UNIVERSITY OF AGDER

Modelling of a Braitenberg inspired guidance system for an Autonomous surface vessel (ASV)

Tore Engebretsen

Supervisors
Filippo Sanfilippo and Morten Kjeld Ebbesen

University of Agder, 2020
Faculty of Engineering and Science
Department of Engineering Sciences

Acknowledgements

I would first like to thank my thesis advisers Associate Professor Filippo Sanfilippo and Associate Professor Morten Kjeld Ebbesen at University of Agder. Even through a time period out of the ordinary, they were always available for guidance and showing me encouragement. They allowed this paper to be my own work, but steered me in the right direction whenever they thought I needed it.

I would also like to thank Morten Rudolfsen and Andreas Klausen at University of Agder for supporting me through the challenges of this thesis.

Finally, I must express my profound gratitude to my parents, brothers and nearest friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Abstract

Aquatic unmanned robotic systems have gained popularity due to their abilities to perform a wide range of operations at low cost and no risk to human lives. Therefore, navigation systems are being developed to increase the controllability and raise the level of autonomy of such vehicles. The research presented in this thesis covers investigations and development of unmanned vehicles (UVs) and their navigation systems. In addition, a control concept called the Braitenberg algorithm is explored as a basis for controlling UVs.

The Braitenberg algorithm is a concept based on a thought experiment that models the animal world in a minimalistic and constructive way, from simple reactive behaviours through the simplest vehicles, to the formation of concepts and generation of ideas. The concept is used on a large variety of experiments, wheeled robots and other land robots, but unknown to the world of boats at this point. The Braitenberg concept is therefore implemented as a control algorithm in a simulation for controlling an unmanned surface vehicle (USV) and navigating in sheltered waters with obstacles. The obstacles are randomly placed in the environment and fictional sensors on the vehicle have a fixed detection range. The algorithm gives control inputs so that the vehicle is attracted to a set of waypoints while avoiding obstacles along the way.

Using the Braitenberg algorithm, the surface vehicle is able to use GPS positioning data to navigate through a course defined by a set of waypoints while detecting and avoiding the obstacles in the environment.

The simulation shows that using the Braitenberg algorithm as a basis for an autonomous navigation system is a viable solution for making a USV able to track waypoints and avoid obstacles autonomously.

Nomenclature

$2D$	Two Dimensional
$3D$	Three Dimensional
A	A
AIS	Automatic Identification System
ASV	Autonomous Surface Vessel
$ATRA$	Autonomy and Technology Readiness Assessment
cmd	Command
$COLREG$	International Regulations for Preventing Collisions at Sea
$DDPG$	Deep Deterministic Policy Gradient
EC	Environmental Complexity
ES	External System
ESI	External System Independence
GNC	Guidance, Navigation and Control
$GNSS$	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
IMU	Inertia Motion Unit
$LiDAR$	Light Radar (Light Detection and Ranging)
MC	Mission Complexity
MNC	Memristive Neuromorphic Circuits
MPC	Model Predictive Control
ROS	Robot Operating System
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vehicle

Contents

Chapter 1

Introduction

1.1 Background

Around two-thirds of the earth is covered by oceans, but compared to land, not much of the ocean has been properly explored. Climate change, environmental abnormalities, personnel requirements, and national security concerns are all contributing to a strong demand from industrial, scientific, and military communities to develop new and innovative unmanned surface vehicles (USVs).

Despite the strong demand for methods of exploration, normally only semi-autonomous USVs have been used rather than fully-autonomous USVs. The reason for this is that fully-autonomous USVs still face numerous challenges. Limitation in autonomy arise due to the challenges in automated and reliable guidance, navigation and control (GNC) [4] functions for all different operating conditions. The vehicles must be able to operate in sophisticated and hazardous environments, and the risk of sensor, actuator and communication failures must be reduced to a minimum. Further development of fully-autonomous USVs is required in order to minimise both the need for human control and the effects to the effective, safe and reliable USVs operation due to human error.

USVs can be developed for a wide range of potential applications in a cost-effective way, such as scientific research, environmental missions [27], ocean resource exploration, rescue missions [18] and military uses [34].

Benefits of using USVs:

- USVs can perform longer and more hazardous missions than manned vehicles.
- Personnel safety is far greater since no crew is needed onboard during operation.
- Maintenance cost can be lowered where no operator equipment or safety equipment for crew are installed.
- The possibility of keeping low weight and a compact design of USVs give them enhanced manoeuvrability and deployability in shallow waters (riverine and coastal areas) where larger crafts cannot operate effectively.
- USVs have greater potential payload capacity and are able to perform monitoring and sampling at deeper depth compared to other vehicles as aircraft/UAVs and spacecraft.

Figure 1.1: The Otter, an USV from Maritime Robotics [23]. The USV is used for research and bathymetric measurements.

With many advantages of small USVs, the motivation to develop small scale and cost effective autonomous systems is high. The development of such systems could include doing more research on self-adapting systems, machine learning, artificial intelligence and of course combining several of these creating sophisticated systems. One step towards a self-adapting intelligent control system can be to utilise a control theory called the Braitenberg algorithm.

The Braitenberg algorithm is a concept based on a thought experiment done by the Italian-Austrian cyberneticist Valentino Braitenberg. The experiment models the world of animals in a minimalistic and constructive way. It is inspired from looking at behaviour of insects and animals around a source of food or a source of light. They use sensor organs to detect the smell of food or the intensity of light and are drawn to the source. In the same way they can be repelled from poison or the light source.

This concept can further inspire the creation of self adapting vehicles from simple reactive behaviours through the simplest of vehicles, to the creation of concepts and ideas for more sophisticated systems. For the simplest vehicles, the motion of the vehicle is directly controlled by input signals from sensors such as light sensitive photo cells. Although the control and the vehicle can be simple, the resulting behaviour may seem to be complex or even intelligent [3]. A vehicle adopting this control theory might able to navigate itself through different unknown environments, seeking out the source of control signal and avoiding different obstacles.

The Braitenberg concept can from an engineer point of view, be seen as and implemented in a vehicle with propulsion actuators and sensors that can detect the strength of a signal source. An example of a Braitenberg vehicle type called 2a / 2b is shown in Figure 1.2.

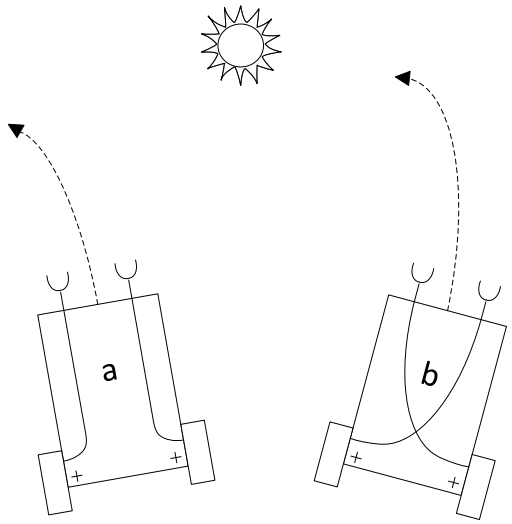


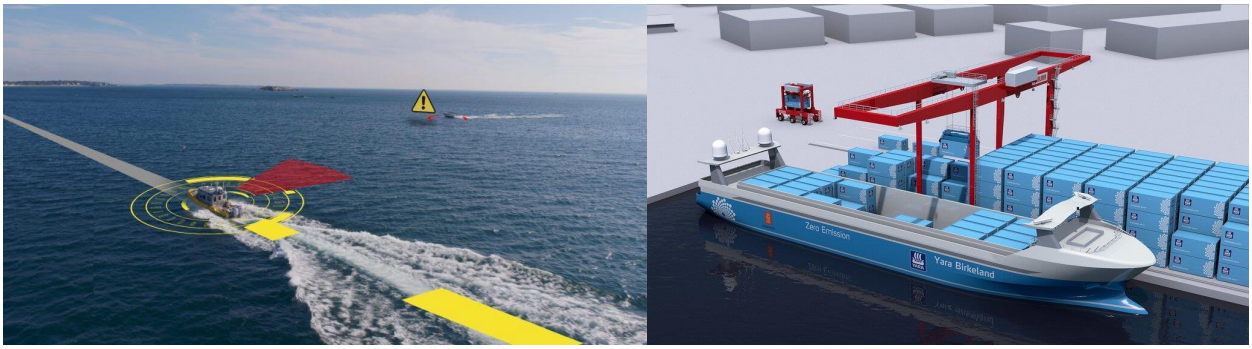
Figure 1.2: Simple Braitenberg vehicle

The vehicle is a two wheeled driven robot with two photo cell sensors. Version (a) gives more power to the right wheel if the sensor on the right is closer to the light source, making the robot drive away from the source. Depending on the application, the connection between the sensors and the wheels can be switched as to have the robot deflect from the light (a) or be attracted to it (b). By moving the light source to a desired position, the Braitenberg algorithm can be used as a navigation system.

1.2 Motivation

There are many reasons for developing fully autonomous vehicles, and the different approaches to the challenge is just as plentiful. Enabling USVs to work in any unstructured or unpredictable environment without human supervision is crucial. A key feature is to develop a navigation system that is self adaptive. A robust self adapting navigation system should be able to be deployed in many different environments and situations, and still be able to perform its prioritised tasks.

There are several existing products and projects that are closing in on achieving fully autonomous surface vessels. Examples are the navigation system from Sea-Machines, which integrates with existing vessel systems and onboard sensors to manage pre-planned and dynamically charted missions [30]. Another project currently in development is the autonomous container ship Yara Birkeland developed together with Kongsberg [35]. When deployed, this ship will autonomously transport fertiliser for Yara in Norway. In addition, it is a zero-emission vessel. Yara Birkeland aims to start autonomous operation this year 2020 [36].



a b

Figure 1.3: (a) A USV with control system delivered by Sea-Machines. (b) 3D model of Yara Birkeland at port.

Although development on autonomous systems are progressing, the systems of Sea-Machines and Kongsberg are applied to large vessels. These vessels have the possibility to carry large sensor packages and great computational power which greatly improves the possible autonomous capabilities. Developing an autonomous navigation system for smaller vessels gives a different set of challenges. Vessels of a size that one person can carry, is not able to house the same amount of sensors or powerful computers. These vessels will most likely be powered by relatively small battery packs and computational power and operation time is limited.

Developing a navigation system that is less dependent on advanced and complex sensor systems, that will require less computational power, will give great benefits to small size USVs. Such a low demanding navigation system will let the USV prioritise tasks other than navigation. Vessels intended for research or surveillance could extend their operational time before needing a recharge.

This is where a bio-inspired Braitenberg algorithm might be able to contribute to a low demanding autonomous navigation system. The concept of Braitenberg describes how the complex behaviour of animals and insects can be described in a simplistic way. If a navigation system with complex behaviour can be solved using simple methods, then the computational power of a robot can be used for other purposes. In addition, through the literature study done for this thesis, the Braitenberg algorithm has not yet been used as a basis for navigation system of USVs. Only for land based vehicles like wheeled robots and snakes. Therefore, exploring the use of the Braitenberg algorithm as a basis for navigation system of USVs is very interesting. The Braitenberg algorithm can contribute to the development of creating fully autonomous USVs.

1.3 Objectives

The main objective of this thesis is to explore the current state of autonomous surface vehicles and improve the level of autonomy by using the Braitenberg algorithm as a means of navigation. The navigation system will be purposely made for smaller size USVs that will navigate through an unknown environment, detect and avoid obstacles and shorelines. An USV developed in previous projects will be used as simulation model and physical robot, see section 2.3.

The Braitenberg algorithm will be implemented in a simulation following these steps:

- Develop a digital robot simulation model that can use GPS data to track a waypoint and navigate to the waypoint. In addition, use the same model to have the robot be repelled from a point.
- Develop a program that can autonomously track a set of waypoints navigating through the environment and avoid obstacles as they are detected along the track.
- Combine human control input with Braitenberg control to allow detours and extra obstacle avoidance if necessary.

As a result of the objectives, the aim is to raise the level of autonomy. An autonomy classification system called ATRA is used to define the level of autonomy during the different stages of robot development. See Table 1.1.

Table 1.1: The ATRA framework [11, 17], applied to aquatic surface robots with the different levels of external system independence (ESI), of environmental complexity (EC) and of mission complexity (MC). (ES) refers to External System.

Level	Description	Guidance	Navigation	Control	ESI	EC	MC
9	Fully autonomous	Human-level decision-making, accomplishment of most missions without any intervention from ES (100% ESI).	Human-like navigation capabilities. Situational wareness in extremely complex environments and situations.	Same or better control performance as human controlled vessels in the same situation and conditions.	100%	Extreme Level	Highest Level
8	Full mission planning	High-level decision making. Evaluation and optimisation of mission performance.	Higher level entities and properties are derived from the environment perception according to the desired task to be performed.	Same as previous levels.	High Level	Difficult Level	High Level
7	Dynamic global planning	Same as Level 6 but planning in a dynamic environment.	Same as Level 6 but mapping in a dynamic environment.	Same as previous levels.			
6	Global planning	Goal waypoint provided by ES. Global path planning determines optimal path to goal.	Global map includes properties of the environment. Localisation of aquatic surface robot relative to map.	Same as previous levels.			
5	Local planning with environment awareness	Same as Level 4 but local motion planner also takes into account properties of the environment (flow speed, resistance, etc.).	Same as Level 4 but local map also includes properties of the environment (flow speed, resistance, etc.).	Same as previous levels.	Mid Level	Moderate Level	Mid Level
4	Navigation from A to B	Drive from start point A and stop at point B using sensors to collect and react to the feedback signals.	Local mapping with geometrical representation of immediate surroundings. Localisation of aquatic surface robot relative to map.	Local adaptive control to compensate for possible deviations between map and actual environment.			
3	Reactive control	Motion planner reacts to sensor input feedback and detects if aquatic surface robot is jammed in environment.	Aquatic surface robot can detect contacts between its own body and the environment. No mapping.	Local adaptation to resolve jammed situations and/or local surface adaptation.	Low Level	Simple Level	Low Level
2	Pre-planned motion	Pre-programmed motion patterns.	Same as Level 1.	Automatic control to follow specified motion pattern. No adaptation.			
1	Remote control	All guidance functions are performed by external systems (mainly human operator).	Sensors may be adopted, but all data is processed and analysed by a remote ES. No mapping. No localisation.	All control commands are given by a remote ES (mainly human operator).	0%	Lowest Level	Lowest Level

The target is to add features to a USV in order to reach level 7. However, regarding level 6, the global path planning and mission planning will be added through manually setting coordinates for waypoints. In addition, the robot might not choose to drive the optimal path as the robot is entirely dependent on the sensor data in an unknown environment for navigation.

1.4 Assumptions and Prerequisites

When deployed, small USVs are exposed to many environmental effects and factors. Many of which are difficult to model. Therefore, the following boundaries are set:

Vessel:

The navigation system will be designed for a prototype robot of size approximately 40x50cm, driven by two water jets and have a very shallow hull construction. The construction is completely water proof, but not intended for subsurface operation. The robot was constructed in a previous project and this thesis is a continuation of that project. More information on the robot can be found in Section 2.3 Review of previous projects.

Environment:

- The robot is designed to operate in sheltered waters with calm seas. No noticeable waves.
- Detectable obstacles are above sea level. Any subsurface obstacles like reefs, rocks and logs will not be detectable.
- Water currents are neglected.
- Wind force can be applied if time allows it, constant force and direction.

Sensors:

- Constant GPS signal, always available. Accuracy is considered ideal.
- Robot is equipped with sensors that can detect obstacles at a distance of 5 meters and is able to place these in the global coordinate system at the correct position. The exact sensors are not defined. However, LiDAR or acoustic sensors are viable options.

1.5 Disposition

Chapter 1 gives an introduction to some commercially available unmanned surface vehicles and their uses. It also gives the objectives of this thesis and sets the boundary limits of expected new research and development.

Chapter 2 explains the work previously done on this project, provides an overview of current state of USV systems and describes the development of navigation systems over the last years.

Chapter 3 describes the implementation of the Braitenberg algorithm and how the simulation is set up.

Chapter 4 describes how the simulation environment is set up and how the controller code is developed.

Chapter 5 contains a selection of important and interesting results obtained from the simulation.

Chapter 6 highlights challenges during setup of the simulation and the development of a fully autonomous USV.

Chapter 7 concludes the thesis and summarises improvements and challenges learned from the project.

Chapter 2

Related works

2.1 State of the art

2.1.1 Mechanical design

USVs that are currently being used and developed need to fulfil a set of minimum requirements in order to become fully autonomous. They need a solid and stable hull design, a propulsion system, a set of sensors and a GNC system [4]. In addition, communications to a ground station allows for configuration of mission and manual control if needed.

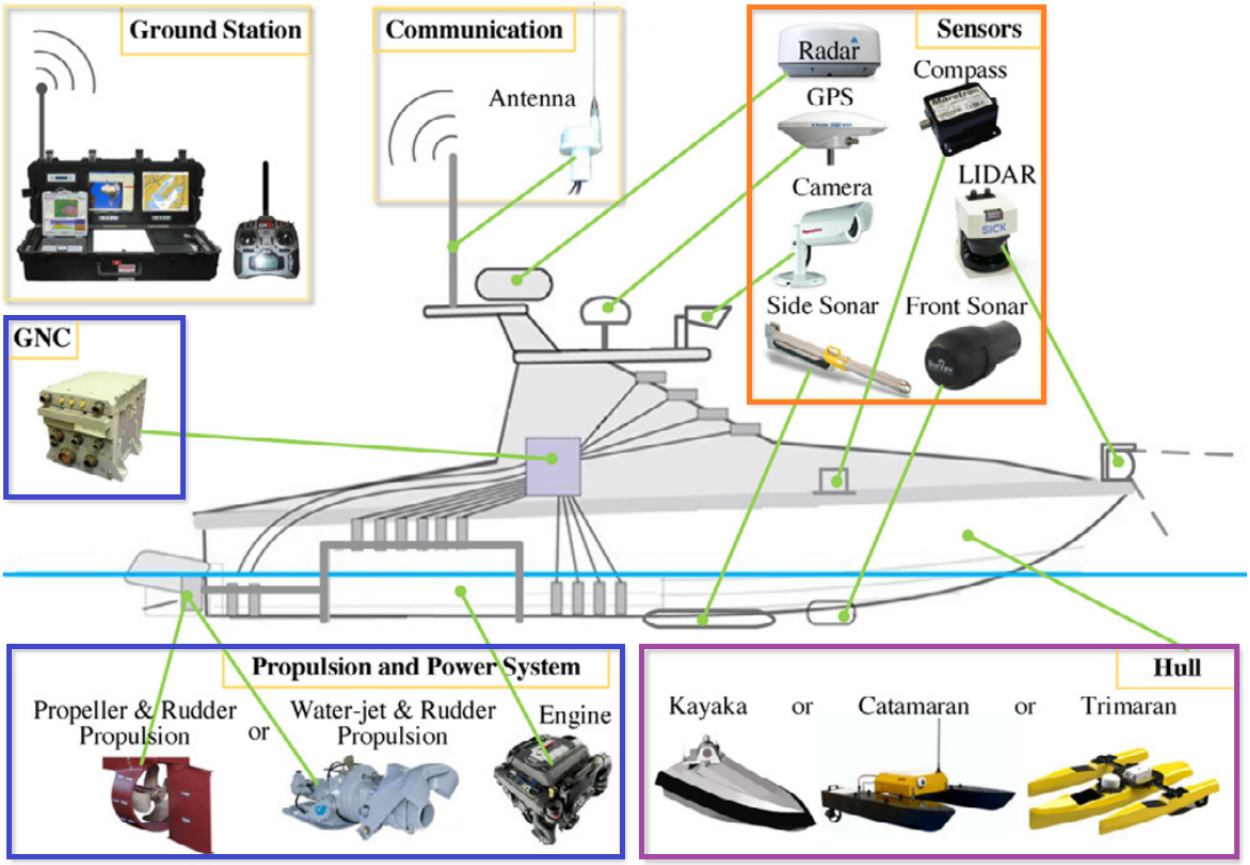


Figure 2.1: Basics of a USV. Schematics showing the main components of a common USV, most of which are necessary to operate autonomously.

As seen in Figure 2.1, the GNC module for USV is the brain of the vessel, utilising the sensor data, guides the vessel according to its guidance algorithms and controls the propulsion to move the vessel to its desired location.

The design of the hull reflects the intended use of the USVs. The kayak design is a robust construction and if needed can be designed to be self righting. It also gives room to connect more internal components close together. However, the kayak design is more exposed to roll motion. This could interfere with measurements of bathymetric data of the seabed [4]. A multi hull design can in this case be more stable and more suitable for bathymetric measurements [27]. All three hull designs are currently being used in the development of any size of USV. From smaller USVs like the single hull GeoSwath 4R [16] and the catamaran Otter [23], to large vessels like the single hull Yara Birkeland [35] and the trimaran Sea Hunter [34].

Mainly two types of propulsion are dominating the USV marked, propeller with rudder and water jet. The propeller and rudder design is a low cost solution with few components and easy to implement in a hull. Making the hole for the propeller shaft water proof can be done with a water proof bearing. The drawback with an open propeller is that seaweed and other contamination can get tangled up in the propeller and rudder, thus stopping the engine or prevent turning. Water jet propulsion on the other hand is a more complex construction [1]. They normally have three ports that must be water proof and tolerances around the propeller must be adhered. Turning can be done using two different methods. One is by directing the water jet with a nozzle and is often combined with a revering mechanism. The other method is by the use of differential thrust if the USV is equipped with two jets. The water inlet to the propeller is normally protected by grating. This makes the water jet a reliable propulsion system running less risk of contamination of the propeller. It is also safer for humans during a search and rescue mission.

A paper written early in year 2020 includes a review of several small size USVs currently available and operational [28]. The paper focuses on USVs of size up to 2m in length and mass up to 55kg. It describes physical design, what sensor packages are included and areas and objectives of typical operation. The reviewed USVs are shown in Figure 2.2 and a summary of their characteristics are shown in Table 2.1.

Figure 2.2: USV designs varying in size and capabilities.

Table 2.1: Characteristics of USVs in Figure 2.2. ("-" not given).

