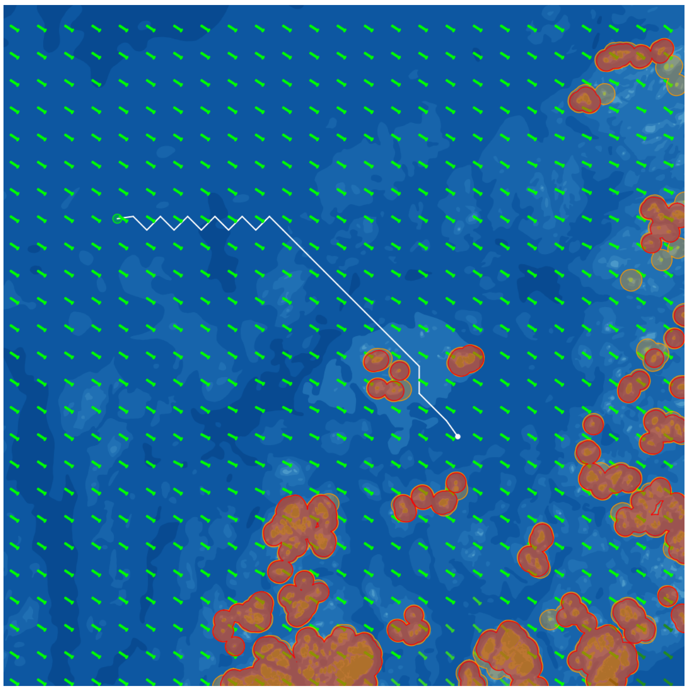
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Bendik Åshaug Holm

Path planning for wave-powered unmanned surface vehicle based on electronic navigational charts and weather data

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Arne Johansen
July 2022



Bendik Åshaug Holm

Path planning for wave-powered unmanned surface vehicle based on electronic navigational charts and weather data

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Arne Johansen
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Abstract

The work documented in this thesis details the development and implementation of a path planning plan for a wave-powered Unmanned Surface Vessel (USV). The USV, known as NTNU AutoNaut, is part of a scientific project hosted by the Norwegian University of Science and Technology (NTNU).

Previous work related to path planning for the AutoNaut was conducted in the fall semester of 2021 in the form of a project thesis. This work used Electronic Navigational Chart (ENC)s with geodatabase data to create paths to avoid grounding of the AutoNaut. A grid-based approach was selected, and the sea chart was divided into distinct nodes. An A* algorithm was implemented to create paths which avoided land and shallows. This algorithm, aided by a euclidean heuristic, would produce way-points from a starting point to a goal point on the map. The system showed promising results, and would produce a list of way-points given the existence of a feasible path.

This thesis expands upon the project thesis. While the project thesis would produce paths solely based on geography and ocean depth, this work aims to take weather data into account when generating paths. Weather forecasts were collected from the The Norwegian Meteorological Institute (MET) Application Programming Interface (API), which was selected for its coverage area, availability and wide selection of data. Forecasts related to wind, waves and current were downloaded, as these forecasts were determined to be the most relevant for path generation. Collection and plotting tools for weather data were created for the SeaCharts API, and these might be integrated into the tool in future work.

A cost function based on the weather data was created with emphasis on safety. The A* algorithm was modified to incorporate this cost function, and movement costs were weighted according to normal and diagonal movement, as well as the angle of attack and magnitude of the weather forces. These costs were then tuned using simulated data. Results appear promising, but the attempts to tune the system were inconclusive. Sensor data from the AutoNaut itself was not gathered due to time constraints and availability. A proper plan for testing and tuning needs to be executed for the path planner to work properly in the real ocean environment.

In addition to the collection of weather forecasts, efforts were made to limit the number of hazardous situations the AutoNaut might find itself in. Buffer zones were created around land and shallows. These were seen as obstacles by the path planner to decrease the chances of grounding. A minimum safe distance to land

to allow for operator intervention was also created.

Furthermore, the paths generated by the path planning algorithm underwent some post-processing. In order to both limit the amount of information needed to be sent to the AutoNaut, as well as decrease the amount of manual inputs needed to start a mission, two algorithms for reducing the amount of waypoints were developed. Two potential "path pruning" algorithms were created and tested, one based on pure skipping of waypoints, and one based on recursive segmentation of the path. In the end, the latter was chosen as it scaled better and was more flexible.

Sammendrag

Denne masteroppgaven gjør rede for utviklingen og implementasjonen av en stifinner for en USV som drives av bølgekraft. USVen, NTNU AutoNaut, er en del av et forskningsprosjekt på NTNU i Trondheim.

Stifinning for AutoNaut ble gjennomført på høstsemesteret i 2021 i form av en prosjektoppgave. Dette prosjektet brukte elektroniske sjøkart (ENCer) med geodatabase-data for å lage ruter AutoNauten kunne følge uten å gå på grunn. En rutenettbasert tilnærming ble valgt, og sjøkartet ble delt opp i noder. Søkealgoritmen A* ble ble brukt for å skape stier som unngikk land og grunner. Denne algoritmen, hjulpet av en euklidisk heuristikk, klarte å produsere en sti fra et startpunkt til et endepunkt på kartet. Systemet fungerte, og produserte waypoints gitt at en gjennomførbar rute eksisterte.

Denne oppgaven utvider på prosjektoppgaven. Mens prosjektoppgaven kun produserte ruter basert på geografi og havdybde, sikter denne masteroppgaven på å også ta hensyn til værdata. Værvarsler ble hentet fra et API utviklet av Meteorologisk Institutt. Dette ble valgt på grunn av både dekningsområde, tilgjengelighet og pris. Seks datapunkter ble hentet inn for denne oppgaven, relatert til vind, bølger og havstrøm, da disse ble ansett for å være mest relevant for ruteplanlegging til havs. Verktøy for innsamling og plotting av værdata ble utviklet for SeaCharts APIet, og disse verktøyene vil kanskje innlemmes i SeaCharts i fremtiden.

En kostnadsfunksjon basert på værdata ble laget med spesielt hensyn på sikkerhet. A*-algoritmen ble modifisert til å inkludere denne kostnadsfunksjonen, og bevegelseskostnader ble vektlagt med hensyn på normal og diagonal bevegelse, såvell som angrepsvinkel og styrke på værkreftene. Kostnadene ble tunet ved hjelp av selvgenerert dummy-data. Resultatene virket lovende, og de resulterende rutene så ut til å vinkle AutoNauten riktig i forhold til værkreftene. Likevel er det nødvendig med ekte sjøtester for å validere systemet tilstrekkelig. Ekte sensordata fra AutoNauten ble ikke hentet inn, både på grunn av tidsbegrensinger og begrenset tilgjengelighet. En gjennomtenkt plan for testing og tuning må gjennomføres for at stifinneren skal virke ordentlig i det ekte havrommet.

Det har også blitt jobbet med å begrense utvalgte faremomenter for AutoNauten. Buffersoner ble generert rundt land og grunner. Disse ble ansett som hindringer av A*-algoritmen, for å begrense sjansene for grunnstøting. En minimum trygg avstand til land ble også funnet.

Videre ble rutene generert av stifinneren utsatt for postprosessering. For å både begrense informasjonsmengden som må sendes til AutoNauten, og i tillegg begrense antallet manuelle inputs nødvendig for å starte et oppdrag, ble algoritmer for å begrense antall waypoints utviklet. To potensielle beskjæringsalgoritmer ble testet, én basert på å hoppe over enkeltpunkter og én basert på rekursiv segmentering av ruten. Sistnevnte ble ansett å være bedre på grunn av bedre skalerbarhet og fleksibilitet.

Preface

This thesis is the result of work related to my master's degree at the Norwegian University of Science and Technology in the five year study program Cybernetics and Robotics. The course itself, TTK4900, counts for 30 ECTS credits. It was completed in collaboration with the NTNU AutoNaut project, under the NTNU AUR-Lab umbrella. The work is a continuation of a project thesis completed in the winter of 2022[1]. Chapter 2 and Section 3.2 were heavily inspired by this paper.

First I would like to thank my supervisor Tor Arne Johansen at the Department of Engineering Cybernetics. Johansen has helped me steer the project in the right direction, and his extensive knowledge on the subjects discussed in this work has been indispensable. I would also like to thank Henning Øveraas and Alberto Dallolio, both PhD candidates at the Department of Engineering Cybernetics. They have both provided insight into the AutoNaut project, and have answered my questions diligently. I also want to give thanks to PhD candidate Simon Blindheim, also at the Department of Engineering Cybernetics. He has been instrumental in teaching me how to use the tools he has created in his research, and answered any and all questions I had. Finally, my thanks to Einar Bjørløw Sønju for his knowledge in the field of path planning.

Contents

Figures

Acronyms

ACO Ant Colony Optimization. 6

ADCP Acoustic Doppler Current Profiler. 14, 16

AIS Automatic Identification System. 12, 15, 59, 63

APF Artificial Potential Field. 6

API Application Programming Interface. i, iii, 18, 22, 23, 27–29, 33, 34, 39, 57, 63–65

AUR-Lab Applied Underwater Robotics Laboratory. v, 9

BBB BeagleBone Black. 12

CA Collision Avoidance. 2, 4, 6, 7, 12, 57

COLREGs Convention on the International Regulations for Preventing Collisions at Sea. 6, 63

CTD Conductivity, Temperature and Depth. 14, 16

DoS Denial-Of-Service. 57

DUNE DUNE Uniform Navigation Environment. 12, 15, 58

ECTS European Credit Transfer and Accumulation System. v

ENC Electronic Navigational Chart. i, iii, ix, 1, 2, 4, 6, 18, 27, 39, 42, 56, 59, 64, 65, 73

FSM Finite State Machine. x, 59, 60

GLUED GNU/Linux Uniform Environment Distribution. 12

GNSS Global Navigation Satellite Systems. 1

GPS Global Positioning System. 1, 27

GUI Graphical User Interface. 15

IMU Inertial Measurement Unit. 14

JSON JavaScript Object Notation. 23, 33, 62

LSTS Underwater System and Technology Laboratory. 15

MET The Norwegian Meteorological Institute. i, 22, 23, 28, 29, 33, 34, 56, 57, 62, 64, 65

NTNU Norwegian University of Science and Technology. i, iii, v, 1, 3, 9

ORCAS Online Risk management and risk Control for Autonomous Ships. 8

PCB Printed Circuit Board. 14

PSO Point Swarm Optimization. 6

RC Routine Correction. 4, 7, 57

RCWS Remote Controlled Weapon Station. 3

USV Unmanned Surface Vessel. i, iii, ix, 2–4, 6, 7, 57, 58, 65, 66

UTM Universal Transverse Mercator. ix, 27, 28

VHF Very High Frequency. 15

WGS World Geodetic System. 27, 28

Glossary

admissibility An admissible heuristic is a heuristic where the estimated cost is always lower than or equal to the real cost of reaching the goal node.. 22

beam sea Sea state where the wave direction is orthogonal to the ship's length.. 29, 44, 53, 55

consistency A consistent heuristic is a heuristic which is always less than or equal to the estimated distance from any neighboring node to the goal node, plus the cost of reaching that neighbor.. 22

following sea Sea state where the wave direction is the same as the ship's own direction.. 29

geodatabase Database containing geographic data.. iii, 4, 18, 39, 59, 73

head sea Sea state where the wave direction is the opposite of the ship's own direction.. 29, 44, 52, 55

heuristic Tool used to find a solution to a problem, but not necessarily the best solution.. 4, 20, 30

ownship One's own vessel.. 30, 58

quartering sea Sea state where the wave direction is around 45 degrees to its heading.. 55

shapefile Format of data containing geometric data as well as other attributes. Commonly used by geographic information systems.. 4, 56

Chapter 1

Introduction

In Chapter 1, recent innovations in marine exploration and navigation will be discussed. Several problems related to autonomous navigation, as well as some proposed solutions, will be presented. Different application areas at sea, as well as different types of seafaring vessels will also be discussed. Then, the specific problem of path planning for autonomous craft at sea will be addressed, and then the problem as related to the NTNU AutoNaut. Lastly, an outline for this thesis will be laid out.

1.1 Navigation at sea in the presence of harsh weather

Seafarers have always been at the mercy of the weather. Tall waves, high winds and strong currents make traversing the oceans a dangerous exploit[2]. This is especially true for smaller vessels, which are the focus of this work, since they are less resistant to weather forces. The weather can also change quickly at sea, and mariners need to be prepared for this. Local geography, the presence of reefs, skerries and islands, as well as shallow waters and banks, all have to be accounted for by seamen.

Modern tools have made ocean navigation easier. Satellite systems have enabled pinpoint accuracy in position with Global Navigation Satellite Systems such as GPS and Galileo[3]. Satellite imagery has also made it possible to generate detailed nautical charts. Combining these technologies has allowed for the creation of sophisticated navigation systems, for example through the use of Electronic Navigational Chart (ENC)s, illustrated in Figure 1.1. Such interactive systems allow non-human actors to automatically handle tasks that have traditionally been the responsibility of humans. ENCs specifically have been particularly useful for solving the problems posed in this thesis.

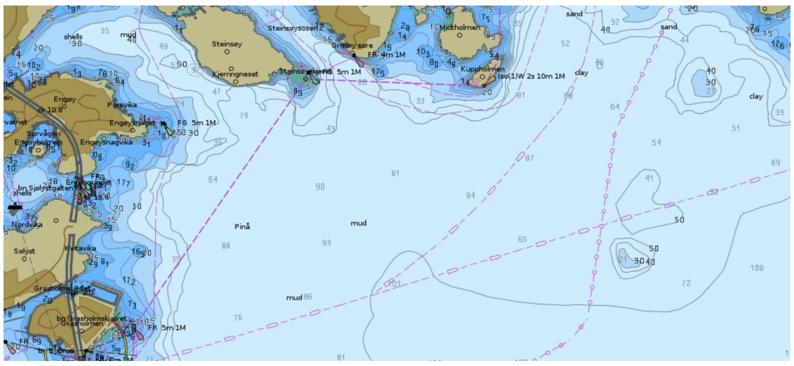


Figure 1.1: Electronic Navigational Chart[4].

1.2 Unmanned maritime navigation

Unmanned maritime navigation is an area of study that has recently gained much traction. Reasons for this include the invention of modern navigational tools and technologies, as well as a dramatic increase in computing power[5][6]. These systems are able to determine position, velocity and bearing of a vehicle automatically and almost instantly. Advanced path planning, path following and sensor fusion algorithms are combined into sophisticated mission planners, allowing vessels to carry out missions at sea with little human input.

Several Unmanned Surface Vessels (USVs) in operation today are capable of autonomous or semi-autonomous missions. One example is the craft *JingHai-I*, pictured in Figure 1.2, developed by Shanghai University[7]. Meant for hydrographic surveying in shallow waters, this vessel is capable of travelling at fast speeds while performing aggressive CA maneuvering. This is made possible by its water jet-based propulsion system.

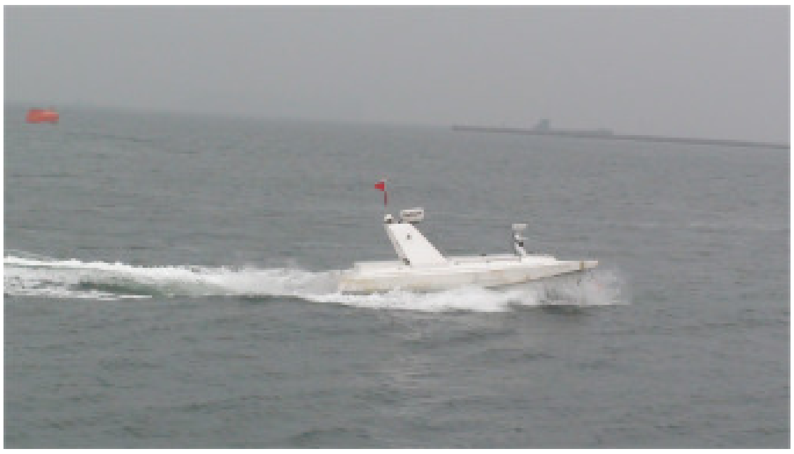


Figure 1.2: JingHai-I USV[7].

Not all USVs are purely scientifically motivated. The craft Protector USV, designed by Israeli defence contractor Rafael Advanced Defense Systems, was created for marine surveillance and shoreline protection. Pictured in Figure 1.3, the Protector comes with an integrated Remote Controlled Weapon Station and an advanced target tracking system[8]. Like the Jinghai-I, the Protector is also water-jet based.

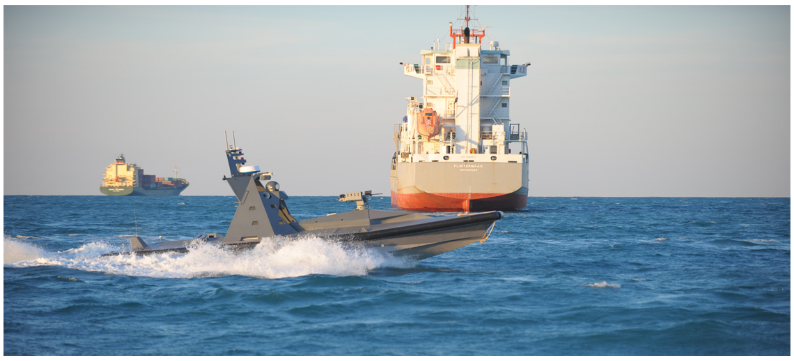


Figure 1.3: Protector USV[8].

Most USVs in operation today are unable to carry out missions of considerable length. This is largely because they often employ water-jet or thruster-based propulsion systems. This limits the potential duration of their missions, as they will quickly run out of fuel or battery capacity. The Protector and JingHai-I are both examples of this, and consequently are limited to operational times of a few hours. The JingHai-I, for instance, was only meant for data collection along a relatively short stretch of shore, and can be brought back for refueling after each of these sessions. The Protector, on the other hand, is remote-controlled, either from a nearby ship or from shore. It can therefore be brought back, refuelled and redeployed whenever necessary.

Some projects require surveillance and data harvesting over large areas. In these circumstances, craft like the two mentioned are at a disadvantage. For missions spanning days, weeks or months, high-speed aggressive maneuvering will not be possible, and travel velocities and actuation will have to be reduced. In addition, the vessel will have to rely on renewable energy sources for extended stays at sea. This work documents one such vehicle: the NTNU AutoNaut.

The AutoNaut will be discussed in detail in Chapter 2. Since the AutoNaut is meant to stay at sea for long periods of time in remote areas, it is infeasible to operate it remotely like the Protector. It needs an automatic path planner, which is the main focus of this thesis. In addition, the AutoNaut is a small, slow vessel at the mercy of the elements. It harnesses renewable energy in the form of wave and solar power for propulsion and operation. It cannot simply be redirected to port on short notice or collected when convenient, and rescue missions may take hours or days depending on the conditions. This thesis aims to perform path planning for the AutoNaut over large distances while reducing the risks related to harsh weather.

1.3 Previous work

This thesis builds on work done in the Autumn semester of 2021 at NTNU[1]. The project thesis written in this time serves as a proof of concept for a grid-based path planning scheme based on geodatabase information. In the project thesis, geodatabase data containing geographic and oceanographic data was collected from the Norwegian Mapping Authority's open source database. The tool SeaCharts was then used to generate shapefiles from this data and merge them into an interactive Electronic Navigational Chart[9]. An adjustable resolution grid consisting of node objects was then created and projected onto the ENC. A graph-search algorithm, the A* algorithm, was subsequently applied to create a feasible path from a given starting point to a given end point[10]. This algorithm relies on a heuristic, a measure of the overall performance of the algorithm at the current time-step[11]. The heuristic chosen in the project was the Euclidean distance between the current position and the goal position, mostly due to its uncomplicated implementation.

Using the method implemented in the project thesis, the system was able to successfully generate paths from one point to another at sea, if such a feasible path existed. An example generated for the project thesis can be seen in Figure 1.4. The algorithm is able to generate a path from the white circle to the green circle, even in the presence of a multitude of obstacles, narrow passages and great distances. The generated path is a list containing a waypoint for each movement decision the algorithm has made. This list can span thousands of entries, and represents a real-life path that can be hundreds of kilometers long.

1.4 Related work

Several other works have been made to solve problems similar to the main focus of this thesis. This section will discuss some of them, both external works created for other autonomous craft, and internal works created specifically for the AutoNaut project.

1.4.1 Similar problems

The problem of path planning for USVs is a popular research area. A 2019 paper published in the Ocean Engineering journal proposes a path planner for a USV capable of avoiding both static and dynamic obstacles[12]. This project handles both Routine Correction (RC) of the path based on the water level, as well as a Collision Avoidance (CA) scheme based on camera data. The resulting path planner is able to handle both static and dynamic obstacles in a marina by modelling the ocean depth to avoid reefs, as well as camera data to avoid the pier and other ships. They reject the usage of the A* algorithm specifically, stating that "the traditional A* approach is exclusively designed for static environments."

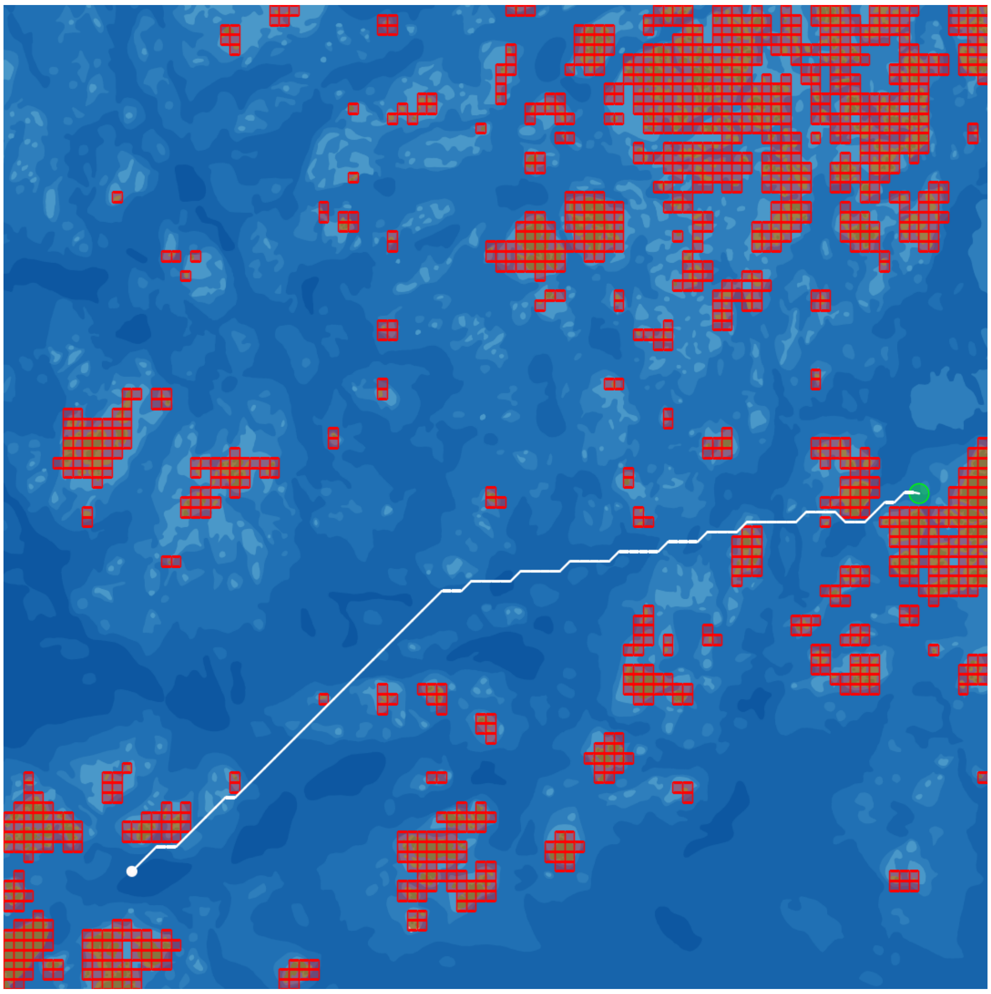


Figure 1.4: Path generated through an archipelago north of Frohavet, with the red squares designating land and shallows. The white circle represents the starting point, and the green circle represents the goal point.

