NTNU

Norwegian University of
Science and Technology

COLREGS compliant Collision Avoidance System for a Wave and Solar Powered USV

Sølve Dahlin Sæter

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Tor Arne Johansen, ITK
Co-supervisor: Giorgio Kwame Minde Kufoalor, ITK
Artur Piotr Zolich, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MSC THESIS DESCRIPTION

Name: Sølve Dahlin Sæter
Department: Engineering Cybernetics
Thesis title: COLREGS compliant collision avoidance system for a wave and solar powered USV

Thesis Description:

The objective of this thesis is to design and implement a navigation and collision avoidance system for a wave and solar powered unmanned surface vehicle (USV) using the LSTS software toolchain. The system shall adhere to the International Regulations for Avoiding Collisions at Sea (COLREGs).

The thesis can be divided into three parts:

1. Software
   a. Further develop an existing collision avoidance algorithm,
   b. Conduct a simulation study in LSTS that includes single and multiple obstacle avoidance.
   c. Implement an autopilot
2. Hardware.
   a. Implement hardware-sensor
   b. Integrate a navigation and collision avoidance subsystem.
   c. Implement a hardware test platform capable of single and dual level testing.
3. Commissioning.
   a. Perform a full scale testing of the USV and its subsystems.

Start date: 2018-01-08
Due date: 2018-06-18

Thesis performed at: Department of Engineering Cybernetics, NTNU
Supervisor: Thor Arne Johansen, ITK
Co-Supervisor: Giorgio D. Kwame Minde Kufoalor, ITK
Artur Piotr Zolich, ITK

Preface

This Master's thesis is written as a compulsory part of the Master's degree in Engineering Cybernetics at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). The thesis builds upon a preliminary project conducted during the autumn of 2017. The project report focused on getting familiar with the LSTS software toolchain (DUNE, Neptus, IMC) and the ongoing software integration of simulator environment for testing the collision avoidance system using LSTS. The parts of the thesis that has been based upon preliminary research from the project are section 2.1, 2.2, 3.1, Chapter 4 and most of Chapter 6.

NTNU has provided with a collision avoidance algorithm to be further developed and an USV. The USV was delivered March 21st, along with the training required to handle the USV. This training was lead by engineers from AutoNaut Ltd, which has also been providing USV related information via email prior and after delivery, and in person during the training. The hardware sensors and equipment necessary to interface and implement this thesis' solution to the USV were made available early May. NTNU has also provided with the software, LSTS software toolchain, and contact information to experienced LSTS developers located at NTNU and at Porto University.

I would like to express my gratitude towards some of the people that made this thesis possible. First of all, my supervisor Tor Arne Johansen, and especially my co-supervisors Giorgio D. Kwame Minde Kufoalor for guidance regarding collision

avoidance simulations and Artur Piotr Zolich for assisting in hardware implementation and making the field testing possible. I would also like to thank Peter Bailey Knutsen for assembling and wiring up the subsystems. Without their effort the sea trials would not have been a reality.

Sølve Dahlin Sæter

Trondheim, 18-06-2018

Abstract

Autonomous ships are the future of modern maritime traffic, but in order to operate in the vicinity of other vessels, Autonomous Surface Vehicles (ASV) must be equipped with a collision avoidance system (CAS). This system must adhere to the rules-of-the-road, the International Regulations for Preventing Collisions at Sea (COLREGS).

In this thesis a navigation and collision avoidance system was implemented to a wave- and solar driven Unmanned Surface Vehicle (USV), namely AutoNaut. The COLREGS compliant collision avoidance system has been further developed and its performance has been validated through various simulations in the LSTS software toolchain. The main scenarios designed to test the CAS and the COLREGS compliance were: head-on, crossing and overtaking. From these situations single and multiple obstacle avoidance scenarios were created, to see how the system behaves in a complex situation. A hardware design was made where the CAS combined with hardware-sensor implementation formed the navigation and collision avoidance level of AutoNaut.

The CAS performed quite well in the simulated scenarios, and the avoidance maneuvers complies with the COLREGs requirements. However, more tuning is required and environmental forces needs to be taken into account for a more realistic and optimal simulation.

Full-scale testing of the USV and the systems were conducted, and the autopilot and

the subsystem interactions were tested. The USV was able to operate, but the autopilot needs more tuning. Several attempts were made to test the collision avoidance, however they were not completed due to hardware limitations and the difficulties of obtaining obstacle vessels.

Sammendrag

Selvstyrende skip er fremtiden for den moderne sjøfarten. For at disse skal kunne operere i nærheten av andre fartøyer, må selvstyrte båter være utstyrt med et kollisjonsunngåelsessystem (CAS). Dette systemet må overholde det internasjonale regelverket til forebygging av sammenstøt på sjøen (COLREG). I denne avhandlingen blir et navigasjons- og kollisionsunngåelsessystem implementert for det bølge- og soldrevede ubemannede overflatefartøyet AutoNaut. Det kollisjonsforebyggende systemet, kompatibelt med sjøveisreglene, blir videreutviklet og ytelsen er validert gjennom ulike simuleringer i LSTS-programvareverktøyet. Hovedscenariene designet for å teste overholdelse COLREGS er: front-mot-front, kryssing og forbikjøring. Fra disse situasjonene ble det opprettet både enkle og komplekse hindringsscenarier.

Det ble designet en hardware system der CAS ble kombinert med hardwaresensorer. Dette dannet grunnlaget for AutoNauts evne til navigasjon og kollisjonsunngåelse. Kollisjonsunngåelsessystemet fungerte ganske bra i de simulerte scenariene, og manøvreringen overholdt COLREG. Det er behov for mer testing og for å oppnå en mer realistisk og optimal simulering må krefter som bølger, strøm og vind tas hensyn til.

Fullskala testing av båten og systemene ble gjennomført inkludert testing av autopiloten og interaksjonene mellom de ulike delsystemene. Båten klarte å navigere til ønsket posisjon, men autopiloten trenger mer tuning. Det ble utført flere forsøk på å teste kollisjonsunngåelses, men disse ble ikke fullført på grunn av maskinvarebegrensninger og tilgang til hindringsfartøyer.

Contents

List of Figures

List of Tables

Acronyms

ADCP Acoustic Doppler Current Profiler

AIS Automatic Identification System

ASV Autonomous Surface Vehicle

BBB BeagleBone Black

CAS Collision Avoidance System

COLREGS International Regulations for Preventing Collisions at Sea

DOF Degrees of Freedom

GPS Global Positioning System

IMC Inter-Module Communication

IMO International Maritime Organization.

LSTS Under-water System and Technology Laboratory

MAS Maritime Autonomous Systems

MRA Mission Review & Analysis

SB-MPC Simulation-Based Model Predictive Control

USV Unmanned Surface Vehicle

Chapter 1

Introduction

1.1 Motivation

The interest and development of autonomous technology has advanced unexpectedly rapid, and autonomous vessels are today a very trending topic. This trend of automation has also been evident in the automotive and maritime industry, industries traditionally performed entirely by human endeavour, and the growth of this autonomous system industry is opening up a world of new opportunities[12].

The demand for autonomous systems such as ASVs, AUVs and UAVs is only expected to grow in the years to come, both in the commercial and military sector, with vast possibilities such as autonomous shipping, science and research, inspection and repairing infrastructure in an offshore environment, etc. [36].

In 2016, the Trondheimsfjord became the world's first official test bed designated for the testing of autonomous vessels, and Trondheim could also soon become the first city to have its own fully autonomous electric passenger ferry. The Norwegian University of Science and Technology (NTNU) is currently working on the autonomous transportation system, which is scheduled to be running between Ravnkloa and Vestre Kanalkai in 2018/2019. The proposed driverless ferry would have the capacity of transporting 12 persons, and crossing the channel will only take a mere minute[29, 21].

(a) Full Scale Ferry [8]

(b) YARA Birkeland [3]

Figure 1.1: Two autonomous vessels.

Additionally to the autonomous trend, there has been an increased focus on creating a greener environment and reducing the environmental footprint of ships, thus a focus on vessels powered by renewable energy, or so called zero emission vessels. As a consequence of the Paris Agreement from 2015, which aims to respond to the global climate change threat by dealing with greenhouse gas emissions, and mitigating and adapting to its effects. International shipping industry leaders gathered in Bonn at the 23rd session of the Conference of Parties (COP 23), with the ambition of making the maritime industry explore and take action on the road to low emission shipping and decarbonization of the industry. High ambition approaches and conclusions for the decarbonization Action Plan were made, such as commitments for applying low carbon and zero emissions technologies to the shipping industry, e.g. to container ships. Today, there are already several major maritime projects that are applying similar technologies, one of them is YARA Birkeland[14, 9].

A collaboration between Kongsberg Gruppen ASA, a Norwegian maritime technology firm, and YARA ASA, a fertilizer manufacturer, will eventually result in YARA Birkeland. YARA Birkeland is to become the world's first fully electric and autonomous ballast free container vessel, with zero emissions. Scheduled to perform fully autonomous operations from 2020, YARA Birkeland will become a game changer for the global maritime transport and the environment, as it contributes to meet the UN

sustainability goals by being zero emission. , It also allows for a seaborne container transportation rather than a land based transportation, whereas 40.000 road trips from Yara's Porsgrunn fertiliser plant to the ports of Brevik and Larvik will be avoided, consequently reducing the level of GHG emitted by haulage trucks[33, 3].

Another vessel with zero emissions is the unmanned surface vehicle (USV) called AutoNaut, developed by MOST (Autonomous Vessel) Ltd. AutoNaut is solely powered by wave- and solarpower, in other words, AutoNaut is self-powered, producing zero emissions and propels itself forward by utilizing the wave energy. With waves always present at sea, AutoNaut is constantly surrounded with easy accessible energy to drive itself forward, thus, making AutoNaut capable of executing long endurance missions. The possibilities and uses are many. [4].

Autonomous ships are the future of modern marine traffic and shipping, which will result in an increase of autonomous vessels of different sizes operating in vicinity of each other. In order to maneuver safely around in its environment, a reliable autonomous collision avoidance system is a necessary prerequisite for an autonomous vessel. To safely operate with other vessels present the collision avoidance systems (CAS) need to adhere to the rules-of-the-road, namely the *International Regulations for Avoiding Collisions at Sea* (COLREGS). This is essentially what this project is focusing on, the integration of a COLREGS compliant collision avoidance system to a software toolchain which is to be implemented and running onboard AutoNaut.

1.2 Assumptions

The following assumptions have been made:

- Every vessel encountered are motor-driven, i.e. situations involving sail-powered vessels have not been considered.
- The necessary information about own vessel is always available.
- The necessary information about other vessels is always available, through different sensors e.g. GPS and AIS.
- There are no environmental forces, such as wind, waves and current present during the simulations.

1.3 Outline

Chapter 2 provides an introduction of the USV and an overview of system design and components of the specific USV related to this thesis. Chapter 3 provides necessary theoretical background on the LSTS software toolchain applied and the COLREGS. The system architecture and theory regarding the design of the collision avoidance are given in Chapter 4.

Part 1 relates to the software development and is composed of Chapter 5, 6 and 7. Chapter 5 enlightens the major changes made to the collision avoidance algorithm, while Chapter 6 explains the design and implementation approach for integrating the collision avoidance. In Chapter 7 the result from the LSTS simulations are presented.

Part 2 constitutes of Chapter 8 and 9, which go in detail on the hardware integration and testing. The results from the full scale testing follows in Chapter 10.

Finally, a conclusion is given in Chapter 11 along with suggestions for future work in Chapter 12. The Appendix contains a Gantt diagram describing the progression of the project.

Chapter 2

AutoNaut

AutoNaut is a self-powered unmanned surface vehicle (USV) developed by MOST (Autonomous Vessel) Ltd. The USV utilizes the wave energy to propel itself forward and is designed to produce zero emissions. AutoNaut is based on experiments stretching back to 1895 when Hermann Linden of the Naples Zoological Station built the original AutoNaut[4, 6]. Linden‘s AutoNaut was equipped with flexible foils mounted at the bow and at the stern, and was reported to have reached speed of about 3-4 mph.

Figure 2.1: AutoNaut [4]

Solely powered by wave- and solar power, Autonaut is capable of operating autonomously in oceanic environments for several months continuously. To generate a forward thrust AutoNaut's patented Wave Foil Technology, consisting of four keel mounted foils, utilizes the energy from the pitch and roll motions of the hull in waves.[19].

There are three photovoltaic (PV) panels on the deck, as illustrated in Figure 2.2. These PV panels generate power to charge the onboard batteries which again provide power for the navigation, communication and payload systems. To maintain speed during flat sea conditions power can be diverted to an auxiliary electric thruster. Ensuring a 3 knots steady propulsion, the auxiliary propulsion unit can also be used for recovery from slipways, remote control or add to the wave propulsion for a certain time[20].

AutoNaut's hull is designed to maximize wave energy, in contrast to conventional hulls that generally are designed to minimize pitching and rolling. When providing plenty of movement in both directions the foils provide thrust and stabilization, while also ensuring self-adjustment in event of a capsize[35].

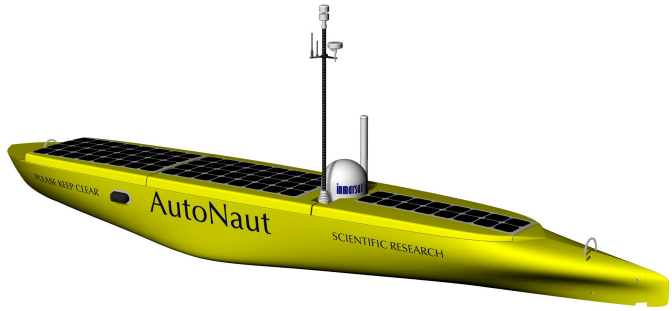


Figure 2.2: 3D-model of AutoNaut[4]

AutoNaut can be deployed directly from a slipway by two personnel before maneuvering to its operating area. Pre-programmed or newly sent mission plans can be executed with command and control oversight from a remote operator who keeps a

continuous watch. During missions, valuable data and navigational updates could be stored on-board AutoNaut or sent back to the operator for processing. With the vast payload capacity and long-term endurance performance AutoNaut supports a high variety of applications[19]. Some of the valuable data AutoNaut possibly could collect:

- Surveillance and detection of vessels, shoals of fish or marine mammals.
- Seawater parameters: Conductivity, dissolved oxygen, pH, temperature, etc.
- Oceanographic data: Wind speed and direction, bathymetry, barometric pressure, local weather conditions, etc.

Such collected data could prove to very valuable for many research areas, e.g. to identify areas which naturally attract high fish numbers or [20].

2.1 Wave Propulsion Technology

Since it is designed to perform long endurance autonomous missions, Autonaut will need to achieve a good all-around performance in all directions relative to the wind and waves. With waves almost always present at sea, AutoNaut's wave propulsion system and special designed hull takes advantage of waves from any direction to generate thrust and propel the USV forward. The forward propulsion of Autonaut is achieved by applying the same technique patented and developed by Linden and Einar Jacobsen [4, 35]. Similarly, Autonaut is equipped with spring-loaded foils attached to the struts under the keel. These foils exploit the wave-induced vessel motion, caused by waves lifting the vessel up, out of the water and dropping it down again, to generate the forward propulsion.

The foils pivots about a point located in front of the center of pressure, so that the foil's angle of attack is reduced by the incoming flow. The foil pitch is, however, constrained by the spring which ensures that the angle of attack is not reduced to zero, which would have resulted in zero forward thrust. If adjusted properly the angle of attack would ensure that the foil generates thrust irrespective of the direction of the relative flow due to the waves[6].

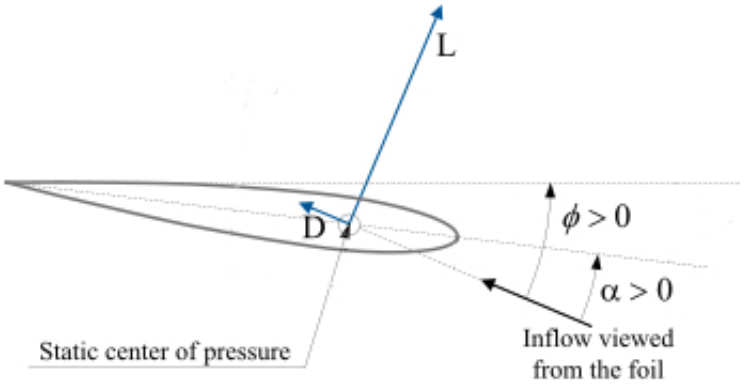


Figure 2.3: Foil forces and angles. Lift is denoted with L while the drag is D. [6]

When the foils moves through the water, they are forced to pitch oscillate due to the incoming waves, and the resulting vessel motion. Consequently, a hydrodynamic lift force, D, acts on the foil. When this lift force resolved in a forward direction, combines they overcome the foil drag and hull resistance, thus, propelling the vessel forward. Assuming there were no foils attached, the ship could end up drifting backwards[7].

2.2 AutoNauts Capability

The ocean is never still. Waves are almost always present and the wave propulsion system would ensure that the AutoNaut doesn't stop, with almost no limits on range or endurance. However, it has its limitations. Flat calm conditions pose a real issue for the wave propulsion system as the wave foils would not be generating much thrust. Still, even on an apparently flat day, will AutoNaut be able travel with a speed around half a knot since undulations are nearly always evident at sea.[35] Additionally to flat calm conditions, the wavelength of the waves affect the performance of AutoNaut's wave foil technology.

The wavelength that have been most effective to drive the hull forward is about one and two times the length of the hull. Which coincides to tests performed by [27]

stating that wave energy is utilized efficiently when the ship length is smaller than the wavelength. Autonaut does not respond well to wave periods that are too long or too short for the required pitch and roll motion. Under such circumstances or to avoid any obstacles, AutoNaut could consider to use the electric propulsion system.

Figure 2.4: A 5m AutoNaut at sea, propelling forward by pitch motions [4]

The wavelength that have been most effective to drive the hull forward is about one and two times the length of the hull. Which coincides to tests performed by [27] stating that wave energy is utilized efficiently when the ship length is smaller than the wavelength. Autonaut does not respond well to wave periods that are too long or too short for the required pitch and roll motion. Under such circumstances or to avoid any obstacles, AutoNaut could consider to use the electric propulsion system.

Table 2.1, provides an indication on how a 5m AutoNaut perform under different wind and wave conditions and its respective generated speed. These measurements are obtained after a certain wind speed has been achieved for more than 1 hour. The hull speed would vary continuously as each wave's size and shape develops different amounts of thrust. As each wave's size and shape develops different amounts of thrust the hull speed would vary continuously. Thus, the more regular the wave series, the

higher the average speed will be[4].

Sea State	Wind [kts]	Average Hull Speed up/down wind [kts]	Average Hull Speed across wind [kts]
0 - 1	0 - 3	Up to 0.5	Up to 0.25
2 – 3	4 - 10	0.5 - 1.5	0.25 – 1
4 – 5	11 - 20	1.0 – 2.0	0.75 – 1.5
6+	>21	1.5 - 3	1.25 – 2.0

Table 2.1: AutoNaut's performance under different conditions. (Courtesy of AutoNaut Ltd

Factor such as the wind fetch, speed and direction of the wind and waves and its developed wave profile, they all affect the performance of AutoNaut when operating at sea. During ideal conditions, the highest speed achieved of a 5m AutoNaut is currently 4 knots.