Figure	USV Name	Manufactorer	L [m]	W [m]	H [m]	Weight [kg]	Range [m]	Endurance [h]	Max Speed [knots]
a	Otter	Maritime Robotics	2,00	1,08	0,81	55	2500	20,0	5,5
b	SR-Surveyor M1.8	Sea Robotics	1,80	0,91	1,00	49	6200	5,5	4
c	Heron	Clearpath Robotics	1,35	0,98	0,32	28	-	2,5	3,3
d	GeoSwath 4R	Kongsberg Maritime	1,80	0,90	-	55	1500	6,0	6
e	SL20	Ocean Alpha	1,05	0,55	0,30	10	2000	2,0	10
f	Z-Boat 1800-RP	Teledyne Marine	1,80	1,00	1,10	38	1200	4,0	10

The same paper also states that commercial applications of these USVs provide evidence that most of the technology necessary for the development of USVs is mature and available, including sensors, communication and control principals [28]. High-speed with acceptable endurance could be achieved at relatively low cost. To the best of the researchers knowledge, a low-cost and open framework for aquatic surface robots is still missing. The paper also suggest a common platform for developers to use, so that future work and development of USVs can be accelerated through collaboration between teams and more easily compared based on the common platform.

2.1.2 Sensors

Sensor packages implemented in the design of smaller USVs can be divided into two groups, sensors used for navigation and sensors used for data sampling.

Sensors used for navigation commonly includes global navigation satellite system (GNSS) sensors. GNSS includes systems as the American developed system GPS and the Russian developed system GLONASS and more. Using a GNSS sensor that can connect to several of the satellite systems can greatly improve accuracy and stability of the positioning data. The Otter and the SR-Surveyor M1.8 have sensors that can utilise several of the satellite positioning systems [16, 25]. The positioning sensor and data will from hereon be called by GPS for simplicity reasons. Another common factor is the lack of a system to avoid obstacles while navigating. In fact, none of the above compared USV models include any sensor for detecting obstacles. The course must therefore be carefully planned by an operator beforehand to ensure that the USV will not collide. However, studies show a test being done on the Otter where low cost 2D LiDAR sensors are used to detect obstacles [14]. Unfortunately, the results are not yet satisfactory due to the functionality of the 2D LiDAR sensor. SR-Surveyor M1.8 is equipped with a LiDAR but it is only used to map the local environment at a side view and therefore not used for obstacle detection or avoidance. To the author's knowledge, a functional obstacle detection and avoidance system intended for a complex environment is yet to be developed.

The USVs are commonly equipped with different sensors used for data sampling. Bathymetrics, temperature, concentration of different substances in the water [22] to name a few. These type of sensors will not be further covered in this thesis.

2.1.3 Related control methods

The control features of the USVs currently in use can vary. The companies producing these vessels are often not very open about their approach and solution to navigation and autonomous features. Therefore, comparison can be a challenge. Information gathered from the manufacturers documentation can be seen in Table 2.2, though it might be incomplete as some information is not open to public.

Table 2.2: Currently implemented USV Control features compared to Table 1.1 of the level of autonomy according to the ATRA classification system.

Level of autonomy	Control Features	Otter	SL20	Z-Boat 1800-RP	GeoSwath 4R	AnSweR Bot	Heron	SR-Surveyor M1.8
9	Fully autonomous							
	Obstacle avoidance							
	Mission planning	x	x	x	x			
	Waypoint control	x	x					
	Course control	x						
	Heading control	x						
	Speed control	x		x		x		
0	Remote control	x		x	x	x	x	

Although being an open source product, Heron seem to either have a lack of developed features, or it is not openly documented. Otter on the other hand, has implemented mission planning and templates for missions to help with for example cover a complete area for scanning. Table 2.2 shows that none of the above compared models include obstacle avoidance nor can be defined as fully autonomous.

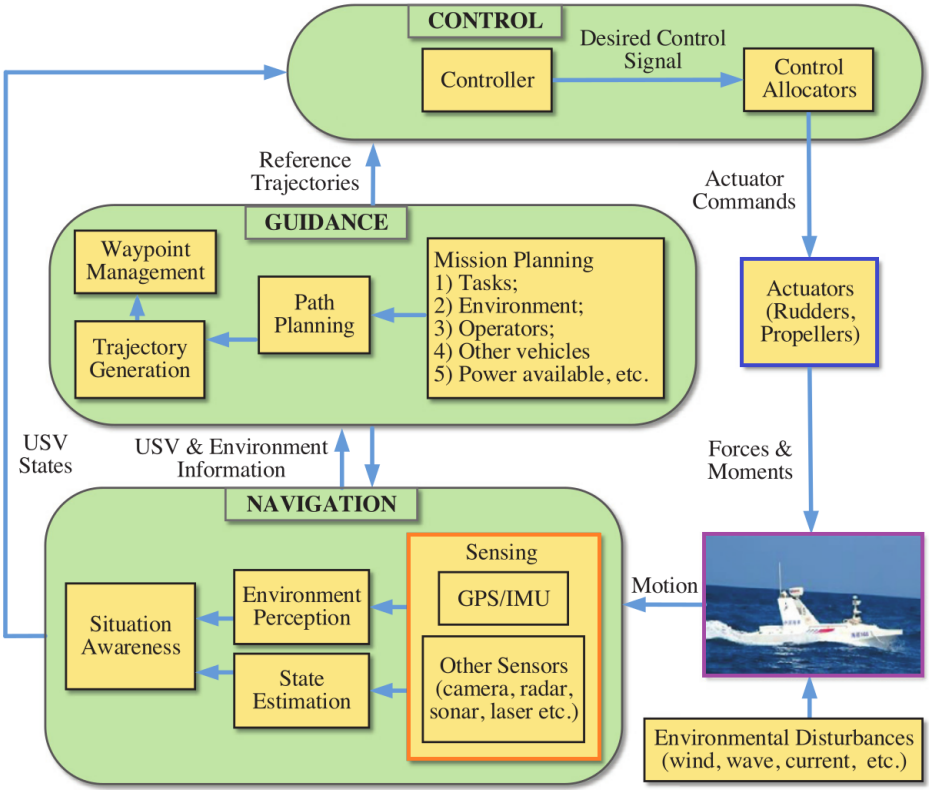


Figure 2.3: GNC schematics showing an overview of the flow of information in an autonomous navigation system [15].

Figure 2.3 describes a GNC model commonly adopted by the robotics community [15].

1. The Guidance system is designed to continuously generate and update trajectory commands to the control system. The commands are generated according to the information given by the navigation system, planned mission and the sensed environmental conditions.

2. The Navigation system identifies the USV's states (position, heading, velocity) and environment based on sensor data.

3. The Control system uses the provided information from the guidance system and navigation system to generate the proper motor control signal.

Recent research show development of different ways of increasing the level of autonomy of surface vessels. Interesting and necessary topics are control theory, path optimisation, obstacle and collision avoidance. Improvements in all these areas will greatly contribute to the fully autonomous USV.

An article on control theory explores the use of loop-shaping in a course-keeping feedback controller, increasing its robustness and stability [9]. A simulation of a course-keeping control of a motor vessel called "YUKUN" was carried out in multi-directional irregular waves and a smart USV simulation. The results show an improvement in course-keeping of up to 12% compared to a nonlinear feedback control method.

Another paper describes the challenges of path planning for an underactuated USV in unknown environments with obstacles [39]. A deep deterministic policy gradient (DDPG)-based path planning algorithm with powerful actor-critic architecture is proposed, aiming at the usability problem caused by the complicated control law of the traditional method. A reward system is specially designed for speed control, attitude correction and target approaching. Simulation show that the optimal path can be automatically generated under unknown environmental disturbance, and therefore giving the proposed DDPG-based path planning scheme effectiveness and meaning.

Obstacle avoidance and anti collision systems are these days widely discussed. Development is highly motivated by increasing demand for such systems, though none of the compared USV models in Figure 2.2 utilises them. This could indicate that the systems are not yet matured to a point where implementation on small USVs is feasible. It could also indicate that a complete working obstacle avoidance and anti collision system is too expensive for small USVs. On larger UAVs however, implementation of such systems seem more cost effective and are more common [30, 34].

Sea trials of an autonomous surface vessel (ASV) have been performed in the North Sea as part of an ASV Challenge posed by Deltares through a Dutch initiative [12]. The ASV was equipped with a collision avoidance system based on model predictive control (MPC). The testing aimed at verifying COLREGs-compliant (International Regulations for Preventing Collisions at Sea) behaviour of the ASV in different challenging situations using automatic identification system (AIS) data from other vessels. The scenarios cover situations where some obstacle vessels obey COLREGs and emergency situations where as some obstacles make control decisions that increase the risk of collision. The MPC-based collision avoidance method evaluates a combined predicted collision and COLREGs-compliance risk associated with each obstacle and chooses the best way out of dangerous situations. The results from the verification exercise in the North Sea show that the MPC approach is capable of finding safe solutions in challenging situations, and in most cases demonstrates behaviours that are close to the expectations of an experienced mariner. According to Deltares' report, the sea trials have shown that the technical maturity of autonomous vessels in practice is already further developed than expected.

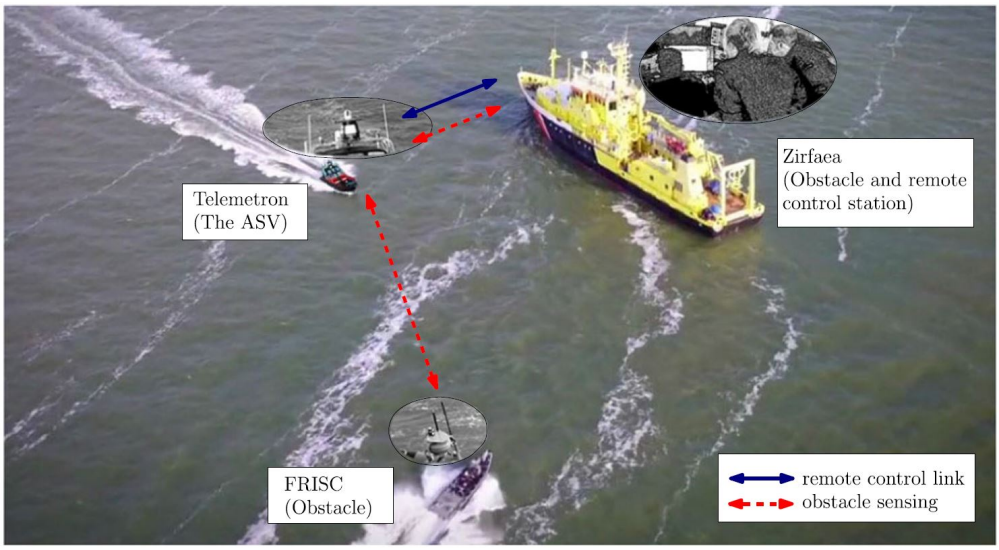


Figure 2.4: A test of the USV in the North Sea.

A paper written for an IEEE Conference in 2010 [7] describes experiments done in Australia of a autonomous surface research vessel capable of navigating in complex inland waters. The vessel is a 16ft (4.9m) long solar powered catamaran designed to collect water column profiles while driving. The size can be characterised as medium being several times larger than the largest of the compared models in Figure 2.2. The catamaran is equipped with an unspecified laser scanner for detecting obstacles. The control system utilises the input from the laser scanner to mathematically place a repelling magnetic field around the obstacles. The repelling magnetic field gives input to the control signal to steer the vessel around and away from the detected obstacles. Results show that the vessel is able to autonomously operate in previously unmapped shallow water environments. Despite the larger hull size of the vessel, the research still gives an indication that laser scanners on smaller vessels can be a viable sensor solution for detecting obstacles.

Figure 2.5: Solar powered research USV.

2.2 Braitenberg

2.2.1 Braitenberg experiment

As touched upon in the Introduction, Valentino Braitenberg illustrates a very interesting thought experiment on insect and animal behaviour explained in a simplistic way. Braitenberg wrote a book called Vehicles: Experiments in Synthetic Psychology, which invites the reader to mentally build progressively more complex machines [3]. Starting with the simplest of vehicles with one sensor and one actuator. In an ideal environment, the vehicle would move only forward, but in the real world, the vehicle is exposed to friction and external forces. The sometimes unknown external factors will make the vehicle tend to deviate from driving in a straight line. And even more so if the vehicle is a boat on a lake, where the behaviour would be more complex. Adding another sensor and another actuator will increase the complexity of the behaviour as described in Section 1.1.

Adding even more sensors further increases the complexity. As the complexity increases, the vehicles seem to exhibit more and more sophisticated behaviour, and even something that resembles feelings. Adding only simple electro-mechanical components, Braitenberg motivates the vehicles towards logic reason (via McCulloch-Pitts neurons [5, 13]).

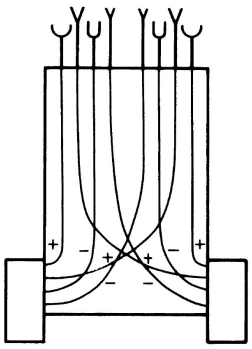


Figure 2.6: A wheeled Braitenberg vehicle with more sensors.

As the complexity of the vehicles increases, shapes and objects are recognised and regularities are represented. Hebbian associations [6] (actions that get stronger the more they are used) allows for more robotic vehicle control concepts to emerge [21]. With more functionality implemented, the vehicles can show signs of learning and memory. Causation (reaction to situations as constant succession) and attention (as self-control over associations) can finally lead to behaviour that can resemble trains of thoughts. At this point one can say that the human mind is born and all Braitenberg must do is to add a system for social and moral skills.

Until this point, the amount of biological and neurophysiological implementation have only been approximated, but these vehicles are easy to create and can achieve high levels of artificial environmental recognition. Braitenberg's thought experiment started development of an entire generation of robots with various configurations and uses.

Even though the Braitenberg concept show simple explanations of animal behaviour, it is clear that it springs roots and has connections to a deeper artificial intelligence. By starting with simple vehicles, increasing the complexity as the system and control algorithms are developed, the Braitenberg concept has the potential of giving robots and vehicles artificial intelligence.

When the foundation of a Braitenberg vehicle and control is laid, different sensors can be implemented. For a USV, this could include water temperature sensors or water salinity sensor, and can be used to have the USV follow a stream of water where the different measured values changes. All together, a vehicle using the Braitenberg concept can navigate autonomously in an environment, while doing other tasks along the way. And the behaviour can seem animalistic as an animal looking for food.

2.2.2 Braitenberg and Bio-inspired robot control

One of the most common vehicles where the Braitenberg concept is being implemented is two-wheeled robots. A student program called SyRoTek at the Czech Technical University in Prague demonstrates robots navigating in an unknown environment using control algorithms based on the Braitenberg concept [8, 37]. Here the robot is set to drive at a constant forward velocity and using LiDAR to detect obstacles. The Braitenberg control algorithm is used to keep the robot at an equidistance to any obstacle. This gives the robot the ability to drive in the middle of a hallway. A simulation was made and compared to a real life test. The simulation shows that the robot was able to navigate in an environment while avoiding obstacles. In comparison with the real robot, tests show that the real robot tend to move closer to the obstacles and take sharper turns. This indicates that the real navigation system is operating slower than in the simulation.

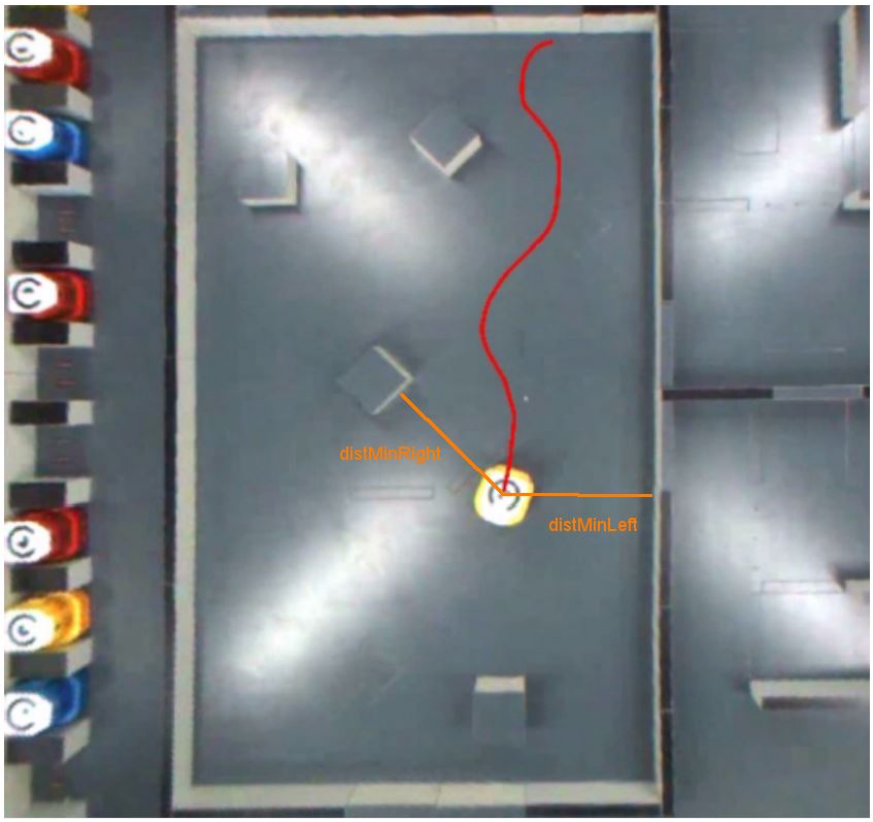


Figure 2.7: Robots in the SyRoTek program. Several robots are stacked on the left, ready for over-the-air programming and testing.

Recent research has also been done on mechanical snake-like robots and the use of Braitenberg algorithm as navigation control system. In order to move, the snake utilises the environment and push itself forward using detected obstacles. The environmental awareness is therefore crucial for an efficient forward motion [29]. Simulations have been done on a robot with passive wheels and active joints using the Braitenberg algorithm to navigate towards a signal source. The snake has two forward facing, symmetrically placed sensors at the head and is moving in a sinusoidal movement. The results show trajectories that resembles the behaviour of a wheeled robot with active wheels [20]. The research therefore shows the potential of using Braitenberg in other type of robots.

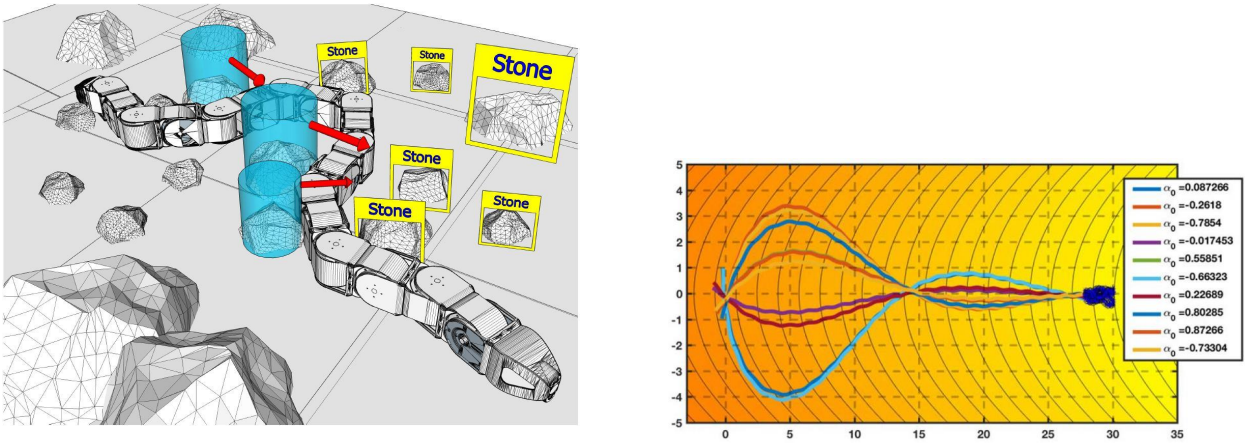


Figure 2.8: (a) shows a simulation of the perception based control where the snake uses the obstacles to move forward. (b) a simulation of the snake with passive wheels and active joints. The snake is attracted to the signal source on the right and is navigating using the Braitenberg algorithm.

The Braitenberg vehicles has also been combined with memristive neuromorphic circuits (MNCs). A memristor is a memory resistor, and combined with neuromorphic circuit architecture, gives potential of creating electronic chips great for AI robotics [26, 32]. A recent article where Braitenberg is combined with MNCs shows that a two-wheeled robot can learn how to follow a line [38]. The self-adapting navigation system shows a very short response latency ($\approx 56ns$) to input sensory information. Using bio-inspired hardware together with bio-inspired control algorithm shows great potential towards creating new robots with advanced AI technology.

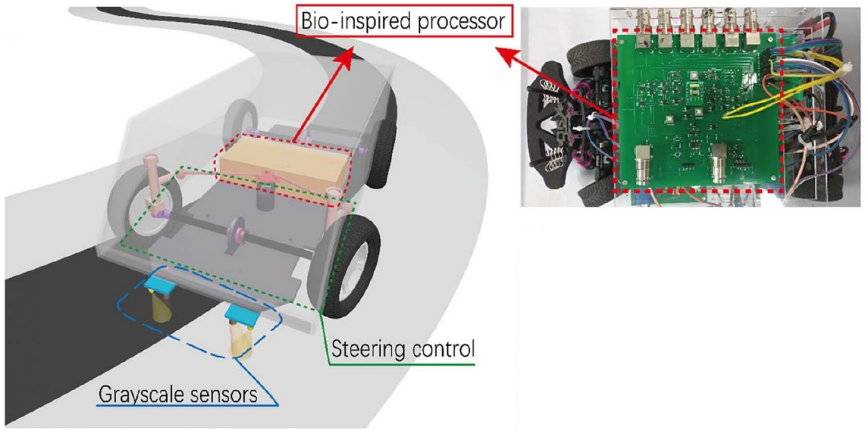


Figure 2.9: A wheeled Braitenberg vehicle using bio-inspired hardware together with bio-inspired control.

2.3 Review of previous projects

Developing a fully autonomous USV is complex and requires work in many different areas, such as mechanical set up, software development, simulation and testing. Since the development of this project is primarily student driven, several projects are established to divide the work load where each project aims to create a piece of the puzzle towards the autonomous USV. Therefore, the work in this thesis builds on two previous projects.

Firstly, a group of students from Kongsberg designed and built a physical robot during a summer project in 2018 called AnSweR [10].

