NTNU

Norwegian University of
Science and Technology

Prototype of HIL Test Platform for Autonomous USV

Simulation and Visualization of Vessel Surroundings

Kjetil Svae Børs-Lind

Master of Science in Cybernetics and Robotics
Submission date: July 2017
Supervisor: Morten Breivik, ITK
Co-supervisor: Rein Anders Apeland, Kongsberg Maritime

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

"What is real? How do you define 'real'? If you're talking about what you can feel, what you can smell, what you can taste and see, then 'real' is simply electrical signals interpreted by your brain."

- Morpheus, The Matrix

This thesis contains the details of my master's project conducted in the spring of 2017, and concludes my degree in Engineering Cybernetics at the Norwegian University of Science and Technology. I am grateful for the knowledge and skills I have acquired through the 5 years of interesting courses and intense studying.

The thesis considers the simulation of the maritime surroundings of Unmanned Surface Vehicles (USVs) in a Hardware-In-the-Loop (HIL) setup for verification tests of the USVs' control systems. The project was conducted under instruction from Kongsberg Maritime AS, who, in cooperation with the Norwegian Defense Research Establishment (FFI), develops the USVs. In cooperation with Even Ødegaard, who considers the modeling and simulation of the dynamics of the vehicles in the loop, the project resulted in the first prototype of a HIL simulator for the USVs. It is hoped that the simulator will be of use in the ongoing USV project of Kongsberg Maritime and FFI.

I would like to thank my supervisor at NTNU, Morten Breivik, for frequent meetings and feedback throughout the spring. I would also like to thank Rein Anders Apeland from Kongsberg Maritime for his guidance and enthusiasm for the project. Lastly, I would like to thank my fellow student Even Ødegaard for the satisfying cooperation and useful discussions during the entirety of the project.

Trondheim, July 6, 2017

Kjetil Svae Børs-Lind

Summary

This thesis considers the simulation and visualization of the maritime surroundings of unmanned surface vehicles (USVs) in a Hardware-In-the-Loop (HIL) simulator. The development of the simulator was conducted as two master's projects, and resulted in the first prototype of a HIL simulator meant for verification testing of two USVs developed by Kongsberg Maritime AS in cooperation with Norwegian Defense Research Establishment (FFI). The two USVs are called Odin and Jolner, where Odin is the primary target of the HIL simulator. The simulator consists of 3 main software modules: the Surroundings simulator, the Dynamics simulator and a graphical user interface (GUI), where the modeling and simulation of the dynamics of the USVs are considered in Ødegaard (2017).

The desired sensor systems to simulate in the Surroundings simulator are an Automatic Identification System (AIS) and a dedicated target detection module (TDM), which is a sensor fusion of AIS, radar and Light Detection and Ranging (LiDAR) sensors. An investigation of these sensor systems and how they can be simulated with appropriate levels of sensor noise are considered. A basic sensor fusion algorithm is applied to artificial data from AIS, radar and LiDAR to make an educated guess in regards to the performance of the TDM, which is currently under development at FFI.

Possible simulation scenarios and objects to include in the simulations are discussed, with emphasis on usefulness in regards to the current development stage of Odin. The Surroundings simulator is implemented based on this discussion and the investigation of the relevant sensor systems. A GUI is also implemented, with real-time plots and visualization of the simulation in both 2D and 3D.

The functionality of the HIL simulator in its entirety is demonstrated through an example, simulating Odin in an open sea scenario under presence of icebergs and other vessels. Odin navigates in the simulated environment using thrust inputs as logged from a real, unrelated mission. Unfortunately, the control system and the TDM of Odin is still under development, so the Surroundings simulator could not be tested with the real control system in the loop. It is, however, implemented simple motion control, which enabled testing of the Dynamics simulator against Odin's control system. This is proven successful in Ødegaard (2017), and indicates that the interface between the

simulator and the control system works as intended. It is hence expected that the simulator in its entirety will provide a complete and functional HIL simulator platform for Odin and Jolner when their respective control systems are ready to receive data from the sensor systems considered in this thesis.

Sammendrag

Denne avhandlingen tar for seg simuleringen og visualiseringen av de maritime omgivelsene til ubemannede overflatefartøy (USVer) i en Hardware-In-the-Loop (HIL) - simulator. Utviklingen av simulatoren ble gjennomført som to masteroppgaver og resulterte i den første prototypen av en HIL-simulator ment for verifikasjonstesting av to USVer utviklet av Kongsberg Maritime AS i samarbeid med Forsvarets Forskningsinstitutt (FFI). De to USVene heter Odin og Jolner, hvor Odin er primærmålet for HIL-simulatoren. Simulatoren består av 3 programvaremoduler: Omgivelsessimulatoren, Dynamikksimulatoren og et grafisk brukergrensesnitt (GUI), hvor modelleringen og simuleringen av USVenes dynamikk er behandlet i Ødegaard (2017).

Sensorsystemene som er ønsket simulert i Omgivelsessimulatoren er et Automatisk identifikasjonssystem (AIS) og en dedikert mål-deteksjons-modul (TDM), som er en fusjon av data fra AIS, radar og en optisk fjernmåler (LiDAR). Detaljer om disse sensorsystemene og hvordan de kan simuleres med passende nivåer av usikkerhet er diskutert. En grunnleggende sensorfusjonsalgoritme blir anvendt på fiktive data fra AIS, radar og LiDAR for å kunne gjøre et kvalifisert estimat på ytelsen man kan forvente av TDMen som fortsatt er under utvikling hos FFI.

Mulige simuleringsscenarioer og objekter man kan inkludere i simuleringene blir diskutert med vekt på nytteverdi i sammenheng med det nåværende utviklingsstadiet til Odin. Omgivelsessimulatoren blir implementert basert på denne diskusjonen og utredningen om de relevante sensorsystemene. Et GUI blir også implementert, med sanntidsgrafer og mulighet for visualisering av simuleringen i både 2D og 3D.

Funksjonaliteten til HIL-simulatoren i sin helhet blir demonstrert gjennom et konkretisert eksempel som simulerte Odin i et åpent hav-scenario med tilstedeværelse av isfjell og andre fartøyer. Odin navigerer i de simulerte omgivelsene ved hjelp av pådragskommandoer loggført fra et urelatert virkelig oppdrag. Odins kontrollsystem er fortsatt under utvikling, så omgivelsessimulatoren kunne dessverre ikke testes med det virkelige kontrollsystemet i loopen. Bevegelsesstyring er imidlertid allerede implementert, noe som legger til rette for testing av Dynamikksimulatoren opp mot Odin. Dette viser seg å fungere som det skal i Ødegaard (2017), noe som indikerer at grensesnittet mellom simulatoren og styringssystemet virker som tilsiktet. Det er derfor ventet

at simulatoren i sin helhet vil utgjøre en komplett og funksjonell HIL-platform for Odin og Jolner når styringssystemene er klare til å motta data fra sensorsystemene i denne avhandlingen.

Contents

List of Tables

List of Figures

List of Abbreviations

AIS	Automatic Identification System
API	Application Programming Interface
AUV	Autonomous Underwater Vehicle
COG	Course Over Ground
COLREGs	International Regulations for Preventing Collisions at Sea
DOF	Degree Of Freedom
DP	Dynamic Positioning
EKF	Extended Kalman Filter
ENC	Electronic Nautical Chart
FFI	Norwegian Defense Research Establishment (Forsvarets Forskningsinstitutt)
GNC	Guidance Navigation and Control
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
HIL	Hardware In the Loop
HW	Hardware
IMU	Inertial Measurement Unit
LAN	Local Area Network

LiDAR	Light Detection And Ranging
MMSI	Maritime Mobile Service Identity
NMEA	National Marine Electronics Association
OS	Operating System
RCS	Radar Cross Section
ROS	Robot Operating System
SMI	Special Maneuver Indicator
SOG	Speed Over Ground
SW	Software
TDM	Target Detection Module
USV	Unmanned Surface Vehicle
VHF	Very High Frequency

Chapter 1

Introduction

1.1 Background

For inspection, surveillance and intelligence operations at sea, the use of smaller autonomous vehicles are considered particularly useful. The Autonomous Underwater Vehicles (AUVs) Hugin and Munin, developed by Kongsberg Maritime, have already achieved great success in the maritime market and are sold to a range of companies worldwide. Today, Hugin is used in a variety of operations, ranging from seabed mapping, reconnaissance and mine countermeasures, and is considered to be among the leading technologies within these fields.

To maintain the position as a leading maritime technology company, Kongsberg Maritime has entered a cooperation with the Norwegian Defense Research Establishment (FFI) to develop a prototype of an unmanned surface vehicle (USV) for mine countermeasures and other unmanned operations at sea. The USV is under development at FFI in Horten, and is at the current time able to follow a simple waypoint plan and to navigate around nearby obstacles. It is 10.9 meters long and driven by two vectorized Hamilton water-jet engines of 225 horse powers each, which allows for fast and flexible maneuvering.

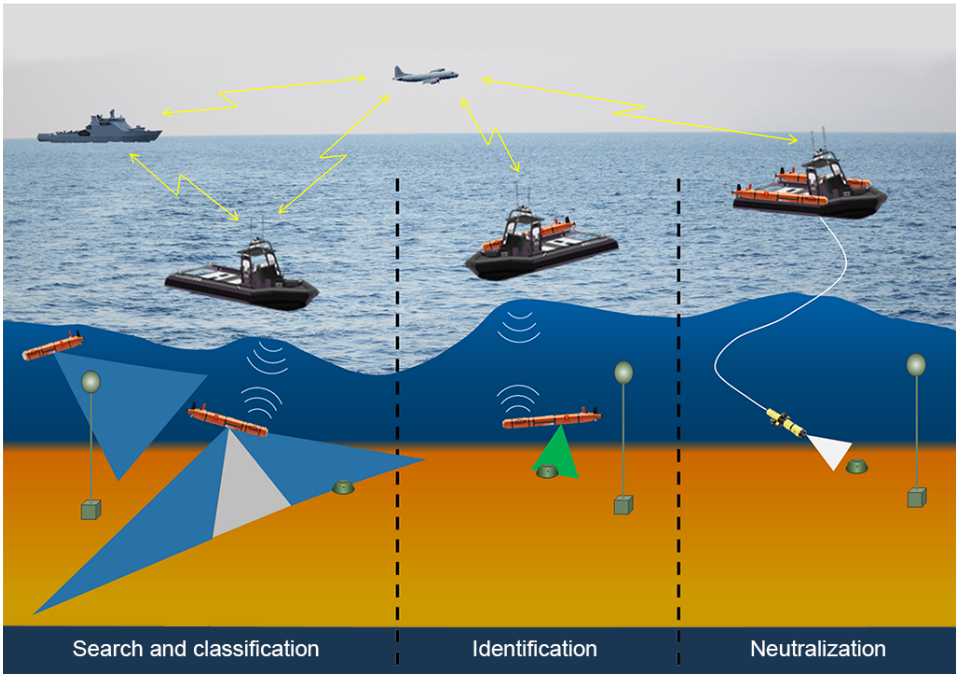

Figure 1.1: Illustration of the USV Odin during a fictitious mine search and neutralization mission working together with the AUVs Hugin and Munin. Photo adapted from FFI.

One of the future goals is for the USV to carry and perform launch and recovery of the AUVs Hugin and Munin to further reduce the need for human intervention during mine countermeasures. Hence, the USV is given the suitable name Odin, after the Norse god who accompanied the ravens Hugin and Munin and received the information they collected from all over Midgard. An illustration of how Odin is planned to work with the AUVs can be seen in Figure 1.1.

In the summer of 2016, Kongsberg Maritime, in cooperation with FFI, facilitated a student summer project called Survey Explorer. The goal of the project was to develop a control system and integrate the necessary sensors on a small USV to perform autonomous seabed mapping. This was achieved with great success, and the USV will continue to function as a development platform for future student summer projects. The USV was given the name Jolner, and can be seen in Figure 1.2. In the future, Jolner might be introduced to the market as a cheaper, less advanced version of Odin.

Figure 1.2: The student team working on Jolner throughout the Survey Explorer summer project of 2016. From the left: Jørgen Apeland, Even Ødegaard, Rune Nordmo, Mariusz Eivind Grøtte, Peder Aaby, Kjetil Børs-Lind. Courtesy of Kongsberg Gruppen.

1.2 Motivation

For a USV to be able to navigate autonomously at sea in a fail safe manner, it has to use a variety of sensors to analyze its own current state and to interpret the surrounding environment. Information from these sensors are used to identify obstacles and other vessels, and to estimate its own position and orientation. Based on this, the USV should plan a safe route avoiding obstacles, while following the range of traffic rules that apply at sea.

Under development of autonomous vehicles such as Odin, it is practical to perform verification tests as often as possible, verifying that the control system of the vehicle works as intended. As Odin currently is considered a working prototype and already is capable of tracking waypoints and avoiding obstacles, there is a need to continuously verify that these functionalities are maintained during further development.

It is however both expensive and time-consuming to perform the verification tests

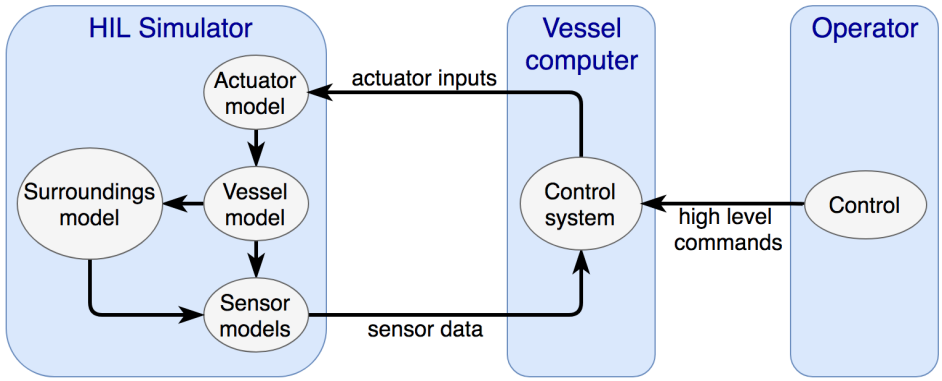

Figure 1.3: Illustration of a HIL simulator setup for an autonomous vessel. Adapted from Ødegaard (2017).

on the actual vessel at sea. This forms an incentive to develop a computerized simulator to test the control system in a fictitious environment.

1.2.1 Hardware-In-the-Loop Simulations

A Hardware-In-the-Loop (HIL) simulator simulates the behavior of a complex real-time system and allows embedded hardware, the hardware-in-the-loop, to interact with this simulation. See Skjetne and Egeland (2005) for definitions related to HIL simulations in marine control systems. The goal is for the hardware in the loop to experience no difference from the real world, so that it will sense and act like it would if the simulation was real. An illustration of this concept can be seen in Figure 1.3.

For the developers of Odin, this kind of simulator can be useful as the verification tests can be applied directly to the control system by simulating the dynamics of the USV in a realistic ocean environment. Simulated sensor data can be be fed to the USV, and by analyzing the resulting inputs to the engines, one can see what the control system chooses to do. Hence, errors in the software of the USV can be exposed prior to launch.

The HIL simulations can also be integrated in automatic tests of software under development. By running short test scenarios automatically every time the developers make changes to the current code, errors can be detected and addressed at an early stage.

Another benefit of HIL simulations is the possibility of testing failure scenarios in a safe environment. Pushing the control system to the limits might be dangerous if performed at sea, but if the testing takes place safely on land with the control system decoupled from the physical actuators, one can test a variety of dangerous scenarios with no risk of damage.

1.3 Problem Formulation and Contribution

This project is divided in two master theses and aims to result in a functional prototype of a HIL simulator platform for two different USVs, Odin and Jolner. The simulator should provide an artificial ocean environment including realistic weather conditions, obstacles and active agents. The software of the simulator should be divided in two separate, standalone software modules: the Dynamics simulator and the Surroundings simulator. Additionally, a graphical user interface (GUI) should be developed to visualize and interact with the simulations. Ødegaard (2017) will cover the simulation of the dynamics of the USVs, while this thesis will focus on simulation of the USV's surroundings and the GUI. Specifically, this thesis will cover:

- Investigation of the sensors used on Odin for analysis of the nearby surroundings.
- Conceptual investigation of a planned Target Detection Module (TDM) on-board Odin using sensor fusion of an Automatic Identification System (AIS), radar and Light Detection and Ranging (LiDAR) sensors.
- Development of a standalone simulator of the ocean surroundings with obstacles and other vessels. The simulator should be modular and initialized by configuration files, so that users of the simulator need minimal knowledge of the underlying code.
- Simulation of sensor data influenced by measurement noise, based on the surroundings of the USV. The desired simulated sensor data in the first prototype of the HIL simulator are the data from AIS and the preprocessed data from the TDM.
- Development of a simple, standalone graphical user interface (GUI) to visualize the simulations. The GUI should include real-time plotting of important param-

eters, a live 2D map of the area of operation and a 3D visualization of the USV in the simulated dynamic environment.

1.4 Outline of the Report

This report is divided in 8 chapters, as summarized below:

- **Chapter 2** presents the sensor systems used on Odin, and discusses the mode of operation, data format and limitations of the sensors that are relevant for this thesis.
- **Chapter 3** considers the TDM and the associated sensor fusion of AIS, radar and LiDAR. A basic fusion algorithm is applied to artificial data from these sensors to make an educated guess of the accuracy one can expect from the TDM.
- **Chapter 4** discusses which simulation scenarios that will be the most useful to implement in the first prototype of the HIL Simulator, in regards to the current development stage of Odin's control system. Which objects to include in the simulations, as well as the behavior of other simulated vessels, are also discussed here.
- **Chapter 5** presents the software and APIs that form the foundation in the implementation of the HIL Simulator.
- **Chapter 6** documents the implementation of the HIL Simulator and provides package, class and message diagrams of the resulting software. Message formats and configurations files are also documented here.
- **Chapter 7** demonstrates the HIL Simulator in action. An open sea scenario with presence of icebergs and two other vessels are simulated, with Odin navigating in the simulation using thrust inputs logged from a real mission. The performances of the simulated TDM and AIS are inspected.
- **Chapter 8** concludes the thesis and provides recommendations for further work.

Chapter 2

Sensors

During operations at sea, the Unmanned Surface Vehicle (USV) will use a variety of sensors to interpret the nearby ocean environment and to measure its own dynamic state. Data from a wind sensor, an Inertial Measurement Unit (IMU) and a Global Positioning System (GPS) is a direct result from the simulation of the USV's dynamics. Hence, data from these sensors are produced in the Dynamics simulator. For situational awareness, the USV will utilize an Automatic Identification System (AIS) transceiver, a radar and a LiDAR. This chapter will discuss the details and limitations of these sensors so that realistic behavior can be simulated in the HIL Simulator. Only the AIS will be used directly by the control system of Odin, and hence, a more thorough discussion of the data format of the AIS transceiver will be made.

2.1 Sensors Used on Odin

A variety of sensors constitutes the eyes and ears of Odin, and are vital for the USV to be able to navigate in a collision free manner between waypoints and obstacles. An illustration of some of the most important sensors for navigation can be seen in Figure 2.1. Some sensors monitor the internal states of the vessel, such as the GPS and the IMU measuring the acceleration and angular rate of the USV. These sensors are called proprioceptive sensors. To analyze the nearby surroundings, the USV relies on exteroceptive sensors. This section will briefly go through the various sensors of Odin relevant for this thesis. The proprioceptive and exteroceptive sensors of Odin relevant for the implementation of the HIL Simulator are summarized in Table 2.1.

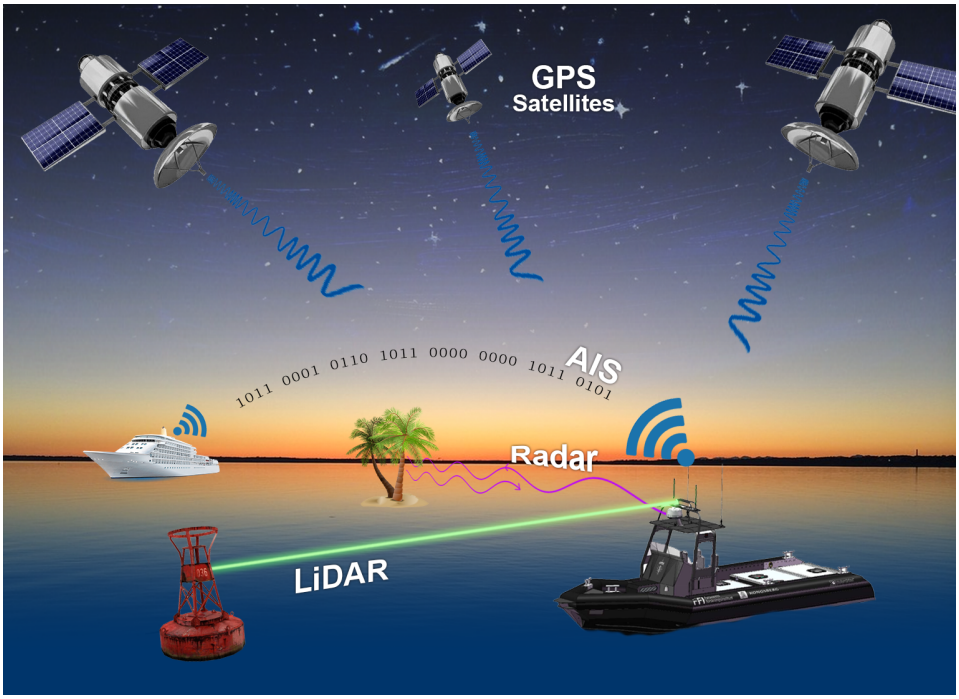

Figure 2.1: Illustration of some of the sensors used on Odin. The satellites along with a receiver onboard Odin constitutes the GPS. The radar transmits radio waves, illustrated as pink sine curves, in a certain direction and measures the reflected signal. Similarly, the LiDAR uses laser beams to measure the distance to nearby objects. With the use of AIS, nearby ships and vessels can share navigational data as illustrated with the radio transmission of data between Odin and a cruise ship.

