2		These thrusters provide the USV with the required thrust and manoeuvrability.
Electronic speed controller	2		The controller will issue the command to the thruster to spin at a certain velocity.
7000mAh battery	2		One battery is reserved to power the onboard computer, while the other powers the thrusters, Cube and Herelink.
Cube Orange Autopilot	1		The autopilot will translate inputs for the manual control to the thrusters and will act as a communication bridge between the NUC and other components.
Intel NUC Onboard computer	1		The computer will process all the data and send commands to the autopilot.
Netgear 4G router	1		The mobile WiFi hotspot to establish remote control of the NUC.
Herelink	1		This is the controller for manual control and on which the onboard computer can be displayed.
RPLIDAR S1	1		The LiDAR is the sensor which is used to detect the obstacles.
Intel RealSense D435i	1		The IMU that is used for the LiDAR and Camera for real-time images.
Here3 GNSS receiver	1		The receiver to locate the USV in latitude and longitude and another IMU.
Trisonica wind sensor	1		The sensor for the wind speed, direction and temperature.

Table 5.1. List of used components

The Heart of the USV is the Cube Pilot, which will be described in Subsection 5.1.6. In the diagram of Figure 5.1, the interconnections between all the electronic components are shown. The red connections are connections to power the individual devices, while the green connections are the communication connections.

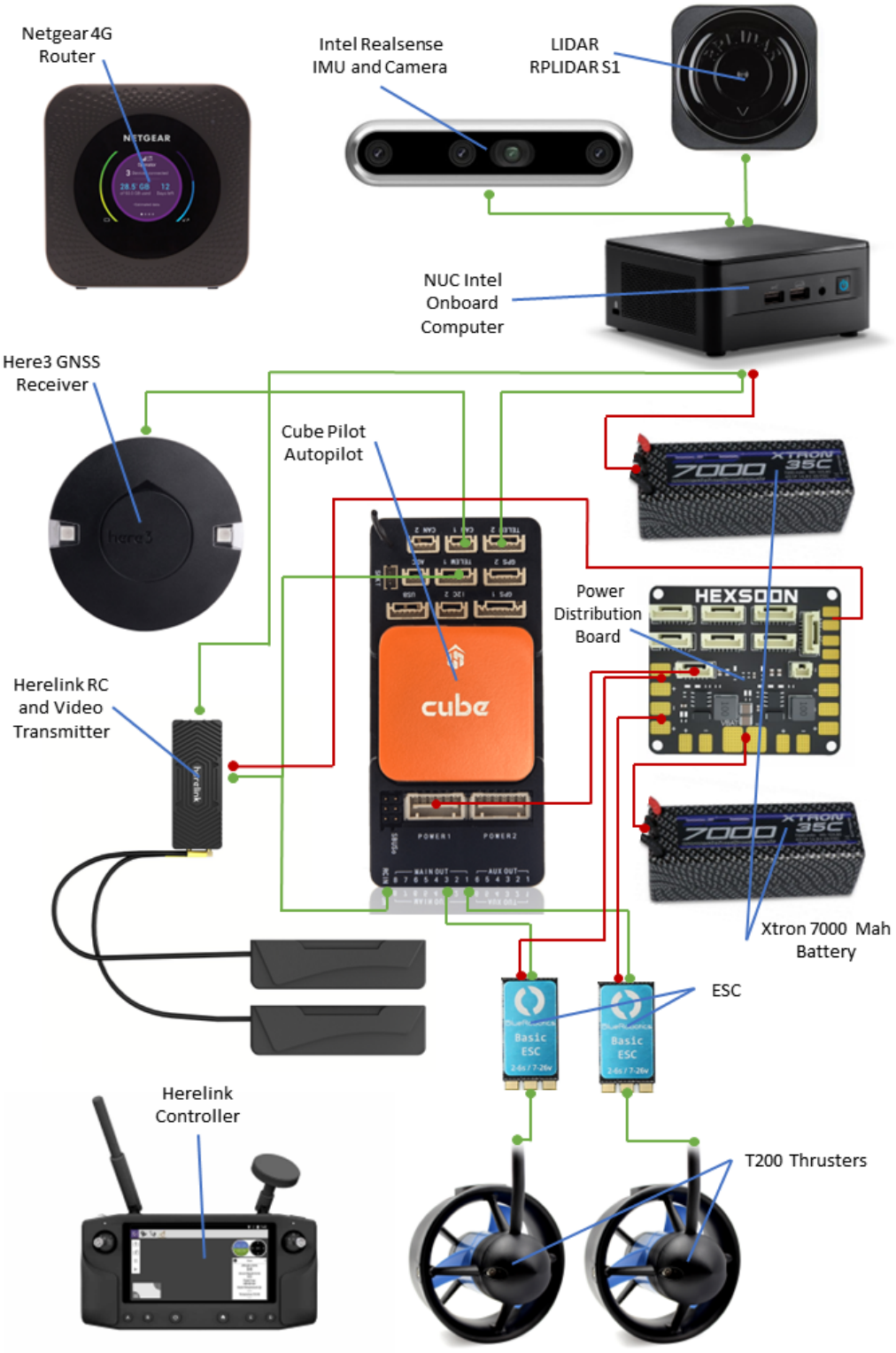


Figure 5.1. A diagram of the components. Red lines represent the power connections, green lines represent communication connections

5.1.2. Boogie Board

The basis of the whole USV is a floatable board on which everything is mounted. The board is 84.5cm long, 46cm wide and 5cm high. The board can carry loads up to 30kg. When a load of 30kg is applied, the board is not fully afloat anymore and the board will be slightly submerged. This is a situation that is to be avoided with the USV because this causes a situation in which water could come in contact with some electronic components. To mitigate the risk of damaging electrical components, these will be placed in a watertight box as explained in Subsection 5.1.3. The board, shown in Figure 5.2, has been chosen because of its wide availability as a small hobby surfing board and its ability to carry all the other components while still being balanced on the water surface.

Figure 5.2. The board used

5.1.3. Watertight box

A watertight box is mounted on the board. This box is a plastic storage box that can be securely closed with a lever on the sides of the lid. The box has inner dimensions of 30cm in length, 23.5cm in width and a height of 18cm. To accommodate the cables that need to be connected to components outside the box, cable glands that provide a watertight connection are used. These cable glands can be tightened around the cable with a hollow rubber ring that provides the watertight property of the gland.

5.1.4. Thrust generation

The USV is equipped with two T200 thrusters from Blue Robotics [23] to generate the thrust for manoeuvring forward, backward and the turning. These thrusters are mounted side-by-side and configured to act in skid steer drive, such that they could be mounted in a fixed position.

Figure 5.3. T200 thruster from Blue Robotics [23]

T200 thruster

The T200 thrusters are ready to use thrusters that are controlled with an electronic speed controller (ESC). The ESC gives the command for the controlled rounds per minute (RPM) for the thruster motor and hence also controls the thrust force through a signal based on pulse widths. As shown in Figure 5.4 and Figure 5.5, the initialise and stop signals lay on 1500μs with a deadband signal of around 25μs on each side. With a pulse width less than the stop signal, the thrusters give a reverse thrust, while with a larger pulse width they give a forward thrust [23].

Thruster Movement	**PWM Signal** (μs)
Forward	1525-1900
Neutral	1475-1525
Reverse	1100-1475

Table 5.2. The PWM signals needed to generate a thrust direction

The thrusters allow an input voltage of 10 Volts up to 20 Volts. The batteries that are used in the USV, however, have a nominal output voltage of 14.8 Volts, so only the profiles for 14 and 16 Volts are of concern and are shown.

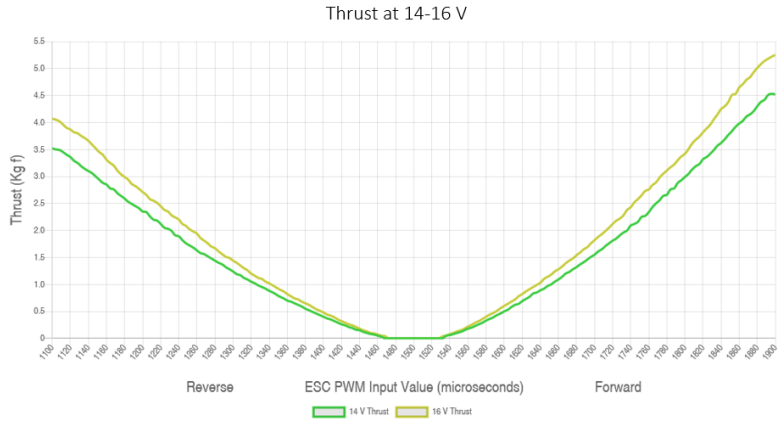


Figure 5.4. The thrust profile of a T200 thruster [23]

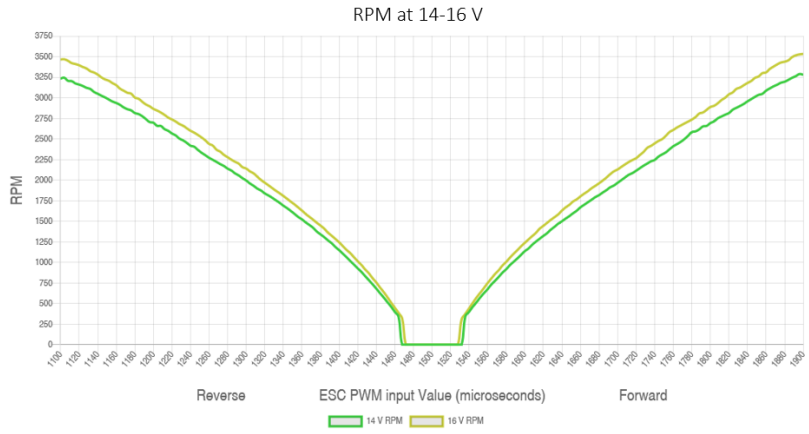


Figure 5.5. The RPM profile of a T200 thruster [23]

With the data of the RPM and some additional dimensions of the thruster, the theoretical flow rate can be calculated as follows [50]:

$$v_{flow} = v_{rot} \cdot tan(\theta) \tag{5.1}$$

with v_{flow} the flow velocity of the water in m/s, v_{rot} the rotational velocity of the propeller in m/s and θ the pitch of the propeller's blade in radians. v_{rot} can be computed by:

$$v_{rot} = \frac{RPM}{60} \cdot 2\pi r' \tag{5.2}$$

with $r' = t_{prop} \cdot r_0$ expressed in $m/revolution$ and where t_{prop} is the propeller's tip radius.

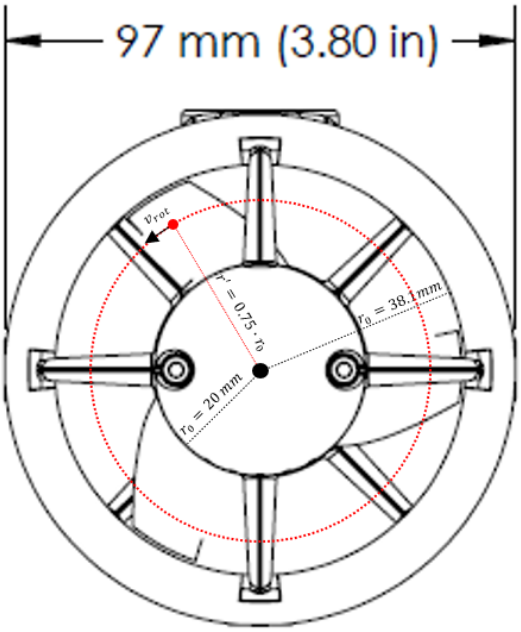


Figure 5.6. The different radii [23]

The pitch of a T200 thruster propeller blade is 22.5°, the propeller's tip radius can be taken at 75% and r_0 is 38.1mm. With the given RPM's from Figure 5.5 the water flow velocity is calculated. If the water flow velocity is then multiplied with the exposed surface of the propellers, the flow rate is obtained. The water flow rate is shown in Figures 5.7 and 5.8.

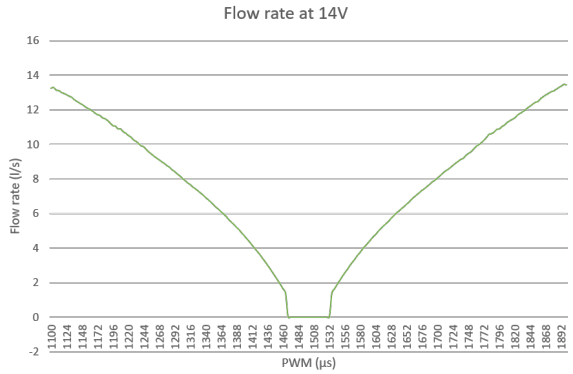


Figure 5.7. Flow rate at 14V

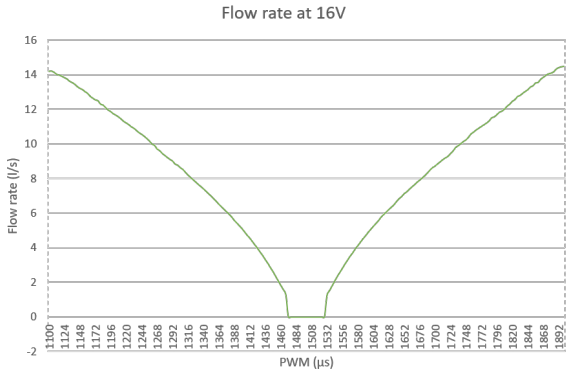


Figure 5.8. Flow rate at 16V

skid steer

The constructed USV uses a skid steer configuration for its manoeuvring. This means that the USV construction is easier because the thrusters are fixed. In this manner, the number of actuators for the USV is limited and the complexity of the USV is reduced. The USV has less complicated components and thus also less components that can break down, but there are only two actuators to control three states. These states are in the x-direction, the y-direction, and the yaw of the USV. Therefore, a balance must be found between the complexity of the construction and the ease of steering. Here, the use of two actuators and thus fixed thrusters is sufficient to attain the goals.