Another paper that seeks to solve the same problem is [13]. This paper from 2021 uses a grid-based approach for path planning like in [1]. Here, however, the path planner itself is a hybrid of the Ant Colony Optimization (ACO) and Artificial Potential Field (APF) algorithms. Global path planning is performed using ACO, while local path planning and undiscovered obstacles are handled by the APF algorithm.

Path planning and mission planning for USVs is a fairly active field of study, as demonstrated by both the mentioned works. Other related projects have also been completed in recent years[14]. A common denominator for many of these works is the limited scale of the missions. Usually, the generated paths will only be a few hundred meters in length, and it is unclear how well the proposed solutions will scale in both time and space. In addition, many path planning algorithms disregard weather effects altogether. Different path planners might be suitable for different vessels, environments and mission scales. Nevertheless, few works incorporate data from weather forecasts into large-scale path planning.

1.4.2 AutoNaut academic work

Multiple theses have been written about the AutoNaut project and the vessel itself. Master's theses, PhD work and journal publications are among the many contributions that have gone into the project.

An investigation into system requirements was conducted in 2018[15]. In this work, many of the hardware specifications were determined, both in terms of power systems, actuators and communications hardware. These specifications would later become the first building block of the system architecture, which is covered in Section 2.2.1. The AutoNaut's communications hardware is described in Section 2.2.3. This work also presented a rudimentary fallback autopilot. Both hardware and the autopilot were then validated with real tests.

A CA scheme was also developed in 2018[16]. This COLREGs compliant system proved capable of avoiding other vessels in simulation. An autopilot is then designed, and both the CA system and autopilot are tested in sea trials. Most of what is now level 2, discussed in Section 2.2.2 was developed in this thesis.

The subject of a 2021 thesis was speed prediction for the AutoNaut[17]. Here, the effects of weather forces on the AutoNaut's ground speed is investigated. An estimate of this speed was then created, based on both sensor data and weather forecasts. This estimate was used to create a time-efficient path planner optimized with a PSO algorithm. The path planner created in this work does not account for geography, so it is unable to avoid obstacles.

A grounding avoidance and CA system for the AutoNaut was developed in 2021[18]. This system relied on ENCs, which is also the basis for this work. Identifying grounding hazards is heavily emphasized in the thesis. The system, which is also COLREGs compliant, was tested in simulation and yielded promising results.

The AutoNaut project officially started in 2017, largely under the guidance of PhD candidate Alberto Dallolio. AutoNaut specifications and tutorials have all been collected by Dallolio in the form of the AutoNaut wiki[19]. The AutoNaut wiki also contains a comprehensive overview of the AutoNaut's mechanical, electrical, sensory and software components. Much of the theoretical work which has gone into the AutoNaut can also be found here[20]. Detailed mission logs are also stored on the wiki[21].

A mission in April 2021 was undertaken during a Spring storm[22]. Rough weather caused the AutoNaut to capsize and left the vessel uncontrollable. Since the GPS receiver was submerged, the AutoNaut also lost positional data. The AutoNaut then beached on an island outside the Trondheim fjord, and was recovered there the following day. Some repairs were necessary, but the project was quickly up and running again. This mission failure demonstrated a need to take weather into account when designing missions. While there is nothing that can be done about the weather itself, weather forecasts can be collected and informed decisions can be made from them.

1.5 Problem statement and outline

The purpose of this thesis is to build on the path planner that was created in the project thesis of 2021 to create a more comprehensive path planner for the AutoNaut[1]. First, the AutoNaut will be introduced to give an impression of the system as a whole. A method of collecting weather forecasts of waves, wind and ocean currents online will then be devised. Then, a means to make this data interact with the current grid-based model will be created. The classic A* algorithm will be modified to take the weather data into account when planning paths, and a tuning scheme will be created to properly weigh the different data inputs against each other, with special emphasis on safe operation. Buffer polygons will be used to virtually enlarge the obstacles posed by land formations and shallows, and the path planning algorithm will be modified to stay clear of these buffers. Lastly, the paths generated by the path planner will be fed into a pruning algorithm to reduce the total number of waypoints.

1.6 Contributions

Several contributions have been made to the AutoNaut project in this paper. Furthermore, the work is not tightly coupled to the AutoNaut platform itself, and can be useful for path planning in other projects involving small to medium sized USVs.

First of all, the AutoNaut is in need of a robust safety-oriented mission planner. Such a mission planner would need a path planner, which is the subject of this paper. In addition to this, an RC system, a CA system, plus multiple other safety systems will be necessary. This thesis covers one of the steps in pursuit of that goal.

Secondly, the work done in this paper is of interest to the author of the SeaCharts project, as well as their colleagues in the ORCAS project[23]. They have signalled interest in incorporating some of the weather functionality developed in this thesis into SeaCharts.

All code related to this project was written in the Python programming language. Scripts related to the work can be found in a GitHub repository created specifically for this paper[24]. The code found in the repository, along with all accompanying plots, are provided under free use by the MIT License listed in Appendix A. Example scripts to get started are discussed in Appendix B.

Chapter 2

NTNU AutoNaut

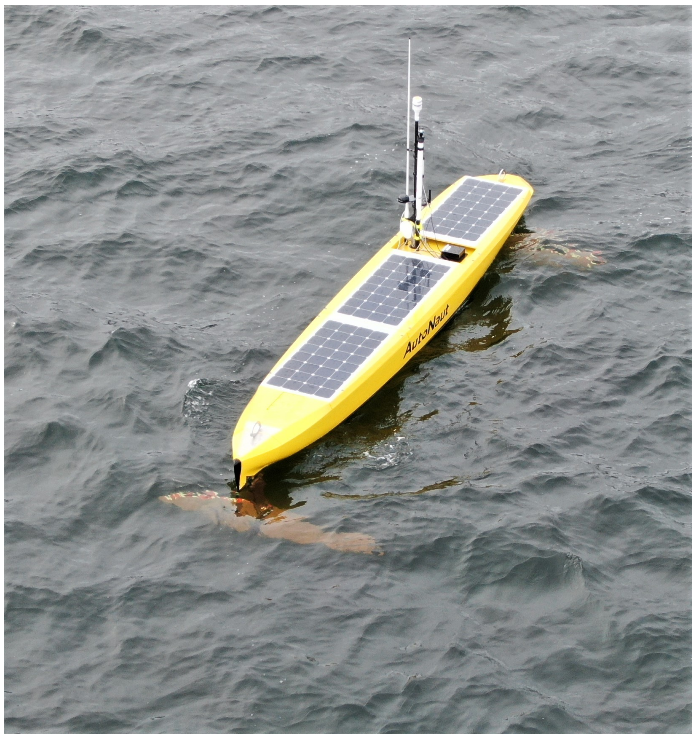
The AutoNaut, pictured in Figure 2.1, is a surface-faring, wave-powered vessel owned by the Applied Underwater Robotics Laboratory (AUR-Lab) at NTNU in Trondheim[25]. This chapter will provide an overview of the AutoNaut's mechanical, electronic and software capabilities. The hull of the craft and its propulsion system will be described, as well as the different hardware components on-board. The middleware and on-board software systems will also be documented. Most of the information and images presented in this chapter were sourced from the AutoNaut wiki[19].

2.1 Mechanical and electrical systems

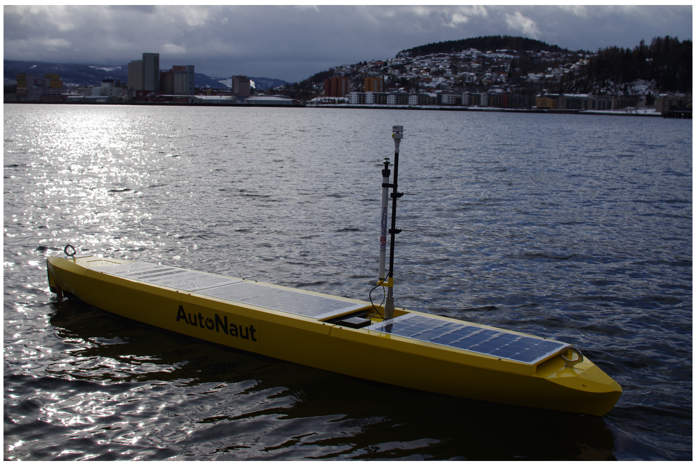
The AutoNaut vessel itself is not designed or manufactured by the NTNU AutoNaut project or AUR-Lab. It is the intellectual property of MOST (Autonomous Vessels) Ltd., a British company, and comes as a package with a suite of internal hardware and software related to ocean exploration and data collection[26]. The majority of this internal hardware has been gutted and replaced with components tailored for the NTNU AutoNaut project. The original solar panels and wave-foil propulsion system still remain.

NTNU AutoNaut is 5 meters long. It has a beam of 0.8m, a draft of 0.7m and a displacement of 230kg. Its hull is made of fiberglass with PVC infill. The primary propulsion system consists of two spring-loaded hydrofoils attached to the underside of the craft. These foils, which can be seen mounted on the fore and aft of the keel in Figure 2.2, give the vessel a maximum speed of around 3 knots. Propulsion is achieved when waves impart energy on these hydrofoils, and this energy is released. The foils are responsive to motion in both roll and pitch. Turning is made possible by an aft-mounted rudder. An auxiliary thruster allows for maneuvering in still water[27].

The electrical system of the AutoNaut is powered by four 12V 70Ah lead-acid batteries. These are located in the dark grey section in Figure 2.3. During operation, these are charged by three 300W solar panels, visible on the craft's deck in Figure 2.1a.

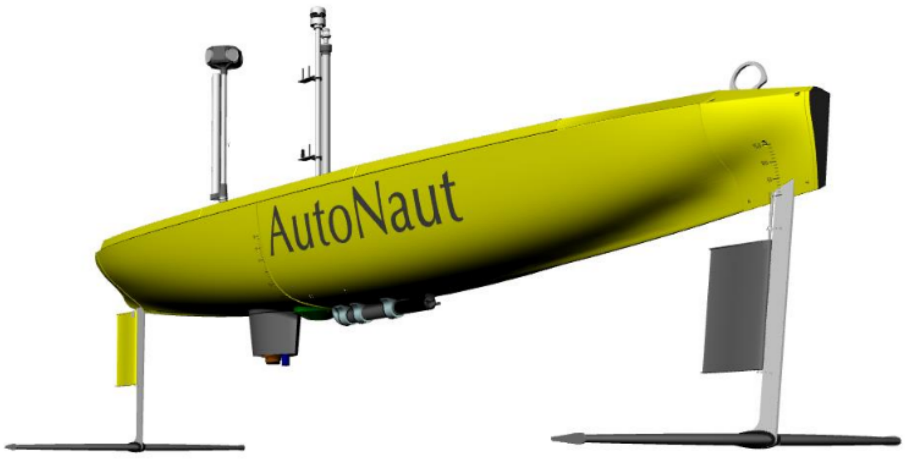


(a) AutoNaut front view.

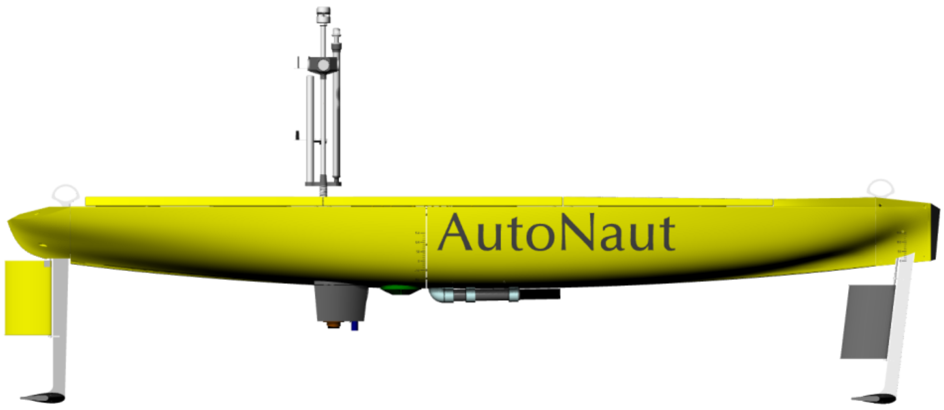


(b) AutoNaut side view.

Figure 2.1: AutoNaut in operation[19].



(a) Exterior of the AutoNaut, profile view.



(b) Exterior of the AutoNaut, side view.

Figure 2.2: Exterior of the AutoNaut[19].

2.2 System Architecture

The AutoNaut consists of three separate levels of system hierarchy. Each level has certain areas of responsibility, and is associated with its own isolated chamber within the craft's hull. These levels will be discussed in detail in this section. Operation and communication with the AutoNaut will also be explained. These systems and their interaction are illustrated in Figure 2.4.

2.2.1 Level 1

Level 1 deals primarily with system monitoring on-board the AutoNaut, and is visualized as the orange box in Figure 2.3. The smart charger for the solar panels, the Victron BlueSolar controller in Figure 2.5c, is located here. The A Campbell Scientific CR6 board is responsible for the software running on layer 1. This board can be seen in Figure 2.5a. The GPS unit, mounted on the front of the deck, also falls under level 1, and is pictured in Figure 2.5b. This is a smart GPS antenna, and in addition to positional data it can also output velocity, heading, pitch, roll and heave.

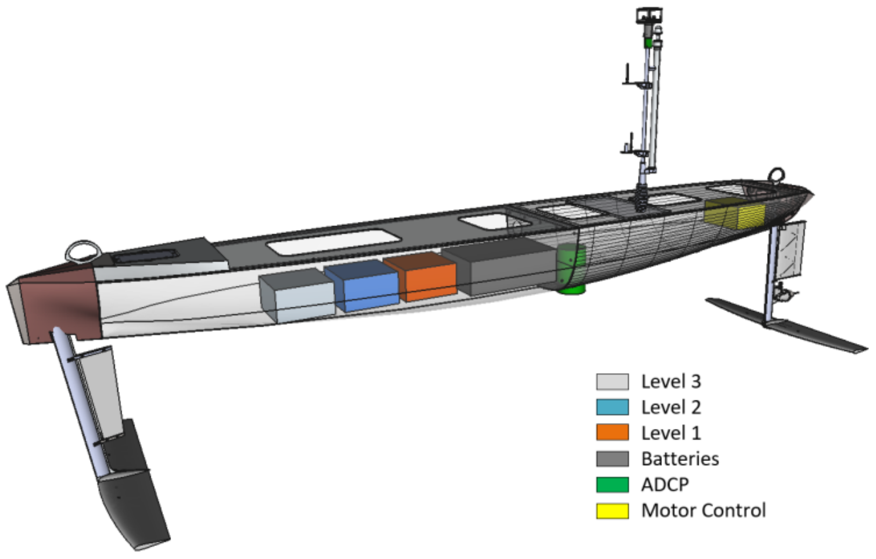


Figure 2.3: Interior of the AutoNaut[19].

2.2.2 Level 2

Level 2 is responsible for the navigation and control of the AutoNaut. This system is compartmentalized in the blue box in Figure 2.3. It contains many of the sensor systems on-board. This sensor data is handled by a timing board, the SenTiBoard, pictured in Figure 2.6e. Data from these sensors will not be accessed directly in this thesis work. However, some of the parameters and constants mentioned in Chapter 3 are created as stand-ins for this data, and should be replaced when live sea-trials take place.

In addition to sensors, level 2 also contains the emergency CA algorithm, and handles operator communication. The software running on level 2 is hosted on the BeagleBone Black (BBB) computer depicted in Figure 2.6a. The operating system running on the BBB is called GNU/Linux Uniform Environment Distribution (GLUED), a Linux distribution made for embedded computers. A session of the CA system mentioned in Section 1.4.2 runs on GLUED. In addition, the middleware running on the AutoNaut, DUNE, is also run on GLUED.

Several sensor systems are also encapsulated by level 2. An Automatic Identification System (AIS) informs the AutoNaut of the position and velocity of nearby

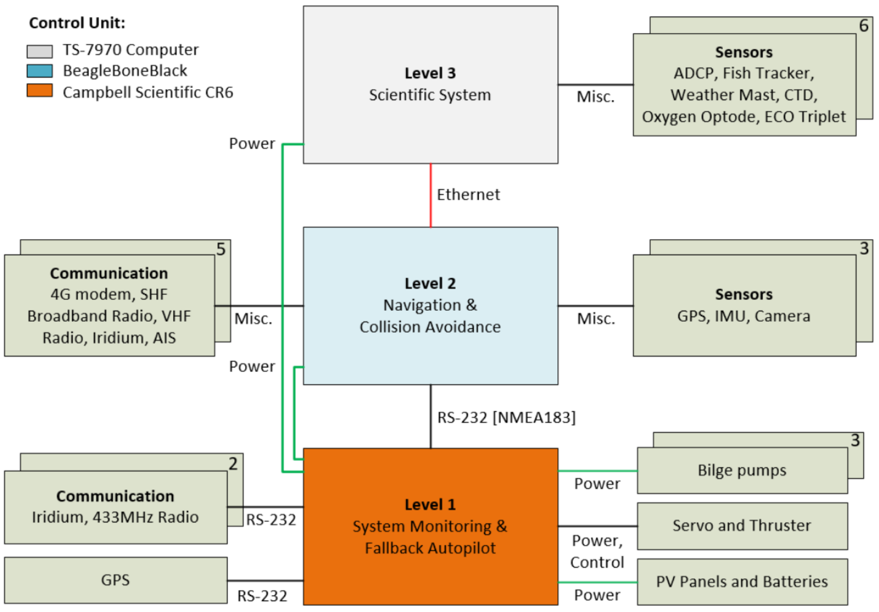


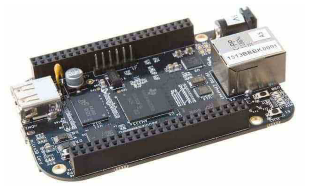
Figure 2.4: AutoNaut System architecture[19].

(a) The Campbell Scientific CR6[28].

(b) Vector V104 GPS Smart Antenna[29].

(c) BlueSolar charge controller[30].

Figure 2.5: Level 1 hardware.


(a) BeagleBone Black[31].

(b) Raymarine AIS650[32].

(c) Airmar Weather Station[33].

(d) HMR3000 Digital Compass[34].

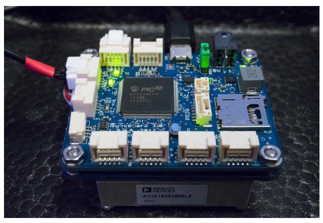

(e) SenTiBoard[35].

Figure 2.6: Level 2 hardware.

ships. It is pictured in Figure 2.6b.The Airmar 120WX Weather Station shown in Figure 2.6c is mounted on the AutoNaut's antenna. The weather station provides a data stream of wind speed, temperature, air pressure and more. Level 2 also contains an IMU, which is not pictured here, as well as the digital compass in Figure 2.6d.

2.2.3 Level 3

Logging of sensor and mission data is the primary concern of level 3. Located in the light grey box in Figure 2.3, it is responsible for the scientific system of the vehicle. All data processing on this layer goes through the TS-7970 PCB shown in Figure 2.7a.

Most of the hydrographic and biological data collected by the AutoNaut is handled by this layer. Data related to dissolved O_2 content is harvested using the keel-mounted oxygen optode illustrated in Figure 2.7d. An Acoustic Doppler Current Profiler (ADCP), seen in Figure 2.7b, can determine the properties of waves and currents. It can also measure dissolved biomass and ice content of the water. Salt concentration is measured by a Seabird Conductivity, Temperature and Depth (CTD) instrument, pictured in Figure 2.7c. Chlorophyll content is measured by the ECO Puck Triplet seen in Figure 2.7f, which can analyze the algal content of water. Lastly, the TBLive pictured in Figure 2.7e is a fish tracker capable of receiving acoustic signals from chipped fish in the operating area.

2.3 Communication

The AutoNaut has a catalog of communication methods. Satellite communication is made possible through the Iridium constellation[42]. The RockBLOCK+ unit pictured in Figure 2.8a is the on-board Iridium network transceiver.

Broadband communication is available with a 4G modem. The MikroTik modem is pictured in Figure 2.8c. Radio communication is also available on the vessel. An Owl VHF radio module with AIS support allows the craft to pick up signals from nearby ships. The radio is depicted in Figure 2.8b.

2.4 Operation of the AutoNaut

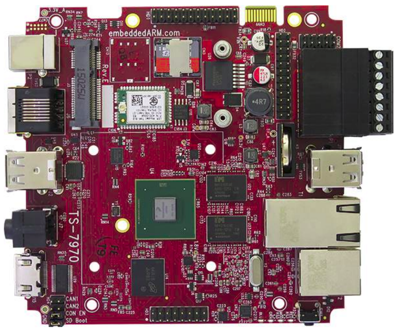
In this section, the middleware used to issue commands to the AutoNaut will be discussed briefly. The middleware and accompanying software was developed by the Underwater System and Technology Laboratory (LSTS) at the University of Porto[46].

2.4.1 Neptus

Neptus is a software run by an operator on land. It is a middleware used to issue waypoints to the AutoNaut, designated as *tags* in the software. Neptus is also used by operators to keep track of the current state of the vehicle, including position and velocity. An example of a mission created and monitored in Neptus can be seen in Figure 2.9. Here, the grey circles represent tags issued to the AutoNaut.