Figure 2.10: USV designed by AnSweR

The robot is approximately 40x60 cm and designed to operate in sheltered waters. Its tasks include doing bathymetric measurements, scanning the bottom of a lake or a river using a 3D camera attached to a clear piece of acrylic on the bottom of the robot. Other sensors attached are a GPS module and a multi-sensor that offers depth, speed, and temperature measurement. A one-card computer Odroid handles the 3D vision and sensor data while an Arduino Mega handles the control and propulsion of the robot. Propulsion is made with two water jets, which would characterise the vessel as underactuated. An underactuated vehicle is a vehicle that has a higher number of degrees of freedom than the number of actuators. This vessel has three degrees of freedom. It can move in the x and y direction and rotate around the z axis. Turning of the robot is done using differential thrust. The motors are not able to do any reverse motion.

With its flat hull design, the robot can be considered a hybrid between a kayak and a catamaran. This gives more stability compared to the traditional kayak and is more suitable for bathymetric measurements. However, if the robot should capsize, all capability of manoeuvring is lost. The use of a self-righting hull would therefore increase the robustness of the robot and should be investigated in the future.

A ROS program was developed giving the possibility to remotely control the bot and read GPS position and depth sensor data, all wirelessly. The program is available as a repository on GitHub.

Secondly, a master thesis made by Min Tang was completed by the end of 2019 where the aim was to make a digital twin of the robot [33]. The digital twin was created in a ROS/Gazebo environment and includes a simplified physical model with outer dimensions, weight and dynamics. Expanding on the ROS program from AnSweR, the program allows for manual control of the robot and the model in simulation. Part of this thesis was conducted in order to bring the level of autonomy up to level 4 based on Table 1.1. Level 1 and 4 was considered achieved at that stage. The program also allows for a ground station to be configured and used to control the digital twin and robot while the robot is out on the water.

The thesis explains how to set up the necessary software on both the onboard computers and the ground station. It also explains how to set up ROS and the Gazebo environment.

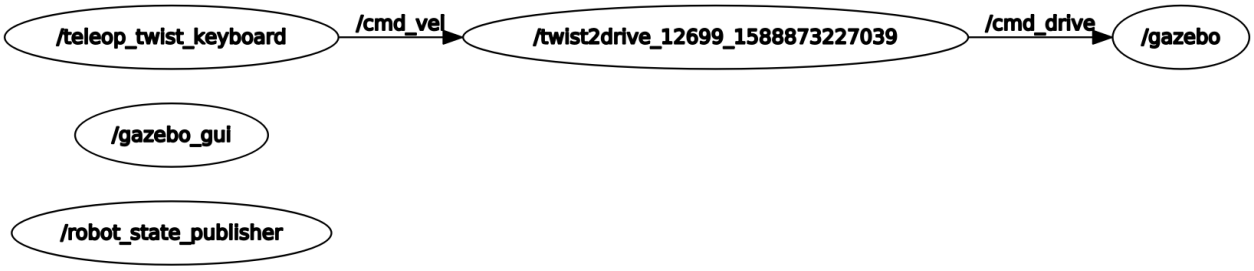


Figure 2.11: rqt_graph of ROS environment of the Digital Twin.

The ROS command *rqt_graph* can show how the ROS nodes and topics interact with each other. Figure 2.11 shows the keyboard inputs publishing messages on the topic */cmd_vel*. The */twist2drive* node subscribes to messages on */cmd_vel* and generates drive messages on topic */cmd_drive*, which are sent to the Gazebo environment.

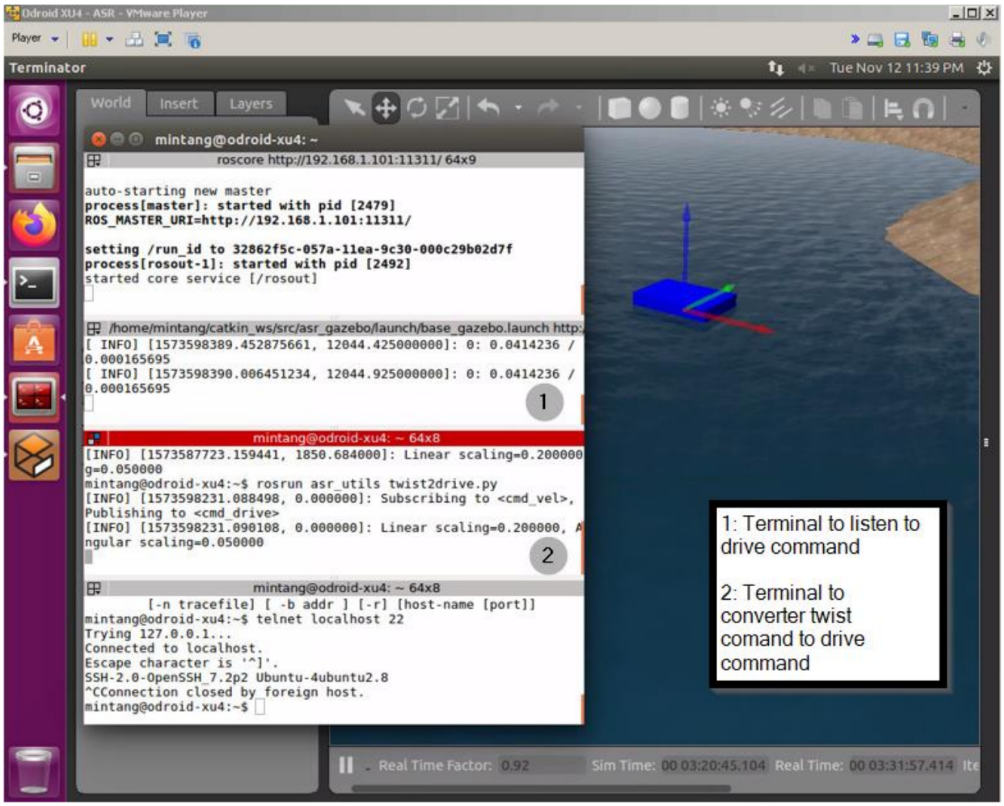


Figure 2.12: Simulation of the robot in ROS/Gazebo environment.

Using terminal windows, keyboard commands can be given in the */teleop_twist_keyboard* window while the drive command can be seen in another. Using the ROS command *rostopicecho*, one can listen to messages being published in any of the topics. Operating the robot using the keyboard is visualised in Gazebo.

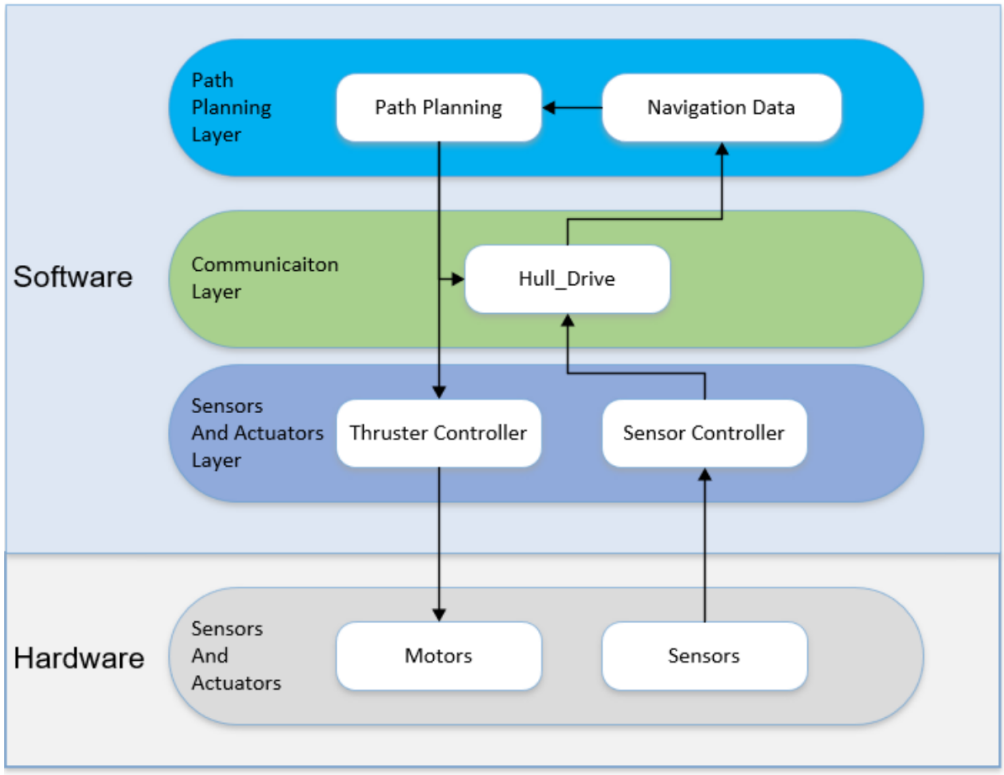


Figure 2.13: Suggested control structure of the digital twin.

Furthermore, in the development of the GNC system, the thesis introduces a hierarchical layered control structure as in Figure 2.13. This will be further used in this thesis to simulate more on the sensors and actuators layer, in addition to exploring a new control algorithm at the path planning layer. A low cost 360-degree LiDAR could be added for environmental awareness and obstacle detection. ROS packages such as "$navsat-transform$", "$razor-imu-9dof$" and "$RPLiDAR$" could be added when using the sensors.

Chapter 3

Methods

3.1 Braitenberg assumptions and hypotheses

The following section derives the mathematical model of a Braitenberg vehicle 2b as a set of equations. The general equations are simplified for the case of a point like stimulus source, a common configuration when using Braitenberg vehicles. A deterministic model is presented.

The robot in this project is designed as a surface vessel to be operated in calm waters where waves are neglected. It is therefore assumed that the robot strictly moves on a flat surface. The state of the robot can be completely represented by the position and orientation values (x, y, α_0) such that $(x, y) \in \Re$ and $\alpha_0 \in (0, 2\pi]$. This means that the position values (x, y) can be both positive and negative and the robot can freely operate in any of the quadrants of the coordinate system.

It is assumed that the robot will be equipped with sensors that can map the local environment and place obstacles as coordinate points in the global frame. The position of the obstacles are purely placed using the GPS and IMU data received from the simulation. These mapping sensors will not be simulated. Instead, obstacles are placed in the global frame and an algorithm checks if the robot is within the area where the mapping sensors can detect them. The position and direction towards the obstacles are then compared to the position of the robot.

In addition, two fictional sensors will be simulated and placed at the two front corners of the robot, left and right. The simulated sensors will use the positional data of the GPS and IMU as input data and generate a signal considered the output of the sensors. The purpose is to determine which of the corner points of the robot is closest to the waypoint or obstacle, then use this information to steer the robot. The formula generating the output signal is called the stimulus function. The stimulus function S is modelled as a smooth parabola function using position values (x, y) as input. The control signal U for the thrust of the robot motors are calculated as $U(S)$.

One of the goals for this project as a whole is to develop a surface vessel for research at sea, inland lakes and rivers. Data sampling and re-sampling should be made possible over an extended period of time. The control algorithm is therefore designed to track a set of waypoints and continues in a loop until data sampling is done. A design choice is made to set the initial state of the robot motors to maximum speed forward. The motor control function shall therefor subtract a control value based on the stimulus function in order for the robot to turn and navigate. The standard state of motor controls are set as $U_{R,L} = U_{max}$, where U_{max} is the maximum speed control signal. Using the stimulus value to subtract control signal from the right motor will make the robot turn right, $U_R = U_{max} - U(S_R)$. This will make the robot drive forward until a control signal is subtracted from one of the motor signals making the robot turn. The driving behaviour can be compared to a tank driving with tracks, a differential speed vehicle.

In order to have the robot avoid obstacles, the stimulus values will be cross-subtracted making the robot turn the opposite direction, $U_L = U_{max} - U(S_R)$. In addition, a combination of the two modes attraction and avoiding will make the robot navigate while avoiding obstacles.

Obstacles are added as single coordinate points without extension or volume. LiDAR is a sensor which creates a cloud of points where a classification system should be able to identify which of the points are an obstacle and which are not. Based on the sensor information received from a LiDAR, adding the obstacles as points seems reasonable.

3.2 Concept

The concept of the simulation and the model setup is based on a parabolic equation. The equation is called the stimulus function. The function generates an output signal from the left and right simulated sensor of the robot which senses the power magnitude of the signal source. The signal source is placed at the global maximum of the stimulus function. Based on the output signal from the sensors, a speed control signal can be computed to control the right and left motor on the robot. The concept will be shown through a set of simulations where waypoints are reached and obstacles are avoided. The waypoints are randomly placed in the environment as a list of known points.

The stimulus function in its general form is

$$S(x, y) = g_0 - a \cdot x^2 - b \cdot y^2, \tag{3.1}$$

where

S is the stimulus value,

x and y are the position coordinates of the robot in meters,

g_0, a and b are coefficients of the function.

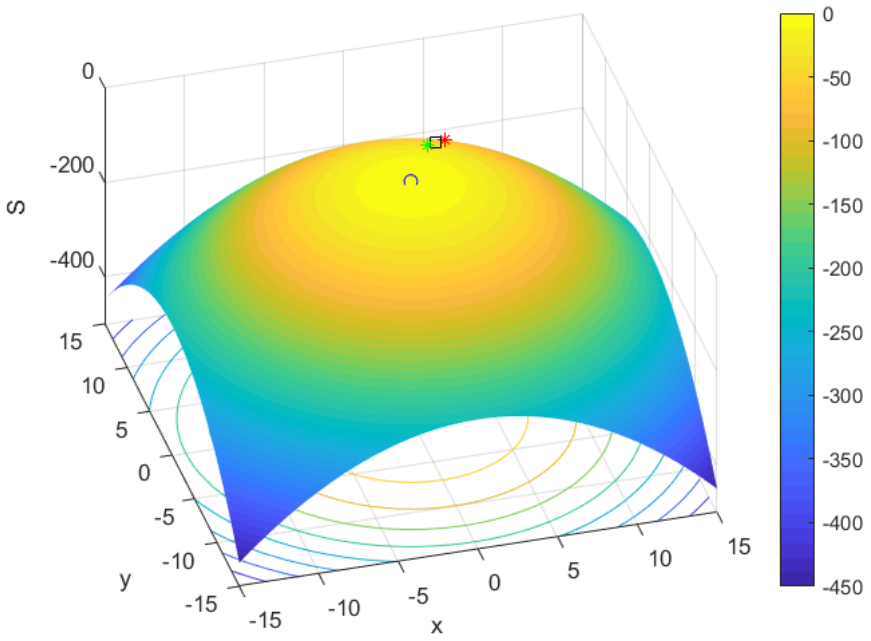


Figure 3.1: Stimulus function around a global minimum at [0,0]. The z-axis gives the stimulus value $S(x, y)$.

Figure 3.1 visualises the stimulus function and how it behaves around a waypoint. Placing a robot anywhere in the coordinate system will give the sensors negative input values. Using these sensor values to subtract from the motor signal should make the robot be attracted to the waypoint or deflect from an obstacle.

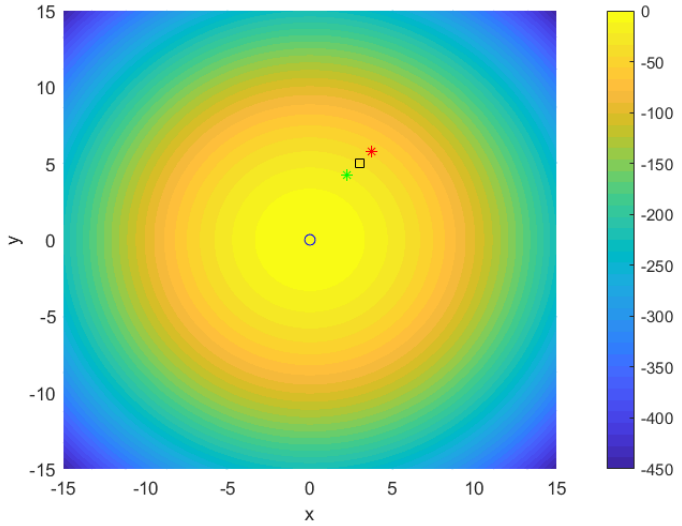


Figure 3.2: Stimulus function around a global minimum at [0,0] including the robot (black square) heading towards the bottom right corner. It also shows the right sensor (green), left sensor (red) and the waypoint (black circle). The sensors are placed on each side of the robot. The stimulus value S for the left sensor will have a greater negative value than the right sensor. Using the S value from the left sensor to subtract speed from the right motor will in this case make it turn clockwise towards the waypoint.

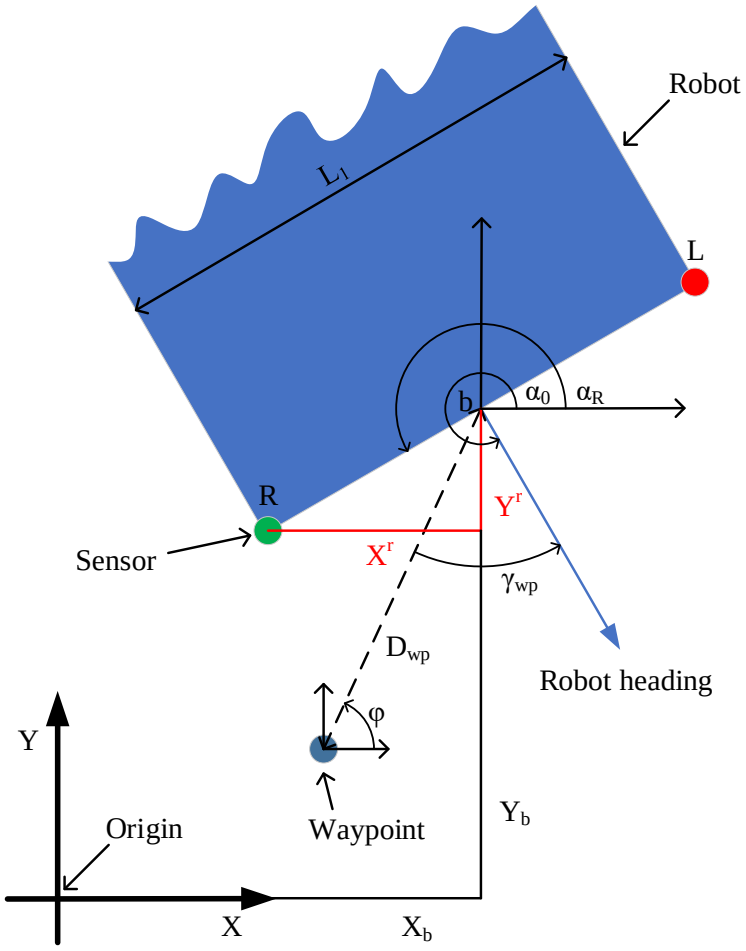


Figure 3.3: Robot with sensors in the global coordinate system.

The angle α_0 and position of the robot at b are read from the onboard IMU and GPS respectively.

The coordinates of the right and left sensor are then calculated as:

$$R_x = X_b + X^r = X_b + \frac{L_1}{2} \cdot \cos(\alpha_R) \tag{3.2}$$

$$R_y = Y_b + Y^r = Y_b + \frac{L_1}{2} \cdot \sin(\alpha_R) \tag{3.3}$$

$$L_x = X_b + X^l = X_b + \frac{L_1}{2} \cdot \cos(\alpha_L) \tag{3.4}$$

$$L_y = Y_b + Y^l = Y_b + \frac{L_1}{2} \cdot \sin(\alpha_L) \tag{3.5}$$

where

X_b and Y_b are the coordinates at point b on the robot,

L_1 is the distance between the sensors,

α_R is the angle towards the right sensor in radians, $\alpha_R = \alpha_0 - \frac{\pi}{2}$,

α_L is the angle towards the left sensor in radians, $\alpha_L = \alpha_0 + \frac{\pi}{2}$.

The position of the sensors are then passed to the stimulus function 3.1 which calculates the stimulus values. In addition to the sensor values, the stimulus value for the centre of the robot at point b is also calculated. Together with the coordinates of the next waypoint, the stimuli value is calculated as:

$$S = g_0 - a_0 \cdot (X_{wp} - X_b)^2 - b_0 \cdot (Y_{wp} - Y_b)^2 \tag{3.6}$$

where

X_{wp} and Y_{wp} are the global coordinates of the waypoint,

X_b and Y_b are the global coordinates of the point where the S value must be found.

g_0, a_0 and b_0 are coefficients of the parabola function.

Equation 3.6 uses the global position of the robot and the waypoint. This evaluates the position of the robot compared to the waypoint. Using the global position of sensors together with the global coordinates of the waypoint, gives the S value of the stimulus equation with a global minimum at the waypoint centre. The S values are then used to control the two motors of the robot in order to track the waypoint.

As shown in Figure 1.2, by switching the path of signal from sensor to motor, the algorithm can also be used to deflect from a point. Once an obstacle is detected by the sensors, the navigation switches to avoidance mode. The method of switching can be done either absolute or with a weighted influence from both tracking and avoidance acting together. Using a weighted method gives the behaviour that as the robot gets closer to an obstacle, the more it tries to avoid it.