When a wheeled vehicle is used in the skid steer configuration, the wheels can only rotate around one axis and the turning ability of the vehicle is achieved by applying different torques on the wheels. An example is shown in Figure 5.9. In the figure, a right turn is performed by applying a smaller torque on the right wheels so that these wheels rotate slower and the vehicle starts to make a right turn. The advantage of skid steer turning is the use of less complicated components because the wheels can be mounted fixed on the body of the vehicle.

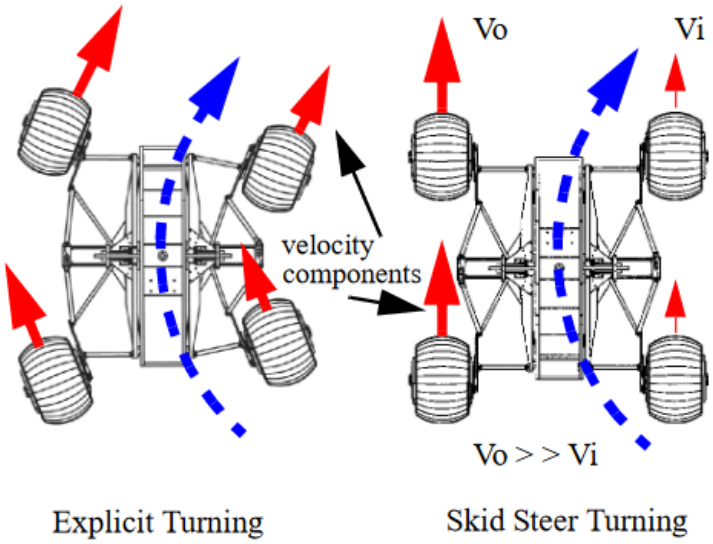


Figure 5.9. skid steer turning on a vehicle [51]

In the case of an unmanned surface vessel, skid steering can be achieved with the use of two fixed thrusters. These thrusters are mounted side by side to the USV and when performing a turn, the thrust given by each of the thrusters is different from each other in the same way that the velocity of the wheels differs on a vehicle. When performing a right turn, the right thruster will rotate less rapidly and will thus give less thrust so that the USV is able to make a turn to the right.

5.1.5. Batteries

Two batteries are used in the USV. One is used to exclusively power the onboard computer, while the second one is used to power all the other components. The batteries that are used are $7000mAh$ Lithium polymer (LiPo) batteries that contain four cells. They have a nominal output voltage of $14.8V$ and a power over time of $103.6Wh$. The continuous discharge is $35C$ *or* $245A$ and they have a maximum burst discharge of $70C$ *or* $490A$. The batteries used are from SLS and it is the XTron model [52]. A battery is shown in Figure 5.10

Figure 5.10. The battery used [52]

5.1.6. Autopilot

An important part of the USV is the autopilot. With an autopilot a USV can achieve full autonomy when everything is set up and programmed correctly. The autopilot used is the Cube Orange from Cube Pilot [53]. However, only the Cube is not enough to function properly; it needs to be connected to a board on which other devices can be connected to communicate with the Cube. The board used here is an ADS-B carrier board and is shown together with the Cube in Figure 5.11. The cube has a Cortex-M7 processor with 1 MB of RAM and 2 MB of flash and works at a frequency of $400MHz$. It also comes equipped with three redundant IMUs, two barometers and one magnetometer. Two of these IMUs are temperature-controlled such that they will always operate under their best working temperature. All these sensors are interconnected with a serial peripheral interface (SPI) . The Cube supports different kinds of firmware. In this thesis, PX4 firmware is used [54]. The Cube is configured with Mission Planner and QGroundControl which are described in Subsection 5.3.1. When fully configured and wired, the Cube is the heart of the USV, because all communication and commands to navigate will go through the Cube. The importance of the Cube is shown in Figure 5.1. In this figure, all the connections between the parts are shown and it is clear that every part has a direct or indirect connection with the autopilot. The manner in which the other hardware components are connected to the Cube is explained in Subsection 5.2.5.

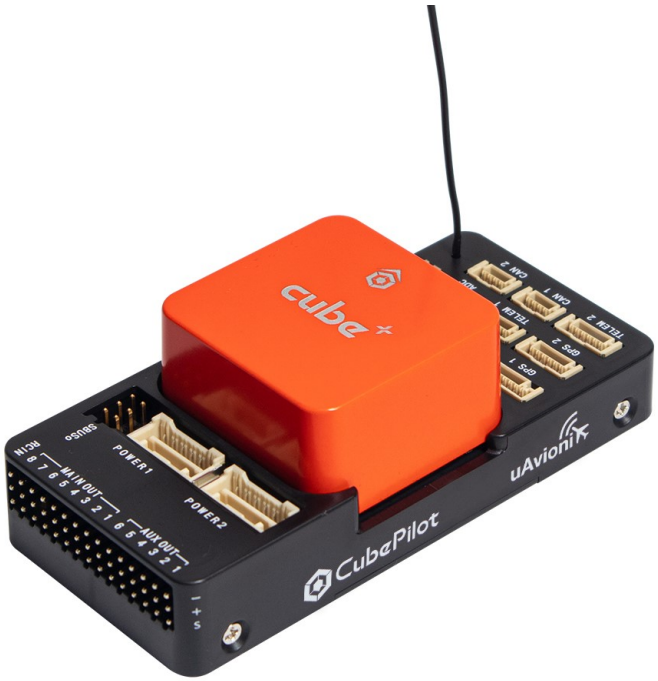


Figure 5.11. The Cube Pilot orange[53]

5.1.7. Onboard computer

To enable autonomous navigation, a lot of data from different sensors, which will be discussed in Subsection 5.1.9, needs to be processed. This is not possible on the autopilot, hence an onboard computer is required. The computer that is chosen is an Intel NUC with an i7 processor [55]. This computer has enough processing power to process all data from the sensors and to send out the appropriate commands to the Cube. The commands are sent using MAVLink [56]. This is a message protocol that is very lightweight and efficient. This is shown with the overhead, which is only eight bytes for MAVLink1 and fourteen bytes for MAVLink2, that is generated per sent packet. MAVLink allows all the components of a drone to communicate with each other. This thus also includes the onboard computer. The NUC runs on a version of Ubuntu 20.04, on which ROS Noetic [39] is used to process all the data.

Figure 5.12. Intel Nuc[55]

Netgear 4G router

The Onboard computer needs to have an internet connection to stream its video output to another computer, but an internet connection also allows for remote access to the onboard computer. The remote access is useful as it allows full access to the computers capabilities while the USV is already in the water. This capability is provided by establishing an internet connection with a mobile router. The mobile router that is used is the Netgear Nighthawk M1 that provides a dual band WiFi hotspot through a 4G LTE connection [57].

Figure 5.13. The Netgear Nighthawk M1[57]

5.1.8. Herelink controller

The USV needs to be steered when it is not navigating autonomously. This is done with the Herelink controller shown in Figure 5.14. Additionally, the controller can also be used to stream the video output of the computer and to adapt some parameters on the fly, because Mission Planner and QGroundControl, which are described in Section 5.3, are installed on the controller. The Herelink receives its signal from a transmitter installed on the USV. The Herelink thus establishes RC control, telemetery data and video transmission up to great distances. The transmission distance is advertised up to $20km$ when there is a direct line of sight between the controller and the USV. The transmission was tested over a distance of $100m$, which is much less than the advertised distance and during this testing the connection was maintained without difficulty. The signals are broadcasted at a frequency of $2.4GHz$ and the sensitivity for receiving is $-99dBm$ at a bandwidth of $20Mhz$ [58]. The Herelink is thus a powerful solution to control the USV and also to monitor the status of the USV using the video transmission of the visual output of the onboard computer. The most important specifications are grouped in Table 5.3.

Figure 5.14. The Herelink controller and transmitter [58]

Herelink specifications	Value
Range	$20km$
Sensitivity	$-99dBm$
Transmission frequency	$2.4GHz$
Bandwidth	$20MHz$
Video delay	$110ms$

Table 5.3. The Herelink specifications

5.1.9. Sensors

In this subsection, all the sensors used are discussed. These sensors include a LiDAR, IMU, GNSS receiver and a wind sensor. For each sensor, the important specifications are listed and a general explanation of their working is given.

LiDAR

The detection of nearby obstacles is done with the RPLIDAR S1 from Slamtec. The RPLIDAR S1 is a 360 degree 2D laser scanner with a 40 metre maximum range on a white object and a 10 metre range on a black object.The range resolution is $3cm$ and the total accuracy is $\pm 5cm$. It has a scanning frequency of $10Hz$ with a sampling rate of $9.2kHz$. The laser that is sent out has a minimal, maximal and typical wavelength of $895nm$, $915nm$ and $905nm$, respectively. The laser power is $28W$ and the pulse length is $10ns$. All of this leads to a laser safety class of 1, which makes it safe for the human eye [59]. All the specifications are grouped in Table 5.4.

Figure 5.15. The RPLIDAR S1 [59]

The RPLIDAR receives the echos back below the window where it was sent out, as shown in Figure 5.16. This means that the LiDAR must not be obstructed over the whole height.

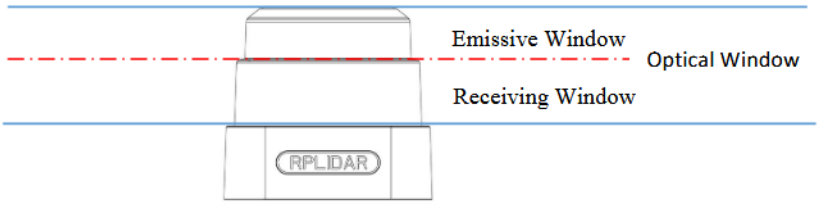


Figure 5.16. The emissive and receiving window of the LiDAR [59]

RPLIDAR S1 specifications	Value
Maximal range	40m *white object* 10m *black object*
Range resolution	0.03m
Total accuracy	$\pm 0.05m$
Scanning frequency	10-15Hz
Wavelength	$905nm \pm 10nm$
Pulse length	10ns

Table 5.4. RPLIDAR S1 specifications

IMU and camera

The RPLIDAR S1 is not equipped with any aid to sense the direction or angle in which it is mounted and moves. This is done with an inertial measurement unit. By combining the output of the LiDAR and the output of the IMU, the laser scan of the surroundings can be built up relative to the orientation and heading of the drone. The IMU used is the Intel RealSense D435i [60] and is equipped with an accelerometer and gyroscope. This system provides detection of movements and rotations in six degrees of freedom. The D435i can thus detect movement in the three axes, but also detect rotations, which are pitch, yaw and roll. The accelerometer has a sampling rate of 62.5 *or* $250Hz$ and an acceleration range of $\pm 4g$. The gyroscope, on the other hand, has a sampling rate of 200 *or* $400Hz$ and a range of $1000deg/s$ [60].

Figure 5.17. The intel Realsense D435i camera and IMU [60]

Additionally, the D435i is also equipped with different cameras. This camera will be used to monitor the movements of the drone on the Herelink controller. The RealSense camera is able to perceive depth using stereoscopy. It is equipped with a right and left camera. The two images of the cameras are then combined to perceive depth up to $10m$. When depth perception is used, the camera has a resolution of up to 1280x720 pixels and an accuracy of less than 2% at $2m$. The cameras for depth perception also have a wide field of view of $87°x58°$. When the camera is only used for a visual image of the surroundings, the red, green and blue (RGB) sensor is used. This sensor has a resolution of two megapixels and a narrower field of view of $69°x42°$ [60]. In this thesis, only the RGB sensor is used to visually see where the USV is heading. The most important specifications of the RealSense D435i are shown in Table 5.5.

RealSense specifications	**Value**
Gyroscope	
Sampling rate	200/400Hz
Range	$1000deg/s$
Accelerometer	
Sampling rate	$250Hz$
Acceleration range	$\pm 4g$
Camera	
FOV	$69°x42°$
Resolution	$2Megapixels$

Table 5.5. RealSense D435i specifications

GNSS receiver

The positioning and localisation of the drone is done with the Here3 from CubePilot. This is a u-blox high precision GNSS receiver that is able to receive the GPS, GLONASS, BeiDou and GALILEO constellations. It uses the DroneCAN communication protocol at a rate of $1Mbit/s$ [61]. When the Here3 is used as a standalone, the accuracy is $2.5m$. If used with RTK, the accuracy can be increased up

to 0.025m. The position determination is updated at a frequency of 8Hz. The Here3 is also equipped with its own IMU, that can be used as a back-up to the IMU of the Intel RealSense D435i. This IMU is an ICM-20948 9-axis motion tracker [62]. It incorporates a gyroscope, acclerometer and compass in one device. The most important specifications of the Here3 are shown in Table 5.6.