Table 2.1: The proprioceptive and exteroceptive sensors of Odin relevant for the implementation of the HIL Simulator.

Proprioceptive	Exteroceptive
IMU	(AIS)
Seapath	Radar
	LiDAR
	Wind sensor

2.1.1 Proprioceptive Sensors

The proprioceptive sensors of Odin relevant for the implementation of the HIL Simulator are an Inertial Measurement Unit (IMU) and a Global Positioning System (GPS). A GPS collects position reports from orbiting satellites and measures the time delay of the updates to estimate its own global position. As a matter of fact, the GPS used on Odin is part of a bigger sensor system called Seapath 130, delivered by Kongsberg Seatex. The Seapath 130 uses 2 GPSs at a distance from each other to calculate the vessel's heading relative to North in addition to position and speed. The IMU uses a combination of accelerometers and gyroscopes to measure the acceleration and angular rate of the USV. The combination of the IMU and Seapath yields an accurate estimate of the USV's position, attitude, heading and speed. As data from the IMU and Seapath are based on the kinematic dynamics of the USV, these sensors are simulated in the Dynamics simulator and described in detail in Ødegaard (2017).

2.1.2 Exteroceptive Sensors

Exteroceptive sensors on Odin are, by definition, highly relevant for the simulation of the surrounding ocean environment in the HIL Simulator. An Automatic Identification System (AIS) transceiver will collect navigational data from nearby vessels that also utilize AIS. As AIS considers an entire protocol of maritime information sharing, AIS by itself can not be considered a sensor. However, for the purpose of Odin and the HIL Simulator, it is convenient to refer to the AIS transceiver as a sensor, as the output is used to gain information of nearby vessels. A radar and a LiDAR both measure the

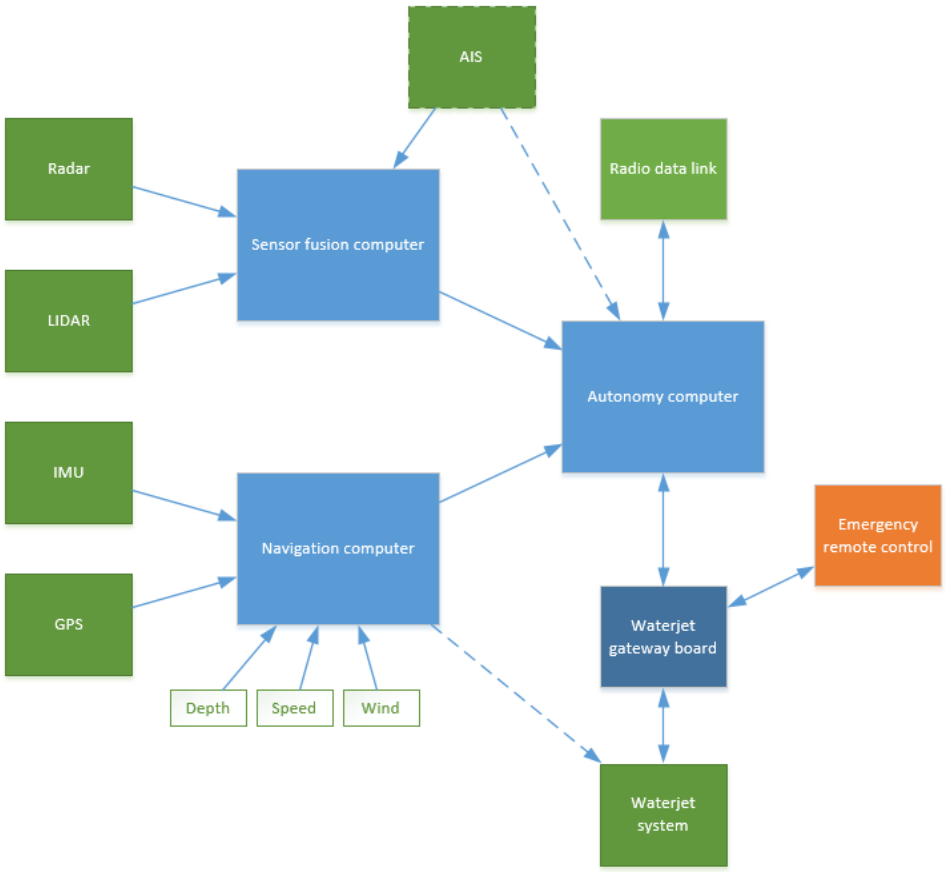

Figure 2.2: Network layout of the different computers and sensors on-board Odin. Courtesy of Kongsberg Maritime.

bearing and distance to objects close to the USV and will, in combination with the AIS, provide detailed information of nearby obstacles and ship traffic. The key properties, mode of operation and limitations of the AIS, radar and LiDAR will be discussed further in sections 2.2 - 2.4.

A wind sensor gives information of the relative direction and speed of the surrounding wind, which can be used as feed-forward to the control system to compensate for external forces resulting from wind. As there is a strong two-way dependency between the relative wind and the kinematics of the USV, this sensor is simulated in the Dynamics simulator and accounted for in Ødegaard (2017).

2.1.3 The Network of Sensors on-board Odin

The data from the proprioceptive and exteroceptive sensors are processed in computers on-board Odin. The sensors and their associated computers are connected in a network as illustrated in Figure 2.2. For transmission of data between the different sensors and computers, it is planned to utilize a framework called ROS, which provides simple and robust message handling between modules. ROS and the framework it provides are investigated further in Chapter 5.

In the network of Figure 2.2, the *Navigation computer* interprets the data from the sensors regarding the dynamic motion and position of the USV. The processed data is passed further to the control system, which runs on *Autonomy computer*. The *Sensor fusion computer* will combine data from the AIS, radar and LiDAR to identify and estimate the status of nearby obstacles and traffic. The sensor fusion of these sensor systems is investigated further in Chapter 3.

Note that data from the AIS is received by both *Sensor fusion computer* and *Autonomy computer*. This is because the information received from the AIS, in addition to being used as supplementary information about objects detected by the radar and LiDAR, is used by the control system for path planning in regards to the rules of traffic that apply at sea.

2.2 Automatic Identification System (AIS)

AIS is a widespread technology and protocol to share navigational data between vessels at sea. Standardized data packets are broadcast from vessels or structures at sea using VHF radio signals (U.S. Coast Guard Navigation Center (2016b)). The packets contain a unique vessel ID number and information about position, heading and navigational status, among others (U.S. Coast Guard Navigation Center (2016a)). Vessels with an AIS receiver can collect data from all the nearby traffic, and satellites in orbit can continually collect data from all around the world to form a worldwide overview of the current marine traffic as seen in Figure 2.3.

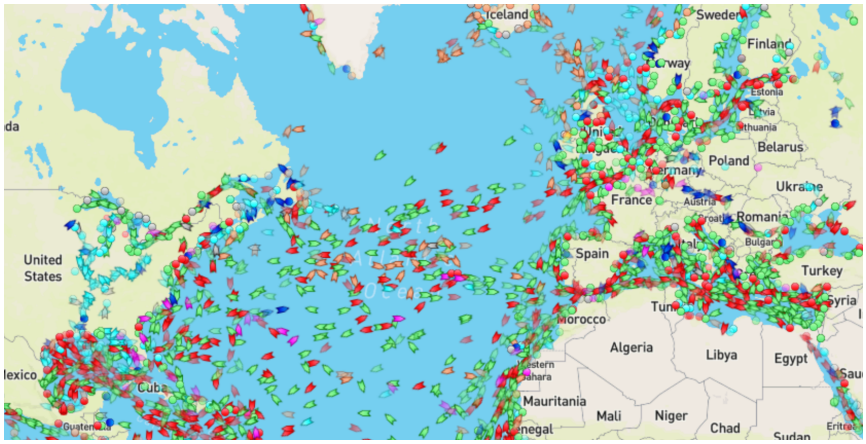

Figure 2.3: Live overview ship traffic utilizing AIS. Courtesy of `www.marinetraffic.com`.

2.2.1 The AIS Data Packet

The AIS transceiver to be used on Odin is not yet decided, but the output from such a transceiver is universal and defined by a strict protocol. An AIS receiver outputs the data packets received from other ships as ASCII strings following the NMEA 0183 or NMEA 2000 data format (SiRF (2005)). The data strings contain different fields of data segregated by commas. An example of a typical AIS Class A position report could be:

```
!AIVDM,1,1,,A,10000001AKPgsTbR0T:;Ka9MP6AP,0*40
```

- Field 1 ("`!AIVDM`"): The first field classifies the data packet as being sent from another ship or from your own vessel. AIVDM packets describes messages from other ships, while AIVDO packets contains information about the mother ship.
- Field 2 ("`1`"): Total number of AIS packets to deliver this specific data payload.
- Field 3 ("`1`"): The packet number of this specific package.
- Field 4 (empty): Sequential message ID if the message contains several sentences.
- Field 5 ("`A`"): AIS radio channel code, corresponding to different VHF radio frequencies.

- Field 6 ("`10000001AKPgsTbR0T:;Ka9MP6AP`"): Data payload bit vector represented as a chain of 6-bit characters. How to interpret and also how to make the data payload is described further below.
- Field 7 ("`0`"): Number of fill bits to achieve the 6-bit encoding. The number (could be 0 - 5) tells how many least significant bits to ignore from the data payload bit vector.
- Field 8 ("`40`"): The last field, separated by the $\star$-symbol, is the NMEA 0183 data-integrity checksum used to verify that the data packet was received correctly.

Several online AIS decoders exist, providing tools to interpret AIS messages. One of them is available at `https://rl.se/aivdm`, where the user can paste a raw AIS string and receive the underlying data in a readable format.

2.2.2 The Data Payload

The data payload contains a strictly defined AIS message containing important information about a vessel's navigational status. There are several types of AIS messages, each defined by a strict protocol. An overview of current AIS messages and a thorough description of the protocols can be found in E. S. Raymond (2016). As an example, a Class A position report message contains 168 bits and is sent every 2 to 10 seconds by larger vessels at sea while underway. The bit vector of 168 bits contains different fields of data such as a unique Maritime Mobile Service Identity (MMSI) number, the ship's current longitude and latitude, course over ground (COG), speed over ground (SOG), track, navigation status, etc.. The data fields of the Class A position report are accounted for in Table 6 in E. S. Raymond (2016). Other AIS messages and their protocols can also be found here.

To generate artificial AIS data packets from simulated ships and vessels one must acquire or define the relevant information and assign it accordingly to the different data fields in the payload. The data payload must then be encapsulated in a proper AIS packet as described in Section 2.2.1 and the checksum must be calculated and added to the data string.

2.2.3 Limitations of AIS

The limitations of the AIS are discussed in Chapter 5.2.2 in Børs-Lind (2017). It is assumed that the VHF radio signals used for AIS transmissions will reach further than the size of the simulated map area under all conditions. The accuracy of the estimated states in the AIS data packets, such as position, COG and SOG, are however limited by the transmitting ship's navigation system. There will always be small errors, and this must be taken into account when implementing an AIS transmission from a simulated ship or vessel. In Chapter 5.2.1 in Børs-Lind (2017) it is described how the stochastic nature of sensor noise can be modeled as a slowly varying bias added with zero-mean white noise. It is suggested that noise modeled in such a fashion is applied to the estimated states in the AIS transmissions of simulated ships and vessels.

Because the AIS receiver relies on the other vessels' transmissions of navigational data, one can not trust the information obtained from an AIS receiver any more than one trusts the equipment and intentions of the other vessels. AIS transmissions can easily be jammed, the data can be intentionally corrupted to confuse nearby traffic, or the navigation system of the transmitter can suffer from malfunctions. Because of this, a collision avoidance system should never solely rely upon data obtained from an AIS receiver. Decisions in regards to collision avoidance should primarily be based on visual aids and radar data.

2.3 Radar

The radar used on Odin is a Simrad Broadband 4G with characteristics as described in Børs-Lind (2017). As the radar will not be simulated directly, the details of the data format and how to generate realistic readings are omitted. More important for this project is which states are actually measured, the accuracy of the information obtained and the limitations of the radar itself.

Figure 2.4: Simrad Broadband 4G radar used on Odin. Courtesy of Simrad.

By firing a short radio beam in a known direction and measuring the reflected echo

signal, the radar can estimate the position of an object relative to the radar. Monitoring the object over time, the radar can also estimate the speed and course of the object. By measuring the power of the reflected signal, the radar can also estimate a measure of size. The measure of size is usually annotated as the radar cross section (RCS), which if interpreted in a literal sense means the cross sectional area of the object as seen from the radar. The RCS is highly dependent on the shape, orientation and material of object, and it might appear both smaller and bigger than it actually is based on these factors.

2.3.1 Limitations of the Radar

A radar does however have some limitations, as was discussed in more detail in Børs-Lind (2017). The key properties and limitations to keep in mind during development of the HIL Simulator are listed below:

- The line of sight to an object must be free of other obstacles for the object to be detected. That is to say, if Object 1 is behind Object 2, Object 1 will not be detected. This phenomena is called radar shadow and is illustrated in Figure 2.5.
- An object will only be detected if it is inside the minimum and maximum detection range of the radar. The power of the radio signal, curvature of the earth and the time it takes for the reflected radio signal to return are important factors that influence the range of the radar. According to SIMRAD (2012), the Simrad Broadband 4G has a range of 200ft to 32 nautical miles (ca. 61m - 59km) at minimum sweep rate. If operating at maximum sweep rate the range of the radar decreases to 1 nautical mile (1.852km).
- The sweep rate of the radar will affect the frequency at which information about tracked objects are updated. For the radar used on Odin the sweep rate is mode dependent with the option of 24/36/48RPM (0.4/0.6/0.8Hz) (SIMRAD (2012)).
- The weather conditions will affect the radar range. During heavy rain the range will decrease as the reflected radio signal will be contaminated by reflected noise from the rain drops.
- The probability of detecting an object is dependent on the current RCS of the object. If shaped and oriented in a certain way, an object might not be detected at

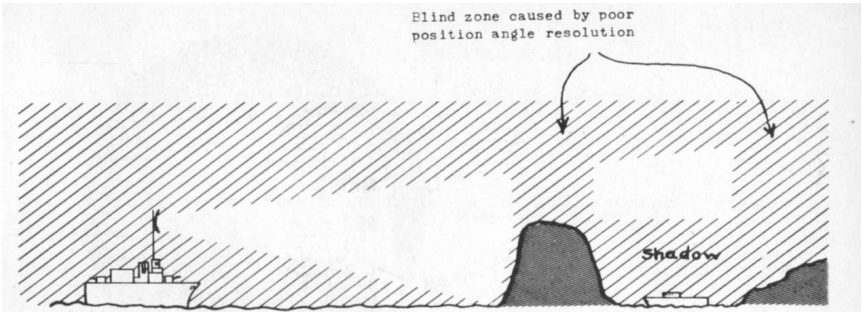

Figure 2.5: Illustration of radar shadow. Objects behind other objects relative to the radar will not be detected. Courtesy of `http://www.ibiblio.org`.

all, a property that is exploited in the development of "stealthy" military aircrafts and vessels.

For the purpose of the HIL Simulator it is desirable to be able to test the USV during different conditions, e.g. heavy rain, which might influence the performance of the radar. The developers of Odin might however want to manually set the operation mode and error parameters of the radar, so instead of designing predefined weather conditions it is suggested to design the software in such a way that these parameters easily can be manipulated individually. As for the radar cross section and the probability of detection, it is suggested that all objects that are inside the radar range and outside the radar shadow are detected by the radar system. This is in accordance with the needs of the development team of Kongsberg Maritime and FFI, as simulating objects that the USV can't see has no purpose in regards to testing of the USV's performance.

2.4 LiDAR

A LiDAR works in a similar way as the radar by firing laser beams and measuring the time it takes for the light to return. The range of the LiDAR is smaller but the accuracy is beyond what can be achieved by a radar. As discussed in Børs-Lind (2017), the LiDAR used on Odin will be the Velodyne HDL-32E. The LiDAR can be seen in Figure 2.6. At short range, this will be the main contributor in detection of nearby targets. As the information from the LiDAR also will be filtered through the sensor fusion of AIS, radar and LiDAR, the details of the hardware and data format are omitted.

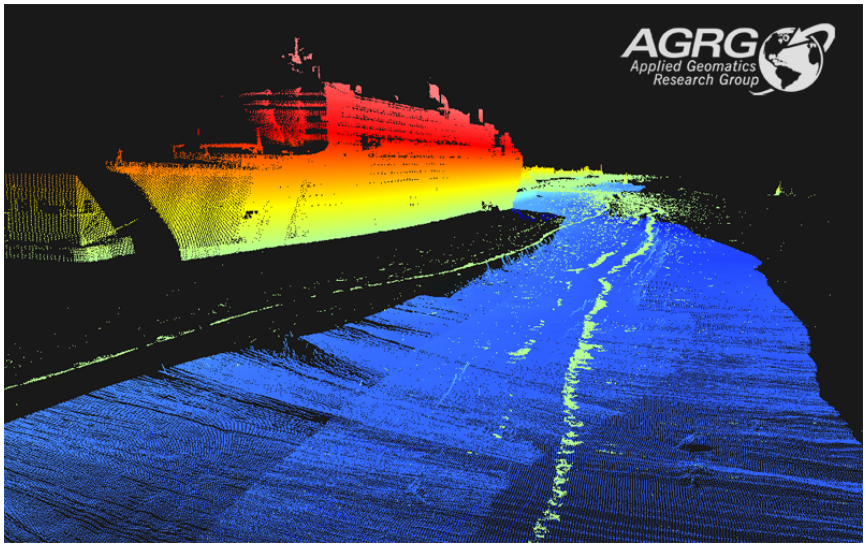

Figure 2.7: Example of how a LiDAR can be used for 3D mapping of a harbor. Source: `http://agrg.cogs.nscc.ca/node/234`.

Figure 2.6: The Velodyne HDL-32E LiDAR used on Odin. Courtesy of Velodyne.

The data from the LiDAR is processed by a dedicated processing unit, which still is under development. It is however assumed that the output from this unit will be a list of detected objects with information about position and size, and that the data will arrive at the same rate as the sweep rate of the LiDAR. The limitations of the LiDAR must however still be discussed to be able to implement a realistic sensor fusion.

2.4.1 Limitations of the LiDAR

As the LiDAR is similar to the radar in its mode of operation, the LiDAR will suffer from similar limitations in regards the requirement of direct line of sight, detection range,

sweep rate, weather conditions and cross section of the object. The key properties of the LiDAR used on Odin are listed below (Velodyne (2017)):

- Up to 100m range, however heavily affected by weather conditions such as rain, fog or snow.
- Accuracy of ±2cm in estimated distance to an object, and and an angular accuracy of 0.1°- 0.4°.
- Sweep rate of 5-20Hz.

As with the radar, it is suggested that these parameters should be easily accessible for the users of the HIL Simulator, so that they can test the USV's performance under different conditions

2.5 Modeling of Sensor Noise

The AIS, radar and LiDAR will all suffer from the presence of noise, which needs to be taken into account when aiming to generate realistic sensor data in a simulated environment. The conceptual modeling of sensors influenced by noise was discussed in Section 5.2.1 in Børs-Lind (2017). It was suggested to model the noise as a slowly drifting bias added with zero-mean white noise with a certain stochastic distribution such as the Gaussian distribution. A measured one-dimensional signal including noise can hence be modeled as shown in (2.1 - 2.3):

$$\dot{b} = -T_b^{-1} b + w_b \tag{2.1}$$

$$e = b + w_m \tag{2.2}$$

$$x_m = x + e, \tag{2.3}$$

where b is the slowly drifting bias, T_b is the time constant of the stabilizing bias feedback, x_m is the measured signal, x is the true value and w_b and w_m are uncorrelated white noise. Intuitively, the variance of w_b will influence how fast the biases deviate, while the time constants of T_b affects the strength of which the biases are pulled

back to zero. The variances of w_m correspond to the scattering of the sensor measurements. The navigation data from simulated ships utilizing AIS will be influenced by minor noise that can be modeled in this fashion.

Errors in measurements from the radar and LiDAR will grow proportionally with greater distance between the USV and the object. To preserve this property in the HIL simulations it is suggested to multiply the error e in 2.3 by an increasing function $f(d)$ where d is the distance between the USV and the object:

$$x_m = x + f(d)e \tag{2.4}$$

$$f(d) = kd, \tag{2.5}$$

where k is a positive constant.

Chapter 3

Sensor Fusion in a Target Detection Module

Data from the AIS, radar and LiDAR are processed in a dedicated computer on-board Odin, which will combine the data in a sensor fusion algorithm to analyze the surrounding environment above the surface and identify objects the USV needs to avoid. This computer is referred to as the Target Detection Module (TDM), and it is decided to simulate the output from this module in the HIL Simulator. The TDM of Odin is still under development, so the format of the output is at the current time not decided. This chapter will however aim to make a qualified guess in regards to this format and discuss how to implement a realistic simulated TDM in the HIL Simulator. A basic sensor fusion algorithm called Simple Fusion will be investigated and applied to simulated data from AIS, radar and LiDAR to attain an idea of the accuracy one can expect from a fusion of these sensor systems.

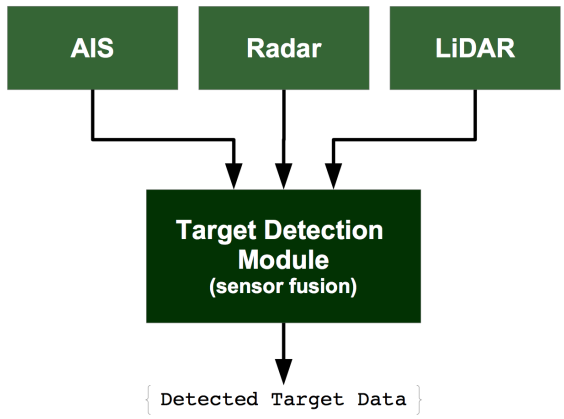

Figure 3.1: Conceptual illustration of the sensor fusion of AIS, Radar and LiDAR in a Target Detection Module (TDM).