2.4.2 DUNE

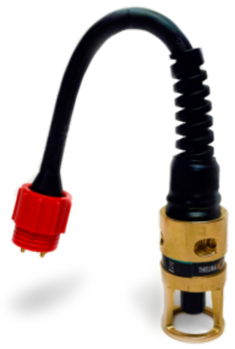
DUNE is a software running on level 2 on-board the AutoNaut. It communicates with a Neptus session on land, and receives way-points from there. These waypoints are then issued to the navigation and control systems on-board. DUNE is purely terminal-based, and has no GUI attached to it.


(a) TS-7970[36].

(b) Signature500 ADCP[37].

(c) Seabird CTD SBE 49[38].

(d) Aanderaa Oxygen Optode 4835[39].


(e) ThelmaBiotel TBLive[40].

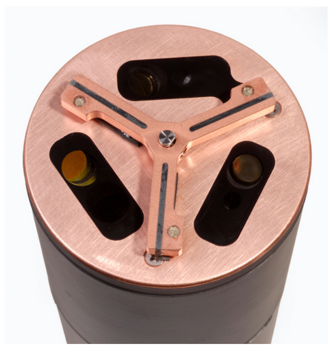
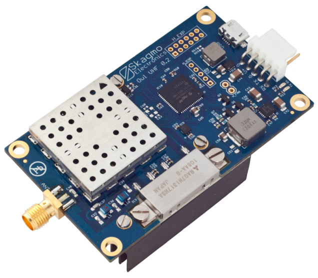
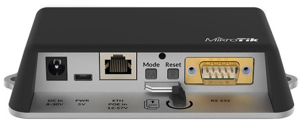

(f) WET Labs ECO Puck Triplet[41].

Figure 2.7: Level 3 hardware.

(a) RockBLOCK+ Iridium unit[43].

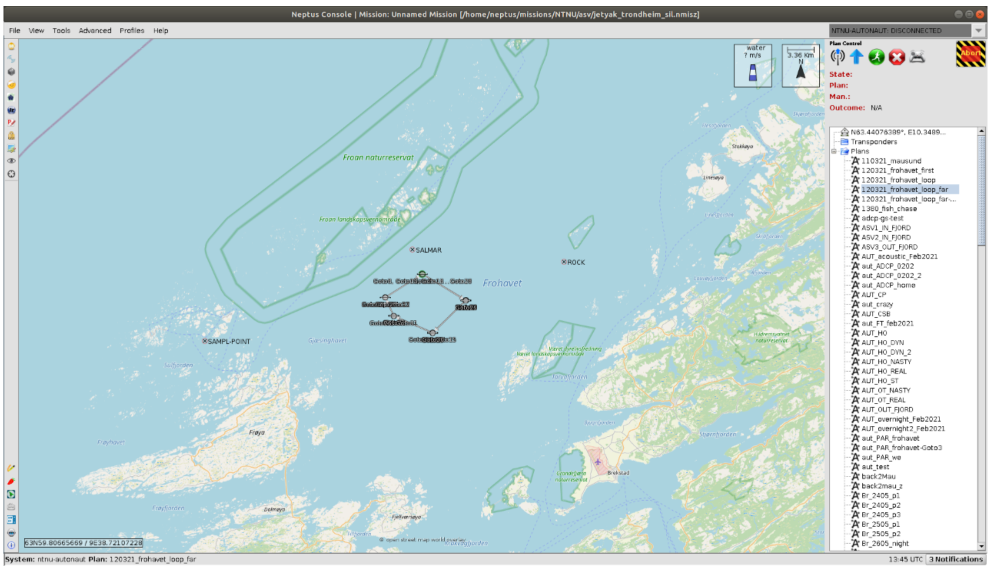


(b) Owl VHF radio[44].



(c) MikroTik 4G modem[45].

Figure 2.8: Communication devices on-board the AutoNaut.


Neptus Console | Mission: Unnamed Mission [/home/neptus/missions/NTNU/asv/jetyak_trondheim_sil.nmisz]
File
View
Tools
Advanced
Profiles
Help
NTNU-AUTONAUT: DISCONNECTED
Plan Control
Abort
water
? m/s
3.36 Km
N
Froan naturreservat
Froan landskapsvernområde
SALMAR
ROCK
Frohavet
SAMPL-POINT
Frøya
Frøyhavet
Frøyfjorden
Dolmøya
Fjellværsøya
Brekstad
Stokkøya
Linesøya
© open street map world overlay
63N59.80665669 / 9E38.72107228

State:
Plan:
Man.:
Outcome: N/A

N63.44076389°, E10.3489...
Transponders
Plans
110321_mausund
120321_frohavet_first
120321_frohavet_loop
120321_frohavet_loop_far
120321_frohavet_loop_far-...
1380_fish_chase
adcp-gs-test
ASV1_IN_FJORD
ASV2_IN_FJORD
ASV3_OUT_FJORD
AUT_acoustic_Feb2021
aut_ADCP_0202
aut_ADCP_0202_2
aut_ADCP_home
AUT_CP
aut_crazy
AUT_CSB
aut_FT_feb2021
AUT_HO
AUT_HO_DVN
AUT_HO_DVN_2
AUT_HO_NASTY
AUT_HO_REAL
AUT_HO_ST
AUT_OT_NASTY
AUT_OT_REAL
AUT_OUT_FJORD
AUT_overnight_Feb2021
AUT_overnight2_Feb2021
aut_PAR_frohavet
aut_PAR_frohavet-Goto3
aut_PAR_we
aut_test
back2Mau
back2mau_z
Br_2405_p1
Br_2405_p2
Br_2405_p3
Br_2505_p1
Br_2505_p2
Br_2605_night

System: ntnu-autonaut **Plan:** 120321_frohavet_loop_far 13:45 UTC **3 Notifications**

Figure 2.9: Neptus session. Image courtesy of Henning Øveraas.

Chapter 3

Theory and implementation

This chapter will provide an overview of the background theory necessary to understand the problems solved in this work. The SeaCharts API and the A* algorithm will be explained. Then, the process of collecting weather forecasts and organizing weather data will be described. Visualizing this data will also be covered. The process of modifying the path planning algorithm to account for weather data will be explained, as well as the tuning of the resulting new algorithm. A method of guaranteeing safe distance to obstacles is then generated. Lastly, post-processing of the generated paths will be discussed.

3.1 SeaCharts

SeaCharts is an API for creating ENCs in Python[9]. It is available in an open GitHub repository for use and modification under the MIT license[47]. The tool allows for the creation of interactive maps built from geodatabase files. Land, shallows and different layers of ocean depth can be accessed through the API, a wide range of operations can be performed on them, and the tool also has a visual component to it. The API is both built on and is compatible with the Shapely library[48]. An example of a figure generated in SeaCharts can be seen in Figure 3.1. Green represents land in the chart, and different shades of blue represent different ocean depths.

3.2 The A* algorithm

An A* path planning algorithm was implemented in the project thesis [1]. It was chosen for its relatively straight-forward implementation, as well as its efficiency in sparse maps with a single goal. The algorithm successfully generated paths which avoided obstacles in the form of land and shallows. This section is a recapitulation of the theory and implementation of the A* algorithm from the project work. A more in-depth explanation can be found on pages 24-27 in the project thesis.

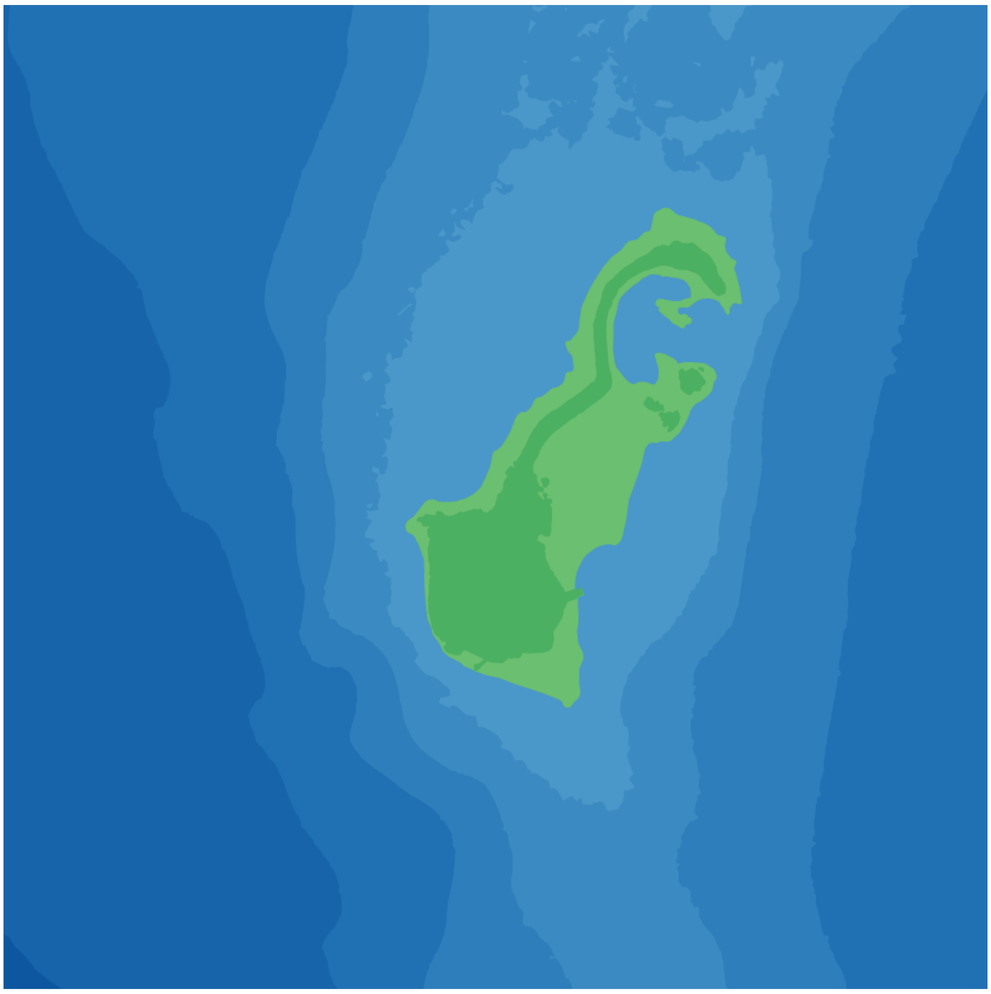


Figure 3.1: Figure generated in SeaCharts.

A* is a grid-search path planning algorithm[10]. The grid consists of cells, also known as *nodes*, and each cell can either take the form of an obstacle or an occupiable cell. This is an informed search algorithm, meaning it uses a heuristic, a measure of success which helps the algorithm terminate quickly[11]. If the right heuristic is used, the solution can be guaranteed to be optimal. The pseudocode for the classic A* algorithm can be seen in Algorithm 1.

Algorithm 1 A* algorithm pseudocode

```
 1: open_list ← list containing start_node
 2: closed_list ← empty list
 3: start_node.g ← 0
 4: start_node.f ← start_node.g + heuristic(start_node, end_node)
 5:
 6: while open_list not empty do
 7:     current ← element in open_list with lowest f
 8:     if current is end_node then
 9:         return reconstruct_path(end_node)
10:     end if
11:     remove current from open_list
12:     add current to closed_list
13:     for each neighbor in neighborNodes(current) do
14:         if neighbor not in closed_list then
15:             neighbor.f ← neighbor.g + heuristic(neighbor, end_node)
16:             if neighbor not in open_list then
17:                 add neighbor to open_list
18:             else
19:                 open_neighbor ← neighbor in open_list
20:                 if neighbor.g < open_neighbor.g then
21:                     open_neighbor.g ← neighbor.g
22:                     open_neighbor.parent ← neighbor.parent
23:                 end if
24:             end if
25:         end if
26:     end for
27: end while
28: return False
```

As the algorithm traverses the graph, which is a grid projected onto a map such as Section 3.1, it keeps track of two lists. The first list, called the *open list*, contains the nodes which have yet to be investigated. The open list is an unordered list, and each node in the list is associated with an *f-score*, which determines the priority of the element. The node with the lowest f-score is investigated first. The element is then removed from the open list. When the node currently being investigated is the goal node, the algorithm terminates as it has reached the goal. If the open list is empty and the goal has still not been found, no feasible path between the start and the goal exists.

The other list is called the *closed list*. This list is the inverse of the open list, and contains the nodes which have been investigated. Nodes are moved here after they have been traversed if they are not the goal node. The current node's neighboring nodes are also investigated. If a neighbor constitutes an obstacle, it is discarded. If not, the cost of moving to each neighbor is calculated. This cost, called the *g-score*, and depends on the type of movement necessary to move to a neighbor node. Movement in 8 directions is possible and the algorithm supports both normal and diagonal movement. Since diagonal movement involves a slightly larger distance than normal movement, diagonal movement is associated with a slightly higher cost. After calculating the g-score of each valid neighbor, a list of valid neighbors are placed in the open list, ready for investigation.

The method for checking a node's neighbors and assigning g-score is written in pseudocode form in Algorithm 2. Each neighbor's g-score is evaluated, and if it is lower than that of the current lowest g-score in the open list, the neighbor is selected as part of the path. This is done by setting the node in the open list's parent node to the current neighbor node. When the algorithm has encountered the goal node, the path must be reconstructed. This is done according to Algorithm 3. The algorithm iterates through all the parent nodes from the goal to the start, and the result is the finished path.

Algorithm 2 Pseudocode for neighborNodes() function.

```
 1: neighbors ← empty list
 2: for each node in all directions do
 3:     if node is a valid direction then
 4:         add node to neighbors
 5:     end if
 6:     for neighbor in neighbors do
 7:         if neighbor is diagonal then
 8:             neighbor.g ← current.g + diagonal_cost
 9:         else
10:             neighbor.g ← current.g + normal_cost
11:         end if
12:         neighbor.parent ← current
13:     end for
14: end for
15: return neighbors
```

The heuristic, *h(n)*, is a function meant to help the algorithm zero in on the goal efficiently[1]. This is done by setting the f-cost of each neighboring node to the sum of the current g-cost and the heuristic, as seen in row 15 in Algorithm 1. A suitable heuristic must be chosen, and will depend on the specific problem to be

[1]Note that the heuristic $h(n)$ is not the same as the wave height h mentioned later.

solved. A good heuristic is an accurate estimate of the total cost from a start node to an end node. Two things are required for the heuristic to be accepted. The first one is admissibility, meaning the heuristic does not overestimate the optimal path from start to goal[49]. An admissible heuristic guarantees an optimal path. The second requirement for a good heuristic is consistency[50]. Consistency entails that the cost of moving to a neighbor node from the current node plus the cost of moving from that neighbor to the goal, will always be larger than or equal to the heuristic. A consistent heuristic is always admissible, but an admissible heuristic is not guaranteed to be consistent.

Several heuristics are possible candidates for a problem like this. In the project thesis, four different heuristics were considered; the maximum number of steps in one cardinal direction; Manhattan distance; euclidean distance; and the squared euclidean distance. The normal euclidean distance was selected for its convenient implementation.

Algorithm 3 Pseudocode for reconstruct_path() function.

```
1: path ← list with only end_node
2: while end_node.parent exists do
3:     end_node ← end_node.parent
4:     add end_node to path
5: end while
6: return path
```

3.3 Forecast data

This section will go over the methods used to collect weather data, as well as the implementation of tools for visualizing the data.

3.3.1 Weather service provider

An acceptable weather service provider had to be found. Requirements were access to wind, wave and current data, both in terms of magnitude and direction. In addition, the provider should be free to use, and offer unlimited daily downloads. Several service providers make weather forecasts available online. Most of them are paid services, and come with fixed monthly data limits or pay-as-you-go downloads. An example of this type of product is the OpenWeather API[51]. The APIs created by AccuWeather are free[52]. However, these only support up to 50 calls a day, and after that requires an upgrade to a paid plan.

A free service with all the requirements was found. Forecasts can be collected from the Weather APIs created by the The Norwegian Meteorological Institute (MET)[53]. MET has a large catalog of tools for geographical, meteorological and climatological research, and is free to use. For this project, wind forecasts were collected from the tool *Locationforecast*[54]. Both wind direction and wind speed are available using this tool. Wind direction comes in the format *wind from direction*. Locationforecast has a global coverage area, but is most accurate and frequently updated in the Nordic and Arctic regions, illustrated in Figure 3.2a and Figure 3.2b, respectively.

Current and wave data were sourced from the *Oceanforecast* tool[55]. Current speed and direction are available here, as well as wave height and direction. It should be noted that current direction is available in the format *current to direction*, while waves come in the format *waves from direction*. Oceanforecast does not have global coverage, however, and data is only available for Western and Northern Europe. The area of coverage is illustrated in Figure 3.3. Therefore, current and wave data needs to be sourced elsewhere for projects operating outside this area.

Weather data models from both APIs are based on MET's own numerical models. Data was harvested in JSON format using the *requests* package in Python[56]. Weather forecasts are available in a nine-day window. Hourly readings are available up to 54 hours into the future, and after that every 6 hours.

3.3.2 Visualization of weather data

The weather data sourced from MET's tools comes in a purely numerical format. Angles come in values between $0°$ and $360°$, wave height comes in meters, and wind and current speeds come in meters per second. To create an understanding of the different weather forces at play over an entire map, it is important to have some way of visualizing the collected weather data.

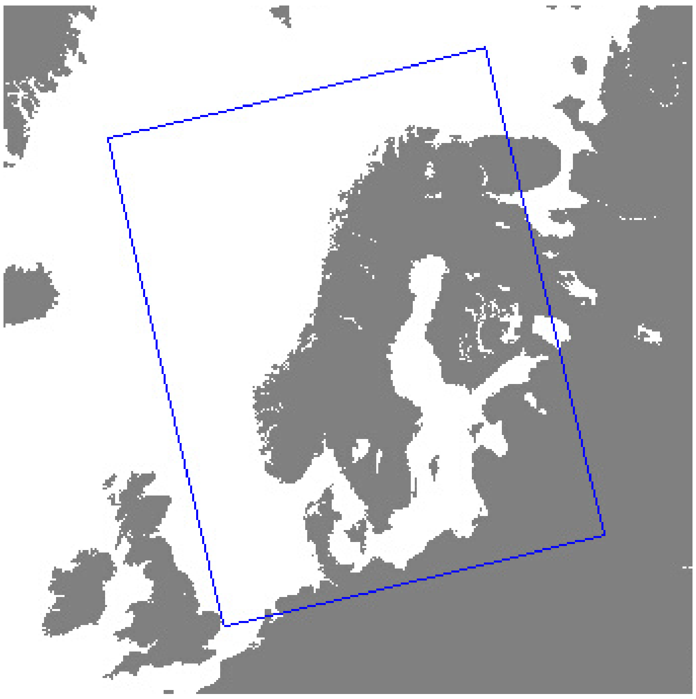
A standard way of visualizing wind, waves and current is vector fields. Data can be collected at each point of interest in the operating area, and each force of nature can be represented by an arrow, or vector. In commercial weather forecasts, these arrows are typically assigned a color based on the magnitude of the force in question. Such a method was also used for the illustrations in this work.

Force vectors for wind, waves and current can be created using basic trigonometry. To display their correct directions, it is important to keep in mind whether the force is designated as *from* or *to*, as mentioned in Section 3.3.1.

Wind and waves come in the *from* format. Wind and wave vectors will therefore take the form of the arrow in Figure 3.4a. Current vectors work the opposite way, and will therefore take the form of the arrow depicted in Figure 3.4b.

Figure 3.4 illustrates the weather vectors of wind, waves and current. In these vectors, a, o and h represent the adjacent side, opposite side and hypotenuse in the triangles, respectively. α is the incident angle of the weather force in question. Trigonometric principles are used to determine the adjacent and opposite sides, and these are used to calculate the end-points x_1 and y_1 of the wind and wave vectors, as well as the points x_0 and y_0 in the current vectors.

To find the end points of the vectors for wind and waves, one can apply trigonometric principles to arrive at Equation (3.1) and Equation (3.2).


(a) Nordic area.

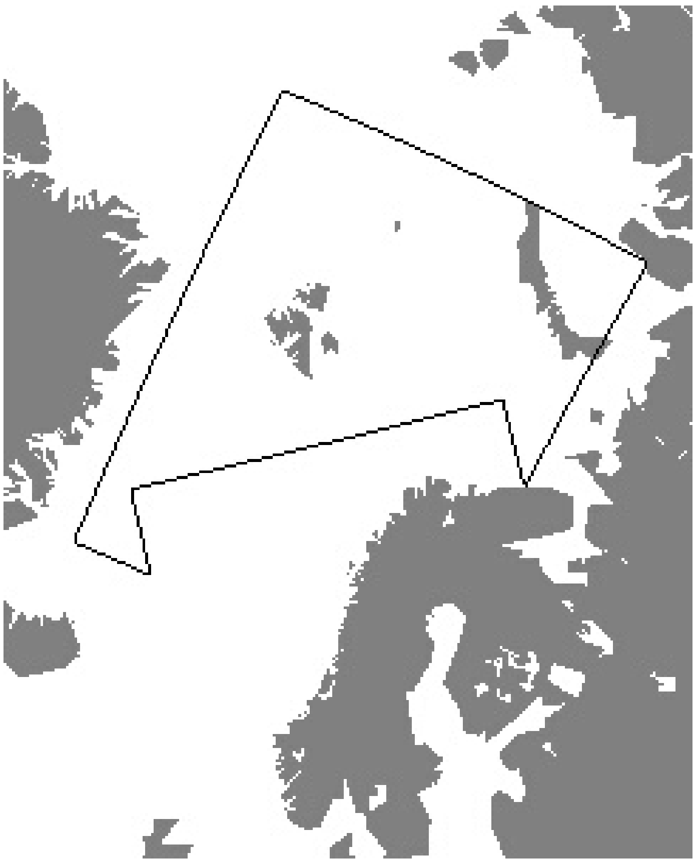

(b) Arctic area.

Figure 3.2: Main coverage area for Locationforecast[54].