2.3 NTNU AutoNaut's Overview

In this section the AutoNaut related to this report will be introduced, including the main components composing the NTNU AutoNaut. Finally, the specific sensors used in the implementation of Level2 will be presented.

2.3.1 NTNU AutoNaut

NTNU AutoNaut is composed of three different Levels, see Figure 2.5, each Level with its own responsibility and functionality. These Levels assembled and wired by Peter Bailey Knutsen, are each fitted into a waterproof IP67 case. Level2 and Level1 are connected through waterproof connectors using the RS-232 protocol, while Level 3 communicate through ethernet.

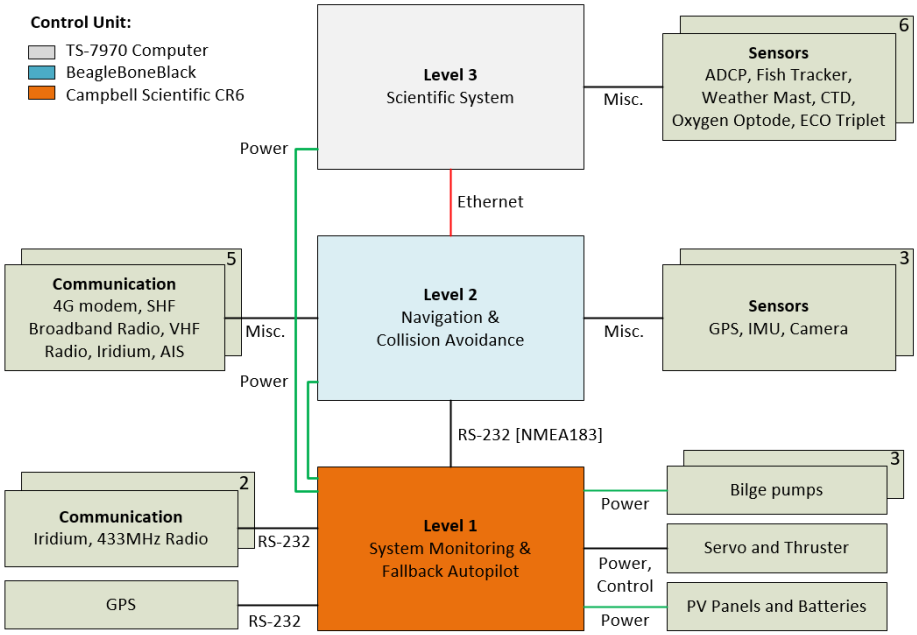


Figure 2.5: Main Levels composing the NTNU AutoNaut system (Courtesy of [2])

Level1: System Monitoring & Fallback Autopilot

Level1 is responsible for the underlying control of AutoNaut. It monitors the system ensuring the state of the vessel. Level1 request navigational updates from Level2 and applies the acquired rudder and thruster commands to the rudder and thruster servo. In the case of failure of Level2 and the navigation, Level1 will enter fallback mode and take control, using its own fallback autopilot. Level1 has being designed and developed by Bendik Agdal, and is presented in the Master Thesis: "Design of Control System for Green USV" published 2018 at NTNU.

Level2: Navigation & Collision Avoidance

Level2 is responsible for the navigation, collision avoidance and communication between AutoNaut and other vessels, land and operator. Level2 is composed of a Bea-

gleBone Black that should run the required software, GLUED and DUNE, to handle the communication, collision avoidance software and interfacing with the necessary sensors. The development of Level2 is the main focus of the report.

Level 3: Scientific System
Level 3 should collect data from scientic sensors, and perform data processing on the collected data from sensors such as the ADCP, CTD, Oxygen Optode, etc. Level 3 has not been the focus for the NTNU AutoNaut project 2017/2018.

2.3.2 Level2 Components

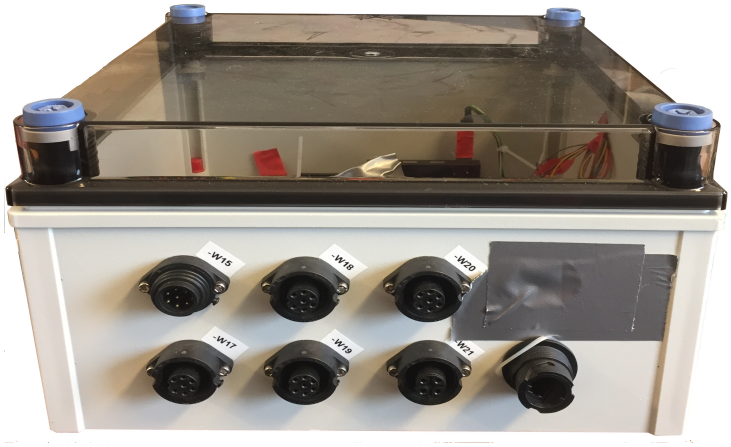


Figure 2.6: Level2 IP67 case

IP67 cases
AutoNaut consists of three subsystems/levels. Each subsystem is enclosed inside a waterproof protective case, namely a IP67 case, meanwhile enabling a variety of different sensors or the other subsystems to connect and interface with one another. Figure 2.6 depicts how a IP67 case looks. There are currently 6 labeled plugs, ranging from W15 to W21, whereas only W15-W17 is used at this stage of the AutoNaut project.

There are also an Ethernet port (bottom right) for connecting to the Ubiquiti Rocket M2, and a final "plug" which has not been fitted yet. This was temporarily sealed during the sea trials as a precaution.
The relevant connections of the Level2 IP67 case:

W15: Power-and-Serial-Communication
W16: Garmin GPS 18x-5Hz
W17: Raymarine AIS650
Ethernet: Ubiquiti Rocket M2

BeagleBone Black

BeagleBone Black (BBB), see Figure 2.7, is the single-board computer used in Level2 on AutoNaut. With a 1GHz ARM-based CPU and two rows of GPIO (general purpose Input/Output) pins mounted along both sides of the board, BBB is well suited to be used as an embedded system. The two 46 pin headers allows a wide range sensors and other hardware components to connect these pins allowing them to communicate with the BBB [5].

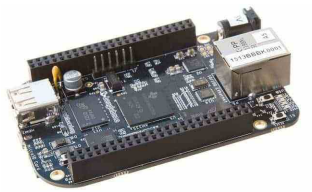


Figure 2.7: BeagleBone Black

Figure 2.8: Raymarine AIS650

Raymarine AIS650

Automatic Identification System (AIS), see Figure 2.8, is used by ships to automatically identify and track vessels in the surrounding area, and transmit their own identity, position, course, speed, and destination to other vessels or to coast stations. The received information could essentially be used as collision avoidance data. AutoNaut is equipped with a Raymarine‘s AIS650. It is a Class B AIS transceiver unit that uses

digital radio signals to exchange, both transmit and receive, information with other vessels equipped with either a Class A or a Class B AIS transceivers[25]. While the VHF antenna supplied with the AIS unit is used for the data exchange, the GPS receiver provides GPS data only intended to the AIS unit. The GPS output status of the AIS650 can be checked by connecting it to the ProAIS2 software.

The information contained in an AIS-messsage can be divided into two categories:

1. Dynamic Information
    - Transmitted every 2s to 3 minutes, depending on the vessel's speed, course change and AIS Class, while every 6 minutes when anchored.
    - Data content: MMSI, Speed over Ground, Position Coordinates, Heading.
2. Static Information
    - Transmitted every 6 minutes regardless of the vessel's movement status.
    - Data content: MMSI, Ship's Name, Cargo type, Dimensions, Destination.

Garmin GPS 18x-5Hz

The Garmin GPS 18x-5Hz LVC, see Figure 2.9,includes an embedded receiver and an antenna. The GPS 18x receiver continuously tracks multiple satellites while computing accurate positioning and velocity estimates. These precise navigation update occur once per second. The GPS 18x LVC is easy to integrate and use, as it interfaces to a serial port using the standard NMEA 0183 sentences. A minimum requirement of a system is to provide the GPS with a source of power and a clear view of the GPS satellites.

Ubiquiti Rocket M2

Ubiquiti Rocket M2, see Figure 2.10, is a base station, which features a long range and a breakthrough speed of up to 150+ Mbps real TCP/IP. It can be deployed in Point-to-Point (PtP) bridging or Point-to-MultiPoint (PtMP, with 2.4 GHz frequency band to support the specific application[28].

Figure 2.9: Garmin GPS 18x LVC

Figure 2.10: Ubiquiti Rocket M2

Chapter 3

Theoretical background

3.1 The LSTS Toolchain

The software used in this project is an open-source software toolchain developed by the Underwater System and Technology Laboratory (LSTS). The toolchain supports networked vehicles systems constituted by human operators, heterogeneous autonomous vehicles and sensors. The software toolchain is primarily composed of the side-shore control software Neptus, the on-board software Dune and IMC, a shared communication protocol, see Figure 3.1. Together they support deployment of multiple air and ocean vehicles, allowing simultaneous control of the vehicles and communication between the dynamic nodes in the network[30]

Neptus is the command and control software providing the human operators with a coherent visual interface for mission planning, control and data review and analysis. Dune is the onboard-software running on the vehicles. It is responsible for the communications, navigation, control, maneuvering, plan execution, vehicle supervision, along with the interactions with sensors, actuators and payloads. The main communication interface in Neptus and Dune is the Inter-Module Communication (IMC) protocol. IMC is a message-oriented protocol, designed and implemented for communication among heterogeneous vehicles, sensors and human operators. It enables a cooperative

exchange of real-time information throughout the system, regarding the environment and updated objectives.[24]

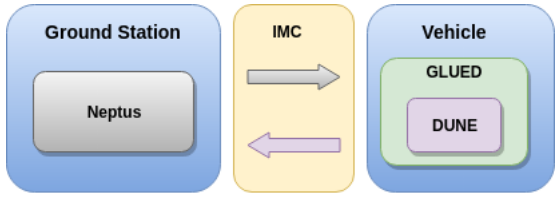


Figure 3.1: The LSTS Toolchain (Image based on [32])

In order to cope with all the different nodes in the networked vehicle system, a layered approach and common interfaces for communication and coordination between the components have been established. The control architecture of the LSTS software toolchain is illustrated in Figure 3.2, where each layer encapsulates lower-level details and provides interfaces for retrieving state and accepting commands[30].

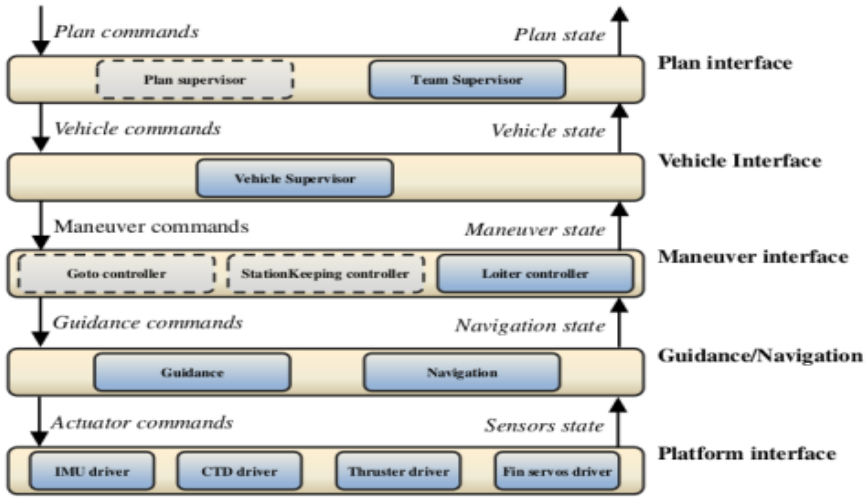


Figure 3.2: LSTS Control Hierarchy (Image from LSTS)[30]

All vehicles provide a platform with sensor and actuator interfaces, which are used by the higher-level interfaces such as the guidance and navigation software components. Based on the sensor data the Guidance/Navigation level provides a command set for controlling the desired vehicle behaviour. The current vehicle state is passed on to maneuver controllers that produce guidance commands. A vehicle supervisor continually verifies that the system is working properly. Depending on the state of the system it instantiates the maneuver controllers according to the requested maneuver specifications or terminates in the event of hardware failure or any safety violations. The vehicle supervisor receives maneuver specifications commands from upper layers. These maneuver commands originate from either a team supervisor or an on-board plan supervisor that, according to a plan specifications, triggers the execution of maneuvers based on the vehicle's state. The plan specifications can be created by human operators through an operator console and used by the plan supervisors to generate maneuvers that are necessary in order to complete the plan objectives. Neptus comprises the Plan interface layer, while the layers Vehicle, Maneuver, Guidance/Navigation and Platform interface, illustrated in Figure 3.2, are implemented using the DUNE framework.

3.1.1 DUNE

DUNE (DUNE Uniform Navigation Environment) is the on-board software running on the vehicle. It is responsible for communications, navigation, control, maneuvering, plan execution and vehicle supervision, and also every interaction with sensors, actuators and payload.

In DUNE, independent tasks containing logical operations run in a separate thread of execution. All these active tasks communicate with one another by publishing and subscribing to messages located on a message bus, visualized in Figure 3.3. For instance, a task may publish information containing details about a certain sensor onto the message bus. This IMC message may later on be consumed by other tasks that needs this certain information to fulfill its purpose. This leads to a high modularity, where new tasks, sensors or controllers easily can be added, and similarly old tasks can be enabled and disabled freely.

Depending on their purpose, DUNE tasks are divided into different groups[32]:

- Sensors: Drivers associated with some hardware that measures the environment.
- Actuators: Device drivers that allow for vehicle movement control and environment interaction.
- Estimators: Tasks that aggregate sensor information to make state estimations, e.g. the Navigation task.
- Controllers: Tasks that transforms high-level commands into low-level commands or actuation according to the current vehicle state.
- Monitors: Tasks that may change the vehicle state based on the received information from other tasks. For instance whenever operational limits are breached the Operational Limits monitor will change the vehicle state to "Blocked"
- Supervisors: Enables/disables other tasks depending on the current vehicle state. For instance, if the vehicle state is in "blocked", the vehicle supervisor would stop a maneuver task from being executed.
- Transports: Tasks transporting messages in and out of the message bus. UDP, TCP and Logging are such transport tasks.

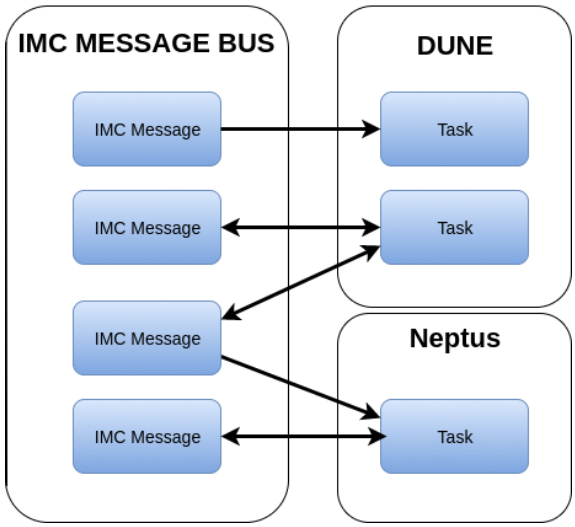


Figure 3.3: The interaction between components, DUNE Tasks and the IMC bus

Figure 3.3 visualizes the interaction between DUNE tasks and the IMC bus, and how the information flow between DUNE tasks, the message bus and Neptus. A newly created task would initially contain methods which are to be filled with desired functionality. Every task executes according to a common life-cycle and contains methods handling for all the messages the task consumes from the IMC bus. The methods involved in a DUNE tasks life-cycle are:

- onUpdateParameters(): Triggered when configuration parameters are changed.
- onEntityReservation(): Ensures unique identifiers by reserving entity identifiers.
- onEntityResolution(): Called to resolve entity names.
- onResourceAquisition(): Called when the task can acquire resources (open serial ports, sockets, etc).
- onResourceInitialization(): Called to initialize resource acquired.
- onResourceRelease(): Called to release acquired resources.
- onActivation() / onDeactivation(): Methods called when the task starts/resumes or stop normal execution, entering an idleness state.
- onMain(): Main task loop. Continuously executed code or function calls are placed here.
- consume(< M > message): Consumes a specific IMC message from the IMC bus.

All running instances of DUNE share the same code structure, but can be run under different configurations. A DUNE configuration file describes which tasks are initially enabled, what their initial parameters are, and also under which profile the task is to be run. These configurations can be edited real-time, either by changing the configuration file itself or through an interface in Neptus. DUNE uses profiles to allow multiple configurations to be defined in a single file. When the operator sets the profile of a task, the operator can choose from different profile options: *Hardware*, *Simulation*, *HIL* (Hardware In The Loop), *Always* or *Never*. For instance, running HIL mode allows for some sensor and actuator drivers to be enabled, along with tasks running in Simulation mode. If DUNE tasks are running in Simulation mode, all the sensor and actuator drivers would be disable and replaced with simulating tasks, i.e. behaving in the exact same manner as under real circumstances.

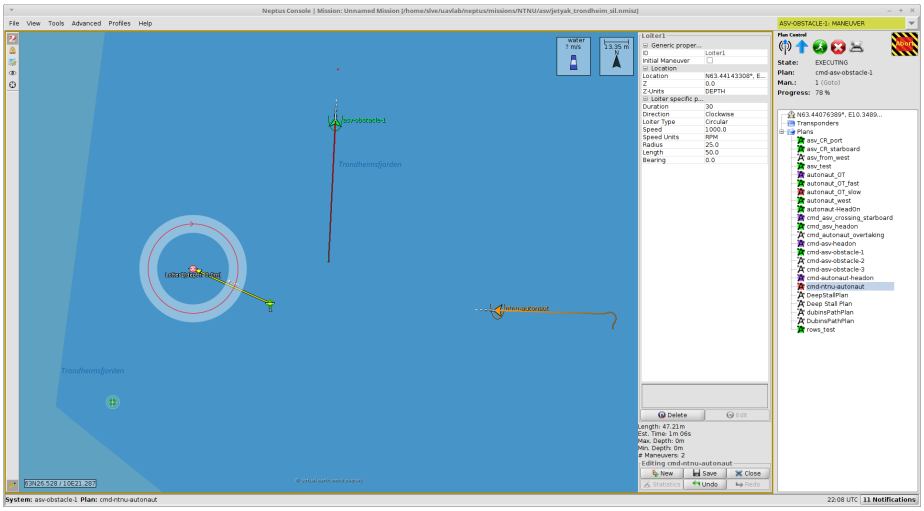
So, for instance, instead of having an GPS task transmitting real GPS messages from a ship, an GPS simulation task would generate a set of simulated GPS messages[30, 32].