Here3 specifications	**Value**
GNSS receiver	
Accuracy 3D fix	2.5m
Accuracy RTK fix	0.025m
Update rate	8Hz
Communication protocol	DRONECAN 1Mbit/s
Gyroscope	
Range	250/500/1000/2000deg/s
Tolerance	±1.5%
Accelerometer	
Acceleration range	±2/4/8/16g
Tolerance	±0.05%
Magnetometer	
Range	±4900μT

Table 5.6. Here3 specifications

Figure 5.18. The Here3 GNSS receiver [61]

Figure 5.19. The TriSonica wind sensor [63]

Wind Sensor

The USV is equipped with a LI-550F TriSonica Mini Wind & Weather Sensor [63]. With this sensor, it is possible to measure the wind direction and speed. The sensor used is a 3D ultrasonic anemometer, which can also measure the temperature. Additionally, it measures the magnetic heading of the USV, the relative humidity and the angle of the wind. The angle of the wind consists of the pitch and roll of the wind. The wind sensor is shown in Figure 5.19. This sensor will not be used in this thesis, but can easily be integrated with the USV with the *trisonica_ros* package [64].

5.2. USV construction and mounting

The USV is a modular construction such that the components like the batteries can easily be interchanged without the need to break the USV down and build back up. In this section, the construction process is described.

The USV mainly consists of a boogie board, on which a watertight box is mounted. In this box, all electrical components are placed. On the outside of the watertight box, the different sensors are installed. The complete USV is shown in Figure 5.20.

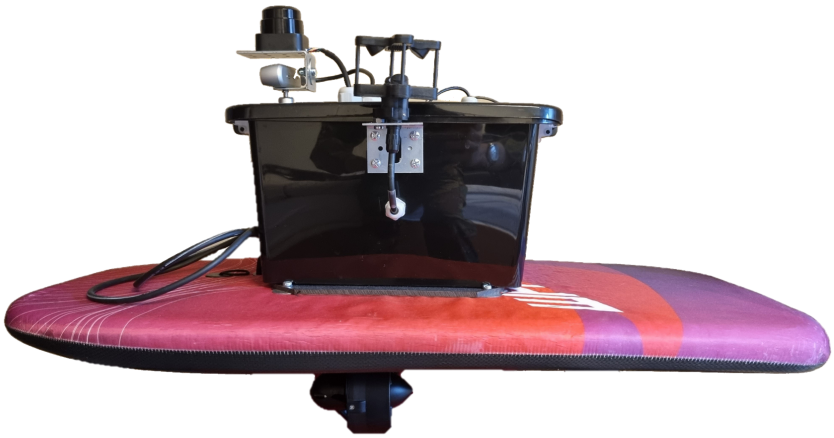


Figure 5.20. The complete USV

5.2.1. Thruster mounting

Both T200 thrusters are mounted in a fixed position and 18*cm* apart from each other on a 25*cm* square plate that is then attached underneath the boogie board as shown in Figure 5.21. The plate is fixed to the board with an identical plate that is put on top of the board, after which they are secured with four bolts through the boogie board. This is shown in Figure 5.22

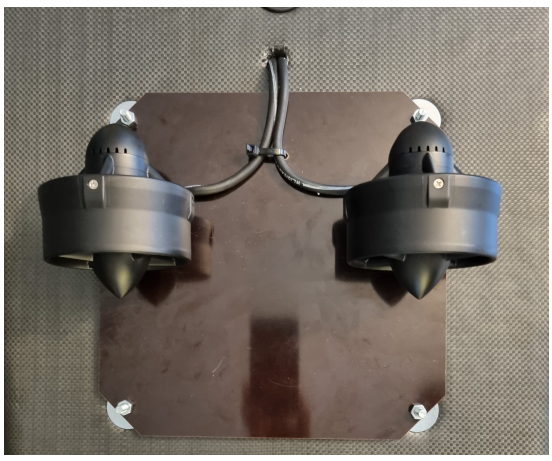


Figure 5.21. The thrusters underneath the boogie board

Figure 5.22. Top view of the board

Figure 5.23. The topview of the boogie board

The power cables of both thrusters are fed through an opening in the board such that they can be connected to the ESCs in the watertight box. This opening is made by drilling a hole in the board and reinforcing the opening with a PVC tube. The tube is wider at the end to protect the inside of the board from water. As can be seen in Figure 5.23, the watertight box is held in place on the board with two Velcro strips. These Velcro strips allow an easy removal of the box while still being strong enough to keep the box in place.

5.2.2. Mounting and interconnection of the electrical components

The connector of the battery needs to be exchanged with an XT60 connector. This is done by cutting the old connector off and soldering the XT60 connector to the cables. To provide power to all the components, a power distribution board is used. This board converts the 14.8V input voltage to 12V for the Herelink transmitter and the Cube and further distributes the 14.8V towards the ESCs and thrusters. The power distribution board is shown in Figure 5.24. The voltage that is needed for each component is shown in Table 5.7

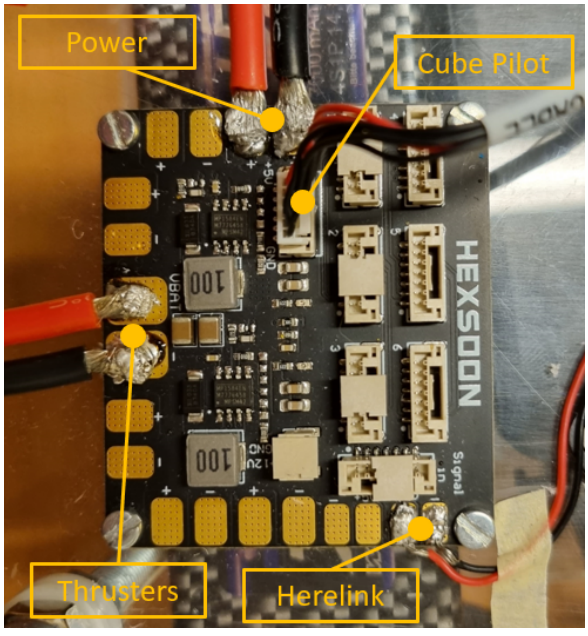


Figure 5.24. The power distribution board

Component	Operating voltage
T200 thruster ESC	7-20V
Cube Orange Autopilot	4.1-5.7V
NUC Onboard computer	12-19V
Netgear 4G router	Own battery
Herelink transmitter	7-12V
RPLIDAR S1	Via USB to NUC (5V)
RealSense D435i	Via USB to NUC (3.3V)
Here3	5V
Trisonica Wind sensor	Via USB to NUC (5-36V)

Table 5.7. Operating voltage of the different components

Figure 5.25. The changed XT60 connector

All the electrical components are put on a two-level construction as shown in Figure 5.26. This is done such that they can easily be put in the watertight box to shield them from water. This includes the two batteries, onboard computer, Cube Orange, Herelink transmitter and ESCs.

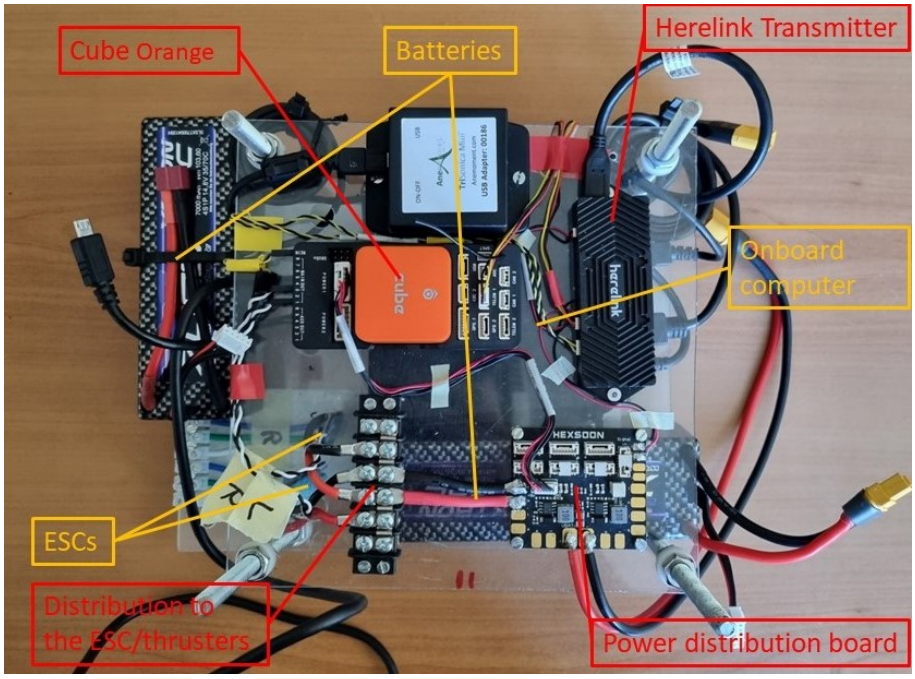


Figure 5.26. The two-level construction with the yellow components on the first level and the red on the second level.

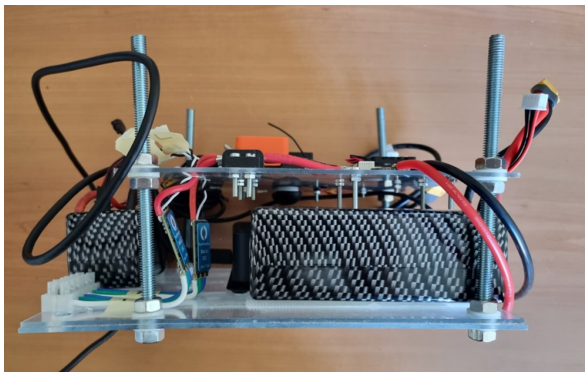


Figure 5.27. Left side

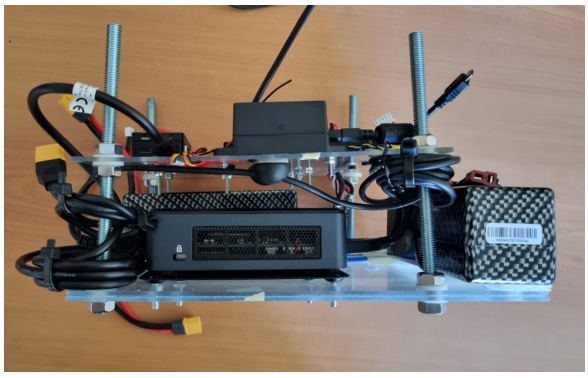


Figure 5.28. Right side

These levels are made with acrylic glass in which holes are drilled to fasten the components in place. All the parts, excluding the batteries are secured in place with bolts and nuts. The two levels are connected to each other by four rods that are threaded, such that nuts and washers can be used to fix the acrylic glass at a certain height. The batteries are attached to the acrylic glass with Velcro patches to facilitate easy removal. The batteries and computer are located at the bottom in order to locate the heaviest objects at the bottom of the USV. This is done to gain stability for the USV. The computer is also protected by placing it at the bottom.

5.2.3. Watertight box

All electrical components are subsequently placed in a watertight box to avoid any risk of damaging them.

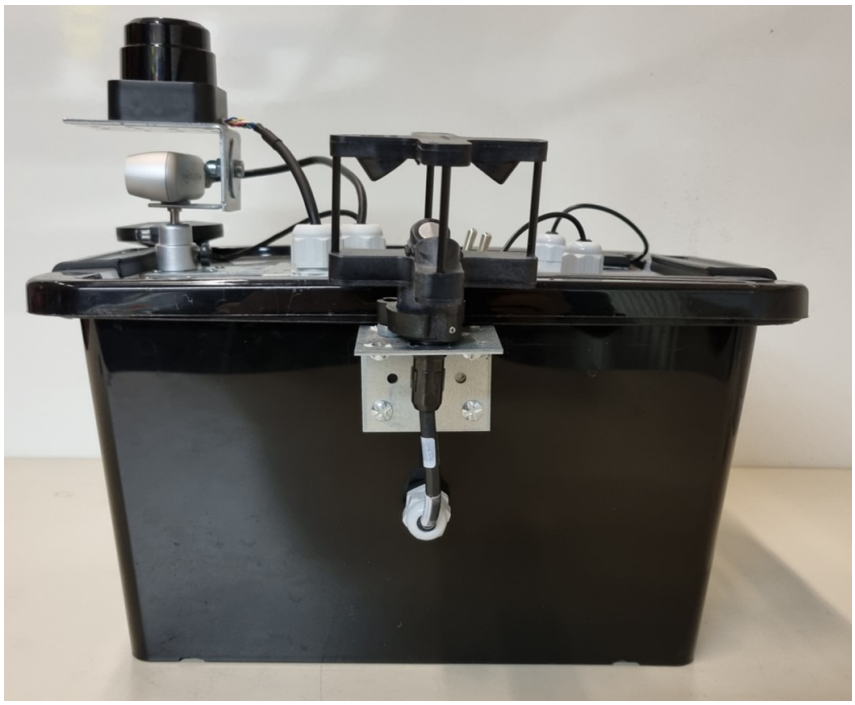


Figure 5.29. The box with everything installed

To facilitate the cables connected to the various sensors and the thrusters, eight holes are drilled in the box on which cable glands are installed. Five of these cable glands are installed on the lid of the box, close to the sensor for which the cable is needed. The remaining three cable glands are installed on the sides of the box. Two in the front for the thrusters and one on the left side for the optional wind sensor.