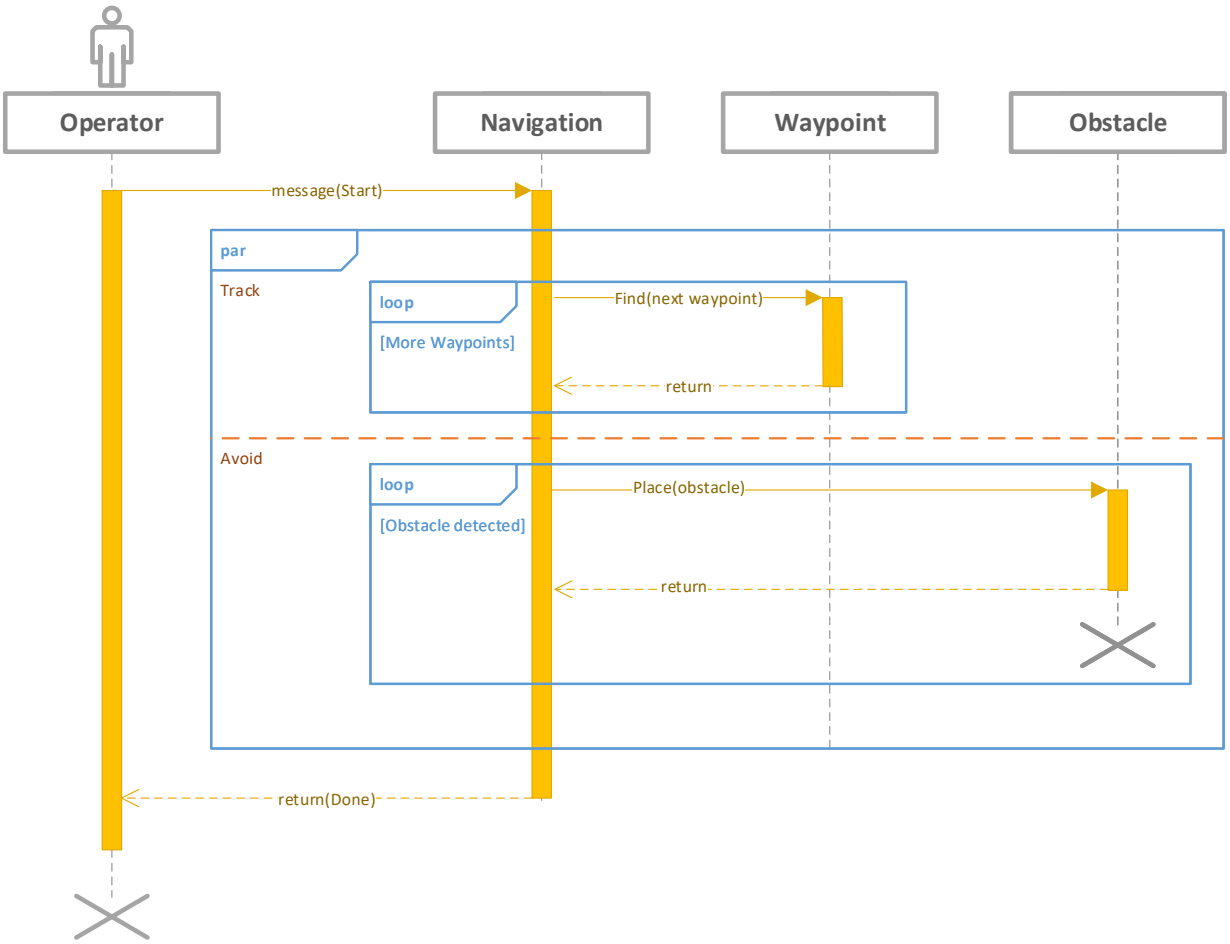


Figure 3.4: Unified Modelling Language (UML) Sequence diagram of the process. The navigation is running two parallel modes. While there are more waypoints, the navigation will track the next waypoint. If there is an obstacle detected, the navigation will take this into consideration. When the obstacle is no longer detected, the avoidance mode will terminate.

3.3 Proposed model

3.3.1 Reaching a waypoint, go to goal

The initial state of the robot includes setting both motors to maximum constant forward velocity. In order to have the robot attract to a waypoint, each of the sensor readings must be passed to the controller of the corresponding motor, right sensor to left motor controller, and left sensor to right motor controller. Using equation 3.6, the right and left sensor readings S_R, S_L including the centre S_b are used to calculate the motor control inputs as follows:

$$\begin{aligned} &if\ \ S_R < S_L: \\ &\qquad U_{RW} = \frac{S_R - S_b}{k \cdot D_{wp}} \\ &\qquad U_{LW} = 0 \end{aligned} \tag{3.7}$$

$$\begin{aligned} &if\ \ S_R > S_L: \\ &\qquad U_{RW} = 0 \\ &\qquad U_{LW} = \frac{S_L - S_b}{k \cdot D_{wp}} \end{aligned} \tag{3.8}$$

where

D_{wp} is the the distance to the waypoint,

k is a normalisation factor set to $2m$. This makes U_{RW} and U_{LW} unitless.

Dividing by D_{wp} makes the stimulus value less dependent on the distance between the robot and the waypoint. This makes the robot behaviour more consistent when navigating on both short and long distances.

The calculated values U_{RW} and U_{LW} are then subtracted from the initial state of forward velocity. Since the S values are already negative, the equations add the values.

$$u_r = U_{max} + U_{LW} \tag{3.9}$$
$$u_l = U_{max} + U_{RW} \tag{3.10}$$

where

U_{max} is the control signal setting maximum speed of the motor.

In case of Figure 3.3, the right sensor has a greater value than the left sensor and equation 3.8 will rule. The left motor will receive 0 input to subtract, while the right motor will receive a value that is subtracted from the speed control signal. This should turn the robot clockwise. Note that the right motor is taking the input of the left sensor and left motor is taking the input of the right sensor. Driving the robot using these equations is considered tracking mode. In practice, this will act as a controller which aims to turn the robot straight towards the waypoint.

When setting the robot to track a random waypoint in the environment, the expected result should be similar to this:

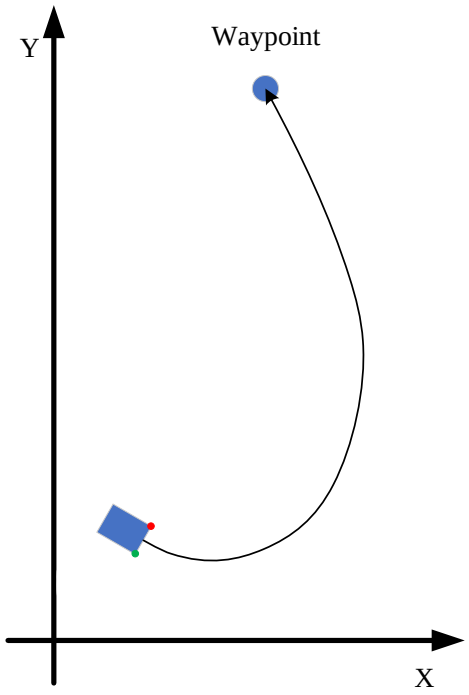


Figure 3.5: Path to waypoint

3.3.2 Avoiding an obstacle

The method used for tracking a waypoint can also be used in a situation where the robot must avoid an obstacle. It is assumed that the sensors can both determine at what direction the obstacle is and measure the distance to it. The obstacle can then be placed in the global coordinate system. The calculations for controlling the motors can be the similar to the track mode, but the control signal must be passed from right sensor to right motor and left sensor to left motor. Then the robot turns away from the obstacle, instead of towards it.

As described in section 3.3.1, the functionality of the tracking mode should keep the robot heading straight towards the waypoint. The exact opposite case would make the robot run away from the object, pointing the heading straight away. Using this method when avoiding obstacles would make the robot want to turn away from the obstacle as soon as it is detected, even if the obstacle is not obstructing a straight path to the waypoint. Another way of solving this is to have the heading turn 90 degrees away from the obstacle. This will make the robot deflect to one of the sides of the obstacle instead of running away from it.

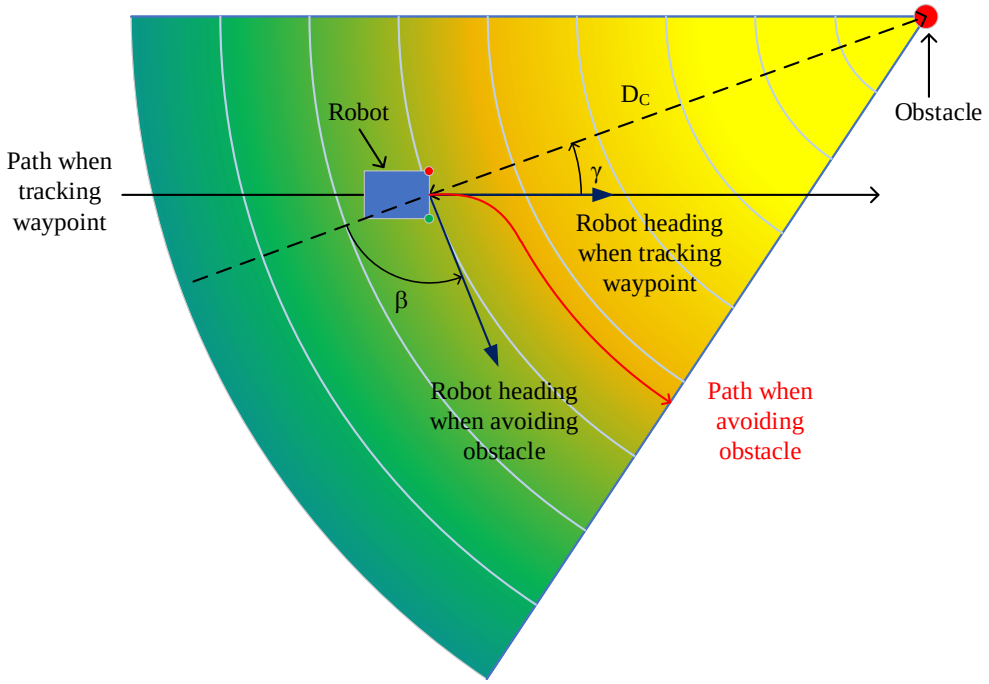


Figure 3.6: Robot behaviour when an obstacle is detected. How the robot deflects from the obstacle depends on the exit angle β.

In avoidance mode the heading is rotated and position of sensors are calculated as follows:

$$R_x = X_b + X^r = X_b + \frac{L_1}{2} \cdot \cos(\alpha_R + \beta) \tag{3.11}$$

$$R_y = Y_b + Y^r = Y_b + \frac{L_1}{2} \cdot \sin(\alpha_R + \beta) \tag{3.12}$$

$$L_x = X_b + X^l = X_b + \frac{L_1}{2} \cdot \cos(\alpha_L + \beta) \tag{3.13}$$

$$L_y = Y_b + Y^l = Y_b + \frac{L_1}{2} \cdot \sin(\alpha_L + \beta) \tag{3.14}$$

where

β is either $\frac{\pi}{2}$ or $-\frac{\pi}{2}$ depending on the heading towards the obstacle.

Then γ gives the direction that the heading is rotated.

$$if\ \ \gamma < 0 :\ \ \beta = \frac{\pi}{2} \tag{3.15}$$

$$if\ \ \gamma > 0 :\ \ \beta = -\frac{\pi}{2} \tag{3.16}$$

where

γ is the angle between the heading of the robot and the direction towards the obstacle.

These positions are then passed to the stimulus function and the motor signals are calculated in the same way as equations 3.7 and 3.8.

$$\begin{aligned} if\ \ S_R < S_L :& \\ U_{RO} &= \frac{S_R - S_b}{k \cdot D_c} \\ U_{LO} &= 0 \end{aligned} \tag{3.17}$$

$$\begin{aligned} if\ \ S_R > S_L :& \\ U_{RO} &= 0 \\ U_{LO} &= \frac{S_L - S_b}{k \cdot D_c} \end{aligned} \tag{3.18}$$

where

D_c is the the distance to the obstacle,

k is a normalisation factor set to $2m$.

The calculated motor signals are then subtracted from the initial state of forward velocity. Since the S values are already negative, the equations add the values as before.

$$u_r = U_{max} + U_{RO} \tag{3.19}$$

$$u_l = U_{max} + U_{LO} \tag{3.20}$$

In addition, the field of view of the sensors can be set using ω.

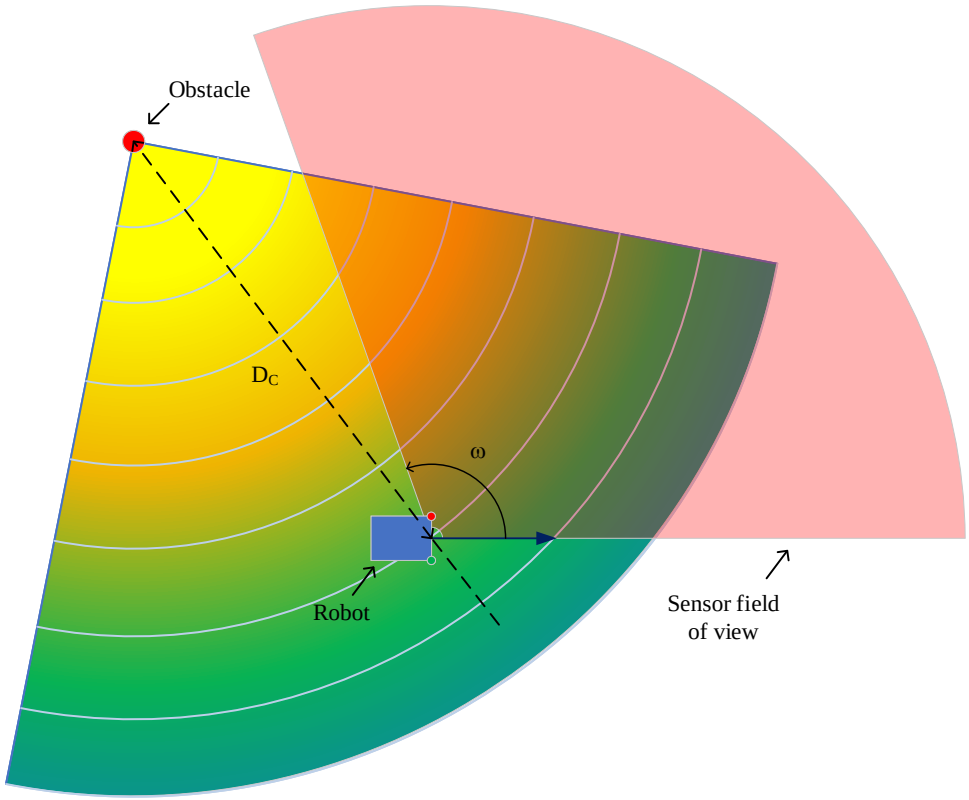


Figure 3.7: Field of view of the robot sensors.

If the robot has passed the obstacle or the obstacle is outside the field of view of the sensors, the obstacle can be ignored.

When robot is avoiding a detected obstacle, the expected result should look like this:

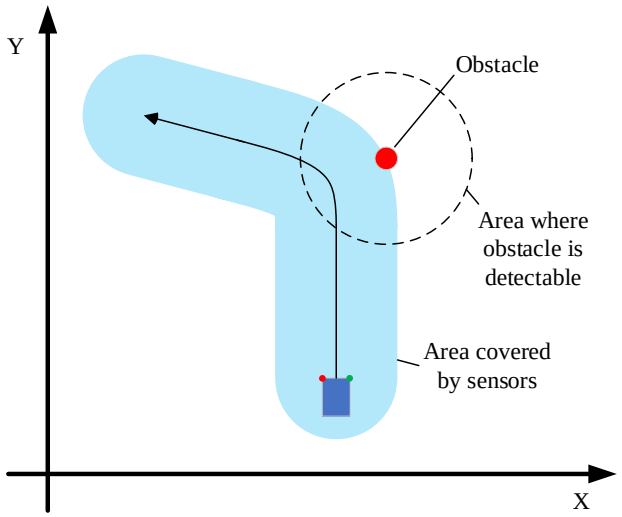


Figure 3.8: Avoiding an obstacle

3.3.3 Reaching a waypoint while avoiding an obstacle

Combining the two methods described above will give the robot the capability of navigating around an obstacle to reach a waypoint. The mapping sensors will have a certain range of detection and an obstacle will only be detected when the obstacle is within this range.

Once an obstacle is detected while tracking a waypoint, the navigation switches to avoidance mode. Switching between the two modes using a weighted method where both tracking and avoidance mode are acting together gives control over the transition between the two modes.

$$u_r = U_{max} + c \cdot U_{LW} + (1 - c) \cdot U_{RO} \tag{3.21}$$
$$u_l = U_{max} + c \cdot U_{RW} + (1 - c) \cdot U_{LO} \tag{3.22}$$

where

c is the weighing factor dependent on the distance to the obstacle.

$$c = \frac{D_c}{2 \cdot O_s} \tag{3.23}$$

where

D_c is the distance to the closest obstacle,

O_s is the range of the sensors in meters.

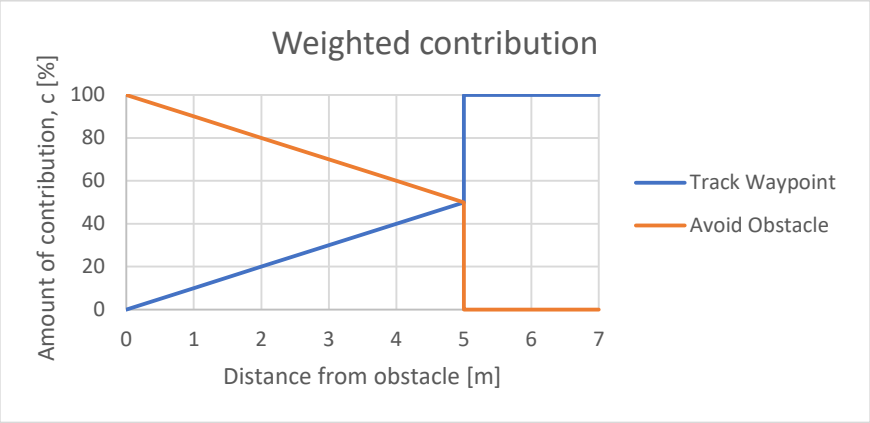


Figure 3.9: Weighted contribution function of c. By setting $O_s = 5m$ and the denominator to $2 \cdot O_s$, the weighted contribution will look as shown. As the robot approaches an obstacle, the tracking mode will switch to a 50-50 contribution from the two modes, followed by a linear decreasing contribution from track mode with equally linear increasing contribution from avoiding mode.

The weighting factor c is also used to ignore the obstacle when the obstacle is outside the sensors field of view. If $\gamma > \omega$ then $c = o$ instead of following equation 3.23.

Expected result:

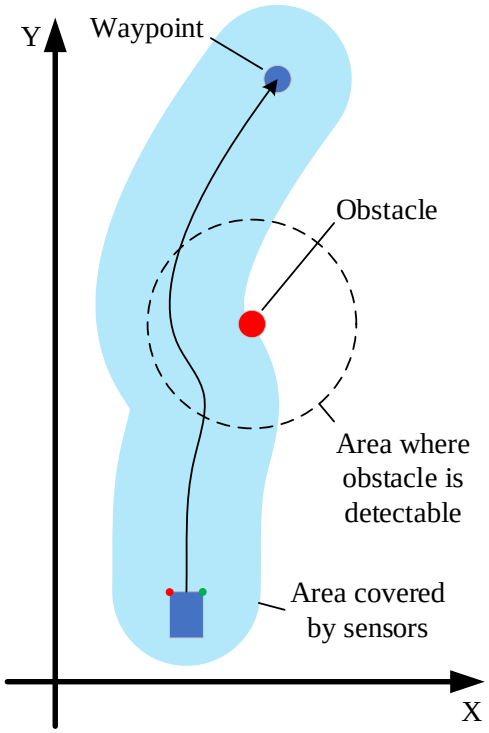


Figure 3.10: Path to waypoint while avoiding an obstacle

3.3.4 Navigate through multiple waypoints with multiple obstacles

The functionality of navigation system in the scenarios described above can at this point be used to have the robot track through several waypoints and at the same time avoid obstacles on the way. When the robot reaches within a certain distance of a waypoint, it can switch to the next set of waypoints looping back to the start.

Expected result:

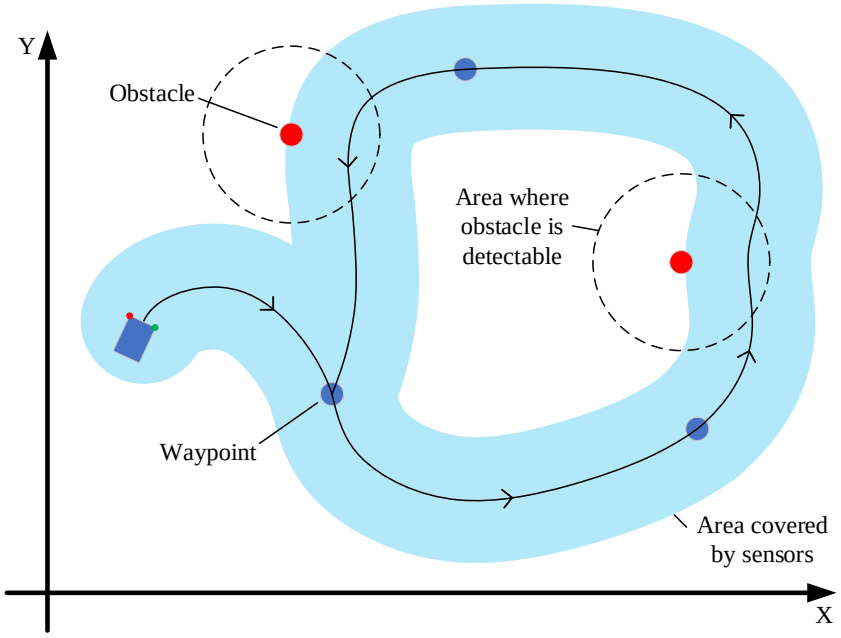


Figure 3.11: Navigating in an environment with obstacles

3.4 Manual override

The navigation system is set up to also receive control input from a human operator. This is implemented as a feature for two reasons.

1. To override the navigation system in case it does not navigate as intended. If a sensor is faulty then the operator can navigate on his own.
2. If there is a slight change in the mission planning, operator can divert to a different location before returning to the autonomous navigation.

The operator can at any point manually drive the robot, overriding the input from the Braitenberg controller and navigation system.

Chapter 4

Tools and implementation

4.1 Implementation

4.1.1 ROS and Gazebo

The main simulation tools used during this thesis is Robot Operating System (ROS) together with a 3D visualisation plugin called Gazebo. ROS is an open source program intended to set the framework for writing robot software. In short, ROS creates an environment where different software packages called nodes, communicate together. A node can be a program that performs a specific task and it can publish the result as a message to a topic. Another node can subscribe to this topic and receive the result from the first node. Any node can publish to any number of topics and any node can also subscribe to any number of topics. The Braitenberg navigation controller is configured as a separate node in the ROS environment. More information on ROS can be found in previous report [31, 33] and on the ROS website [24].