3.1 The Target Detection Module (TDM)

The Target Detection Module (TDM) on Odin corresponds to the *Sensor fusion computer* in the network layout of the different hardware components seen in Figure 2.2. The control system is running on the *Autonomy computer*. The aim of the TDM is to produce standardized detected target messages sent to the USV's control system as seen in Figure 3.1. The TDM will also output a layered 2D map combining data from the AIS, radar and LiDAR meant for advanced path planning, but this is planned to be implemented at a later stage. In the early development stages, the control system of Odin will only respond to the detected target messages. Hence, there is no use in simulating the layered 2D map in the first prototype of the HIL Simulator.

As the primary goal for the first prototype of the HIL Simulator is to be able to test the USV's control system, it is considered sufficient to simulate only the data output from the TDM, and not the raw data from the Radar and LiDAR. The AIS is however also planned to be used directly by the control system to aid in path planning based on COLREGs[1], and must hence still be simulated in its raw format. A more thorough discussion of the reasons for this decision can be found in Børs-Lind (2017), and the decision is also encouraged by Kongsberg Maritime.

[1] COLREGs = International Regulations for Preventing Collisions at Sea

3.1.1 The TDM Data Format

The TDM will combine data from the AIS, Radar and LiDAR to form an overview of nearby obstacles and traffic. Examples of obstacles can be ice bergs or buoys. A discussion of different kinds of maritime traffic can be found in Chapter 3 of Børs-Lind (2017). For simplicity the traffic was classified in 3 groups: ships, vessels and boats, each with different characteristics in regards to size, average speed, turning radius, degree of unpredictability and the active use of AIS. The TDM will aim to identify obstacles and traffic and estimate the position, size, speed over ground (SOG) and course over ground (COG) of these objects. The output from the TDM will be data packets which contain this information along with a unique ID for each detected object. Unfortunately, the TDM on-board Odin is still under development, so the exact data format of the output from this module is still unclear. A qualified guess in regards to this format can however still be made.

It is known that the ROS framework will be used for information sharing in the network of sensors and processing units on-board Odin. It is therefore a fair assumption that the data format of the TDM output is based on ROS messages, for example one ROS message for each detected object containing information about the object's given ID, position, COG, SOG and size. A descriptive keyword is also likely to be included in the message to separate between different types of objects. Such a keyword can be an enumeration number or a short string, e.g. "vessel" or "land". For readability, it is of the author's personal preference to use a descriptive string.

The TDM and the control system need to agree on a protocol for message type and the different data fields. For the HIL Simulator to be useful for the developers of Odin in the future it is important that the format of the ROS messages from the simulated TDM can be easily changed in accordance to their preference.

Based on the above discussion, the suggested output from the simulated TDM is periodic ROS messages with information about the detected objects. As long as an object is detected and stored in the TDM, a ROS message will be sent periodically with the most recently updated information of the object at the current time. The suggested ROS message structure for this purpose is illustrated in Table 3.1.

Table 3.1: Suggested structure of a detected object ROS message.

Data Type	Variable Name	Unit
int	ID	-
string	Descriptor	-
float	Longitude	[deg]
float	Latitude	[deg]
float	COG	[deg]
float	SOG	[m/s]
float	CrossSection	[m^2]

3.1.2 Limitations of the TDM

In a computer-generated ocean environment it would be easy to simulate a perfect TDM that detects all the simulated objects with perfect precision. This would however not be the case during real operation as the AIS, Radar and LiDAR all suffer from inevitable imperfections. The limitations of all the relevant sensors as well as a discussion of conceptual modeling of sensor noise are discussed in detail in Chapter 5.2 in Børs-Lind (2017). Based on this it is assumed that the simulated TDM will have the following properties:

- Obstacles such as smaller boats, ice bergs and buoys will not be detected if they are not visible to the Radar and LiDAR.
- Ships and vessels that utilize AIS will always be detected as the AIS range is assumed to be greater than the simulated map. The navigational parameters received in the AIS data packets from other simulated ships and vessels will suffer from small inaccuracies.
- Inaccuracies in regards to position, COG, SOG and size of objects that don't utilize AIS are growing with larger distance to said object.
- Inaccuracies are stochastic by nature as described in Section 2.5.

3.2 Sensor Fusion Example Using Kalman Filters

Intuitively, having two uncorrelated sensors measuring the same state should make it possible to produce a better estimate of the state than either one of the sensors could produce by itself. A lot of research has been made aiming to develop effective algorithms to combine data from different sensors. Kazimierski (2013) investigates some of the popular methods for fusion of data from an AIS receiver and a radar tracking system. Given the knowledge of the characteristics of the AIS, radar and LiDAR used on Odin, it would be interesting to implement a simple sensor fusion algorithm to investigate what accuracy one can expect from the TDM.

One of the most popular methods described in Kazimierski (2013) is simply called the Simple Fusion Algorithm, which calculates a weighted average of the elementary estimates of the Kalman filtered data from AIS and radar. Which weights to apply to the different estimates are calculated from the covariance matrix P of the respective Kalman filters. Using the P matrices renders it possible to put more weight on the estimates with low variance. The fused estimate is generated using the following formula:

$$\hat{x} = (P_a^{-1} + P_r^{-1})^{-1}(P_a^{-1}\hat{x}_a + P_r^{-1}\hat{x}_r), \tag{3.1}$$

where P_a and P_r are the covariance matrices of the estimates from AIS and radar, respectively, using regular Kalman filters. Similarly, $\hat{x}_a$ and $\hat{x}_r$ are the estimated states from AIS and radar. The Simple Fusion algorithm is easily expandable to also utilize measurements from a LiDAR by including P_l and $\hat{x}_l$ in (3.1):

$$\hat{x} = (P_a^{-1} + P_r^{-1} + P_l^{-1})^{-1}(P_a^{-1}\hat{x}_a + P_r^{-1}\hat{x}_r + P_l^{-1}\hat{x}_l). \tag{3.2}$$

3.2.1 Kinematic Model of Tracked Object

To apply the Kalman filters to the tracking data from AIS, radar and LiDAR, one must obtain a mathematical model of the dynamics of the tracked object. As discussed Section 3.1.1, the TDM aims to track an object's position, COG, SOG and cross section. Based on

this, a simple 3DOF kinematic model of a detected object to track is suggested below:

$$\dot{\mathbf{x}}_{\mathbf{do}} = \mathbf{f}(\mathbf{x}_{\mathbf{do}}) \tag{3.3}$$

where

$$\mathbf{x}_{\mathbf{do}} = \begin{bmatrix} \phi \\ \lambda \\ \psi \\ u \\ r \\ \alpha \end{bmatrix}, \qquad \mathbf{f}(\mathbf{x}_{\mathbf{do}}) = \begin{bmatrix} k_\phi usin(\psi) \\ k_\lambda ucos(\psi) \\ r \\ b_1 \\ b_2 \\ 0, \end{bmatrix} \tag{3.4}$$

where ϕ, λ and ψ are the target's longitude, latitude and heading, respectively. r and u are the heading rate (ROT) and speed (SOG), while α is the cross section of the object, which is assumed constant. k_ϕ and k_λ transforms the speed measured in [m/s] to longitude and latitude degrees per second, and can be assumed constant in the area of operation, unless the USV is fairly close to the South or North Pole. Note that the change in ROT and SOG are unknown to the observer, and hence modeled as slowly varying biases as discussed in Section 2.5.

The radar and LiDAR will only measure the position and cross section of an object, while the data from AIS will contain information about position, heading, speed and rate of turn (ROT):

$$\mathbf{y_a} = \begin{bmatrix} \phi_a \\ \lambda_a \\ \psi_a \\ u_a \\ r_a \end{bmatrix}, \qquad \mathbf{y_r} = \begin{bmatrix} \phi_r \\ \lambda_r \\ \alpha_r \end{bmatrix}, \qquad \mathbf{y_l} = \begin{bmatrix} \phi_l \\ \lambda_l \\ \alpha_l \end{bmatrix} \tag{3.5}$$

where $\mathbf{y_a}$, $\mathbf{y_r}$ and $\mathbf{y_l}$ are the measured data from AIS, radar and LiDAR, respectively.

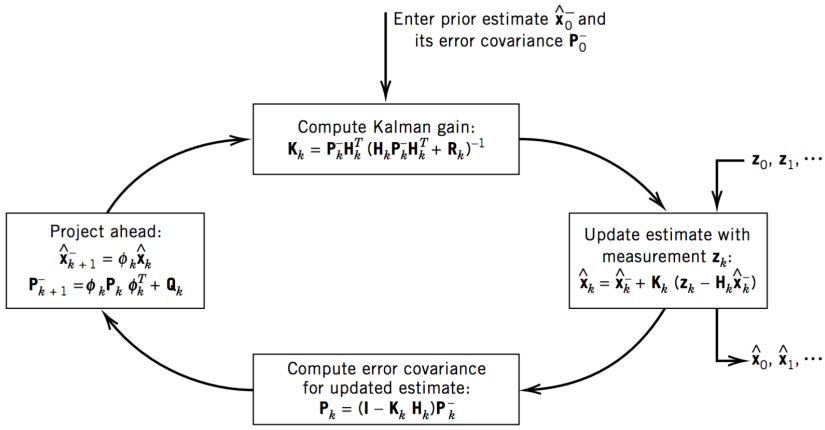

Figure 3.2: The steps of discrete-time Kalman filtering, visualized as the famous Kalman loop. Source: Brown and Hwang (1997).

3.2.2 The Extended Kalman Filter (EKF) Applied to Tracked Object

It is assumed that the reader is familiar with basic linear estimation theory and Kalman filtering. The Kalman filter combines measurements from sensors and knowledge of the dynamics and stochastic properties of the process to produce an optimal estimate of the relevant states. Figure 3.2 visualizes the discrete-time Kalman filter algorithm, and the theory behind it can be found in Brown and Hwang (1997). If the process contains the following properties (*ref.* Brown and Hwang (1997)):

- the process noise and measurement noise are white and Gaussian,
- the initial state is Gaussian.
- the system is linear,
- the system is observable,

the Kalman filter is asymptotically stable and the optimal state estimator with minimum variance. From (3.3) it is unfortunately clear that the process we aim to estimate is not linear, so the basic linear Kalman filter can not be used for the purpose of estimating the states of a tracked object.

The Extended Kalman Filter (EKF) is an extension of the linear Kalman Filter and can be applied to nonlinear systems by linearizing about the estimated current states

Design matrices	$\boldsymbol{Q}(k) = \boldsymbol{Q}^\top(k) > 0$, $\boldsymbol{R}(k) = \boldsymbol{R}^\top(k) > 0$ (usually constant)
Initial conditions	$\bar{\boldsymbol{x}}(0) = \boldsymbol{x}_0$ $\bar{\boldsymbol{P}}(0) = E[(\boldsymbol{x}(0) - \hat{\boldsymbol{x}}(0))(\boldsymbol{x}(0) - \hat{\boldsymbol{x}}(0))^\top] = \boldsymbol{P}_0$
Kalman gain matrix	$\boldsymbol{K}(k) = \bar{\boldsymbol{P}}(k)\boldsymbol{H}^\top(k)\,[\boldsymbol{H}(k)\bar{\boldsymbol{P}}(k)\boldsymbol{H}^\top(k) + \boldsymbol{R}(k)]^{-1}$
State estimate update	$\hat{\boldsymbol{x}}(k) = \bar{\boldsymbol{x}}(k) + \boldsymbol{K}(k)\,[\boldsymbol{y}(k) - \boldsymbol{H}(k)\bar{\boldsymbol{x}}(k)]$
Error covariance update	$\hat{\boldsymbol{P}}(k) = [\boldsymbol{I} - \boldsymbol{K}(k)\boldsymbol{H}(k)]\,\bar{\boldsymbol{P}}(k)\,[\boldsymbol{I} - \boldsymbol{K}(k)\boldsymbol{H}(k)]^\top$ $+\boldsymbol{K}(k)\boldsymbol{R}(k)\boldsymbol{K}^\top(k), \quad \hat{\boldsymbol{P}}(k) = \hat{\boldsymbol{P}}(k)^\top > 0$
State estimate propagation	$\bar{\boldsymbol{x}}(k+1) = \mathcal{F}(\hat{\boldsymbol{x}}(k), \boldsymbol{u}(k))$
Error covariance propagation	$\bar{\boldsymbol{P}}(k+1) = \boldsymbol{\Phi}(k)\hat{\boldsymbol{P}}(k)\boldsymbol{\Phi}^\top(k) + \boldsymbol{\Gamma}(k)\boldsymbol{Q}(k)\boldsymbol{\Gamma}^\top(k)$

Figure 3.3: The Extended Kalman Filter algorithm presented in Fossen (2011).

and covariance. This has been proven very effective, and the EKF is widely used in state estimation of nonlinear processes. Fossen (2011) gives a brief introduction to the EKF, and the modified equations of the standard Kalman filter are summarized in Figure 3.3.

Before we can apply the filters, we must consider the presence of noise in both the process and the measurements. Noise as discussed in Section 2.5 are assumed to influence all the measured states from the AIS, radar and LiDAR. With this in mind, a complete model of the tracked object is suggested below:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}) + \mathbf{w_x} \\ \mathbf{y_a} &= \mathbf{H_a}\mathbf{x} + \mathbf{w_{ya}} \\ \mathbf{y_r} &= \mathbf{H_r}\mathbf{x} + \mathbf{w_{yr}} \\ \mathbf{y_l} &= \mathbf{H_l}\mathbf{x} + \mathbf{w_{yl}}, \end{aligned} \tag{3.6}$$

where

$$\mathbf{x} = \begin{bmatrix} \phi \\ \lambda \\ \psi \\ u \\ r \\ \alpha \\ b_1 \\ \vdots \\ b_{13} \end{bmatrix}, \qquad \mathbf{f}(\mathbf{x}) = \begin{bmatrix} k_\phi u sin(\psi) \\ k_\lambda u cos(\psi) \\ r \\ b_1 \\ b_2 \\ 0 \\ -T_1^{-1} b_1 \\ \vdots \\ -T_{13}^{-1} b_{13} \end{bmatrix}. \tag{3.7}$$

In (3.6), $\mathbf{w_x}$ and $\mathbf{w_y}$ are vectors of Gaussian white noise. Only the biases are affected by $\mathbf{w_x}$. A total of 13 independent biases as described in Section 2.5 are necessary: b_1 and b_2 to model the rate of change in u and r, $b_3 - b_7$ for biases in AIS data, $b_8 - b_{10}$ for biases in radar data and $b_{11} - b_{13}$ for biases in LiDAR data.

EKFs as described in Figure 3.3 were applied to simulated data from AIS, radar and LiDAR in the case of tracking an object moving at constant speed with slowly varying heading. Noise was added to the measurements based on the specifications of each sensor discussed in sections 2.2.3, 2.3.1 and 2.4.1. The process and measurement noise covariance matrices **Q** and **R** were tuned accordingly. For simplicity, the update rates of the sensors were kept equal and synchronized at 0.5Hz.

3.2.3 Results From Simple Fusion Algorithm

5 minutes of AIS, radar and LiDAR data with appropriate levels of noise was constructed from an artificial vessel in MATLAB. The vessel was designed to move at a constant speed of 5m/s with a slowly oscillating heading. The Simple Fusion algorithm summarized in (3.2) was applied to the estimates from the EKF's described in Section 3.2.2. The estimated speed, heading and position from AIS, radar and LiDAR are visualized in Figure 3.4 together with the resulting estimate from the sensor fusion. The deviations from the true states are plotted in Figure 3.5. The resulting standard deviations of the errors from true states after 50s of stabilization time are summarized in Table 3.2.

The AIS and radar was designed to give the least credible measurements. This is reflected in the plots as the AIS and radar consistently give the highest error and variance in the different state estimates. The Kalman filtered LiDAR data yields good estimates in all states as the precision of the LiDAR is superior to the other sensors. The AIS is however the only system that directly measures the heading of the tracked object. The output of the Kalman filtered AIS data will hence give the most reliable estimate in regards to heading along with the LiDAR, who also clearly is able to estimate this state with great accuracy.

From the state plots in Figure 3.4 and the error plots in Figure 3.5, it seems that the estimated states of the sensor fusion consistently follows the best estimates of the other sensors. In Table 3.2 it is indeed confirmed that the sensor fusion outperforms the other

Table 3.2: Resulting steady state standard deviations of errors from true values in the Kalman filtered data from AIS, radar and LiDAR, as well as standard deviations of the output from the Simple Fusion algorithm. The first 50 seconds of data were excluded from the calculation to give the filters some time to reach steady state.

	σ_ϕ [m]	σ_λ [m]	σ_ψ [deg]	σ_u [$\frac{m}{s}$]	σ_α [m^2]
AIS	2.315	1.499	3.240	0.013	0.000
Radar	0.863	2.414	4.839	0.018	0.024
LiDAR	0.807	2.426	4.066	0.008	0.013
Simple Fusion	0.795	1.562	3.002	0.005	0.003

sensors in the estimation of every individual state.

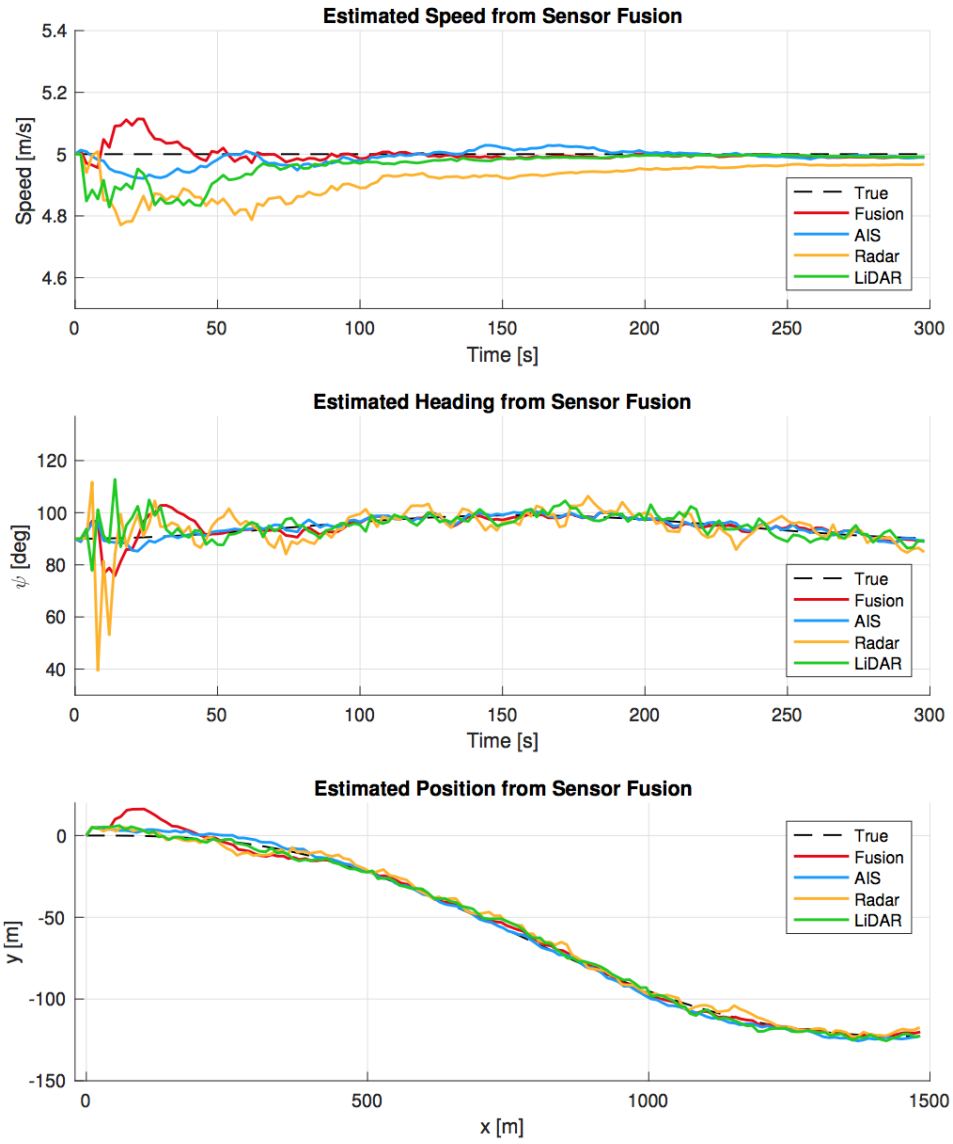

Figure 3.4: Plots of the estimated speed, heading and position of a tracked object using Extended Kalman filters on data from AIS, radar and LiDAR (blue, orange and green, respectively). Red is the result from the weighted average calculated from the Simple Fusion algorithm.