Figure 5.30. One of the used cable glands

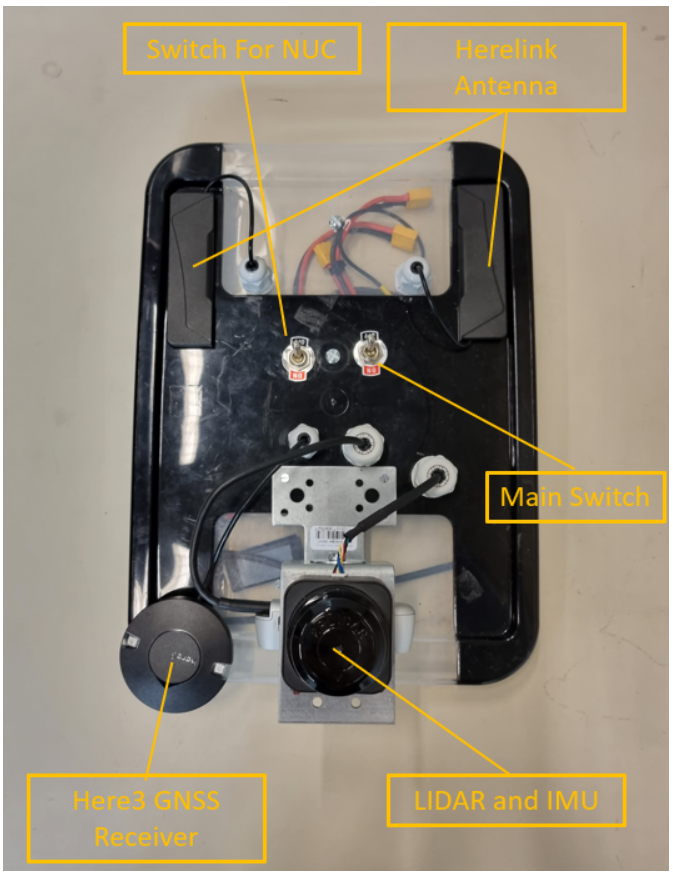


Figure 5.31. The lid of the box with everything installed

Aside from the cable glands, two switches are installed on the lid. These two switches are meant to easily power on and off the USV without having to connect/disconnect the power cables inside the box. One switch is used to connect a battery to the onboard computer, while the other is used to connect a battery to the power distribution board.

The watertight box itself is attached to the boogie board with Velcro straps for an easy attachment and removal. The box is placed in the centre of gravity of the board such that the nose of the USV will not be tilted upwards. When the box is installed and the USV is placed in the lake, it stays level while operating on the water surface.

5.2.4. Mounting of the sensors

The sensors of the USV are installed outside of the watertight box to guarantee the optimal functioning. The connecting cables of the sensors are put through the cable glands to the inside of the box, such that the box can remain watertight.

LiDAR and IMU

The RPLIDAR S1 does not have its own IMU system and will use the IMU from the Intel RealSense D435i. To be able to do this, the LiDAR and IMU need to move exactly in the same manner, with the same acceleration and velocity. Therefore, they are physically connected to each other, as shown in Figures 5.32 and 5.33. The LiDAR is mounted on top of the IMU which is connected to the watertight box.

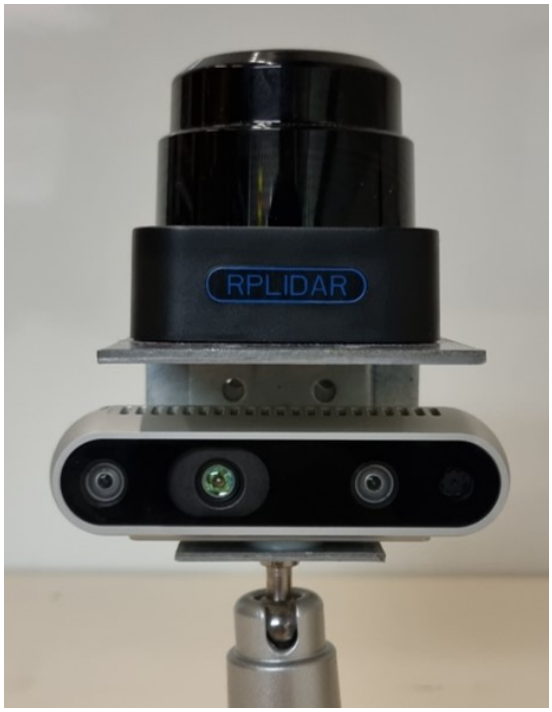


Figure 5.32. The IMU and LiDAR frontview

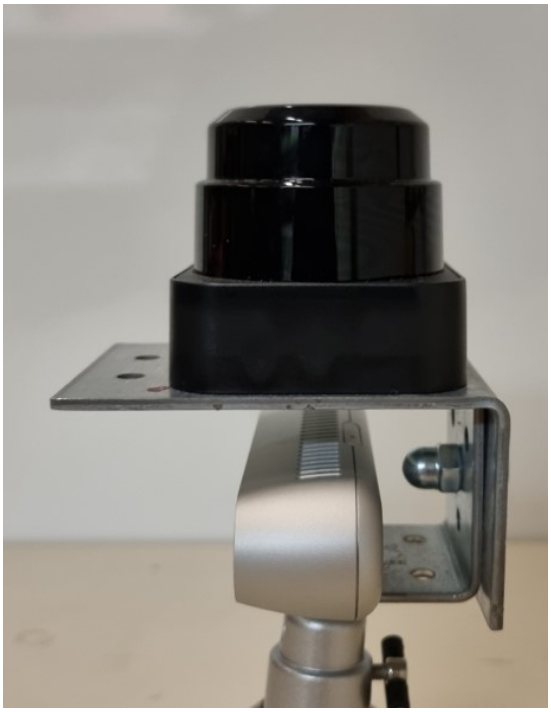


Figure 5.33. The IMU and LiDAR sideview

The connection between the box and IMU is reinforced with a steel plate to lessen the vibrations transferred from the flexible plastic lid to the sensors.

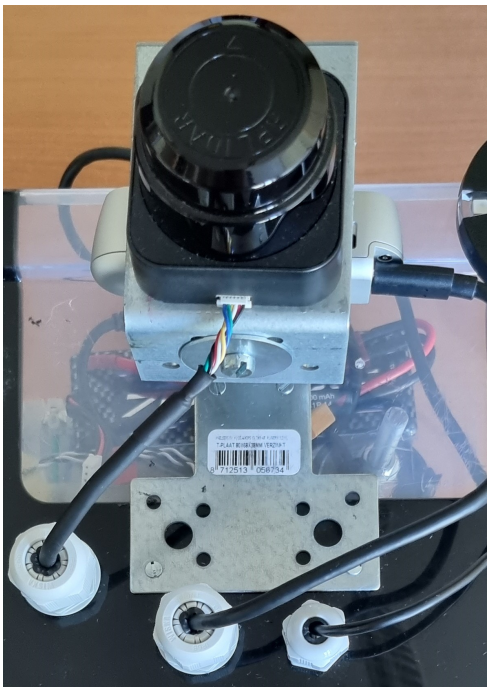


Figure 5.34. The metal reinforcement for the IMU and LiDAR

Here3 and Herelink antennas

As shown in Figure 5.31, the Here3 and the Herelink antennas are placed on the lid of the box. Upon testing this gave sufficient range for the experiments. They can both be attached to the box using double-sided tape.

Wind Sensor

A TriSonica wind sensor is placed on the left side of the watertight box. It is important that the wind sensor is free from the other sensors to guarantee the most noise-free measurements. Therefore, it is placed on a metal extension where the opening has been enlarged to accommodate the cable. The sensor is held in place with a clamp.

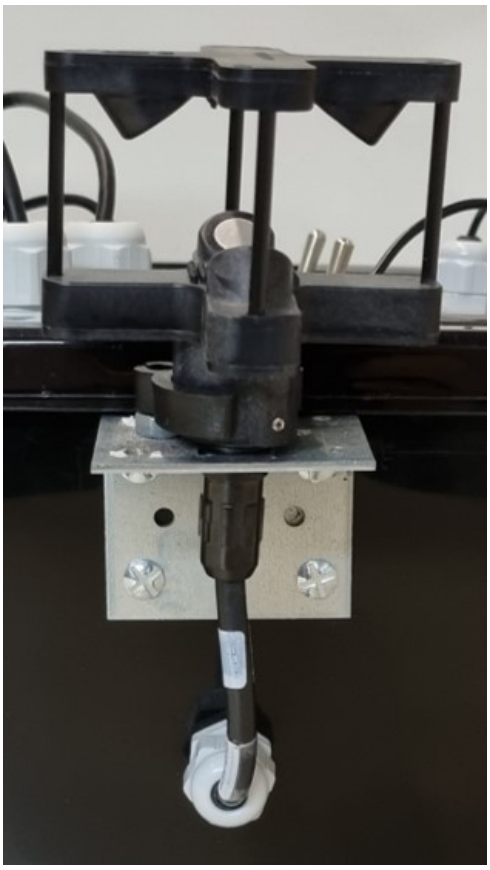


Figure 5.35. The Trisonica wind sensor

These are all the sensors that are mounted on the USV for this thesis. The USV platform lends itself to easily swap sensors with other models of the same type or to add sensors that could be useful for new tasks.

5.2.5. Wiring

When all the components are installed on the USV, they need to be interconnected with each other. The Cube Pilot has the most connections with other parts. The connections are shown in Figure 5.36. The Cube is powered by a connection from the power distribution board as was shown in Figure 5.24. The Herelink is connected via the telemetry 2 port and the RC IN port. The onboard computer can be connected in two ways. The first one is with the telemetry 1 port with a serial cable that is then converted to USB with the adapter shown in Figure 5.37. The second manner is with a micro-USB cable that is plugged in to the side of the Cube. When the first method is chosen, a serial cable with 3 connected wires must be made. These 3 wires are TX, RX and ground and are plugged into the Cube with a JST-GH 6pin connector. The wire is also shown in Figure 5.37. The thrusters and ESC are connected to the main out 1 and 3 for the left and right thrusters, respectively. This is done with a 2pin serial cable. Finally, the Here3 is connected to the Cube via the CAN1.

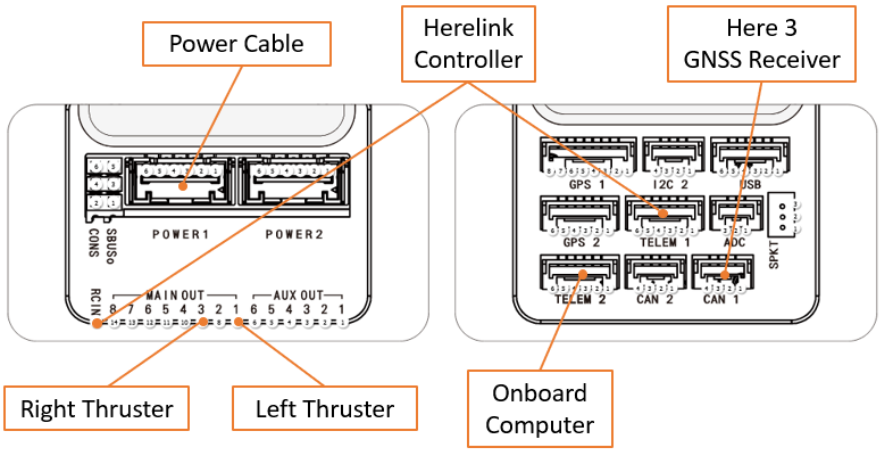


Figure 5.36. Scheme of all connections with the Cube Pilot[53]

The LiDAR and IMU are connected to the onboard computer with a simple USB connection. The video output of the computer is streamed to the Herelink with a HDMI to micro-HDMI converter.

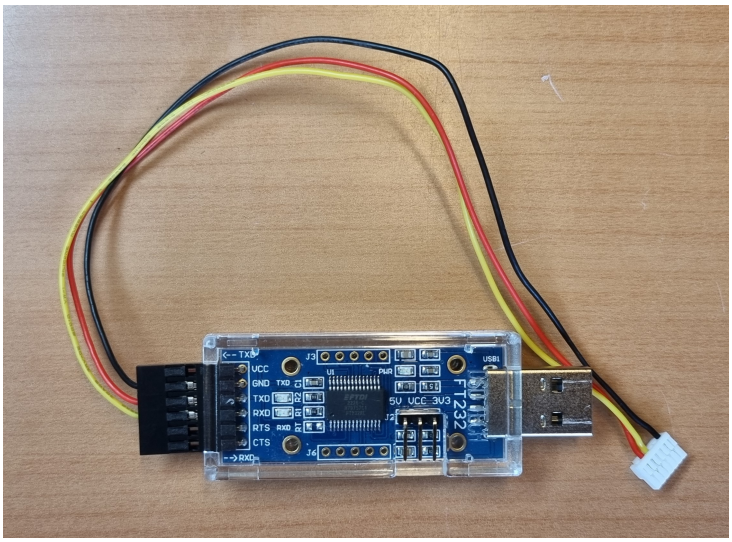


Figure 5.37. Serial to USB converter

5.3. USV configuration and setup

In this section, the configuration of the Cube Pilot and how all the other components are configured to correctly communicate with the Cube are explained. Furthermore, the communication from the onboard computer to the Cube is described. Finally, the communication between the LiDAR, IMU and computer is explained.