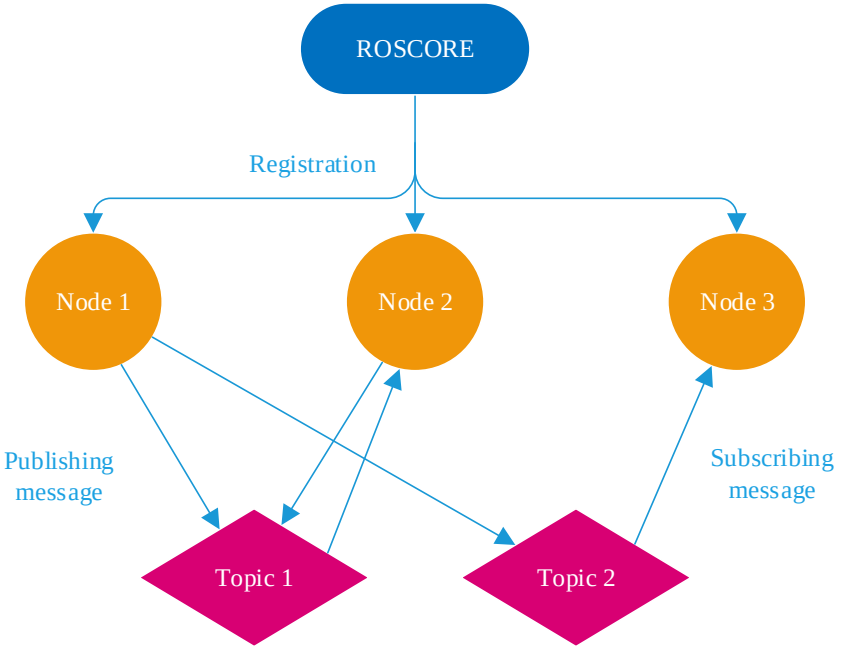


Figure 4.1: ROS environment

Gazebo is a plugin for ROS that can visualise a 3D environment. ROS can communicate with Gazebo and receive information about the environment and any object placed in the environment. The Gazebo world was set up as described in a previous thesis [33]. See section 2.3.

The robot is spawned in the Gazebo environment. In this environment, Gazebo is capable of simulating a GPS system and onboard IMU sensor. These sensor data are published on topics in the ROS framework where any other node can subscribe to the topics.

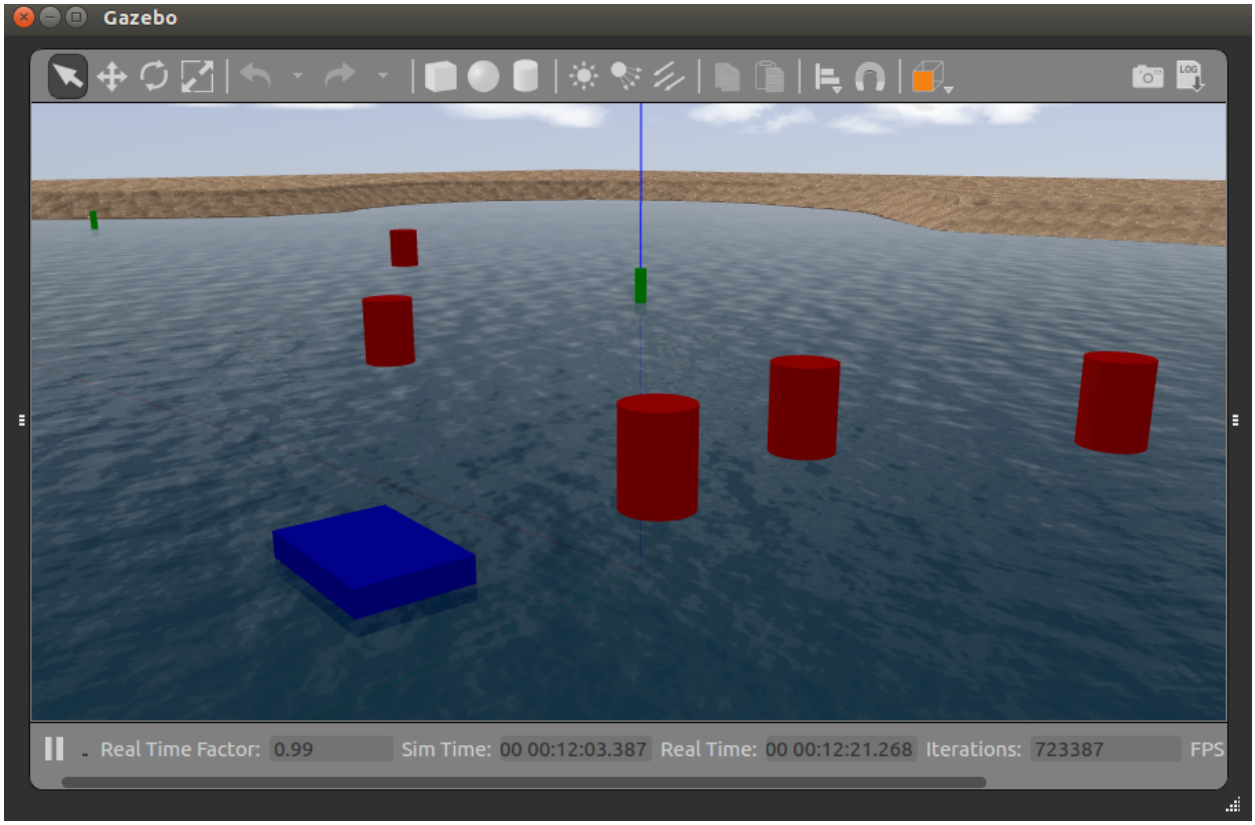


Figure 4.2: Gazebo environment with the blue robot, green waypoints and red obstacles. The origin of the coordinate system is shown by the blue line (z-axis).

Since the Braitenberg controller is using positional data to navigate, it is important that the controller can handle both positive and negative coordinate data. This makes the robot able to navigate in all the quadrants of a coordinate system. The origin of the coordinate system can then be placed anywhere on earth while the controller is still able to use the data to navigate. Therefore, the origin is placed in the centre of the Gazebo world and simulations can be done in all quadrants.

4.1.2 Controller

The ROS node of the Braitenberg controller consists of a script written in Python. The following code is described as pseudo-code to give an overview of the steps. The complete script and step by step explanation can be found the Appendix A and B.

Firstly, placement of waypoints and obstacles are defined together with constants and variables used for geometries and sensor specifications. For all the simulation steps described in Section 3.3, the following parameters have been used:

```
L1=0.2 #Distance between sensors [m]
u_max=0.25 #Max speed of the motors [unitless]
Dacc=0.8 #Accuracy when reaching waypoint [m]
Os=5 #Range of obstacle avoidance sensor [m]
w=100 #Field of view of the sensor [deg]
beta=90 #Exit angle from obstacle [deg]
```

Listing 4.1: Parameters

First objective is to have the robot track a waypoint. The position of the robot (X_b, Y_b) and the orientation α_0 is read using the $navCallback()$ function.

Algorithm 1 Tracking a waypoint without any obstacle

```
Read global position of robot incl. orientation
Calculate distance to waypoint (Dwp)
Calculate global position of sensors (Rx,Ry),(Lx,Ly)
if tracking then
    Calculate stimulus values (Sb, SR, SL)
    if sensor.right < sensor.left then
        U_RW = (SR − Sb)/(2 · Dwp)
        U_LW = 0
    else
        U_RW = 0
        U_LW = (SL − Sb)/(2 · Dwp)
    end if
    Set motor control:
    ur = u_max + U_LW
    ul = u_max + U_RW
end if
```

Within the mode of tracking, the control signal U_{RW} uses the difference between the stimulus value of the centre of the robot minus the stimulus value of the right sensor. Similarly, the left control signal uses the left sensor stimulus value. When the control signal is added to the maximum motor speed signal u_{max} it gives the motor control signals ur and ul. Since it is in tracking mode, U_{LW} is added to ur and U_{RW} is added to ul.

Algorithm 2 Avoiding an obstacle

```
Read global position of robot incl. orientation
Calculate distance to obstacle (Dc)
Calculate global position of sensors (Rx,Ry),(Lx,Ly)
if Dc < Os then
    if avoiding then
        Calculate stimulus values (Sb, SR, SL)
        if sensor.right < sensor.left then
            U_RO = (SR − Sb)/(2 · Dc)
            U_LO = 0
        else
            U_RO = 0
            U_LO = (SL − Sb)/(2 · Dc)
        end if
        Set motor control:
        ur = u_max + U_RO
        ul = u_max + U_LO
    end if
end if
```

The algorithm for avoiding an obstacle is very similar to the algorithm for tracking a waypoint. The only exception is that when calculating the motor control speeds, U_{RW} is added to ur and U_{LW} is added to ul. This switches the behaviour from attracting to avoiding.

Algorithm 3 Tracking a waypoint while avoiding an obstacle

Read global position of robot incl. orientation
Calculate distance to waypoint (Dwp)
Calculate distance to obstacle (Dc)
Calculate global position of sensors (Rx,Ry),(Lx,Ly)
if $Dc < Os$ **then**
    if *avoiding* **then**
        Same as Algorithm 2 but with the addition of:
        if $abs(gamma_O) > w$ **then**
            $c = 1$
        else
            $c = Dc/(2 \cdot Os)$
        end if
    end if
else
    $U_{RO} = 0$
    $U_{LO} = 0$
    $c = 1$
end if
if *tracking* **then**
    Same as Algorithm 1 but with the addition of:
    Set motor control:
    $ur = u_{max} + c \cdot U_{LW} + (1 - c) \cdot U_{RO}$
    $ul = u_{max} + c \cdot U_{RW} + (1 - c) \cdot U_{LO}$
end if

When combining the two modes together, the weighting factor c is introduced following the Equation 3.23. ur and ul are calculated by adding the contribution from tracking mode and avoidance mode together using c.

Algorithm 4 Navigating through a set of waypoints in a continues loop

Same as Algorithm 3 with the addition of:
if $Dwp < Dacc$ **then**
    $Nwp = Nwp + 1$
end if
if $Nwp > numberofwaypoints - 1$ **then**
    $Nwp = 0$
end if

In order to have the robot track each of the waypoints in a loop, Nwp is added to keep track of which is the next waypoint. Every time the robot comes within the set accuracy of the system $Dacc$, Nwp is increased by one.

Using the ROS command *rqt_graph* shows the ROS environment when the Braitenberg controller is running. Running ROS, Gazebo and the controller from Algorithm 4 uses the following nodes, topics and messages.

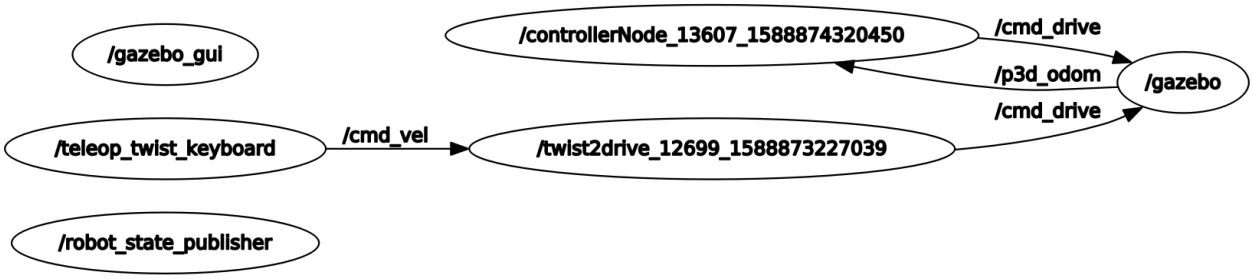


Figure 4.3: rqt_graph of ROS environment. The circles represent operating ROS nodes. The nodes publish messages to topics */cmd_vel*, */cmd_drive* and */p3d_odom*. The controllerNode is running the Braitenberg algorithm and uses GPS and IMU data from Gazebo to generate and publish motor drive messages.

Chapter 5

Results

As described through Chapter 3, the navigation simulation consists of several levels of functionality. By going through the different levels adding more functionality for each step, it gives a better understanding of how the navigation system works. The different levels are:

1. Tracking a waypoint.
2. Avoiding an obstacle.
3. Avoiding an obstacle on the way to a waypoint.
4. Tracking several waypoints in succession in an environment with randomly placed obstacles.

Each of the points are verified using a separate program file containing a controller node, each made to solve the different steps. Separating the steps makes is easily accessible for later development at each step of the program. Results are recorded using the *rosbag* command. Positional data is then exported, plotted and overlaid with multiple simulations.

5.1 Tracking a waypoint

The first step is to have the robot track a waypoint. The motor speeds are initially set to maximum speed forward and the sensor data is used to subtract and lower the control signal going to the motors.

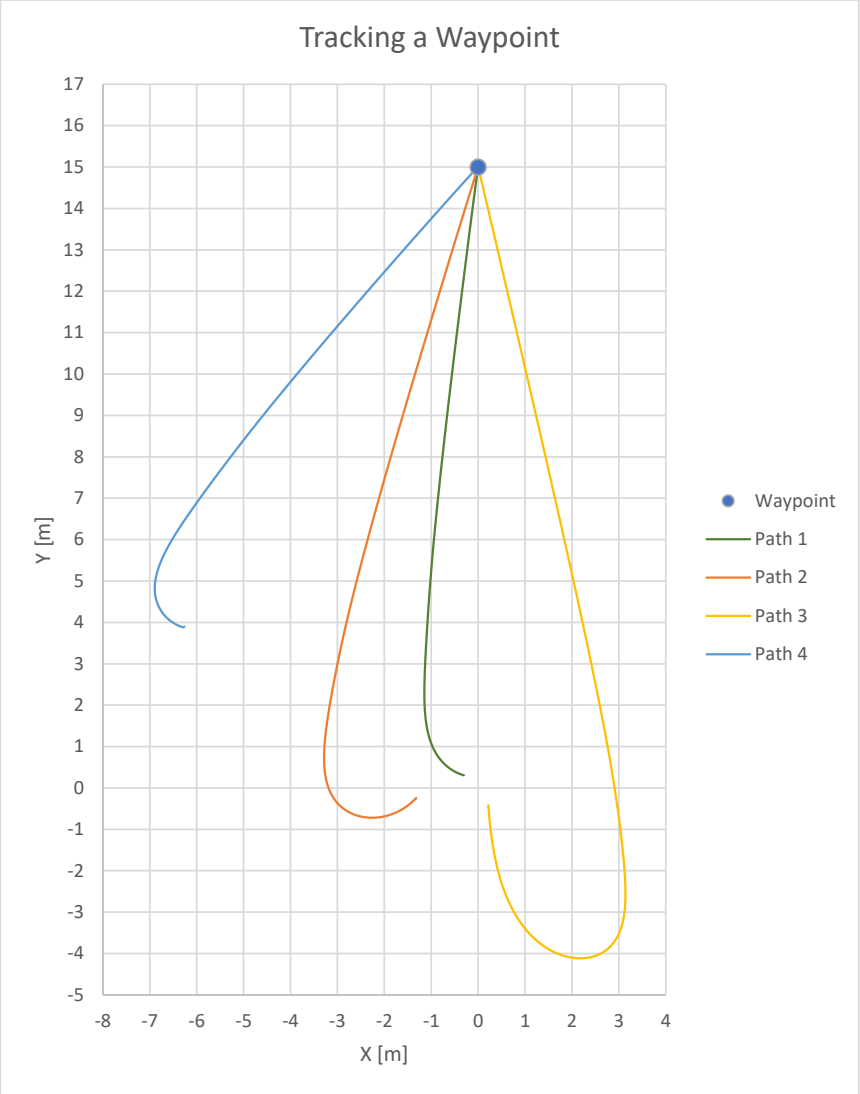


Figure 5.1: The figure shows four simulations where the robot starts at a random position and orientation. All four simulations shows that the robot takes a path that ends at the waypoint.

We can see from Figure 5.1 that the robot is quick to find the correct heading and go straight for the waypoint. Path 3 shows the robot with an initial heading away from the way point and is taking a slower turn before heading straight towards the waypoint.

Another observation is that there seem to be no overshoot when the robot is heading towards the waypoint. There is no sinusoidal movement. If this controller would be compared to a PD controller, one can estimate that the robot driving on water is a critically damped or overdamped system.

The result corresponds with that which was expected.

5.2 Avoiding an obstacle

The next step is to add an obstacle. The motor speeds are initially set to maximum speed forward. When the robots sensors detect an obstacle the navigation system should make the robot deflect from the obstacle.

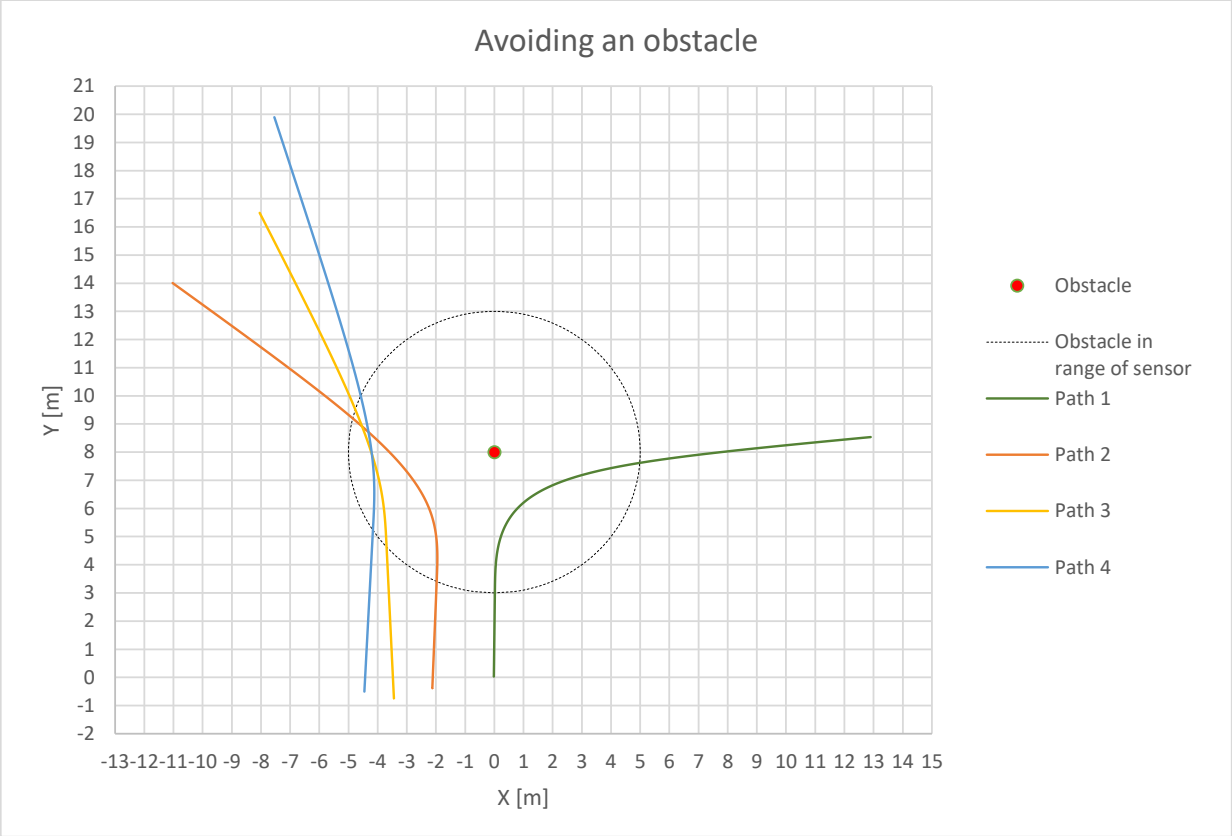


Figure 5.2: Four simulations of the robot starting at a random position. The robot is driving forward until the obstacle is detected.

Figure 5.2 shows how the robot reacts to an obstacle depending on the position of the obstacle. Path 1 shows that when robot is heading straight for the obstacle, then the deflection is large and the behaviour resembles an insect running away from danger. If the obstacle is detected further out in the vicinity as Path 4 shows, then the deflection is minor.

5.3 Reaching a waypoint while avoiding an obstacle

Combining the two features from Section 5.1 and 5.2, a simulation can be done to see how the robot reacts when it is obstructed by an obstacle while tracking a waypoint.

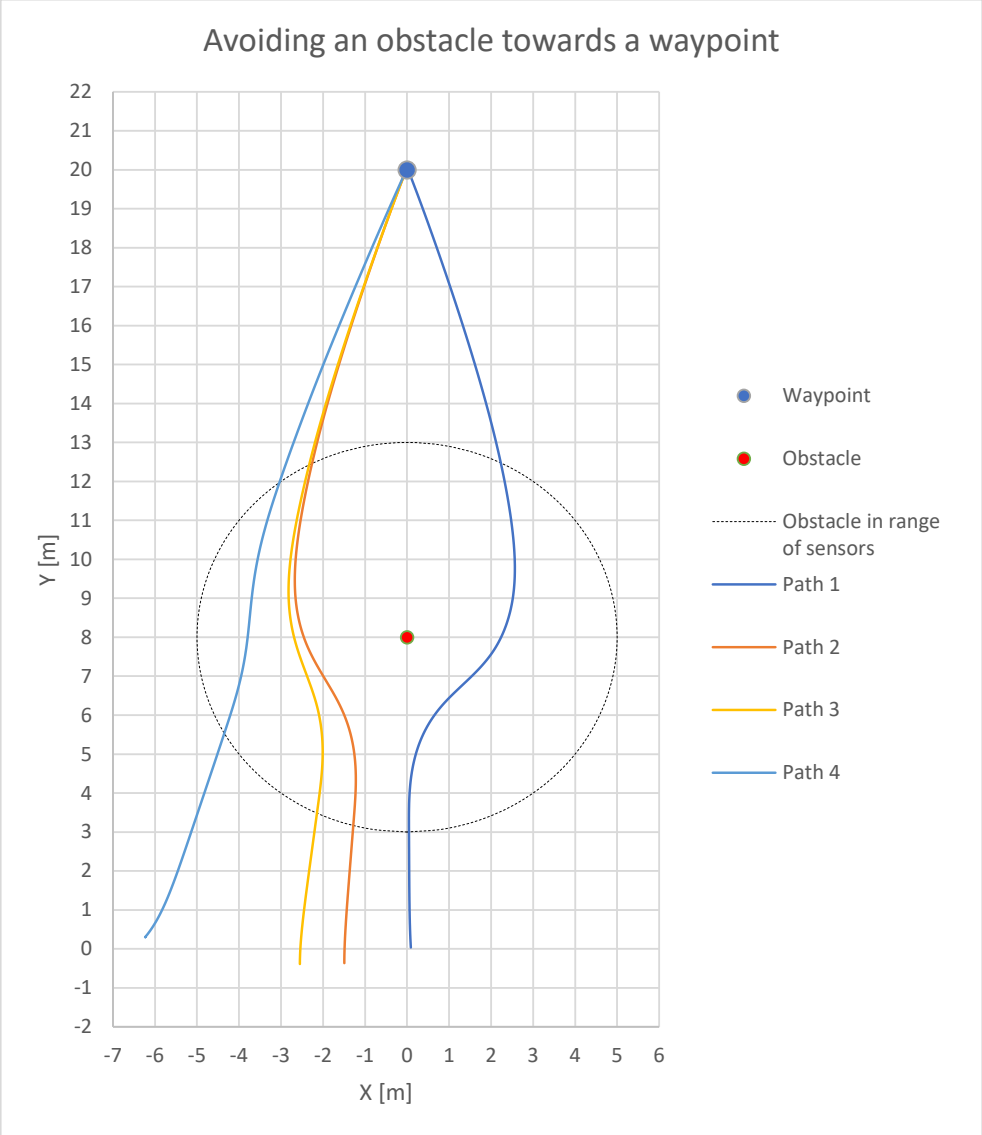


Figure 5.3: The simulations show the robot starting at four different positions. Path 1 shows the robot taking a path straight towards both the waypoint and the ahead obstacle. On Path 4 the robot is only slightly detecting the obstacle as the robot is passing by.

