



Master's thesis

2020

Master's thesis

Nikolai Lauvås

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

Nikolai Lauvås

# Design and development of a robotic fish tracking vehicle

July 2020





Norwegian University of  
Science and Technology

# Design and development of a robotic fish tracking vehicle

**Nikolai Lauvås**

Engineering Cybernetics

Submission date: July 2020

Supervisor: Jo Arve Alfredsen, NTNU/DEC

Co-supervisor: Alberto Dallolio, NTNU/DEC

Norwegian University of Science and Technology  
Department of Engineering Cybernetics





# Abstract

Positioning of acoustic fish tags over an extent of time is desired for purposes such as monitoring the migration of fish in fjords. The use of autonomous vehicles equipped with acoustic receivers has in recent studies been shown to be promising candidates for obtaining such data. This thesis describes the development of a multi-vessel system designed to track individual fish over extended time periods. The concept is based on using the time-difference-of-arrivals (TDOA) to provide an estimate of an acoustic fish tags position. Because the tag is attached to the fish, this can be seen as the fish position. The vessels can then follow the estimated position to track the fish over time.

To be able to operate autonomously, knowledge about vehicle surroundings is paramount. To increase the situational awareness of the vessel, a system using electronic navigational charts in combination with available depth soundings is developed. The system is then implemented in the middleware DUNE that is running on-board the vessel. Based on this system, two functions is implemented on the vessel: an anti-grounding monitor and a path planning algorithm.

For communication between vessels, and from vessel to operator, a cellular modem and a 5GHz wireless AirMax radio has been integrated to the vessel. The design is discussed and completed in this thesis, along with the addition of a virtual private network (VPN). This aims to simplify communication with the vessels over Internet.

The data collected by the Otters are made accessible by the development of an online visualization system consisting of three parts: A time-series database, a database ingester, and the Grafana online monitoring solution. Combined, they make near real-time monitoring of vessel telemetry available in a password protected online interface.

Lastly, the hydrophone software is improved upon, along with the upgrade to the new Thelma Biotel TB Live hydrophone.

---

---

# Sammendrag

Posisjonering av akustiske fiskemerker (tags) over tid er ønsket til bruksområder som å spore fiskers vandring gjennom fjorder. Bruken av autonome fartøy utstyrt med akustiske mottakere har i de senere årene dukket opp i forskning som et lovende alternativ for å samle inn slik data. Denne avhandlingen beskriver utviklingen av et system bestående av flere små overflatefartøy som er designet for å kunne spore enkeltfisk over tid. Konseptet er basert på å sammenligne tidspunktene hvert fartøy detekterer et signal sendt fra fiskemerket, og ved det gi et estimat på dets posisjon. Ved å plassere fiskemerket på fisken blir dette dermed et estimat av fiskens posisjon. Videre skal fartøygruppen i formasjon følge estimatet for å spore fiskens vandring over tid.

For å kunne operere autonomt må fartøyet ha kunnskap om miljøet det opererer i. Dette blir i denne avhandlingen løst ved implementasjonen av et system basert på elektroniske sjøkart og offentlig tilgjengelige dybdemålinger. Videre blir dette systemet implementert i middelvaren DUNE som kjører på fartøyet. Den tilgjengelige dataen brukes så i designet av en global stifinner og et system for å unngå grunnstøting.

Kommunikasjon mellom fartøyene i gruppen og til systemets operatør skjer gjennom et mobilt bredbåndmodem og en 5GHz AirMax radioløsning. Dette systemet blir ferdigstilt og beskrevet i denne avhandlingen, sammen med oppsettet av et virtuelt privat nettverk (VPN). Dette blir innført for å forenkle kommunikasjonen med fartøyene over Internett.

For å gjøre den innsamlede dataen lettere tilgjengelig for andre forskere, blir det så designet et nettbasert visualiseringssystem. Dette består av tre hoveddeler:

En tidsseriedatabase, en applikasjon som fyller telemetri i databasen og visualiseringsløsningen Grafana. Ved å kombinere dem kan systemet visualisere all telemetri fra fartøyet i nesten sanntid gjennom et passordbeskyttet og brukervennlig grensesnitt.

Det blir også gjort forbedringer i programvareintegrasjonen til hydrofonen, sammen med beskrivelsen av oppgraderingen til den nye akustiske mottakeren fra Thelma Biotel, TB Live.

---

# Preface

This thesis represents the continuation of work performed during attendance in the course *TTK4550 - Engineering Cybernetics, Specialization Project*, which work is documented [1], and summarized in appendix A.2. The authors project during the course was to perform a system integration on a catamaran, bringing it from a state with partially-finished hardware (and no software) to a functioning autonomous vessel.

The author is a student of engineering cybernetics at NTNU, specializing in embedded systems design. He has an interest in programming, database systems, GNU/Linux systems and the development of networked home monitoring systems. These hobbies have greatly benefited the thesis work, and has also affected its content.

The author would like to acknowledge the help of his supervisors: Associate professor Jo Arve Alfredsen and PhD student Alberto Dallolio. Their expertise has benefited both the quality of the work, and the authors zeal in exploring its main subjects, autonomous marine vessels and acoustic telemetry. Alberto Dallolio was also a contributor in the development of situational awareness tasks in DUNE.

The author would also like to thank Håkon Jarand Dugstad Johnsen for his detailed and helpful comments on the thesis report.

During the spring of 2020, at the time when the thesis work was performed, the global pandemic of Covid-19 hit Norway. The national measures taken to reduce and halt its spread, including a lockdown that restricted work at NTNU, led to some of the thesis goals being changed in the middle of its duration. Multiple of the original goals not met were related to the use of hardware and performing sea trials, both which became unavailable for most of the thesis work. Supervision was also made more complicated, as it could no longer be done in person.

As an end to this preface, the author would like to thank his loving wife for her support, and his daughter for ensuring that brakes were taken.

Nikolai Lauvås

---

# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>Sammendrag</b>	<b>iii</b>
<b>Preface</b>	<b>v</b>
<b>Table of Contents</b>	<b>x</b>
<b>List of Tables</b>	<b>xi</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Code</b>	<b>xvii</b>
<b>Master’s Thesis Assignment</b>	<b>xviii</b>
<b>Abbreviations</b>	<b>xix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Goal and State of the Fish Otter Project . . . . .	2
1.2.1 Outline of a Fish Tracking Mission . . . . .	2
1.2.2 Project State Before Thesis . . . . .	2
1.2.3 Thesis Contributions . . . . .	2
1.3 State of the Art . . . . .	3
1.3.1 Global Path Planning for Marine Vessels . . . . .	4
1.3.2 A Priori Situational Awareness for Unmanned Marine Vessels . . . . .	4
1.3.3 Mobile Acoustic Fish Tracking . . . . .	5
1.4 Thesis Structure . . . . .	6