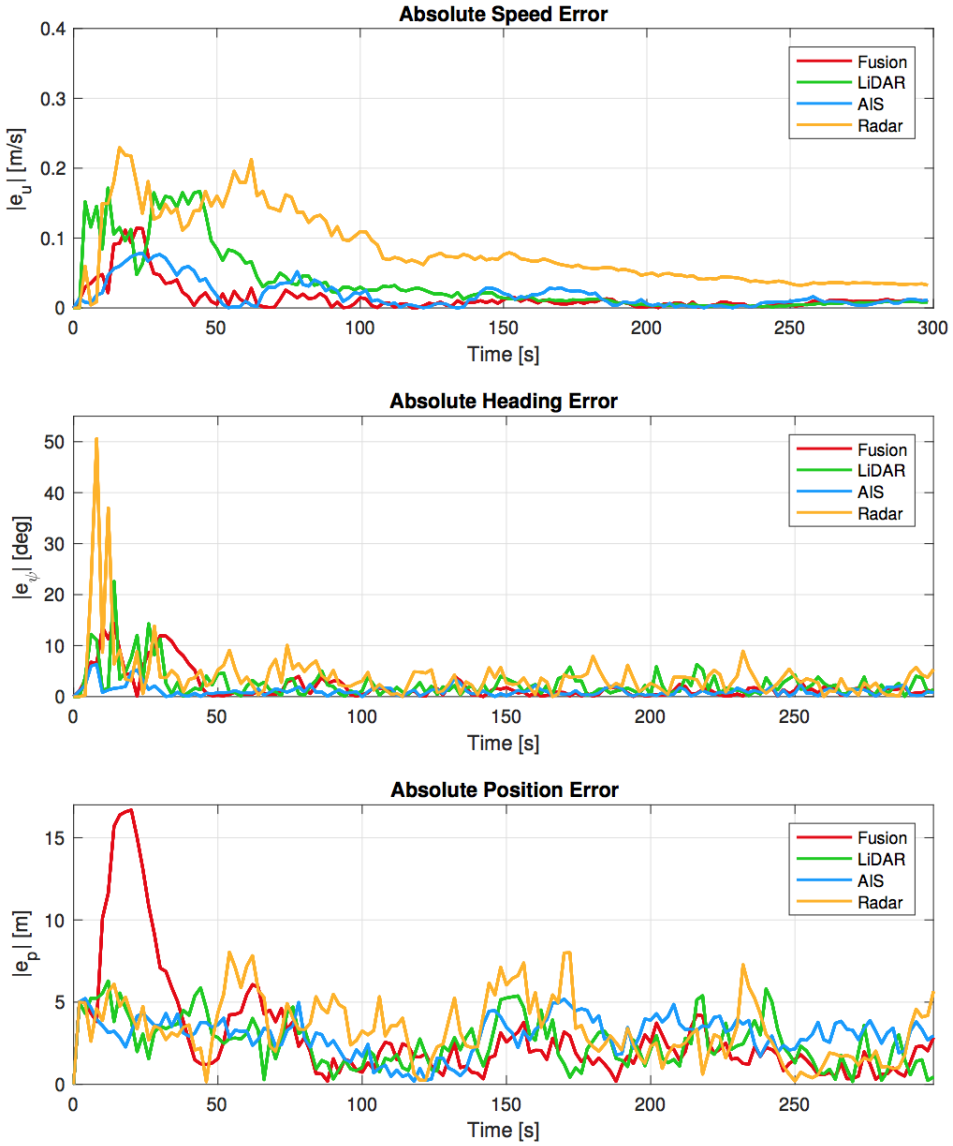

Figure 3.5: Plots of the errors in speed, heading and position in the estimates of a tracked object using Extended Kalman filters on data from AIS, radar and LiDAR (blue, orange and green, respectively). Red is the error of the estimates from the Simple Fusion algorithm.

3.2.4 Simple Fusion Algorithm as Inspiration for Simulated TDM

As the developers of Odin's software is likely to use a more sophisticated fusion algorithm than the Simple Fusion, the real TDM will probably perform even better than the

results presented in Section 3.2.3. The performance of the sensor fusion in this simulation can hence be used as a conservative inspiration in the implementation of the simulated TDM in the HIL Simulator.

Chapter 4

Simulation Scenarios and Objects

This chapter will discuss which scenarios that are the most useful to simulate for proper testing of the USV's collision avoidance system and what kind of objects it is necessary to include in these simulation. An open sea scenario with fixed obstacles and moving ships is suggested, as they challenge the collision avoidance system of Odin on a suiting level in regards to the current development stage. Emphasis will be put on the fact that the development of Odin's control system still is in the early stages, so that advanced simulation scenarios with complex traffic and terrain is unnecessary in the first prototype of the HIL Simulator. Finally, a short discussion will be made concerning the digital representation of land and complex shapes in 2D.

4.1 Simulation Scenarios

Several simulation scenarios for testing of the USV's performance was suggested and discussed in Børs-Lind (2017). The scenarios were generalized as 3 different cases with increasing complexity in regards to traffic and obstacles:

- The Open Sea scenario
- The Near Shore scenario
- The Busy Port scenario

The busy port scenario is characterized by the presence of many obstacles and heavy traffic from both larger vessels and smaller leisure boats. The traffic is relatively unpredictable, which challenges the intelligence of the USV's collision avoidance systems.

The traffic is less heavy and more predictable in the near shore scenario. Here, the vessels and boats are further apart and more likely to go in a straight line from A to B while following simple traffic rules of the sea. Operations near land require the USV to be able to navigate around complex shapes, possibly under presence of leisure boats with somewhat unpredictable behavior.

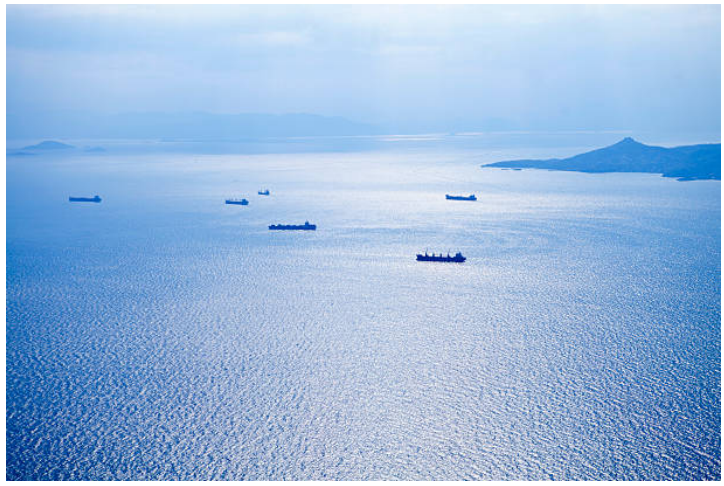

Figure 4.1: Illustration of a real world open sea scenario. Courtesy of `www.alamy.com`

The open sea scenario, as illustrated in Figure 4.1, takes place far away from the coast line, with a few obstacles representing ice bergs or small islands as well as larger

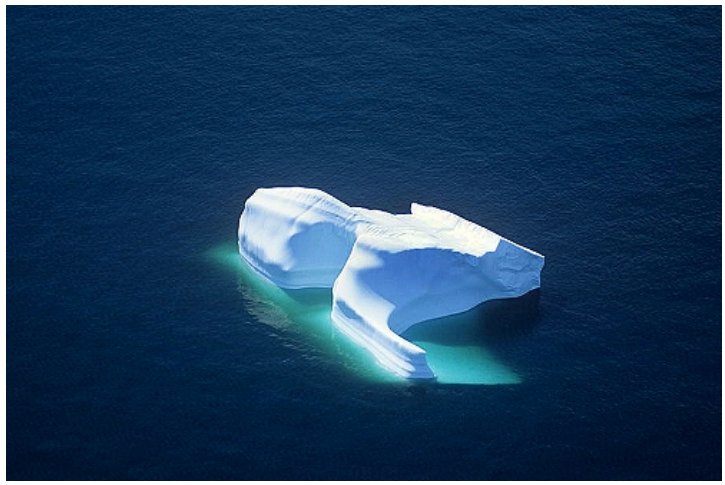

Figure 4.2: Lone iceberg in the Atlantic Ocean. Courtesy of `www.airphotona.com`

ships moving predictably from A to B. This was considered to be the simplest scenario to implement in the HIL Simulator, and also the most useful for testing of the early versions of the USV's control system. The USV's ability to maneuver correctly and efficiently in open water as well as simple collision avoidance can be tested in this scenario, which is sufficient for verification purposes in the early development phases of the USV's control system. It is hence suggested to implement this scenario as the first prototype of the HIL Simulator.

Useful simulation objects such as ships and fixed obstacles are needed for implementation of this scenario, which will be discussed further in the next sections.

4.2 Fixed Obstacles

For the USV to have some simple objects to avoid in the simulations, it is suggested to include the presence of fixed obstacles. Such obstacles can represent an iceberg, a buoy or a small island. An example of an iceberg perfect for testing of simple collision avoidance can be seen in Figure 4.2. Even though the icebergs drift with the ocean currents, it is suggested to simplify their behavior to a stationary object. It is not considered beneficial to implement drifting models at this early stage, but it might become useful for future verification of the USV's collision avoidance in polar waters.

As the goal of the first prototype of the HIL Simulator is to test the performance of the USV's early stage collision avoidance system, it is not considered necessary with too complex shapes. It is sufficient for the control system of the USV to know the coordinates and approximate size of the obstacle, so that it can plan to go around it with a relatively large safety margin. Obstacles like these will obviously not utilize AIS, so these objects can only be detected by the radar and LiDAR.

The fixed obstacles are by definition not able to move, and must hence stay at the same place throughout the simulations. They can vary in size, but for convenience it is suggested that they are modeled as discs with a constant radius.

4.3 Vessels

To test the performance of the USV's collision avoidance system under presence of moving vessels, it is suggested to include other ships in the simulations. This way, the USV will have to act in accordance with the fact that not all obstacles will stay at the same place throughout the mission. Examples of cargo ships that could be included in the open sea scenario can be seen in Figure 4.3.

Including ships ships in the simulations also makes it possible to test the USV's handling of incoming AIS messages. As the ships normally utilize AIS, they might be detected by both AIS, radar and LiDAR. The transmission of AIS data from the simulated ships should be possible to switch off to verify the performance of the USV during presence of "silent" moving obstacles and sudden loss of AIS signals.

It is advantageous for the scenario complexity to have the ships vary in size and speed. As it is suggested to implement an open sea scenario, it is natural to design the ships to move in straight lines as if they are underway to distant destinations. To complicate the scenarios further, the ships can also be designed to follow a waypoint plan, traveling from waypoint to waypoint. During turns between waypoints, the ships should behave with fairly realistic kinematics, i.e. turning slowly from one heading to another. Other than this, it is not considered necessary with complex dynamics such as swaying and drifting because of external environmental forces. The ships should move at constant speed, traveling in a straight line in the direction of the current heading.

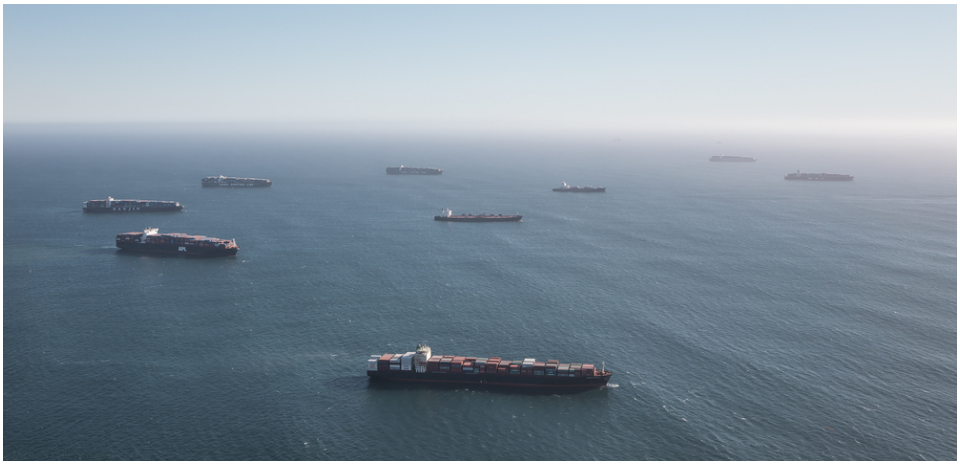

Figure 4.3: Cargo ships. Courtesy of Mike Kelly, `www.mpkelley.com`

4.4 Representation of Land

For larger islands and land, a simple disc with constant radius is not sufficient to describe the object. It is not considered necessary by Kongsberg Maritime to include obstacles this big and complex in the first prototype of the HIL Simulator, but future versions should contain the possibility of simulating scenarios close to land.

A common way of defining large, complex shapes in 2D maps is through the use of polygons, which can be represented as a vector of corner coordinates. An example of a harbor represented as a polygon with corner coordinates can be seen in Figure 4.4. The details of the edges can be improved by increasing the number of corner points in the vector. A lot of research has been made worldwide to make this method more effective in regards to processing time and number of corner points.

As the representation of land is not considered necessary in the first prototype of the HIL Simulator, it is suggested to make this a low priority and rather include land and complex shapes at a later stage when the control system of the USV is further developed. Further research of methods to represent land and complex shapes is needed before implementation in the HIL Simulator.

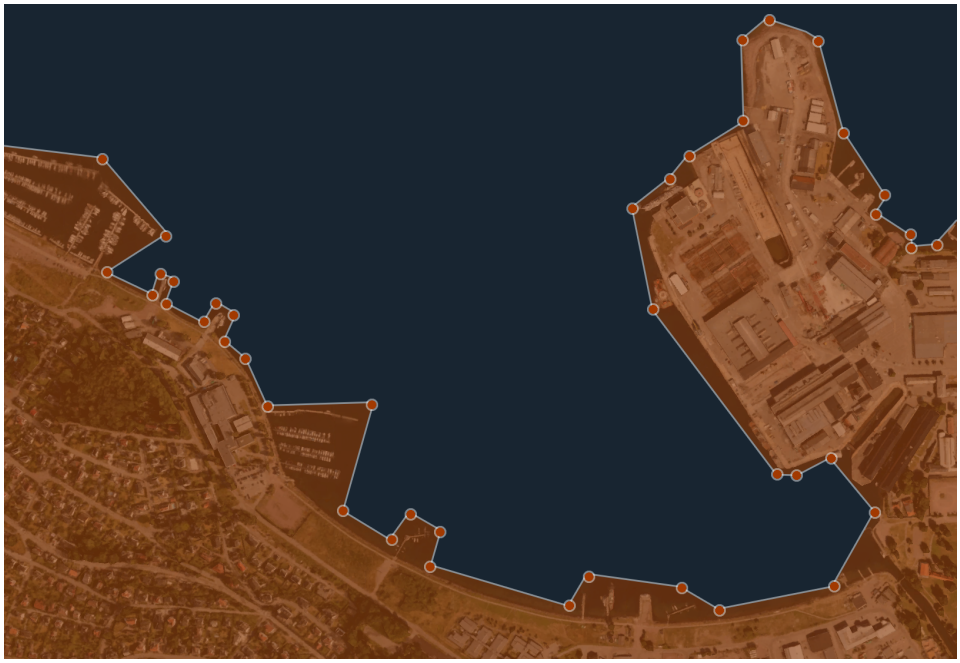

Figure 4.4: Sælavika inner harbor in Horten, represented as a polygon with corner coordinates.

Chapter 5

Software and APIs for Use in the HIL Simulator

The Robot Operating System (ROS) is used for information sharing in the network of sensors and computers on-board Odin. To interface between the HIL Simulator and Odin's control system will hence be based upon ROS. Additionally, ROS provides several features which are of interest in the implementation of the HIL Simulator, such as the 3D visualization tool RViz. Qt is an application programming interface (API) that provides robust tools for implementation of graphical computer programs, which is particularly interesting in the implementation of the GUI.

This chapter will investigate these software frameworks and discuss some of the tools that will be used in the implementation of the HIL Simulator. Finally, the programming language for use in the implementation is decided, based on compatibility with the different frameworks and the advantages of the languages of interest.

5.1 Robot Operating System (ROS)

The Robot Operating System (ROS) is a result of collaborative development across research labs all over the world and forms an open source framework for developing software in robotics applications (ROS (2014)). In Section 2.1 in Børs-Lind (2017) the key properties of ROS was briefly discussed, and it was decided that ROS could be a suitable framework in development of the HIL Simulator. Particularly interesting were the simple and robust message handling between processes utilizing ROS. As the software constituting the HIL Simulator is planned to consist of separate software modules running asynchronously it is convenient to utilize a robust framework to manage the message handling between modules. In addition to this, the network of sensors and computers on-board Odin is planned to utilize ROS in their message handling.

ROS is restricted to Unix-based platforms, with Ubuntu being the preferred operating system (OS) of choice. It is therefore recommended that the HIL Simulator will be developed in Linux Ubuntu. C++, Python and Lisp are supported languages. ROS is licensed under the open source BSD license Free Software Foundation (2016).

5.1.1 ROS Packages

Software utilizing ROS is organized in ROS packages[1]. A ROS package is a directory that can contain source and header files, configuration files, messages types and a build file, among others. Every package must have a package manifest file called `package.xml`[2] which defines different properties of the package such as author, package name, version number etc.. A ROS package can be created from the terminal using the `catkin_create_pkg` command.

The ROS packages might also have so called ROS nodes, which simply is a software process that performs computations[3]. Several nodes can communicate using ROS topics[4]. Topics are buses where nodes publish and receive ROS messages[5]. A node that produces a certain type of data can publish the data as ROS messages on a dedicated

[1] `http://wiki.ros.org/Packages`
[2] `http://wiki.ros.org/Manifest`
[3] `http://wiki.ros.org/Nodes`
[4] `http://wiki.ros.org/Topics`
[5] `http://wiki.ros.org/Messages`

topic. Nodes that are interested in this data can subscribe to the same topic. The structure of a ROS message is defined in a `.msg` file.

As a relevant example, the Surroundings simulator might be implemented as a ROS package with several source- and header files, one of them being an executable containing the `main()`-function. The simulated sensors can publish their data on dedicated topics, as will be the case in the real sensor network of Odin. In the HIL Simulation setup, the control system of Odin will subscribe to these topics as it would during real operation.

5.1.2 YAML Configuration Files

Configuration files can be used to add user defined parameters to the parameter server[6] of ROS. The files are defined using YAML, a Unicode based data serialization language particularly convenient for this purpose (Ben-Kiki et al. (2009)). Any ROS node in a package can read from the parameter server. By including a `.yaml` file during launch of a ROS package, one can define a range of parameters that the nodes can read. This is considered very useful in regards to initialization of the HIL Simulator, as it enables users with no knowledge of the underlying code to configure the simulations with important parameters.

5.1.3 3D Visualization Using RViz

RViz[7] is a ROS package that provides a simple 3D visualization interface. Using ROS messages one can spawn and define the position and orientation of basic shapes or handmade 3D models in a rendered 3D environment. The 3D environment is visualized in a separate RViz window with basic tools like zoom, orientation of the camera angle and much more; perfect for easily visualizing a simulated environment including the USV with obstacles and other simulated ships.

[6] `http://wiki.ros.org/Parameter%20Server`
[7] `http://wiki.ros.org/rviz`

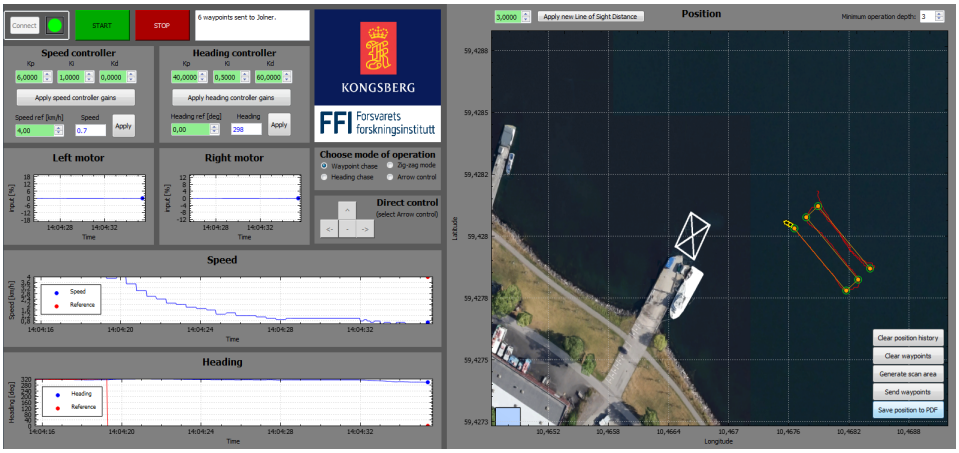

Figure 5.1: A GUI for control and monitoring of Jolner developed using Qt in the Survey Explorer summer project of 2016.

5.2 Qt

Qt is an application programming interface (API) for development of graphical computer programs, but can also be used in non-graphical programs as a framework to manage signal handling and threads in an intuitive manner. During the Survey Explorer summer project of 2016, Qt was used to develop a GUI for control and monitoring of the USV Jolner during operations at sea. The GUI can be seen in Figure 5.1.

Qt is available under the GNU LGPL (Free Software Foundation (2007)), which makes Qt popular among developers of free software, but it can also be used in proprietary software under a commercial license. C++ is the main supported programming language along with Java, but other languages are also available through third party binders such as PyQt[8]. The entire documentation of the Qt framework can be found in The Qt Company Ltd. (2017).

Using the Qt API it is possible to make both simple and advanced graphical user interfaces (GUI) to interact with a computer program. Qt gives intuitive and straight forward methods to define a wide variety of widgets such as windows, buttons, and labels, and the layout of the application can be designed using Qt Creator[9], Qt's own cross-platform Integrated Development Environment (IDE). Qt Creator is a convenient

[8] https://wiki.python.org/moin/PyQt
[9] http://doc.qt.io/qtcreator/index.html

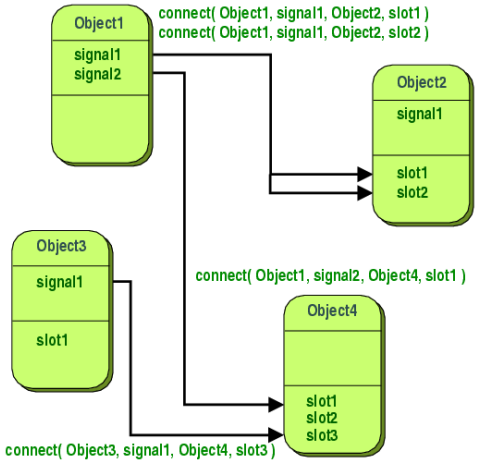

Figure 5.2: Illustration of signals and slots in Qt. Courtesy of The Qt Company Ltd.

tool to play with different layouts, but widgets can also be defined directly from code, and programs can be compiled using `CMake` directly from the terminal. The IDE is however very useful to perform debugging of the application by running the program in debugging mode.