3.1.2 Neptus

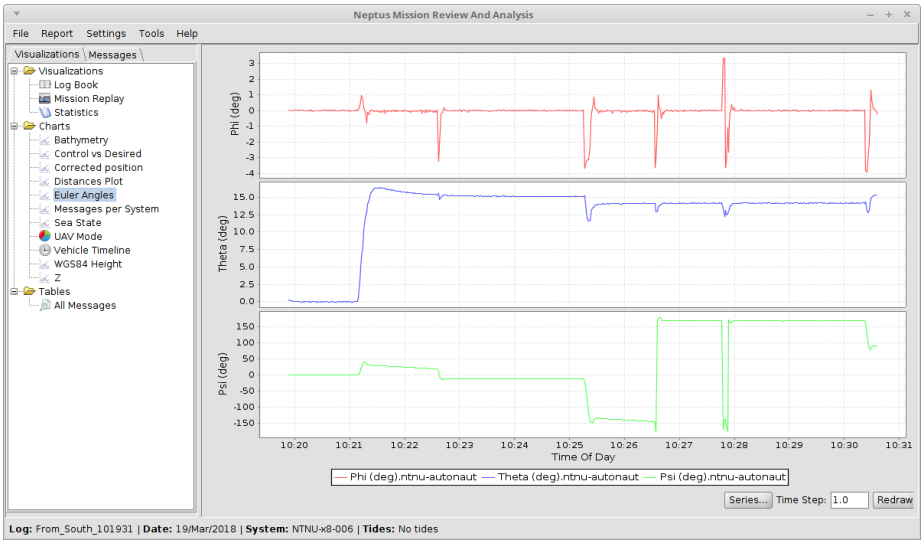
Neptus is a distributed C2 (Command and Control) framework used to command and monitor operations consisting of human operators, sensors and networked vehicles like AUVs, UAVs or ASVs. Neptus provides a coherent visual user interface to command and control the autonomous vehicles in the system, and uses IMC as the main communication interface. Neptus is used to assist the operator to take advantage of the capabilities these assets has to offer in terms of sensors and actuators without going into specific software details[31, 11].
The interactions of an human operator are classified according to the three phases of the mission life-cycle:

1. Planning: Prior to the execution of a mission. The operator prepare the mission plans and run simulations according to the mission objectives.
2. Execution: Preparation of vehicle deployment, system telemetry monitoring and mission plan execution.
3. Review and Analysis: This phase is performed on-site or after concluding the mission. Collected data is processed and analyzed, or mission plans are evaluated and replanned to achieve another desired outcome.

In Neptus, a mission is specified as a set map features, vehicle configurations and programmed plans. From the operator console the operator can generate mission plans that comprises different kinds of maneuvers and transitions which can be accomplished by a vehicle. The operational console can easily be adapted by adding from a range of plugin components, thereby customizing the user interface according to the mission specifications and operator demands. The console is defined by an XML configuration specifying the plug-ins and desired layout for a specific virtual mission environment. Mission planning can be done virtually in Neptus. A graphical editor provides a map representation of the mission site where maneuvers can be added and edited, see Figure 3.4a. The mission file containing the map, vehicle information, and plans are



(a) Operator Console



(b) Mission Review & Analysis

Figure 3.4: Neptus tools [24]

stored in an XML file. Each vehicle has a configuration file specifying the vehicles various capabilities. This includes information related to different maneuvers, on-board sensors, communication protocols, etc. Prior execution of a mission plan, the operator performs simulations to see the rough behaviour of the vehicle. Neptus provides three different types of simulation: behaviour prediction, software-in-the-loop (SIL) and hardware-in-the-loop. During SIL simulation sensor values and actuations are simulated, while HIL simulation can be used to test hardware during dry-run tests of the vehicle, sensors or actuators. Mission execution is performed through the the use of operational console, which enables operators to monitor vehicle execution, alter or create new plan specifications, send plans for execution or teleoperate vehicles, etc. Too aid the operator, Neptus also provides rough behavior simulations during mission execution whenever the vehicle is disconnected from the base station or is out of range[30]. After completing a mission the vehicle's stored mission data can be review in the Mission Review and Analysis (MRA), illustrated Figure 3.4b. In the interest of inspecting and analyzing the mission data, MRA decompresses the data into text files (which later on can be imported to programs like *Excel* or *MATLAB*), and provides several visualization and utilities to process and export data. MRA also has replay functionality, allowing post-visualization of entire missions.

3.1.3 The Inter-Module Communication (IMC) protocol

The Inter-Module Communication (IMC) is a message-oriented protocol developed and designed by LSTS for communication between heterogeneous vehicles, sensors, and human operators. IMC defines a common message set that is understood by the whole system, and all processes and devices in the toolchain use these messages to communication by exchanging real-time information about the environment and updated objectives[23]. IMC is also used for inter-process communication, inter-vehicle, and operator-vehicle communication data logging and dissemination to the network.

IMC defines a modular infrastructure and are divided into several logical message groups[26]:

1. *Mission control messages* are passed between Neptus and the Mission Supervisor defining the specification of a mission and its life-cycle.
2. *Vehicle control messages* are sent from the vehicle to issue maneuver commands and monitor the vehicle's state.
3. *Maneuver messages* defines maneuvers such as a GoTo message and related commands.
4. *Navigation messages* are used define the vehicle's navigation state.
5. *Guidance messages* defines the guidance and vehicle movement between waypoints
6. *Sensor messages* contains sensor readings from hardware controllers, e.g. IMU, GPS etc.
7. *Actuator messages* specifies the interface with hardware actuator controllers such as thrusters, rudders etc.

Figure 3.5 visualize the how IMC message flow in AutoNaut is executed. Navigation and Guidance controllers consumes sensor messages containing sensor data, e.g. GPS fix. Navigation tasks approximate the current state (position, attitude etc) of the vehicle and dispatches the EstimatedState message onto the IMC bus, where the message is ready to be consumed by other DUNE tasks. The EstimatedState message is consumed by multiple tasks, e.g Mission supervision, Vehicle Supervision and Maneuver Controllers, which consequently publish their respective IMC message. The message flow eventually reaches the Actuation Controller where, e.g. the thrust actuation and rudder position are set.

All IMC messages are comprised of a header, payload, and a footer. The header contains information such as type, version, message identifier, timestamp, origin and destination. The content of the payload varies according to the message identifier, it usually includes multiple variables. It could, for instance, contain data from a sensor or the vehicle states. The footer contains a checksum for verification.

The header also contain a synchronization number marking the beginning of a message and allows for endianness detection. In order to store or transmit a message, the message has to be encapsulated in a packet and serialized. Serialization is performed

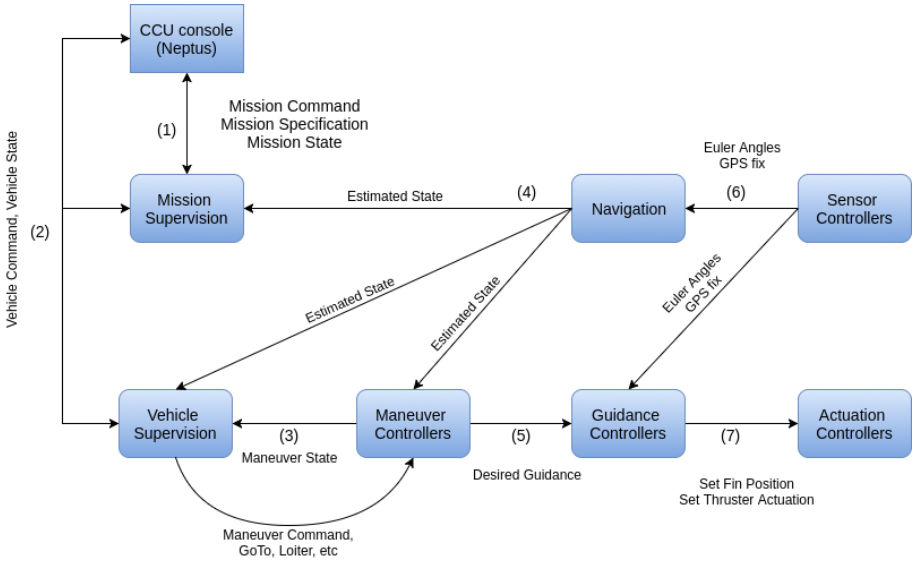


Figure 3.5: IMC message flow (Based on image from [26])

by translating the data fields into a binary stream, still defined in the same order. The recipients are able to ensure data integrity by inspecting the synchronization number, deducing the byte order of the remaining data fields and perform the necessary conversions for correct interpretation[30, 26].

```
<message id="301" name="Set Thruster Actuation" abbrev="SetThrusterActuation" source="vehicle">
  <description>
    Actuate directly on a thruster.
  </description>
  <field name="Thruster Number" abbrev="id" type="uint8_t">
    <description>
      The identification number of th destination thruster.
    </description>
  </field>
  <field name="Actuation Value" abbrev="value" type="fp32_t" min="-1" max="1">
    <description>
      Actuation magnitude.
    </description>
  </field>
</message>
```

Figure 3.6: SetThrusterActuation message from IMC.xml

The complete IMC protocol is defined and documented in a single eXtensible Markup Language (XML) file. Depicted in Figure 3.6 is a SetThrusterActuation IMC message, which specifies the message identifier, the source of where it originates from and the payload the message contains. Describing the IMC protocol with a XML document is very practical for continuous development, testing and creating new IMC messages.

3.1.4 GLUED

GLUED (GNU/Linux Uniform Environment Distribution) is a minimal Linux distribution designed to be targeted at embedded systems. GLUED has a small footprint, around 10MB large, and has a boot time of 2-5 seconds depending on the target machine and peripherals. To build GLUED for a target system a configuration file needs to be created for the particular system. The configuration file contain details which specifies the system architecture, network configuration and packages that will be a part of the target distribution. GLUED is easy to configure as packages can easily be added and removed from the distribution by editing or creating new configuration file[22].

The GLUED distribution is cross-compiled for the target distribution, i.e. building the binaries for a platform other than the one on which the compiler is running. This allows for software development on a more powerful computer rather than on the embedded system itself. As a result of the cross compilation and the fast boot time, the time to build a software projects are significantly reduced and the uncontrollable time while booting are minimised[10].

3.2 COLREGS

In the history of maritime activities the time shows that the needs of safety gradually came to the fore, and in the wake of accidents at sea, and have resulted in huge changes in order to reduce the hazards encountered at sea. For the maritime industry collisions at sea have always been a great problem, and the causes for maritime collision are many. In 1972, further measures to prevent collision at sea where made, when the

International Marine Organization (IMO) formalized the International Regulations for Preventing Collisions at Sea (COLREGs). COLREGs is a set of regulations and navigation rules to be followed by ships and other marine vessels at sea to prevent collisions between two or more vessels[1].

All marine surface vessels are required to adhere to COLREGs at all times in order to minimize or eliminate the risk of collisions. Consequently, it is necessary for the CAS to also adhere to COLREGs in order to produce predictable maneuvers when encountering other vessels. The COLREGs rules are divided into five parts (A-E), however, the rules covered in Part B - Steering and Sailing is the most relevant and applicable for AutoNaut and this thesis.

The most relevant rules for the ASV in order to act satisfyingly in a collision avoidance situation are specified in the following section[17, 16]:

3.2.1 COLREGs Rules

Rule 8 - Actions to avoid collision

(b). *Any alteration of course and/or speed to avoid collision shall, if the circumstances of the case admit, be large enough to be readily apparent to another vessel observing visually or by radar; a succession of small alterations of course 16 and/or speed should be avoided.*

(d). *Action taken to avoid collision with another vessel shall be such as to result in passing at a safe distance. The effectiveness of the action shall be carefully checked until the other vessel is finally past and clear.*

Rule 13 - Overtaking

(a). *Notwithstanding anything contained in the Rules of part B, sections I and II, any vessel overtaking any other shall keep out of the way of the vessel being overtaken.*

(b). *A vessel shall be deemed to be overtaking when coming up with another vessel from a direction more than 22.5 degrees abaft her beam, that is, in such a position with*

reference to the vessel she is overtaking, that at night she would be able to see only the sternlight of that vessel but neither of her sidelights.

(c). *When a vessel is in any doubt as to whether she is overtaking another, she shall assume that this is the case and act accordingly.*

(d). *Any subsequent alteration of the bearing between the two vessels shall not make the overtaking vessel a crossing vessel within the meaning of these Rules or relieve her of the duty of keeping clear of the overtaken vessel until she is finally past and clear.*

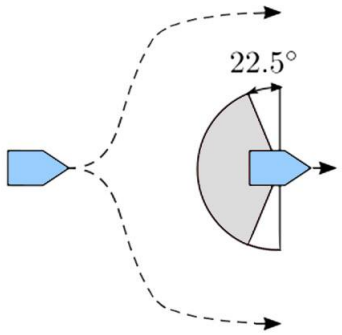


Figure 3.7: COLREGS rule 13: Overtaking. In overtaking situations the vessel overtaking another vessel may pass on either side with a safe distance [34]

Rule 14 - Head-on situation

(a). *When two power-driven vessels are meeting on reciprocal or nearly reciprocal courses so as to involve risk of collision each shall alter her course to starboard so that each shall pass on the port side of the other.*

(b). *Such a situation shall be deemed to exist when a vessel sees the other ahead or nearly ahead and by night she could see the masthead lights of the other in a line or nearly in a line and/or both sidelights and by day she observes the corresponding aspect of the other vessel.*

(c). *When a vessel is in any doubt as to whether such a situation exists she shall assume that it does exist and act accordingly.*

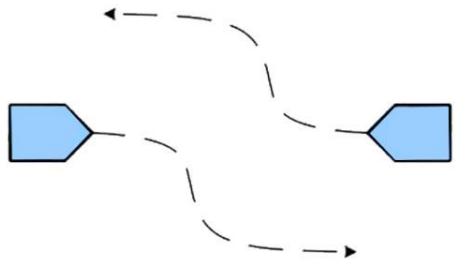


Figure 3.8: COLREGS rule 14: Head-on. In head-on situations both vessels are give-away vessels and should change their course starboard.

Rule 15 - Crossing situation

When two power-driven vessels are crossing so as to involve risk of collision, the vessel which has the other on her own starboard side shall keep out of the way and shall, if the circumstances of the case admit, avoid crossing ahead of the other vessel.

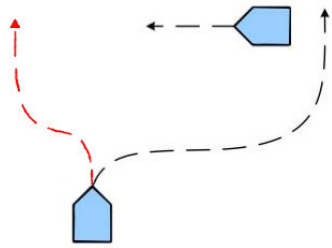


Figure 3.9: COLREGs rule 14: crossing. In a crossing situation the vessel which has the other vessel on its starboard has give-away, and should keep out of the way. The red trajectory shows an incorrect course of action.

Rule 16 - Actions by give-way vessel

Every vessel which is directed to keep out of the way of another vessel shall, so far as possible, take early and substantial action to keep well clear.

Rule 17 - Actions by stand-on vessel

(a). (i). *Where one of two vessels is to keep out of the way the other shall keep her course and speed.*
(ii). *The latter vessel may however take action to avoid collision by her manoeuvre alone, as soon as it becomes apparent to her that the vessel required to keep out of the way is not taking appropriate action in compliance with these Rules.*

(b). *When, from any cause, the vessel required to keep her course and speed finds herself so close that collision cannot be avoided by the action of the give-way vessel alone, she shall take such action as will best aid to avoid collision.*

3.2.2 Rule Selection for an ASV

In the case of a possible collision situation, the ASV needs to determine how to react to the various COLREGs situation it is facing. In order to identify the applicable COLREGs rule in a certain COLREGs situation, the ASV needs to figure out the position of the obstacles relative to itself. This can be done by calculating the relative bearing between the ASV and the obstacles. The relative bearing is deduced from:

$$\beta = \text{atan2}(y_A - y_B, x_A - x_B) - \psi_A \tag{3.1}$$

where A is the ASV while B is the obstacle. The algorithm applied in this thesis, does not use the relative bearing alone to determine if there is a COLREGs situation, the exact approach is described in section 4. However, the relative bearing is used to determine whether the obstacle is located at the port or starboard side, which is essential in deciding which actions should be taken.

To differentiate the COLREGs situation and when to apply actions, four different sectors are defined. These sectors are visualized in Figure 3.10 and defined as

1. Head-on: $\beta \in [-22.5°, 22.5°)$
2. Crossing from Port: $\beta \in [-111.5°, 0°)$
3. Crossing from Starboard: $\beta \in [0°, 111.5°)$
4. Overtaking: $\beta \in [111.5°, 180°) \cup [-180°, -111.5°)$

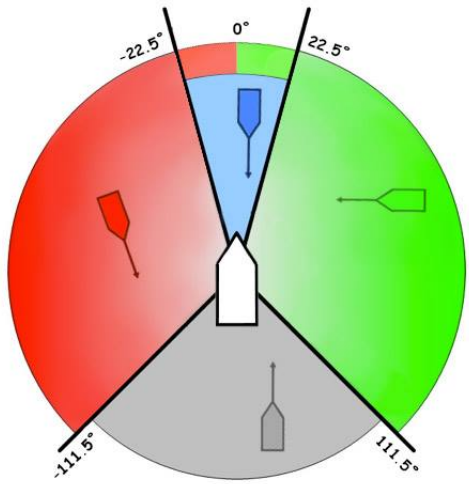


Figure 3.10: COLREGs rule selection. A diagram displaying how a COLREGs situation is classified.

The limits of the different sections was defined in [18], and it has been tested during simulations seen in chapter 7 and the choices seems reasonable.

Note that the Crossing sectors are overlapping the Head-on section. The reason is that you can still experience a Crossing situation when the obstacle is inside the

Head-on section. Consider, for example, a situation where an obstacle is in the head-on section, and moving with a heading of 90°relative to the ASV heading. Hence not moving towards the ASV, so this situation be a crossing situation, but still with a potential collision cost in the head-on section. This is why the velocity vectors and the angle between the velocity vectors, mentioned in section 4, are important for deciding whether it is a COLREGs situation or not.

Chapter 4

Collision Avoidance

The following section describes the system architecture and the collision avoidance system integrated to the LSTS system in this project. The CAS is based on the concept described in the article *Ship Collision Avoidance using Simulation-Based Control Behavior Selection with Predictive Hazard Assessment*[18], and further explained by Inger Hagen in [15].

4.1 System Architecture

An overview of the system architecture, displaying the main modules of the navigation systems and the information flow between them, is given in Figure 4.1.

For the ASV to obey the rules-of-the-road during when encountering other manned and/or unmanned vehicles, and perform the correct and predictable actions in hazardous situations the CAS needs to be COLREGS compliant. From the Mission Planner the CAS receives a desired course and speed based on a sequence of waypoints. These waypoint could for instance be generated by a human operator in Neptus. Meanwhile, the Sensor System composed of e.g. Radar, AIS, Lidar, etc. continually provides navigational measurements, measured obstacle positions and predicted trajectories. Based on this information the CAS module searches for a COLREGs compliant and collision-free

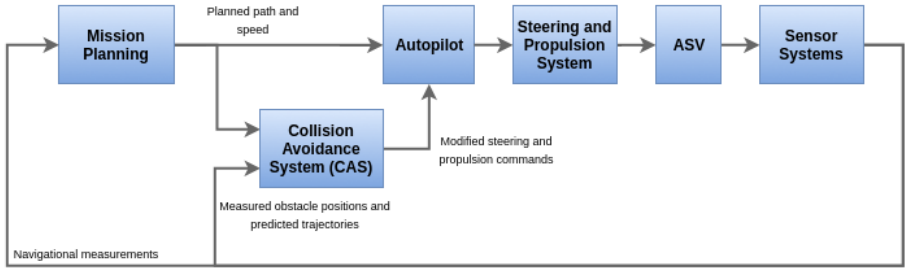


Figure 4.1: Overview of the main modules in the system and the information flow between them.

trajectories through a series of simulations with a finite set of offsets to the nominal course. The offset associated with the lowest cost while producing a collision-free and COLREGs compliant trajectory is selected as the new modified course reference and is passed on to the autopilot. If there is no obstacles in the vicinity the desired course and speed from the Mission Planner are sent to the Autopilot, composed of a Heading and a Speed Controller. Consequently, the ASV's Autopilot and Steering and Propulsion System will apply these (modified) steering and propulsion commands, ensuring a COLREGs compliant trajectory associated with the lowest collision risk as possible. When the original desired course and trajectory no longer contains any possible potential collisions, the ASV would return to the desired nominal path and proceed towards the target destination.