---

<b>2</b>	<b>Background Theory</b>	<b>7</b>
2.1	Acoustic Fish Telemetry . . . . .	7
2.1.1	Acoustic Transmission in Water . . . . .	7
2.1.2	Acoustic tags . . . . .	8
2.1.3	Acoustic Receivers . . . . .	10
2.1.4	Strategies for Acoustic Tags Positioning . . . . .	10
2.1.5	Mobile Multi-Receiver Fish Tracking . . . . .	11
2.2	Robotic Middleware . . . . .	13
2.3	The LSTS Toolchain . . . . .	14
2.3.1	IMC . . . . .	15
2.3.2	Neptus . . . . .	16
2.3.3	DUNE . . . . .	17
2.3.4	IMCProxy . . . . .	19
2.3.5	GLUED . . . . .	20
2.4	Geoinformatics and Cartography . . . . .	20
2.4.1	Geodetic Datums and Projections . . . . .	20
2.4.2	The S-57 ENC format . . . . .	21
2.4.3	Grid-based Decomposition . . . . .	23
2.4.4	Visualizing Spatial Data in QGIS . . . . .	24
2.4.5	Manipulating Spatial Data in FME Desktop . . . . .	25
2.5	Path Planning for Marine Vessels . . . . .	26
2.5.1	Complexity Measures for Algorithms . . . . .	27
2.5.2	The A* Graph Search Algorithm . . . . .	28
2.6	Online Services . . . . .	30
2.6.1	Wiki . . . . .	30
2.6.2	Grafana . . . . .	30
2.7	Database Systems . . . . .	32
2.7.1	Relational Databases . . . . .	32
2.7.2	Indexing and Keys . . . . .	33
2.7.3	SQL . . . . .	33
2.7.4	SQLite . . . . .	33
2.7.5	Time-Series Database Management Systems . . . . .	33
2.7.6	InfluxDB . . . . .	34
<b>3</b>	<b>Design of a Robotic Fish Tracking Vehicle System</b>	<b>35</b>
3.1	The NTNU Fish Otter System for Fish Tracking . . . . .	35
3.1.1	Design Problem Definition . . . . .	35
3.1.2	System Requirements for the NTNU Otter Project . . . . .	36
3.2	Hardware Design . . . . .	40
3.2.1	The Fish Otter ASV . . . . .	40
3.2.2	Supporting Hardware . . . . .	44
3.3	Software Design On the NTNU Fish Otter ASV . . . . .	46
3.3.1	Operating System . . . . .	46
3.3.2	DUNE on the NTNU Otter . . . . .	46
3.4	The Otter Server . . . . .	48



---

<b>4</b>	<b>Designing a Database for Providing A Priori Environmental Information</b>	<b>49</b>
4.1	Available data . . . . .	49
4.2	Storing and Making Data Accessible . . . . .	50
4.3	Data Extraction . . . . .	50
4.3.1	Extracting Points of Interest from S-57 ENCs . . . . .	51
4.3.2	Two-dimensional Batymetric Grids . . . . .	51
4.3.3	Extracting Vertices from S-57 Polygons . . . . .	54
4.4	Discussion . . . . .	55
4.4.1	Choosing the Resolution . . . . .	55
4.4.2	Optimizing Database Performance . . . . .	56
4.4.3	Weather Forecast Services . . . . .	57
<b>5</b>	<b>Utilizing the A Priori Data in LSTS Toolchain</b>	<b>59</b>
5.1	Implementation Design Choices in DUNE . . . . .	59
5.2	Situational Awareness Classes in DUNE . . . . .	60
5.2.1	Data Containers . . . . .	60
5.2.2	PointOfInterest . . . . .	60
5.2.3	DensifiedVertices . . . . .	62
5.2.4	TwoDGrid . . . . .	62
5.2.5	SQL Queries . . . . .	63
5.2.6	Discussion . . . . .	65
5.3	Anti-Grounding Task in DUNE . . . . .	65
5.3.1	By two-dimensional grids . . . . .	65
5.3.2	By DEPARE Contour Vertices . . . . .	67
5.3.3	Discussion . . . . .	67
5.4	Path Planning in DUNE . . . . .	68
5.4.1	Results . . . . .	72
5.4.2	Discussion . . . . .	73
5.4.3	Further Work . . . . .	75
5.5	Situational Awareness in Neptus . . . . .	78
<b>6</b>	<b>Software Integration</b>	<b>81</b>
6.1	Hydrophone Integration . . . . .	81
6.1.1	Assessing the Implemented Hydrophone Synchronization . . . . .	82
6.1.2	Improvements to the DUNE Hydrophone Task . . . . .	85
6.2	Preparation for Multi-Vessel Operations in the LSTS Toolchain . . . . .	85
<b>7</b>	<b>Communications Design</b>	<b>87</b>
7.1	Computer Networking . . . . .	87
7.1.1	5GHz Wireless . . . . .	87
7.1.2	4G LTE Wireless . . . . .	88
7.1.3	VPN Setup and Deployment . . . . .	89
7.1.4	IMCProxy . . . . .	90
7.1.5	Discussion . . . . .	90
7.2	Website for sharing and analysing data . . . . .	91
7.2.1	System Design . . . . .	92

---

---

7.2.2	The IMC to InfluxDB Ingestor Design . . . . .	92
7.2.3	Result . . . . .	93
7.2.4	Discussion . . . . .	93
7.3	Summary . . . . .	95
<b>8</b>	<b>Conclusion and Further Work</b>	<b>97</b>
8.1	Further Work . . . . .	97
	<b>Bibliography</b>	<b>98</b>
	<b>Appendices</b>	<b>107</b>
<b>A</b>	<b>Additional Background and Theory</b>	<b>107</b>
A.1	Networked Remote Device Access . . . . .	107
A.1.1	VPN . . . . .	107
A.1.2	Network Device Management . . . . .	108
A.2	Summary of TTK4550 Project Work on the NTNU Fish Otter . . . . .	108
A.3	S-57 ENC Object Quick Reference . . . . .	109
<b>B</b>	<b>FME Workbenches</b>	<b>111</b>
<b>C</b>	<b>RUT950 Configuration</b>	<b>115</b>
<b>D</b>	<b>Source Code</b>	<b>117</b>
D.1	GitHub Repositories . . . . .	117
D.2	IMCProxy Data Ingestor . . . . .	117
D.3	DUNE . . . . .	121
D.3.1	LocationData Class . . . . .	121
D.3.2	DensifiedVertices Class . . . . .	124
D.3.3	TwoDGrid Class . . . . .	128
D.3.4	PointsOfInterest Class . . . . .	137
D.3.5	PathPlanner Class . . . . .	139
D.3.6	Grounding Task . . . . .	146
D.3.7	ThelmaBiotel Hydrophone Task . . . . .	152
D.3.8	Specific Otter Configuration File . . . . .	165
D.3.9	Common Otter Configuration File . . . . .	166

# List of Tables

1.1	Overview of the phases in a Fish Otter mission. . . . .	3
2.1	DPPM code types. . . . .	9
2.2	S-57 naming convention used by Kartverket. . . . .	23
2.3	Description of the FME Workbench nodes used in this thesis. . . . .	26
2.4	Relational database schema example with values. . . . .	33
3.1	Operational Limits of the Fish Tracking System. . . . .	37
3.2	Overview of the hardware components of the NTNU Fish Otter ASV. . . . .	42
3.3	Overview of the hardware components used in the control box of the NTNU Fish Otter ASV. . . . .	44
4.1	Datasets available for two-dimensional grids. . . . .	52
5.1	C++ classes implemented in DUNE to get data from the DB. . . . .	60
5.2	PathFinder benchmark results. . . . .	73
5.3	PathFinder benchmark results on 2x2 grid based on 5H1620 DEPARE object. . . . .	75
5.4	Performance comparison with differently scaled heuristic function. . . . .	75
6.1	TB Live comparison to TBR700RT. . . . .	81
6.2	The IMC addresses of the NTNU Fish Otters. . . . .	86
7.1	Planned subnetworks for the NTNU Fish Otter System. . . . .	89
A.1	Overview of the S-57 objects used in this report (Source: [55]). . . . .	110