As seen in Figure 5.3, combining the method of tracking a waypoint while also avoiding an obstacle works. While the obstacle is in front of the robot and the robot is close to the obstacle, the avoidance mode is dominant. Where the robot is at a distance further away from the obstacle, the tracking mode is dominant.

5.4 Navigate through multiple waypoints and obstacles

When the two modes of tracking and avoidance are combined in a navigation system, the robot can handle more complex environments.

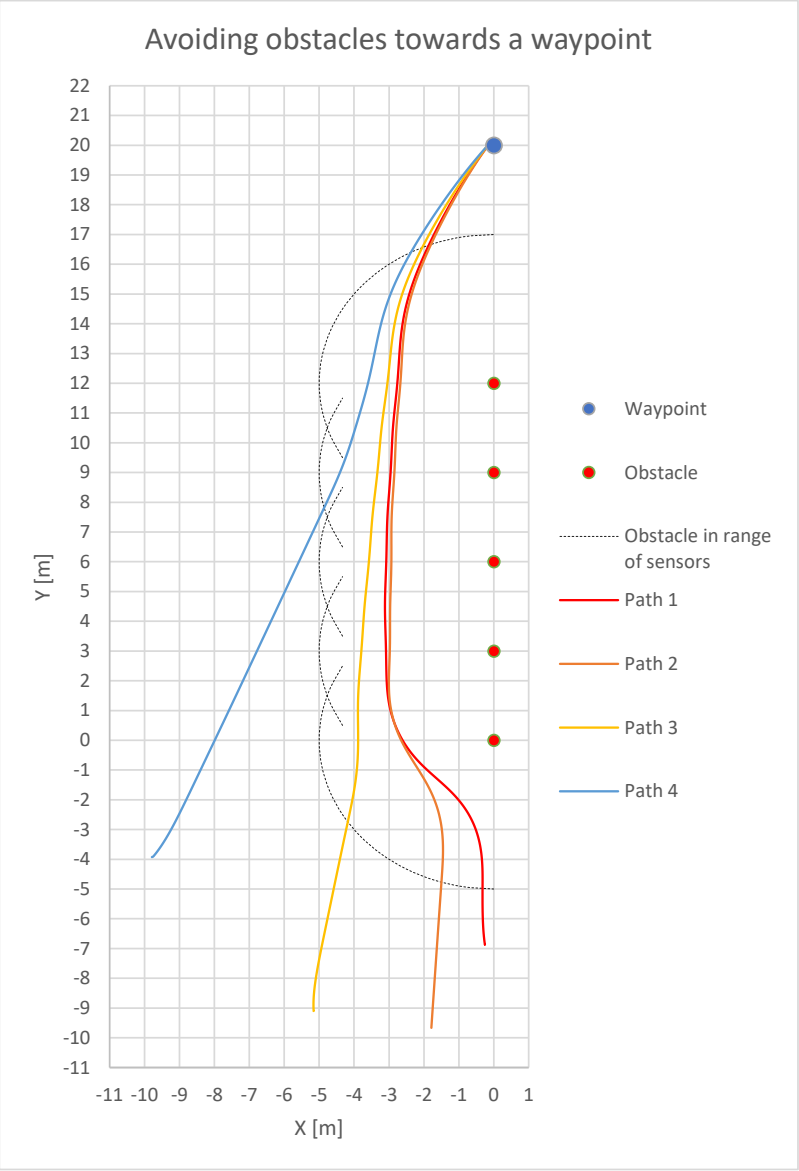


Figure 5.4: Simulation shows the robot placed at four random locations. The robot is tracking a waypoint located behind a series of obstacles. The obstacles can resemble a straight wall.

When following a line of obstacles, the robot keeps a certain distance to the obstacles. We can see from Figure 5.4 that Path 1 and Path 2 quickly follow each other even though the robot is entering the area where the first obstacle is detectable at different places.

We can also see that once the robot has passed the first obstacle, all three Path 1, 2 and 3 has a slight angle towards the obstacles. They are not completely parallel.

When the robot is following a line of obstacles, or a wall, it seems that the robot will eventually collide with the wall. To amplify the effect, a simulation was done where the waypoint was moved further behind the wall. This increases the contribution from the tracking mode compared to the avoidance mode.

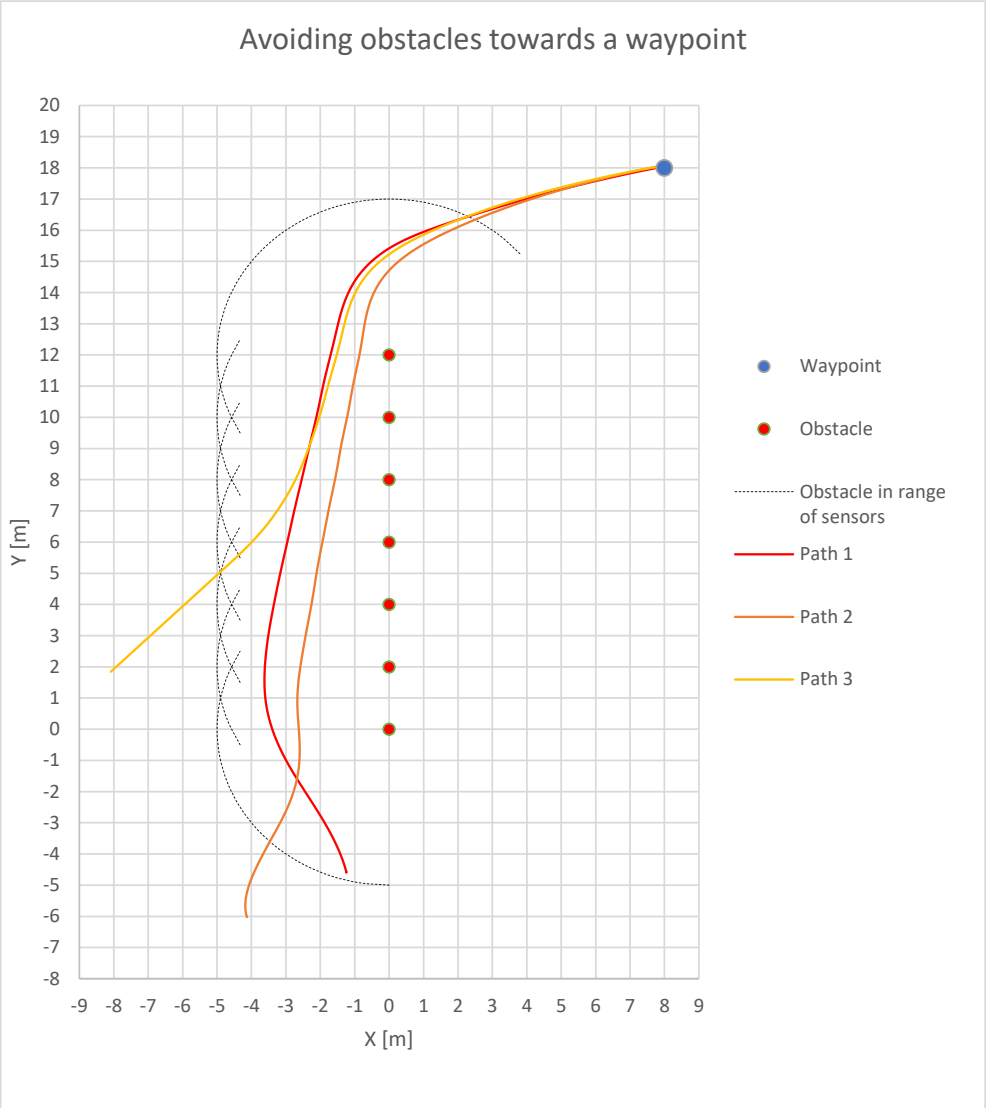


Figure 5.5: Simulation of three robots tracking a waypoint behind a wall. Robot taking Path 1 starts at a position where the first obstacle is already detectable and therefore has no forward momentum at the start. Robot taking Path 2 has a momentum going into the area where the first obstacle is detectable and therefore reaches a closer distance to the obstacles.

From Figure 5.5 we can see that the robots follow a steeper angle towards the obstacles compared to Figure 5.4.

An even more challenging simulation was done where the robot tracks a waypoint directly behind a wall. The robot is then forced to manoeuvre so that the distance to the waypoint is increasing and therefore the pull towards the waypoint will increase. The robot must keep a safe distance even though the pull towards the waypoint increases.

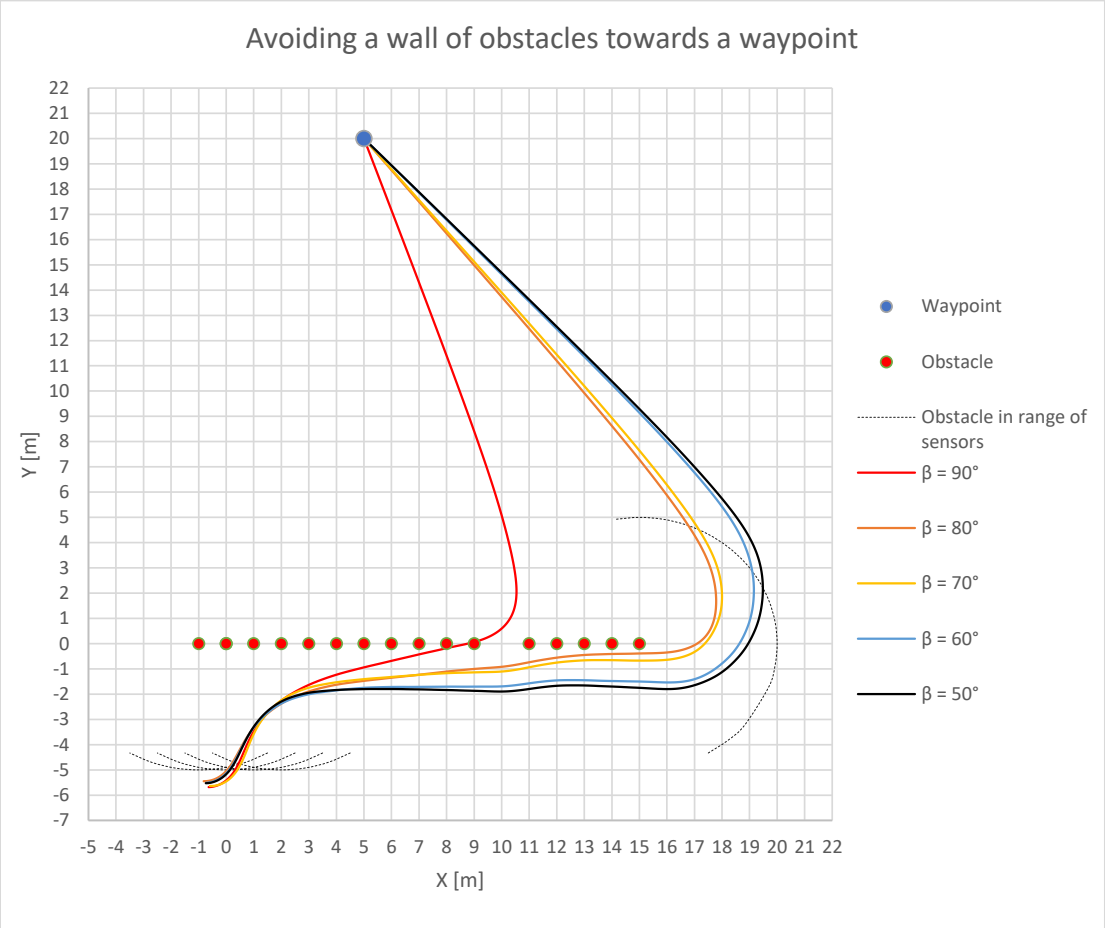


Figure 5.6: Simulation of robots heading towards a wall of obstacles, tracking a waypoint behind it. The different paths are done with different exit angle β in the avoidance mode. An anomaly is placed at [10, 0] to interfere with the choice of path. This acts as a hole in the wall.

We can see from Figure 5.6 that the exit angle β is influencing the behaviour of the robot. The anomaly in the wall is simulating a crack or hole in the wall where the robot should not enter. It can also be interpreted as a faulty sensor reading giving the robot another challenge to navigate around.

Keeping $\beta = 90°$ makes the robot collide after following the wall for a few meters. Decreasing β forces the robot to deflect from the obstacles more. For $\beta = 80°$ and $\beta = 70°$, the hole in the wall causes the robot to close in on the wall significantly. The robot is not able to keep a safe distance to the wall over a longer distance. Adjusting to $\beta = 60°$ and $\beta = 50°$ makes the robot drive at an angle outwards from the wall. Even when driving passed the hole in the wall, it initially decreasing the distance, but the robot soon drives away from the next detected obstacle.

Then the last step in verifying the functionality was to simulate how the robot will navigate in an unknown environment. A track was set by placing four waypoints in the global coordinate system. The robot is set to track one at the time. When reaching a waypoint, the navigation system switches to the next waypoint and makes the robot drive in a loop. Several obstacles were randomly placed in the environment, some obstructing a direct path between the waypoints.

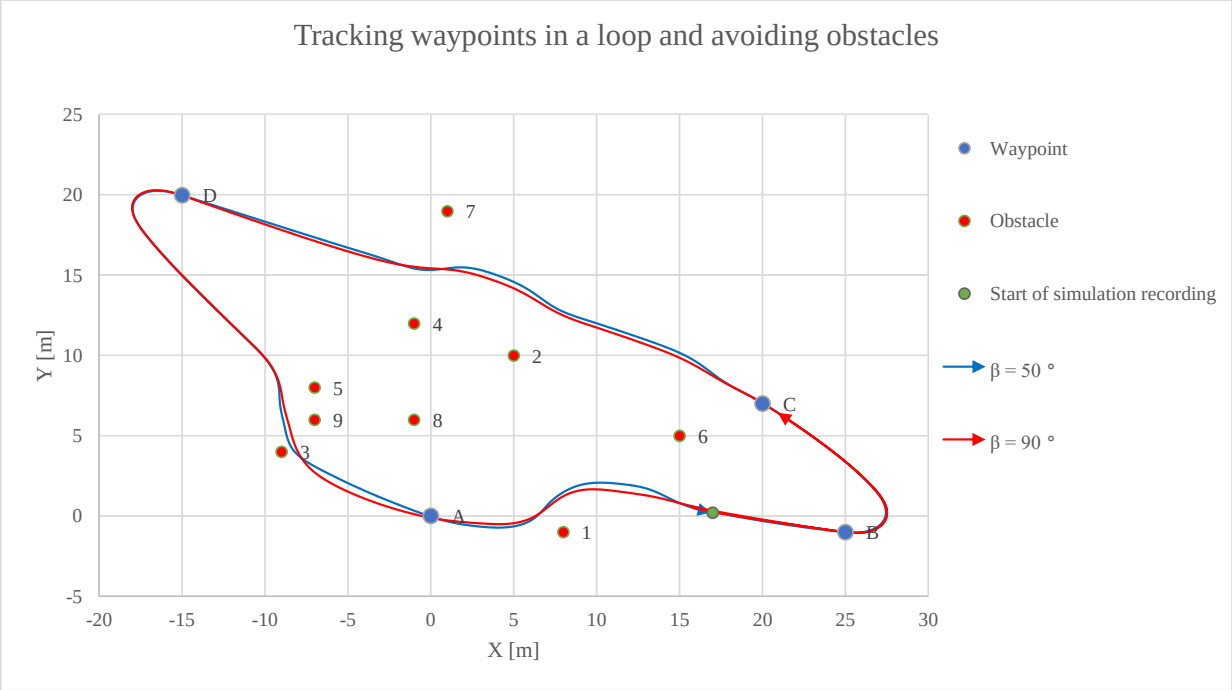
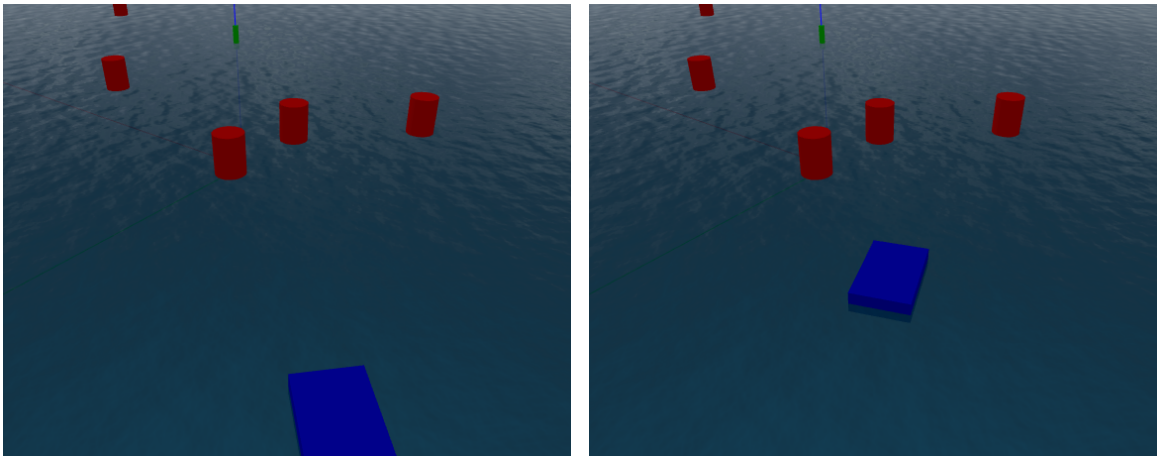


Figure 5.7: Simulation of the robot driving around counter clockwise in a loop, tracking four waypoints in sequence starting at A. Obstacles are placed to interfere or obstruct the robot in its path. Two simulations with different β angle shows the difference in behaviour.

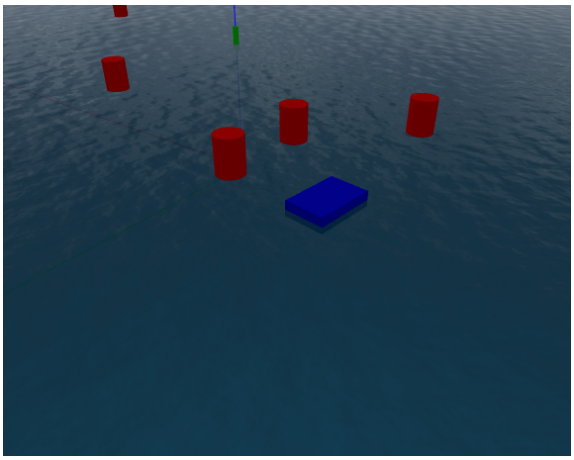
We can see from Figure 5.7 that the robot reached all waypoints going around the track. We can also see how the obstacles influence the path of the robot. Starting from waypoint A, the robot first detect obstacle 1 on its way to waypoint B. With $\beta = 50°$ the robot is taking more aggressive turns when avoiding obstacles. This can also be seen at obstacle 2, 4 and 7. Even though the obstacles are not obstructing the direct path, the sensors detect the obstacles and tries to avoid them.

We can also see that when the robot approaches obstacle 3, the distance to the obstacle is very short. Depending on the sensor accuracy and the extent of the obstacle, this might be a near miss in the real world.

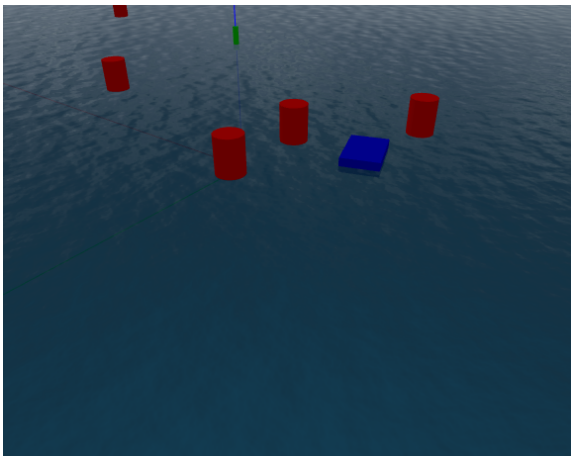


(a)

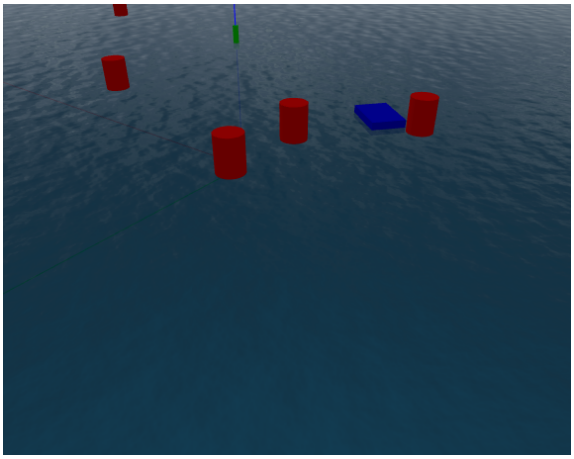
(b)



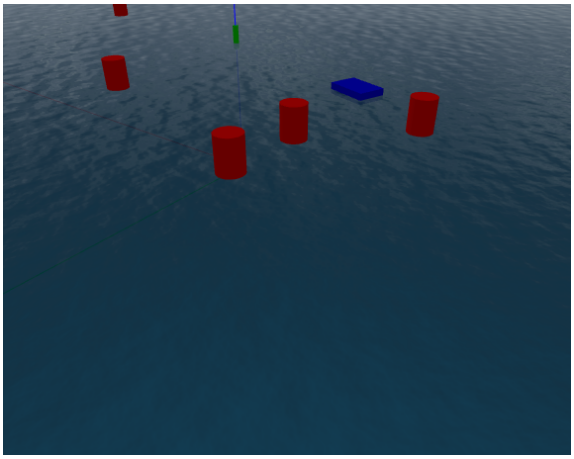
(c)



(d)



(e)



(f)

Figure 5.8: Still images of the simulation in the Gazebo environment. Starting from (a), the blue robot is heading straight for the green waypoint (A) at the top of the image, coming from (D) as seen in Figure 5.7. When the robot gets withing 5m of a red obstacle, the obstacle is detected and the robot is navigating around the obstacles.