The following information must be available for the collision avoidance to be applicable:

- List of obstacle's position, velocities .
- A desired nominal path to the target.
- A mathematical model of the ship.
- Real time measurements of the ship's states, e.g. position, velocity, heading.

4.2 Collision Avoidance System

Figure 4.2, depicts an overview of the architecture of the CAS control algorithm from [18]. The architecture has been simplified whereas the weather scenarios is currently removed as it is not considered during the simulation, and still have not been taken in to account.

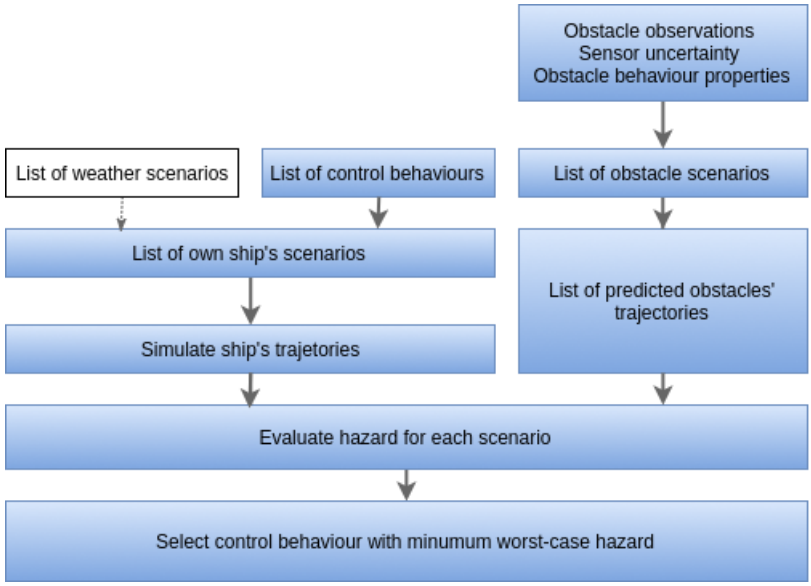


Figure 4.2: Collision avoidance control algorithm (Based on image from [18])

The collision avoidance and selection of the control behavior with the minimum hazard associated to it is realized as a optimization problem. A finite number of control behaviors and predicted obstacles' trajectories are combined into finite scenario hazard minimization problems, which, consequently, are being evaluated over a finite horizon. The hazard associated to a certain scenario resulting from a given control behavior is evaluated using a simulator, provided in the LSTS toolchain, that make predictions based on the dynamics of the ship, steering and propulsion system, the control behavior and the current position and velocity. In the center of the hazard

evaluation is a cost function performing the evaluation based on the collision risk, collision cost and compliance with the rules of COLREGs for the different scenarios. The control behaviour with the minimum hazard associated is the desired control behavior. This optimization problem is re-optimized based on updated information at regular intervals, e.g. 5 seconds.

In the following subsections the main components of the CAS, and their interactions, are being described in some detail.

4.2.1 Trajectory Prediction

When predicting an obstacles' future trajectory there are considerable uncertainties present. Turns or quick maneuvers in different directions, are actions an obstacle could make which would it harder to predict its future trajectory. To simplify the problem obstacles' are assumed to only move in a straight-line trajectory.

The straight-line prediction of an obstacle are given by

$$\bar{x}_i(t) = \hat{x}_i + \hat{u}_i(t - \tau_i) \tag{4.1}$$

$$\bar{y}_i(t) = \hat{y}_i + \hat{v}_i(t - \tau_i) \tag{4.2}$$

where x and y are the positions, t is a future point in time, and τ_i is the time of the last observation.

Similarly for AutoNaut, whose purposes is to operate far out at sea where there are fewer obstacles, a straight-line trajectory prediction would work sufficiently for the prediction of the ASV's motion. Two methods have been looked on and tested in [15]. It will provide a computational load that is not too heavy for the computational limitations on-board AutoNaut.

4.2.2 Control Behaviors

To determine the desired control behaviours the CAS simulates the ship's trajectory by evaluating a finite set of control behaviours. The predictive simulation runs scenarios defined by the current state of the ship, the set of control behaviours and the predicted

trajectories of obstacles in the vicinity. Each predicted trajectory is then assigned a computed cost, based on an evaluation of the hazardous situation and COLREGS compliance.

The following is the set of control behaviours which is to be considered when computing the least hazardous control behaviour

- Course offset: -90, -75, -60, -45, -30, -15, 0, 15, 30, 45, 60, 75, 90 degrees
- Speed factor: 1.0 , which is equal to nominal (wave) propulsion

Resulting in a $13 \times 1 = 13$ possible control behaviours to choose from. In typical implementation the set of control behaviours should be as extensive as computation time allows. The minimum set of alternative control behaviors that [18] recommends are not met, due to AutoNaut not having the same freedom to change its propulsion as it normally is generated by waves.

From a safety point of view it is desirable to evaluate as many scenarios as possible. In order to increase of control behaviors There are a few measures that would increase the number of scenarios, one is increasing the number of course offsets, another one is to apply auxiliary propulsion. Due to the limited power availability applying auxiliary propulsion should only be enabled whenever the collision hazard associated to a scenario is above a certain threshold, during flat calm conditions or when solar energy is easily accessible. However, at this stage the logic that autonomously switches over to auxiliary propulsion when needed have yet not been implemented, thus the focus has been on merely the wave propulsion.

4.2.3 COLREGS Compliance

For AutoNaut to operate with a risk-reducing and predictable behaviour to the operators and other vessels it is required to adhere to the rules specified by the COLREGs. How the CAS uses the available information to evaluate COLREGS compliance in a hazardous situation, e.g. encountering an obstacle, is illustrated in Figure 4.3.

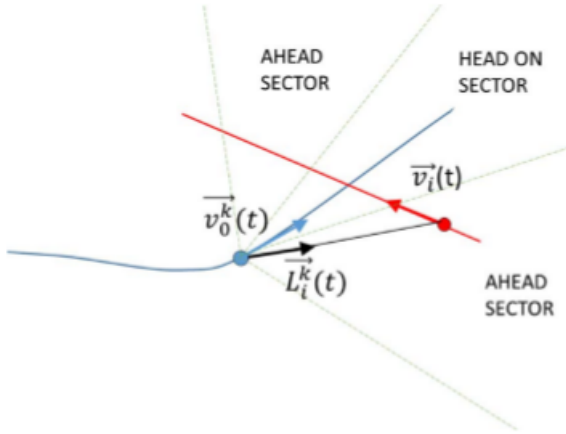


Figure 4.3: Hazard evaluation in scenario k at a future time t [18]

Table 4.1: Parameters used during evaluation of COLREGs compliance

Parameter	Description
$\overrightarrow{v}_0^k$	Predicted velocity of the ASV at a future time instant t in scenario k.
$\overrightarrow{v}_i$	Predicted velocity of obstacle i at a time t in scenario k.
$d_{0,i}^k$	Predicted distance between the vessels at a time t in scenario k.
d_i^{cl}	The distance within which the COLREGS is said to apply.

The blue curve illustrates AutoNaut's predicted trajectory, whereas the red curve illustrates the obstacles straight-line predicted trajectory, both based on their respective position, velocity and heading. The blue and red dots illustrates the predicted position at some future time instant t. The black vector, denoted $L_i^k(t)$, is a unit vector in the Line-of-Sight (LOS) direction from AutoNaut to the obstacle i. Along with the two velocity vectors $\overrightarrow{v}_0^k$ and $\overrightarrow{v}_i$, are the two distance parameters d_i^{cl} and $d_{0,i}^k$ used in the evaluation of COLREGS compliance to determine the level of risk at a given time t in the scenario k. The description on the evaluation of COLREGS compliance when interacting with other vehicles is defined as:

- **CLOSE**: A obstacle is said to be close to AutoNaut at a time i in a scenario k if $d^k_{o,i}(t) < d^{cl}_i$.

- **OVERTAKEN**: AutoNaut is said to be overtaken by an obstacle i at time t in scenario k if

$$\overrightarrow{v}^k_0(t) \cdot \overrightarrow{v}_i(t) > \cos(68.5^\circ)|\overrightarrow{v}^k_0(t)||\overrightarrow{v}_i(t)| \tag{4.3}$$

  and if it is close, and if the obstacle has a higher speed than AutoNaut, $|\overrightarrow{v}_i(t)| > |\overrightarrow{v}^k_0(t)|$.

- **STARBOARD**: An obstacle i is said to be STARBOARD of AutoNaut at time t in scenario k if the bearing angle of of $L^k_i(t)$ is larger than the heading angle of AutoNaut.

- **HEAD-ON**: An obstacle i is said to be HEAD-ON at time t in scenario k if it is close to AutoNaut and its speed is not close to zero and

$$\overrightarrow{v}^k_0(t) \cdot \overrightarrow{v}_i(t) < -\cos(22.5^\circ)|\overrightarrow{v}^k_0(t)||\overrightarrow{v}_i(t)| \tag{4.4}$$

$$\overrightarrow{v}^k_0(t) \cdot \overrightarrow{L}^k_i(t) > \cos(\phi_{ahead})|\overrightarrow{v}^k_0(t)| \tag{4.5}$$

  where the angle ϕ_{ahead} is to be selected.

- **CROSSING**: An obstacle i is said to be CROSSING if it is close to AutoNaut and

$$\overrightarrow{v}^k_0(t) \cdot \overrightarrow{v}_i(t) < \cos(68.5^\circ)|\overrightarrow{v}^k_0(t)||\overrightarrow{v}_i(t)| \tag{4.6}$$

The angles used in the evaluation of COLREGs are illustrated and defined in Section 3.2.2 and Figure 3.10.

Any violations of the COLREGS rule 14 or 15 between AutoNaut and a obstacle i

is denoted by a binary indicator $\mu_i^k(t) \in 0, 1$

$$\begin{aligned} \mu_i^k(t) &= RULE\ 14\ or\ RULE\ 15 \\ RULE\ 14 &= CLOSE\ \&\ STARBOARD\ \&\ HEAD\text{-}ON \\ RULE\ 15 &= CLOSE\ \&\ STARBOARD\ \&\ CROSSING\ \&\ NOT\ OVERTAKEN \end{aligned}$$

Which incorporate rule 13 as it states that it is the overtaking vessel that shall keep out of the way.

4.2.4 Hazard Evaluation Criterion

A collision risk factor can be expressed as

$$R_i^k(t) = \begin{cases} \frac{1}{(t-t_0)^p}\left(\frac{d_i^{safe}}{d_{0,i}^k(t)}\right)^q, & if \quad d_{0,i}^k(t) \leq d_i^{safe} \\ 0, & otherwise \end{cases} \tag{4.7}$$

where t_0 and $t > t_0$ are the current and the prediction time. The distance d_i^{safe} and the exponent $q \geq 1$, are affecting the system's ability to comply with COLREGS rule 16, i.e. to take early and substantial action to ensure a safe distance, and they must be chosen accordingly. The exponent $p \geq 1/2$ weighs how the risk evolves as a function of time until the event occurs. This is achieved by using the inverse proportionality with time to prioritized occurrences that are close over those more distant.

The cost associated with collision with an obstacle at a given time i in scenario k is given by

$$C_i^k(t) = K_i^{coll}(t)|\overrightarrow{v}_0^k(t) - \overrightarrow{v}_i^k(t)|^2 \tag{4.8}$$

The cost of a collision is directly connected to the kinetic energy involved, as given by the squared velocity of the obstacle and the ASV. The factor $K_i^{coll}(t)$ depends on the size of the obstacles. In the event of a hazardous situation with multiple obstacles and where a collision may be unavoidable, these factors are important to consider as they

help limit the consequences.

The hazard associated to a scenario k at time t_0 is

$$H^k(t_0) = \max_i \max_{t \in D(t_0)} \left(C_i^k(t) R_i^k(t) + \kappa_i \mu_i^k(t) \right) + f(P^k, \chi_{ca}^k) \tag{4.9}$$

where t_0 is the current time, and $D(t_0) = \{t_0, t_0 + T_s, ..., T_0 + T\}$ is the discrete sample times. T_s and T are the discretization interval and the prediction horizon respectively.

In order to enhance the predictability of the control behaviour, and to favor a straight trajectory of AutoNaut the term $f(\cdot)$ is included

$$f(P, \delta) = k_P(1 - P) + k_\chi \chi_{ca}^2 + \Delta_P(P - P_{last}) + \Delta_\chi(\chi_{ca} - \chi_{ca,last}) \tag{4.10}$$

where two penalty functions, Δ_p and Δ_χ, are applied along with two tuning parameters k_p and k_χ to favor straight trajectories and prioritize keeping constant speed and course. This is done by penalizing any commands resulting in course offsets to port rather than starboard, thus violating COLREGS rules 13, 14 and 15.

4.2.5 Control Decision

The optimal control behaviour for a scenario $k \in \{1, 2, ..., N\}$ at time t_0 is the control behaviour with the minimal hazard $H^k(t_0)$ associated to it after comparing their hazards. The optimal control behavior with minimal $H^k(t_0)$ can, therefore, be expressed as:

$$k^\star(t_0) = \arg\min_k H^k(t_0). \tag{4.11}$$

The newly evaluated optimal control behaviour will consequently be sent to the autopilot for action execution. This optimization process is repeated at regular intervals such that new sensor information acquired can be utilized, [18] suggest intervals of 5 seconds.

There are a selection of tuning parameters involved in this control behaviour optimization is important, and there are many factors to consider when tuning these

parameters. In the CAS algorithm the main aspects regarding tuning these parameters have been to [15]:

- Avoid collisions
- Adhere to the COLREGS
- Keep nominal course

Part I

Part 1: Software

Chapter 5

Collision Avoidance Development

Previously, the CAS has been applied to a more responsive system, e.g. Telemetron[15], a system that has more degrees of freedom regarding change of course and speed depending on the circumstances than AutoNaut. The combination of not having the same operational capabilities as Telemetron and integrated on a new platform, LSTS, could be some of the factors to a rather interesting behaviour during simulations. During different simulation scenarios various issues required attention and handling in order to achieve the correct performance. This chapter will mentioned the main changes applied to the CAS to ensure corrective measures are made to handle similar situations.

Replaced the Collision Cost

The cost associated with collision with an obstacle at a given time, $C_i^k(t)$, defined in equation 4.8 has currently been replaced with the simpler version:

$$C_i^k(t) = K_i^{coll}(t) \tag{5.1}$$

This comes as a result of the behaviour expressed during certain simulation scenarios. If the USV, for some reason, find itself inside the distance d_i^{safe}, the collision risk factor, $R_i^k(t)$ along with $C_i^k(t)$ will dominate the cost function. As a result the USV starts to go in different directions not taking into considerations where the obstacle is located. This is due to the kinetic energy of equation 4.8 which only consider the velocities of the vessels and for this reason it has temporarily been simplified. With the simplified version the cost associated with a collision is merely dependant on the size of the vessel, hence trying to avoid larger vessels before smaller, since smaller vessels are usually more maneuverable and more likely too perform evasive manoeuvres themselves. If the USV now finds itself inside d_i^{safe} it will proceed towards the desired waypoint, even if it had a heading offset. Although such situations normally would not occur, they might as an effect of a certain chain of events. In these situations it has so far been concluded that it is better with a straight trajectory towards the waypoint rather than a set of random actions. While the current handling is admittedly not preferable, it has proved sufficient a this stage of the project, however a better handling of such events is required later on.

Course penalty function

If an obstacle entered the region where it is recognized to be close and at the same time creating either a head-on or crossing situation, the USV would for some reason execute a port turn rather than starboard, thus violating COLREGS. This issue was also related to the cost function, more specifically the penalty function Δ_χ.

$$\Delta_\chi = \begin{cases} K_{\Delta_\chi,port}(\chi_{ca} - \chi_{ca\_last})^2, & \text{if turn to port.} \\ K_{\Delta_\chi,starboard}(\chi_{ca} - \chi_{ca\_last})^2, & \text{if turn to starboard.} \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

The penalty function favours maintaining the course while at the same time penalize course changes to port more than to starboard, and for this reason it is necessary in the implementation. So, in order to obtain a more COLREGs compliant performance a

similar penalty function has been added, namely χ:

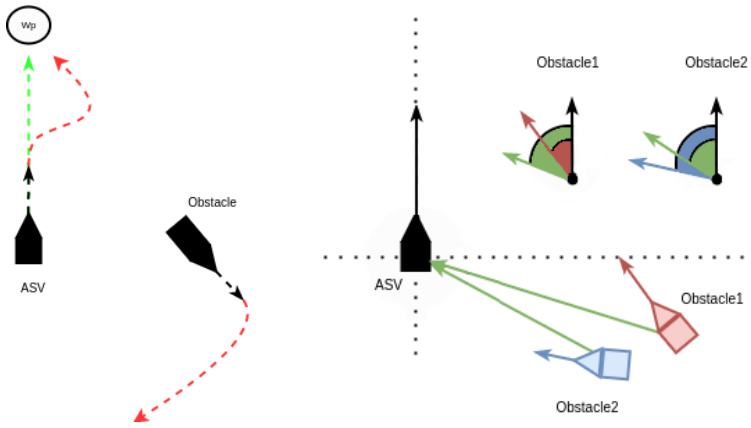
$$\chi = \begin{cases} K_{\chi,port}(\chi_{ca})^2, & \text{if turn to port.} \\ K_{\chi,starboard}(\chi_{ca})^2, & \text{if turn to starboard.} \end{cases} \tag{5.3}$$

In contrast to Δ_χ, χ weighs the actual course and not the difference of the course. Similarly, the penalty function still penalize course changes towards port above starboard, however now the original heading offset is used as reference. To illustrated the difference better the χ-function will be composed of a symmetric cost if the penalty parameters $K_{\chi,port} = K_{\chi,starboard}$, conversely, the Δ_χ continuously changes its reference dependant to the current course.

Passing

In Figure 5.1a has an obstacle already passed the USV when it initiate a starboard turn, visualized as the red trajectory. This special event causes an unwanted response in the COLREGs weighting inside the cost function. The COLREGs implementation recognizes this event as a crossing, consequently the USV also make a turn. Already passed, the vessel should not affect the behaviour of the USV which should behave as illustrated by the green trajectory.

Figure 5.1b shows how this was issue was resolved. Two obstacles are located inside the fourth quadrant, $\phi > 90$, and still inside the crossing section defined in Figure 3.10. The green arrow symbolizes the distance vector between the vessel and the USV, while the red, blue and black arrows corresponds to the velocity vector of each vessel. In the first quadrant the four different vectors form two subfigures, namely *Obstacle1* and *Obstacle2*. Each subfigure contains two angles formed by the different vectors. One angle is composed of the velocity vector of the USV and the direction vector while the second is formed by the velocity vectors of the USV and the obstacle. The angles are found by applying the dot product. By comparing the two angles it can be decided whether the vessel will cross behind or in front of the USV. For instance the red angle is smaller than the green angle stating that it could possibly cross in front of the USV, if the speed and distance allow it, and therefore the vessel still has to be considered.