---

# List of Figures

2.1	Thelma Biotel Acoustic tags (Source: [28]). . . . .	8
2.2	An example of DPPM S256 fish tag registration. . . . .	10
2.3	Time difference of arrival for $n = 4$ . . . . .	12
2.4	The console window of Neptus. . . . .	17
2.5	The Neptus MRA. . . . .	18
2.6	IMCProxy network overview. . . . .	20
2.7	IHO S-57 theoretical data model (Source: Figure 1.1 and 2.1 from [13] combined). . . . .	22
2.8	Grid-based decomposition with squares. . . . .	24
2.9	QGIS main window showing a S-57 ENC on top of the OpenStreetMap. . . . .	25
2.10	The basic path planning framework of an intelligent marine surface vessel (Based on figure 1 from [6]). . . . .	27
2.11	Grid movement examples for searching algorithm. . . . .	29
2.12	A screenshot of the DokuWiki made for the Otter, accessed 09/07/2020. . . . .	31
2.13	A screenshot of the Grafana querying interface. . . . .	32
3.1	Fish Tracking Scenario (The additional satellites needed in a GNSS is omitted for simplicity). . . . .	36
3.2	A picture of the Fish Otter with component labels (Before the hydrophone upgrade mentioned in section 6.1). . . . .	41
3.3	NTNU Fish Otter Hardware. . . . .	41
3.4	Figure showing the updated wiring in the NTNU Fish Otter control box. . . . .	45
3.5	NTNU Fish Otter Software. . . . .	46
3.6	Overview of the most important tasks used in the DUNE configuration of the Otter, as well as an (incomplete) view of communications between them. . . . .	47
4.1	A selection of points of interest from NO ENCs. . . . .	51
4.2	Square limits. . . . .	51
4.3	The bathymetry readings from GeoNorge. . . . .	53

---

4.4	Two-dimensional array of points based on DEPARE from NO4 ENCs. The purple points all have DRVAL1 and DRVAL2 defined. . . . .	54
4.5	Two-dimensional array of points based on DEPARE from NO4D1620 with higher resolution in shallow areas. 10m·10m in shallower than 20m, 50m·50m for depth between 20m and 100m, and 75m·75m for depths deeper than 100m. . . . .	55
4.6	Vertices on contour around depth areas (DEPARE) from NO4 ENCs. DRVAL1 and DRVAL2 is defined for all the red points. . . . .	56
5.1	Class diagram showing all the C++ classes implemented in DUNE to get data from the DB. . . . .	61
5.2	Class diagram showing the PointsOfInterest C++ class developed for DUNE. . . . .	61
5.3	Class diagram showing the DensifiedVertices C++ class developed for DUNE. . . . .	62
5.4	Class diagram showing the TwoDGrid C++ class developed for DUNE. . . . .	64
5.5	Plan used to demonstrate anti-grounding transect checking. . . . .	66
5.6	The results of running <code>checkTransect</code> on Munkholmen, Trondheim with grids of depth. Red points are detected land area, other colors show the navigable waters of the transects (with no depth limit set). . . . .	67
5.7	The results of running <code>checkTransect</code> on Munkholmen, Trondheim with contour vertices. Points show returned vertices with known DRVAL1 and DRVAL2 for each transect. . . . .	68
5.8	Area the Anti-Grounding implementation considers grounding. Inner area: No data. Outer area, depth below 5m. . . . .	69
5.9	Class diagram showing the PathPlanner C++ class developed for DUNE. . . . .	70
5.10	Benchmark paths found by path planner. . . . .	72
5.11	A 2x2m grid made of 5H1620 ENC DEPARE object. . . . .	73
5.12	Paths created by PathPlanner on high-resolution grid. . . . .	74
5.13	Paths showing the effect of increasing the heuristic scaling D. . . . .	76
5.14	Effect on searched area by increasing the heuristic scaling D. . . . .	77
6.1	The setup used in the PPS experiment. . . . .	83
6.2	Comparison between SLIM and RPIPPS synchronization of TBR700RT (389ms had to be added to to RPIPPS to account for constant offset). . . . .	84
7.1	The Teltonika RUT950 4G router used in the fish Otter (Picture from Teltonika). . . . .	88
7.2	VPN topology. . . . .	90
7.3	IMCProxy network with spectator functionality. . . . .	92
7.4	An example screenshot of the Grafana Dashboard. . . . .	94
7.5	A Grafana Dashboard showing fish tag detections at the locations the registrations were made. . . . .	95
7.6	Communication overview after thesis work completion. . . . .	96
B.1	FME flow for reading .xyz files in utm33 projection, and reprojecting them to WGS84, then saving in database. . . . .	111

---

---

B.2	FME Workbench for extracting points of interest from POI. . . . .	112
B.3	FME workbench that extracts vertices from a polygon, makes more vertices at uniform intervals around the polygon, and writes the result to a SQLite3 DB. . . . .	112
B.4	FME workbench that creates point cloud with values from S-57 polygon areas. . . . .	113
C.1	RUT950 port forwarding configuration. . . . .	115
C.2	RUT950 OpenVPN configuration. . . . .	116

---



# List of Code

2.1	Example of how IMC messages are defined in IMC.xml. . . . .	15
2.2	Dune configuration file example. . . . .	19
2.3	Pseudocode for the graph searching A*-algorithm (Source: [2]). . . . .	28
2.4	Pseudocode for the function expanding a child (Source: [2]). . . . .	28
5.1	SQL "where" clause created by makeSquareWhereClause. . . . .	63
5.2	SQL query that returns a square. . . . .	63
5.3	SQL query that returns the four closest depths around the vessel. . . . .	64
D.1	ImcToInfluxDB.java . . . . .	117
D.2	LocationData.hpp . . . . .	121
D.3	LocationData.cpp . . . . .	122
D.4	DensifiedVertices.hpp . . . . .	124
D.5	DensifiedVertices.cpp . . . . .	125
D.6	TwoDGrid.hpp . . . . .	128
D.7	TwoDGrid.cpp . . . . .	130
D.8	PointsOfInterest.hpp . . . . .	137
D.9	PointsOfInterest.cpp . . . . .	137
D.10	PathPlanner.hpp . . . . .	139
D.11	PathPlanner.cpp . . . . .	140
D.12	Task.cpp Grounding Supervisor. . . . .	146
D.13	Task.cpp ThelmaBiotel Hydrophone Sensors. . . . .	152
D.14	ntnu-otter-04.ini . . . . .	165
D.15	basic.ini . . . . .	166



NTNU  
Norwegian University of  
Science and Technology

Faculty of Information Technology and  
Electrical Engineering  
Department of Engineering Cybernetics

## MASTER'S THESIS ASSIGNMENT (30 Stp.)

Name: Nikolai Lauvås  
Program: Engineering Cybernetics  
Title: Design and development of robotic fish tracking vehicle  
Title (Norw.): Design og utvikling av robotisert fiskesporingsfartøy

### Project description

The overall aim of the project is to make deep integration of an acoustic fish telemetry receiver in an autonomous surface vehicle (ASV Otter catamaran) to create a novel platform for robotic search, localization and tracking of migrating fish. The master project builds on results achieved in a preceding specialization project and aspires to develop the prototype further towards a fully operational autonomous vehicle system that includes all required capabilities related to vehicle control, communications, payload integration, autonomy and endurance. The project should target the following tasks:

- Integration of the acoustic telemetry receiver (payload) in the vehicle and verification of the associated time synchronization mechanism with DUNE
- Integration of a robust and persistent communication interface for the vehicle based on WiFi and 4G cellular communication. Discuss vehicle behaviour strategies in case of communication loss.
- Provide onboard situational awareness in terms of electronic charts and develop efficient algorithms for path planning and safe navigation (anti-grounding)
- Develop a system for Internet-based real time access and visualization of vehicle data
- Discussion and documentation of vehicle properties. Updating of vehicle's Wiki pages.