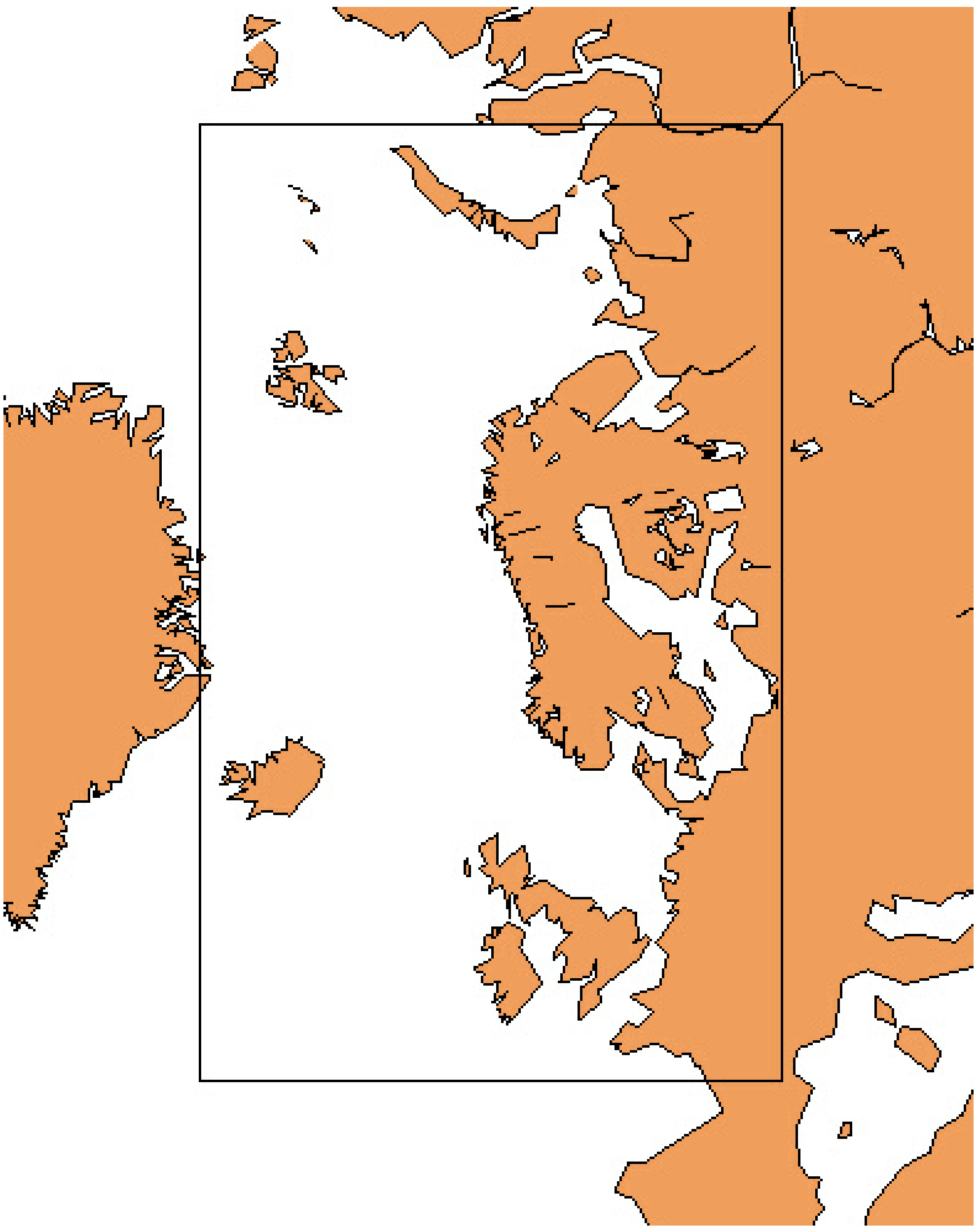
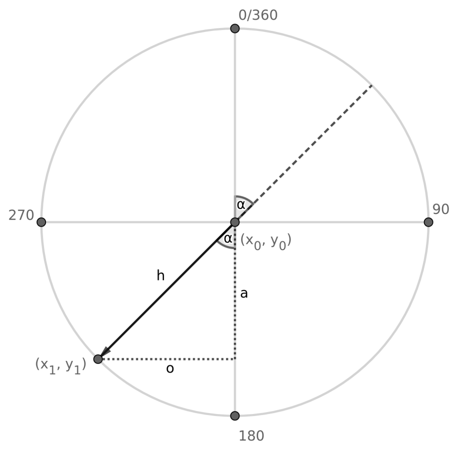
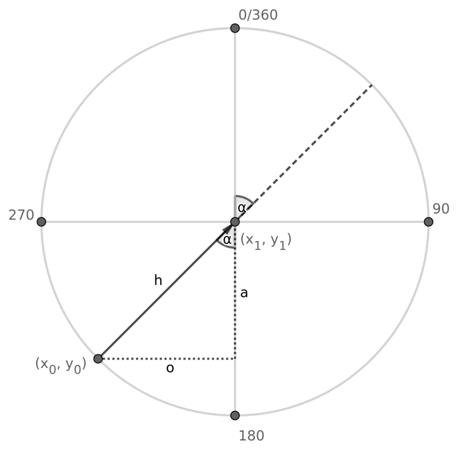


Figure 3.3: Oceanforecast coverage area[55].



(a) Wind and wave vectors.



(b) Current vectors.

Figure 3.4: Trigonometry for wind, waves and current vectors. Generated using the GeoGebra online calculator[57].

$$
\begin{aligned}
a &= hcos(\alpha) \\
o &= hsin(\alpha) \\
x_1 &= x_0 - o \qquad (3.1) \\
y_1 &= y_0 - a \qquad (3.2)
\end{aligned}
$$

An almost identical method was used to generate the vectors for current. The triangle is then slightly different, and the expression is flipped such that it is now necessary to locate the starting point of the vector instead of the end point. This point is can then be calculated from Equation (3.3) and Equation (3.4).

$$
\begin{aligned}
a &= hcos(\alpha) \\
o &= hsin(\alpha) \\
x_0 &= x_1 - o \qquad (3.3) \\
y_0 &= y_1 - a \qquad (3.4)
\end{aligned}
$$

With the mathematical expressions for the weather vectors in place, the vectors must be drawn on the map. Shapely functionality is used to draw the wind and wave vectors onto the existing ENC. Functionality for drawing both single weather vectors and entire vector fields were created. The path finding work in this thesis is not based on vector fields composed of force vectors. However, vector fields illustrate the wind, wave and current conditions present in the operating area, and are useful to the observer and developer. It is therefore valuable to generate these fields and draw them onto the figures.

3.4 Coordinate conversion

Multiple coordinate systems are used for navigation on the surface of the Earth. The most well-known of these would be the World Geodetic System (WGS) employed by the Global Positioning System. Here, coordinates are represented by latitude and longitude with units in degrees. These coordinates are generated by mapping a spherical coordinate system onto an ellipsoid model of the Earth[58]. Another commonly used coordinate system is the Universal Transverse Mercator (UTM) coordinate system[59]. This system divides the Earth into 60 almost equally sized zones, each 6 ° wide. These zones range from the West coast of Alaska to the East coast of Siberia. The UTM system does not distinguish between the ellipsoid model of the Earth and the Earth itself. Which coordinate system is optimal will depend on the application area.

The SeaCharts API uses the UTM coordinate system for generating shape-files and ENC objects. When using the UTM system, the UTM zone must be known beforehand to make the coordinates unique. Mainland Norway alone spans five of these zones, from 32N to 36N. It should be noted that not all UTM zones follow the

6 ° width rule. For example, all the zones spanning the Norwegian mainland have been modified somewhat. Zone 32N for instance, which is the primary operating area of this thesis' work, is an exception to the rule. This zone has been extended Westwards so it encompasses all of Southern Norway, as illustrated in Figure 3.5. It is therefore important to keep track of which zone overlaps with the current operating area.

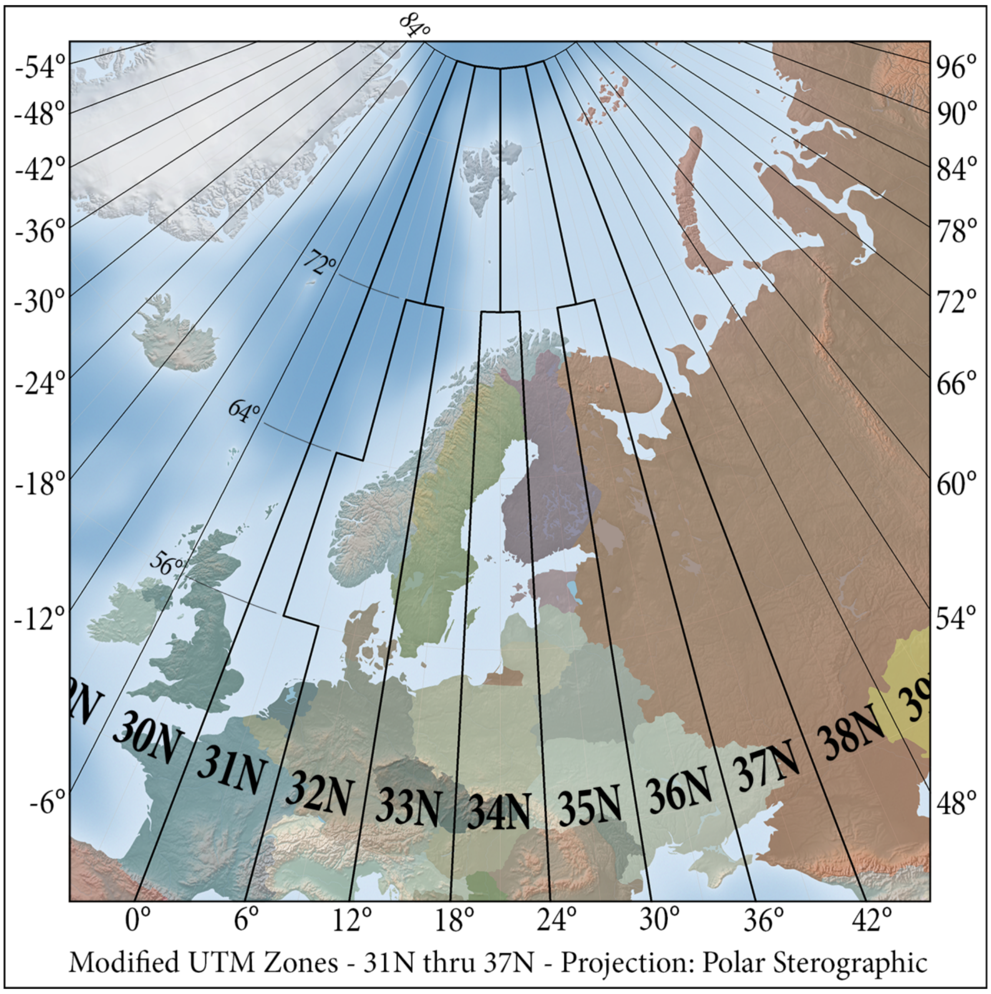


Figure 3.5: UTM zones spanning Norway[60].

Current and wave forecasts are collected at specific geographic points from the MET APIs, and these points have to be indexed using WGS coordinates. This is unfortunate, since conversion between WGS and UTM coordinates is not trivial, and requires lookup-tables and mathematical projection. Luckily there is a project dedicated to solving this problem, the PROJ project[61]. PROJ is a library for converting between different map projection methods. It conveniently has a Python wrapper called *pyproj*. A conversion function between the two coordinate systems was created using this wrapper. This allows for interaction between the SeaChart and MET APIs without having to manually translate between coordinate systems.

3.5 Cost function

Now that the infrastructure to collect weather data is in place, the cost of movement between grid cells needs to be recalculated. Cost of movement should now reflect weather parameters.

3.5.1 Existing cost function

In the project thesis work, the cost of moving from one cell to its neighbor is the result of a calculation which takes into consideration both the direction of movement and the proximity to the goal. The *neighbornodes* function seen in Algorithm 2 assigns a cost to a node based on whether the node is positioned normally or diagonally relative to the current node. The cost of moving diagonally is set slightly higher than moving normally since it represents a slightly longer path. Then, the heuristic is added to the existing cost, meaning movements which decrease the distance to the goal will be favored.

3.5.2 Waves

Waves are the primary source of propulsion for the AutoNaut. It is therefore not desirable to navigate in completely still waters, as such conditions would render the AutoNaut essentially adrift. On the other hand, the risk of capsizing increases in very tall waves.

During the capsizing which occurred in April 2021, waves reached around 4m in height[62]. Had the AutoNaut been able to better align itself with the waves during this time, it might not have capsized at all. A proper way to penalize both wave height and angles needs to be developed. A cost function based on both the incident angle and magnitude of the waves should be created. The result of this function should be added to the cost of movement already present in the A* algorithm.

Wave angles were accounted for first. The wave angle cost was based on the assumption that capsizing is primarily caused by motion in roll caused by external forces[62]. In the case of waves, a vessel will experience periodic motion in roll when waves impart energy on the vessel's hull. Generalized, the vessel will be most sensitive to roll in a beam sea, and thus most at risk. A cost function should therefore take the highest value when the waves are broadside, meaning the difference between the direction of travel and wave direction is 90°. The least hazardous scenario is assumed to be when there is little to no roll motion due to waves. This occurs in a head sea or following sea. In these cases, the cost function should take its minimum value. The absolute value of the difference between the sines of these two angles is such a function. This must then be multiplied by some gain in order to scale the cost with the other cost parameters.

Wave height also plays a major role in safe navigation at sea. As already mentioned, it is unfavorable for the AutoNaut to operate in both too calm and too rough seas. Nonetheless, it is better to be adrift waiting for enough waves to continue, than to capsize as in described in [22]. A discreet cost gain which takes on a different value as it reaches certain wave height thresholds was created to capture this. This gain is written out in Equation (3.5).

$$K_h = \begin{cases} K_{h_-} & \text{if } h < h_- \\ 0 & \text{if } h_- < h < h_+ \\ K_{h_+} & \text{otherwise} \end{cases} \tag{3.5}$$

The cost of both wave height and angle was then summarized into one wave penalty. Equation (3.6) is the total wave cost, the sum of both the wave angle penalty and the wave height penalty.

$$C_{waves} = K_{\phi,waves}\left|sin(\phi_o) - sin(\phi_{waves})\right| + K_h h \tag{3.6}$$

In Equation (3.6), $K_{\phi,waves}$ and K_h are the wave angle and height gains, respectively. ϕ_o represents the ownship heading, while ϕ_{waves} represent the direction of the waves in the global frame. h is the wave height, and should not be confused with the heuristic mentioned in Section 3.2.

3.5.3 Wind

The wind's angle of attack also plays a significant role in a ship's movement, as can be seen in Figure 3.6. Wind perpendicular to the ship's broadside, or wind parallel to the ship's movement, will only work to move the ship sideways or longitudinally, respectively[63]. This can be seen in the charts for longitudinal and side force constants seen in Figure 3.7. Here, the longitudinal wind constant is zero for a 90° incident angle. The longitudinal wind effect will be at its maximum in tailwind, i.e. in 180° wind, and its minimum will be in direct headwind, or 0°. Conversely, the side-force will be at its peak with a 90° angle of attack. From the different craft that are represented in the graphs, these statements appear to generalize to most ship-like watercraft, although the ones exemplified in the charts are considerably larger than the AutoNaut. Nonetheless, they are assumed to apply to the AutoNaut as well.

When the wind's angle of attack is not exactly parallel or perpendicular to the ship's beam, the ship will experience movement in yaw and roll. This is illustrated by the bottom figures in Figure 3.7. This yaw moment may cause the ship to turn, causing it to veer off course. The roll moment will cause the vessel to rock back and forth. These moments are illustrated in Figure 3.6. The bottom left graph in Figure 3.7 implies that closer the incident angle is to either 45° or 135°, the more the vessel will turn. The yaw motion is smallest for incident angles of 0° and 180°, and the turning motion is minimized near these angles. By looking at the bottom right graph, it can be observed that the rolling moment of the vessel will be minimized around these angles as well. This is in line with the cost philosophy from Section 3.5.2.

Since the vessel experiences similar movements in the presence of wind as in the presence of waves, the cost function for wind took almost the same form. However, the AutoNaut is not powered by wind, so the absence of wind is not penalized. The resulting wind cost function can be seen in Equation (3.7). It should be noted that the wind is generally the main generator of surface waves[64]. They are nonetheless assumed to be separate phenomena in this work.

$$C_{wind} = K_{\phi,wind}|sin(\phi_o) - sin(\phi_{wind})| + K_{v,wind}v_{wind} \tag{3.7}$$

$K_{\phi,wind}$ and $K_{v,wind}$ in Equation (3.7) are the wind angle and speed gains, respectively. ϕ_{waves} is wind direction, and v_{wind} is the wind speed.

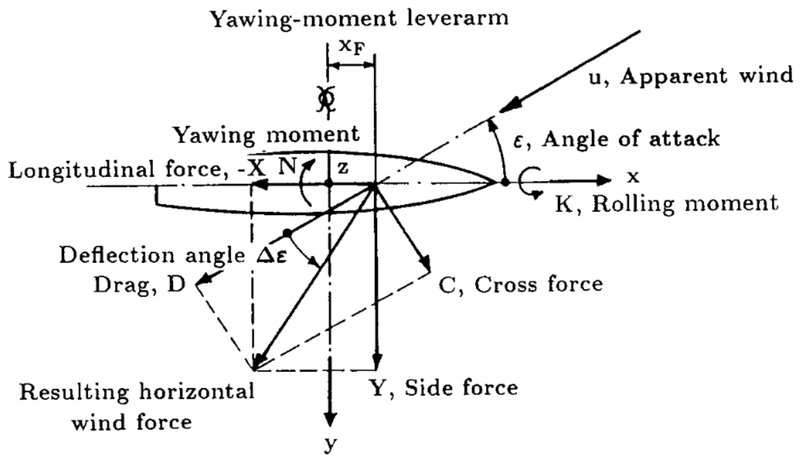


Figure 3.6: Ship moments[63].

3.5.4 Current

Wind forces act above a ship's waterline, while wave forces act both above and below the waterline. Current forces, on the other hand, are only present below the waterline. Nonetheless, current forces can still be modelled as a vector acting on a ship's body. It was therefore assumed that current could be modelled as a vector in the same way as wind in Figure 3.6, with both a longitudinal, side and cross force. Furthermore, it was assumed that the linear and rotational forces would cause the craft to act in a similar manner. Therefore, current penalties were handled in the same way as wind and waves. The cost function for waves was put on an identical form, and can be seen in Equation (3.8).

$$C_{current} = K_{\phi,current}|sin(\phi_o) - sin(\phi_{current})| + K_{v,current}v_{current} \tag{3.8}$$

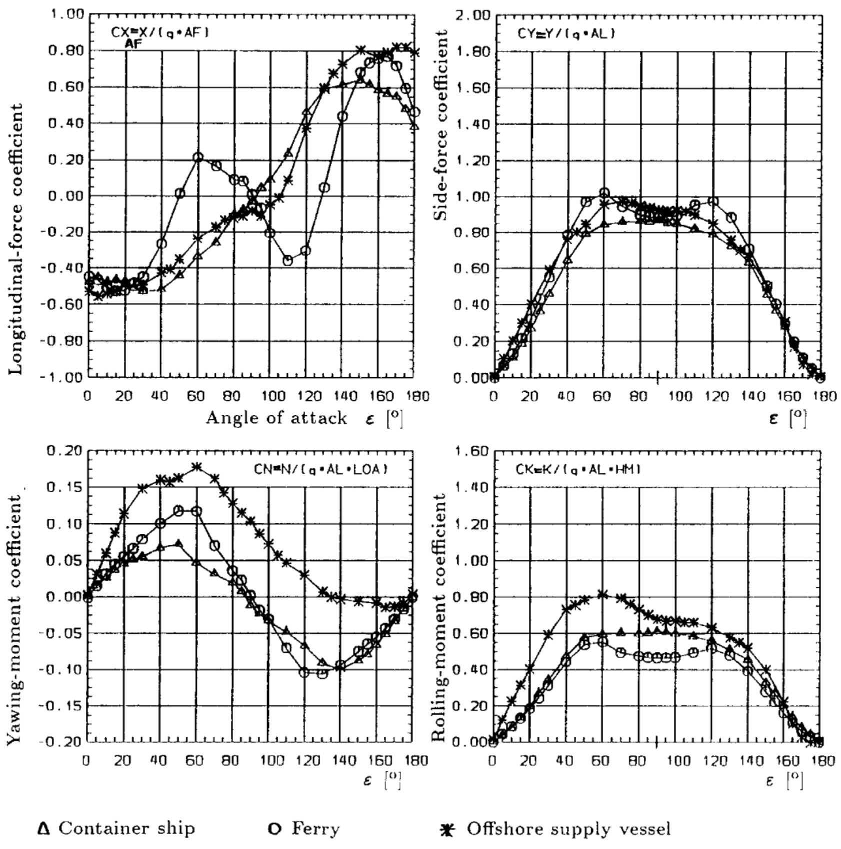


Figure 3.7: Wind coefficients.[63]

$K_{\phi,current}$ and $K_{v,current}$ in Equation (3.8) are the current angle and speed gains, respectively. $\phi_{current}$ is current direction, and $v_{current}$ is the current speed.

3.5.5 New cost function

A total cost function with all weather modes included was created according to Equation (3.9).

$$\begin{aligned} C_{total} &= C_m + C_{weather} \\ C_{weather} &= C_{wind} + C_{waves} + C_{current} \end{aligned} \tag{3.9}$$

C_m in Equation (3.9) represents the cost of distance, which depends on whether the movement is normal or diagonal.

3.5.6 Download frequency

Initial testing of the MET APIs involved gathering weather data each time the A* algorithm calculated the cost of moving to a neighboring node. Collecting wind, wave and current data for each node in the neighboring nodes drastically increased the program runtime, even without performing any operations on the data itself.

Downloading weather data requires sending queries to MET, waiting for the specified data in return, and then picking out the desired data from the resulting JSON file. Doing this at every neighboring node caused runtimes to balloon to a scale of minutes even for maps as small as 3km^2. Collection of weather data at every point of interest therefore proved to be untenable as a solution.

Another possible solution was to only download weather once, at the beginning of program execution. This data could then be stored until it was necessary to calculate weather-related costs. The data for wind, waves and current was stored in a grid by reusing functionality from the project work. This grid was then available whenever weather data needed to be accessed. Since this method could be completed in a reasonable time, it was chosen as the method for forecast collection going forward.

3.5.7 Forecast time

Weather forecasts are available in hourly readings. By the time the AutoNaut has reached a location some distance away, the forecasts may already be outdated. This information should be incorporated into the path planner. In essence, weather in locations further away from the vessel should also be captured further away in time. There is little point in capturing weather data in a certain area in the present moment if the AutoNaut will not reach this area for several hours.

An estimate of how long the AutoNaut will take to reach the different cells of the weather grid was needed. If each cell is associated with a value denoting the approximate time in hours it will take the AutoNaut to reach this cell, it is possible to know how far into the future the system should look for data.