(a) Wrong behaviour during obstacle passing

(b) Corrected by comparing angles

Figure 5.1: Passing logic. Problem and solution.

In the other subfigure the blue angle is greater than the green resulting in the vessel crossing behind the USV, consequently these vessels are no longer considered crossing.

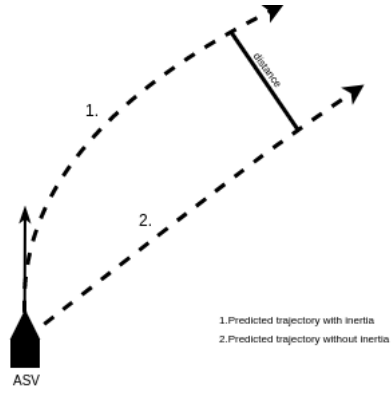


Figure 5.2: Linear Prediction with or without inertia

Work in progress

Some functionality is still under development and have not been finalized, due to time limitations and prioritizing the hardware implementation. Most important is the consideration of the moment of inertia of AutoNaut. The USV is a slow-moving vessel it is important to consider the inertia experienced during a turn when predicting the path trajectory. The collision avoidance algorithm uses linear prediction to estimate future position, illustrated in Figure 5.2 as trajectory nr.2. By not considering the inertia it will occur a deviation in the distance between the actual position of AutoNaut and the estimated, which is something that should be minimized as much as possible.

Chapter 6

System Design

This chapter explains the system overview and design of the simulation environment designed in the preliminary project, while the image have been reused the text has been rewritten and corrected during this thesis.

6.1 System Overview

In order to simulate the behaviour of the collision avoidance a simulation environment providing the necessary information and functionality had to be designed. Figure 6.1 depicts an overview of the designed system. The system overview illustrates the information flow in the system and between AutoNaut and other simulated vehicles, including the DUNE tasks performing the necessary computations required to generate predictable actions for AutoNaut in any situations, hazardous or not.

To achieve this simulation system, four elements were required:

1. DUNE instances of AutoNaut and obstacle vessels transmitting vehicle states.
2. A DUNE task that converts *EstimatedState* messages from the obstacles into *ObstacleState* messages.
3. A DUNE task that acquires the necessary information to determine the behaviour and actions required of AutoNaut in any situation.

4. A collision avoidance algorithm, which generates an offset heading and speed to avoid any hazardous situations.

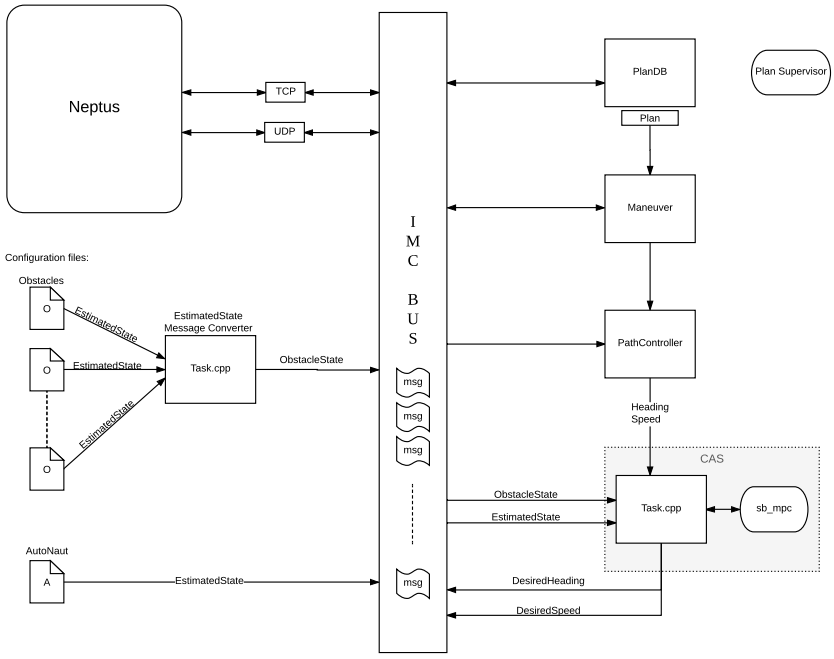


Figure 6.1: System Overview

6.2 Implementation

To achieve the functionality the system overview illustrates, instances of AutoNaut and the obstacle need to be created in both Neptus and DUNE. In Neptus, a new vehicle is created by adding an XML file containing vehicle specific information to the *vehicles-defs* folder. These XML files describe the attributes and properties the vehicle possesses in Neptus, such as name, appearance, feasibleManeuvers, communication-means, etc.

In DUNE, configuration files for AutoNaut and each of the obstacle vehicle need

to be created. These configuration files specify the tasks that are enabled or disabled and state the profile a task is to be run at. As an default every vehicle transmits the IMC message *EstimatedState*. The *EstimatedState* contains vehicles states such as the vehicles velocity, heading, the static longitude and latitude, and its offset in North and East direction with respect to LLH. In Figure 6.1 AutoNaut dispatches the EstimatedState message onto the IMC bus, where it is available to be used by other DUNE tasks such as CAS or Pure Pursuit. However, to be able to detect a potential collision situation, AutoNaut needs to somehow receive information about the obstacles in the vicinity. Thus, a new IMC message had to be initialized, ObstacleState, with the same functionality as an AIS, stating obstacle states e.g. position, velocity, heading, size, etc. The IMC message ObstacleState, is based on EstimatedState, and customized to fit the input variables of the sb-mpc algorithm[1].

The configuration file of the obstacle enables a DUNE task, *EstimatedStateToTarget*, which consumes the *EstimatedState* from the obstacles, converts them into ObstacleState and dispatches them onto the IMC bus. Consequently, enabling a simulated AIS network between AutoNaut and the obstacles, which also happens to be a system requirement for the real AutoNaut - AIS detection of ships and boats.

After a plan is created in Neptus by the operator, the plan supervisor takes care of commands and controls regarding the execution of mission plan and triggers the required maneuver. According to Figure 4.2 maneuvering commands are sent to Guidance/Navigation, the PathController, where the desired heading and speed are computed. These values does not take into account if there are any obstacles present.

However, the CAS Task.cpp, see 6.1, now has all the necessary information required in order to compute a COLREG compliant trajectory whenever encountering other vehicles. Enabled in the AutoNaut configuration file, the CAS Task.cpp consumes several IMC messages as depicted in Figure 6.2.

Inside the CAS Task.cpp the sb-mpc algorithm is called whenever there are any obstacles inside a predetermined distance, a distance where the COLREGS should count. The algorithm and new offsets are computed every 3 seconds. This iterative

[1]The simulation-based model predictive control (sb-mpc) algorithm computes the necessary heading and speed offsets required to navigate to a safer trajectory while maintaining COLREGS compliance. The code and further information about the algorithm can be found in [15].

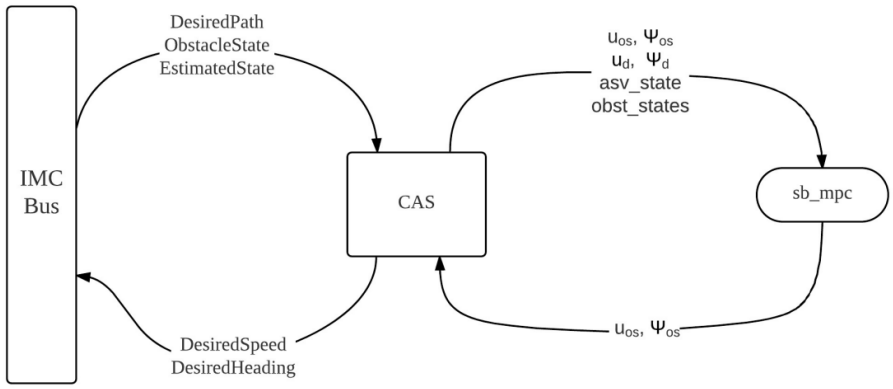


Figure 6.2: IMC messages going in to and out of the CAS

computation cycle is visualized in Figure 6.2 When there is no obstacles within the COLREGS range, the sb-mpc would not compute any heading or speed offset, resulting in the heading and speed from the PathController becoming the desired heading and speed. The AutoNaut continues to follow its initial path.

Chapter 7

Simulation Results

To investigate the performance of the collision avoidance system a wide range of simulation scenarios have been studied. Scenarios involving single and multiple obstacle avoidance have been created and executed with different goals in mind:

1. Debug: Find, identify and fix issues and situations where the vessel behaves unexpectedly.
2. Development: Develop functionality required for a predictable and logical behaviour.
3. Results: Produce results, and show the robustness and versatility of the CAS.

As a result of systematically following these steps, issues such as those mentioned in Chapter 5 has been fixed.

In all of the presented simulations the same parameter settings have been used. The parameters used are displayed in Table 7.1, and the possible control offsets are as mentioned in section 4.2.2:

- Course offset: -90, -75, -60, -45, -30, -15, 0, 15, 30, 45, 60, 75, 90 degrees
- Speed factor: 1.0 , which represents nominal propulsion

Table 7.1: Simulation and cost function parameters

Parameter	Value	Description
T	400.0	Predicted horizon (simulation time)
dt	1	Time step for the simulation
p	0.5	Weight on time to evaluation instant
q	4	Weight on distance at evaluation instant
d_{close}	400.0	Distance where COLREGs apply
d_{safe}	50.0	Minimal distance which is considered as safe
K_{coll}	0.5	Collision cost
ϕ_{AH}	60.0	Angle which specify if obstacle is ahead
ϕ_{OT}	68.5	Angle which specifies if an obstacle is overtaking
ϕ_{HO}	22.5	Angle which specifies if an obstacle is crossing
ϕ_{CR}	68.5	Angle which specifies if an obstacle is head-on
κ	3	Cost of not complying with the COLREGs
$K_{\chi,starboard}$	0.9	Cost of deviating to starboard from the nominal heading
$K_{\chi,port}$	10.0	Cost of deviating to port from the nominal heading
$K_{\Delta\chi,starboard}$	0.1	Cost of changing the heading offset to starboard
$K_{\Delta\chi,port}$	0.5	Cost of changing the heading offset to port

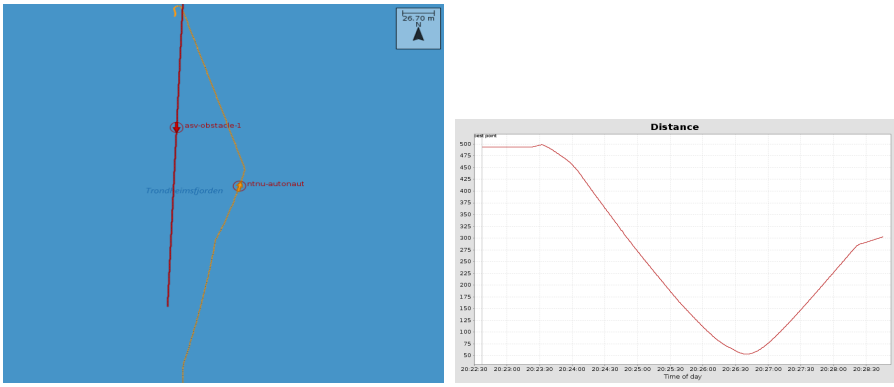
The simulation results presented are snapshots of the different scenarios. The snapshots are obtained, post simulation, from the MRA-tool provided by the LSTS toolchain. In the snapshots the orange arrow and line represent AutoNaut, while the other colors, e.g purple, white, red, represent the obstacles. The waypoints are not displayed, however the obstacles will always during simulation follow a straight-path trajectory towards their waypoints, not taking any action to avoid collision nor complying with COLREGs. This is left up to AutoNaut. During the simulations there have been made a few assumptions, in order to emphasize the behavior of the control algorithm. These are:

- Vehicle information is always available, both AutoNaut and obstacle.
- Ideal conditions, with no environmental forces present, e.g. current, wind, waves.

7.1 Single Obstacle Avoidance

Head-On Situation

Head-on is one of the first scenarios to be tested as it might be the simplest one. To operate according to the COLREGS the USV simply has execute to an early and clear evasive maneuver to starboard, so that the obstacle is avoided while maintaining a safe distance.

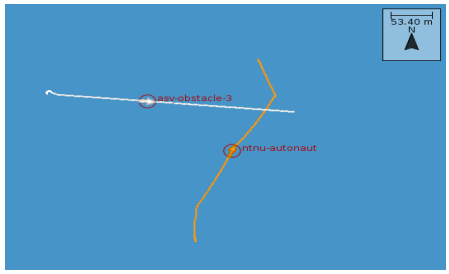


(a) Head-on snapshot (b) Distance between AutoNaut and head-on vessel

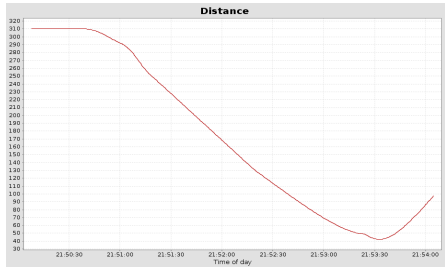
Figure 7.1: Head-on scenario. The USV initiate an early and clear evasive starboard maneuver. After a while the heading offset increases in order to ensure the minimum acceptable distance between the vessels are obtained. When the collision hazard disappears from the linear predicted trajectory the heading offset returns to zero, and the USV passes the obstacle while maintaining a safe distance.

Crossing from Port

In this scenario the USV has the right to stay on and the obstacle is the give-way vessel which normally should take action to avoid collision. However, as the obstacle violates the COLREGS the USV is forced to take action.



(a) Vessel crossing from port snapshot

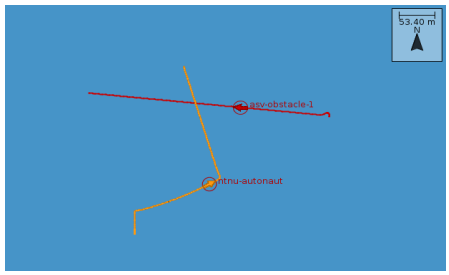


(b) Distance between AutoNaut and vessel crossing from port

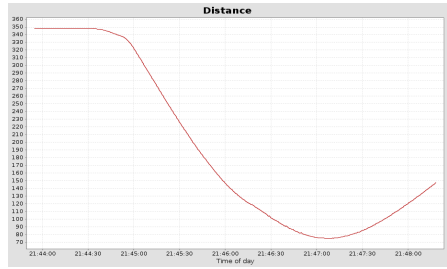
Figure 7.2: Vessel crossing from port scenario. In this scenario the USV passes in front of the obstacle. While passing the USV is not able to maintain the minimum safe distance. As mentioned in Chapter 5 the heading offset is consequently set to the original reference heading, and the course is kept constant until a safe distance is attained.

Crossing from Starboard

In a crossing from starboard situation the USV is the give-way vessel while the obstacle has the right to hold its course. As a result, all the trajectories causing the USV to cross in front of the obstacle are penalized as they violate the COLREGS, leading to trajectories passing behind the obstacle to be cheaper.



(a) Vessel crossing from starboard snapshot

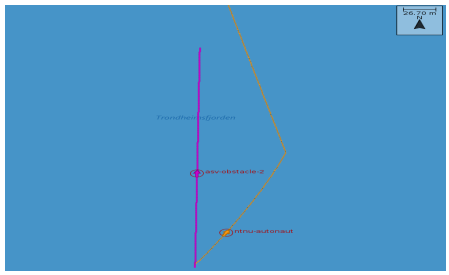


(b) Distance between AutoNaut and vessel crossing from starboard

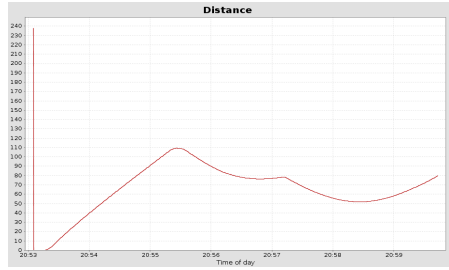
Figure 7.3: Vessel crossing from starboard scenario. The USV clearly indicates its purpose by applying a heading offset of 75°. When the obstacle passes the USV and the COLREGS costs are reduces, leading to the USV passing behind the obstacle with good distance.

Overtaking a vessel

COLREGS states that a vessel being overtaken has the right to stay on, while the vessel doing the overtaking can pass on either side, see Figure 3.7. Since the function χ penalizes port trajectories more than starboard, the USV would normally perform an overtaking on starboard side. This is of course situation dependent.



(a) Overtaking a vessel snapshot



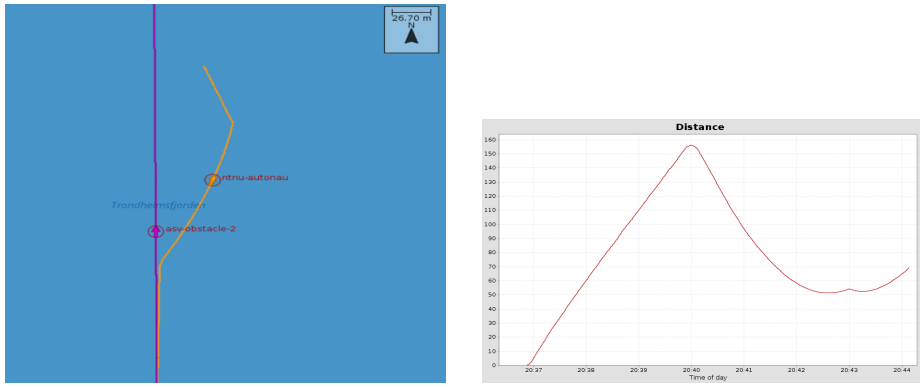
(b) Distance between AutoNaut and overtaking vessel

Figure 7.4: Overtaking a vessel scenario. The USV clearly indicates its intentions and passes the obstacle while keeping a safe distance.

To clarify the Figure 7.4b, the obstacle is started before the USV leading to the increase in distance. Traveling with a greater speed than the obstacle the distance is immediately decreased. Similarly for Figure 7.4a, except the roles are switched.

Overtaken by vessel

In this scenario the USV is the stay-on vessel, but is forced to take action when the obstacle do not show any indication of changing course. The USV changes course to starboard in order to avoid collisions and let the obstacle vessel pass before it again heads towards the waypoint.