Project start: 3 February 2020  
Project due: 28 June 2020  
Host institution: NTNU, Department of Engineering Cybernetics  
Supervisors: Jo Arve Alfredsen, NTNU/DEC  
Alberto Dallolio, NTNU/DEC

Trondheim, 29 January 2020  
Jo Arve Alfredsen

# Abbreviations

**ASV** - Autonomous Surface Vehicle.

**AUV** - Autonomous Underwater Vehicle.

**CAN** - Controller Area Network.

**CSV** - Comma-separated values.

**DB** - Database.

**DBMS** - Database management system.

**DEC** - Department of engineering cybernetics[at NTNU].

**DPPM** - Differential pulse-position modulation (PPM).

**DUNE** - DUNE: Unified Navigational Environment.

**ENC** - Electronic Navigational Charts.

**ETRS** - European Terrestrial Reference System, also called EUREF89 .

**GDOP** - Geometric dilution of precision.

**GIS** - Geographic information system.

**GNSS** - global navigation satellite system.

**GPIO** - General Purpose Input/Output.

**GPL** - GNU General Public License.

**GPS** - Global Positioning System.

**GUI** - Graphical user interface.

**HTTP** - Hypertext Transfer Protocol.

**ILOS** - Integral Line-of-Sight Guidance.

**IMC** - Inter module communication.

**IMU** - Inertial measurement unit.

**IO** - Input/Output

**ISO** - International Organization for Standardization.

---

**LOS** - Line-of-Sight Guidance.

**LSTS** - "Laboratório de Sistemas e Tecnologia Subaquática", or Underwater Systems and Technology Laboratory at the University in Porto.

**LTE** - Long-Term Evolution.

**MIMO** - multiple-input and multiple-output.

**NTNU** - "Norges teknisk-naturvitenskapelige universitet", or Norwegian University of Science and Technology.

**NMAP** - The Norwegian mapping authorities, named Kartverket in Norwegian.

**OS** - Operating System.

**OSI** - Open Systems Interconnection.

**PID** - Proportional–integral–derivative controller.

**PoE** - Power over Ethernet.

**PPS** - Pulse-Per-Second.

**RAM** - Random-access memory.

**RFC** - Request for Comments, standards memorandums for Internet related standards.

**ROV** - Remotely operated underwater vehicle.

**RPI** - Raspberry Pi.

**RPM** - Revolutions Per Minute.

**SBAS** - Satellite-based augmentation systems.

**SBC** - Single-board computer.

**SISO** - Single-input single-output.

**SoC** - System on Chip.

**SPI** - Serial Peripheral Interface.

**SQL** - Structured Query Language.

**TCP** - Transmission Control Protocol.

**TDOA** - Time difference of arrivals.

**UAV** - Unmanned aerial vehicle.

**UDP** - User Datagram Protocol.

**URL** - Uniform Resource Locator.

**USB** - Universal Serial Bus.

**USV** - Unmanned Surface Vehicle, not necessarily autonomous.

**UTM** - Universal Transverse Mercator coordinate system.

**VPN** - Virtual private network.

**WGS** - World Geodetic System.

# Introduction

In a country with as long a coastline as Norway, the ocean naturally becomes an important resource. The fishing industry, and in more recent years, aquaculture, have for centuries provided the Norwegian people with both food and valuable goods for export. To ensure that the industry can thrive, while also minimizing the impact on wildlife, good management is dependent on knowledge about life beneath the surface. One aspect of this is observing the movement of individual fish over time, in order to gain insights into their behavior and how human activities such as aquaculture affect it. The Fish Otter project at NTNU aims to develop an autonomous platform for gathering spatiotemporal data relating to individual fish by employing a fleet of autonomous surface vehicles (ASV). The ASVs are equipped with acoustic fish telemetry receivers that can detect and distinguish acoustical fish tags that has been attached to fish.

## 1.1 Background

At DEC NTNU<sup>1</sup>, there is a long tradition for doing research on acoustic fish telemetry. This dates back to its founder, Jens Glad Balchen, and his experiments on tracking the life and behavior of fish in Hopanvågen during the 1970s. In one of these experiments, he developed what he called a fish spy. This was what is now called an unmanned surface vehicle (USV), and its goal was to follow right above a fish that had been previously tagged with an acoustic transmitter. The position of the vehicle would then be collocated with the position of the tracked fish, with only vertical position differing. To position the fish, directional hydrophones was used, with differences in signal strength being used to decide in what direction the USV was to go. Lack of funding ultimately led to the project being abandoned after several development iterations [3], but research on acoustic fish telemetry have continued at DEC.

Developments in digital technology has been made since the fish spy project was abandoned, and mass production has also reduced the cost of components and equipment. This

---

<sup>1</sup>Department of Engineering Cybernetics, Norwegian University of Science and Technology.

has lead to a significant increase in the available computing power, paving way for improvements both in the acoustic fish tags, and in hydrophones. Digital hydrophones now have the ability to not only register sound waves, but also automatically detect tag registrations in them. In addition, satellite based positioning systems like GPS has made accurate global positioning available, meaning that location can be added as context for the tag detections.

## **1.2 Goal and State of the Fish Otter Project**

The main goal of this thesis, is to progress the NTNU Fish Otter Project on its ultimate goal, which is providing an autonomous multi-agent system for search, localization and persistent tracking of acoustic fish tags beneath the water surface. The information such a system provides, can in collaboration with scientists from other research disciplines like marine sciences or aquaculture potentially provide a better understanding of the spatiotemporal distribution and behaviour of fish and other aquatic animals.

### **1.2.1 Outline of a Fish Tracking Mission**

With the help of the thesis supervisor Jo Arve Alfredsen, the author has created an overview of how a fish tracking mission with the NTNU Fish Otter ASVs is to be conducted, which has been divided into the phases shown in table 1.1. In section 3.1.2, the phases are then used as the basis in a discussion about the system requirements.

### **1.2.2 Project State Before Thesis**

Prior to the work described in this thesis, the NTNU Fish Otter ASV hardware had been procured, and the controlling hardware specified and installed. During a project in the autumn of 2019, the author integrated the basic hardware components, installed a GNU/Linux based operating system, and made an integration of the vessel for use in the LSTS toolchain, with the middleware DUNE running on the vessel. Using this setup, the vessel could successfully perform basic maneuvers such as going to waypoints and remote operation in an controlled manner. The tuning of the controllers were left as further work. Telemetry from the vessel was made available to the operator through the console Neptus, and the payload returned tag registrations. The synchronization of the hydrophone was implemented, but not verified.

A more detailed summary is available in section A.2.

### **1.2.3 Thesis Contributions**

This thesis contributes to the project by the implementation of an anti-grounding supervisor, a global path planner to DUNE, a web based telemetry monitoring solution, and improvements to the hydrophone integration.

#	Mission Phase	Description
1	Transport	Bring the equipment to the deployment area.
2	Deployment	Assemble and prepare the vessel. Ensure supporting systems (Server/Operator console) are running. Launch vessels.
3	Travel to search area	Vessels travel from deployment area to search area.
4	Collaborative search	Search area is divided between vessels. Ends after first tag detection. If first search yields no result, perform predetermined action, like searching again or extending search area.
5	First tag contact	First contact with acoustic tag. The detecting vessel waits or tries single vessel tag positioning. Other vessels congregate to its vicinity, entering formation.
6	All vessel tag contact	Enter tag formation to commence collaborative fish tracking.
7	Collaborative fish tracking	Keep formation around tag to estimate its position.
8	Tracking end	Mission finished or battery level low – wait for pickup or return to predetermined point ( like deployment point ).
9	Berthing/ Dismantling equipment	Removing the vessel from the water.
10	Transport	Pack up equipment and leave deployment area.