A good starting point would then be the determination of speed based on position and time. Using the velocity formula, it is possible to use the time estimate Equation (3.10). Here, $\hat{t}$ is the time estimate, $\hat{v}$ is the velocity estimate, and Δp is the known displacement in position of the vessel. Since weather data comes in hourly readings, rounding $\hat{t}$ to the nearest hour will then provide the required time information for data collection.

$$\begin{aligned} v &= \frac{\Delta p}{t} \\ \implies \hat{t} &= \frac{\Delta p}{\hat{v}} \end{aligned} \tag{3.10}$$

Now, determining a speed predictor $\hat{v}$ remains. This is not trivial, as velocity estimation is a multifaceted problem. In fact, this problem was the basis for the work related to an entire master's thesis written by Heggernes from 2021[17]. As there was insufficient time to incorporate Heggernes' work into this thesis, a less complicated speed estimate was created. After reviewing several mission logs from Neptus, the AutoNaut appears to travel around 1 knot on average in most weather conditions. Therefore, $\hat{v}$ was set to $\hat{v} = 1knot \approx 0.514m/s$.

A weather forecast grid was then generated and projected onto the already existing sea chart. This can be seen in Figure 3.8, where green represents the starting node, and yellow, orange and red cells represent increasing distance away from the start node. The grid cell size was set to 2.5 km by 2.5 km, the resolution of the weather data. With a time estimate based on the difference in grid indexes in place, the grid itself is generated according to Algorithm 4. This algorithm uses an already constructed base grid with resolution equal to the weather data resolution, as well as the start node to calculate the distance to each cell and the straight-line angle needed to travel there.

3.5.8 Compact data

MET have made it possible to import a more "bare-bones" version the data for some of their APIs. Using the keyword *compact* while querying limits the downloaded data to only include the most commonly used forecasts. This works for wind data, as wind angle and speed are both commonly used forecasts. Oceanforecast does not support the compact functionality.

3.6 Safe distance to land

This section will cover the topic of buffer polygons. These buffers will then be used to find a minimum safe distance to land for the AutoNaut.

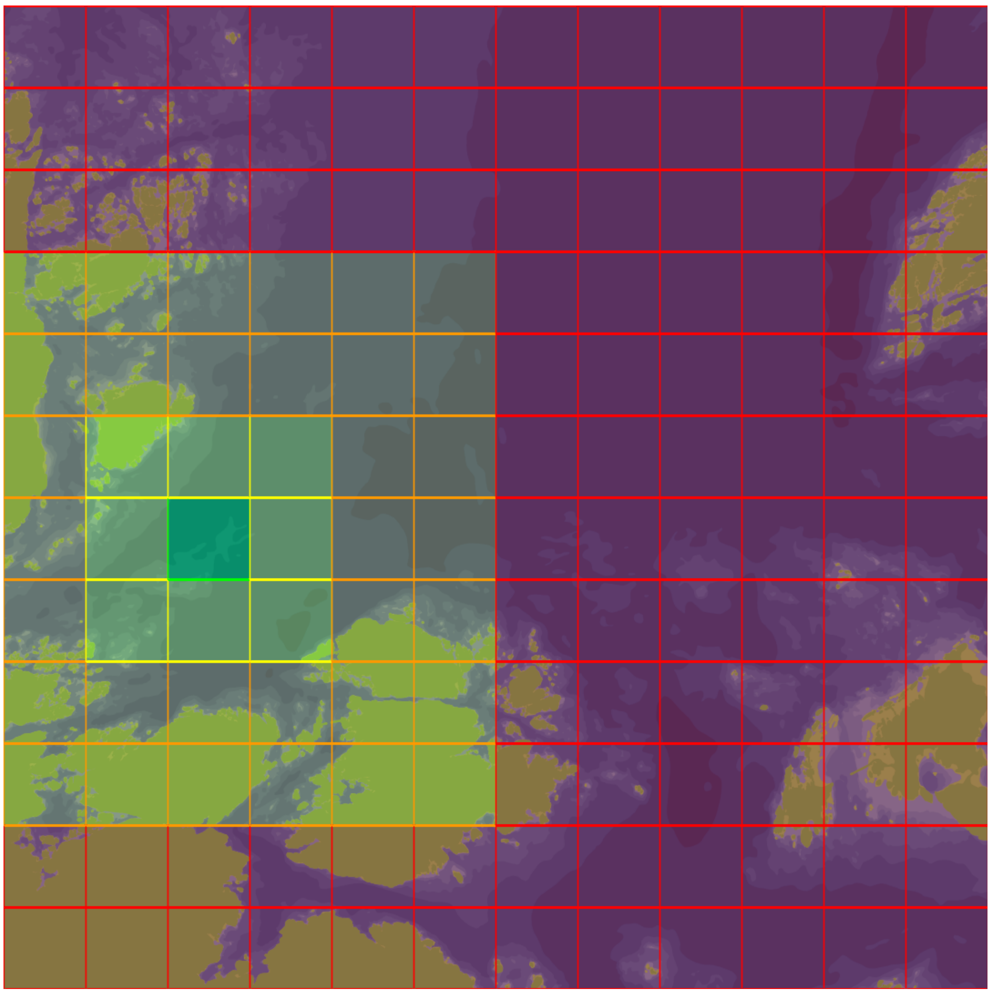


Figure 3.8: Distance grid for forecast time indexing.

Algorithm 4 Pseudocode for constructing weather grid.

```
1: base_grid ← base_grid
2: weather_grid ← empty list
3: **for** row in base_grid **do**
4:     **for** node in row **do**
5:         $\Delta x \leftarrow |start\_node.x - node.x|$
6:         $\Delta y \leftarrow |start\_node.y - node.y|$
7:         maximum_distance ← max($\Delta x$, $\Delta y$)
8:         $\hat{t}$ ← integer($K_t maximum\_distance$)
9:         weather_data ← get_weather_data(coordinates, $\hat{t}$)
10:        append weather_data to weather_grid
11:    **end for**
12: **end for**
13: return weather_grid
```

3.6.1 Buffer polygons

In the project thesis work, the distance to obstacles was not considered. This meant that any path circumventing said obstacles was considered valid, no matter how close the path actually approached them. For instance, a path would be considered just as feasible irrespective of whether it kept a 10km distance from an island, or whether the path avoided the island by a centimeter.

The grid cells that make up the graph are treated as obstacles if even a sliver of land or shallows intersect with the cell. This means that if a grid cell of dimensions 200m by 200m has a minuscule land formation at its center, it would still be considered an obstacle, and any path would have to circumvent it by at least 100m. But no such guarantee is given for land and shallows that are just barely within the limits of a grid cell, with open water bordering them. In theory, it would then be possible for a piece of rock to be centimeters away from a valid path, as long as it just barely avoids intersecting with a water cell. This was a major safety concern, as it gave no guarantee for the actual distance the vessel is from land. A workaround therefore had to be found.

Preferably, the AutoNaut should always travel at the very least hundreds of meters away from land and shallows. This is due to its relatively slow actuation, and for a safety margin in case of changing weather or system malfunction. If the craft strays too close to danger moments, there might not be time to make it correct its course. In the worst-case scenario, there might not even be time to launch a rescue mission.

The Shapely library provides a tool which solves the problem of proximity to land and shallows. Within it is the geometry package, which supports buffer polygons. Buffer polygons are made from other already existing polygons, but with a given buffer distance from all vertices of said polygon. Creating such polygons from the already existing land and shore polygons then guarantee that a path will never be created closer than the specified distance.

3.6.2 Minimum time before grounding

Harsh weather, system malfunction and other hazards might drive the AutoNaut towards land. It is desirable to have plenty of time to rectify the problem, either by intervening manually or remotely. The minimum amount of time should be specified by the operator, and should be adjustable for different operating scenarios. This minimum amount of time can then be used to calculate the buffer size to achieve the specified intervention time. A good engineering practice would be to base this calculation on a worst-case scenario.

In a worst-case scenario, the AutoNaut will find itself heading straight towards land at its maximum velocity. Assuming it is desirable to have at least 30 minutes of time to intervene, and the AutoNaut is travelling at its specified maximum speed of 3 knots, the buffer radius r_b would have to be at least 2772m according to Equation (3.11).

$$\begin{aligned} r_b &= v_{max} t_{min} \\ &= 3knots * 30min \\ &\approx 1.54\frac{m}{s} * 1800s \\ &= 2772m \end{aligned} \tag{3.11}$$

Because of the aforementioned calculation, the buffer radius should be set to 2.8km under normal operation. This might not always be possible however, depending on where the AutoNaut is operating. Narrow inlets and dense archipelagos are normally not the AutoNaut's typical operating area, but that might change in the future. It should be noted that the buffer zone was significantly reduced during testing and tuning in this thesis, and was usually no larger than 500m. This is due to the exorbitant time it would take to generate maps large enough for a 2.8km buffer zone to not occupy too much of the map.

3.7 Path pruning

Paths in a grid-based path finding scheme are generated based on movement from one grid cell to another. Therefore, the complete path between a starting point and an end point will depend on the resolution of the grid itself. In practice, this implies that the higher the resolution, the more waypoints will be generated. This is fine for the machine to run, as waypoint generation does not take an inordinate amount of time to compute. However, these waypoints then have to be transferred to the vessel itself over Neptus. For now, these waypoints have to be entered manually by an operator. If the number of manual entries is on a scale of hundreds, or even thousands, the process becomes wholly impractical for the operator.

Large amounts of waypoints can also lead to network congestion if sent over wireless. This is especially undesirable in remote areas where bandwidth is a scarce resource. In addition, the AutoNaut is slow to actuate, and might be unable to reach each waypoint fast enough in strong wind and current. For exploratory missions, which is after all the primary objective of the craft, pinpoint precision in location is not important.

All the reasons mentioned made it apparent that a reduction in the number of waypoints sent to the vessel was necessary. A pruning function had to be created to solve the problem of bloated paths. This function should take in the current path, and attempt to skip waypoints by creating lines between existing waypoints. These lines should then be investigated for intersection with land, and waypoints between the extremities of the lines should be removed from the path.

The AutoNaut mainly operates in the open ocean, where land and shallows are rare. Some stretches of ocean might consist of tens, hundreds or even thousands of kilometers without a single land formation. Removing all waypoints in between start and goal on such a map would just leave the original two, which is counterproductive. To still maintain a reasonable amount of waypoints, the pruning function needs to accept a user-specified number representing the number of points the algorithm will attempt to trim away between each waypoint in the final path.

The first attempt at creating a pruning algorithm was based on skipping a given number of waypoints and removing the points in between. This algorithm generated lines between a starting point and the point a given distance away, and checked for intersection between the line and land or shore. If it was not possible to skip to the given waypoint without a collision, the algorithm would not remove the waypoints in between and would instead skip to the next section of the path. This algorithm used to generate a simplified path can be seen as the pseudocode written out in Algorithm 5.

Algorithm 5 Path pruning algorithm pseudocode

```
1: pruned_path ← path
2: $i \leftarrow 0$
3: while $i$ < length(pruned_path) - skip - 1 do
4:     start_point = pruned_path[$i$].coordinates()
5:     end_point = pruned_path[$i$ + skip].coordinates()
6:     line = LineObject(start_point, end_point)
7:     if Line not intersect land or shore then
8:         for j in range($i+1$, $i+skip-1$) do
9:             remove pruned_path[j] from pruned_path
10:        end for
11:    end if
12:    $i \leftarrow i+1$
13: end while
14: return pruned_path
```

Another, more involved algorithm for path reduction was also created. This one, conceptualized in Algorithm 6, is dubbed "recursive segmentation" and is based on sectioning up the original path into a given number of sub-paths, or segments. A maximum skipping distance is provided as in the first algorithm, and the number of sub-paths depends on the maximum skipping distance. Like in the first algorithm, lines are generated from these sub-paths, and these are checked for intersection with land and shore polygons. If, however, an intersection occurs, the algorithm does not simply give up and move on. Now, the algorithm is ran recursively on the segment in question, with the skipping parameter reduced by a factor of 2. Sub-paths are thus broken down further, and the process is repeated until a non-intersecting line is found. If no line is found, no waypoints will be removed, and the original waypoints are allowed to stand. Finally, the pruned segments are reassembled into a pruned path.

Algorithm 6 Recursive path pruning algorithm pseudocode

```
1: pruned_path ← path
2: min_seg_num ← ⌈pruned_path/max_skip)⌉
3: segment_list ← empty list
4: for i in range(min_seg_num) do
5:     segment ← pruned_path[$i * max\_skip{:}i * max\_skip + max\_skip + 1$]
6:     append segment to segment_list
7: end for
8: reduced_segment_list ← empty list
9: for segment in segment_list do
10:     line = LineObject(segment_start_point, segment_end_point)
11:     if max_skip < 2 then
12:         reduced_segment ← segment
13:     else if line not intersect land or shore then
14:         reduced_segment ← [segment[0], segment[-1]]
15:     else
16:         reduced_segment ← prune(segment with max_skip ← ⌈max_skip/2⌉)
17:     end if
18:     append reduced_segment to reduced_segment_list
19: end for
20: return reduced_segment_list
```

3.8 System workflow

Figure 3.9 illustrates the condensed workflow from the project thesis discussed in Section 1.3. Here, an ENC object is generated from geodatabase files using the SeaCharts API, and a grid of node objects are created using the Shapely API. The A* path planning algorithm then uses the information from the ENC object and node grid to generate waypoints. With the inclusion of weather data, the workflow becomes Figure 3.10. In this scheme, weather data is organized into a weather grid, which is also fed into the path planning algorithm along with the ENC object and the Node object grid to create a "raw", unprocessed path. The path pruning algorithm is then used to simplify the path, generating the final waypoints. These can then be issued to the AutoNaut, either through Neptus or directly to the vessel itself.

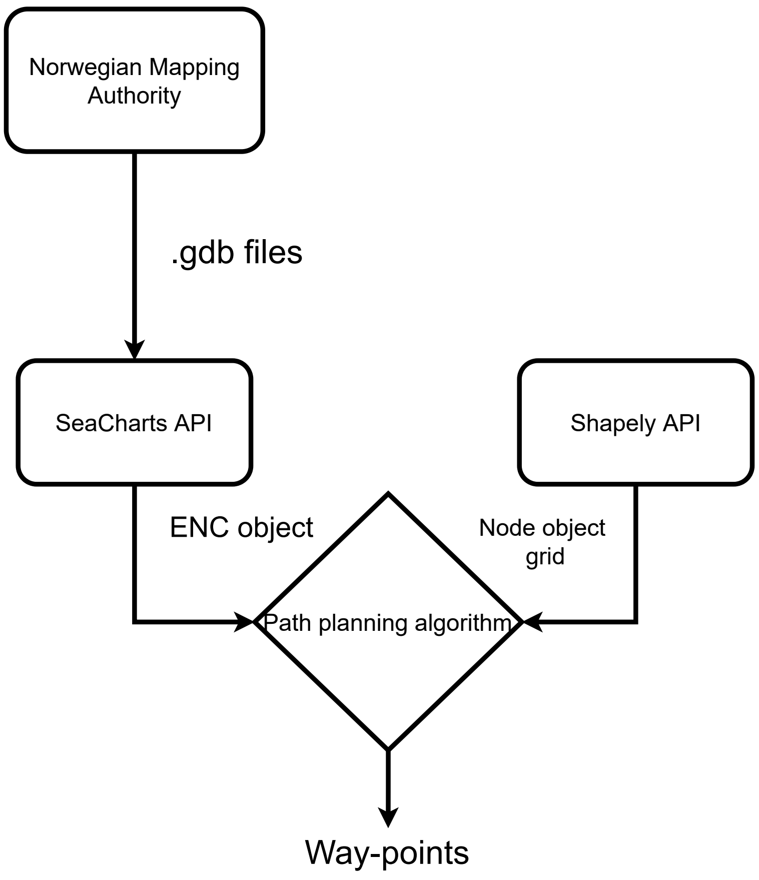


Figure 3.9: Original flow diagram from project thesis. Generated using the web tool diagrams.net[65].

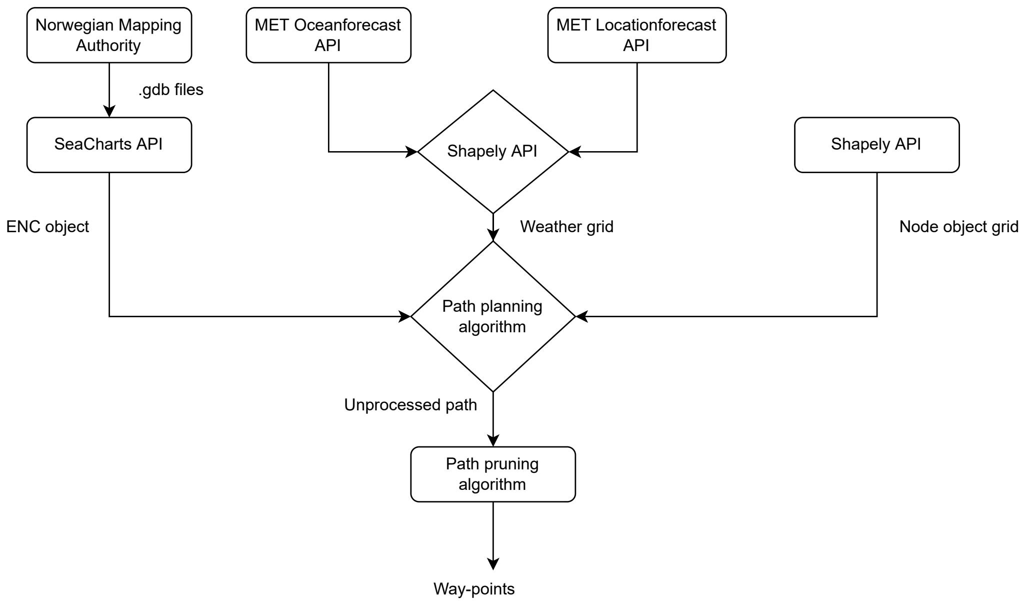


Figure 3.10: New system flow diagram. Generated using the web tool diagrams.net[65].

Chapter 4

Results

This chapter will document the results of running the path planner on several real charts, all of them in the waters outside of Trondheim, Norway. Different weather conditions, tuning parameters, buffer zones and path pruning schemes will be illustrated.

Please note that the results shown here are all generated on a ThinkPad T480 running Ubuntu 18.04.6 LTS. The machine carries an Intel® Core™ i7-8550U CPU @ 1.80GHz × 8 processor, an Intel® UHD Graphics 620 (KBL GT2) GPU and has 31.2 GB of RAM. Results may differ significantly when run on other systems.

4.1 Vector fields for weather forces

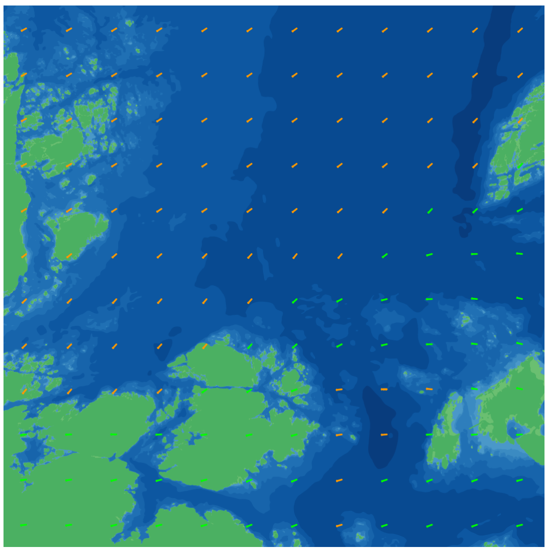
Vector fields generated for wind, waves and current can be seen in Figure 4.1. In this map, live weather data was collected and visualized in three ENCs. For wind, seen in Figure 4.1a, wind speeds above 5 m/s are depicted in orange, while wind speeds below this are shown in green. Figure 4.1b contains wave vectors, with waves shorter than 1 m depicted in green. Figure 4.1c illustrates a current vector grid. Currents slower than 0.3 m/s are depicted in cyan.

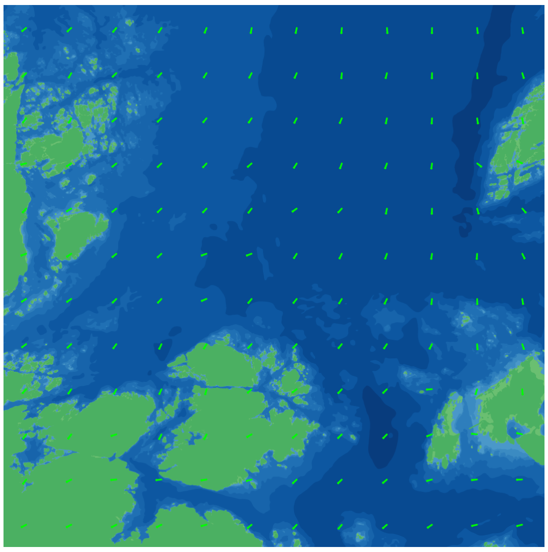
4.2 Buffer polygons

Figure 4.2 illustrates two sea charts containing a small island and a port environment. The path planning algorithm created in [1] was instructed to create a path circumventing the island, and the resulting path can be seen in Figure 4.2a. Then, a buffer of 100m was applied to both the land and shore data on the chart. The algorithm was modified to view the buffers as obstacles, and again instructed to create a path between the same two points, resulting in Figure 4.2b.

4.3 Cost tuning

Each weather parameter was tuned individually using fabricated weather data.


(a) Wind vector field.


(b) Wave vector field.

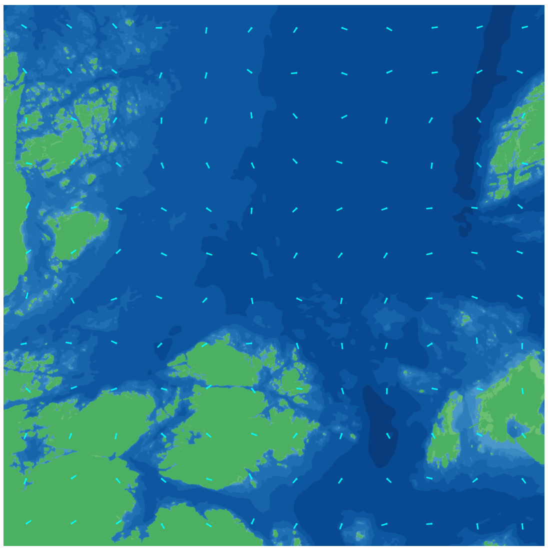
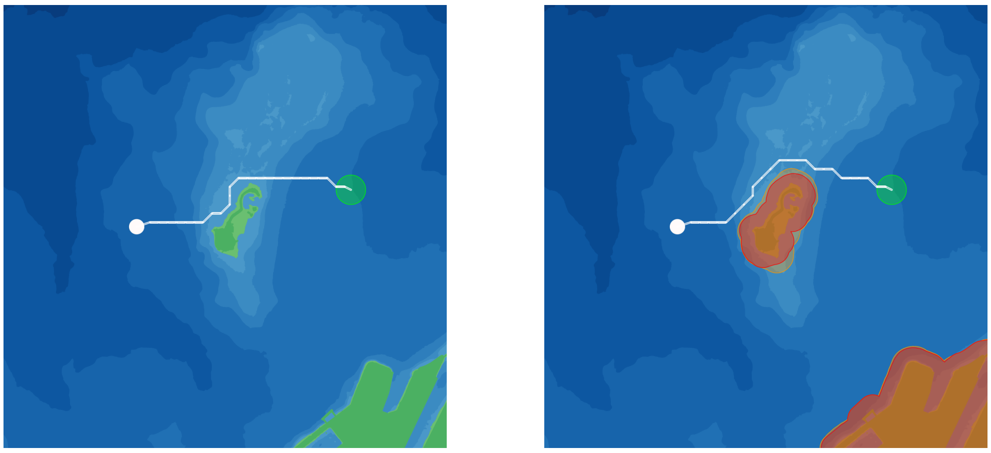

(c) Current vector field.