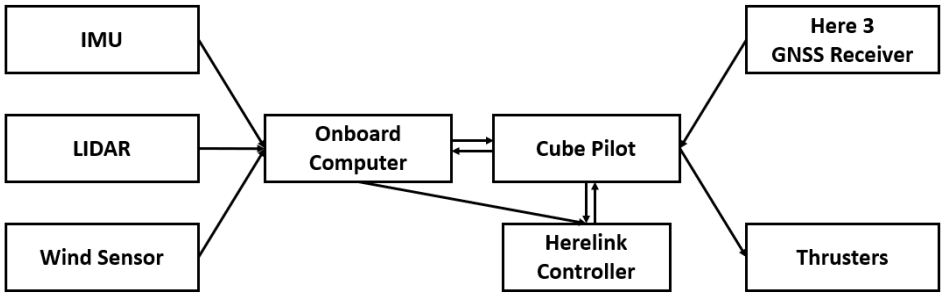


Figure 5.38. Communication between the components

5.3.1. Mission Planner and Cube Pilot

The settings of the Cube Pilot and the components directly connected to the Cube are all configured in the application Mission Planner [24]. Mission Planner, designed by Michael Oborne, is an application in which an autopilot board can be fully configured. First of all, it can be used to download and install the correct firmware on the Cube. Secondly, it is used to fully configure the ESCs that are connected to the thrusters, the Herelink controller and the Here3 GNSS receiver. All these individual components are configured and tuned in Mission Planner.

The firmware that is used for the USV is firmware that is suited for a rover and is Rover 4.2.3. This firmware is chosen because there is no firmware easily available for a vessel. However, the rover firmware is suitable for this application if the USV is operated in a skid steer configuration.

T200 thrusters

The T200 thrusters are linked to the Cube with the ESCs, which transform and transmit the commands from the Cube to the thrusters. To allow correct communication between these, the parameters of the thrusters need to be set on the Cube. This is done as follows.

- The function of the RC IN needs to be selected. This is done with the parameter SERVO1_FUNCTION and SERVO3_FUNCTION for the left and right thrusters respectively. Here, ThrottleLeft for SERVO1_FUNCTION and Throttleright for SERVO3_FUNCTION are selected.
- The throttle and roll channels need to be set, while yaw and pitch are not used in the rover model used in this case. All these parameters are found in the RCMAP list. The throttle is set at channel 3, the roll at 1. The pitch and yaw, while not used, still have a designated channel, which are 2 and 4, respectively.
- Different parameters can be used as arming checks. These parameters can be set in the ARMING list. Here, all the arming checks are disabled. The USV can be armed with a simple push of a button.
- The velocity of the USV is determined by pulsed signals. The maximal and minimal pulse width of these signals is already set in the parameter list. The ESC thus works with pulse width modulation (PWM) to send the correct thruster setting.

Herelink

The Herelink controller is automatically recognised by the Cube and supports the Mission Planner application, as well as the QGroundControl application [65] that is used for the manual control of the USV. After a calibration of the Herelink controller, the USV can be controlled manually. This calibration consists of moving the joysticks to the outer limits to set the maximal and minimal thrust that can be given to the thrusters, when controlling the USV manually.

Here3 GNSS receiver

The Here3 also needs to be setup such that the Cube and thus also the onboard computer receive an accurate position of the USV. To enable the use of the Here3, some parameters need to be adapted. First, CAN_D1_PROTOCOL is set to the DRONECAN driver. Then CAN_P1_DRIVER is set to 1 to activate the bus and GPS_TYPE is set to DRONECAN such that the Here3 is recognised. Finally, DRONECAN is added to NTF_LED_TYPES such that the visual lights of the Here3 correspond to certain statuses. These lights are useful to immediately recognise if there is a GNSS fix or not. When there is a GNSS fix, the lights will blink green when the USV is disarmed and continually green when the USV is armed.

When the initial configuration is completed, the Here3 needs to be calibrated. This is shown in Figure 5.39. After the calibration, the Here3 is ready for use.

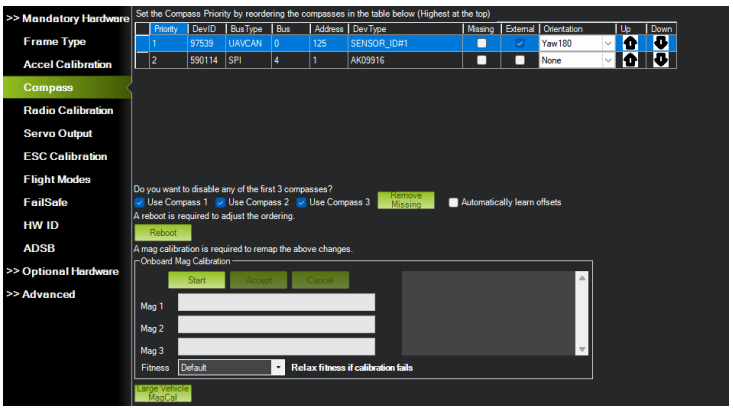


Figure 5.39. The calibration of the Here3

The Here3 can further be tuned by selecting certain GNSS constellations and excluding others. The USV uses the GPS and Galileo constellations to determine its position.

5.3.2. Onboard computer

To enable the USV to navigate autonomously, a companion or onboard computer is used. This computer needs to be able to issue commands to the Cube Pilot which will further distribute it to the correct system, for example to the thrusters. Additionally, the computer also needs to receive information about the different components to generate a complete picture of the surroundings. In this study, the information from the LiDAR, the Intel RealSense IMU and the Cube, together with Here3, is used.

The onboard computer can be accessed remotely through Remmina, the screen sharing application of Ubuntu [66]. This application uses Virtual Network Computing (VNC) [67]. VNC relates the keyboard and mouse inputs from the remote computer to the onboard computer using the Remote Frame Buffer protocol, while the onboard computer transmits the visual screen updates to the remote computer. There are 2 methods to access the onboard computer. The first is when both computers are connected to the same network. If this is the case, it suffices to connect with the onboard computer by searching on its IP-address. The second case is when the two computers are on different networks. Here, port forwarding must be set-up on the onboard computer and router such that the incoming connection to

the router is forwarded to the onboard computer. When this is done, both computers (the onboard and the remote computer) can communicate and the onboard computer can be accessed remotely.

Cube Pilot

The Cube Pilot is connected to the computer by USB-to-micro-USB. On this link all communication will be shared. To enable the link, the parameter SERIAL0_PROTOCOL needs to be set to MAVlink1 and the baud rate needs to be set to an appropriate rate of 57600 baud. For the computer, there are different options available. The first one is using MAVProxy. MAVProxy is installed from the github repository of Ardupilot_wiki [68]. With MAVProxy commands can be issued from the computer to the Cube and data from the Cube can be read out. MAVProxy is ran with the following commands:

```
$ sudo chmod 666 /dev/ttyACM0
$ mavprox.py --master=/dev/ttyACM0
```

The first line gives the appropriate permissions for the used port on the computer. The second line establishes a two-way connection with the Cube.

The second possibility is using MAVROS. MAVROS is a package from ROS that allows to have MAVLink communication between a computer on which ROS is ran and an autopilot, which is the Cube Pilot in this case. MAVROS thus has the same features as MAVProxy with the addition of the ROS compatibility. MAVROS is installed from the github repository of ROS [69]. The installation of the MAVROS files needs to be from source because MAVROS is not updated anymore on ROS noetic. The supported community has shifted towards the newer ROS2.

LiDAR

The LiDAR is connected to the computer via USB. Just like with the Cube, the port needs to be granted permission to send and receive data. This is done in the same manner as with the Cube, but with ttyUSB instead of ttyACM. With the aid of the rplidar_ros package [70] , a driver for the LiDAR is provided which reads the RPLIDARs Software development kit (SDK) and then converts them in ROS *sensor_msgs/LaserScan* messages. Each *LaserScan* message contains an array of ranges and corresponding angles for each measured point. An output of such a message is shown in Figure 5.40.

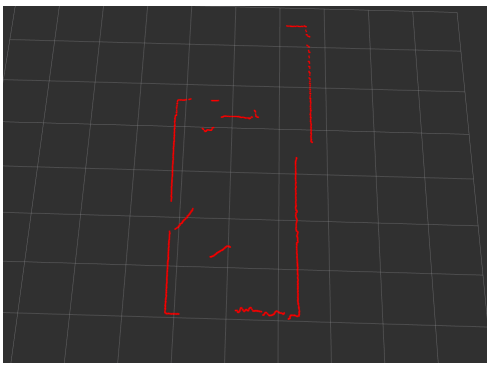


Figure 5.40. Laserscan of a room using the RPLIDAR S1

IMU

The Intel RealSense D435i is connected with an USB cable to the NUC and the same permissions must thus be granted as with the LiDAR before the IMU can function. The measurements of the IMU are used, through the ROS wrapper for Intel RealSense devices [71]. When the ROS node is launched for the IMU, a ROS topic containing the information from the IMU is published, namely */camera/gyro/imu_info*. This topic is published alongside several other topics that are of lesser interest for this thesis. An example

of use for the other topics is to generate point clouds of the camera image to do SLAM with the D435i. Alternatively, the gyro and acceleration of the motion module can also be visualised using the Intel RealSense-Viewer application. This is shown in Figure 5.41. The RealSense-Viewer application is a user-friendly application to easily check the correct functioning of the D435i and to download additional firmware if necessary.

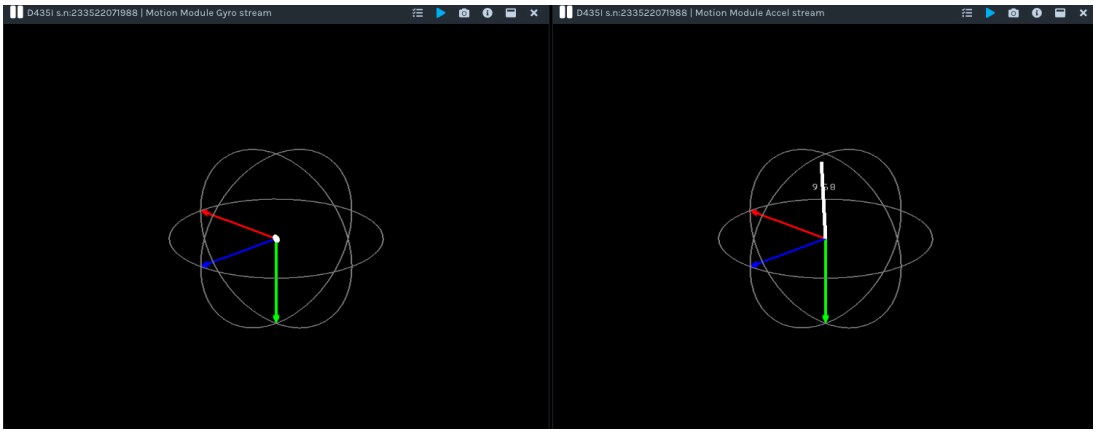


Figure 5.41. Visualisation of the IMU in Intel RealSense-Viewer, left is the output from the gyroscope and right the output of the accelerometer

It is important to note that the definition of the coordinate system of the IMU is different from the default coordinate system of ROS. This means that a transformation between both coordinate systems has to be done. The transformation to the ROS coordinate system can be done in two steps. First, a counterclockwise rotation of 90° around the y-axis and then a clockwise rotation of 90° around the x-axis. This can be described with the following rotation matrices:

$$
\begin{aligned}
&\textit{First and second rotation}: \\
&\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \; ; \begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \\
&\textit{Total rotation}: \\
&\begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}
\end{aligned}
\tag{5.3}
$$

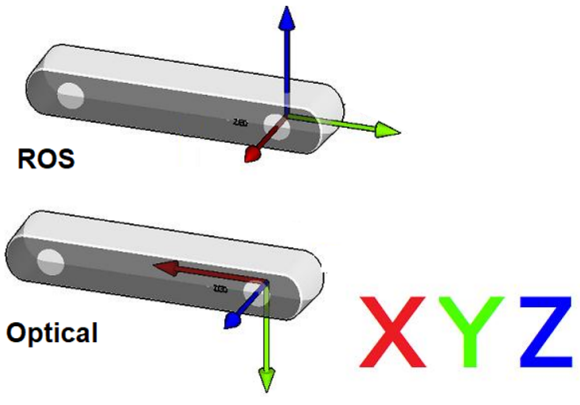


Figure 5.42. The coordinates sytems from ROS and the RealSense IMU [60]

The IMU needs to be calibrated when it is used for the first time. This is done with the python calibration file *rs-imu-calibration.py* from Intel [60]. The calibration consists of holding the IMU in six different positions. These positions each correspond to placing the IMU on one of its sides. Each position needs to be held for around 4 seconds. When the calibration of the side is done, it will be shown in the terminal of the Python script. Afterwards, the results of the calibration will be written to the IMU. This calibration is needed to ensure the best accuracy for the IMU.

Additionally, the effect of gravity must be removed from the accelerometer. This is done by subtracting 9.80356. This is the effect of gravity as obtained in the calibration.