**Table 1.1:** Overview of the phases in a Fish Otter mission.

## 1.3 State of the Art

Multiple unmanned surface vehicles (USV) has been developed over the years in order to satisfy an increasing demand by scientific, commercial and military interests. An overview of some USVs, as well as a discussion of the enabling technologies making them a viable solution for an increasing range of applications, is given in [4].

A more recent overview is given in [5], spanning from 1985 and to the publishing year (2016). The paper also gives an introduction to most of the subjects that are relevant in the development of an USV. On the subject of cooperative USV systems, which the NTNU Fish Otter project is slated to become one of, it is noted that although there are some systems that has been validated through experiments, most of the research is limited to simulations.

Autonomous Surface Vehicle (ASV) is an often used synonym for USV. The name puts more of an emphasis on the autonomous capabilities that has been developed for USVs. In [6], a survey is presented on how marine surface vessels can achieve a higher degree of

autonomy<sup>2</sup>. Among the topics, path planning is listed as important.

### 1.3.1 Global Path Planning for Marine Vessels

Path planners for marine vessels is often divided into global and local path planners, where the global considers the overall goal, and the local takes care of aspects observed while traveling. As an A\* based global path planner is developed in this thesis, this will be the focus in this section.

When designing a global path planner, having a priori environmental data is a prerequisite. A grid-based map decomposition is often used to represent the operational area of the vessel [7] [8], on which a path is generated by finding a sequence of nodes in the grid that takes the vessel from a point towards its goal.

The heuristic A\*-search algorithm is mentioned in both [5] and [6] as a popular approach for global path planners in marine vessels. In [9], electronic navigational charts (ENC) are used to create a grid with an octree structure between grid nodes. Based on this grid, the A\*-algorithm creates a path between a starting point and the goal. Another A\* based implementation is given in [10], which also considers water current, traffic separation and berthing when planning a path. Other variations include: the Hybrid A\* considering non-holonomic constraints in the algorithm [11], Theta\* which is an any-angles A\* based algorithm that does not constrain movements along grid edges [8] and the ARC-Theta\*, which constrains the yaw angular rate allowed in the Theta\* algorithm [12].

### 1.3.2 A Priori Situational Awareness for Unmanned Marine Vessels

A system for providing environmental data is a way to increase the a priori situational awareness for a marine vessel, and can also be used as the basis for the global path planning. For the ocean, vector-based electronic navigational charts (ENCs) contain all data deemed necessary for human navigation, and the International Hydrographic Organization (IHO) has standardized the format for transferring such data in the S-57 standard [13].

In the S-57 format, spatial information is stored as vectors. To make use of the data in a faster and more practical way for autonomous maritime vessels, the data is often transformed before use. In prior works, various approaches have been taken:

Unpacking the data with an open-source ISO8211 library and creating a custom parser is performed in [9], resulting in a grid of the operational area.

Grids are also discussed in [14], both regular and irregular. In the irregular grids, the resolution is increased for certain areas where a higher detail level is required, such as around obstacles.

In [15], the open-source GDAL<sup>3</sup>/OGR<sup>4</sup> Python API is used to extract obstacle information on a polygon based obstacle map. This is then used in combination with a collision avoidance system in the ROS middleware.

For the MOOS middleware, [16] stores the processed ENC data in a database. This is then used to increase sensor reliability in telling where obstacles are expected. Finally, the information is used in a local path planner to avoid the obstacles.

---

<sup>2</sup>Increasing autonomy is defined as improving the intelligence in the article.

<sup>3</sup>Geospatial Data Abstraction Library

<sup>4</sup>The library in GDAL handling vector IO.



### 1.3.3 Mobile Acoustic Fish Tracking

Static networks of acoustic receivers have for a long time been used to gather spatiotemporal data relating to aquatic animals. With the development of more capable vehicles for use in water, placing the acoustic receivers on mobile platforms has many advantages, and has therefore become a subject of research.

Downloading telemetry data from a fish tag is mentioned as one of the motivating scenarios by the European GREX project for developing a heterogeneous vehicle system. When discussing the impact of the project, [17] states that: *Critical impacts include the use of spatial behaviour in fisheries stock assessment and the design of Marine Protected Areas.*

[18] claims to be the first application of AUVs in collecting telemetry from fish at large, and presents multiple methods that can be used to increase the usefulness of the data.

In a more recent integration with an acoustic receiver in an AUV, [19] proposes to use an extended Kalman filter or a particle filter as iterative estimators for position. In an experiment with a stationary fish tag, the AUV travels about the estimated location, in order to gather multiple tag detections. Localization errors below 20 m compared to a GPS position is achieved after 20 transmissions from the fish tag. Using tags that transmitted every seven seconds, that means it would take 140 seconds to locate the tag, so this method is only suited for slower moving targets.

Using multiple vehicles results in multiple detections being made for a single tag signal, reducing the time and increasing the accuracy of tag positioning. This was proposed in [20], and has been further developed in multiple experiments since. In [21], a proof-of-concept is demonstrated through an experiment with a formation of unmanned vehicles carrying hydrophones. The formation follows another vehicle with a acoustic tag mounted below its waterline, estimating its position using the time difference of signal arrival. Comparing the estimated position with the GPS position of the tag carrying vehicle achieves a median localization error of 4.7 m, and an average accuracy of 6.34 m.

Another experiment was performed in [22], with three USVs carrying acoustic receivers and a fourth USV carrying a submerged acoustic fish tag. Using an eXogenous Kalman filter, the location of the fourth USV is estimated and compared to other estimators and the GPS position. After the Kalman filter has stabilized, it's possible to locate the fish tag. It also demonstrates the performance benefit of using an eXogenous Kalman filter over an extended Kalman filter.

A more exotic experiment done tracking leopard sharks with an AUV is described by [23]. The system is further extended in [24] to use multiple AUVs that collaborates to position the tagged shark, using a Particle Filter. An important difference between these AUVs, and the one described in [19], is that these use stereo hydrophones, while [19] used a mono hydrophone. Using a stereo hydrophone had the advantage of making relative bearing from the AUV to the acoustic tag available.

## 1.4 Thesis Structure

The structure of this thesis is laid out as follows: **Chapter 2** gives the reader an introduction to concepts used in the following chapters. **Chapter 3** presents the NTNU Fish Otter system and its purpose, which is the motivation and background for this thesis. **Chapter 4** covers the design and building of a database containing a priori environmental data to be used onboard the vessel. **Chapter 5** describes how the database is made available in the onboard middleware of the vessel, as well as the design and implementation of anti-grounding systems and a global path planner for the vessel. **Chapter 6** details other software development made for the middleware, mainly consisting of improving the hydrophone integration. **Chapter 7** describes the design, implementation and deployment of an online vessel monitoring system that facilitates sharing telemetry with collaborating researchers. **Chapter 8** concludes the thesis and suggests further work that could improve and extend its results.

Further details and background theory for this thesis, such as developed code and configuration files is available in the Appendices.

# Background Theory

## 2.1 Acoustic Fish Telemetry

To do remote sensing on aquatic animals and their surroundings, the field of fish telemetry has emerged, often combined with acoustic data transmission. The main parts of an acoustic fish telemetry system, is an acoustic transmitter (AT) attached to the animal, as well as an acoustic receiver (AR) that reads the transmitted data. The AT is often called a tag, because it identifies and give information about the subject it's attached to. Various sensors has been integrated in fish tags, measuring heartbeat, jaw movement, water salinity or temperature<sup>1</sup> to reveal either something about the fish or its environment. This section aims to give a short introduction on the subjects most relevant to the NTNU Fish Otter project. For a more detailed introduction about acoustic transmission and challenges in fish telemetry, see [25]. A more recent discussion on the wider topic of aquatic animal telemetry is given in [26].