Figure 4.1: Vector fields of weather forces.



(a) Path without buffer. **(b)** Path with a buffer of 100m.

Figure 4.2: Two paths with the same start and end points, one with and one without the use of buffer polygons.

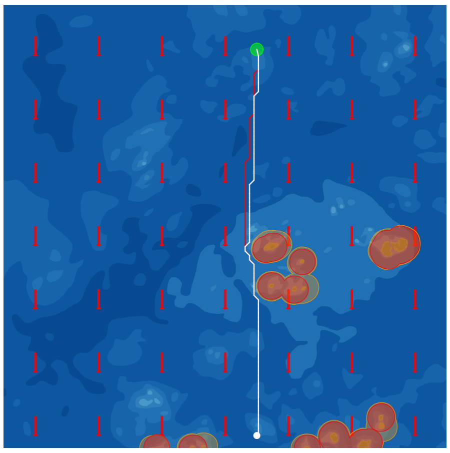
4.3.1 Tuning weather cost gains

The maps used for tuning wave angle cost are all in the same area, in the ocean bordering an archipelago north of Frohavet. These maps are 7km^2 in size, have a grid resolution of 100 by 100 cells and a buffer size of 200m for both land and shore. The maps used to tune for wave height cost is 3km^2 in size, with a resolution of 50x50 and 200m buffer. The white circle represents the starting point, and the green circle represents the end point for all generated paths.

Figure 4.3 shows two paths running directly south to north. The red path in Figure 4.3a represents the control; the original path planner with $C_d = 1.1$ and $C_n = 1.0$ and all weather costs set to 0. The red path in Figure 4.3b was generated using $C_d = 0.55$ and $C_n = 0.5$ and $K_{\phi,waves} = 0.5$.

The red arrows in Figure 4.3 represent waves of 12m in height, in this case a head sea. Both paths end up being approximately 6.2km in length. The script used to generate the individual figures, including node and weather grid generation, generation of the two paths, as well as plotting, took approximately 8s to run. Next, two different wave angle parameters were tested in beam sea. The results can be seen in Figure 4.4. Two different values for the wave height parameter K_h were then tested, as can be observed Figure 4.5. An identical approach was used for tuning the wind and current angle and speed gains, respectively.[1] The final values for all cost gains can be seen in Table 4.1. After values were found for every tuning parameter, the system was tested with real weather data. The result of this can be seen in Figure 4.6.

[1]Figures omitted for the sake of brevity.


(a) High wave angle penalty, normal distance penalty. Red path: $C_n = 1.0, C_d = 1.1$, white path: $C_n = 0.5, C_d = 0.5, K_{\phi,waves} = 0.5$.

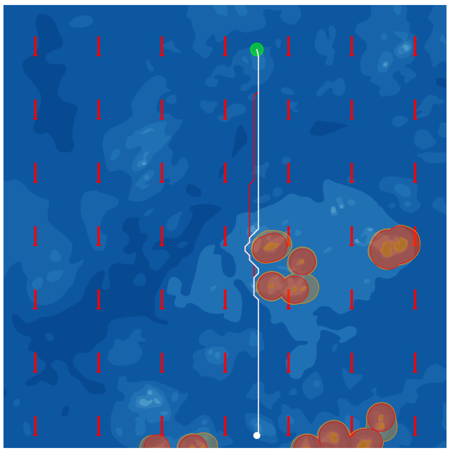
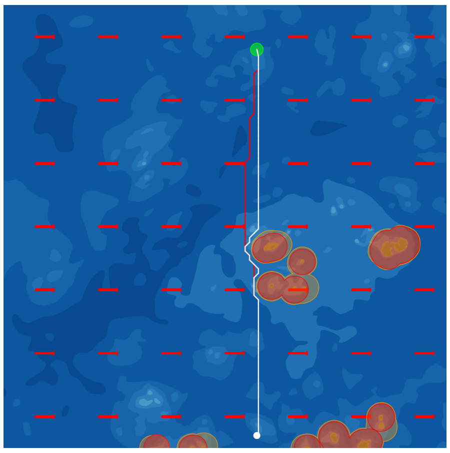

(b) High wave angle penalty, low distance penalty. Red path: $C_n = 1.0, C_d = 1.1$, white path: $C_n = 0.05, C_d = 0.055, K_{\phi,waves} = 0.5$.

Figure 4.3: Paths for tuning wave angle parameter $K_{\phi,waves}$ in head-on waves.


(a) Red path: $C_n = 1, C_d = 1.1$, white path: $C_n = C_d = 0.5, K_{\phi,waves} = 0.2$.

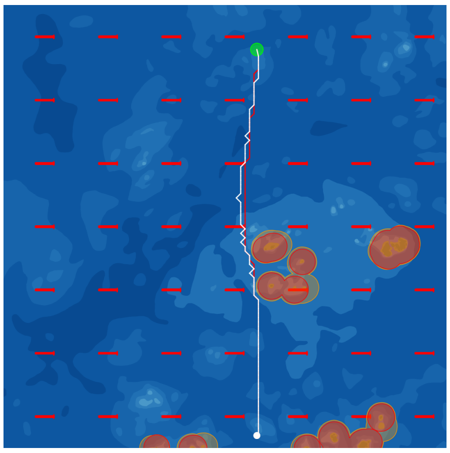
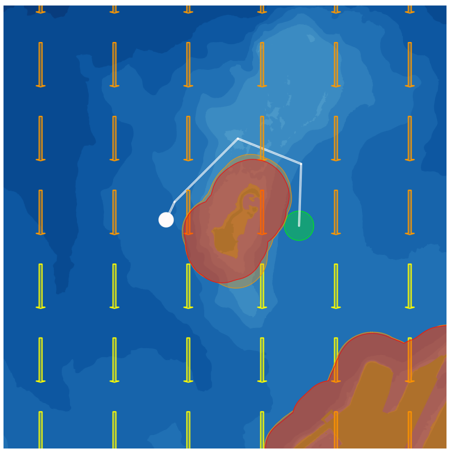
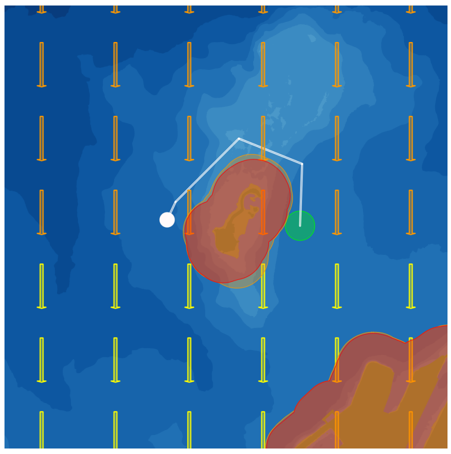

(b) Red path: $C_n = 1, C_d = 1.1$, white path: $C_n = C_d = K_{\phi,waves} = 0.5$.

Figure 4.4: Paths for tuning wave angle parameter $K_{\phi,waves}$ in broadside waves.



(a) Small wave height gain: $K_h = 0.2$.



(b) Large wave height gain: $K_h = 1.2$.

Figure 4.5: Paths for tuning wave height gain with two different values for K_h. Orange waves are 4m tall, while yellow waves are 2m tall.

Gain	Value
C_n	0.5
C_d	0.55
$K_{\phi,wind}$	0.001
$K_{v,wind}$	0.001
$K_{\phi,waves}$	0.2
$K_{\phi,current}$	0.001
$K_{v,current}$	0.001
K_{h_+}	0.05
K_{h_-}	0.05
h_+	$2.5m$
h_-	$0.1m$

Table 4.1: Tuned cost gains.

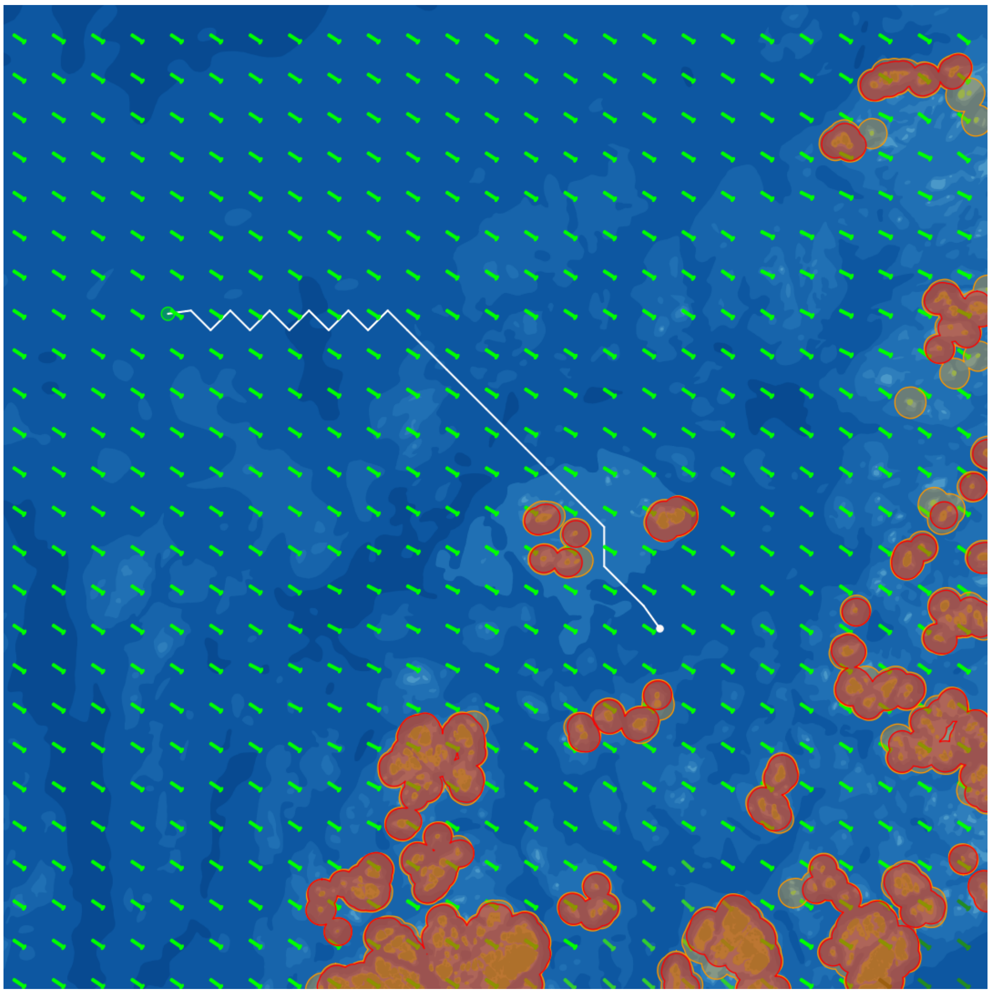
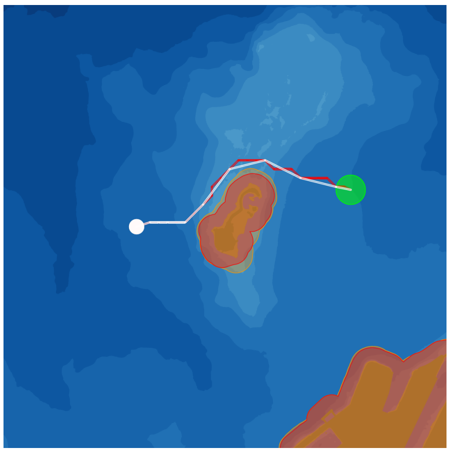
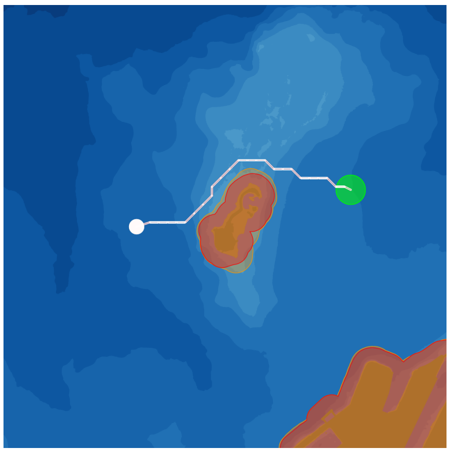


Figure 4.6: Path generated with all parameters set. Real weather data.



(a) Skip-based pruning with skip distance 5.



(b) Skip-based pruning with skip distance 25.

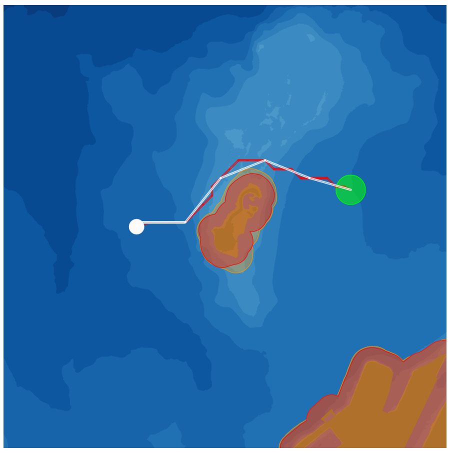
Figure 4.7: Skip-based pruning around Munkholmen island. Original path in red, pruned path in white. Starting point highlighted in white, goal point in green. Grid: 100x100

4.4 Pruning

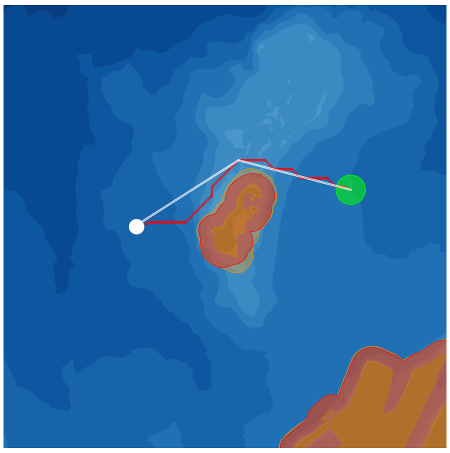
The two path pruning schemes presented in Section 3.7 were applied to identical paths circumventing Munkholmen island off the coast of Trondheim. Two different pruning parameters were given to each pruner. The results of skip-based pruning can be seen in Figure 4.7, and the results of recursive segmentation-based pruning can be seen in Figure 4.8.

4.5 Runtime

A selection of runtimes were logged while generating figures for this dissertation. Table 4.2 shows runtimes with different configurations. The variables were whether or not pruning was utilized, the size of the map, the grid resolution, and whether weather data was collected or not. The sixth column shows the average runtime for each map, with a sample size of 3.



(a) Recursive segmentation-based pruning with maximum segment size 5.



(b) Recursive segmentation-based pruning with maximum segment size 25.

Figure 4.8: Recursive segmentation pruning around Munkholmen island. Original path in red, pruned path in white. Starting point highlighted in white, goal point in green. Grid: 100x100.

	Fig.	Pruning	Map size	Grid res.	Weather	Runtime ($n = 3$)	Notes
1	4.3a	no	7km^2	100x100	dummy	8.2s	no
2	4.3a	no	7km^2	100x100	dummy	6.1s	no vis.
3	4.3b	no	7km^2	100x100	dummy	8.0s	no
4	4.3b	no	7km^2	100x100	dummy	11.4s	new data
5	4.3b	no	7km^2	50x50	dummy	5.9s	no
6	4.4a	no	7km^2	100x100	dummy	8.1s	no
7	4.4b	no	7km^2	100x100	dummy	13.1s	no
8	4.8b	recseg	3km^2	50x50	no	3.8s	no
9	4.5a	no	3km^2	35x35	dummy	2.5s	no
10	4.5a	skip	3km^2	35x35	dummy	2.5s	no
11	4.5a	recseg	3km^2	35x35	dummy	2.5s	no
12	4.5b	recseg	3km^2	35x35	dummy	2.6s	no
13	4.6	no	15km^2	50x50	yes	55.6	no
14	4.6	no	15km^2	50x50	yes	65.7s	new data
15	4.6	no	15km^2	50x50	yes	32.3s	no vis.
16	4.6	no	15km^2	50x50	dummy	23.2s	no

Table 4.2: Runtimes for path planner under different settings.

Chapter 5

Discussion

This chapter will discuss and interpret the results produced in Chapter 4. First, the vector fields for weather data will be covered. Then, tuning of the weather costs, system performance and runtime, as well as the system viability, will be described.

5.1 Weather plotting

Figure 4.1 shows three vector fields with wind, wave and current forecasts, respectively. The magnitude of the weather force is denoted by color, and the direction is denoted by an arrowhead. The online weather service Yr, also hosted by the Norwegian Meteorological Institute, was used to verify the results. The magnitudes from Yr of wind, waves and current, as well as their directions, are consistent with the values in the vector fields.

By observing the plots in Figure 4.1, information about local weather conditions can be deduced. For instance, the figures indicate that vector fields were generated on a moderately calm day. Looking at historical data for the day reveals that the maximum wind speeds reached around 6 m/s out at sea. This is reflected in Figure 4.1a, where areas of open ocean contain mostly orange, which denote wind speeds above 5m/s. This day also saw fairly short waves, with observed waves no taller than 1m. These are illustrated in green. Current was also slow when these grids were generated, with all arrows indicating less than 0.3 m/s current speeds. This information gives an impression of the local conditions at that moment in time. All in all, these conditions point towards somewhere between a *light breeze* and a *gentle breeze* on the Beaufort scale and a *slight* on the Douglas scale[66][67].

As mentioned, a lot of information can be gleamed from a quick glance at the weather charts in Figure 4.1. However, these vector fields for weather forecasts are not immediately useful for path planning, and they are in fact never accessed by the path planner itself. Nonetheless, they are still useful for visualization purposes. For example, they can be generated before a voyage to get a sense of what an optimal path *should* look like, and then compare this to the actual result. They can also be generated during a mission, and the real-time behavior of the vessel can be monitored along with a detailed view of the local conditions.

These figures are also useful for generating reports. Vector field generation has been beneficial in the writing of this paper, and they can also accompany other AutoNaut documentation in the future. Mission logs, theses and presentations stand to benefit from these types of plots. They are a useful tool for informing the reader, and contain a lot of information in a small format.

5.2 Buffers

The two paths created in Figure 4.2 demonstrate the value of buffer polygons. Figure 4.2a illustrates a path generated to circumvent Munkholmen without buffers. If the vessel were to follow the bufferless path, the vessel would at times stray dangerously close to land. On the other hand, Figure 4.2b shows a path generated with a buffer of $100m$. Here, the vessel is guaranteed to be at least $100m$ from land and shore at any given time. For real-world applications, buffers should therefore always be used. The $2.8km$ minimum grounding distance discussed in Section 3.6.2 is a good starting point for large-scale testing.

For operations in denser maps, it will not be possible to use the $2.8km$ minimum distance. Figure 4.6 alone is proof of this. Shoreline exploration, operation in pier and marina environments, as well as archipelago missions, will need much smaller buffer zones if viable paths are to be found. This can be customized by the operator before mission start.

5.3 Weather cost tuning

Tuning of a multivariate problem such as this is not trivial. There are six individual weather data inputs that need to be weighted, since both wind and current speeds, wave height and all their respective angles need to be accounted for. In addition to this, the cost parameters for normal and diagonal movement would also need to be reevaluated since the problem to be solved is now completely different compared to the problem solved in [1]. This section will cover the tuning of the new system with these new cost parameters.

5.3.1 Original tuning parameters

The path planner created in the project thesis was used as a baseline for tuning, with the goal of improving on the original path planner. Some interesting questions to answer are whether weather data actually improved the path planner significantly, and if so, how much.

The two original tuning parameters C_n and C_d had to do with the direction of movement the AutoNaut could take. The costs of normal and diagonal movement were weighted to 1.0 and 1.1, respectively. These values were found through the

Pythagorean principle, and adjusted by trial and error. Now, however, weather also has to be accounted for. Since the costs here are weighted, their actual values do not matter, only their values relative to each other. The total cost of movement when weather is accounted for will be the sum of the normal/diagonal movement costs and the weather costs. C_n and C_d can then be scaled as necessary to allow the weather costs more or less influence over the total cost.

5.3.2 Tuning wave cost

Since waves are perceived to be the most dangerous of the three weather modes, wave cost tuning was performed first. First, the system was tuned for the wave angles, and then for wave height.

In terms of wave angles, a head sea was considered the best-case scenario, and broadside waves were considered to be the worst-case scenario. The wave angle gain was therefore tuned with this in mind, and the results of this can be seen in Figure 4.3 and Figure 4.4. Wave angle tuning was performed independently of wave height tuning, and K_h was therefore set to 0. All tuning was performed to the north of Frohavet, in open ocean with a few islets to make sure the system continued to behave well even in the presence of obstacles. The two tuning parameters to be determined are the gains $K_{\phi,waves}$ corresponding to wave angle, and K_h corresponding to wave height. In addition, the thresholds h_- and h_+ also needed to be set.

$K_{\phi,waves}$ was set to zero for sea states between *smooth* and *moderate* on the Douglas sea scale, that is waves between 0.1m and 2.5m in height[67]. It was seen as unnecessary to account for waves when path finding in relatively calm seas, and penalizing maneuverability in these conditions would cost more in terms of efficiency than it would provide benefits in terms of safety. The same logic applies to tuning for the wave height itself. h_+ was therefore set to 2.5m, and h_- was set to 0.1m. This can be seen in Table 4.1.

Figure 4.3 illustrates paths with two different tuning parameters for 0 incident wave angles. With tall waves such as these, the best course of action would be to travel into the waves with as small an attack angle as possible to avoid motion in roll. A good path for this particular map would then be a rather straight path between the starting point and end point. The red paths serve as the controls, with all weather costs set to zero. The white paths are the new weather-dependent paths.