5.4. Moment of Inertia

In this section, the total moment of inertia of the USV is calculated theoretically. To calculate the moment of inertia, the USV can be modelled as a combination of two cylinders for the thrusters, a cuboid for the boogie board and an additional cuboid for the watertight box.

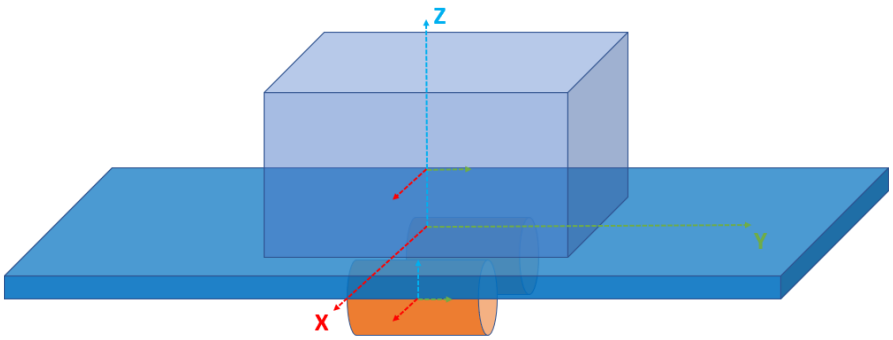


Figure 5.43. Approximation of USV for the calculation of the moments of inertia

Thrusters

The T200 thrusters are modelled as a cylinders with a length of 11.3cm and a radius of 5cm. The mass is 0.344kg in air and 0.156kg in water [23]. The thrusters are placed at a distance of 9cm from the centre of mass in the x-direction and at a distance of 7.5cm in the z-direction. The moments of inertia can then be calculated as follows:

$$\begin{aligned} I_{xx} &= \frac{1}{12}m(3r^2 + l^2 + d_x) = \frac{1}{12} \cdot 0.344(3 \cdot 0.1^2 + 0.113^2 + 0.075^2) = 7.4229 \cdot 10^{-4} kgm^2 \\ I_{yy} &= \frac{1}{2}mr^2 + md_y^2 = \frac{1}{2}0.344(0.1^2 + 0.09^2 + 0.075^2) = 0.0028 kgm^2 \\ I_{zz} &= \frac{1}{12}m(3r^2 + l^2 + d_z) = \frac{1}{12} \cdot 0.344(3 \cdot 0.1^2 + 0.113^2 + 0.09^2) = 8.1324 \cdot 10^{-4} kgm^2 \end{aligned} \tag{5.4}$$

Where d_x, d_y and d_z are the distances needed to have the moment of inertia in the centre of mass of the whole USV.

Boogie board

The board is modelled as a cuboid with a length of $84.5cm$, a width of $46cm$ and a height of $5cm$. The board weighs $0.1kg$. The centre of mass of the entire USV coincides with the centre of mass of the board and therefore no translations must be added to the formulas of the moments of inertia. The moments of inertia are calculated as follows:

$$\begin{aligned} I_{xx} &= \frac{1}{12}m(l^2 + h^2) = \frac{1}{12}0.1(0.845^2 + 0.05^2) = 0.0060kgm^2 \\ I_{yy} &= \frac{1}{12}m(w^2 + h^2) = \frac{1}{12}0.1(0.46^2 + 0.05^2) = 0.0018kgm^2 \\ I_{zz} &= \frac{1}{12}m(l^2 + w^2) = \frac{1}{12}0.1(0.845^2 + 0.46^2) = 0.0060kgm^2 \end{aligned} \tag{5.5}$$

Watertight box

The box is also modelled as a cuboid with a length of $30cm$, a width of $23.5cm$ and a height of $18cm$. The box weighs $4.65kg$. The box is placed on the board with its centre of mass $11.5cm$ above the centre of mass of the board and hence of the whole USV.

$$\begin{aligned} I_{xx} &= \frac{1}{12}m(l^2 + h^2 + d_x^2) = \frac{1}{12}4.65(0.3^2 + 0.18^2 + 0.115^2) = 0.0526kgm^2 \\ I_{yy} &= \frac{1}{12}m(w^2 + h^2 + d_y^2) = \frac{1}{12}4.65(0.235^2 + 0.18^2 + 0.115^2) = 0.0391kgm^2 \\ I_{zz} &= \frac{1}{12}m(l^2 + w^2) = \frac{1}{12}4.65(0.3^2 + 0.235^2) = 0.0474kgm^2 \end{aligned} \tag{5.6}$$

Total moment of inertia

The total moment of inertia is then the sum of all the individual ones. This yields the following moments of inertia:

$$\begin{aligned} I_{xx} &= 0.0600kgm^2 \\ I_{yy} &= 0.0464kgm^2 \\ I_{zz} &= 0.0550kgm^2 \end{aligned} \tag{5.7}$$

6. Experiments

In this chapter, the built USV platform is tested. First, the manual control of the USV is discussed in Section 6.1. Then, in Section 6.2 the different sensors are tested and the accuracy of the GNSS receiver is calculated. In Section 6.3, the thruster control from the onboard computer is discussed.

6.1. Manual Control

The USV can be controlled manually through the QGroundControl application. The settings of the USV in QGroundControl for this are already discussed in chapter 5. The RC controller is calibrated in the RADIO section of QGroundControl, after which the USV can be controlled manually. This manual control is possible without the use of the onboard computer; only the Cube is needed to relay the commands of the controller to the ESCs. The ESCs will finally give the command to the thrusters. When fully configured, the USV can move back and forward and turn to both directions, hence it can be fully controlled. The USV can also be operated without a direct LOS, because the RealSense camera's output is shown on the Herelink controller and thus one can see where the USV is headed.

6.2. Sensor testing

In this section, the different sensors used are tested and their precision is calculated if necessary.

6.2.1. IMU

As already said in Chapter 5, the IMU of the RealSense D435i is calibrated using the *rs-imu-calibration.py* file. After this calibration the IMU is corrected and there is no output from the gyroscope or accelerometer anymore when it is static, as it should be. The IMU measurement data is provided to the onboard computer with the ROS topic *\camera\imu*, which sends out data messages at a rate of $200Hz$. The other IMUs from the Cube and the Here3 are calibrated in the QGroundControl application by turning and moving the USV over all the axes.

6.2.2. GNSS receiver

The Here3 determines its position using the GPS and Galileo GNSS constellations and sends its determined position using MAVlink to the onboard computer. The onboard computer converts the received data to a ROS message that is sent out. This message is *\mavros\global_position \position*. The position is also converted to the local map used by ROS, and this is sent out with the message *\mavros\local_position \pose*.

The GNSS receiver is tested with static positions measurements over a longer period of time. The measurements were done in the position 50.84426 *Lat* and 4.39191 *Long* over a period of 25 minutes. These results are then converted to the Universal Transverse Mercator coordinate system (UTM) to easily show the difference in position between measurements in metres. The result is shown in Figure 6.1. To asses the accuracy of the measurements, the probable circular error (CEP) is calculated. The CEP has a probability of 50%. Twice the distance root mean squared (2DRMS) is also calculated. The

2DRMS has a probability of 95%. These are two of the most commonly used position accuracy measures and they are calculated as follows [72]:

$$\begin{aligned} CEP &= 0.59 \cdot (\sigma_x^2 + \sigma_y^2) \\ 2DRMS &= 2\sqrt{\sigma_x^2 + \sigma_y^2} \end{aligned} \tag{6.1}$$

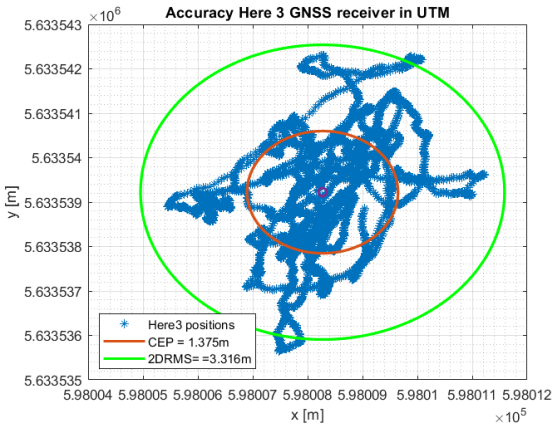


Figure 6.1. Accuracy measurement of the Here3 GNSS receiver

The CEP of the measurements is equal to 1.375m, while the 2DRMS is equal to 3.316m. This means that statistically 50% of the measurements will be closer than 1.375m to the actual position and 95% will be closer than 3.316m. This accuracy is not great when navigating in small environments where a precise position is needed. To improve the accuracy, RTK can be used. With RTK a permanent static reference station is used to correct for errors in the position estimation. This can then correct the position up to centimetre-level accuracy. Another possibility is the fusion of IMU and GPS data to correct for errors in the GPS data [73]. The fusion is done with a Kalman filter through which the reliability of the measurements is improved as the extreme outliers are filtered out [74]. The IMU used for this is the IMU of the Cube. When this is done, the accuracy of the positioning data increases drastically. This is shown in Figure 6.2, where the real position is in the origin $(0, 0)$ and the variations in x and y are shown. The measurements now result in a CEP of 0.3404m and a 2DRMS of 0.8165m, which is accurate enough to position the drone in the world.

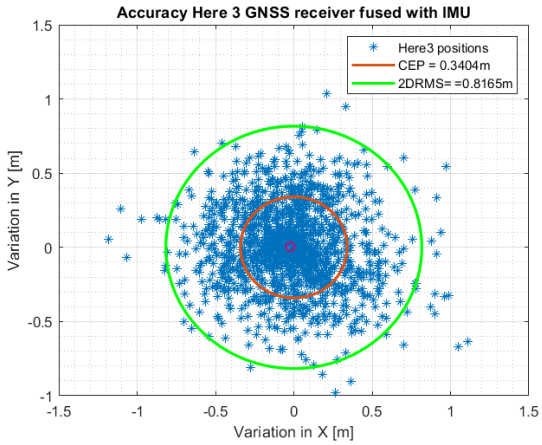


Figure 6.2. Accuracy of the Here3 after fusion with the IMU

However, this enhanced accuracy was not always obtained with the Here3 and sometimes it falls back on the poorer accuracy of the first test. The accuracy of 2.5m as given in the specifications of the Here3 in Chapter 5 is best used to characterise the overall accuracy of the Here3.

6.2.3. LiDAR

The LiDAR is one of the most important sensors on board of the USV. It provides detection of the surroundings under all conditions. The output of the LiDAR is visualised in Figure 6.3. The output of the LiDAR is just as in the simulation, further processed to track obstacles and make estimations. When the LiDAR is on it sends measurement data towards the onboard computer. Between measurements, there is a variance on the position that is caused by the accuracy of the LiDAR, but also by the accuracy of the GNSS receiver. The range of the object is mapped to the coordinate system of the USV and that coordinate system is positioned in the global coordinate system using the GNSS receiver. Therefore, the inaccuracies of the GNSS receiver also affect the accuracy of the LiDAR measurements. When the signal from the GNSS receiver is bad and the positioning is not accurate, it will not be possible to correctly position the obstacle. However, this is not a problem for the detection and tracking of the obstacles as these are mapped relative to the coordinate system of the USV.

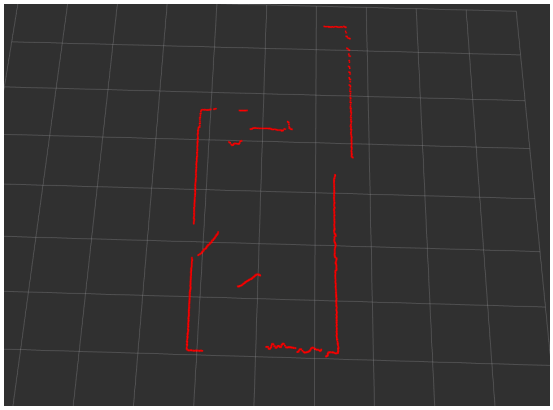


Figure 6.3. Laser scan of a room

Detection and tracking

The DATMO package is also implemented on the real-life USV to enable the tracking of dynamic obstacles. It is tested by positioning the USV in a static position and then a person walks towards the USV. When a person approaches the USV, he is detected and tracked, as can be seen in Figure 6.4. In the figure, the two red groups are objects coming towards the USV. In reality, it are the legs of the person walking towards the USV. The figure also shows that the objects are tracked as indicated by the arrows that represents the velocity. In Figure 6.5, the prediction of the trajectory is shown. As can be seen, the prediction of the objects' position is still done, which means that the objects can still be linked to each other over longer periods of time between measurements.