### 2.1.1 Acoustic Transmission in Water

For wireless data transmission in water, acoustic transmission is often the preferred choice. Additional substances in the  $H_2O$ , like salts, plankton or air-bubbles, absorbs the high frequency waves used in radio transmission, resulting in range issues. Although the range of acoustic signals is also significantly reduced in saline water compared to fresh water, transmission from higher-powered ATs can still achieve a range in the order of kilometers, while for smaller, power constrained ATs, it can be as low as 200 meters<sup>2</sup>.

For detection of acoustic signals in water, the receivers achieve best reception when submerged in the medium, because of the different impedance between air and water causes most of the signal to reflect back into the water. This is also a problem between water and the seabed, making multi-path propagation a major hurdle in acoustic water

---

<sup>1</sup>This list is not exhaustive. For more alternatives, see [25] or [26]

<sup>2</sup>In many cases lower, due to disturbances in the medium.

transmission. For this reason, only the front of the transmitted signal can be timed precisely [25]. Another interesting phenomenon appears due to higher water density in deeper water, making the acoustic signal bend as it propagates towards the surface.

### The Speed of Sound in Water

The main components when deciding the speed of sound in water  $c$ , are salinity, pressure<sup>3</sup> and temperature. In the introduction of [27], multiple methods for estimating  $c$  are discussed.

A simple equation is given by Leroy’s empirical formula for calculating the speed of sound in water:

$$c = 1492.9 + 3 \cdot (T - 10) - \frac{6 \cdot (T - 10)^2}{10^3} + 1.2 \cdot (S - 35) - \frac{(T - 18) \cdot (S - 35)}{10^2} + \frac{D}{61} \quad (2.1)$$

where

$T$  = Water temperature in Celsius [ $C^\circ$ ]

$S$  = Water salinity in parts per thousands [ $ppt$ ]

$D$  = Water depth in meters [ $m$ ]

A newer equation is given by Leroy et al. in [27], including also the effect of latitude in the equation. The article also contains a useful table of sound speeds at different depths in various seas, showing values for  $c$  between  $1455m/s$  and  $1660m/s$  at extreme pressures/depths.

### 2.1.2 Acoustic tags

The acoustic tags consists of three main parts: power, transceiver and instrumentation. Power, in the form of a battery, is used to drive the sensors and data-conditioning circuits of instrumentation, as well as the transceiver which transmits the data.

The size of a tag has to be chosen according to the target it will be attached to, or implanted in [29]. For smolt, a large tag would alter their behavior, while a whale or a shark could carry larger tags. Advantages of larger tags are higher battery capacity, larger transducers and more space for instrumentation.

The tags used for in the Fish Otter project are digital tags, using the DPPM modulation, and only sends a message at regular intervals. Other options are continuously transmitting tags, or fully analog tags.



**Figure 2.1:** Thelma Biotel Acoustic tags (Source: [28]).

---

<sup>3</sup>Sometimes exchanged with depth for convenience.

## Transducers

The transducers used are piezoelectric, and have a resonance frequency that is governed by size, where larger size yields lower resonance frequency. The resonant frequency is used in the tags to achieve maximum signal power at a smaller energy consumption. For small tags, transducers have to be sized accordingly, and therefore, most often there is a relationship where larger tags transmit at lower frequencies, while higher frequencies are used on smaller tags. This in turn means that telemetry on smaller fish involves shorter transmission ranges [30]. For the tags developed by Thelma Biotel which is used in the Fish Otter project, the range of tag frequencies is between 63kHz and 77 kHz.

## DPPM

One way of mitigating the multi-path propagation problems, is to modulate the signal into a series of signal fronts/pulses. For the equipment used in the Fish Otter project, pulse-position modulation (PPM) is used. This also goes by the extended name, differential PPM (DPPM), because the information is encoded as the time-difference between two pulse positions. The protocol used in the tags provided by Thelma Biotel<sup>4</sup> use a similar DPPM protocol as is described in [30]. The short transmitting time minimizes energy consumption, which is an important aspect in tag design.

An example is given in figure 2.2, where a full-spectrum hydrophone was used to record one of the tags<sup>5</sup>, and then the tags transmitting frequency was isolated through a band-pass filter. The eight pulses can be seen as spikes in signal strength, and the information can be extracted by measuring time differences between spikes. For the figure, they are 0.36 - 0.38 - 0.50 - 0.40 - 0.46 - 0.42 - 0.52[seconds].

The first time difference specifies the code type, used to decide how the remainder is to be interpreted. As an example of this, 360ms is the code type S256, and the data is interpreted using the formula:

$$HexValue = \frac{D - [GT]}{[BT]} = \frac{D - 0.38}{0.02} \quad (2.2)$$

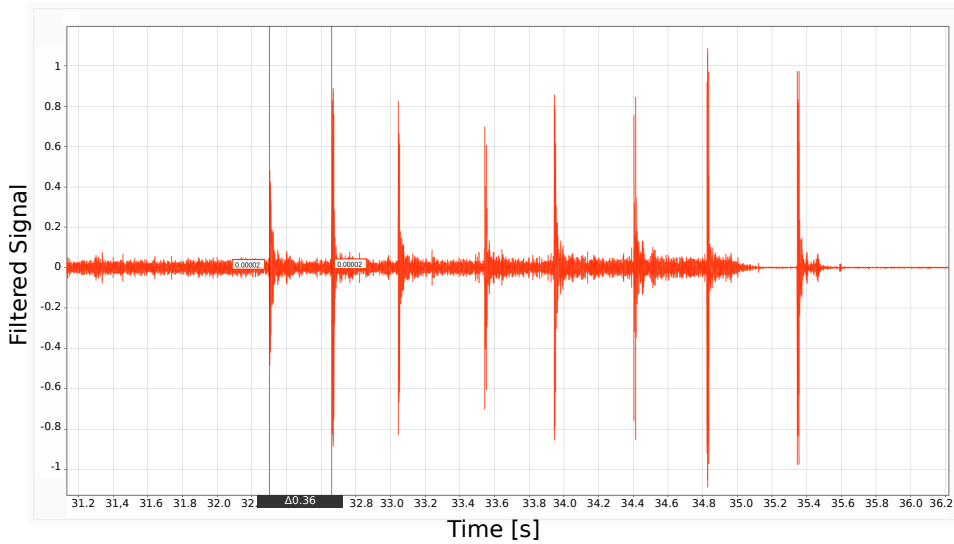
where  $D = \text{Pulsetimedelta}$ ,  $[GT] = \text{GuardTime}$  and  $BT = \text{Bintime}$ . Decoding the pulse train from figure 2.2 gives the result: "S256 - 0 - 6 - 1 - 4 - 2 - 7".

Code Type	R64K	S256	R04K	R256
Time between two first pulses	320	360	380	401
Guard Time	340	380	400	418
Bin time	20	20	20	22
Pings	8	8	7	6

**Table 2.1:** DPPM code types.

<sup>4</sup>Which are used in the Fish Otter project

<sup>5</sup>The data was collected in one of the sea trials with the Otter from [1] with an OceanSonics icListen HF hydrophone.



**Figure 2.2:** An example of DPPM S256 fish tag registration.

### 2.1.3 Acoustic Receivers

Acoustic receivers, called hydrophones, are in their simplest form just a microphone used in water. Like microphones in the air, there are differences in how directional the hydrophone is, which can classify the hydrophone as directional or omnidirectional. Directional hydrophones detect signals coming from one direction, and because of this property, can be used to find the direction of the tag. Omnidirectional hydrophones detect signals in all directions, and gives information about tag presence in the vicinity of the AR. These are more suited for use on buoys or where a wide detection area is needed. Combining multiple readings from omnidirectional hydrophones has to be performed to get more accurate tag localization.