(a) Overtaken by vessel snapshot (b) Distance between AutoNaut and overtaken vessel

Figure 7.5: Overtaken by a vessel scenario

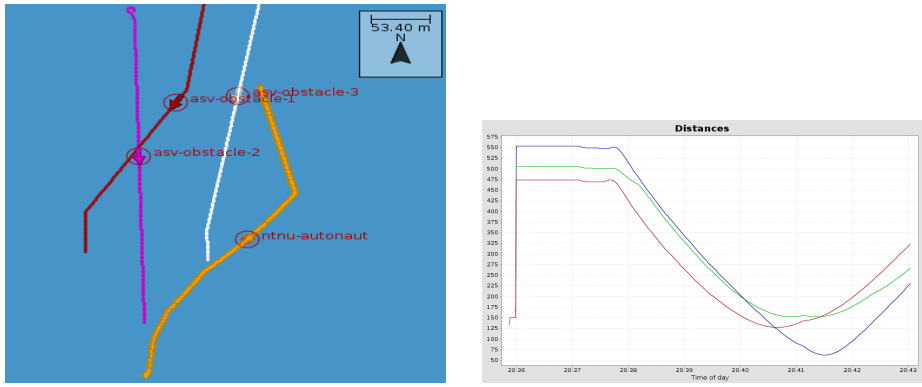
7.2 Multiple Obstacle Avoidance

A situation immediately becomes more complicated when involving multiple obstacles as there are more hazardous situations to take into consideration and there might exists several viable options.

Head-On Situation

With several head-on obstacles incoming the USV performs similarly as with a single

obstacle. An evasive starboard maneuver are initiated to avoid all the obstacles. The new course is kept until the minimum safe distance between the USV, and the obstacles are achieved for the linear predicted trajectory that will bring the USV to the waypoint.



(a) Head-on scenario

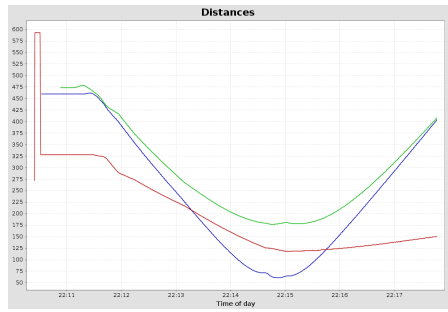
(b) Distance between AutoNaut and head-on vessel

Figure 7.6: Head-on scenario. The USV performs a clear evasive maneuver and stay on a safe distance before resuming its on mission plan.

Head on, Crossing and Overtaking with course change - Part 1

This is a more complex situation. In this scenario the USV faces multiple obstacle vessels creating head-on, crossing from starboard and overtaking situations. Recognizing the head-on and crossing the USV initially intends to pass behind the white obstacle with a heading offset of 90°, however it quickly changes the course to 45°to pass between the red and white obstacle. Just before the red obstacle changes its course the USV corrects its course to 30°, and when the risk related to predicted trajectory to the waypoint is reduced the USV resumes its original path.

(a) Multiple obstacle scenario

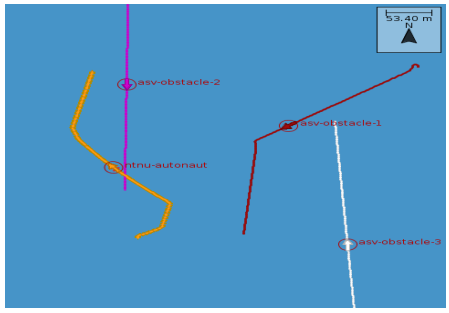


(b) Distance between AutoNaut and vessels

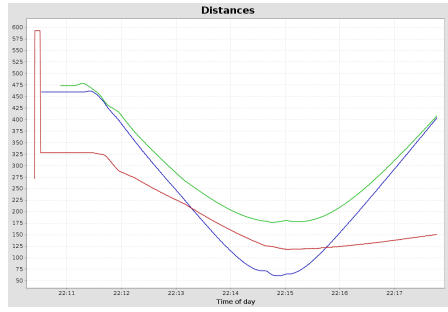
Figure 7.7: Multiple obstacles changing course scenario.

Head on, Crossing and Overtaking with course change - Part 2

Similar as the previous situation, except from the red obstacle which has more South-West direction. After trying to pass between the purple and red obstacle, it concludes the hazard is less extensive if an avoidance manoeuvre to port is performed. In this case the USV concludes that it is better to navigate to port even though it initially is higher penalized, however during the prediction it comes to the conclusion that the other viable option involve an even higher collision risk.



(a) Multiple obstacle scenario



(b) Distance between AutoNaut and vessels

Figure 7.8: Multiple obstacles changing course scenario.

In the last to scenarios most of the viable options require a violation to the COLREGS, except passing behind the white obstacle.

7.3 Special Case Testing

Passing Situation

Figure 7.9 visualize one of the scenarios used to test the performance of the algorithm and the implemented logic. The obstacle has been provided with a set of waypoints that requires clear course changes. As a result of the sudden changes in course the USV reevaluates the situation and act according to the situation it now faces. After performing the last turn the previously mentioned passing error was identified, see Figure 5.1, thus, allowing for this specific error to be corrected.

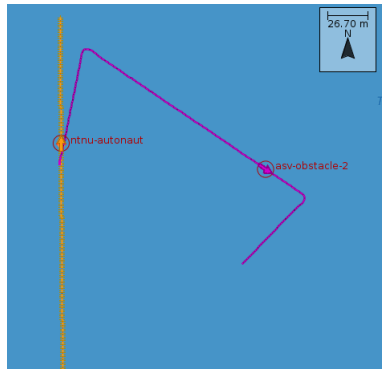


Figure 7.9: Test scenario used to identify the passing error

7.4 Discussion

The overall performance of the collision avoidance is rather good, for both single and multiple obstacle avoidance. The USV is able to safely maneuver passed the obstacle while also obeying the COLREGS, dependent on the situation set up. However, it struggles to keep a safe distance during crossing from port situation if the crossing

vessel decides to not obey the COLREGS. With some tuning this performance can be improved. Considering

The algorithm shows that it can handle multiple obstacles and decides whichever outcome that is associated with the lowest risk, based on the current tuning parameters, and consequently execute that action. Considering the alternatives, only being able to apply heading corrections and the fact that the USV in some situations was "locked in" the algorithm handles the situations good.

Some situations, especially Figure 7.3, provides a clear view of the need to take into the moment to inertia into account. The real application would never be able to achieve the quick course change performed in some of the situations due to the dynamic and limitations of the systems.

Part II

Part 2: Hardware

Chapter 8

Hardware Integration

8.1 System Overview

The hardware platform sensors, main components and the different levels involved are described in Section 2. Figure 8.1 depicts a simple schematic of the hardware setup which is required in order to make Level2 operational. Level1 supply Level2 with sufficient power, 12 V, so that Level2 will be able to operate and power up the sensors connected to the BeagleBone.

The Raymarine AIS650 and Garmin GPS are connected to the BeagleBones USB and serial port. The AIS receiver receives AIS messages from other vessels equipped with AIS transponders. This information will be used to located and track the movements of any vessels that might become a collision threat, while the acquired GPS data ensures fast and accurate positioning and velocity estimates. The Ubiquiti Rocket M2 is connected to the ethernet port of the BeagleBone providing a point-to-point protocol and communication link between the AutoNaut and the operator. DUNE and the CAS software is executed on the BeagleBone, that runs GLUED. Similar to the simulation setup, see Section 6, the desired heading and speed are continually computed in order to guide AutoNaut according to the mission plan. To be able to maneuver AutoNaut and follow a given mission plan, a set of rudder and thruster

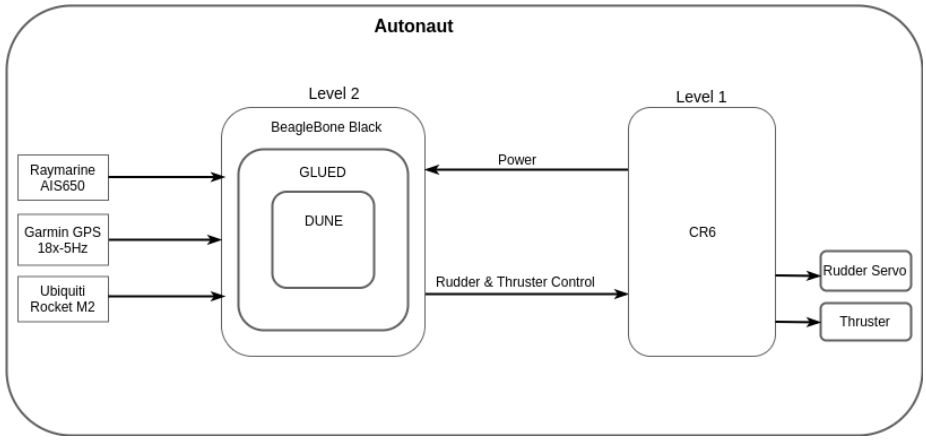


Figure 8.1: Block overview of the Hardware setup

commands needs to be computed and applied to the actual rudder and thruster. In order to achieved this, the desired heading and speed are consumed by the autopilot, containing a heading controller and a speed controller, which computes the necessary rudder command and thruster actuation required to guide the vessel towards a desired waypoint. For these control values to actually be applied to the actual rudder and thruster a communication link between the Level1 and Level2 must be established. Level1 is connected to the Level2 through serial communication. So, whenever the CR6, Level1s computer, requests rudder and thruster control values from the Level2 the Level2 replies Level1 with a set of rudder and thruster values. When received the CR6 apply these new control values to the actual rudder and thruster servo, consequently resulting in a new rudder angle and thrust actuation. In the Section that follows is this setup and sensor integration explained more in details.

8.2 System setup

Figure 8.2 illustrates a more detailed system overview on how the software and hardware components are interconnected in Level2. The different parts will be gone through

in the following subsections.

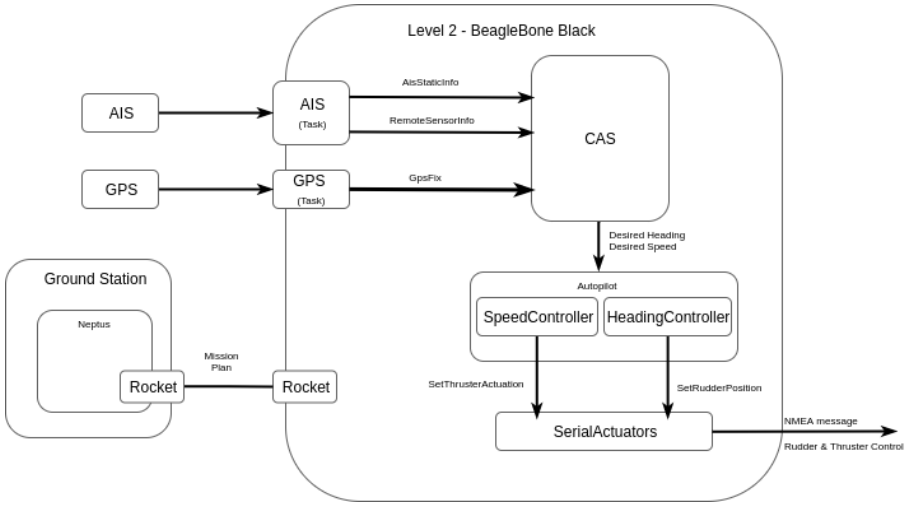


Figure 8.2: System overview of the hardware and software setup

8.2.1 Level2

Figure 8.3, shows the wiring setup of Level2 and the Level2 components.

8.2.2 AIS

The Raymarine AIS650 equipped with a GPS antenna and a VHF antenna, see Fig 2.8, receives continually AIS data from surrounding vessels. These vessel data, containing information such as vessels position, velocity and heading is to be used in a similar manner as the simulated ObstacleState from Chapter 6. The AIS unit requires a 12V DC power source, which was provided from the batteries connected to the Level1. When powered the AIS unit provides AIS data to the BeagleBone via NMEA 0183 connections, made via the units power/data cable. For the BeagleBone to actually receive this NMEA 0183 data, the data cable had be converted into USB format. The data cable consists of 8 signal wires whereas the four wires(green,grey,blue,brown)

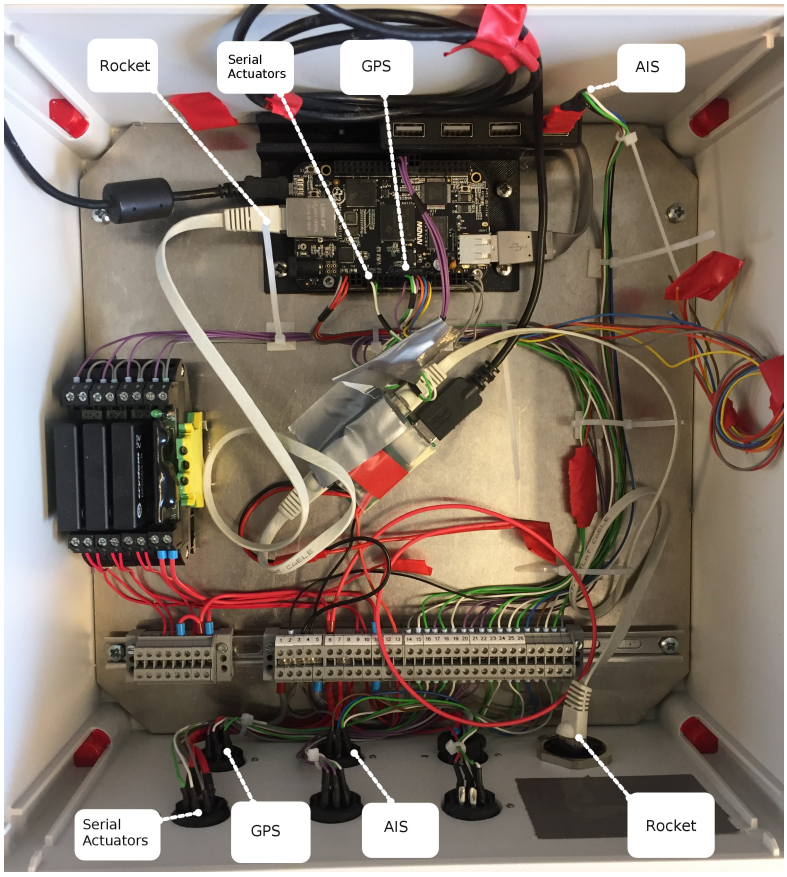


Figure 8.3: Level2 Wiring

are being used to output GPS and AIS data by the units NMEA 0183 (38400 high baud rate) connections. These wires are made into USB format and consequently, plugged into the USB port of the BeagleBone.

To handle the incoming AIS data, DUNE supplies an AIS Task to assist in this data handling. However, in order to get it to work the AIS Task needs to be enabled with the correct baudrate and serial port in the configuration file, see Figure 8.4, and also the NMEA lookup table needs to be activated inside the AIS Task. Now, the AIS Task handles the incoming AIS data, and dispatches the dynamic AIS information as the IMC message RemoteSensorInfo. However, the static information was not handled, so a new IMC message AisStaticInfo was created inside the AIS Task. Whenever static AIS information related a vessel was received, it was stored into the AisStaticInfo and dispatched to the IMC bus.

```
[Sensors.GPS]
Enabled = Hardware
Entity Label = GPS
Serial Port - Device = /dev/ttyO1
Serial Port - Baud Rate = 19200
Sentence Order = GPGGA, GPRMC
Debug Level = Spew

[Sensors.AIS]
Enabled = Hardware
Entity Label = AIS
Serial Port - Device = /dev/ttyUSB0
Serial Port - Baud Rate = 38400
Debug Level = Spew
```

Figure 8.4: Enabling AIS and GPS Tasks

Consequently, both the dynamic and the static AIS data are consumed by the CAS Task where they are handled in different matters. If the distance to the vessel dispatching the dynamic data is inside a distance parameter set by the operator, for instance 5000 meters, the vessel data will be stored and investigated as a possible collision threat, while being outside they are discarded. On the other hand, when a static AIS data arrives it checks whether the MMSI number of the static data is equal to any of the vessels MMSI numbers already stored in the list of potential collision threats. If a

match occur the dimensions of the vessels are updated with its real dimensions, until then the dimensions of a vessel is initialized to resemble a mid-sized oil tanker.

8.2.3 GPS

The Garmin GPS 18x-5Hz LVC and the serial port on the BeagleBone operate on different voltage levels. Thus, MAX3232 voltage converter was used to convert the 5V signals of the GPS to 3.3V signals the BeagleBone serial port supports. The newly converted 3.3V signals are connected to the Rx and Tx pin headers of the UART1. To be able to receive the GPS data a GPS Task needed to be enabled in the same fashion as AIS, see Figure 8.4. The IMC message GpsFix were then dispatched by the GPS Task and later consumed by the CAS Task. Here, the current GPS position (latitude, longitude), speed over ground and heading was set to be AutoNaut new system states.

8.2.4 Autopilot

DUNE did have a autopilot for ASV, however, it had a combined heading and speed controller, namely HeadingAndSpeed. It was designed for a differential drive vessel. Hence, a new Autopilot had to be created, resulting in a HeadingController and a SpeedController. Both controllers consist of a PID controller that continously calculates the difference between the desired heading, or speed, of AutoNaut and the actual heading, or speed. Based on the proportional, integral and derivative gain, if set, corrections are applied to the error value resulting in accurate and responsive control, if applied correctly. After the desired heading and speed are computed by the CAS, these values are consumed by the HeadingController and SpeedController, see Figure 8.2, which creates a rudder angle and a thruster actuation that should be applied to the actual rudder and thruster servo in order to guide AutoNaut towards a desired position with a certain speed. The IMC messages SetRudderPosition and SetThrusterAcuation are dispatched separately by the HeadingController and SpeedController to the IMC bus where they awaits to be consumed by the function responsible for getting these values applied.

In order to get a satisfyingly behaviour of the controllers they need to be tuned

specifically for AutoNaut. To be able to tune the controller real time during the sea trials, there are two options. These PID terms could be real time edited through a System Configuration, Ctrl-Z, inside a Neptus console, or by changing the values inside the configurations file by SSH-protocol.

8.2.5 SerialActuators

To be able to communicate and provide Level1 with the necessary rudder and thruster commands a serial communication link had to be opened between Level1 and Level2. As illustrated in Figure 8.5, the Rx and Tx from the CR6 are connected to the Tx and Rx of UART4 of the BeagleBone, thus, enabling serial communication between the two levels.

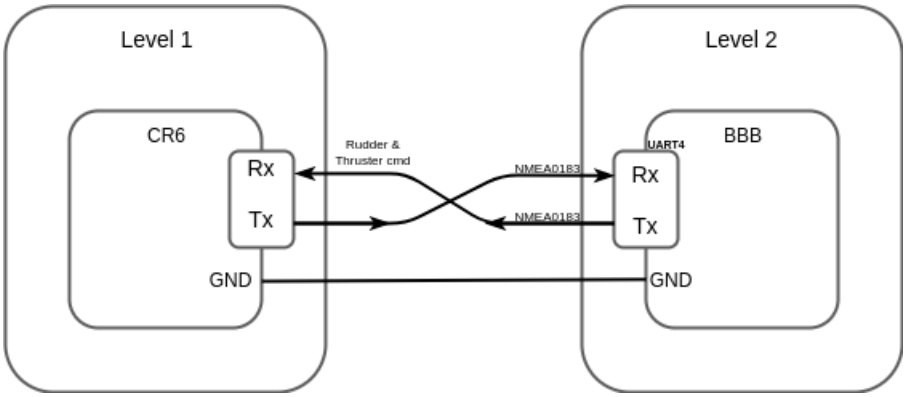


Figure 8.5: Level1 and Level2 Serial Communication

Level1 will once per second transmit a request for rudder and commands to the Level2, in the form of a NMEA message. Whenever this message is being received by Level2 and the SerialAcutors, a Poll sequence is initiated. The Poll sequence would run as long as there is something received on the UART-buffer or until the whole message is received. This is achieved by continuously checking for the character "\n", marking the end of the Level1-NMEA sentence. After receiving the full NMEA sentence, Level2 generates a NMEA sentence reply. The reply contains the rudder and thruster commands just

consumed, and recently computed by the Autopilot. When this message containing control commands have been generated, SerialActuators writes this message to the UART4, essentially the Tx. Consequently, the Rx of Level1 connected to Tx on UART4 would receive this NMEA message of control commands, handle the message and apply the rudder and thruster actuation on the rudder and thruster servo.