5.2.1 Signals and Slots

As GUI applications by nature are required to respond to inputs in real-time, signal handling is of great importance in a solid development framework. Whenever some "unexpected" event happens, such as a button click, one usually wants the application to run a function to perform an appropriate response. Traditionally this has been achieved using callbacks, where a pointer to the function (the callback) is passed to the processing function. Qt offers a different approach, using *signals* and *slots*[10]. A signal emitted from one object can be connected to a slot in another object, causing the slot to be called whenever the signal is emitted. A slot is just a normal function with the additional property that it can be connected to a signal. The concept is illustrated in Figure 5.2. As an example, the basic Qt class `QTimer` emits the signal `timeout()` after a predefined amount of time. This signal can be connected to a slot to perform computations at certain time intervals.

[10] `http://doc.qt.io/qt-4.8/signalsandslots.html`

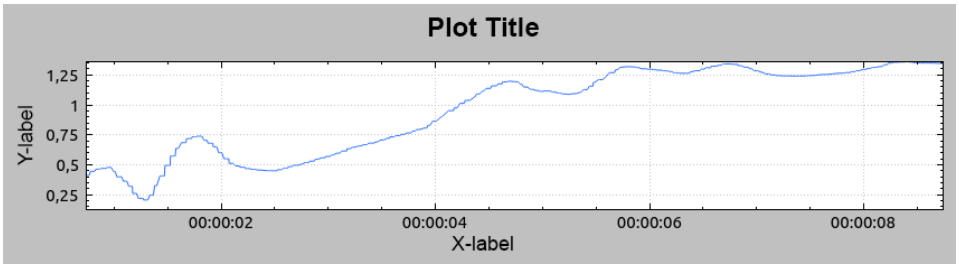

Figure 5.3: Example of a plot made with the QCustomPlot package.

5.2.2 Threads in Qt

Qt also offers methods to organize the code in separate threads. Using threads one can have separate parts of an application running concurrently, which is convenient when the application must perform time consuming computations or blocking operations while still being reactive to real-time events. In Qt, one can let a user defined class `myClass` inherit from the `QThread` class in the basic Qt library, which gives objects of `myClass` the ability to have the overloadable function `void QThread::run()`[11] running from a separate thread.

The signals and slots discussed in Section 5.2.1 can only be handled using dedicated Qt event loops. In the `QThread` class, the blocking function `QThread::exec()` will enter such an event loop, where the `exec()` function will wait for and handle events as they arrive in real-time. Because of this, Qt's signals and slots will not work from an `std::thread`[12] in the standard C++ library, which would be the more traditional way of managing threads in C++ software.

5.2.3 Parameter Plotting with QCustomPlot

QCustomPlot is a third party Qt widget available under GNU GPL for use with C++, and contains functionality for creating advanced plots in Qt applications. An example of a plot made with the QCustomPlot package is illustrated in Figure 5.3. Especially interesting is the ability to make real-time graphs of time varying data and using curves to plot the two dimensional position of an object on a map. QCustomPlot (2017) contains

[11] `http://doc.qt.io/qt-4.8/qthread.html#run`
[12] `http://www.cplusplus.com/reference/thread/thread/`

the full documentation of the widget. To use the widget in a Qt project the developer must simply download the source and header file and include it in the project.

5.3 Choice of Programming Language

It is decided to use ROS and Qt in the implementation of the HIL Simulator, and it is possible to use both Python and C++ with these frameworks. Python is arguably a more high level programming language, relieving the programmer from the low level details such as data types and memory allocation. On the other hand, in the case of artificial AIS data, as discussed in Section 2.2.1, the format of the data packets are defined down to each individual data bit. C++ provides robust functionalities for bitwise operations and is hence considered a more suitable language for these kinds of arithmetics. C++ also provides more explicit control of concurrent threads. It is hence decided to use C++ in the implementation of the HIL Simulator.

Chapter 6

Implementation of the HIL Simulator

This chapter will go through the code design and implementation of the software that constitutes the Surroundings simulator and the GUI. Packages, classes and ROS messages will be discussed in appropriate detail, to form an overview of the structure of the code and dependencies between the software modules. Lastly, the configuration files used to configure the simulations are presented, with information of how to properly initialize the Surroundings simulator with the desired obstacles, other vessels and appropriate sensor noise.

6.1 The General HIL Simulator Software Architecture

The software of the HIL Simulator consists of 3 standalone software modules: the Dynamics simulator, the Surroundings simulator and the graphical user interface (GUI). A visualization of the architecture and information flow between the modules can be seen in Figure 6.1.

The Dynamics simulator will calculate the motion of the USV with respect to the weather conditions and the thruster inputs, and is accounted for in Ødegaard (2017). Utilizing ROS, this module will periodically publish messages with data from the simulated GPS, wind sensor and the inertial measurement unit (IMU). The Surroundings simulator will listen to the GPS messages to track the USV's position in the simulations. This information, combined with information about other simulated objects, is used to generate appropriate sensor readings from the simulated Target Detection Module (TDM). The control system of Odin will expect to receive detected target messages and AIS data, and hence, the Surroundings simulator will publish the data from the TDM and AIS on dedicated ROS topics.

For visualization purposes, the Surroundings simulator will also publish customized ROS messages on a separate topic with detailed and accurate information about the objects' configurations in the simulated environment. For visualization in the first prototype of the HIL Simulator, the GUI module will listen to these messages and use the information to visualize the simulated environment in both a 2D map and a 3D view in ROS/RViz. The GUI will also subscribe to the GPS messages from the Dynamics simulator to put the USV on the map and to plot relevant parameters such as velocity and heading. ROS messages to spawn simulation objects are sent from the GUI to the Surroundings simulator.

6.2 The Surroundings simulator

The key responsibilities of the Surroundings simulator are to manage simulation objects such as ships and fixed obstacles around the USV and to generate appropriate data from an artificial AIS and TDM. With this in mind, the architecture of the software constituting the Surroundings simulator was designed as presented in the combined

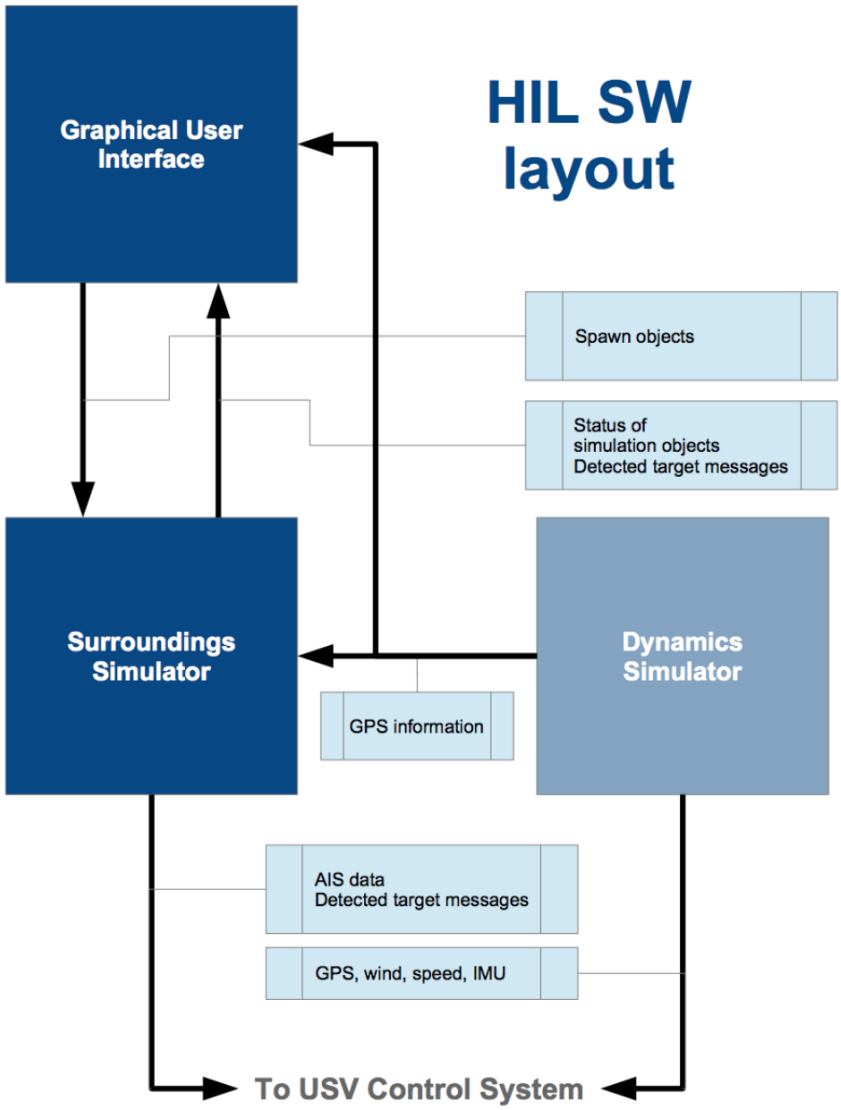

Figure 6.1: Simplified overview of the software constituting the HIL test platform. The software modules colored in dark blue are considered in this report. Light blue is considered in Ødegaard (2017).

package and class diagram in Figure 6.3. A diagram of the flow of ROS messages between modules can be seen in Figure 6.4. For readability purposes, the class descriptions are omitted in these diagrams, but the package and class implementations are further explained later in this section. The arrows used in the diagrams of this thesis are

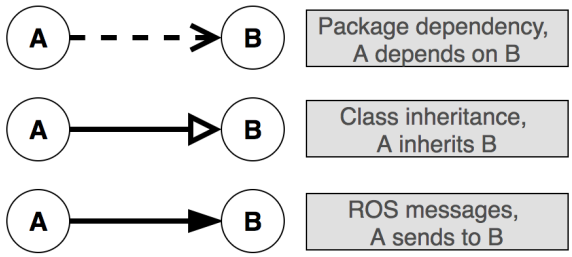

Figure 6.2: Definitions of arrows used in the package, class and message diagrams of this thesis.

explained in Figure 6.2.

The `main()`-function of the Surroundings simulator is located in the `surroundings`-package. To manage the simulation objects, an instance of the `obstacleManager`-class is created. For simulation of sensors, an instance of the `targetDetectionModule`-class is also created. Both classes contain `run()`-functions which handle events such as timeouts and incoming ROS messages. The two objects are run in separate threads as they need to do concurrent work.

An object of the `obstacleManager`-class will spawn and manage simulation objects such as ships and fixed obstacles. A configuration file with information about simulation objects to include in the simulation is read during initialization of this object. The `obstacleManager`-object also subscribes to a dedicated ROS topic, and new simulation objects can be spawned in real-time by passing appropriate messages to this topic. The various simulation objects for use in the simulations are defined in the `simObjects`-package.

The `targetDetection`-package contains the functionality to simulate a credible TDM. An object of the `targetDetectionModule`-class will subscribe to internal position updates from the simulation objects and use this to generate appropriate detected object messages. Functionalities needed to generate these messages with appropriate inaccuracies are assembled in the `detectedObject`-class.

Several classes and packages depend on the `navData`-package. This package contain the `navData`-class, which is used to store and organize navigational data for AIS purposes. AIS data packets can be extracted directly from objects of this class through a dedicated member function.

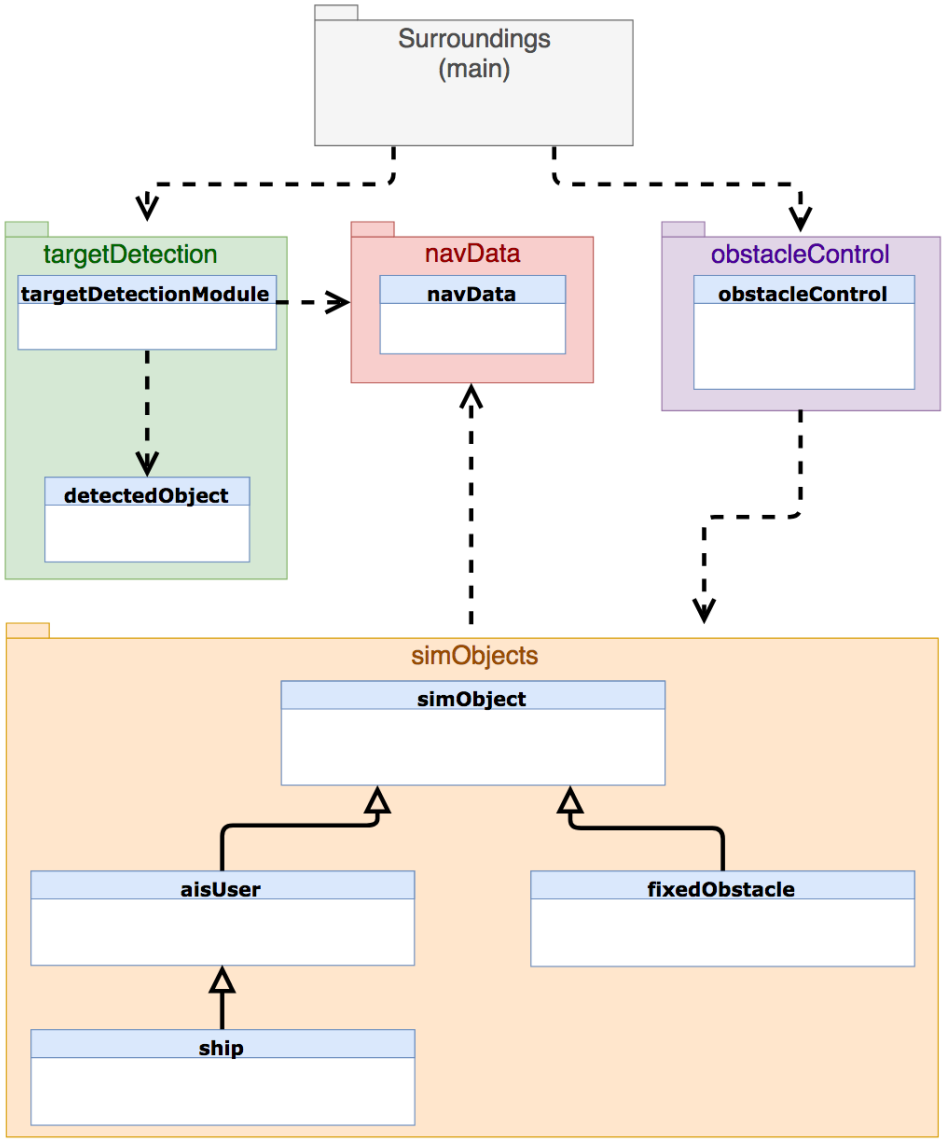

Figure 6.3: Combined package and class diagram describing the general structure and dependencies between packages and classes within the Surroundings simulator.

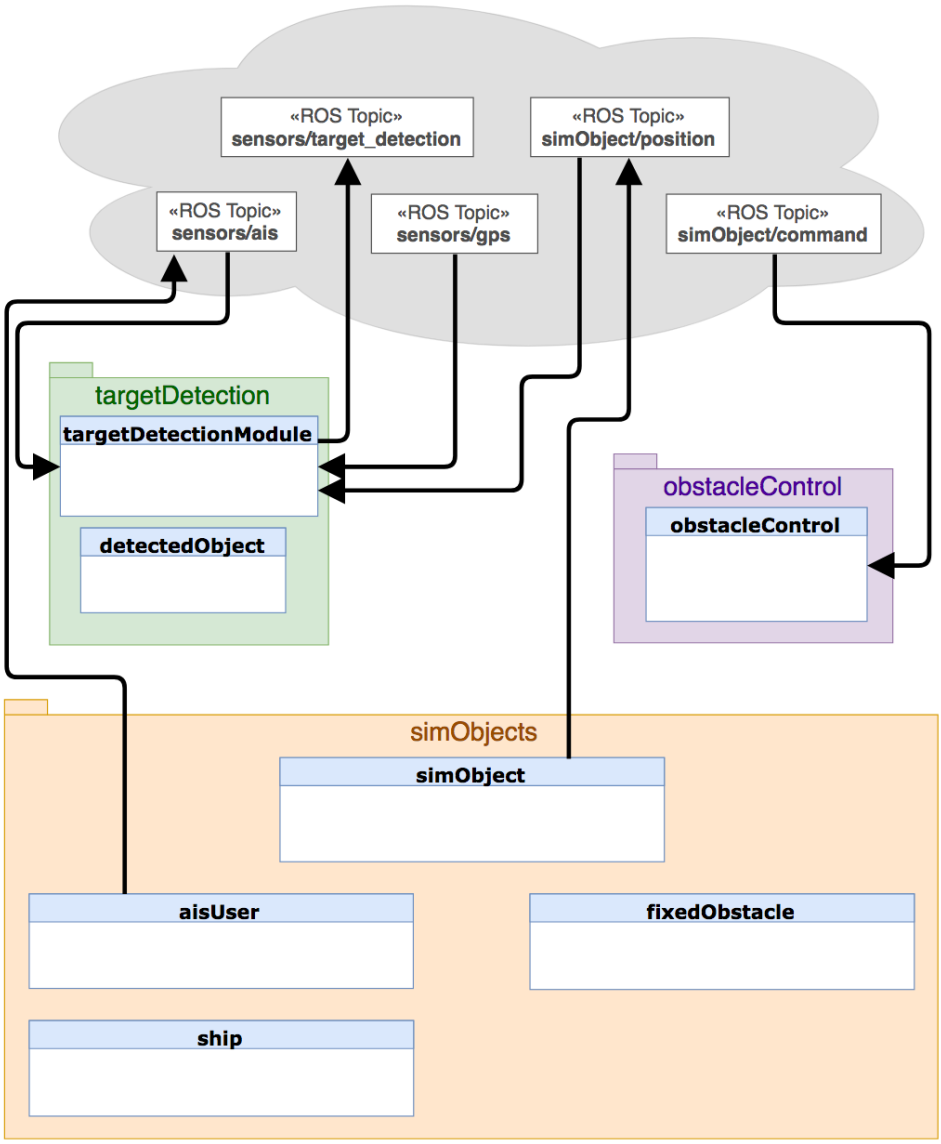

Figure 6.4: Message diagram describing the flow of ROS messages in the Surroundings simulator.

6.2.1 Package: obstacleManager.h

The `obstacleManager`-package contains only the `obstacleManager`-class, and the purpose is to manage the simulation objects throughout the simulations. The package is summarized in Figure 6.5. Instances of this class are meant to run continuously to monitor and manage the simulated objects. Therefore, the class is designed to be run as a separate thread using the `QThread` framework.

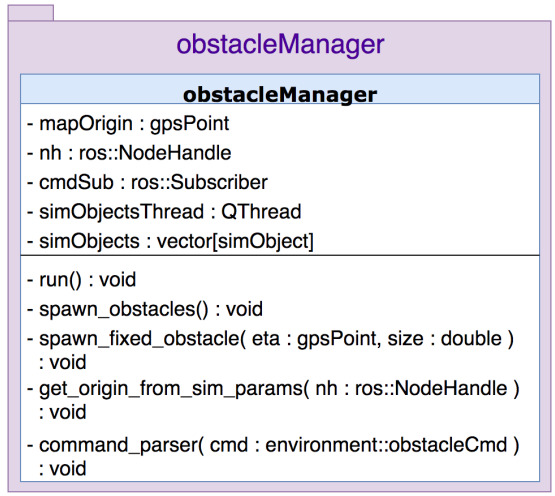

Figure 6.5: Class description of the `obstacleManager`-class.

The `spawn_obstacles()` member function will attempt to find information about obstacles to include in the simulation from the ROS parameter list. A `.yaml` configuration file with this information should be included in the `.launch`-file that launches the HIL Simulator. The details of the configuration file are discussed in detail later in this chapter. Instances of the `obstacleManager`-class subscribes to the ROS topic `/simObject/command`, which can be used to spawn new objects in real-time through the use of the associated `obstacleCmd` message type.

The simulation objects are run as separate processes as they also need to do their work concurrently. A pointer to each object is stored in a dynamically allocated array.

6.2.2 Simulation Objects

The `simObjects`-package contain class declarations for several simulation objects and is described in the class diagram of Figure 6.6. The purpose of the package is to provide classes that form different simulation objects for use in the simulations. The classes are designed to run independently through a `run()`-function using the `QThread` framework.

`simObjects::simObject`

The base class is the `simObject`-class, which contain the basic properties and functionality for all objects to exist in a simulation. The class contains the virtual function `run()`, which has no declaration in the base class. `simObject` is hence an abstract class and can not be instantiated directly.

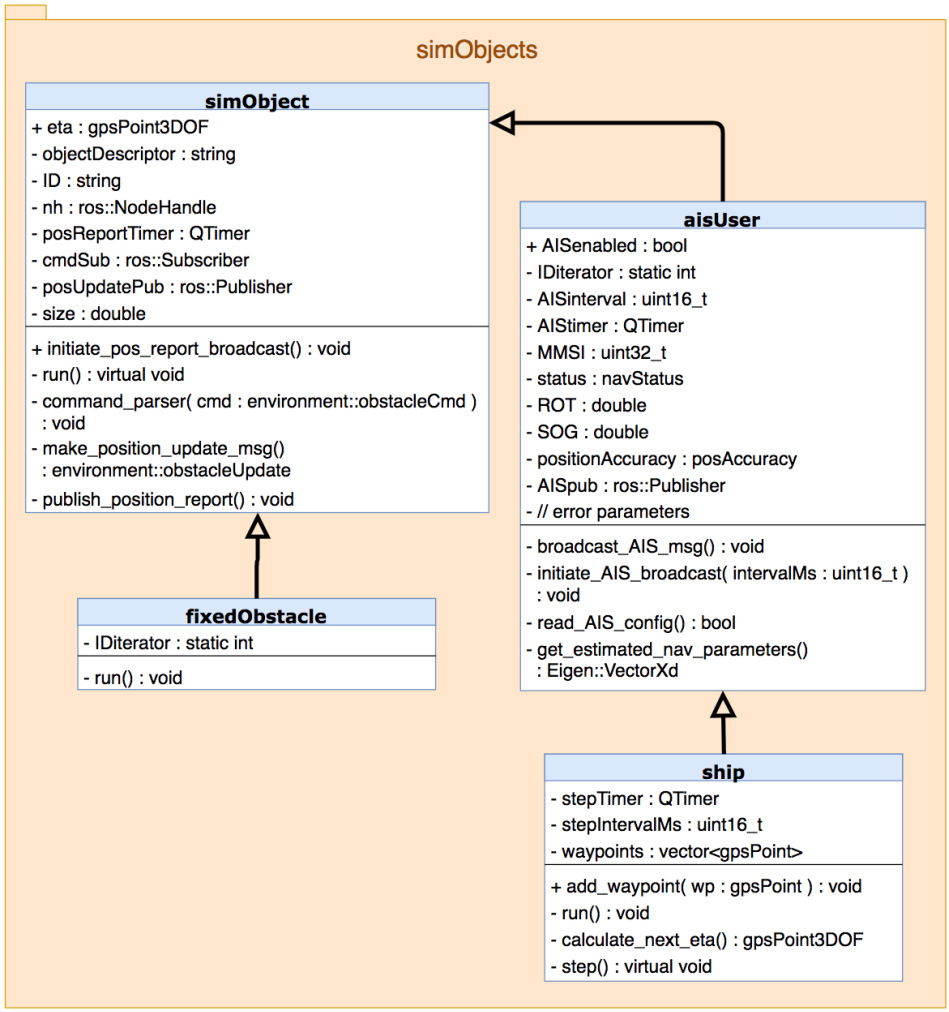

Figure 6.6: Class diagram of the `simObject`-classes declared in the `simObjects`-package.