A first attempt at generating a northbound path can be seen in Figure 4.3a. Here, $K_{\phi,waves} = 0.5$, and normal and diagonal movement costs were 0.5 and 0.55, respectively. The two paths are identical until after the path has circumvented the islets, and even after that only diverge slightly from one another.

The similarity of the paths generated in Figure 4.3a prompted an investigation into how sensitive the path planner was to the angles of the incoming waves, and both diagonal and normal movement costs were lowered. The result was the white path seen in Figure 4.3b. Here, $K_{\phi,waves}$ was set to 0.5. To demonstrate the effects of incoming waves, normal and diagonal costs were lowered by an order of magnitude to 0.005 and 0.0055, respectively. This means that the wave angle is potentially twenty times as influential as C_n and C_d. In this case, the expected outcome is that the red control path and the white new path should still be relatively similar. This is because the angle of the incoming waves should cause the path to be a more or less straight northbound line from start to goal. Indeed, the paths remain similar, with the new white path being the more straight of the two. This demonstrates that the system is capable of discriminating against diagonal maneuvers in the presence of incoming waves. However, it does not provide a conclusive answer to what $K_{\phi,waves}$ should be. More testing was therefore necessary.

Another test was performed, this time in the beam sea visible in Figure 4.4. C_d and C_n were both set to 0.5, so diagonal and normal movement were entirely dependent on the wave angle cost. This time, the path should contain more diagonal movement, since the wave angle cost will be higher for broadside waves. First, $K_{\phi,waves}$ was set to 0.2. The result, seen in Figure 4.4a, is an identical path to the one in Figure 4.3b. The wave angle gain was then increased to 0.5. Results were more promising for this, and the path can be seen in Figure 4.4b. This zigzag behavior is more in line with the expectations, and the vessel will now spend less time in beam sea. This angle gain was therefore kept.

Wave height was ignored in Figure 4.3 and Figure 4.4, and were only shown for illustration purposes. The constant waves used in these figures were set to 12m in height, corresponding to a high 8, or *very high*, on the Douglas sea scale. This was assumed to be worst-case or near worst-case sea state. Waves at sea can and do grow taller than this, but the assumption was that if the AutoNaut encountered such conditions, no amount of tuning would save it. In fact, the waves during the capsizing of the AutoNaut were only around 4m tall. It would be better to attempt to avoid travel in these areas altogether, and this is the desired outcome when tuning K_h.

In a large open stretch of water, there may be several different sea states depending on local conditions. If weather conditions are relatively calm, wave height should be ignored entirely. However, if conditions are too calm, the AutoNaut will be unable to propel itself. If the choice stands between sailing in an area of $4m$ tall waves and $2m$ tall waves, the AutoNaut should choose the area with the shortest waves for optimal safety. Conversely, if the options are travel through areas with $0m$ waves and $1m$ waves, the area with the taller wave height should be selected for optimal efficiency. K_{h_+} and K_{h_-} should have values that reflect this behavior.

Figure 4.5 illustrates a map where dummy wave data has been projected onto a grid. The start and end point are on each side of an island, and the way around it is virtually identical in length for either side. Half of the map consists of $4m$ tall waves, pictured in orange, and the other half consists of $2m$ tall waves pictured in yellow. As mentioned, the desired behavior is for the AutoNaut to choose the southern way around every time, since the conditions here are safer.

Several configurations were tried with different values for K_h, two of which are shown in Figure 4.5. In Figure 4.5a, K_{h_+} was set to 0.2, but the algorithm still selected the northern way around the island. When the wave height gain was set to 1.2 as in Figure 4.5b, the path was identical. Other values for both K_{h_+} and K_{h_-}, both higher and lower were tried, but the result was the same. The algorithm seems entirely unresponsive to the wave height gain.

A definite answer to why the algorithm does not appear to respond to different wave height parameters was never found. It was hypothesized that the wave height cost in Equation (3.6) was overshadowed by the wave angle cost. This notion was discarded, however, as the wave angle cost can at most reach a value of $0.5 * |2| = 1$. When the wave height gain was set to 1.2, the wave height cost should still be $1.2 * 4 = 4.8$ for the $4m$ waves and $1.2 * 2 = 2.4$ for the $2m$ waves. Since an answer was not found, both K_{h_+} and K_{h_-} were set to a relatively low value of 0.05 in anticipation that an answer will be found in the future. This will be discussed in Chapter 6.

5.3.3 Tuning current and wind costs

After wave costs, wind and current costs were tuned using an almost identical process. Dummy data was used for both magnitude and direction of current and wind. The only difference between the tuning scheme for waves and the tuning schemes for current and wind was the omission of the thresholds present in Equation (3.5). The paths generated by tuning these parameters were predictably almost identical to the paths seen in Figure 4.3 and Figure 4.4, and were therefore omitted. Again, the system was entirely unresponsive to different gains for current and wind speeds, and results are again inconclusive.

Wind and current's effects on the vessel were seen as less safety-critical than wave effects. Therefore, the gains for current and wind angles and speeds were set lower than the gains for wave angles and heights. The wind angle and speed costs, as well as the current angle and speed costs, were set to the minuscule value of 0.001. Proper values for them can be found when an improved tuning scheme is devised. All the gains for wind and current costs can be seen in Table 4.1.

5.4 Performance

A test was conducted to investigate the performance of the system after tuning. With all tuning gains set, the path planner was tested on the relatively large map seen in Figure 4.6. In this scenario, real weather data for waves, current and wind was collected and taken into account by the system. Only the wave vectors were drawn, to increase readability. This was a calm day with no waves above $0.5m$, and so the threshold values h_+ and h_- were discarded in order to capture behavior related to wave angles.

A feasible path was found between the white starting point and the green end point, and the algorithm managed to avoid both land and shallows. Furthermore, the path planner avoids the $200m$ buffer zones applied to all obstacles. The algorithm seems to favor diagonal movement throughout the path, especially towards the end. In the first two thirds of the path, the vessel aligns itself to sail in a head sea. In the final stretch of the run, zigzag behavior starts to occur to avoid sailing in a quartering sea. The result of this, however, is a beam sea for half of the zigzag behavior, which is not optimal. There may be an error in the cost function created in Section 3.5. This will have to be investigated further, and will be covered in Chapter 6.

5.5 Pruning

Results from the testing of both pruning algorithms are promising. Both the skip-based approach and the recursive segmentation pruning were able to successfully simplify the red paths generated in Figure 4.7 and Figure 4.8. Where the two differ is in their behavior under differing skip distances.

For the skip-based pruning illustrated in Figure 4.7, two different skip distances were attempted. In Figure 4.7a, a skip distance of 5 waypoints was selected. The pruning algorithm managed to trim away 5 waypoints at every step. Then the skip distance was increased to 25, as seen in Figure 4.7b. Here, it was not able to trim away any waypoints, and the trimmed and original paths were identical. This is because the skip-based algorithm will just move on to the next step of the path when it is unable to perform a skip. In that regard, the algorithm is somewhat naive. Even so, this will only happen around obstacles. Since the AutoNaut's typical operating area is a large stretch of open sea, the skip-based approach is still viable.

The recursive segmentation pruner has almost identical behavior to the skip-based pruner for small segment sizes. This is illustrated by Figure 4.8a, which produces the exact same path as in Figure 4.7a. When the maximum segment size was increased to 25 the behavior was completely different, as depicted in Figure 4.8b. Now, the algorithm trims away as many waypoints as it can before rounding the island, and there only remains a single waypoint between the start and end points. Instead of giving up and moving on, this algorithm recursively divides up the path until no more divisions are necessary, and then reconstructs the pruned path. The recursive approach therefore seems to be superior to the skip-based one.

5.6 Runtime

In Table 4.2, different runtimes for different system configurations were collected. The effects of map size, weather data collection, as well as which pruning method was employed, were considered.

Post-processing of paths by pruning seems to have little to no effect on runtime. In rows 9, 10 and 11 in Table 4.2, the map from Figure 4.5a was generated with no pruning, skip-based pruning and recursive segmentation-based pruning, respectively. The runtimes for these configurations were all almost exactly 2.5s, indicating that pruning does not put significant strain on computing resources. Since the process of pruning in essence only involves iterating through a list and remove elements from it, this is not surprising.

Map size impacted runtimes significantly. In general, the larger maps take notably longer to generate than the smaller ones. For example, the 3km^2 maps all require less than 4s to generate, while nearly all the 7km^2 maps take twice that time. The 15km^2 in row 13 took almost a minute to complete, and trials with maps of up to 70km^2 have resulted in runtimes of several minutes. Runtimes of a few minutes are generally not a problem, since the program is not required to run in real-time. For infrequent path updates at specified intervals, these runtimes are perfectly acceptable. Nonetheless, it is advantageous to reduce the runtime to both free up computing resources and to increase the quality of life of the user.

Grid resolution also impacts the generation time, exemplified by the difference in runtime between rows 3 and 5. With all other variables kept constant, the only difference between the two is the grid resolution of $100x100$ for row 3 and $50x50$ for row 5. The resulting average runtimes were 8.0s and 5.9s. This is to be expected since it takes longer to generate a higher resolution map. Runtime is further increased since there are more nodes for the A* algorithm to traverse.

Displaying the ENC, paths, vectors fields and buffer zones seems to be associated with increased runtimes. This can be seen in the differences in runtime between rows 1 and 2, where the displayed figure takes over 2s longer to run. Another example of this phenomenon is the runtimes for rows 13 and 15, which are 55.6s and 32.3s, respectively. The same weather collection and path planning took place between the two figures, the only difference being that visualization was turned off for row 15. Plots are useful for reports and human operators, but can be switched off to save quite a bit of runtime if this system is run on-board the AutoNaut.

When SeaCharts generates an ENC, new data has to be loaded in. The geodatabase files then have to be converted into shapefiles, and these are stitched together into the ENC object itself. This requires a noticeable amount of time, and is more time-consuming the larger the map. The runtime for row 3, for instance, is 8.0s, and 11.4s for row 4. The same thing applies to row 13 and 14, where the runtimes are 55.6s and 65.7s, respectively. This discrepancy of over 10s demonstrates the toll loading new data takes on runtime.

Weather collection also constitutes a large contribution to total runtime. In row 13, the path seen in Figure 4.6 took 55.6s to generate, with weather collection activated. Comparing this to row 16, where dummy data was used and no data downloading took place, resulted in a marked decrease in runtime. The figure with synthetic data only took 23.2s to run, which is under half the time.

A lot of the runtime related to weather collection can likely be attributed to limitations set deliberately by the service provider. For example, download speeds from MET would sometimes drop significantly during testing. This reduced data speed was likely due to MET throttling data speeds for users who take up too much bandwidth. This practice is also mentioned in the MET API documentation[53].

Weather in this project is collected in large grids, and these weather grids would sometimes have hundreds of grid cells that needed to be filled. Since this project does not put any synthetic delays between queries, the requests for new data might come in so fast that MET considered the query string a DoS attack. Sometimes queries would have to be resent, and sometimes queries resulted in timeouts. These timeouts were likely another security precaution implemented by MET. It should be mentioned that these drops in download speed happened very rarely, and only during periods of multiple tests in a minute. Timeouts only happened twice during the entire Spring semester of 2022. During real missions, throttling and timeouts are not expected to pose a serious problem. Under frequent testing, however, some wait period should be implemented between each query.

5.7 Viability

Whenever considering a proposed solution to any engineering problem, it is useful to ask the question: is this a viable solution?

So is the A* algorithm, modified to take weather data into account, viable for path planning in a marine environment? Going back to Wang et. al., the answer is no[12]. This is mainly due to the A* algorithm's inability to account for dynamic obstacles due to e.g. periodically changing ocean depth, weather factors and other vessels. The exorbitant computation time of the A* algorithm is also pointed out. Wang et. al. go even further, and state that A* is "unsuitable for CA and RC".

Work done in this thesis has shown that the modified A* algorithm is capable of path planning in a weather-dependent setting. By modifying the cost of movement based on weather data, the system successfully generates paths that take weather into account. All that is needed to account for the dynamic behavior of shifting weather would be to perform RC by rerunning at certain intervals. Since CA is already handled on a lower level of system architecture, there is no immediate need to handle it on the path planning level.

The AutoNaut is quite slow compared to other USVs, and the computation times recorded in Section 4.5 are therefore not seen as prohibitive to normal operation. If the vessel had been considerably faster, recomputing the A* algorithm at appropriate intervals might not have been possible. A* search could therefore be ill-advised for the JingHai-I and Protector USVs mentioned in Section 1.2, even though it works well for the AutoNaut. For projects involving other USVs, runtime requirements should be determined before selecting a path planner.

Chapter 6

Future Work

Several problems remain to be solved in order to create an autonomous and robust system capable of long-duration independent ocean exploration. A few of them are discussed in this chapter.

6.1 Communication with AutoNaut

As is, the programs generated in this work have no mechanism with which to communicate with the AutoNaut. Waypoints need to be communicated to the AutoNaut, either through Neptus, DUNE or other means directly. At the same time, sensor data needs to be transmitted to the mission planner and path planner. This would for instance allow the system to access the ownship angle of the AutoNaut directly through the compass, which would be much more precise than estimating the angle using the assumed average over a distance. Gathering this data requires the addition of a communication module to the project. Much of this can be skipped by porting this project directly onto the vessel itself, running entirely on-line. This will be discussed in Section 6.2.

6.2 Improved autonomy

For now, the program runs off-line, and waypoints generated from the path planner are passed to the AutoNaut manually using Neptus. The long-term goal should be almost total autonomy in order to support missions of longer duration at sea. Everything from mission- and path planning to emergency maneuvers should then be undertaken on-board the platform itself. The roles of sender and receiver should then be reversed; the AutoNaut should transmit to the operator where it is going, and the operator should receive and plot this data in the middleware.

Large areas of ocean could be explored and monitored efficiently through the use of multiple USVs. For example, a swarm of AutoNauts could be dispatched to a given area. They would then divide the area between them, and patrol it using paths generated by the system developed in this work. This could be beneficial for larger operational areas at sea, for instance mapping the saline concentration along one of the great ocean currents. Other examples include monitoring fish activity within breeding grounds, keeping an eye on algae blooms or even monitoring hydrocarbon content in the sea after an oil spill.

6.3 Mission planner

The ultimate objective of the project should be total or near total autonomy, and to avoid human intervention whenever possible. This is both for the safety of land-based personnel and to increase the possible duration of missions. An assumption made in this section is that the program can be run on the AutoNaut itself in the future. This is beneficial for several reasons, mostly because it allows the AutoNaut to be more independent of land-based operations.

This section documents a proposal for a rudimentary mission planner in the form of a Finite State Machine (FSM), which aims to make the vessel completely autonomous. Figure 6.1 illustrates an abstract concept version of the proposal. Four states are proposed, including an emergency mode in the case of extreme weather or system malfunction. These states will be discussed in more detail.

6.3.1 Idle mode

A setup phase is necessary before the FSM can be initiated. This is depicted as the entry point in Figure 6.1. Setup would include loading of new geodatabase data. After the system has been set up correctly, it would enter the IDLE state. In this state, the vessel would wait for a new mission to be issued.

6.3.2 Calculating path

The work done in this thesis is meant to serve the functions of the CALCULATING PATH state. Issuing a new mission will trigger a state transition into the CALCULATING PATH state. Weather collection will be performed in this state. AIS messages will also be collected here. Then, ENC generation and subsequent path calculation should take place. Recalculation of paths also takes place here periodically.

When the AutoNaut has reached its goal, the mission is confirmed as completed. The system will then revert back to the IDLE state and await new orders.

Some risk assessment should also be conducted in this state. If the weather is determined to pose significant risk to the craft, it should enter the EMERGENCY state. Examples of possible triggers for this is winds or currents exceeding certain speeds, or waves exceeding a specified height.

6.3.3 In transit

The IN TRANSIT state will be where the AutoNaut will spend most of its time at sea. When the previous state signals that path calculation is finished, a transition

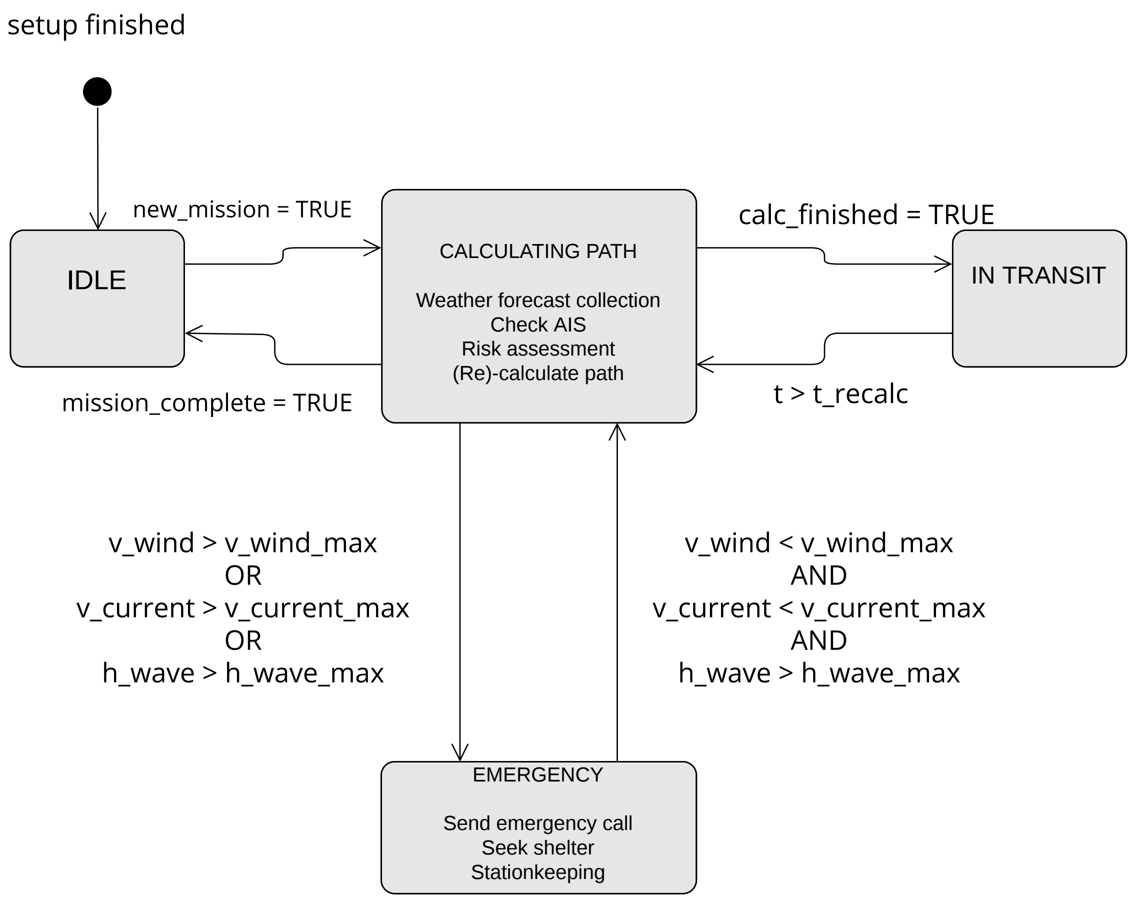


Figure 6.1: FSM of mission planner. Generated using an online tool by Visual Paradigm[68].

into the IN TRANSIT state will occur. This state is only responsible for keeping time while the vessel follows the generated path. The path should be recalculated at certain intervals to account for weather changes, nearby ships and tidal events. When the timer exceeds a specified threshold time t_{recalc}, the system should transition back to the CALCULATING PATH state. A recommended starting value for t_{recalc} is 30 minutes. This value can then be increased or decreased as necessary.

6.3.4 Emergency

When the system enters the EMERGENCY state, it should first broadcast an emergency message to operators on land. This should serve to draw the attention of the operators, even when they are not actively monitoring the AutoNaut remotely. The proposed solution for this is to send text messages to operators over cellular network. The message should contain some information about the cause of the emergency, for instance wave height. If the vessel experiences a sudden spike in acceleration, this might be due to a collision with land or another ship. In that case, such a message should also be sent. Emergency messages should also contain the last known position of the vessel, so rescuers know where to look in case of system failure.

After the initial emergency message has been sent out, the vessel should attempt to move away from the hazardous area. It should search for areas with less severe weather, and attempt to move there as quickly as possible. If this is not possible, it should seek to position itself as far away from land as possible. Seeking shelter in the open ocean limits the probability of grounding, and allows more time for operators to react. Stationkeeping should then be performed until the weather has calmed and the mission can resume.

6.4 Robustness, redundancy and availability

The system needs to be made fault tolerant in the event of power outage, software crash or similar fault modes. There should be functionality for restoring the system to its previous stable state. Storing the state of the vessel at certain intervals is one way of doing this.

As mentioned in Chapter 3, throttled download speeds and temporary unavailability of MET data has already been experienced during testing. This needs to be handled by the system when running on its own. The system should attempt to download data, and then check that the download was successful. If not, it should wait a given period before attempting again. If access is still denied by the service provider, previous data could be recovered and assumed to still be valid. This data could be stored along with the last stable state of the vessel. If stored weather data is not available, sensor data directly from the AutoNaut could be assumed to be valid for the entire operating area.

6.5 Reducing download speeds

A lot of data is downloaded at each query to the MET database. Two JSON files are downloaded for each grid cell in the weather data grid, one for Locationforecast and one for Oceanforecast. Efforts have been made to reduce the amount of data downloaded by increasing the weather grid's cell sizes to 2.5km in each direction. This has already greatly reduced runtimes, but it is still possible to reduce them further by being selective about when and where to download data.

Usually the craft will operate in the open ocean, with very few obstacles present for hundreds of kilometers. The optimal path will usually be an almost straight line from the starting point to the goal, if the weather is relatively calm. Even if there exists an occasional reef or shallow, the path planner will usually just move around it and keep going following the original line. A lot of runtime could be saved by only collecting weather data along this line, with a small standard deviation interval around the line to capture deviations due to weather effects. If this turns out to be insufficient for a particular location, the algorithm could restart and collect data for the entire weather grid.

Two JSON files are downloaded for each cell in the weather grid, one for Loactionforecast and one for Oceanforecast. An entire JSON file might not have to be downloaded. Contained in the downloaded JSON files are unnecessary data like sea water temperature, relative humidity and air pressure at sea level. If it is possible to query only the relevant data, runtimes can likely be reduced. More specific weather requests might help bring the runtime down significantly.

6.6 New cost function and further cost tuning

Some cost tuning was performed for current, wind and wave parameters. However, the tuning proved inconclusive, likely due to a sub-optimal cost function. An improved cost function should be created. Such a function should be based on the hydrodynamics of the AutoNaut, and should respond to both the magnitudes and directions of the weather forces.