The current composition of the NMEA reply is

$$\$BBB01, <1>, <2>*hh\ <CR>\ <LF>$$

where the character $ indicates the start of the NMEA sentence, "BBB01" indicates that it originates from the BeagleBoneBlack. Then a few values follow

<1>	Rudder angle	[-450,450]
<2>	Thruster actuation	[-100,100]

Finally, the checksum "*hh" is for parity checking, the parity bytes (hh) are ASCII representations of the exclusive-or(XOR) sum of all the characters between the "$" and "*" characters. The message is terminated with <CR><LF>, being the carriage return "\r" and line feed "\n"[13]. The content of these NMEA messages can easily be adapted to contain more information if necessary.

The maximum value of the rudder angle can be determined by the operator through the configuration file or in System Configuration. The default is currently at 45°. The rudder angle is represented with three digits, e.g. -358, defining -35.8°towards port side, while positive angle would be towards starboard. The thruster, on the other hand, is defined by percentage, e.g. -50 represents -50% of thruster capability backwards, while 60 is 60% forward propulsion.

Configuration file, also defines the serial port and baud rate used for the serial communication. The baud rate on each level has to be of the same value in order to read sensible values from the other.

Depicted in Figure 8.6, there are the parameters *Enable Thruster Controller* and *Enable Heading Controller*. These parameters allow the operator to real-time enable/disable the different controllers as the operator pleases. By doing so, the operator could for instance test how AutoNaut behaves with only the HeadingController active,

```
[Transports.SerialActuators]
Enabled                                 = Hardware
Entity Label                            = Servo Serial
Serial Port - Device                    = /dev/ttyO4
Serial Port - Baud Rate                 = 9600
Send NMEA                               = True
Maximum Rudder Angle                    = 45
Print Serial message                    = False
Enable Thruster Controller              = True
Enable Heading Controller               = True
User Defined Rudder Angle               = 0
User Defined Thruster Actuation         = 0
Debug Level                             = Spew
```

Figure 8.6: Enabling SerialActuators task in the Configuration file

and tune its behaviour accordingly. The operator is also able to remote control AutoNaut by disabling both controllers and use the *User Defined* parameters to set new rudder and thruster commands.

8.2.6 Rocket

In order to establish a communication link between Neptus and AutoNaut, both need to be on the same subnet. When on the same subnet Ubiquiti Rocket M2 establishes the communication link required for the operator to send mission plans and receive navigational updates from AutoNaut.

Chapter 9

Hardware Testing

Before conducting any sea trials the hardware setup of Level1 and Level2 needed to be tested thoroughly in order to achieve a satisfactory behaviour, both by themselves and when connected together.

The procedure for the hardware testing involved three stages

- Stage 1: Single level testing on the workspace.
- Stage 2: Dual level testing, connecting Level1 and Level2
- Stage 3: Land-based level testing on-board AutoNaut with Level1 and Level2.

Stage 1 - Single Level Testing
Stage 1 involves the development of the Level2 functionality. This should ensure the correct handling of the received GPS and AIS data, both static and dynamic information, and also establish a reliable serial communication for the exchange of rudder and thruster commands with Level1.

Before the interfacing any devices GLUED had to be set up on the BBB, see appendix for the procedure. With GLUED running it was time to get the serial communication up and running. With not too much hardware experience and knowing that debugging hardware is possibly very time consuming a step-wise and rather systematic approach was chosen during testing and hardware integration. The steps for the serial

communication was

1. Ensure the UARTs are enabled
2. Test serial communication between one, and then two UARTs, with keyboard interrupts
3. Send NMEA messages from SerialActuators over serial to itself.
4. Make SerialActuators ready to communicate with Level1.

In the NTNU version of GLUED there has already been included a script enabling the UART1, 2 and 4. So, in order to verify that the serial ports works Tx and Rx were joined of a port, and the characters entered from the keyboard was displayed on the microcom for the given serial port, e.g. tty01 being the UART1. Similarly, the Tx and Rx of two serial ports where connected, P9_21 to P9_26 and P9_22 to P9_24, and microcoms for each serial port opened which enabled the characters pressed in one to appear in the other one, vice versa. The same approaches were used when trying to send NMEA messages from SerialActuators, to see if the NMEA handling and serial communication worked as intended. This was tested by connecting to two serial ports, they both had to be defined in the configuration file just as depicted in Figure 8.5, and create two NMEA messages with and unique code, e.g. "BBB01" and "CR6". The idea was to send a CR6-NMEA message when the Task was initialized to for instance UART1, the message would be received. If the message code was "CR6" an reply would be generated with BBB01 as the message code. This communication cycle should occur every 4 seconds. However, it did not. Eventually, it showed there was an issue with the Poll implementation, described in the previous chapter, and how it was used. When correctly used, the serial communication between the ports worked as expected.

Meanwhile, the Garmin GPS and a power supply dedicated to provide the device with 5V was arranged. A similar systematic approach were used when interfacing the devices, the GPS and the AIS.

1. Find the set up requirements in the device documentation
2. Figure out how to interface to the BBB.
3. Verify that the serial port receives device.
4. Apply code that handles the necessary device information.

From the GPS documentation, the serial output levels was 4-5.5V, meaning the

voltage had to be transformed before being applied to the pins of the BBB that supports 3.3V[13]. This was achieved by a transceiver MAX3232, which enables true RS-232 performance from a 3-5.5V operating range. The device was powered with 5V from the power supply, while the white Transmit Data and green Receive Data were connected to the MAX3232.

The 5V signal wires, Transmit Data (white), and Receive Data (green) were transformed to operate at 3.3V by the MAX3232 and then connected the pins Rx (P9_26), and Tx(P9_24) of UART1 on the BBB. In order to be converted to 3.3V a reference power from the BBB pin P9_03, stating +3.3V, needed to be connected to the MAX3232, additionally to the ground from e.g. P9_01. Finally, the power supply should provide 5V to the device for activating it. When powered it transmits the newly acquired data, which was visualized through microcom. For real-time data acquisition in DUNE, the code segment depicted in Figure 8.4 stating the serial port and baudrate were added to the configuration file. As stated in the previous chapter, the newly acquired device data were handled in the CAS Task, being set to be the no longer simulated system states of AutoNaut.

Shortly after, the Raymarine AIS650 were picked up. The AIS was interfaced in the same manner as the GPS. The device has eight signal wires, described in Section 8.2. The relevant wires already been made into a USB RS422 dongle only required to be plugged in the USB port and 12V from the power supply to activate the device, additionally to ground. Through the microcom device information were displayed and the data acquisition and data handling logic in DUNE were initiated.

Finished interfacing the devices to the BBB it was time for stage 2 - testing Level1 and Level2 together.

Stage 2

After the plugs of the sensors had been fitted and the wiring of the Level1 and Level2 was completed it was time to test the Level1 and Level2 together.

The purpose of the testing for Level2 was to

1. Create a reliable communication link between the subsystems.
2. Find/fix hardware and software related bugs.

3. Prepare the subsystem for the sea trials.

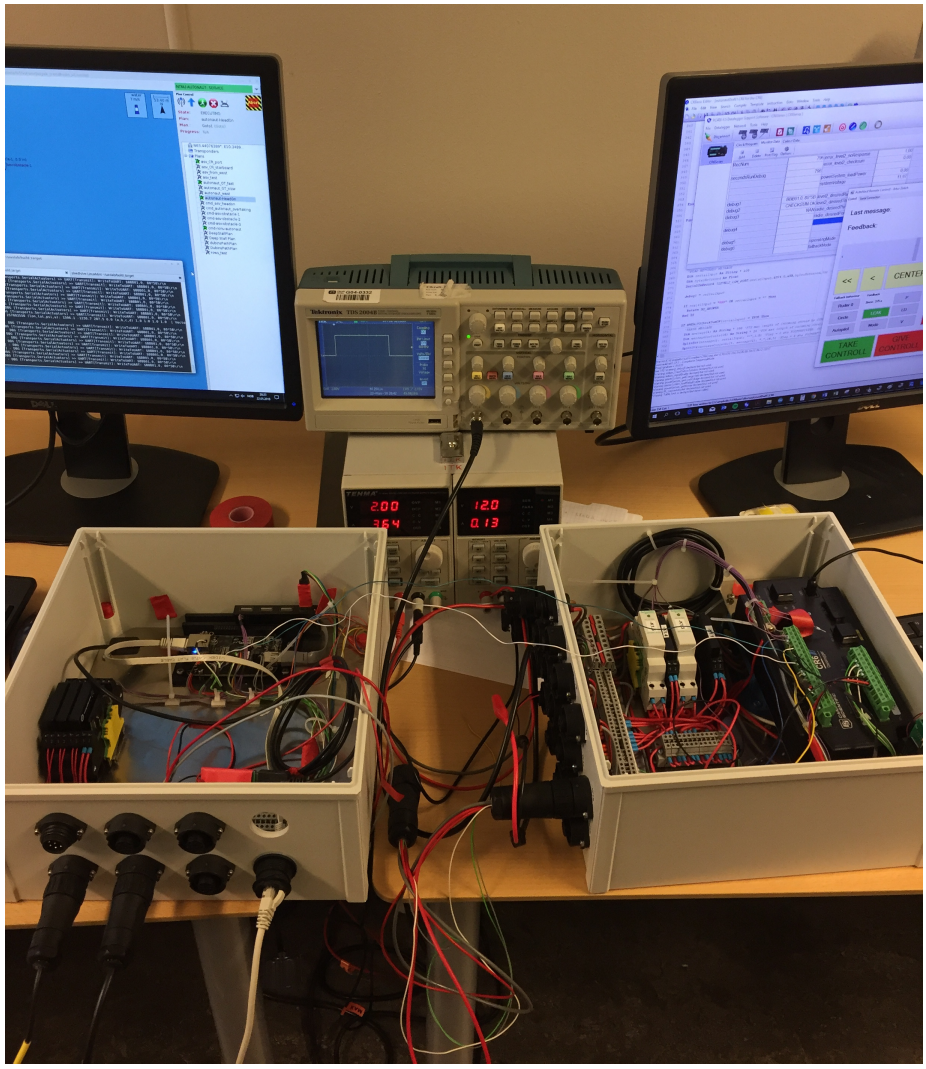


Figure 9.1: Hardware testing of subsystems

Figure 9.1 illustrates the typical subsystem test set up.

The components required for the interconnected subsystemtest were

1. Subsystems (Level1 & Level2) with active computers (BBB and CR6)
2. Plug to interconnect the subsystems
3. An established Wi-Fi connection to Level2, provided by the Rockets
4. Power supply
5. Visualization tool of the input and output of the subsystems
6. Level2 sensors: GPS, AIS (Not required but prefered)

The power supply provided Level1 power and when the subsystems were interconnected by the Power-And-Serial-Communication plug Level2 was also powered up. This also resulted in the Level2 sensors and the Rocket to be activated. When on the same network a communication link was created between the Level2 Rocket and the Rocket at the Ground station, as depicted in Figure 6.1. Then the operator could access the BBB using SSH, transfer the latest DUNE software, including the newest SerialActuators Task, and run it to see the performance. With running software on both "onboard" computers, BBB and CR6, CR6 would send a NMEA message and the BBB should send a NMEA reply, thus, testing the performance of the serial communication. The GPS and AIS sensors were connected to Level2, as mentioned in the Subsection 2.3.2. It was preferred to have the devices interfacing with the subsystem to keep an eye on the total power consumption and also since they were a requirement for the final system.

Note that in this current set up, Figure 9.1, the Power-and-Serial-Communication plug has not been used. Normally, the plug would be connected from Level1 to the W15 plug, providing Level2 with power. However, the reason it was not applied here was that the Transmit/Receive Data wire in the Power-and-Serial-Communication plug was switched resulting in no data acquisition. So instead regular wires were used to establish serial communication between the subsystems and to power up Level2.

Stage 3

A few days before the planned commissioning the Land-based test was conducted.

One part of AutoNaut's hull were lifted up on a trolley, and the subsystems with their sensors along with a battery to provide power were fitted into one of the hull compartments. The idea was to send mission plans to AutoNaut from a temporary ground station and depending on the rudder output push the cart around a parking lot, located just outside the office. The goal of the land-based test was to test

- The Autopilot, mainly the HeadingController
- The rudder response
- The serial communication on the actual system

Figure 9.2: Hardware testing of Level1 and Level2 with the rudder and thruster, prior the land-based test

During the test some problems arised, and there were some software preparation that were essential to be fixed before the commissioning. They were

1. Serial Communication
    - "No proper checksum found" or "Checksum mismatch".
2. Neptus preparations
    - How to change parameters real-time
    - Create real-time printing functionality which could be enabled or disabled

Even though the serial communication worked in the office over an extent period of time it did not necessarily mean it worked when applied in the field. This was also one of the reasons the land-based test was conducted, to see what worked and what had to be prioritized to be fixed in order to operate AutoNaut on the actual field test.

Part III

Part 3: Results

Chapter 10

Sea Trails

The sea trials was to be conducted at Trondheim Biological Station (TBS). TBS is NTNU's centre for marine biological research, nicely situated directly at the Trondheimsfjord, A pier in front of the station is equipped with a crane and the necessary equipment required to get AutoNaut into the water. The sea trials was composed of two days.

10.1 Day One

The itinerary of day one of the sea trials was mainly to transport AutoNaut from NTNU to the TBS and put it together to prepare it for the actual testing conducted on day two. The transportation and AutoNaut assembly procedure took approximately a day. Shortly summarized, the procedure involved

1. Connecting the back and front hull.
2. Fit the different sensors to the hull, e.g. ADCP, CTD.
3. Put the subsystems into the compartments and interconnect them.
4. Connect Level1 to pumps and the rudder and thruster servo-controller.
5. Make sure that every step of the installation and wiring during the assembly is waterproof.

Figure 10.1: Assembling AutoNaut and fitting the sensors

After the assembling and installation of AutoNaut was complete, AutoNaut was pushed over to the pier in order to undergo a float- and leak-test. A crane developed for heavy lifting, operated by an authorized employee at TBS, were used to lower AutoNaut into the water. As illustrated in Figure 10.2 and Figure 10.3, straps were fastened on hooks located at the bow and stern of AutoNaut, and by using guiding ropes to keep it from rocking while it was lowered. Consequently, by sending different control values the rudder and thruster were tested. AutoNaut was kept in the water for some time before bringing it back up to the pier. It had stayed afloat and the inside was dry. Finally, AutoNaut was hosed down with freshwater before being stored in a storage room at TBS, ready for day two.

Meanwhile the subsystems were brought back to the office for some further debug-

ging. The same issue with the serial communication had occurred during the rudder and thruster tests, so before Level2 could be tested on day two, this had to be resolved. The issue became apparent when applying the DUNE-function NMEAREADER, so the temporary solution was a minor redesign of the message handling logic inside SerialActuators. The code of the received NMEA message, e.g. "CR6", was checked and if it matched the anticipated code a reply was generated. So the Task would now whenever receiving a NMEA sentence it replies with the set of rudder and thruster commnads, as described in section 8.2.

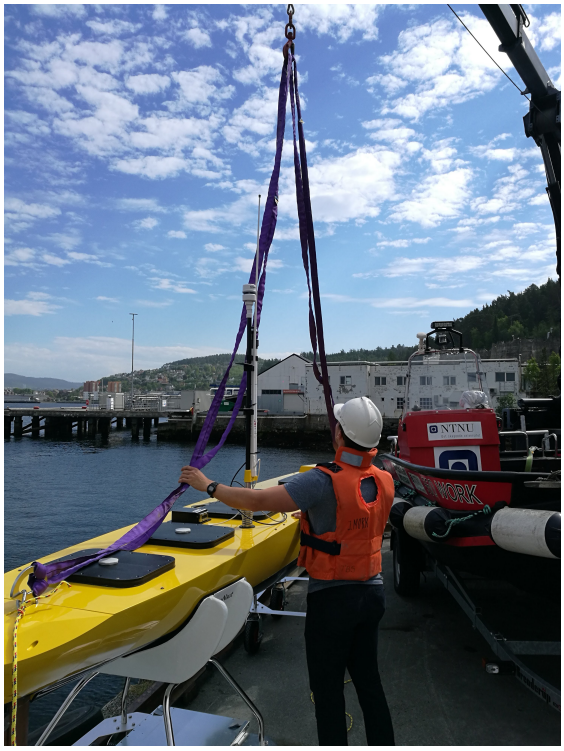


Figure 10.2: The procedure of moving AutoNaut in/out of the water.

Figure 10.3: Performing rudder, thruster and leakage tests

10.2 Day Two

The main focus of day two was to make AutoNaut operative, which involves making both the subsystems. Consequently, the focus during the Level2 testing was to:

- Establish a reliable serial communications between AutoNaut and the ground station.

- Test the Autopilot, both the heading and the speed controller.
- Test the real-time parameter changing functionality.
- Test the collision avoidance system.

To clarify, the main focus on the sea trials have not been to get the collision avoidance up and running. The priority has been to establish a reliable communication between the subsystems and an operational autopilot, which both are mandatory requirements before even running the collision avoidance algorithm.

After arranging an escort boat and connecting the subsystems, the final day of testing was ready to be initiated. A ground station was set up on the pier, equipped with a laptop running Neptus, a handheld VHF radio and a clear view of the test area. From the ground station the operator could monitor and send waypoints and control commands to the AutoNaut (Level2).

Figure 10.4: The ground station, CC unit, with the view of the test area.

The schedule for the sea trails was fairly simple. First test Level1, then Level2.

1. Level1:
    - Autopilot, Fallback system, Remote Control, Pump test, etc.
2. Level2:
    - Serial communication
    - The autopilot. HeadingController with/without the SpeedController.
    - Remote control (autopilot is turned off).
    - Real-time parameter corrections from the Ground Station.
    - Collision avoidance, encountering simulated obstacles.

After completing the Level1 testing. Level2 testing was initiated. First, the Remote Control were tested by disabling both of the controllers. Further the controllers were enabled and disabled in turn testing each case. Meanwhile, by alternating remote control and different controllers, different rudder angles and thruster actuation commands were sent from the operator. This was achieved through Neptus and System Configurations where the parameters could be changed real-time. During these minor tests a small issue came forth. When the operator defined and sent a rudder command with a value outside the range -128 to 127 ($\pm 12.7^{\circ}$) a warning was displayed. However, this warning was not critical, it only limited the remote control range. Apparently, the parameter had accidentally been initialized as a signed 8-bit integer,$int8t$. This was kept in mind during the rest of the testing.

Second, waypoints were generated at the Ground Station and sent to AutoNaut for path control. The heading controller was continuously computing new rudder commands, while the speed controller was occasionally disabled in order to see how the heading controller behaved when only using wave propulsion. These control commands was still applied once per second to the rudder and thruster servo, due to the Level1 design. While maneuvering towards a given waypoint the controller parameters, Kp, Ki and Kd, could just as easily as the rudder angles be updated with new values.