All simulation objects will periodically broadcast their position, orientation and size, both for visualization purposes, and for simulation of sensors that track the ob-

jects throughout the simulation. Hence, they need to store their 3DOF orientation η (`eta` in the class diagram) in the simulated environment. The base class takes care of the periodic position reports. The position reports are broadcast along with an object descriptor and a unique ID, so that the receiver will know what and who sent the report. It is the responsibility of any derived class to define an appropriate object descriptor and ID.

simObjects::fixedObstacle

The subclass `fixedObstacle` is the simplest of the simulation objects. It does not move, so the `run()`-function does nothing but wait, while the position reports are automatically published.

simObjects::aisUser

For simulation of objects that utilize AIS, a dedicated `aisUser`-class is derived from the basic `simObject`. `aisUser` is also an abstract class as it does not implement the `run()`-function, but it contains all the necessary functionality to broadcast artificial AIS data packets. Classes that derive from `aisUser` can broadcast periodic AIS ROS messages by running the protected `initiate_AIS_broadcast()`-function. The broadcasting of AIS messages can be switched off with the `AISenabled` member variable. AIS ROS messages of type `simulator_messages::AIS` are published to the `sensors/ais`-topic.

To contaminate the AIS messages by appropriate inaccuracies, the `get_estimated_nav_parameters()`-function will add small errors to the true values as described in Section 2.5. The bias and white noise parameters are defined in a `.yaml` configuration file launched with ROS and interpreted in the `read_AIS_config()`-function.

simObjects::ship

Currently, the only simulation object that derives from `aisUser` is the `ship`-class. Several other classes might be useful in the future as the simulations become more advanced, such as other vessels with different characteristics, lighthouses or oil platforms.

The `ship` class is designed to move between waypoints in a straight line with constant speed. The speed of the ship is set in the constructor function of the class. New waypoints are easily added through the `add_waypoint()`-function, and the simulated ship will move to the waypoints in the added order. When changing heading, as is usually the case when a new waypoint is active, the heading of the ship is slowly changed through a regular discrete-time low-pass filter until the heading equals the bearing to the active waypoint. Because it derives from `aisUser`, the ship will periodically publish AIS messages throughout the simulation.

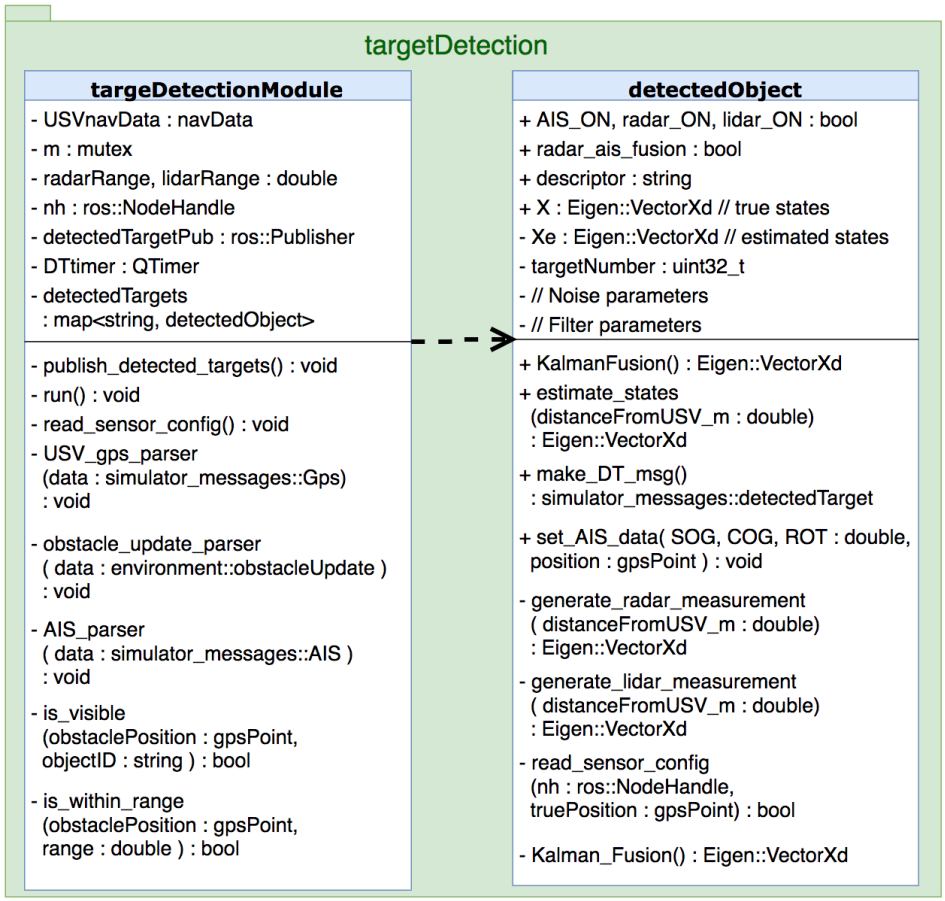

Figure 6.7: Class diagram of the `targetDetection`-package. Some of the functions and parameters are generalized to save space.

6.2.3 Simulation of the Target Detection Module

The purpose of the `targetDetection`-package described in Figure 6.7 is to provide functionality to simulate a realistic Target Detection Module (TDM) based on the USV's and simulated objects' positions in the simulated environment. The mode of operation and the suggested data output of the TDM was discussed in Chapter 3.

`targetDetection::targetDetectionModule`

The information and functionalities needed to simulate the TDM are assembled in the `targetDetectionModule`-class. Instances of this class are designed to run continuously using the member function `run()` utilizing the `QThread` framework. The `run()`-function will subscribe to GPS data from the USV, AIS messages and position reports from simulated objects and evaluate whether these objects will be detected by the TDM. The limitations of the AIS, radar and LiDAR are taken into account in this evaluation.

An object that transmits AIS messages will always be detected if it is inside the AIS range defined in the configuration files. If the object is inside the radar or LiDAR range, the object will be detected by these sensors only if it is directly visible from the USV. That is to say, objects in the radar shadow will not be detected. Whether an object is in the radar shadow of another object is calculated in the `is_visible()`-function, which estimates the associated radar shadow of each single object in the simulation to investigate if a specific object is within visibility. The estimated radar shadow is illustrated in Figure 6.8.

If within detectability, the object will be stored in the `detectedTargets`-map. Once they are no longer within detectability, they will be removed from the map. The `targetDetectionModule`-object will periodically publish detected object messages and publish them on the ROS topic `sensors/target_detection` with the `publish_detected_targets()`-function. The detected object messages are extracted from instances of the `detectedObject`-class, which is documented below.

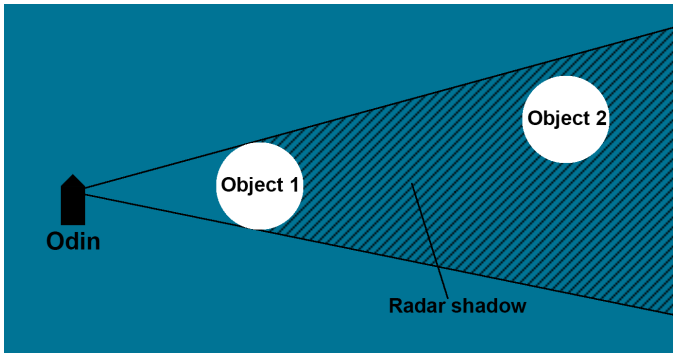

Figure 6.8: Illustration of the implementation of radar shadow in the simulator. Object 2 will not be detected by the radar or LiDAR because it exists in the radar shadow of Object 1.

`targetDetection::detectedObject`

Information and convenient functions of a detected object are collected in a dedicated `detectedObject`-class. This class stores the navigational data of the detected object and makes sure that each object is assigned a unique target ID.

Based on the navigational data of the object, artificial radar and LiDAR measurements are made if the object is detectable by these sensors on Odin, and noise as defined in (2.4) is added to the measurements. The owner of an instance of this class will manually set the AIS data associated with the detected object, and the data obtained from AIS is already influenced by noise. Extended Kalman filters are applied to the artificial data from AIS, radar and LiDAR, individually, and the Simple Fusion algorithm is used to compute the weighted average of the output from the Kalman filters. The output from the Simple Fusion is used to define the data of detected object ROS messages.

Detected object ROS messages of type `simulator_messages::detectedTarget` can be extracted directly from instances of the `detectedObject`-class through the `make_DT_msg()`-function. The messages have the same structure as suggested in Table 3.1.

The error parameters of the measurements from radar and LiDAR are read from the ROS parameter list, so a `.yaml` configuration file with this information should be included during launch of the HIL Simulator. The format of the error parameter list is discussed later in this chapter.

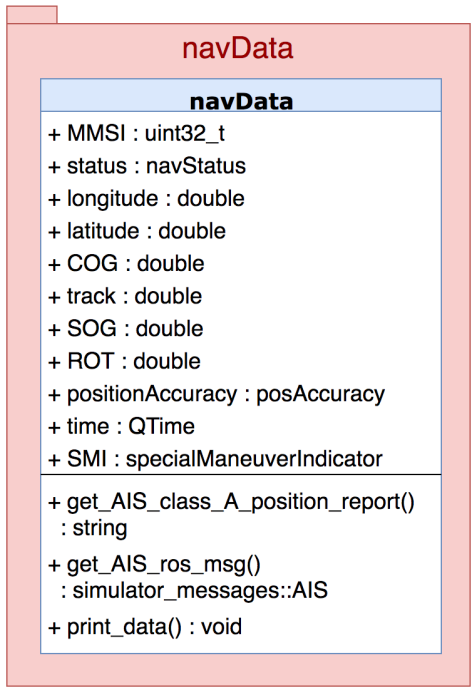

Figure 6.9: Class description of the `navData`*-class.*

6.2.4 Navigational Data

Several packages depend on the `navData`-package, which is a generalized interface for easy handling of navigational data. All data needed in a standard AIS message are stored as member variables of the `navData`-class. Enumeration types are included in the package to simplify the declaration of data fields such as navigation status, position accuracy and the Special Maneuver Indicator (SMI), all accounted for in E. S. Raymond (2016).

A basic AIS Class A position report, as described in Section 2.2.1, can easily be extracted from objects of this class using the public member function `get_AIS_class_A_position_report()`. The AIS message obtained from this function is complete and valid with the NMEA 0183 data-integrity checksum. A ROS message containing the raw AIS message as well as the included data represented as individual data fields can also be obtained using the function `get_AIS_ros_msg()`. The ROS message is accounted for later in this chapter.

6.3 The Graphical User Interface (GUI)

Using Qt, a Graphical User Interface (GUI) was developed to visualize the simulations. It serves only as a visualization tool for proof of concept purposes, and it is likely that the developers of Odin will visualize the simulations through the use of visualization tools already developed for mission control of the AUV Hugin, by Kongsberg Maritime.

The GUI was designed to be a standalone module, independent from both the Dynamics and the Surroundings simulator. Apart from being able to request new obstacles at specified positions, the GUI does nothing but visualizing the simulation data already available on the various ROS topics. As the main product of this master's thesis is the Surroundings simulator, only superficial details of the software constituting the GUI will be accounted for here. A combined package and class diagram of the GUI software can be found in Figure 6.10. Similarly, a message diagram describing the flow of ROS messages related to the GUI can be found in Figure 6.11.

6.3.1 The Main Window

The `main()`-function of the GUI will create an object of class `MainWindow`. This object will open and initialize the GUI window. The window will contain real-time plots, visualizing data from the simulations such as speed and heading of the USV. It will also have a 2D map of of the simulation as seen from above and a "Spawn new obstacles"-interface. The main window also owns a position update handler, an object of type `posUpdateHandler`, listening to ROS messages with information about the position and orientation of objects in the simulations.

6.3.2 Real-Time Plots

The `realTimePlot`-package contains a class to easily create a real-time plot widget using the Qt framework. The plot can be initialized with title and labels, and the values of the plot can be updated through a dedicated member function. The time axis will scroll automatically as the time goes by.

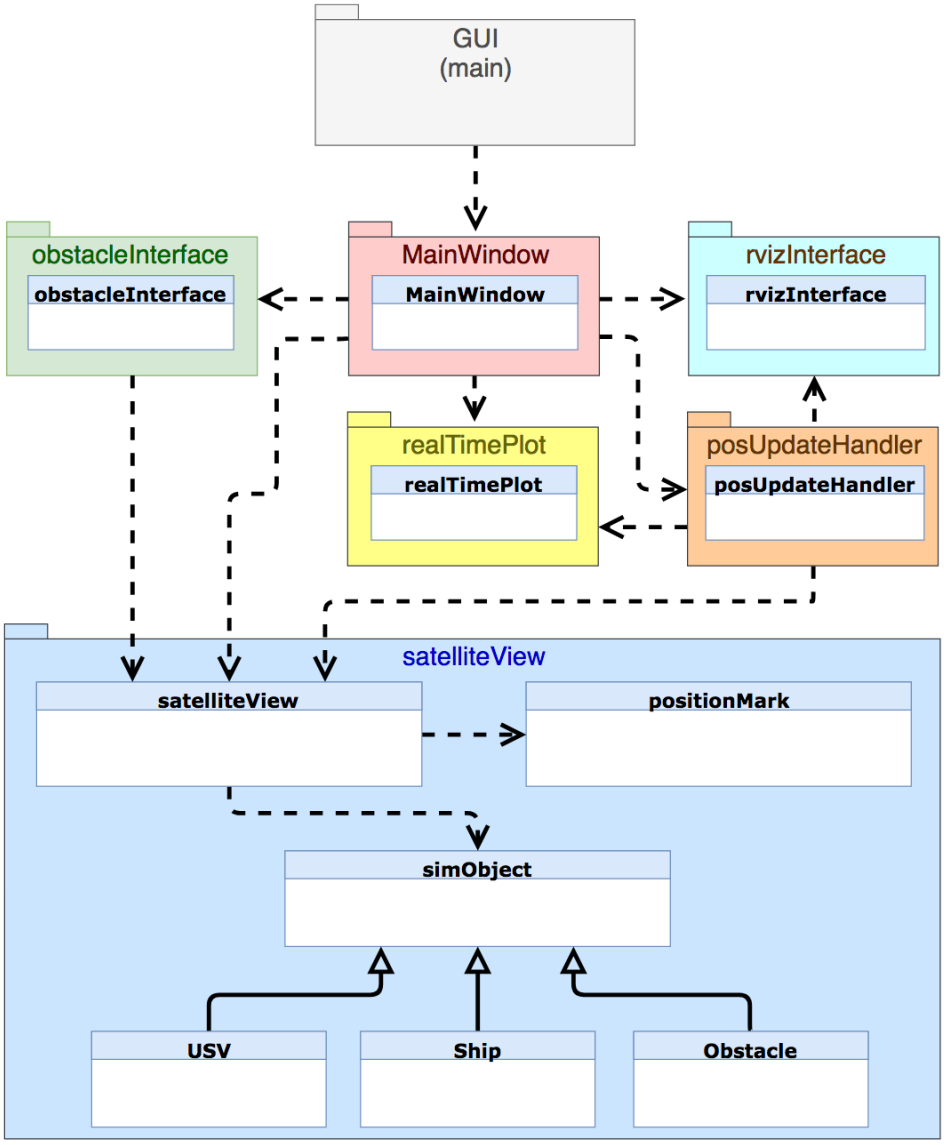

Figure 6.10: Combined package and class diagram describing the general structure and dependencies between packages in the GUI module.

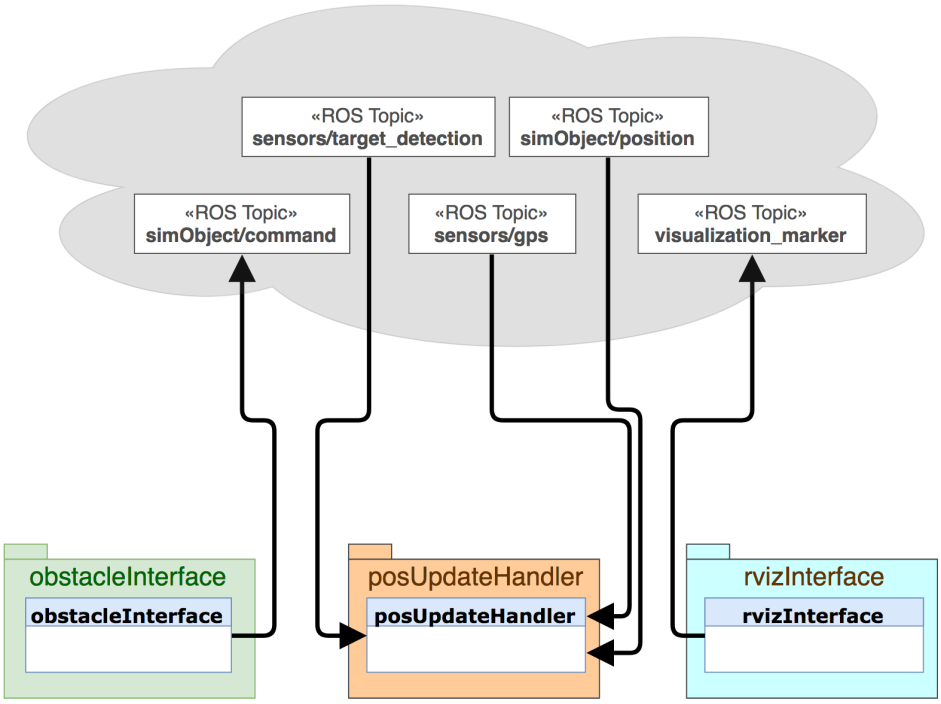

Figure 6.11: Message diagram describing the flow of ROS messages in the GUI module.

6.3.3 The Obstacle Control Panel

The `obstacleInterface`-package contains a class that represents a control panel for control of obstacles. The initial goal of this panel was to provide advanced manipulation options of objects in the simulations, such as adding, deleting and moving obstacles around in the simulated environment. However, it became clear during the project that not all of these functionalities were needed in the first prototype of the HIL Simulator, as all included objects in the simulations will be defined beforehand in a configuration file. Hence, the obstacle control panel at the current time only provides a button for spawning of new objects at positions marked in the 2D map. This was convenient throughout development of the HIL Simulator to easily spawn new objects in real-time.

6.3.4 The Position Update Handler

The `posUpdateHandler` runs as a separate process listening to ROS messages on several topics. Specifically, it subscribes to messages from the GPS topic to visualize the USV's whereabouts, the detected target topic to illustrate the detected targets in the simulated environment and the position update topic used by simulation objects in the Surroundings simulator to broadcast their exact position and orientation at all times. The `posUpdateHandler` uses this information to update the real-time plots and the 2D map of the main window. It also owns an object representing an interface to RViz, so that the information obtained also can be visualized in 3D.

6.3.5 The RViz Interface

To visualize the simulation in 3D, RViz was considered to be a suitable platform. The package `rvizInterface` provides an interface to visualize the simulated objects in RViz. Using member functions of the included class with the same name one can set the position of the simulated objects as well as the detected targets. To ensure that the Dynamics simulator works as a standalone package, the Dynamics simulator interfaces with RViz on its own to set the position and orientation of the USV.

Table 6.1: Format of the `obstacleCmd` ROS message.

obstacleCmd.msg		
Data Type	**Variable Name**	**Unit**
string	cmdSpecifier	-
string	receiverID	-
float64	x	[deg]
float64	y	[deg]
float64	psi	[deg]

6.3.6 The 2D Map

To visualize the simulation in a 2D map, the `satelliteView`-package provides functionality to setup a customized Qt widget to plot the position and heading of the USV as well as other simulated objects, such as ships and fixed obstacles. The included class `satelliteView` has member functions to set and update the position of objects on the map, to mark positions with mouse clicks (for use by the obstacle interface to spawn new obstacles) and to zoom in and out.

During development of the 3D visualization of the simulations, it became clear that the 2D map became somewhat redundant as the same information became visible in RViz, which is even more interactive and intuitive to use. The development of the 2D map was hence put on hold, but was kept as a part of the GUI in its current form.

6.4 ROS Messages of the Surroundings simulator

The following section will go through the different ROS messages used in the simulation of the USV Surroundings.

6.4.1 The Obstacle Command ROS Message

The command message `obstacleCmd` is primarily used for spawning of new simulation objects. The `cmdSpecifier`-field is used to specify what kind of command the message carries. Currently, the only command type in use is "spawn", but other possible command types could be "delete", "moveTo", etc.. All simulation objects subscribe to the `simObject/command`-topic and wait for such messages. If the `receiverID`

matches the object's own ID, the object will respond to the command. For message types that specify a position and heading, as is the case with the "spawn"-command, the position and heading are defined with the `x, y` and `psi` data fields. New command types can easily be implemented by manipulating the `command_parser()`-function of the `simObject`-class to also interpret messages of the new command specifier.