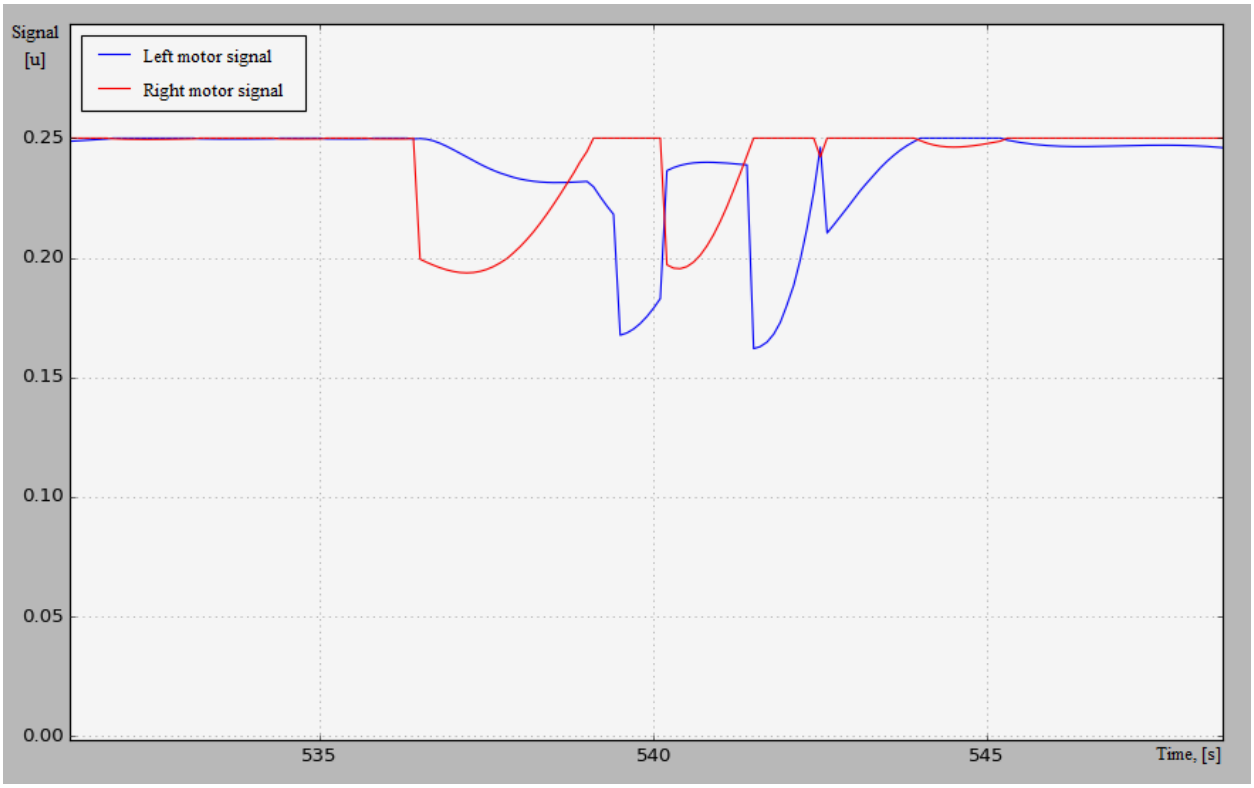


Figure 5.9: Plot of motor control signal during the avoidance sequence shown in Figure 5.8. X-axis is time in seconds and Y-axis is control signal. As the robot is driving towards the waypoint, without detecting an obstacle, both motors receive the same signal. As soon as an obstacle is detected on the left side, the right motor slows down making the robot take a right turn.

5.5 Manual override

The last step in the process is to simulate a manual override during an autonomous operation. A simulation is done using the same environment as in Figure 5.7 with exit angle $\beta = 90°$.

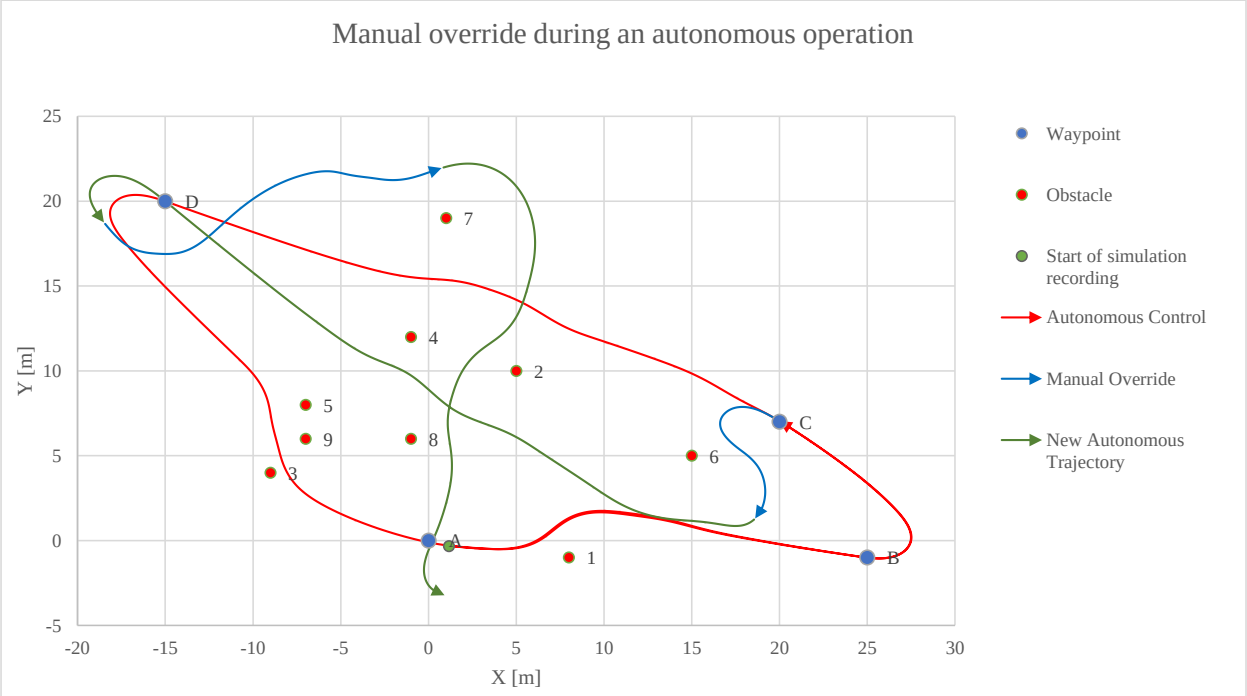


Figure 5.10: Simulation of the robot driving around counter clockwise in the same course as in Figure 5.7. The robot does one complete round before control is taken over by an operator at two occasions. The robot then navigates to the next waypoint by taking a new route.

We can see from Figure 5.10 with manual override, an operator can at any time take control of the robot and drive it top a new position. Once the operator lets go of the control, the autonomous control resumes the navigation. The robot is able to find a new path to the next waypoint.

A short demonstration video on some of the simulations can be found on YouTube on this link: https://youtu.be/dvXg_25bLAc

Chapter 6

Discussions

6.1 State of level of autonomy regarding small USVs

Part of this thesis was dedicated to explore the state of the art of USVs. Although recent research and development show improvement on the level of autonomy, the development must be separated into different parts. The majority of the autonomous systems that are closing in on becoming fully autonomous tend to be developed for medium to larger sized vessels. Whereas the smaller USVs still have their challenges.

The large USVs have the possibility to utilise complex sensor technology. The smaller USVs lack the same potential. It is necessary to develop sensor technology that is both inexpensive and accurate enough to be used in a navigation system on a small vessel. From the literature study, such a package either is not yet developed or it is not commercially available. Since the proper sensor technology has yet to be implemented in smaller USVs, a navigation system including obstacle avoidance has yet to be fully developed. However, simulating a navigation system gives the possibility to neglect the lack of accurate sensors and therefore focus on the software algorithm needed for a fully autonomous system. Adequate sensor packages can be developed in parallel and merged with software at a later stage. This applies to both a simulated model of the sensors and the real world sensor package. Implementation of more realistic sensors in the simulation would improve the accuracy of the simulation compared to a real world test.

6.2 Simulated model

The main objective of this thesis was to develop and simulate a navigation system for a small size USV. An existing simulation model was further developed. The simulation includes a robot, obstacles and waypoints. The navigation system utilises positional data from a simulated GPS, pose state from a simulated IMU and navigates through the environment. The Braitenberg algorithm was used as a basis for the navigation where the behaviour of the robot could resemble the behaviour of an animal or insect.

When navigating in an open environment tracking only one waypoint, results show that the robot quickly navigates its way towards the waypoint as in Figure 5.1. Compared to the distance of the path covered, the robot uses a small area to find the correct heading towards the waypoint. This indicates that the behaviour is aggressive and very attracted to the waypoint.

Furthermore, as expected when an obstacle is detected, the algorithm is switched to avoidance mode and the behaviour switches completely as shown in Figure 5.2. The robot shows behaviour of fear of the obstacle and the robot tends to want to escape from the location of the obstacle. This corresponds to the behaviour of animals and follows the mindset of Braitenberg through his thought experiment.

However, the method of combining these two modes is not described in Braitenberg's book [3, 19] and leaves the developer of the navigation system with the possibility for interpretation. The implemented weighting factor c seems to handle the merging of the two modes well. There is an abrupt change in behaviour where the obstacle is just within detection range at $5m$ giving a sense of surprise when the obstacle is detected in the first place. It also seemed necessary compared to have a smoother and complete linear transition between the two modes. Through experiments, having a smoother transition seemed to slow down the reaction time of the navigation system to a point where the robot often collided with the obstacle. The dynamics and the speed of the robot implied that a sharper transition was needed. The resulting behaviour shows that instead of an escaping behaviour, the robot tends to avoid and navigate around the obstacles. This behaviour is more suited for a navigation system.

Through experimentation of the Braitenberg controller, some unwanted behaviours arose. As the controller is currently configured, there is a singularity when the waypoint is located directly behind the robot, when the desired heading is exactly 180° from current heading. In this case, the two sensors will interpret the stimulus value with equal magnitude. Since the initial state of the robot is set to maximum speed forward the robot will continue to drive straight forward, away from the waypoint. However, in practice and as experienced in the simulation, this would only mean that the robot will be slower at turning towards the waypoint. Due to external forces and inaccuracy in GPS signal, the desired heading would never be exactly 180° from current heading and the robot would eventually choose a direction to turn. The effect can be seen to some extent in Figure 5.1 where the initial orientation of path 4 shows a desired heading is close to 180° from initial heading. The robot initially drives away from the waypoint, but eventually turns around heading for the waypoint.

A similar situation could occur when heading straight towards an obstacle. But for the same reasons, the heading would never be interpreted as completely dead on and the robot will choose a side to turn. As a precaution to this situation, instead of setting the weighting factor c to a one-step function, a two-step function could be implemented. A second area, closer to the obstacle, could be classified as restricted area. Here, avoidance mode would be weighted 100% if the robot is closer then i.e. $2m$ to the obstacle. In addition, the escape angle β could be set to 0°. This could prevent collision in a dead on situation.

In the end, results from Figure 5.7 show that navigating using the Braitenberg algorithm as basis, can make it possible for the robot to navigate around in an environment where only the waypoints are known. As an additional feature, should a dangerous situation occur, an operator by the ground station can always remotely take manual control of the robot. Furthermore, the control algorithm can be set up as separate nodes in ROS, where the desired functionality can be selected. A new node can be added, taking in sensor data from other type of sensors and deciding the behaviour. A new node using temperature sensors can be set to either be attracted to areas with higher temperatures or deflect from them.

Compared to the autonomy classification system ATRA in Table 1.1, the robot with the current navigation system can be characterised to have achieved level 6, with one exception regarding optimal path planning. In order for the robot to be able to autonomously choose the optimal path, knowledge of the area between current position all the way to desired position must be acquired. If the sensors meant for mapping the environment is unable to map this entire area, then the robot can only navigate using the local surroundings. The optimal path might be undetectable for the sensors.

Chapter 7

Conclusions and future work

7.1 Conclusions

In this thesis, literature study was done on the current state of the art regarding level of autonomy of smaller Unmanned Surface Vehicles (USVs). In addition, a bio-inspired Braitenberg control algorithm has been designed, implemented and simulated using a previously designed robot.

The purpose of the literature study was to gain knowledge about the current state of small USVs and how development is progressing in making fully autonomous USVs. The study shows that, although larger USVs are closing in on becoming fully autonomous, smaller USVs lack the proper sensor packages and computational power. Both suitable sensor packages and navigation software must be developed. Once an adequate sensor package can be utilised, the Braitenberg algorithm can provide a simple control algorithm and still navigate with complex behaviour.

The second purpose of this thesis was to develop and simulate a control algorithm intended for smaller USVs in order to increase the level of autonomy. As seen from the literature study, the development is highly motivated and necessary to increase the usability of USVs. The bio-inspired control algorithm from Braitenberg's thought experiment has inspired many researchers to develop autonomous land vehicles with success. However, the algorithm has not yet been used on sea going vehicles. It was therefore interesting to use the Braitenberg algorithm to explore the following:

- Simulate a USV in a 3D environment using GPS (global positioning system) and IMU (inertia motion unit) data, to be attracted to a waypoint and repelled from an obstacle.
- Develop a program that simulates the robot navigating around a track without colliding with randomly placed obstacles.
- Implement a manual override so that an operator could manually take control over the robot at any time.

The navigation software was designed using a parabola function as basis for the bio-inspired Braitenberg algorithm. The algorithm was used both as a tracking mode for finding a waypoint and as an avoidance mode for avoiding obstacles along the way. Implementing the controller step-wise showed how the behaviour of the robot changed by adding more control features. Using the control algorithm, the robot is able to track any number of waypoints, avoid obstacles, navigate parallel to obstacles resembling a wall and navigate through a course of waypoints while avoiding obstacles. Assuming a good GPS system and sensors for mapping the local area is in place, the robot should be able to autonomously navigate in closed areas like rivers and lakes, as well as on the open sea.

The simulation shows that the simple Braitenberg algorithm can give small USVs autonomous functionality.

7.2 Future work

For further development of this project the following tasks can be explored.

Hardware:

- A crucial point is to develop an adequate sensor package or software such that low-cost LiDAR can be utilised on this type of robot.
- Redesign the hull to be self-righting.

Simulation:

- The ROS/Gazebo can be further utilised. A standardised node can be created where type of sensor input is selected and the desired behaviour is selected, attraction, deflection or follow. For a research vessel, this feature could be handy where new data is to be gathered.
- Simulate a LiDAR or equivalent sensor in Gazebo and add more complex environments. Add external factors as waves, wind or water currents. LiDAR simulation could be visualised in RVIZ giving the ability to see what the robot sees.
- Moving obstacles can be added and proper anti-collision algorithm implemented.

Additional software:

A user interface should be developed giving the ability to add waypoints and show the robot position on a map. Manual control of the robot and viewing live sensor data should be possible.

References

[1] World Maritime Affairs. *Learn about Waterjet Propulsion System*. Mar. 2019. URL: https://www.worldmaritimeaffairs.com/learn-about-waterjet-propulsion-system/.

[2] Martin John Baker. *Maths - Quaternion to AxisAngle*. May 2020. URL: https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToAngle/index.htm.

[3] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. first. ISBN: 0-262-02208-7 (hard). The MIT Press, 1986.

[4] M. Caccia. "Autonomous Surface Craft: prototypes and basic research issues." In: *2006 14th Mediterranean Conference on Control and Automation*. 2006, pp. 1–6.

[5] Snehashish Chakraverty, Deepti Moyi Sahoo, and Nisha Rani Mahato. "McCulloch–Pitts Neural Network Model." In: *Concepts of Soft Computing: Fuzzy and ANN with Programming*. Singapore: Springer Singapore, 2019, pp. 167–173. DOI: 10.1007/978-981-13-7430-2_11. URL: https://doi.org/10.1007/978-981-13-7430-2_11.

[6] Yoonsuck Choe. "Hebbian Learning." In: *Encyclopedia of Computational Neuroscience*. Ed. by Dieter Jaeger and Ranu Jung. New York, NY: Springer New York, 2013, pp. 1–5. ISBN: 978-1-4614-7320-6. DOI: 10.1007/978-1-4614-7320-6_672-1. URL: https://doi.org/10.1007/978-1-4614-7320-6_672-1.

[7] M. Dunbabin and A. Grinham. "Experimental evaluation of an Autonomous Surface Vehicle for water quality and greenhouse gas emission monitoring." In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 5268–5274.

[8] Jan Faigl et al. "Syrotek a robotic system for education." In: (Jan. 2010).

[9] W. Guan et al. "Unmanned surface vessels concise robust course control based on planning and control scheme." In: *Harbin Gongcheng Daxue Xuebao/Journal of Harbin Engineering University* 40.11 (2019), pp. 1801–1808. DOI: 10.11990/jheu.201807036.

[10] Erlend Helgerud et al. *AnSweR*. Material in posession of Filippo Sanfilippo. July 2018.

[11] Farid Kendoul. "Towards a unified framework for uas autonomy and technology readiness assessment (atra)." In: *Autonomous Control Systems and Vehicles*. Springer, 2013, pp. 55–71.

[12] D.K.M. Kufoalor et al. "Autonomous maritime collision avoidance: Field verification of autonomous surface vehicle behavior in challenging scenarios." In: *Journal of Field Robotics* (2019). DOI: 10.1002/rob.21919.

[13] Akshay Chandra Lagandula. "McCulloch-Pitts Neuron—Mankind's First Mathematical Model Of A Biological Neuron." In: *Towards Data Science [online].[cit. 2019-04-25]. Dostupné z: https://towardsdatascience. com/mcculloch-pitts-model-5fdf65ac5dd1* (2019).

[14] Jan Henrik Lenes. "Autonomous online path planning and path-following control for complete coverage maneuvering of a USV." MA thesis. NTNU, 2019.

[15] Zhixiang Liu et al. "Unmanned surface vehicles: An overview of developments and challenges." In: *Annual Reviews in Control* 41 (2016), pp. 71–93.

[16] Kongsberg Geoacoustics LTD. *GEOSWATH USV*. Data Sheet. Kongsberg, Oct. 2017.

[17] Kenzo Nonami et al. “Autonomous control systems and vehicles.” In: *Intelligent Systems, Control and Automation: Science and Engineering* 65 (2013).

[18] Mark CL Patterson, Anthony Mulligan, and Fernando Boiteux. “Safety and security applications for micro-unmanned surface vessels.” In: *2013 OCEANS-San Diego*. IEEE. 2013, pp. 1–6.

[19] I. Rañó. “A model and formal analysis of Braitenberg vehicles 2 and 3.” In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 910–915.

[20] Ignacio Rano, A Gómez Eguíluz, and Filippo Sanfilippo. “Bridging the gap between bio-inspired steering and locomotion: A braitenberg 3a snake robot.” In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE. 2018, pp. 1394–1399.

[21] Iñaki Rañó. “A systematic analysis of the Braitenberg vehicle 2b for point-like stimulus sources.” In: *Bioinspiration & biomimetics* 7.3 (2012), p. 036015.

[22] Clear Path Robotics. *Heron*. 2020. URL: https://clearpathrobotics.com/heron-unmanned-surface-vessel/.

[23] Maritime Robotics. *OTTER Unmanned Surface Vehicle [USV]*. Data Sheet. Maritime Robotics, 2020.

[24] Open Robotics. *ros.org*. May 2020. URL: https://www.ros.org/.

[25] Sea Robotics. *SR Survayor M1.8*. Data Sheet. Sea Robotics Corporation, 2020.

[26] G. S. Rose, R. Pino, and Q. Wu. “A low-power memristive neuromorphic circuit utilizing a global/local training mechanism.” In: *The 2011 International Joint Conference on Neural Networks*. 2011, pp. 2080–2086.

[27] J. Rowley. “Autonomous Unmanned Surface Vehicles (USV): A Paradigm Shift for Harbor Security and Underwater Bathymetric Imaging.” In: *OCEANS 2018 MTS/IEEE Charleston*. 2018, pp. 1–6.

[28] Filippo Sanfilippo, Min Tang, and Sam Steyaert. “Aquatic Surface Robots: the State of the Art, Challenges and Possibilities.” In: *Accepted for publication to the Proc. of the 1st IEEE International Conference on Human-Machine Systems (ICHMS 2020), Rome, Italy*. 2020.

[29] Filippo Sanfilippo et al. “Perception-driven obstacle-aided locomotion for snake robots: the state of the art, challenges and possibilities.” In: *Applied Sciences* 7.4 (2017), p. 336.

[30] sea-machines.com. *Top Capabilities Sea Machines Adds to Commercial Surface Vessels*. Feb. 2020. URL: https://sea-machines.com/top-8-capabilities-sea-machines-adds-to-commercial-surface-vessels.

[31] Thomas Stenersen. “Guidance system for autonomous surface vehicles.” MA thesis. NTNU, 2015.

[32] Changhyuck Sung, Hyunsang Hwang, and In Kyeong Yoo. “Perspective: A review on memristive hardware for neuromorphic computation.” In: *Journal of Applied Physics* 124.15 (2018), p. 151903.

[33] Min Tang. “Digital Twin of an Aquatic Surface Robot with ROS and Gazebo.” MA thesis. University of South-Eastern Norway, Nov. 2019.

[34] Naval Technology. *Sea Hunter ASW Continuous Trail Unmanned Vessel (ACTUV)*. 2016. URL: https://www.naval-technology.com/projects/sea-hunter-asw-continuous-trail-unmanned-vessel-actuv/.