Figure 10.5: The escort boat observing AutoNaut's performance. [Courtesy of Bendik Agdal].

As the priority had been to ensure a reliable serial communication, real-time corrections and due to limited time, the preparations for a collision avoidance test had not been completed. However, with some improvisation, attempts were made. Not having the necessary DUNE code on the laptop, both DUNE instances, AutoNaut and the

simulated vessel had to be executed simultaneously on the BBB. As a result of both instances running an error occurred after a couple of seconds, stating that the CPU was too high. Which lead to the termination of one of the DUNE instances. Despite several attempts trying to resolve this, it was decided to conclude the testing.

Optimally, if there had been more test days, DUNE instances of one or more simulated vessels should have been executed on the laptop instead of the BBB. This would probably resolve the high CPU error that occurred, and then the performance of the collision avoidance could be studied.

After concluding the sea trials AutoNaut was cleaned and put back into the storage room while the subsystems were brought back to review the mission and the logged data.

Figure 10.6: AutoNaut maneuvering towards a waypoint.

10.3 Results

During the sea trial different waypoints were sent to AutoNaut in order to study the onboard autopilot. This data analysis has been done post mission execution.

Waypoint - South

The first waypoint transmitted was situated 240 meters south of AutoNaut, illustrated in Figure 10.7. The direction of the waves were also directed south. At this point the heading and speed controller were both active. After 3 minutes, completing a turn and continuing more or less straight, the speed controller was disabled, hence, only the thrust available was achieved through wave propulsion. During the last 160 meters to the waypoint it oscillates before reaching the waypoint.

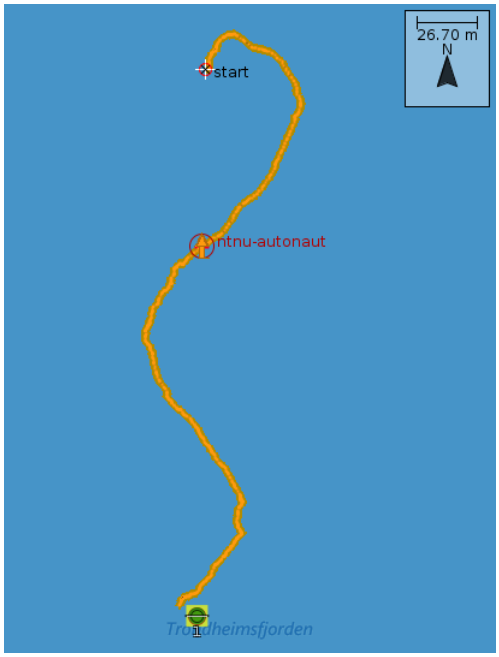


Figure 10.7: AutoNaut maneuvering towards a waypoint position South.

Waypoint - North-West

In Figure 10.8 a waypoint located 270 meter North-West has been received. However, for the first part of the path AutoNaut was controlled remotely. It was not until the location of the ntnu-autonaut marker in Figure 10.8 that the heading controller became in charge of the control. The speed controller was still kept disabled. Again, the oscillation started to occur and the USV struggled to gain height. The oscillations continued in the same way after enabling the speed controller. The Kp value was eventually adjusted from being 1.5 to 0.15. As result of the tuning, the major oscillations ceased and AutoNaut adjusted its course towards the waypoint. Eventually the USV started to drift with the current, away from the waypoint, while not being able to reach the reference value. Therefore, the Kp values was increased again to 1.0 in order to correct the course, however lacking a Kd-term to dampen the oscillations, the oscillations continued.

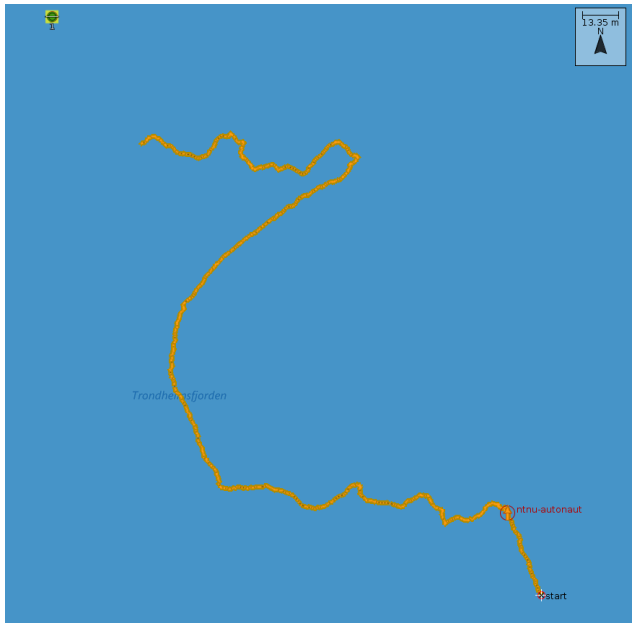


Figure 10.8: AutoNaut maneuvering towards a waypoint position North.

Waypoint - North-East

Lastly, AutoNaut received a waypoint positioned 215 meters North-East. The USV maneuvered fairly straight when the Kp was 0.3, but struggled to gain height additionally to overshooting the waypoint. When the Kp was increased to 0.8 it corrected itself more towards the waypoint but again some overshoot occurred at the same time as reaching the waypoint.

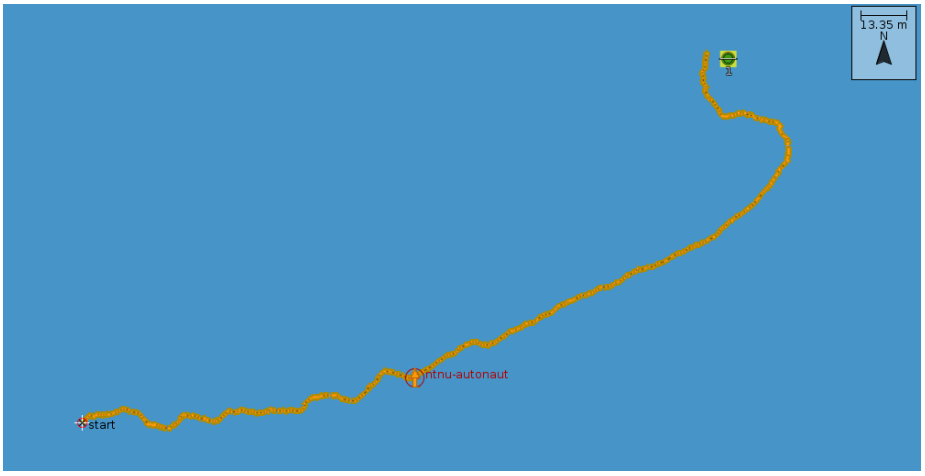


Figure 10.9: AutoNaut maneuvering towards a waypoint position East.

The rudder performance is illustrated in Figure 10.11, along with the DesiredHeading in Figure 10.10. The figures clearly show that the proportional gain does not work well as the rudder angle overshoots after reaching the desired heading. The oscillations occurs as a consequence of the mass and inertia stored in the system when reaching the desired heading. To dampen the oscillations a derivative gain should be tuned to apply the right correction to stop AutoNaut at just the right time. Furthermore, an integral gain would assist in keeping the correct heading by minimizing the steady state error.

To summarize the sea trials, it is apparent that a P-controller is not sufficient to control the vessel, due to the many environmental disturbances present and the mass

and moment of inertia of AutoNaut. Despite not adding a Ki and Kd term to the controller, which should have been done, the test shows that the real time adjustments of the PID controller is functioning, and it is obvious that more tuning of the PID controller is required to obtain a stable controller performance. On the other hand, the serial communication worked much better than the previous day, not getting any issues related to the subsystem communication or the NMEA message handling. Similarly, the real-time adjustments, allowing the operator to easily make corrections, and the data logging worked really well. Even though the collision avoidance test was not accomplished due to CPU limitations, a brief output during the few operative seconds stated that the collision avoidance was running and a corrective action had been made. Thorough preparations are essential and the method of generating simulated vessels has to be in order before conducting new collision avoidance tests.

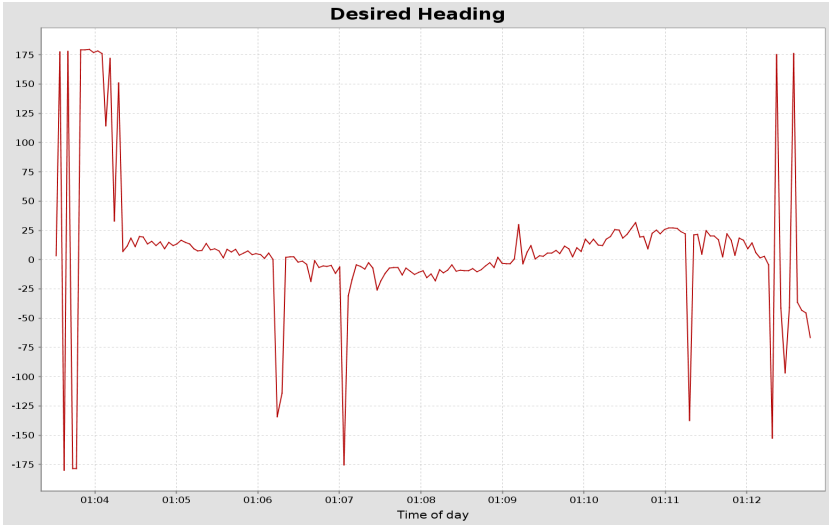


Figure 10.10: Waypoint South: Desired heading of AutoNaut.

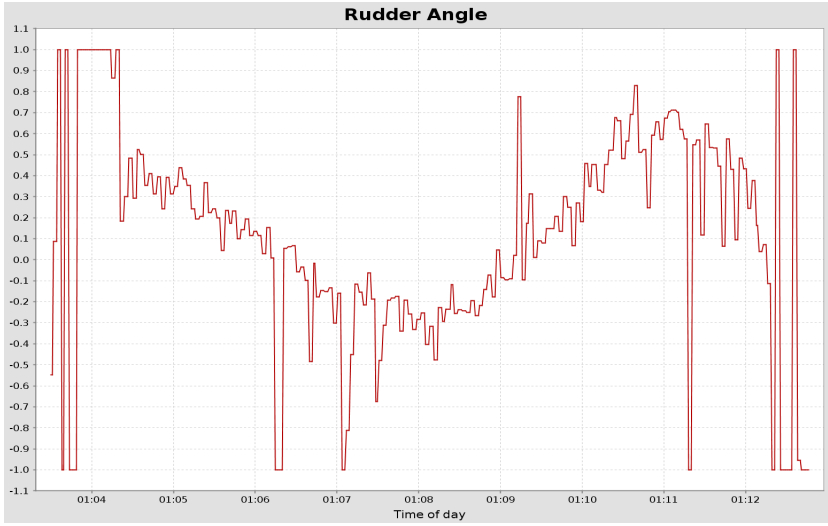


Figure 10.11: Waypoint South: Rudder angle of AutoNaut.

Chapter 11

Further Work

In this section some recommendations for the further development of the collision avoidance system, simulation and hardware implementation will be mentioned. Points relevant for the future development of the CAS and simulator performance are:

- Moment of inertia. Consider the inertia during a turn when predicting possible trajectories.
- USV simulator model. A more realistic vessel model is required in DUNE in order to achieve of a more realistic simulated behaviour.
- Add Weather scenarios. Environmental forces, e.g. waves, wind, current, should be considered as they affect the performance of AutoNaut, see Table 2.1.
- Auxiliary thrust. Add logic that determine when to apply the auxiliary thrust, whether to assist in collision avoidance or during flat calm conditions.
- Integration of new sensors, e.g. ADCP, CTD, and the necessary software for on-board processing and data collection.

It is apparent that there are some issues that needs to be resolved before conducting a new sea trial. They are:

- Improve and tune the autopilot performance.
- Improve and further develop a robust hardware test platform for Level2.
- Organize obstacle vessels and conduct experimental testing of the CAS.

Chapter 12

Conclusions

In this thesis a COLREGS compliant collision avoidance system has been fully integrated with the LSTS software toolchain and implemented in the *Navigation and Collision Avoidance*-subsystem of AutoNaut. The CAS has been tested through various simulation scenarios to study and further develop the performance of the system. The system avoided the simulated obstacles while adhering to the COLREGS, and the overall performance was quite well.

A hardware design for Level2 of AutoNaut were also implemented, composed of an autopilot, serial communication between subsystems, various hardware sensors and the CAS.

Two days of sea trials were conducted to test the performance of the subsystems and AutoNaut. The subsystems proved themselves capable of controlling the USV, and the AutoNaut maneuvred towards waypoints provided by the ground station. The autopilot were successful in guiding the USV. However, the results highlights the need of additional tuning have to prioritized in future tests, along with obtaining the necessary obstacle vessels required to test of the CAS.

A systematic method for the operator to perform real-time parameter tuning of the collision avoidance and autopilot has been implemented and tested. Similarly, the functionality of being able to remote control both the heading and speed controller.

This worked really well during the testing and provided the operator with high level of control of the USV.

Appendix A

Appendix

A.1 Thesis schedule - Gantt diagram

The Gantt diagram provides an overview of the progression of the work, see Figure A.1. The Collision Avoidance development was highly prioritized in the beginning in order to obtain a COLREG compliant behaviour in the simulations. From early May, after receiving the hardware components the focus was primarily on getting the subsystems up and running to be able to conduct the sea trials.

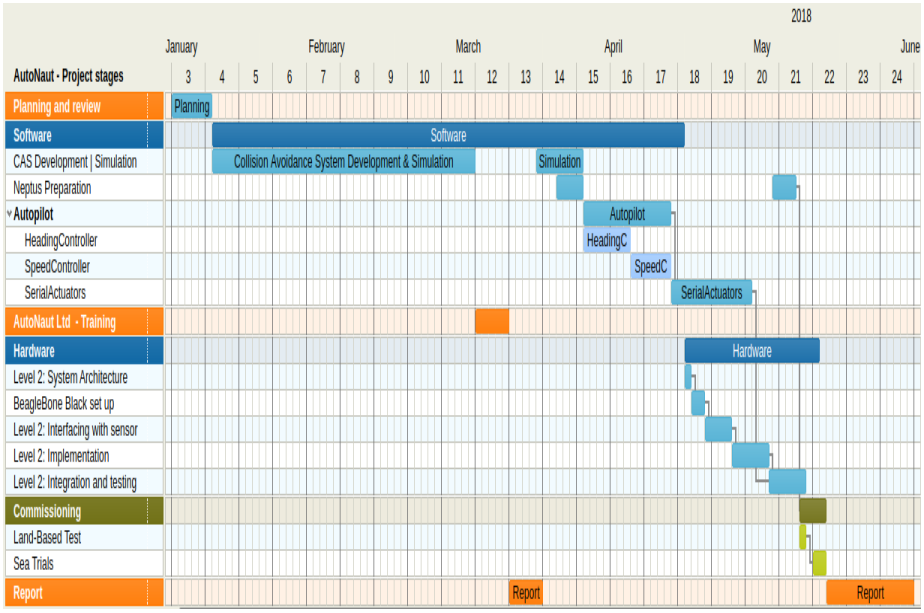


Figure A.1: Gantt diagram of the project stages

References

[1] Ship officers home of marine navigation officers. Accessed: 05-12-2017.

[2] B. O. Agdal. Design of control system for green usv. 2018.

[3] K. G. ASA. Yara and kongsberg enter into partnership to build world's first autonomous and zero emissions ship. Accessed: 11-05-2018.

[4] AutoNaut. Autonaut usv - a new kind of unmanned surface vessel. Accessed: 10-11-2017.

[5] BeagleBone. Beaglebone black. Accessed: 02-06-2018.

[6] E. Bøckmann and S. Steen. Experiments with actively pitch-controlled and spring-loaded oscillating foils. pages 227–235, 2014.

[7] Bowker, Tan, Townsend, and Shenoi. Experimental analysis of submerged flapping foils implications for autonomous surface vehicles (asvs). *IEEE Transactions on Intelligent Transportation Systems*, 2016.

[8] E. F. Brekke. Autonomous shuttle ferry in trondheim. Accessed: 20-12-2017.

[9] COP23. Cop23 in bonn: Setting the rules of the climate game. Accessed: 11-05-2018.

[10] P. Dias and R. Martins. An open source software suite for air and ocean vehicles, 2014.

[11] P. S. Dias et al. Mission planning and specification in the neptus framework. 2006.

[12] T. Economist. Autonomous-vehicle technology is advancing ever faster. Accessed: 10-06-2018.

[13] I. Garmin International. Gps 18x technical specifications. *190-00879-08 Rev B number: 87140-5*, 2008.

[14] Greenship. Ambition 1.5 degree: Global shippings action plan. Accessed: 11-05-2018.

[15] I. B. Hagen. Collision avoidance for asvs using model predictive control. pages 1–74, 2017.

[16] IMO. Colregs - international regulations for preventing collisions at sea. Accessed: 05-12-2017.

[17] IMO. Colregs - preventing collisions at sea. Accessed: 05-12-2017.

[18] T. A. Johansen, T. Perez, and A. Cristofaro. Ship collision avoidance and colregs compliance using simulation-based control behavior selection with predictive hazard assessment. *IEEE Transactions on Intelligent Transportation Systems*, pages 3407–3422, 2016.

[19] P. Johnston and C. Pierpoint. Deployment of a passive acoustic monitoring (pam) array from the autonaut wave-propelled unmanned surface vessel. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[20] P. Johnston and M. Poole. Marine surveillance capabilities of the autonaut wave-propelled usv. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[21] Konsgberg. World's first official test bed for autonomous shipping opens in norwa. Accessed: 20-12-2017.

[22] LSTS. Glued. Accessed: 16-04-2018.

[23] LSTS. Imc. Accessed: 16-04-2018.

[24] LSTS. Our toolchain. Accessed: 20-10-2017.

[25] R. U. Ltd. Ais350/650 installation instructions. *Document number: 87140-5*, 2014.

[26] R. Martins et al. Imc: A communication protocol for networked vehicles and sensors. *IEEE OCEANS 2009 - EUROPE*, 2009.

[27] S. Nagata, Y. Imai, and K. Toyota. Thrust generation by waves. *The International Society of Offshore and Polar Engineers*, 2010.

[28] U. Networks. Rocket m. *Datasheet*, 2011.

[29] H. S. News. The future of shipping is autonomous. Accessed: 20-12-2017.

[30] J. Pinto, P. Calado, et al. Implementation of a control architecture for networked vehicle systems. *IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles (NGCUV'2012)*, 2012.

[31] J. Pinto, P. S. Dias, et al. Neptus – a framework to support the mission life cycle. 2009.

[32] J. Pinto, P. S. Dias, et al. The lsts toolchain for networked vehicle systems. *IEEE OCEANS - Bergen*, 2013.

[33] A. Skredderberget. The first ever zero emission, autonomous ship. Accessed: 11-05-2018.

[34] T. Stenersen. Guidance system for autonomous surface vehicles. 2015.

[35] U. S. Technology. *Master Mariner - AutoNaut's wave-propelled USV*. February/March 2017.

[36] H. F. Willan. Autonomy at sea - the future? Accessed: 20-12-2017.