6.4.2 The Position Update ROS Message

All simulation objects defined by classes derived from `simObject` will periodically broadcast their position and heading using the ROS message `obstacleUpdate`. This message carries the position and heading, as well as the ID of the transmitting object. The message descriptor data field `msgDescriptor` makes it possible to define several different update messages, such as position updates, waypoint reached etc.. The only message type currently in use is the position update message, specified by the string "posUpdate" in the `msgDescriptor` field. The `objectDescriptor`-field renders it possible for the subscriber of these messages to separate between fixed obstacles and vessels, which is of interest for visualization purposes in the GUI. Current position, heading and size of the object are specified with the `longitude`, `latitude`, `heading` and `radius` fields.

Table 6.2: Format of the `obstacleUpdate` ROS message.

obstacleUpdate.msg		
Data Type	**Variable Name**	**Unit**
string	msgDescriptor	-
string	objectDescriptor	-
string	objectID	-
float64	radius	[m]
float64	longitude	[deg]
float64	latitude	[deg]
float64	heading	[deg]

Table 6.3: Format of the `detectedTarget` ROS message.

detectedTarget.msg		
Data Type	**Variable Name**	**Unit**
int32	targetID	-
string	objectDescriptor	-
float64	longitude	[deg]
float64	latitude	[deg]
float64	COG	[deg]
float64	SOG	[m/s]
float64	crossSection	[m^2]

6.4.3 The Detected Target ROS Message

The detected target message described in Table 6.3 is published from the simulated Target Detection Module (TDM) and has the same format as suggested in Table 3.1, with the exception of the included `objectDescriptor`-field. This field is reserved for a written classification of the object. Detected objects in the first prototype of the HIL Simulator are classified as either vessels or fixed obstacles, but other future classifications could for example be buoys, ice bergs or leisure boats. It is reasonable to believe that the real TDM will classify detected objects in such a manner, as the USV should behave differently around larger vessels and simple buoys.

6.4.4 The AIS ROS Message

An AIS ROS message is broadcast periodically from all simulation objects defined by classes derived from the `aisUser`-class. The messages contain navigational data as included in a regular AIS Class A position report as well as the raw AIS message string as described in the Section 2.2.1.

Table 6.4: Format of the `AIS` ROS message.

AIS.msg		
Data Type	**Variable Name**	**Unit**
uint32	MMSI	-
uint8	status	-
float64	ROT	[deg/min]
float64	SOG	[knots]
uint8	positionAccuracy	-
float64	longitude	[deg]
float64	latitude	[deg]
float64	COG	[deg]
float64	track	[deg]
uint8	hour	[hr]
uint8	minute	[min]
uint8	second	[s]
uint8	SMI	-
string	raw_data	-

6.5 Configuration Files

Configuration files of type `.yaml` can be included during launch of ROS, and may contain declarations of parameters, which will be added to the ROS parameter server. The parameters in the parameter server can be extracted from any ROS node handle through the use of the function `ros::NodeHandle::getParam()`. For readability, the configuration parameters of the Surroundings simulator in the HIL Simulator are declared in 2 separate configuration files: one for obstacle parameters and one for sensor sensor parameters. The files and their parameters are accounted for in this section.

6.5.1 Obstacle Parameters

The obstacle parameters needed for initialization of the simulation objects must comply with the protocol described and exemplified in Listing 6.1.

The fixed obstacles included in the simulation must be defined in a map called `fixed_obstacles`. Both keys and values in the map are of type `string`. The key defines a temporary name of the object and is only used for readability purposes; the name is not used in the simulation. The associated value is a string which defines the radius and coordinate of the object. The radius field contain the radius in meters of the obstacle, while the coordinate field gives the coordinate of the center point of the object in decimal degrees. The resulting obstacle is represented as a solid disc with the defined radius at the defined coordinate.

The ships included in the simulations are defined in a similar map called `ships`, with the key being a string representing a temporary name of the ship. The value parameter is also a string, defining the length in meters, speed in knots and a vector of waypoints in decimal degrees. The waypoints are added to the simulated ship's waypoint vector in the same order. The first waypoint will be the starting position of the ship.

The obstacle parameters of the default scenario of the HIL Simulator available on `github.com/kjetilbl/hilsim` are located in `obstacle_params.yaml` in the environment/config-folder.

```
fixed_obstacles : {
  "obstacle_1" : "RADIUS=30 COORD=10,4613435/59,444234"
  "obstacle_2" : "RADIUS=25 COORD=10,4515451/59,434341"
  # add as many obstacles as preferred...
}

ships : {
  "ship_1" : "LENGTH=50 SpeedInKnots=19 WP=10,46715/59,44006 WP=10,46146/59,43564"
  "ship_2" : "LENGTH=20 SpeedInKnots=21 WP=10,46146/59,43564 WP=10,46715/59,44006"
  # add as many ships as preferred...
}
```

Listing 6.1: Protocol of parameters for initializing simulation objects in the HIL Simulator.

6.5.2 Sensor Parameters

The sensor parameters define the noise conditions by which the AIS, radar and LiDAR are influenced, as well as the limitations of each sensor system. The protocol of the pa-

rameters is exemplified in Listing 6.2. The ranges of each sensor system are defined in meters, and will influence whether objects far away from the USV will be detected or not. The error per distance gain is dimensionless and corresponds to k in (2.5). Comparing the parameters of Listing 6.2 to the sensor noise model in (2.1) - (2.3), the measure variances correspond to the variance of w_m, the bias variances correspond to the variance of w_b and the bias time constants correspond to the time constants of T_b.

```
AIS_range_m: 10000
radar_range_m: 5000
lidar_range_m: 100
TDM_update_rate_Hz: 1
error_pr_distance_gain: 0.01

# Radar error parameters:
radar_position_measure_variance_m2: 1
radar_radius_measure_variance_m2: 5
radar_position_bias_variance_m2: 0.5
radar_radius_bias_variance_m2: 2
radar_position_bias_time_constant_sec: 10
radar_radius_bias_time_constant_sec: 5

# Similar parameters are defined for AIS and LiDAR...
```

Listing 6.2: Protocol of sensor parameters for initialization of the simulation.

Chapter 7

Running the HIL Simulator

In this chapter, the performance and functionalities of the first prototype of the HIL Simulator are demonstrated by example. Emphasis is put on the Surroundings simulator and the GUI, which are the products of this thesis. An open sea scenario with presence of icebergs and other vessels are simulated, with Odin navigating in the simulated environment using predefined thrust commands logged from a real mission. Data from the simulated TDM is visualized by the GUI using RViz, and data from the AIS is printed to the Terminal for analysis of the performance of these sensor systems.

7.1 Setting Up the Simulator

The simulator can be downloaded from github[1] and should be installed on a computer running the Linux Ubuntu operating system which has ROS installed. The details of how to configure a ROS workspace and install the HIL Simulator can be found in Appendix A.

An important note is that the 3D models representing Odin and Jolner are confidential. Users of the HIL Simulator must request the models from FFI or Kongsberg Maritime, or use other 3D models to represent the target vehicle in the 3D visualization of the simulation.

7.2 Simulating a Busy Open Sea Scenario

By example, this section will provide the details of how to configure a customized simulation scenario. As Odin's control system currently is unable to receive information from any of the sensors considered in this thesis, it is not useful for the control system to interact with the simulation. A log file with thrust inputs from a real mission is hence used to enable the USV to navigate in the simulated environment. Because of this, Odin can be considered blind in this simulation, with no feedback from neither the proprioceptive or exteroceptive sensors. Nevertheless, the simulation will still visualize the output from the sensors to illustrate what the control system would be able to see if it was able to interpret this information.

Now, consider a mission where Odin will navigate at open sea, e.g. to search for possible naval mines, during moderate weather conditions and under presence of several icebergs. A support vessel is operating nearby, and an unrelated research ship is passing by at cruise speed. To simulate this mission, one must define the configuration parameters to create the desired obstacles, vessels and conditions of the scenario. The parameters, as defined in the configuration files used in this simulation, can be found in Appendix B.

The noise parameters of the sensors in use were inspired by the limitations of each sensor, presented in sections 2.2-2.4. To visualize the behavior of the simulated sensors

[1] `https://github.com/kjetilbl/hilsim`

when influenced by noise, some of the parameters were tuned to yield a worse behavior than what can be expected in a real scenario. Similarly, based on the dynamics of the biases accounted for in Section 2.5, some of the bias parameters were also tuned to give a behavior that visualizes the effect of sensor deviations. The wind was defined as 10m/s from the east, the current as 0.5m/s from the west, and the waves were defined to have 2m significant wave height, coming from the east. The influence of the weather parameters on the dynamics of the USV is accounted for in Ødegaard (2017).

A `.launch` file was created to initialize ROS with the configurations files summarized in Appendix B, as well as starting the Surroundings, GUI and Dynamics simulators. The configuration and launch files used in this example are included in the "config"- and "launch"-folders of the Surroundings simulator, available on github.

7.3 Running the Simulation

Launching the `.launch` file from Terminal starts the simulation. The RViz and GUI application windows appear as two separate windows which both illustrate the simulation, as seen in figures 7.1 and 7.2, respectively. As expected, Odin now finds itself at sea, surrounded by several obstacles and two other vessels as defined in the configuration files. The two nearby vessels appear at their first waypoint and travel at constant speed in a straight line to their next waypoint. The thrust input file feeds the Dynamics simulator with thrust commands, which enables Odin to move around in the simulated environment.

7.3.1 3D Visualization in RViz

The main feature of RViz is the ability to visualize the simulation in 3D, as illustrated in Figure 7.3. The fixed obstacles are visualized as white cylinders representing icebergs, with radius and position as defined in the configuration file. The vessels are represented as 3D models of simple ships[2], with lengths also as defined in the configuration file. A model of Odin is used to visualize Odin's position and attitude, with arrows connected to the thrusters to visualize the direction and force of thrust. The arrows are visible

[2]The 3D models of the included ships are available for free at `www.turbosquid.com`

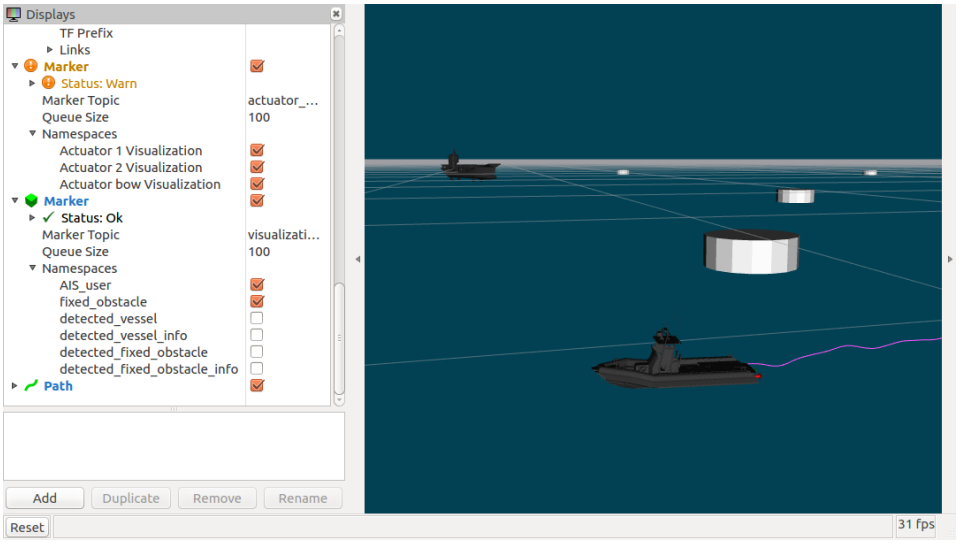

Figure 7.1: The RViz window visualizing the simulation in 3D, with the control panel to the left providing options to show and hide groups of simulation objects. Fixed obstacles, represented as white cylinders, as well as a nearby vessel, are visible around Odin.

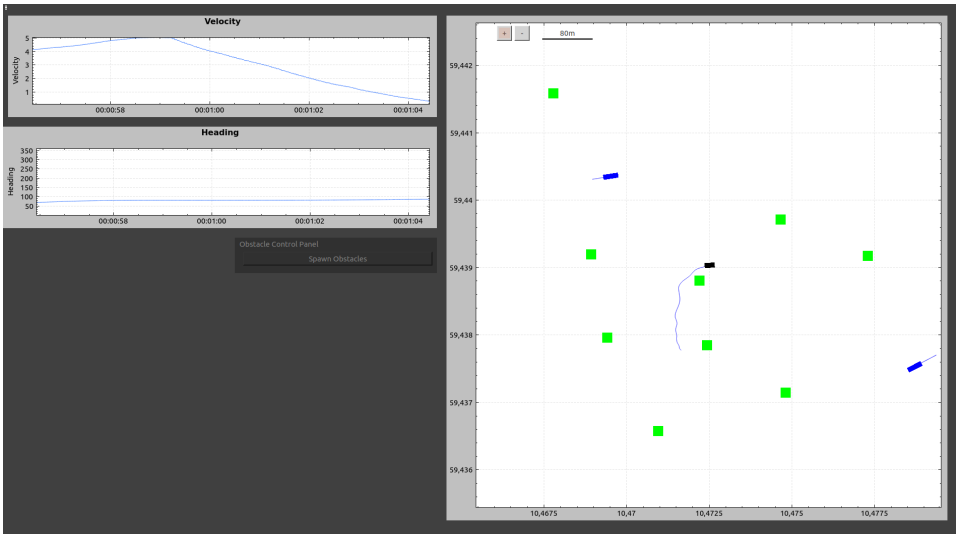

Figure 7.2: The GUI application window with real-time plots of the heading and velocity of Odin and a 2D map giving an overview of the simulation.

in Figure 7.4, which also illustrates the representation of detected objects by the TDM.

The user can manipulate the 3D view in RViz with mouse movements to change the

view angle, zoom in and out and choose to hide or show different groups of objects in the simulation.

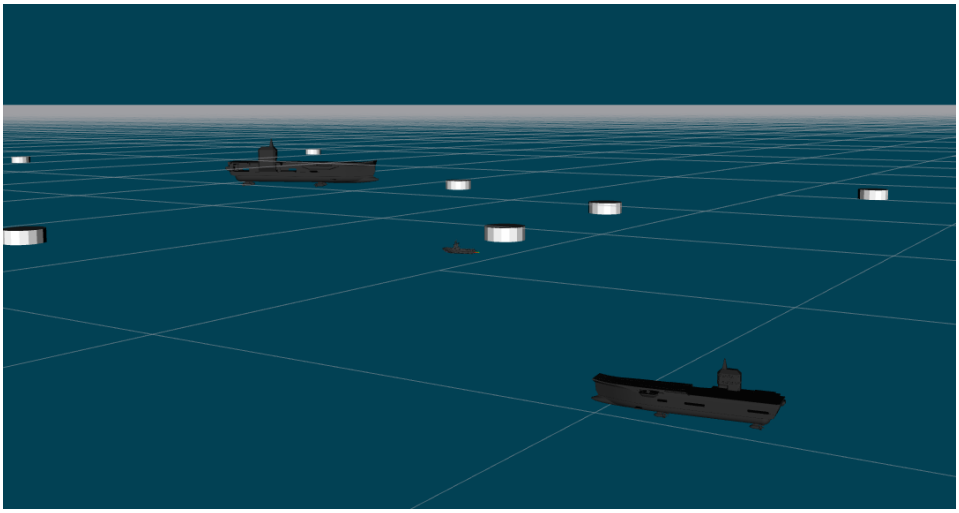

Figure 7.3: Odin in the simulated environment, seen in the center of the figure, surrounded by icebergs and other vessels.

The information in the detected object messages published by the simulated TDM is visualized in RViz as partially transparent, orange discs with estimated information as text hovering above the assumed position of the object. The sizes and positions of the discs are defined by the estimated data in the associated messages. For detected vessels, the discs are replaced by oblong ellipses, representing the estimated length of the vessel and oriented to line up with the estimated heading. This is illustrated in figures 7.4 and 7.5. The visualization of detected objects can be turned on and off in the RViz control panel to the left in Figure 7.1.

The track history of Odin can also be illustrated, as seen in Figure 7.5. A purple line will appear, illustrating the path of Odin as the simulation evolves.

7.3.2 Visualization Using the GUI

The GUI provides a 2D map and plots of the velocity and heading of Odin, as a proof of concept as to what one can visualize using real-time information obtained from the simulation. To create new obstacles as the simulation goes, the user can click on the 2D

Figure 7.4: Odin during simulation of the busy open sea scenario. Detected objects broadcast by the simulated TDM are illustrated as partially transparent, orange discs, representing the estimated size, position and heading of the object. A descriptive keyword, the object's given ID, COG and SOG are illustrated as text hovering above the estimated position of the detected object. Arrows representing the angle and force of thrust are connected to Odin's thrusters.

map at the desired position of the new obstacle. A blue circle will appear, as illustrated in Figure 7.6, indicating that the position is marked. Several positions may be marked at the same time. By clicking the "Spawn obstacles"-button, located in the center of the application window illustrated in Figure 7.2, a request is sent to the Surroundings simulator to create fixed obstacles at the marked positions.

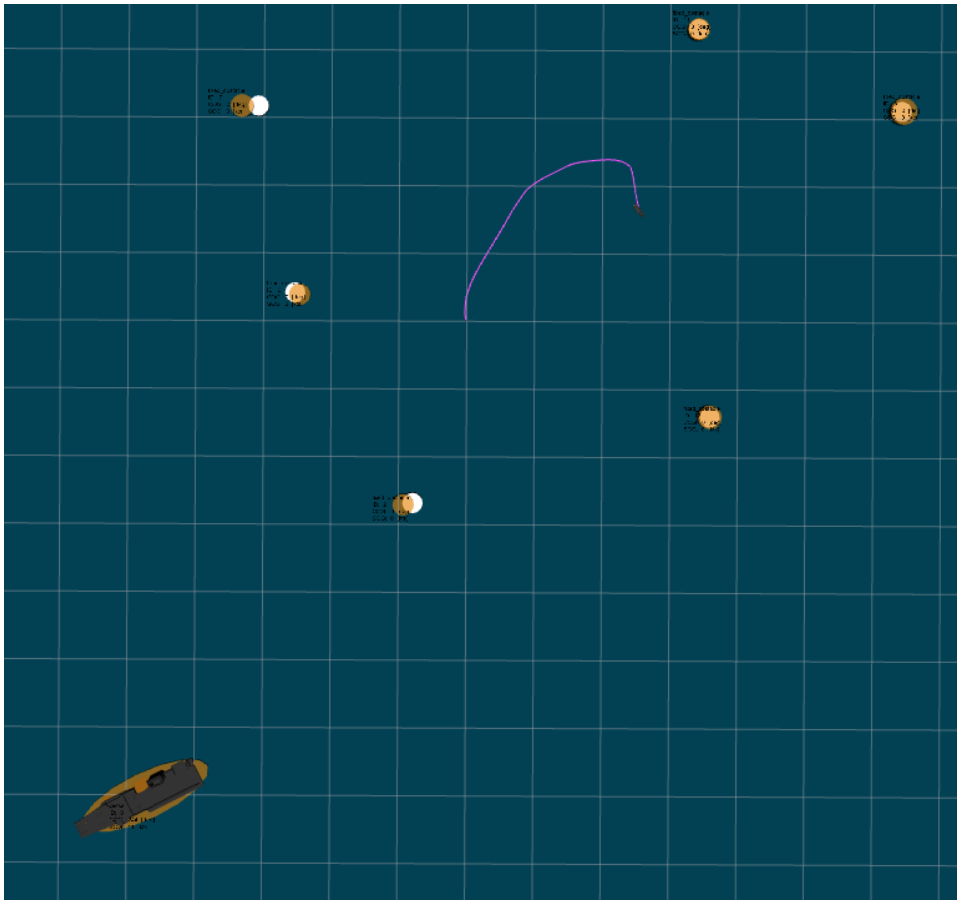

Figure 7.5: The busy open sea scenario seen from above, illustrated in RViz. The path of Odin is visible as a purple line.

7.4 Performance of the Simulated TDM

The output of the simulated TDM is visualized in RViz. The purpose of the simulated TDM is not to be perfect, but to behave in accordance with the specifications and limitations discussed in Section 3.1. By comparing detected objects relative to their true counterparts, the performance of the simulated TDM can be visually inspected. Partially transparent, orange discs and compact text are used to represent the information of detected objects received from the detected object ROS topic.