Figure 6.4. Objects moving towards the USV

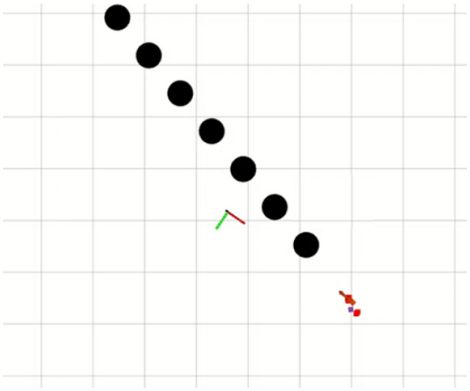


Figure 6.5. Prediction of trajectory

6.2.4. Combination of sensors

All individual sensors are combined to be able to perform the same task as done in the simulation in chapter 4, namely detecting the obstacles and mapping them in blocked cells such that the USV can navigate through the free cells. The sensors used include the 2D RPLIDAR S1, D435i for its IMU and the Here3 GNSS receiver. Here, all the sensors work and give their respective outputs; however, due to the inaccuracy of the GNSS receiver, the allocation of cell states in the occupancy grid does not function properly. As the position given by the Here3 changes constantly, the position of the detected objects also changes location in the generated map. Due to these inaccuracies, the USV cannot rely on the supplied locations of itself and the object, hence obstacle avoidance becomes difficult. An example of what is registered with the sensors is shown in Figure 6.6. In the figure the cell radius is set at $0.5m$.

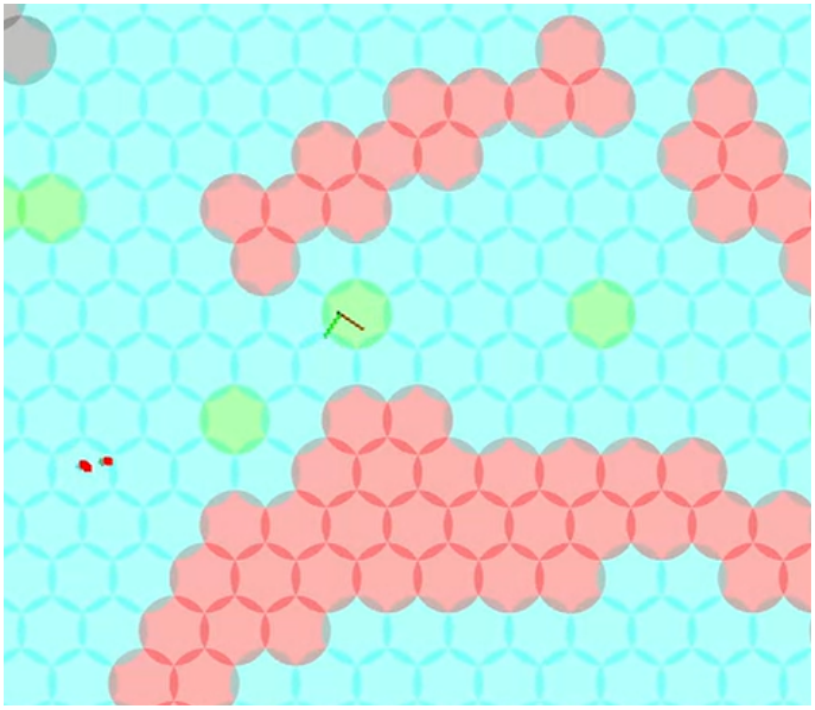


Figure 6.6. The navigation method output in real-life

In Figure 6.6, it is shown that the USV jumps in position through the cells that are coloured green. It is noted that not all the cells where the USV was for an instance according to the position from the Here3 are coloured green. It is also seen that the USV registers the obstacle, shown by the two red dots in the figure. The cells on which the red dots are now are not yet coloured red, but it is seen that the objects also shift unwantedly in position. The object moved around the USV in a circle at a constant distance. However, this circle is not clearly visible in the figure. This is mainly caused by the inaccuracy of the GNSS receiver as the object detection and tracking work fine, as explained in Subsection 6.2.3. The difference between detection and tracking and the combination of the sensors is that the detection and tracking are relative to the coordinate system of the USV, while for the combination the positioning of the USV and the objects is relative to the global coordinate system.

The error in positioning is thus a problem for the proper functioning of the navigation method. Alternative methods must be provided to correct for these errors. One could look to Simultaneous Localisation And Mapping (SLAM) as a solution to the problem. With SLAM, the USV receives input to calculate its position and that of the objects from the 2D LiDAR or the RealSense Camera. If the LiDAR is used for SLAM, the *hector_mapping* package [75] can be used. If one wants to use the RealSense camera for SLAM with the cameras, ORB-SLAM2 [76] can be implemented.

6.3. Thruster control from NUC

The thrusters rotation speed can be set and changed from the onboard computer with the following commands. When this is done, the Cube acts as a relay station to convert the sent message into actual PWM signals that are used by the ESC to control the thrusters.

```
$ rosrun mavros mavsys mode -c guided
$ rosrun mavsafety arm $ ids 255,240
$ rosrun mavros mavparam set SYSID_MYGCS 1
$ rostopic pub -r 1 /mavros/rc/override mavros_msgs/OverrideRCIn...
$       '[1500,1500,1500,1500,875,875,875,875,0,0,0,0,0,0,0,0,0,0]'
```

In this configuration, the thrusters can be commanded with the eight first channels. In reality, only two channels need to change values to steer the USV. These are channels three and one. To command forward, the value of channel three must be modified. When the USV needs to turn right, the value in channel one must be modified to be lower than 1500. If it needs to turn left, the value must be larger than 1500. The value for which the thrusters remains static is 1500 in channel three. At that moment the thrusters will not spin, hence no thrust will be generated. Between the zero value of 1500 for left/right steering and forward thrust and the maximal input there is a difference of 400. The needed PWM signal to the thruster for a certain velocity can be read from the graphs that are shown in Figure 5.7 and Figure 5.8.

7. Conclusion and outlook

This work had two objectives, which were the development of a navigation method in a dynamic environment and the development of a low-cost USV platform from readily available materials. The concepts of USV navigation and development were explored in Chapter 2. The theoretical concepts of USV navigation and what is needed for the obstacle avoidance were explained in Chapter 3. This theory was then implemented and tested in a simulation environment based on the Virtual RobotX and USV Simulator. The second objective, the development of a USV platform, was described in Chapter 5. In this chapter, the building process of a USV and its configuration is fully explained. Finally, Chapter 6 described the tests that were performed with the USV and its sensors.

The development of a navigation method in a dynamic environment is done with Dubins path as a path generation method and the obstacles are detected, tracked and their trajectory is predicted with a 2D LiDAR. The implementation of this method in the simulation showed that the USV is capable of avoiding static as well as low-velocity obstacles. The static avoidance was tested with a simple obstacle on its path and in a dense environment where it is constantly surrounded by obstacles. In these situations, the USV was able to keep a safe distance of the obstacles, while navigating towards its waypoint. The avoidance of dynamic objects was tested with the three most frequent vessel-to-vessel interactions at sea. These are avoiding a head-on vessel on a reciprocal course, overtaking a head-on vessel and avoiding a crossing vessel. In all three situations, the USV could stay clear of the obstacle and sail to the waypoint.

The second objective, the construction of a low-cost USV platform was done with on the one hand hobby materials for the hardware construction, such as the boogie board and watertight box, and, on the other hand, easily accessible and affordable components, like the T200 thrusters and Cube Orange autopilot. The USV was then configured with open source software, such that the accessibility of the hardware components was extended to the software. The constructed USV was configured for remote control and tested with its sensors. The GNSS receiver that was used was only accurate up to $2.5m$, which caused the detected obstacles and USV to jump in the map. These jumps created a non-reliable detecting and positioning of the obstacles with respect to the USV. A low-cost USV was thus constructed that can be controlled remotely and on which the sensors are fully configured; however, the accuracy was not good enough to also test the simulated navigation method on the real-life USV.

Further work

Finally, some ideas and recommendations for further work to mitigate the shortcomings of this thesis are discussed.

Simulation

- The navigation of the USV was only tested once in an environment with wind and this at low wind speeds. Therefore, this only gives information about the behaviour of the navigation method in sheltered waters, for example, in a harbour or a lake. The navigation method should be further tested to be able to know how the USV navigates in a higher sea state, hence with waves and

currents, and with wind. These waves, currents, and wind can not just be added to the simulation because the USV must be able to counteract its displacement caused by these phenomena. The wind speed and direction can be measured with a wind sensor and from these measurements the force on the USV, hence also the displacement caused by the wind, can be calculated. The current can be measured with a current sensor and in the same way as for the wind, the force can be calculated. The trickiest part are the waves, as these cause a horizontal as well as a vertical displacement. With vertical displacements, the measurements of the 2D LiDAR will be disturbed and only measurements of the LiDAR for which the IMU is level or almost level to the horizon may be taken into account, as an upward angle of the USV can give an error in range. When the environmental factors are taken into account, there are thus many changes that need to be made to still be able to receive correct and useful information from the sensors.

- The USV was tested in dynamic environments with one obstacle that interacted with the USV. No tests were carried out for navigation with multiple obstacles. Just as obstacle avoidance was tested in a crowded static environment, the navigation method could be extended to a crowded dynamic environment to test how it handles this.

- Although the USV is able to stay clear of dynamic obstacles, it has no regard for the regulations defined by the International Maritime Organization in *the Convention on the International Regulations for Preventing Collisions at Sea (ColRegs)* [77]. The implementation of the ColRegs is certainly interesting for larger seagoing USVs, as these certainly need to follow the regulations at sea, but the implementation can first be done and tested on small-scale USVs as the one built in this thesis.

USV development

- To be able to test the navigation method thoroughly in real-life, the accuracy of the GNSS receiver needs to be much better. This could be achieved with RTK. One could also look to Simultaneous Localisation And Mapping (SLAM) to position the USV and obstacles in real-life and to achieve better results. SLAM can be implemented using sensors that are already present on the USV. One can use the 2D LiDAR with the *hector_mapping* package [75]. Another possibility is through the use of the RealSense D435i camera. This D435i is equipped with two RGB cameras to provide depth perception. The frames from the cameras together with ORB-SLAM2 [76] supply the USV with real-time localisation of itself and the environment around it. ORB-SLAM can compute the trajectory and position of the camera and thus also of the USV with monocular, stereo and RGB-D cameras. These methods can offer a solution to the GNSS receiver problems or provide navigation in GNSS-denied environments.

- The implementation of a Proportional-Integral-Derivative (PID) controller for full autonomous navigation is missing in this thesis. The right commands, which were discussed in Chapter 6 for the thruster can then be given from the onboard computer with the aid of the PID controller. This combined with RTK for the GNSS receiver would give an accurate positioning and heading of the USV. With these two additions, full autonomy can be achieved. The USV will then be able to navigate to the waypoints as was done in the simulation.

A. Code and used packages

A.1. Code

All the code for the results and the packages used can be found on GitHub via the following URL: `https://github.com/LouisSomers/Autonomous-navigation`. On the GitHub page, the installation of the different packages and plugins for the simulation and sensors is explained in the *README.md*. The following packages were used for this thesis:

Simulation

The following packages in addition to own code were used in the simulation. The use of these packages is explained in Chapters 3 and 4.

- USV simulator [46]: The simulator that is used.
- Complete coverage [8]: This was used as a basis to begin the implementation of the navigation method.
- Costmap 2D [31]: The package that maps the sensor data to an occupancy grid.
- Cartographer [78]: The package used for the occupancy grid used for navigation, which is explained in Chapter 3.
- DATMO [10]: The package that enables the detection and tracking of moving obstacles.
- Dynamic model plugin [48]: The method such that the spawned obstacles are dynamic.

Real-life

Supplementary to the shared packages with the simulation, such as the DATMO, other packages needed to be installed to allow the sensors to interact with ROS, so that everything could be combined. These extra packages are given below. Each package that is used is discussed in Chapter 5.

- RPLIDAR SDK [70]: The package that is needed to use the RPLIDAR S1 with ROS.
- RealSense SDK [71]: This SDK is used to extract information of the IMU in ROS.
- MAVROS [69]: The package that enables MAVLink communication between the Cube and NUC, so that data can be transmitted and read using ROS.
- MAVLink (MAVProxy) [56]: The package that enables communication from and to the Cube from a terminal.
- Trisonica SDK [64]: The SDK that can be used to activate the wind sensor.

A.2. Running the simulation

The simulation can be run with the following commands, each in a separate terminal window.

```
$ roslaunch sim_otter otter.launch
$ roslaunch sim_guidance guidance.launch
$ roslaunch usv_navigation usv_navigation.launch
```

Bibliography

[1] Glenn Wright. *Unmanned and Autonomous Ships: An Overview of MASS*. Routledge, 03 2020.

[2] Zhixiang Liu, Youmin Zhang, Xiang Yu, and Chi Yuan. Unmanned surface vehicles: An overview of developments and challenges. *Annual Reviews in Control*, 41, 05 2016.

[3] Mariusz Specht, Cezary Specht, Andrzej Stateczny, Paweł Burdziakowski, Paweł Dąbrowski, and Oktawia Lewicka. Study on the Positioning Accuracy of the GNSS/INS System Supported by the RTK Receiver for Railway Measurements. *Energies*, 15:4094, 06 2022.

[4] Yan Peng, Dong Qu, Yuxuan Zhong, Shaorong Xie, Jun Luo, and Jason Gu. The obstacle detection and obstacle avoidance algorithm based on 2-D lidar. *2015 IEEE International Conference on Information and Automation*, 08 2015.

[5] Zhi Li. Survival Strategies for Unmanned Surface Vehicles in Harsh Ocean Environments. PhD thesis, Memorial University of Newfoundland, 05 2018.

[6] Yi Guo and Zhihua Qu. Coverage control for a mobile robot patrolling a dynamic and uncertain environment. In *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No.04EX788)*, volume 6, pages 4899 – 4903 Vol.6, 07 2004.