The virtual scenarios used for testing here are likely insufficient for real-world scenarios. After an improved cost function is created, more tuning will be necessary for real long-term missions spanning larger distances. Weather costs in this paper have also only been adjusted according to the local conditions in Frohavet and nearby areas. This area consists of largely open ocean, but is still nestled and somewhat protected by the island Frøya, the islands surrounding Mausund, as well as the Froan archipelago to the north. Areas without land formations would probably need another tuning scheme since there are less obstacles to account for. Extensive testing and tuning would have to be conducted for different operating scenarios for the system to be more reliable.

6.7 Surfing on the weather

The AutoNaut is not very fast. In fact, it is considerably slower than the wind speed even on calm days. Instead of struggling against the elements, the power of the wind, waves and currents can be exploited in a more efficient manner. With a proper model of sea and wind states, as well as a hydrodynamic model of the AutoNaut itself, a more efficient route can probably be found by aligning the AutoNaut with these weather forces.

An example could be to align the AutoNaut with the ocean current, allowing it to "freeload" off of the motions of the ocean. Another example is to use the tides when the AutoNaut operates close to shore. This will likely require extensive knowledge in sea-state modelling, as well as knowledge in nonlinear optimization.

6.8 AIS

AIS messages are available using the transceiver on-board the AutoNaut. These messages contain the position, bearing and velocity of nearby marine vessels. It is possible to combine this information with the existing path planning algorithm to avoid coming within close proximity of other craft. There already is a COLREGs-compliant collision avoidance system running on level 1. This system was created in Bergh's master's thesis[18]. This system, however, will only kick in in close proximity to other ships, and it may then already be too late to avert a collision. It could therefore be worth the investment to penalize proximity to other ships within the cost function itself. On the other hand, the AutoNaut is relatively slow, and is not guaranteed to be capable of getting out of harm's way in time either way. To a certain extend, one must trust that other actors at sea know what they are doing.

6.9 Sun

The AutoNaut runs all its on-board electronics from the battery bank housed within the hull. This battery is only charged using the solar array discussed in Chapter 2. However, the Norwegian climate does not always lend itself to solar power.

Overcast data for several days or weeks in advance could be harvested. A model of remaining battery capacity should be created to estimate how much time the vessel has left before losing power. When the batteries are nearly empty, the vessel should go into a "bare-bones" mode, where extraneous computing and sensors are disabled temporarily to save power. Then, the vessel should wait for clear skies to return to its mission, or wait to be rescued. Cloud cover as a percentage is available in the Locationforecast API, and can be used to determine when to resume a mission.

6.10 Additional parameters

Apart from sun, many other parameters might be relevant for operations. Some of these are region-specific, for example the high Arctic or Antarctic seas, where icebergs are present. Iceberg data is available from MET's APIs, and these should be treated as obstacles just like land and shallows. Other examples include areas which high volcanic activity, for instance the Hawaiian archipelago and Norway's own Jan Mayen. Volcanic ash forecasts are also available with MET's tools.

Another parameter that could be used for path planning is ocean depth. Ocean depth is already available within SeaCharts itself, and the ocean is segmented into many layers of varying depths. Depth has a large influence on the velocity of oncoming waves, with shallow water corresponding to increased wave speeds[69]. Grid nodes intersecting with shallow water segments could then be associated with a slightly higher cost to penalize movement in these higher risk areas.

6.11 SeaCharts

The SeaCharts project is currently without a weather engine. The plotting tools created for this thesis were made specifically for the ENCs in SeaCharts, and could be incorporated into the project with few modifications. The weather tools created here allow for flexible plotting with customizable weather resolution, vector density, colors and object sizes.

As mentioned in Section 6.10, more data points than wind, waves and current can be accessed through MET's APIs. Only minor modifications to the software is necessary to plot new weather phenomena. Examples of these include temperatures, barometric pressure, cloud coverage and volcanic ash. These could be visualized in a similar manner to the the wind, waves and current already developed, by using a grid projected onto a SeaChart ENC.

Chapter 7

Conclusion

A graph-search path planning algorithm capable of accounting for weather effects was created in this master's thesis. The algorithm is tailored for a small, wave-actuated USV, and is based on the A* search algorithm coupled with weather forecasts sourced from two MET APIs. The resulting modified A* algorithm proved itself capable of generating paths from one point to another on a large map, based on both local geography and weather data.

While it was demonstrated that the path planner was responsive to the effects of the weather forces' attack angles, it remained largely unresponsive to the magnitude of the forces. Tuning of the system remains inconclusive. A more comprehensive cost function, as well as an improved tuning scheme, are likely needed to generate paths that are optimized for both safety and time. When this is in place, the path planner can become a part of a complete mission planner for the AutoNaut.

Different configurations were tested on various maps to investigate how much each part of the system contributes to the total runtime. Weather collection was found to be the single largest contributing factor to runtime, while map size, grid resolution and plotting were also found to impact runtime significantly. Post-processing in the form of pruning was found to have little impact on total runtime.

Both of the pruning functions created for this project were found to be effective at reducing the number of waypoints of the paths. The recursive segmentation-based pruning algorithm was deemed superior due to its performance when navigating around obstacles.

Detailed ENCs were generated using the SeaCharts API. Tools for visualizing the collected weather data were created, and are capable of interacting with SeaCharts. These plotting tools can be incorporated into SeaCharts with minor modifications in the future.

Bibliography

[1] B. Å. Holm, 'Graph-based path-finding for wave powered unmanned surface vehicle,' *Norwegian University of Science and Technology*, Jan. 2022.

[2] L. Kobylinski, 'Capsizing scenarios and hazard identification,' *8th International Conference on the Stability of Ships and Ocean Vehicles*, 2003, Accessed: 2022-07-12. [Online]. Available: https://shipstab.org/files/Proceedings/STAB/STAB2003/Papers/Paper%2061.pdf.

[3] A. L. Kornhauser, 'Global navigation satellite system (gnss),' *Department of Operations Research Financial Engineering at Princeton University*, Accessed: 2022-07-12. [Online]. Available: https://www.princeton.edu/~alaink/Orf467F07/GNSS.pdf.

[4] N. M. Euthority, 'Electronic navigational charts (enc),' 2022, Accessed: 2022-07-12. [Online]. Available: https://www.kartverket.no/en/at-sea/nautical-charts/elektroniske-sjokart-enc.

[5] Ø. J. Rødseth and H.-C. Burmeister, 'Developments toward the unmanned ship,' 2012, Accessed: 2022-07-12. [Online]. Available: https://www.sintef.no/en/publications/publication/?pubid=947033.

[6] N. C. Thompson, S. Ge and G. F. Manso, 'The importance of (exponentially more) computing power,' *MIT Initiative on the Digital Economy*, 2022, Accessed: 2022-07-08. [Online]. Available: https://arxiv.org/pdf/2206.14007.pdf.

[7] Y. Peng, Y. Yang, J. Cui, X. Li, H. Pu, J. Gu, S. Xie and J. Luo, 'Development of the usv 'jinghai-i' and sea trials in the southern yellow sea,' *Ocean Engineering*, vol. 131, pp. 186–196, 2017, Accessed: 2022-07-12, ISSN: 0029-8018. DOI: https://doi.org/10.1016/j.oceaneng.2016.09.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801816303705.

[8] R. A. D. S. Ltd., 'Protector™ usv,' *Protector USV brochure*, 2022, Accessed: 2022-07-12. [Online]. Available: https://www.rafael.co.il/worlds/naval/usvs/.

[9] S. Blindheim and T. A. Johansen, 'Electronic Navigational Charts for Visualization, Simulation, and Autonomous Ship Control,' 2022.

[10] P. E. Hart, N. J. Nilsson and B. Raphael, 'A formal basis for the heuristic determination of minimum cost paths,' *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. DOI: 10.1109/TSSC.1968.300136.

[11] V. Kenny, M. Nathal and S. Saldana, 'Heuristic algorithms,' *Northwestern University Open Text Book on Process Optimization*, 2014, Accessed: 2022-01-31. [Online]. Available: https://optimization.mccormick.northwestern.edu/index.php/Heuristic_algorithms.

[12] N. Wang, X. Jin and M. J. Er, 'A multilayer path planner for a usv under complex marine environments,' *Ocean Engineering*, vol. 184, pp. 1–10, 2019, Accessed: 2022-07-12, ISSN: 0029-8018. DOI: https://doi.org/10.1016/j.oceaneng.2019.05.017.

[13] Y. Chen, G. Bai, Y. Zhan, X. Hu and J. Liu, 'Path planning and obstacle avoiding of the usv based on improved aco-apf hybrid algorithm with adaptive early-warning,' *IEEE Access*, vol. 9, pp. 40 728–40 742, 2021. DOI: 10.1109/ACCESS.2021.3062375.

[14] H. Mousazadeh, H. Jafarbiglu, H. Abdolmaleki, E. Omrani, F. Monhaseri, M.-r. Abdollahzadeh, A. Mohammadi-Aghdam, A. Kiapei, Y. Salmani-Zakaria and A. Makhsoos, 'Developing a navigation, guidance and obstacle avoidance algorithm for an unmanned surface vehicle (usv) by algorithms fusion,' *Ocean Engineering*, vol. 159, pp. 56–65, 2018, Accessed: 2022-07-12, ISSN: 0029-8018. DOI: https://doi.org/10.1016/j.oceaneng.2018.04.018.

[15] B. O. Agdal, 'Design and implementation of control system for green unmanned surface vehicle,' *Norwegian University of Science and Technology*, Jun. 2018, Accessed: 2022-07-12. [Online]. Available: https://autonaut.itk.ntnu.no/lib/exe/fetch.php?media=autonaut_level1.pdf.

[16] S. D. Sæter, 'COLREGS compliant Collision Avoidance System for a Wave and Solar Powered USV,' 2018, Accessed: 2022-07-12. [Online]. Available: https://autonaut.itk.ntnu.no/lib/exe/fetch.php?media=autonaut_level2.pdf.

[17] A. Heggernes, 'Mission planning and control system for wave powered autonomous surface vessel (autonaut),' *Norwegian University of Science and Technology*, Jun. 2021.

[18] T. K. Bergh, 'Enc-based collision and grounding avoidance system for a green-energy autonomous surface vehicle,' *Norwegian University of Science and Technology*, Jun. 2021.

[19] A. Dallolio, 'The AutoNaut - Wave-Propelled Unmanned Surface Vehicle,' 2022, Accessed: 2022-07-12. [Online]. Available: https://autonaut.itk.ntnu.no/doku.php.

[20] A. Dallolio, L. Bertino, L. Chrpa, T. A. Johansen, M. Ludvigsen, K. A. Orvik, L. H. Smedsrud, J. Sousa, I. B. Utne, P. Johnston and K. Rajan, 'Long-duration autonomy for open ocean exploration: Preliminary results & challenges,' *Norwegian University of Science and Technology*, 2019, Accessed: 2022-07-12. [Online]. Available: https://autonaut.itk.ntnu.no/lib/exe/fetch.php?media=autonaut_rss.pdf.

[21] A. Dallolio, 'Research missions data collection,' *AutoNaut wiki*, 2020, Accessed: 2022-07-12. [Online]. Available: https://autonaut.itk.ntnu.no/doku.php?id=mission.

[22] H. Øveraas, A. Dallolio, P. R. D. L. Torre and T. A. Johansen, 'Monitoring the growth of coastal algae blooms in harsh weather conditions using a wave-propelled asv: Challenges and lessons learned,' submitted, Jun. 2022.

[23] 'Orcas,' *Norwegian University of Science and Technology*, Accessed: 2022-07-12. [Online]. Available: https://www.ntnu.edu/imt/orcas.

[24] B. Å. Holm, 'AStarWeather,' 2022, Accessed: 2022-07-11. [Online]. Available: https://github.com/bendikah/AStarWeather.

[25] N. U. of Science and Technology, 'Aur-lab homepage,' 2022, Accessed: 2022-07-12. [Online]. Available: https://www.ntnu.edu/aur-lab.

[26] A. Ltd, 'Autonaut homepage,' Accessed: 2022-07-12. [Online]. Available: https://www.autonautusv.com/home.

[27] Blue Robotics Inc., 'T200 Thuster Product Description,' Accessed: 2022-01-29. [Online]. Available: https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/.

[28] Campbell Scientific, 'CR6 Measurement and Control Datalogger,' Accessed: 2022-01-31. [Online]. Available: https://www.campbellsci.com/cr6.

[29] Hemisphere GNSS, Inc., 'Vector V104 GPS Smart Antenna,' Accessed: 2022-01-31. [Online]. Available: https://www.subseatechnologies.com/media/files/files/7db3d1ad/V104_WEB_09_2016_STI.pdf.

[30] Victron Energy, 'VSmartSolar MPPT 75/10, 75/15, 100/15 100/20 SmartSolar MPPT 75/10, 75/15, 100/15 100/20,' Accessed: 2022-01-31. [Online]. Available: https://www.victronenergy.com/solar-charge-controllers/smartsolar-mppt-75-10-75-15-100-15-100-20.

[31] beagleboard.org, 'BeagleBone Black,' Accessed: 2022-01-31. [Online]. Available: https://beagleboard.org/black.

[32] FLIR Systems, 'Raymarine AIS650 Transceiver and Raymarine AIS350 Receiver,' Accessed: 2022-01-31. [Online]. Available: https://www.raymarine.com/ais/ais650-350/index.html.

[33] Airmar Technology Corp., '120WX WeatherStation Instrument,' Accessed: 2022-01-31. [Online]. Available: https://www.airmar.com/weather-description.html?id=152.

[34] Sensorwell, 'HMR3000 Digital Compass Module,' Accessed: 2022-01-31. [Online]. Available: https://www.sensorwell.at/en/products/sensors/magnetic-sensors-and-transducers/integrated-compassing-solutions/hmr3000-digital-compass-module/.

[35] SentiSystems, 'SentiBoard - Integrate your sensors,' Accessed: 2022-01-31. [Online]. Available: https://sentisystems.com/sales-2/.

[36] Technologic Systems, Inc., 'TS-7970 - Powerful SBC with Wireless, Dual GbEth, and Audio/Video I/O,' Accessed: 2022-01-31. [Online]. Available: https://www.embeddedts.com/products/TS-7970.

[37] Nortek, 'Signature500 - Mean currents and turbulence, plus wave height, direction and ice tracking,' Accessed: 2022-01-31. [Online]. Available: https://www.nortekgroup.com/products/signature-500.

[38] Sea-Bird Scientific, 'SBE 49 FastCAT CTD Sensor,' Accessed: 2022-01-31. [Online]. Available: https://www.seabird.com/sbe-49-fastcat-ctd-sensor/product?id=60762467704.

[39] Aanderaa, 'Oxygen Optode 4835,' Accessed: 2022-01-31. [Online]. Available: https://www.aanderaa.com/media/pdfs/d385_aanderaa_oxygen_sensor_4835.pdf.

[40] Thelma Biotel, 'TB Live (specifications),' Accessed: 2022-01-29. [Online]. Available: https://www.thelmabiotel.com/receivers/tbr-700-rt/.

[41] Sea-Bird Scientific, 'ECO Triplet-w,' Accessed: 2022-01-31. [Online]. Available: https://www.seabird.com/eco-triplet-w/product?id=60762467721.

[42] M. Kris, C. Devieux and P. Swan, 'Overview of iridium satellite network,' *Proceedings of WESCON'95. IEEE*, 1995, Accessed: 2022-07-12. [Online]. Available: https://www.researchgate.net/profile/Peter-Swan-2/publication/3622510_Overview_of_IRIDIUM_satellite_network/links/54fe3f9b0cf2eaf210b24d72/Overview-of-IRIDIUM-satellite-network.pdf.

[43] Ground Control Systems, Wireless Innovation Ltd and Rock Seven Mobile Services Ltd, 'RockBLOCK+,' Accessed: 2022-01-31. [Online]. Available: https://www.groundcontrol.com/?refer=rock7.

[44] Skagmo Electronics, 'Owl VHF High performance radio transceiver Datasheet rev. B,' Accessed: 2022-07-12. [Online]. Available: https://autonaut.itk.ntnu.no/lib/exe/fetch.php?media=owldatasheet.pdf.

[45] MikroTik, 'LtAP mini LTE kit,' Accessed: 2022-01-31. [Online]. Available: https://mikrotik.com/product/ltap_mini_lte_kit#fndtn-specifications.

[46] Underwater System and Technology Laboratory, University of Porto, 'LSTS Reference Documentation,' 2021, Accessed: 2022-01-19. [Online]. Available: https://www.lsts.pt/docs/.

[47] S. Blindheim, 'Seacharts github repository,' 2021, Accessed: 2022-01-19. [Online]. Available: https://github.com/simbli/seacharts.

[48] S. Giles, 'The Shapely User Manual,' 2021, Accessed: 2022-01-19. [Online]. Available: https://shapely.readthedocs.io/en/stable/manual.html.

[49] A. E. Prieditis, 'Machine discovery of effective admissible heuristics,' *Kluwer Academic Publishers*, p. 1, 1993, Accessed: 2022-01-30. DOI: 10.1007/BF00993063. [Online]. Available: https://link.springer.com/article/10.1007/BF00993063.

[50] W. Yuan, N. Ganganath, C.-T. Cheng, G. Qing and F. C. Lau, 'A consistent heuristic for efficient path planning on mobility maps,' pp. 1–5, 2017, Accessed: 2022-01-30. DOI: 10.1109/WoWMoM.2017.7974356. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7974356.

[51] 'Openweather api pricing,' *OpenWeather webpage*, 2022, Accessed: 2022-07-12. [Online]. Available: https://openweathermap.org/price.

[52] 'Accuweather api packages,' *AccuWeather webpage*, 2022, Accessed: 2022-07-12. [Online]. Available: https://developer.accuweather.com/packages.

[53] *MET Weather API*, Accessed: 2022-07-12, Norwegian Meteorological Institute, 2022. [Online]. Available: https://api.met.no/.

[54] I. A. Seierstad, 'Locationforecast data model,' *MET Norway*, 2020, Accessed: 2022-05-06. [Online]. Available: https://docs.api.met.no/doc/locationforecast/datamodel.

[55] I. A. Seierstad, 'Oceanforecast data model,' *MET Norway*, 2021, Accessed: 2022-05-06. [Online]. Available: https://docs.api.met.no/doc/oceanforecast/datamodel.

[56] N. Prewitt and S. M. Larson, 'Requests: Http for humans™,' Accessed: 2022-07-12. [Online]. Available: https://requests.readthedocs.io/en/latest/.

[57] *Geogebra online calculator*, Accessed: 2022-05-27, GeoGebra GmbH, 2022. [Online]. Available: https://www.geogebra.org.

[58] W. Heiskanen, 'On the world geodetic system,' *The International Hydrographic Review*, 1953, Accessed: 2022-07-12. [Online]. Available: https://journals.lib.unb.ca/index.php/ihr/article/view/26831/1882519590.

[59] J. A. O'Keefe, 'The universal transverse mercator grid and projection,' *The Professional Geographer 4.5*, pp. 19–24, 1952, Accessed: 2022-07-12. [Online]. Available: https://www.tandfonline.com/doi/pdf/10.1111/j.0033-0124.1952.45_19.x?casa_token=U7edfFQa3PEAAAAA:-ys91Jt066pk_DMor05KwmcrOZfLP_r6gN5AkyZRt1pQKNDeWaRf9ZTC5zgXEzi3XEbLntYvOJ1M.

[60] Kbellis, 'Modified utm zones,' *Wikimedia Commons*, 2017, Accessed: 2022-07-12. [Online]. Available: https://commons.wikimedia.org/wiki/File:Modified_UTM_Zones.png#filelinks.

[61] PROJ contributors, *PROJ coordinate transformation software library*, Accessed: 2022-07-12, Open Source Geospatial Foundation, 2022. DOI: 10.5281/zenodo.5884394. [Online]. Available: https://proj.org/.

[62] R. A. Ibrahim and I. M. Grace, 'Modeling of ship roll dynamics and its coupling with heave and pitch,' *Mathematical Problems in Engineering Volume 2010*, pp. 1–5, 2009, Accessed: 2022-07-09. [Online]. Available: https://downloads.hindawi.com/journals/mpe/2010/934714.pdf.

[63] W. Blendermann, 'Parameter identification of wind loads on ships,' *Journal of Wind Engineering and Industrial Aerodynamics*, 1994, Accessed: 2022-07-12. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167610594900671.

[64] N. Oceanic and A. Administration, 'Why does the ocean have waves?' *U.S. Department of Commerce*, 2021, Accessed: 2022-07-09. [Online]. Available: https://oceanservice.noaa.gov/facts/wavesinocean.html.

[65] 'Online diagram tool,' 2022, Accessed: 2022-07-12. [Online]. Available: https://www.diagrams.net/.

[66] E. L. Delmar-Morgan, 'The beaufort scale,' *Journal of Navigation*, vol. 12, no. 1, pp. 100–102, 1959, Accessed: 2022-07-12. DOI: 10.1017/S0373463300045902.

[67] F. Alfahmi, O. Hakim, R. Dewi and A. Khaerima, 'Utilization of data mining classification techniques to identify the effect of madden-julian oscillation on increasing sea wave height over east java waters,' *IOP Conference Series: Earth and Environmental Science*, vol. 399, p. 012 062, Dec. 2019, Accessed: 2022-07-12. DOI: 10.1088/1755-1315/399/1/012062.

[68] 'Online state machine diagram tool,' 2022, Accessed: 2022-07-12. [Online]. Available: https://online.visual-paradigm.com.

[69] C. C. Mei, M. A. Stiassnie and D. K.-P. Yue, 'Theory and Applications of Ocean Surface Waves,' *Advanced Series on Ocean Engineering: Volume 42*, p. 18, 2018, Accessed: 2022-05-28. [Online]. Available: https://www.worldscientific.com/worldscibooks/10.1142/10212.

Appendix A

The MIT License

Appendix B

Example scripts

A collection of example scripts has been created and made available for open use. The purpose of this is to educate the reader about the use and development of this toolbox, and how the tools developed for this project interact with the Shapely and SeaCharts frameworks.

A few highlighted examples from the example scripts:

- How to generate a SeaCharts ENC object and plot it.
- How to generate and draw weather, wind and current grids.
- How to generate and draw paths.

The example scripts are located at the following GitHub repository:
https://github.com/bendikah/AStarWeather

Note that runtimes of the example scripts will likely be much longer the first time the example is ran. This is because the geodatabase data needs to be collected and parsed into shapefiles by SeaCharts. When the program has been executed once, the argument *new_data* can be set to *False* in the function call *seacharts.ENC()*, which is responsible for generating the ENC object. This will reduce runtime considerably.