#### Thelma Biotel Hydrophones

On the Fish Otter ASV, the hydrophones used are delivered by Thelma Biotel. The TBR700RT and the TB Live hydrophones are omnidirectional and uses digital signal processing (DSP) to analyze the acoustic waves, and detect tags transmitting using the DPPM protocol. The messages are then decoded, and sent over a NMEA 0183 inspired format. In addition to the tag detections, the signal-to-noise ratio is given, as well as an internal temperature sensor reading.

The tag frequencies supported are between 63 kHz and 77 kHz.

### 2.1.4 Strategies for Acoustic Tags Positioning

Due to heavy attenuation of radio waves in saline water, satellite positioning does not work in tags below the surface. For animals that surface at regular intervals, positioning at these

moments might be an option, but for the rest of the time, other positioning strategies must be developed.

There are multiple strategies for positioning acoustic tags below the surface, using both omnidirectional and directional hydrophones.

### **Directional Hydrophone Tag Positioning**

With directional hydrophones, a manual fish tracking method can be to have a human operator that searches with the hydrophone in the water, and steers in the direction where signal strength peaks. To accurately position a tag with this method, it's necessary to hover with the hydrophone right above the tag, which could affect the behavior of the animal, due to noises made by the operator or his vessel. Another drawback of this method, is that it's labor intensive, making long duration series of data hard to gather. Yet another drawback is that tags not transmitting continuously would be harder to locate. Tags that continuously transmit needs more energy, and thus can be expected to deplete the battery faster.

Extending the system to use multiple directional hydrophones, like the fish spy, is a way to enhance the system, making automation feasible, but would still come with the drawback of having to hover over the fish for precise positioning.

### **Omnidirectional Hydrophone Tag Positioning**

For long term tag detection, a commonly used method is placing multiple omnidirectional acoustic receivers at strategical locations, or at regular intervals in a grid [31]. The position of the receivers are known, so its inferred that when a detection is done, the animal is in the vicinity of the receiver. The data may also be processed, in order to estimate a more accurate fish position from multiple sensor detections, through multilateration<sup>6</sup>. Some drawbacks with these fixed arrays approach are; the area of interest has to be decided ahead of time, and it's expensive/impractical to have too large of a tracking field. Also, if the tagged fish wanders outside the tracking field, the tag position is unknown.

## **2.1.5 Mobile Multi-Receiver Fish Tracking**

To expand the operational area of a fish-tracking system with omnidirectional hydrophones, mobile acoustic receivers are introduced. The mobility is achieved by having AUVs, USVs or ROVs<sup>7</sup> as platforms carrying the hydrophone. Through combining multiple readings from receivers at strategic locations performed with collaborating vehicles, the accuracy of the position estimates are improved.

Such a system would still require the vessels to follow the moving animal, but would allow for them to keep a distance to avoid disturbing it.

---

<sup>6</sup>Described in section 2.1.5.

<sup>7</sup>UAVs could also be introduced, where a hydrophone beneath the body of the vehicle could be submerged at strategic locations.

### Multilateration

Multilateration is a localization method based on measuring the difference of the distances between an entity with unknown location  $\mathbf{r}$  and  $n$  entities with known location  $\mathbf{s}_n$ . Often, the true range is unknown, so a pseudo-range has to be used. The pseudo-range between  $\mathbf{r}$  and  $\mathbf{s}_n \forall n$  is found by sending a signal from  $\mathbf{r}$  at  $t_r$ , and then registering the time of arrival  $t_{s_n} \forall n$ . The pseudo-range is then calculated with  $d_n = c(t_{s_n} - t_r)$  where  $c$  is the signal propagation speed.

If  $t_r$  is unknown, but  $t_{s_n} \forall n$  is known, the time difference of arrival (TDOA) can be used to estimate location, by using  $n \geq d + 1$ , where  $d$  is the number of dimensions unknown in  $\mathbf{r}$ . For positioning in three dimensions,  $n = 4$  is required, and the time difference of arrival can be calculated as  $\tau_{s_1} = 0$ ,  $\tau_{s_2} = t_{s_2} - t_{s_1}$ ,  $\tau_{s_3} = t_{s_3} - t_{s_1}$  and  $\tau_{s_4} = t_{s_4} - t_{s_1}$ , as shown in figure 2.3.

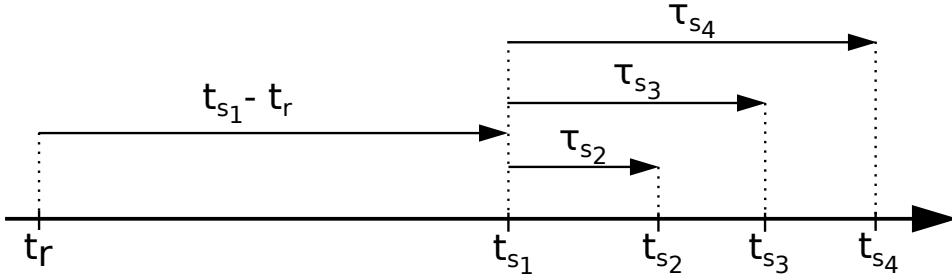


Figure 2.3: Time difference of arrival for  $n = 4$ .

The location  $\mathbf{r}$  can then be found by calculating  $\|\mathbf{r} - \mathbf{s}_n\| = c(\tau_n + t_{s_1} - t_r) \forall n = 1, 2, 3, 4$ . Defining the locations as  $\mathbf{r} = [x_r, y_r, z_r]^T$  and  $\mathbf{s}_n = [x_{s_n}, y_{s_n}, z_{s_n}]^T \forall n$  gives these four equations to solve for  $x_r, y_r, z_r$  and  $t_r$ :

$$\sqrt{(x_r - x_{s_1})^2 + (y_r - y_{s_1})^2 + (z_r - z_{s_1})^2} = c(t_{s_1} - t_r) \quad (2.3a)$$

$$\sqrt{(x_r - x_{s_2})^2 + (y_r - y_{s_2})^2 + (z_r - z_{s_2})^2} = c(\tau_{s_2} + t_{s_1} - t_r) \quad (2.3b)$$

$$\sqrt{(x_r - x_{s_3})^2 + (y_r - y_{s_3})^2 + (z_r - z_{s_3})^2} = c(\tau_{s_3} + t_{s_1} - t_r) \quad (2.3c)$$

$$\sqrt{(x_r - x_{s_4})^2 + (y_r - y_{s_4})^2 + (z_r - z_{s_4})^2} = c(\tau_{s_4} + t_{s_1} - t_r) \quad (2.3d)$$

To remove the unknown emission time  $t_r$ , equation 2.3a is applied to 2.3b, 2.3c and 2.3d, giving the three TDOA hyperboloid equations:

$$\sqrt{(x_r - x_{s_n})^2 + (y_r - y_{s_n})^2 + (z_r - z_{s_n})^2} - \sqrt{(x_r - x_{s_1})^2 + (y_r - y_{s_1})^2 + (z_r - z_{s_1})^2} = c(\tau_{s_n}) \quad \forall n = 2, 3, 4 \quad (2.4)$$

Or written in vector form:

$$\|\mathbf{r} - \mathbf{s}_n\| - \|\mathbf{r} - \mathbf{s}_1\| = c(\tau_n) \quad (2.5)$$

A solution for  $x_r, y_r, z_r$  lies at the intersection of the three hyperboloids of equation 2.4.