As seen in Figure 7.7, the estimated size and position of the obstacle in the front deviates from the true size and position of the associated iceberg. The inaccuracy is

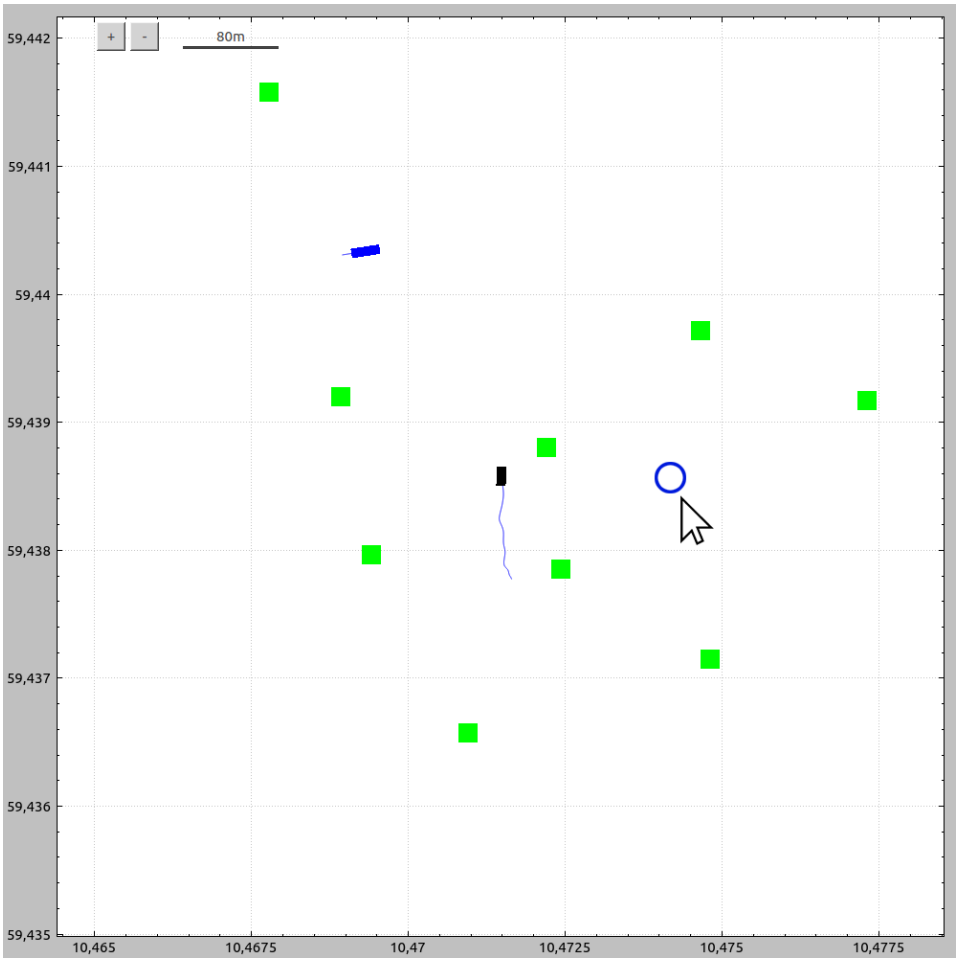

Figure 7.6: The 2D map included in the GUI. Green rectangles illustrate fixed obstacles, blue rectangles illustrate other simulated vessels. Odin is locked to the center of the map, represented by a black rectangle. The tracks of Odin and other vessels are visible as blue lines. A mouse click on the map will mark a position with a blue circle. By clicking the "Spawn obstacles"-button, a request to create an obstacle at this position is sent to the Surroundings simulator.

modeled as in Section 2.5, with increasing errors with larger distance between the USV and the obstacle. At close range, the LiDAR assists in the tracking of nearby objects, which enables accurate estimations of these objects. In the case of a moving vessel utilizing AIS, the AIS data is combined with the radar measurements, which enables fairly accurate tracking of these vessels in regards to position, heading and speed.

The effect of radar shadow, as discussed in sections 2.3.1 and 6.2.3, is apparent in

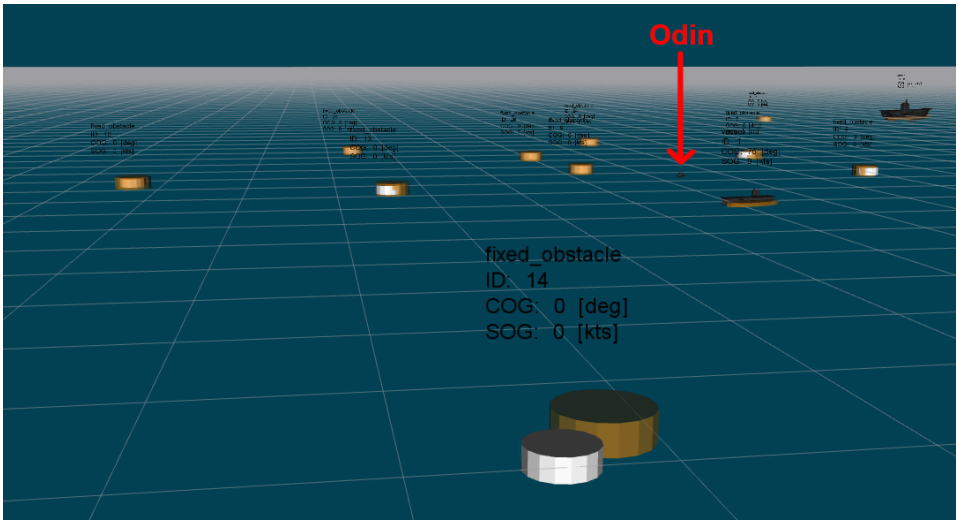

Figure 7.7: Illustration of how the accuracy of the position and size estimates of detected objects increases with the distance from the USV. Odin is barely visible in the top right corner of the picture (red arrow). At this distance, only the radar is used for position estimates.

the simulated TDM. If an object is located behind another larger object as seen from the USV, the radar will not detect this object. Figure 7.8 illustrates this effect, where an iceberg is not detected by the TDM because it is located behind a larger vessel.

Based on the above mentioned observations, it is confirmed that the TDM behaves in accordance with the properties listed in Section 3.1.2.

7.5 Performance of Simulated AIS

The vessels included in the simulation are simulated with the option of continuously broadcasting AIS messages, with a few seconds between transmissions. Small errors with parameters as defined in the configuration files are added to the navigational data. The AIS data is received by the target USV through dedicated ROS messages as discussed in Section 6.4. By printing the raw AIS data string to the Terminal, one can verify that the messages are valid. An example of an AIS string obtained from the simulation is quoted below:

```
!AIVDM,1,1,,A,10000008AePgwGLR0eVIR7`aP5`l,0*55
```

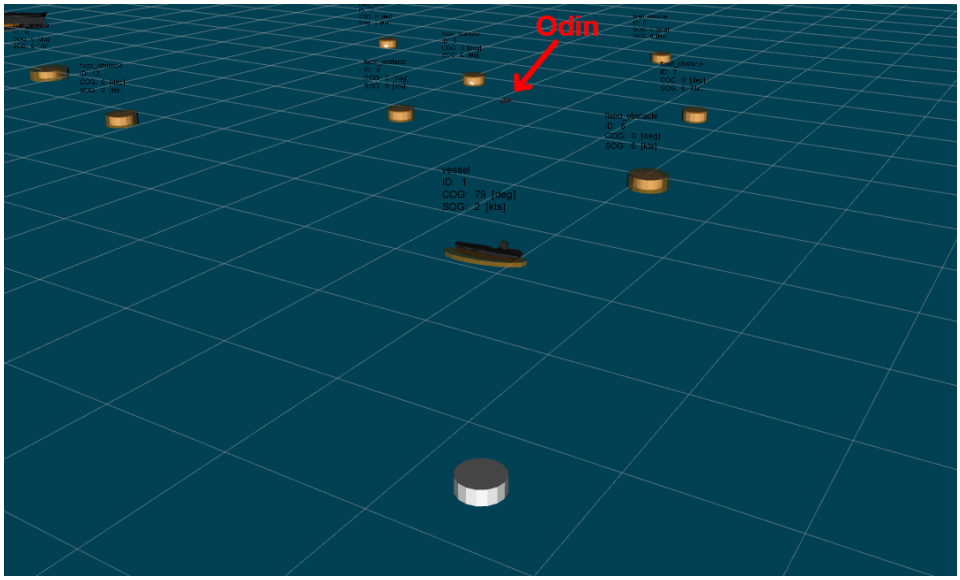

Figure 7.8: Illustration of the effect of radar shadow. When an object is located behind another object relative to the USV, the radar will not detect the first object. This is the case in this situation, where an iceberg is located behind a larger vessel. Hence, the iceberg is not detected by the TDM.

Using the online AIS decoder available on `https://rl.se/aivdm`, it is easily verified that the message is valid, containing a standard Class A position report as accounted for in Section 2.2.1. Table 7.1 summarizes the relevant data as obtained from the AIS message compared to the true values, obtained by printing the true data to the Terminal. The deviations from the true states correspond nicely with the error variances defined in the configuration files, indicating that the production of artificial noise works as intended. The Rate of Turn (ROT) obtained from the AIS message has an error of 48 $\frac{deg}{min}$, which is a significant deviation compared to what one would expect from a real AIS message. Recall, however, that the noise parameters were increased to demonstrate their effect. As seen in Listing B.3, the measurement variance of the ROT of AIS messages is set to 2000 $(\frac{deg}{min})^2$, and hence, a deviation of 48 $\frac{deg}{min}$ can be considered normal.

Table 7.1: Comparison of navigational data obtained from a simulated AIS message compared to their true associated values. The deviation in position corresponds to an error of approximately 3m.

Data Field	**From AIS data**	**True Value**	**Unit**
Latitude	59.438762	59.438792	[deg]
Longitude	10.483597	10.483607	[deg]
Speed Over Ground	10.9	10	[knots]
Course Over Ground	244	243	[deg]
Rate of Turn	48	0	[deg/min]

7.6 True HIL Simulation of Motion Control

Although the control system of Odin currently is unable to receive data from the sensors considered in this thesis, the control system was able to perform waypoint guidance in a HIL setup against the Dynamics simulator alone. The trial is accounted for in Ødegaard (2017), and provided promising results in regards to the model of Odin used in the Dynamics simulator. The results also show that the ROS interface between the simulator and Odin works as expected.

As the Surroundings simulator is based upon the same ROS interface, it is expected that the control system of Odin will be able to receive data from the simulated TDM and AIS when the time is ready. The format of the ROS messages for these purposes is, however, not yet decided. It is therefore likely that the format of the messages from the simulated TDM and AIS must be configured in accordance with the decided protocol.

When the control system of Odin is ready to receive data from the TDM and AIS as planned, it is hence expected that the Surroundings simulator in combination with the Dynamics simulator will provide a complete and functional HIL setup.

Chapter 8

Conclusion and Further Work

8.1 Conclusion

A standalone Surroundings simulator, simulating the surroundings of an unmanned surface vehicle (USV), was developed. The simulator was built upon the Robotic Operating System (ROS), using ROS topics and messages to publish sensor data. Modularity was preserved, decoupling the Surroundings simulator from the Dynamics simulator accounted for in Ødegaard (2017). The two simulators, although decoupled, together constitute the first functional prototype of a Hardware-In-the-Loop (HIL) Simulator for the USVs Odin and Jolner developed by Kongsberg Maritime in cooperation with Norwegian Defense Research Establishment (FFI). The simulator is customized for Odin, with the Dynamics simulator providing the possibility of changing the simulated dynamics to a model that better fits Jolner.

The Surroundings simulator manages obstacles and other vessels in the surrounding environment of the USV. Information about these objects are used to generate artificial data and measurements from an Automatic Identification System (AIS) transceiver, a radar and a Light Detection and Ranging (LiDAR) sensor. The radar and LiDAR measurements are not published to the control system of the USV directly, but are combined with the AIS data to generate detected object ROS messages from a fictitious target detection module (TDM). This is as requested by Kongsberg Maritime, as it is how the sensors are planned to be used on Odin. The measurements and data were combined

using the Simple Fusion algorithm, which calculates an optimal weighted average of the Kalman-filtered data from each sensor system. The algorithm was demonstrated by an example in MATLAB with good results. Using this algorithm in the simulated TDM was considered a conservative approach, as the algorithm of the real TDM is expected to perform even better.

The data from the AIS transceiver is planned to be used by the control system of Odin directly, and hence, simulated data from an AIS transceiver collecting AIS messages from nearby traffic is published on a dedicated ROS topic.

Configuration files launched with ROS initialize the simulations, which enables users with no knowledge of the underlying code to easily launch customized simulation scenarios. Editing the configuration files, the user can customize the range and sensor noise of the AIS, radar and LiDAR and define obstacles and other vessels to include in the simulation.

A graphical user interface (GUI) was developed, where the main feature is the ability to visualize the simulations in 3D with ROS RViz. The GUI also provides real-time plots of the heading and speed of the USV, a 2D map of the simulated area and functionality to spawn new obstacles throughout the simulation.

The control system of Odin is still under development, and is currently not able to receive any data from the TDM or AIS. Hence, the Surroundings simulator can not be used in a HIL setup for testing of collision avoidance at the current time. The control system is, however, able to receive and utilize data from the sensors used for motion control. The Dynamics simulator interacted successfully with Odin's control system in a HIL simulation of a waypoint guidance mission in Ødegaard (2017). This indicates that the ROS interface between the HIL Simulator and the control system of Odin is working as intended. It is hence likely that the Surroundings simulator also will interact correctly with the control system when the time comes, however with some modifications needed to the message formats and topics used for transmission of simulated AIS and TDM data.

The functionality of the Surroundings simulator was demonstrated by an example, where an open sea scenario with icebergs and other vessels was simulated. A model of Odin navigated in the simulation using predefined thrust inputs logged from a real mission at sea. The simulation was visualized using the GUI. The output of the simulated

TDM could be inspected in RViz, where the estimated position, size and orientation of the detected objects were visualized in 3D. By analyzing simulated AIS messages and the behavior of the detected objects, it was verified that the functionality of these sensor systems was simulated in accordance with the specifications and requirements of Kongsberg Maritime.

8.2 Further Work

For further work, it is recommended to consider the following aspects:

- When the control system of Odin is ready to receive data from the TDM and AIS, the format of the ROS messages and topics used for transmission of data from these sensor systems must be adapted to the decided protocol. The formats of these messages are currently not known, but a qualified guess was used as a temporary solution.
- Representation of coast lines and land to be included in the simulations.
- Rendering of raw data from radar and LiDAR based on the simulated surroundings of the USV.
- Including agents with varying degree of intelligent behavior in the simulated environment, such as vessels and boats that can react to their surroundings and perform path planning based on COLREGs.
- Using data from genuine ship traffic, e.g. obtained from historic AIS data available online, to simulate traffic in the simulations.
- Implementation of failure scenarios.

Appendices

Appendix A

Installation of the HIL Simulator

The following steps should be performed in order to get the simulator running for the first time:

1. Install Ubuntu on the computer intended for simulation. (`https://www.ubuntu.com/download/desktop/install-ubuntu-desktop`).
2. Install the latest distribution of ROS for Ubuntu (`http://wiki.ros.org/kinetic/Installation/Ubuntu`).
3. Open the terminal and create a ROS workspace:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
```

4. Clone the HIL-simulator repository in to the src-folder:

```
$ git clone https://github.com/kjetilbl/hilsim
```

5. Build the `simulator_messages` package first, as several simulator packages depend on this package:

```
$ cd .. && catkin_make --pkg simulator_messages
```

6. Build the workspace:

```
$ catkin_make
```

7. Source your new setup.*sh file:

```
$ source devel/setup.bash
```

8. Obtain the 3D models of Odin and/or Jolner. Copy these into `/home` in the computer used for simulation.

9. Set the desired parameters of the simulation in the configuration files located in `src/environment/config/`. The default configuration files already included in the folder configures an open sea scenario with presence of icebergs and ships.

10. The simulator should now be ready for use. Test it by launching the open sea scenario of Odin navigating using predefined actuator inputs:

```
$ roslaunch environment busy_open_sea_scenario.launch
```

Appendix B

Configuration Files of the Busy Open Sea Scenario

This appendix includes the configuration parameters of the open sea scenario with presence of icebergs and two vessels, simulated in Chapter 7. The parameters are defined in `.yaml` configuration files located in `src/environment/config/` in the default HIL Simulator package available from `https://github.com/kjetilbl/hilsim`.

B.1 Sensor Parameters of the Dynamics Simulator

```
wind_speed: 10
wind_direction: 90
current_speed: 0.5
current_direction: 270
wave_height: 2
wave_direction: 90

dt: 0.05
gps_frequency: 20
mru_frequency: 100
imu_frequency: 100
speed_sensor_frequency: 20
wind_sensor_frequency: 50
start_latitude: 59.4377744115633
start_longitude: 10.4716511800410
```

Listing B.1: Parameters of the external forces and sensors regarding the dynamics of the target USV, in this case Odin, during the open sea scenario.

B.2 Obstacles and Vessels

```
fixed_obstacles : {
  "Iceberg_1" : "RADIUS=15 COORD=10,470952/59,436573",
  "Iceberg_2" : "RADIUS=15 COORD=10,469416/59,437963",
  "Iceberg_3" : "RADIUS=15 COORD=10,468931/59,439197",
  "Iceberg_4" : "RADIUS=15 COORD=10,474656/59,439714",
  "Iceberg_5" : "RADIUS=15 COORD=10,477303/59,439172",
  "Iceberg_6" : "RADIUS=15 COORD=10,474814/59,437146",
  "Iceberg_7" : "RADIUS=15 COORD=10,480263/59,441991",
  "Iceberg_8" : "RADIUS=15 COORD=10,474666/59,443562",
  "Iceberg_9" : "RADIUS=15 COORD=10,467777/59,441583",
  "Iceberg_10" : "RADIUS=15 COORD=10,483189/59,436865",
  "Iceberg_11" : "RADIUS=15 COORD=10,481345/59,434604",
  "Iceberg_12" : "RADIUS=15 COORD=10,465075/59,433254",
  "Iceberg_13" : "RADIUS=15 COORD=10,472437/59,437849",
  "Iceberg_14" : "RADIUS=15 COORD=10,472209/59,438807" }

ships : {
"support_vessel" : "LENGTH=30 SpeedInKnots=1 WP=10,468943/59,440308 WP
    =10,474462/59,440736 WP=10,480279/59,439919 WP=10,481580/59,437513",
"research_ship" : "LENGTH=100 SpeedInKnots=10 WP=10,483932/59,438875 WP
    =10,460502/59,432853" }
```

Listing B.2: Configuration of simulation objects in the open sea scenario with icebergs and other vessels.

B.3 Sensor Parameters of the Surroundings Simulator

```
AIS_range_m: 10000
radar_range_m: 1852
lidar_range_m: 100
TDM_update_rate_Hz: 1
error_pr_distance_gain: 0.01

## AIS error parameters
AIS_position_measure_variance_m2: 5
AIS_COG_measure_variance_deg2: 1
AIS_track_measure_variance_deg2: 1
AIS_ROT_measure_variance_deg2_pr_min2: 2000
AIS_SOG_measure_variance_knots2: 0.5
AIS_position_bias_variance_m2: 0.5
AIS_COG_bias_variance_deg2: 0.5
AIS_track_bias_variance_deg2: 0.5
AIS_ROT_bias_variance_deg2_pr_min2: 500
AIS_SOG_bias_variance_knots2: 0.1
AIS_position_bias_time_constant_sec: 20
AIS_COG_bias_time_constant_sec: 10
AIS_track_bias_time_constant_sec: 10
AIS_ROT_bias_time_constant_sec: 10
AIS_SOG_bias_time_constant_sec: 10

## Radar error parameters
radar_position_measure_variance_m2: 10
radar_radius_measure_variance_m2: 20
radar_position_bias_variance_m2: 0.5
radar_radius_bias_variance_m2: 3
radar_position_bias_time_constant_sec: 10
radar_radius_bias_time_constant_sec: 5

## LiDAR error parameters
lidar_position_measure_variance_m2: 0.1
lidar_radius_measure_variance_m2: 0.1
lidar_position_bias_variance_m2: 0.1
lidar_radius_bias_variance_m2: 0.1
lidar_position_bias_time_constant_sec: 0.1
```

```
38 lidar_radius_bias_time_constant_sec: 1
```

Listing B.3: Noise parameters of AIS, radar and LiDAR in a simulation scenario with icebergs and other vessels.

Bibliography

Ben-Kiki, O., Evans, C., and döt Net, I. (2009). YAML Ain't Markup Language (YAML™) Version 1.2. *http://yaml.org/spec/1.2/spec.pdf*, Accessed: 23.06.2017.

Børs-Lind, K. S. (2017). *Specification of Simulated Environment and User Interface for HIL Testing of USV*. Specialization Project Report, Norwegian University of Science and Technology, Trondheim, Norway.

Brown, R. G. and Hwang, P. Y. C. (1997). *Introduction to Random Signals and Applied Kalman Filtering*. John Wiley & Sons, Ltd, 3rd edition.

E. S. Raymond (2016). AIVDM/AIVDO protocol decoding. *http://catb.org/gpsd/AIVDM.html*, Accessed: 14.05.2017.

Fossen, T. I. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd.

Free Software Foundation (2007). GNU Lesser General Public License. *https://www.gnu.org/licenses/lgpl.html*, Accessed: 25.05.2017.

Free Software Foundation (2016). Modified BSD license. *https://www.gnu.org/licenses/license-list.html#ModifiedBSD*, Accessed: 16.01.2017.

Kazimierski, W. (2013). Fusion of Data from AIS and Tracking Radar for the Needs of ECDIS. Technical report, Maritime University of Szczecin, Poland.

Ødegaard, E. (2017). Prototype of HIL Test Platform for Autonomous USV - Simulation of Vessel Dynamics. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway.

QCustomPlot (2017). QCustomPlot 2.0.0-beta Documentation. *http://www.qcustomplot.com/documentation/index.html*, Accessed: 25.05.2017.

ROS (2014). ROS Introduction. *http://wiki.ros.org/ROS/Introduction*.

SIMRAD (2012). Simrad 4G Brochure. *http://www.simrad-yachting.com/Root/Brochures/SimradYachting/English/Simrad%204G%20Brochure-AMER.pdf*, Accessed: 21.10.2016.

SiRF (2005). *NMEA Reference Manual*. SiRF Technologies, Inc.

Skjetne, R. and Egeland, O. (2005). Hardware-in-the-loop testing of marine control systems. Technical report, Marine Cybernetics, Trondheim, Norway.

The Qt Company Ltd. (2017). Qt Documentation. *http://doc.qt.io/qt-5/reference-overview.html*, Accessed: 25.05.2017.

U.S. Coast Guard Navigation Center (2016a). AIS Messages. *http://www.navcen.uscg.gov/?pageName=AISMessages*, Accessed: 15.04.2017.

U.S. Coast Guard Navigation Center (2016b). How AIS Works. *https://www.navcen.uscg.gov/?pageName=AISworks*, Accessed: 10.05.2017.

Velodyne (2017). Velodyne LiDAR HDL-32E Datasheet. *http://velodynelidar.com/docs/datasheet/97-0038_Rev%20K_%20HDL-32E_Datasheet_Web.pdf*, Accessed: 03.06.2017.