[35] theexplorer.no. *The world’s first zero-emission autonomous container ship*. May 2020. URL: https://www.theexplorer.no/solutions/yara-birkeland--the-worlds-first-zero-emission-autonomous-container-ship/.

[36] tu.no. *Yara Birkeland: Leveres fra Vard Brattvaag etter sommeren*. Feb. 2020. URL: https://www.tu.no/artikler/yara-birkeland-leveres-fra-vard-brattvaag-etter-sommeren/484547.

[37] Lhotský Vojtěch. *Demonstration Tasks for the SyRoTek System*. 2014. URL: https://cyber.felk.cvut.cz/theses/detail.phtml?id=516.

[38] Cong Wang et al. "A Braitenberg Vehicle Based on Memristive Neuromorphic Circuits." In: *Advanced Intelligent Systems* (2019). DOI: 10.1002/aisy.201900103.

[39] H. Xu et al. "Deep reinforcement learning-based path planning of underactuated surface vessels." In: *Cyber-Physical Systems* 5.1 (2019), pp. 1–17. DOI: 10.1080/23335777.2018.1540018.

Appendix A

Code with explanations

The ROS node of the Braitenberg controller consists of a script written in Python. The node must first be initialised before subscribing to the ROS topic where sensor data is published. The sensor data on position and orientation of the robot is published on the ROS topic *p3d_odom* on the message *pose*.

```
def Braitenberg():
  rospy.init_node('controllerNode', anonymous=True)
  rate = rospy.Rate(10)

  rospy.Subscriber('p3d_odom',Odometry,navCallback)

  while not rospy.is_shutdown():

    rate.sleep()

if __name__ == '__main__':

  try:
    Braitenberg()
  except rospy.ROSInterruptException:
    pass
```

Listing A.1: Initial setup of node.

This gives the node a unique anonymous ID number and sets the refresh rate of the node to 10Hz.

Then the pose data can be read using the *navCallback* function.

```
def navCallback(msg):
    Xb = msg.pose.pose.position.x
    Yb = msg.pose.pose.position.y

    Quat_x = msg.pose.pose.orientation.x
    Quat_y = msg.pose.pose.orientation.y
    Quat_z = msg.pose.pose.orientation.z
    Quat_w = msg.pose.pose.orientation.w
```

Listing A.2: Reading sensor position data.

The x and y position data is given in meters and defines the position of the robot in the global frame. The position in z axis is neglected as the robot is floating on a flat water without any wind or wave disturbance. The orientation however is given in Quaternion rotation coordinates. This is a three dimensional rotational system that is cumbersome to use if only a two dimensional system is evaluated. The Quaternions are therefore converted to Cartesian angle magnitude [2].

```
def quat2angle(qx, qy, qz, qw):
  angle = 2*math.acos(qw)
  s = math.sqrt(1-qw**2)
  x = (angle*qx/s)*180/math.pi
  y = (angle*qy/s)*180/math.pi
  z = (angle*qz/s)*180/math.pi
  return (x, y, z)

pose_angle = quat2angle(Quat_x, Quat_y, Quat_z, Quat_w)

alpha_0_temp = pose_angle[2]
```

Listing A.3: Calculate rotation angle.

Then *pose_angle*[2] gives the pose angle of the robot around the z axis in degrees with reference to the x axis of the global frame. The Quaternion system gives the pose angle in the range from $-360°$ to $+360°$. This is converted to a range of $0° - 360°$ before giving the actual angle α_0. This angle is used as the heading of the robot in the global frame.

Next, the position of the two sensors mounted on the robot is calculated before passing the positions to the stimulus function. Here shown with the code for tracking mode.

```
Rx=Xb+(L1/2)*math.cos(math.radians(alpha_R))
Ry=Yb+(L1/2)*math.sin(math.radians(alpha_R))
Lx=Xb+(L1/2)*math.cos(math.radians(alpha_L))
Ly=Yb+(L1/2)*math.sin(math.radians(alpha_L))

Sb=stimvalue(Xb, Yb, Xwp, Ywp)
SR=stimvalue(Rx, Ry, Xwp, Ywp)
SL=stimvalue(Lx, Ly, Xwp, Ywp)
```

Listing A.4: Calculate the robot sensor positions in the global frame.

This defines the position of the sensors with respect to the global frame. When passing these position along with the coordinates of the next waypoint at $[X_{wp}, Y_{wp}]$ to the stimulus function, it returns the stimulus value S with respect to the waypoint.

```
def stimvalue(Xb, Yb, Xwp, Ywp):
  S=g0 -a*(Xwp-Xb)**2 -b*(Ywp-Yb)**2
  return S
```

Listing A.5: Calculating stimulus value.

The values of S is the used to calculate the amount of control signal that should be subtracted from each of the motors. Here shown with the code for tracking mode.

```
if SR < SL:
  U_RW= (SR-Sb) / (2*Dwp)
  U_LW= 0
else:
  U_RW= 0
  U_LW= (SL-Sb) / (2*Dwp)
```

Listing A.6: Calculating control signal using the stimulus values

The equations of listing A.6 can give high values of control signal when the distance to the waypoint is long. The signal is therefore limited to the maximum setting of the motors.

```
if U_RW < -u_max:
  U_RW = -u_max

if U_LW < -u_max:
  U_LW = -u_max
```

Listing A.7: Setting maximum control signal.

If an obstacle is detected, then the script also calculates the stimulus values in avoidance mode.

```
if Dc < Os:
  print "Obstacle detected"

  #Equations for avoidance mode

else:
  U_RO = 0
  U_LO = 0
  c = 1
```

Listing A.8: If an obstacle is detected.

The equations used in avoidance mode are described in 3.3.2. They are set up in a similar way to the equations in tracking mode. When the controller is running in avoidance mode, only the closes obstacle is taken into account. This distance is D_c and will change to whatever obstacle is closest.

The two control signals from tracking mode and avoidance mode are then combined. Since the control values U are negative, they are added with plus sign when combined.

```
ur = u_max + c*U_LW + (1-c)*U_RO
ul = u_max + c*U_RW + (1-c)*U_LO
```

Listing A.9: Combining tracking mode and avoidance mode.

The last step is to publish the control signal to the message which in the end drives the motors.

```
drivemsg.right = ur
drivemsg.left  = ul

publisher.publish(drivemsg)
```

Listing A.10: Publish to drivemsg.

A list of the parameters are defined at the start of the script together with a list of waypoints and a list of obstacles.

```
Nwp=0 #Setting next waypoint to the first in the list

L1=0.2 #Distance between sensors [m]

#Constants in stimulus function
g0=0
a=1
b=a

u_max=0.25 #Max speed of the motors [unitless]

Dacc=0.8 #Accuracy when reaching waypoint [m]

Os=5 #Range of obstacle avoidance sensor [m]

w=100 #Field of view of the sensor [deg]

beta=90 #Exit angle from obstacle [deg]
```

Listing A.11: Parameters

The parameter u_{max} sets the maximum speed of the motors. This is a unitless value but can be normalised to a specified speed when compared to the real life robot.

```
wp1 = [0.0, 0.0]
wp2 = [25.0, -1.0]
wp3 = [20.0, 7.0]
wp4 = [-15.0, 20.0]

wps = [wp1, wp2, wp3, wp4]
```

Listing A.12: Defining position of waypoints

```
obstacles = []
obstacles.append(("o1", 8.0, -1.0));
obstacles.append(("o2", 5.0, 10.0));
obstacles.append(("o3", -9.0, 4.0));
obstacles.append(("o4", -1.0, 12.0));
obstacles.append(("o5", -7.0, 8.0));
obstacles.append(("o6", 15.0, 5.0));
obstacles.append(("o7", 1.0, 19.0));
obstacles.append(("o8", -20.0, 25.0));
obstacles.append(("o9", -1.0, 6.0));
obstacles.append(("o10", -7.0, 6.0));
```

Listing A.13: Defining position of obstacles

Using the ROS command *rqt_graph* shows the ROS environment when the Braitenberg controller is running.

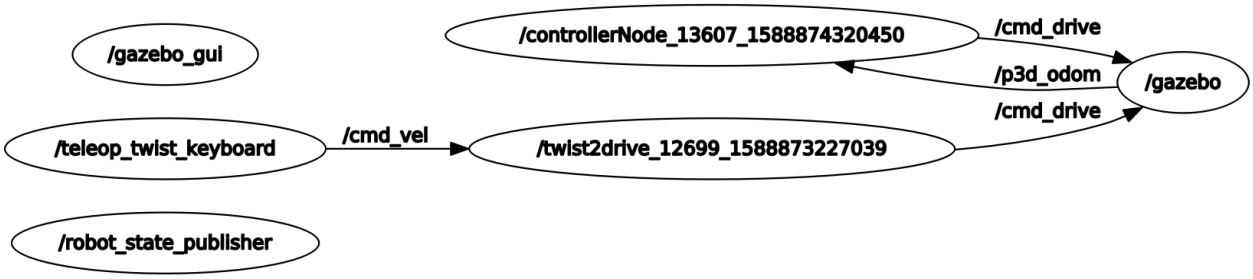


Figure A.1: rqt_graph of ROS environment

Appendix B

Complete Code of Braitenberg Controller

```
1
2  #!/usr/bin/env python
3  '''
4  Using instances of the pypid Pid class to control yaw and velocity
5  '''
6  # Python
7  import sys
8  import math
9  import numpy as np
10
11 # ROS
12 import rospy
13 import tf
14 from nav_msgs.msg import Odometry
15 from asr_msgs.msg import Drive
16
17
18 #defining waypoints:
19 wp1 = [0.0, 0.0]
20 wp2 = [25.0, -1.0]
21 wp3 = [20.0, 7.0]
22 wp4 = [-15.0, 20.0]
23
24 wps = [wp1, wp2, wp3, wp4]
25
26 #defining obstacles:
27 obstacles = []
28 obstacles.append(("o1", 8.0, -1.0));
29 obstacles.append(("o2", 5.0, 10.0));
30 obstacles.append(("o3", -9.0, 4.0));
31 obstacles.append(("o4", -1.0, 12.0));
32 obstacles.append(("o5", -7.0, 8.0));
33 obstacles.append(("o6", 15.0, 5.0));
34 obstacles.append(("o7", 1.0, 19.0));
35 obstacles.append(("o8", -20.0, 25.0));
36 obstacles.append(("o9", -1.0, 6.0));
37 obstacles.append(("o10", -7.0, 6.0));
38
39 #setting next waypoint to the first in the list
40 Nwp=0
41
42 #distance between sensors [m]
43 L1=0.2
44
45 #constants in stimulus function
46 g0=0
```

```
a=1
b=a

#Max speed of the motors [?]
u_max=0.25

#Accuracy when reaching waypoint [m]
Dacc=0.8

#Range on obstacle avoidance sensor, for example 5 [m]
Os=5

#Field of view of the sensor [deg]
w=100

#Angle of deflection from obstacle [deg]
beta=90



#Stimulus function
def stimvalue(Xb, Yb, Xwp, Ywp):
  S=g0 -a*(Xwp-Xb)**2 -b*(Ywp-Yb)**2
  return S

#convert from quaterinon coordinates to angle magnetude
def quat2angle(qx, qy, qz, qw):
  angle = 2*math.acos(qw)
  s = math.sqrt(1-qw**2)
  x = (angle*qx/s)*180/math.pi
  y = (angle*qy/s)*180/math.pi
  z = (angle*qz/s)*180/math.pi
  return (x, y, z)

#Euclidean distance to next waypoint:
def distance(x1, y1, x2, y2):
  dx = x1 - x2
  dy = y1 - y2
  return math.sqrt((dx**2+dy**2))

def angle(x1, y1, x2, y2):
  a_temp = math.atan2(y1-y2,x1-x2)*180/math.pi
  if a_temp < 0:
    a = 360+a_temp
  else:
    a = a_temp
  return a

def navCallback(msg):
  publisher = rospy.Publisher('cmd_drive',Drive,queue_size=1)
  drivemsg = Drive()

  Xb=msg.pose.pose.position.x
  Yb=msg.pose.pose.position.y

  print ""
  print ""

  print "Robot position: [x = %s, y = %s]"%('%.2f'%Xb, '%.2f'%Yb)

  global Nwp

  numberofwaypoints = len(wps)
  numberofobstacles = len(obstacles)


  #reset waypoint counter when last waypoint is reached
```

```
114   if Nwp > numberofwaypoints-1:
115     Nwp = 0
116
117   #print "Next WayPoint: %s"%(wps[Nwp])
118
119   goal = wps[Nwp]
120   Xwp = goal[0]
121   Ywp = goal[1]
122
123   Dwp = distance(Xb, Yb, Xwp, Ywp)
124
125   #print "Distance to next WayPoint [%s]"%('%.2f'%Dwp)
126
127   print "Next Waypoint is WP%s at %s, [%s] meters away"%(Nwp+1, wps[Nwp], '%.2f'%
        Dwp)
128
129   #Finding distance to closest obstacle:
130   closest_ob = None
131   Dc = None #Distance to closest obstacle
132   for o_name, l_x, l_y in obstacles:
133     dist=distance(Xb, Yb, l_x, l_y)
134     if Dc is None or dist < Dc:
135       closest_ob = o_name
136       Dc = dist
137       Xobs=l_x
138       Yobs=l_y
139
140
141
142
143   Quat_x=msg.pose.pose.orientation.x
144   Quat_y=msg.pose.pose.orientation.y
145   Quat_z=msg.pose.pose.orientation.z
146   Quat_w=msg.pose.pose.orientation.w
147
148   pose_angle = quat2angle(Quat_x, Quat_y, Quat_z, Quat_w)
149
150   alpha_0_temp = pose_angle[2]
151
152
153   #set angle range 0-360, instead of -360 -> +360
154   if alpha_0_temp < 0:
155     alpha_0 = 360+alpha_0_temp
156   else:
157     alpha_0 = alpha_0_temp
158
159
160   #Coordinates of sensors
161   alpha_R=alpha_0-90
162   alpha_L=alpha_0+90
163
164   theta = angle(Xb, Yb, Xwp, Ywp)
165
166   gamma_wp_temp = -(alpha_0 - (180 + theta))
167
168   if gamma_wp_temp > 180:
169     gamma_wp = -(alpha_0 + (180 - theta))
170   else:
171     gamma_wp = gamma_wp_temp
172
173   print "Angle to rotate (gamma_wp): [%s]"%('%.1f'%gamma_wp)
174
175
176
177   #avoidance stimuli:
178   if Dc < Os:
179     print "Obstacle detected"
```

```
    print "Closest obstacle is %s, [%s] meters away"%(closest_ob, '%.2f'%Dc)

    #angle towards obstacle, global
    theta_O=angle(Xb, Yb, Xobs, Yobs)

    #angle towards obstacle in regards to heading of robot
    gamma_O_temp= -(alpha_O - (180 + theta_O))

    if gamma_O_temp > 180:
      gamma_O = -(alpha_O + (180 - theta_O))
    else:
      gamma_O = gamma_O_temp

    #Turning heading 90 degrees
    if gamma_O < 0:
      y=beta
    else:
      y=-beta

    Rx=Xb+(L1/2)*math.cos(math.radians(alpha_R+y))
    Ry=Yb+(L1/2)*math.sin(math.radians(alpha_R+y))
    Lx=Xb+(L1/2)*math.cos(math.radians(alpha_L+y))
    Ly=Yb+(L1/2)*math.sin(math.radians(alpha_L+y))

    Sb=stimvalue(Xb, Yb, Xobs, Yobs)
    SR=stimvalue(Rx, Ry, Xobs, Yobs)
    SL=stimvalue(Lx, Ly, Xobs, Yobs)

    #convert stim value to control signal
    if SR < SL:
      U_RO= (SR-Sb) / (2*Dc)
      U_LO= 0
    else:
      U_RO= 0
      U_LO= (SL-Sb) / (2*Dc)

    #limit signal to v_max
    if U_RO < -u_max:
      U_RO = -u_max

    if U_LO < -u_max:
      U_LO = -u_max

    #ignore obstacle if it is behind the robot
    if abs(gamma_O) > w:
      c=1
    else:
      c=Dc/(2*Os)


  else:

    U_RO = 0
    U_LO = 0
    c = 1

  print c

  #tracking stimuli:
  Rx=Xb+(L1/2)*math.cos(math.radians(alpha_R))
  Ry=Yb+(L1/2)*math.sin(math.radians(alpha_R))
  Lx=Xb+(L1/2)*math.cos(math.radians(alpha_L))
  Ly=Yb+(L1/2)*math.sin(math.radians(alpha_L))

  Sb=stimvalue(Xb, Yb, Xwp, Ywp)
  SR=stimvalue(Rx, Ry, Xwp, Ywp)
  SL=stimvalue(Lx, Ly, Xwp, Ywp)
```

```
247   #print "Stim values: [SL = %s, Sb = %s, SR = %s]"%('%.3f'%SL, '%.3f'%Sb, '%.3f'%
        SR)
248
249   if SR < SL:
250     U_RW= (SR-Sb) / (2*Dwp)
251     U_LW= 0
252   else:
253     U_RW= 0
254     U_LW= (SL-Sb) / (2*Dwp)
255
256
257   #print "Motor subtractions 1: [V_RW = %s, V_LW = %s]"%('%.3f'%V_RW, '%.3f'%V_LW)
258
259   if U_RW < -u_max:
260     U_RW = -u_max
261
262   if U_LW < -u_max:
263     U_LW = -u_max
264
265
266
267   #motor setpoints:
268   ur = u_max + c*U_LW + (1-c)*U_RO
269   ul = u_max + c*U_RW + (1-c)*U_LO
270
271
272   drivemsg.right = ur
273   drivemsg.left  = ul
274
275   publisher.publish(drivemsg)
276
277   print "Motor setpoints: [ur = %s, ul = %s]"%('%.3f'%ur, '%.3f'%ul)
278
279   if Dwp < Dacc:
280     Nwp = Nwp + 1
281
282
283
284 def Braitenberg():
285   rospy.init_node('controllerNode', anonymous=True)
286   rate = rospy.Rate(10)
287
288   rospy.Subscriber('p3d_odom',Odometry,navCallback)
289
290   while not rospy.is_shutdown():
291
292     rate.sleep()
293
294 if __name__ == '__main__':
295
296   try:
297     Braitenberg()
298   except rospy.ROSInterruptException:
299     pass
300
```

Listing B.1: Braitenberg controller node.

Appendix C

Digital twin ROS/Gazebo Setup

This is a guide on how to set up ROS and Gazebo on an already running linux PC.

Create the workspace on the ground station by entering each of the lines of code, one by one:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin init workspace
catkin_make
```

To avoid repeating source on the workspace when a new terminal is opened, do

```
sudo gedit ~/.bashrc
```

The .bashrc is opened. Add the following content in the end of the bashrc-file

```
source ~/catkin_ws/devel/setup.bash
```

save and exit.

Go to /catkin_ws/src

Download teleop_twist_keyboard plugin by

```
git clone https://github.com/ros-teleop/teleop_twist_keyboard.git
```

Copy the ASR files to the /catkin_ws/src folder from Bitbucket either download direclty from https://bitbucket.org/dtofasr/asr/src/master/ or use git command (Repo has to be cloned to your own bitbucket account. Change USERNAME to your bitbucket account name)

```
git clone https://USERNAME@bitbucket.org/dtofasr/asr.git
```

and then compile by

```
catkin_make
```

Open 4 terminals using tmux and start roscore in 1st terminal

```
roscore
```

Spawn the robot model in gazebo in 2nd terminal by

```
roslaunch asr_gazebo base_gazebo.launch
```

Start the node twist2drive in 3rd terminal by

```
rosrun asr_utils twist2drive.py
```

Start the node teleop_twist_keyboard in 4th terminal by

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Now click on the teleop terminal window and use the keyboard to drive the ASR.

u i o (full speed on right motor / both motors / left motor)

j k l (slow speed on right motor / stop / slow speed on left motor)

m , . (slow speed backwards right motor / slow backwards / slow backwards left motor)

Appendix D

Setup of the Braitenberg controller node

Follow instructions from digital twin.

Install 3 missing packages. Find them on the ROS wiki as the catkin_make errors occur.

Install pypid (several versions of pypid, this is the correct one)

```
cd ~/WorkingCopies (or repos or downloads)
git clone git@github.com:bsb808/pypid.git
cd pypid
sudo python setup.py develop
```

Use tmux to open several terminal windows.

Launching simulator in 1st terminal:

```
roscore
```

Start Gazebo in the 2nd terminal:

```
roslaunch asr_gazebo base_gazebo_launch.launch
```

In terminal 3: To drive the bot with keyboard (in addition to the controller):

```
rosrun asr_utils twist2drive.py
```

Then in terminal 4:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
(click inside this terminal to control boat)
```

In terminal 5: Run the Braitenberg controller node

```
rosrun asr_control braitenberg_control_loop.py
```

In terminal 6: To spawn obstacles, change directory into where the urdf file is located (/catkin_ws/src/asr/asr_

```
rosrun gazebo_ros spawn_model -file Obstacle.urdf -urdf -z 5 -model obstacle
```

In the same terminal, spawn waypoints:

```
rosrun gazebo_ros spawn_model -file Waypoints.urdf -urdf -z 5 -model waypoint
```

To record posision data: use rosbag data logger. (ex: <filename> = 1_1)

```
mkdir ~/bagfiles
cd ~/bagfiles
rosbag record -O <filename> /p3d_odom
```

Press ctrl-c in the terminal to stop recording.

Convert .bag file to .csv format. (ex: <bag_file_name.bag> = 1_1.bag). Change directory into folder with bag files that was recorded.

```
rostopic echo /p3d_odom -b <bag_file_name.bag> -p > <csv_file_name.csv>
```

View bag file in PLotJuggler. Open PlotJuggler and go to

File -> Load Data. Choose your bag file

Choose which messages to import, cmd_drive, fix_velocity, p3d_odom etc. Drag and Drop variables to plot. Select both x and y with shift, drag and drop, to plot position as x-y plot.

To stop roscore during an error:

```
killall -9 roscore
```

All controller nodes of the different steps in the result can be found in:

```
~/catkin_ws/src/asr/asr_control/nodes
```

All urdf models of waypoints and obstacles can be found in:

```
~/catkin_ws/src/asr/asr_gazebo/urfd
```