[7] Bowen Xing, Xiao Wang, Liu Yang, Zhenchong Liu, and Qingyun Wu. An Algorithm of Complete Coverage Path Planning for Unmanned Surface Vehicle Based on Reinforcement Learning. *Journal of Marine Science and Engineering*, 11:645, 03 2023.

[8] Jan Henrik Lenes. Autonomous online path planning and path-following control for complete coverage maneuvering of a USV. Master's thesis, Norwegian University of Science and Technology, 07 2019.

[9] Xinyu Liu, Yun Li, Jing Zhang, Jian Zheng, and Chunxi Yang. Self-Adaptive Dynamic Obstacle Avoidance and Path Planning for USV Under Complex Maritime Environment. *IEEE Access*, 7:114945–114954, 2019.

[10] Kostas Konstantinidis. Development of a Detection and Tracking of Moving Vehicles system for 2D LIDAR sensors. Master's thesis, Delft University of Technology, 02 2020.

[11] Michael Thuy and Fernando Puente Leon. Non-linear, shape independent object tracking based on 2D lidar data. In *2009 IEEE Intelligent Vehicles Symposium*, pages 532–537, 2009.

[12] Michal Mihálik, Marian Hruboš, Peter Vestenický, Peter Holečko, Dušan Nemec, Branislav Malobický, and Ján Mihálik. A Method for Detecting Dynamic Objects Using 2D LiDAR Based on Scan Matching. *Applied Sciences*, 12:5641, 01 2022.

[13] Ji-wung Choi, Renwick Curry, and Gabriel Elkaim. Path Planning Based on Bézier Curve for Autonomous Ground Vehicles. In *Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pages 158–166, 2008.

[14] Joshué Pérez, Jorge Godoy, Jorge Villagrá, and Enrique Onieva. Trajectory generator for autonomous vehicles in urban environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 409–414, 05 2013.

[15] Sambhunath Biswas and Brian Lovell. *B-Splines and Its Applications*, pages 109–131. Springer London, London, 2008.

[16] Gauthier Rousseau, Cristina Stoica Maniu, Sihem Tebbani, Mathieu Babel, and Nicolas Martin. Minimum-time B-spline trajectories with corridor constraints. Application to cinematographic quadrotor flight plans. *Control Engineering Practice*, 89:190–203, 2019.

[17] Ngoc-Huy Tran, Quang-Ha Pham, Ji-Hyeong Lee, and Hyeung-Sik Choi. VIAM-USV2000: An Unmanned Surface Vessel with Novel Autonomous Capabilities in Confined Riverine Environments. *Machines*, 9:133, 07 2021.

[18] Yalçin Kaya. Markov-Dubins path via optimal control theory. *Computational Optimization and Applications*, 68:719–747, 12 2017.

[19] Satyanarayana Manyam, David Casbeer, Alexander Von Moll, and Zachariah Fuchs. Shortest Dubins path to a circle. *AIAA Scitech 2019 Forum*, 01 2019.

[20] Thierry Fraichard and Alexis Scheuer. From Reeds and Shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, 12 2004.

[21] Maritime Robotics. Otter USV. `https://www.maritimerobotics.com/otter`.

[22] Joga Dharma Setiawan, Muhammad Aldi Septiawan, Mochammad Ariyanto, Wahyu Caesarendra, M. Munadi, Sabri Alimi, and Maciej Sulowicz. Development and Performance Measurement of an Affordable Unmanned Surface Vehicle (USV). *Automation*, 3:27–46, 03 2022.

[23] Blue Robotics. T200 Thruster for ROVs, AUVs, and marine robotics. `https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/`.

[24] ArduPilot Dev Team. Mission Planner Home — Mission Planner documentation. `https://ardupilot.org/planner/index.html`, 2023.

[25] Wonse Jo, Yuta Hoashi, Lizbeth Leonor Paredes Aguilar, Mauricio Postigo-Malaga, José M. Garcia-Bravo, and Byung-Cheol Min. A low-cost and small USV platform for water quality monitoring. *HardwareX*, 6:e00076, 10 2019.

[26] Barak Or. What is IMU? `https://towardsdatascience.com/what-is-imu-9565e55b44c`, 08 2021.

[27] Industrial Inspection & analysis. what-are-the-6-degrees-of-freedom. `https://industrial-ia.com/what-are-the-6-degrees-of-freedom-6dof-explained/`, 10 2022.

[28] National Oceanic US Department of Commerce and Atmospheric Administration. What is LIDAR? `https://oceanservice.noaa.gov/facts/lidar.html`, 2019.

[29] Bill Schweber. LIDAR and Time of Flight, Part 2: Operation. `https://www.microcontrollertips.com/lidar-and-time-of-flight-part-2-operation/`, 12 2019.

[30] National Coordination Office for Space-Based Positioning Navigation and Timing. GPS.gov: Other Global Navigation Satellite Systems (GNSS). `https://www.gps.gov/systems/gnss/`, 2020.

[31] ROS contributors. costmap_2d - ROS Wiki. `http://wiki.ros.org/costmap_2d`, 2013.

[32] Abdul Hadi Abd Rahman, Hairi Zamzuri, Saiful Amri Mazlan, and Mohd Azizi Abdul Rahman. Model-Based Detection and Tracking of Single Moving Object Using Laser Range Finder. *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, 01 2014.

[33] Aurelio Ponz, Cesar H. Rodríguez-Garavito, Fernando García, Philip Lenz, Christoph Stiller, and Jose M. Armingol. Automatic Obstacle Classification using Laser and Camera Fusion. In *Proceedings of the 1st International Conference on Vehicle Technology and Intelligent Transport Systems - Volume 1: VEHITS,*, pages 19–24. INSTICC, SciTePress, 2015.

[34] Saishruthi Swaminathan. Linear Regression — Detailed View. `https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86`, 02 2018.

[35] Israel Lugo-Cárdenas, Gerardo Flores, Sergio Salazar, and Rogelio Lozano. Dubins path generation for a fixed wing UAV. In *International Conference on Unmanned Aircraft Systems (ICUAS 2014)*, pages 339–346, Orlando, FL , United States, May 2014.

[36] Anastasios Lekkas and Thor Fossen. Minimization of cross-track and along-track errors for path tracking of marine underactuated vehicles. In *2014 European Control Conference (ECC)*, pages 3004–3010, 2014.

[37] Stephane Mondoloni, Sip Swierstra, and Mike Paglione. Assessing trajectory prediction performance - metrics definition. In *24th Digital Avionics Systems Conference*, volume 1, pages 3.C.1–31, 2005.

[38] Anastasios Lekkas. Guidance and Path-Planning Systems for Autonomous Vehicles. PhD thesis, Norwegian University of Science and Technology, 04 2014.

[39] Open Robotics. ROS.org | Powering the world's robots. https://www.ros.org/, 2020.

[40] Brian Bingham, Carlos Aguero, Michael McCarrin, Joseph Klamo, Joshua Malia, Kevin Allen, Tyler Lum, Marshall Rawson, and Rumman Waqar. Toward Maritime Robotic Simulation in Gazebo. In *Proceedings of MTS/IEEE OCEANS Conference*, Seattle, WA, 10 2019.

[41] Robonation. RobotX. https://robonation.org/programs/robotx/, 2023.

[42] Sean Fish, Manuel Roglan, Douglas Chin, Jeffrey Pattison, and Jorge Solano. Virtuoso, A New Architecture for RobotX 2022. https://robonation.org/app/uploads/sites/2/2022/10/RX22_TDR_Georgia-Tech-V2.pdf, 2022.

[43] Tanmay Vilas Samak, Chinmay Vilas Samak, Chern Peng Lee, and Ming Xie. SINGABOAT-VRX | Virtual RobotX (VRX) Competition. https://github.com/Tinker-Twins/SINGABOAT-VRX, 11 2022.

[44] Clearpath Robotics. HERON unmanned surface vessel tm built for rapid prototyping. https://levelfivesupplies.com/wp-content/uploads/2020/08/Clearpath-Heron-USV-Brochure.pdf.

[45] ROS Contributors. Heron USV Simulation. https://github.com/heron/heron_simulator, 05 2023.

[46] Jan Henrik Lenes. usv_simulator. https://github.com/jhlenes/usv_simulator, 02 2022.

[47] Yung-Yu Chen, Ming-Zhen Ellis-Tiew, Wei-Chun Chen, and Chong-Ze Wang. Fuzzy Risk Evaluation and Collision Avoidance Control of Unmanned Surface Vessels. *Applied Sciences*, 11:6338, 07 2021.

[48] ROS Contributors. MRS resources for Gazebo. https://github.com/ctu-mrs/mrs_gazebo_common_resources, 04 2023.

[49] DFCscience. Maker Boat Basic Quick Start Guide. https://squirm.tech/maker-boat-basic-quick-start/, 11 2016.

[50] Blue Robotics Community. Flow rate/s for T200? https://discuss.bluerobotics.com/t/flow-rate-s-for-t200/4117/6, 01 2019.

[51] Benjamin Shamah, Dimitrios Apostolopoulos, Michael D. Wagner, and William L. Whittaker. Effect of Tire Design and Steering Mode on Robotic Mobility in Barren Terrain. In *Proceedings of 2nd International Conference on Field and Service Robotics (FSR '99)*, pages 287 – 292, 08 1999.

[52] SLS StefansLipoShop. Lithium batteries and accessories for RC model making. https://www.stefansliposhop.de/index.php?language=en, 2021.

[53] CubePilot. The Cube Module Overview - CubePilot. https://docs.cubepilot.org/user-guides/autopilot/the-cube-module-overview, 2014.

[54] Edgar A. Niit and Willie J. Smit. Integration of model reference adaptive control (MRAC) with PX4 firmware for quadcopters. In *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pages 1–6, 2017.

[55] Intel Corporation. Intel® NUC Mini PCs - Next Unit of Computing | Intel. https://www.intel.com/content/www/us/en/products/details/nuc/mini-pcs/products.html, 2021.

[56] Anis Koubaa, Azza Allouch, Maram Alajlan, Yasir Javed, Abdelfettah Belghith, and Mohamed Khalgui. Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey. *IEEE Access*, 7:87658–87680, 2019.

[57] NETGEAR. Nighthawk M1 4G LTE Mobile Router - MR1100. https://www.netgear.com/be/home/mobile-wifi/hotspots/mr1100/, 2023.

[58] CubePilot. Herelink Overview. https://docs.cubepilot.org/user-guides/herelink/herelink-overview.

[59] LTD SHANGHAI SLAMTEC CO. RPLIDAR S1. https://www.slamtec.com/en/Support#rplidar-s1, 2022.

[60] Intel Corporation. Intel RealSense D435i. https://www.intelrealsense.com/depth-camera-d435i/.

[61] CubePilot. Here 3 Manual. https://docs.cubepilot.org/user-guides/here-3/here-3-manual, 2020.

[62] InvenSense. ICM-20948. https://invensense.tdk.com/products/motion-tracking/9-axis/icm-20948/, 2023.

[63] Anemoment LLC. LI-550F TriSonica Mini Wind & Weather Sensor. https://anemoment.com/shop/sensors/trisonica-mini-wind-and-weather-sensor/, 2022.

[64] Floris Van Breugel. ROS node for trisonica mini. https://github.com/vanbreugel-lab/trisonica_ros, 02 2023.

[65] QGroundControl – Drone Control. QGC - QGroundControl - Drone Control. http://qgroundcontrol.com/, 2015.

[66] Antenore Gatta. Remmina - The Gtk remote desktop client. https://antenore.simbiosi.org/remmina-project/, 06 2018.

[67] RealVNC® Ltd. All you need to know about VNC remote access technology. https://discover.realvnc.com/what-is-vnc-remote-access-technology, 2022.

[68] Ardupilot contributors. ArduPilot Wiki Sources. https://github.com/ArduPilot/ardupilot_wiki/blob/master/mavproxy/source/docs/getting_started/download_and_installation.rst, 04 2023.

[69] ROS contributors. MAVROS. https://github.com/mavlink/mavros, 04 2023.

[70] ROS contributors. RPLIDAR driver. http://wiki.ros.org/rplidar, 08 2019.

[71] ROS contributors. RealSense. http://wiki.ros.org/RealSense, 09 2021.

[72] Michael Coyle. Calculating your own GPS accuracy. https://blog.oplopanax.ca/2012/11/calculating-gps-accuracy/, 11 2012.

[73] Mokhamad Nur Cahyadi, Tahiyatul Asfihani, Ronny Mardiyanto, and Risa Erfianti. Performance of GPS and IMU sensor fusion using unscented Kalman filter for precise i-Boat navigation in infinite wide waters. *Geodesy and Geodynamics*, 2022.

[74] Francois Caron, Emmanuel Duflos, Denis Pomorski, and Philippe Vanheeghe. GPS/IMU data fusion using multisensor Kalman filtering: introduction of contextual aspects. *Information Fusion*, 7:221–230, 06 2006.

[75] ROS Contributors. hector_mapping - ROS Wiki. http://wiki.ros.org/hector_mapping, 03 2021.

[76] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 06 2017.

[77] International Maritime Organization. Convention on the International Regulations for Preventing Collisions at Sea, 10 1972.

[78] ROS Contributors. Cartographer - ROS Wiki. http://wiki.ros.org/cartographer, 04 2016.