### Geometric Dilution of Precision

The Geometric Dilution of Precision (GDOP) characterizes how the geometry of the receivers<sup>8</sup> influence the accuracy of the position estimate, as described by [32]. Minimizing GDOP should therefore be part of the strategy for choosing the receiver positions. A thorough description of the position Configurations with the Lowest GDOP is given in [33], with an example of buoys used for underwater positioning. The problem of high GDOP in coplanar geometries, caused by only having receivers at the surface is discussed, which is relevant for systems with only surface vessels like the Fish Otter project.

### Estimators for Multilateration on Acoustic Transmitters in Water

In the scenario where multilateration is performed by mobile watercrafts, there are multiple challenges to overcome, such as:

- Uncertainty in receiver position (from GPS).
- Unknown and non-homogeneous medium where signal propagation speed varies, making the measured pseudo-range different from the true range.
- GDOP makes directly solving the TDOA equations have poor accuracy and precision.
- Intermittent signal loss from causes such as other sound sources leading to low signal-to-noise ratios.

To improve the performance of the position estimate, some filtering can be introduced. Due to the non-linearity in the equations, as shown in equation 2.4, the design of the estimator is further complicated.

The Kalman filter can be used for this purpose. The filter utilizes a linear model of the system, discrete in time, that estimates the internal system parameters through received measurements. To be able work on non-linear problems, the extended Kalman filter is used, which creates a linear model around the current state at iterations. An example of use in fish tracking is found in [19], where an AUV is utilized to localize an acoustic transmitter. As is mentioned in that text, this linearization makes the estimator less robust, with the possibility of incorrect convergence. This problem can be solved by using an eXogenous Kalman filter, as described in [34]. In [22], this was demonstrated for collaborative fish-tracking, where three USVs with acoustic receivers tries to locate a fourth USV carrying an acoustic tag.

Another possible estimator is a Particle Filter, used for fish tracking in [19] and [23]. This filter works by assuming multiple possible states, called particles, and spreading them through a pre-defined state-space. Iteratively, the particles are weighted according to their probability as new measurements become available.

## 2.2 Robotic Middleware

In the field of applied robotics and autonomous systems, an abstraction layer between the operating system and the software applications controlling different aspects of the robot

---

<sup>8</sup>For the case of acoustic fish tracking. In GNSS systems, the transmitter position is used for GDOP.

called robotic middleware is often used. This is motivated by reducing the complexity of developing software for robots, as well as allowing the same code to be used in different settings. This reusability is commonly achieved through the separation of software into multiple user-defined components. By limiting the purpose of each component to performing a single task, the developer needs only to consider one task at a time, resulting in a divide and conquer manner of developing. The communication between components is transparent to the developer, which makes it possible to analyze how a component affects the system.

Another benefit is that many of these middlewares are open-source, making inspecting the code possible for anyone, as well as implementing new features and fixing faults in the existing code base.

Among the available middlewares, ROS is probably the most popular judging by the number of components available [35]. In addition, a multitude of other options has been developed with different design philosophies or intended use. A literature study comparing multiple middlewares is available in [36], but does not include DUNE from the LSTS toolchain. A brief comparison between ROS and DUNE is available in [37].

At NTNU, the DUNE middleware has already been deployed in a number of vehicles, such as the NTNU AutoNaut [38] ASV, the AUVs of the AUR Lab<sup>9</sup> and the UAVs of the UAV lab<sup>10</sup>. Because interoperability with these vehicles was a desired property for the fish Otter ASV, the LSTS toolchain with DUNE was selected, and is described in some detail in section 2.3.

## 2.3 The LSTS Toolchain

The software used to control the NTNU fish Otters is based on the LSTS<sup>11</sup> toolchain, which was developed as a system for operating networked vehicles. It is designed in a modular manner, in order to be used in heterogeneous vehicles with different configurations. The common software base is written in a manner that lets multiple vehicles communicate, making multi-vehicle operations possible, as well as multiple human operators through consoles. The Otter is an autonomous surface vehicle (ASV), but the toolchain also support autonomous underwater vehicles (AUV), unmanned aerial vehicles (UAV), remotely operated vehicles (ROV) and more, as described in [39].

The components of the toolchain being used by the Otter ASV are called DUNE, Neptus, IMC, IMCProxy and GLUED. IMC is the communication protocol used by all other components, Neptus is the operators console and DUNE is the software running onboard the vehicles. IMCProxy is used to bridge IMC networks and GLUED is a lightweight GNU/Linux distribution. For the Otter, only the cross-compilation tools supplied with GLUED is used, not the GNU/Linux distribution.

---

<sup>9</sup>AUR lab homepage: <https://www.ntnu.edu/aur-lab/>

<sup>10</sup>UAV lab homepage: <https://www.itk.ntnu.no/english/lab/unmanned>

<sup>11</sup>“Laboratório de Sistemas e Tecnologia Subaquática”, or Underwater Systems and Technology Laboratory at the University in Porto.

### 2.3.1 IMC

Inter-Module Communications is the message protocol used throughout the LSTS toolchain for communications, as described in [40]. It is a transport-agnostic protocol, meaning that it can be carried over Ethernet, as well as acoustic transmission, satellite based transmission, cellular, and between threads running in an executable.

The protocol is message oriented, meaning that every concept of the language is split into one or more types of messages. An example of this is information about a battery, which is split into multiple messages, one for charge, one for voltage, one for current and so on. The same also applies to control signals between threads, so the high level path controller communicate with the lower level course, speed and depth controllers through IMC messages, who also send messages to the actuator controllers.

The IMC protocol also contains mechanisms for broadcasting the existence of a device supporting IMC, as well as discovering other devices. During this process, an Announce message is sent, containing device state and its capabilities. For the Otter, this means that as long as both the console and the vehicle is on the same network, a connection will be established automatically.

All IMC messages are divided into a header and a body part. The header has meta-data such as protocol version, sender, type of message and a timestamp. The senders are distinguished by two address fields, one for the program instance, and one for the sending entity. The standard comes with a list of pre-known addresses that is to be stored along with the software.

The body part contains the actual data, as defined by one of the message types supported by IMC. The messages of the IMC protocol is defined and documented in a single XML<sup>12</sup> file. Multiple tools are then used to create software bindings for the messages, like IMCJava<sup>13</sup> creating bindings for Java. An example of a message in the XML file named *IMC.xml* is given in code listing 2.1. When messages are sent over the network, it is serialized to consume less resources. Some IMC messages can also contain nested messages, like the message for sending a plan consisting of multiple maneuvers to a vehicle.

**Code 2.1:** Example of how IMC messages are defined in IMC.xml.

```

1  <message id="251" name="Voltage" abbrev="Voltage" source="
   vehicle" flags="periodic">
2  <description>
3  Report of electrical voltage.
4  </description>
5  <field name="Measured Voltage Value" abbrev="value" type="
   fp32_t" unit="V">
6  <description>
7  The value of the internal electrical voltage as
   measured by
8  the sensor.
9  </description>
10 </field>
11 </message>

```

<sup>12</sup>eXtensible Markup Language. Used for describing and storing data. Human readable and machine readable.

<sup>13</sup>IMCJava GitHub repository: <https://github.com/LSTS/imcjava>

To interface IMC messages, a publish–subscribe pattern is used, where messages are not sent directly between entities, but are classified by content. The middleware then distributes the published messages to subscribing entities. The seven main content categories, as defined by [40] are:

1. Mission control: Defines a mission through plans. The plans are the mission language in the toolchain.
2. Vehicle control: Vehicle interfaces allowing commands to be issued, information to be requested or state to be monitored.
3. Maneuver: The vehicle primitives, used to define what movements a vehicle is to perform, and how.
4. Guidance: Describes the desired state of the vehicle, like heading and velocity.
5. Navigation: Describes the vehicles navigational state.
6. Sensor: Reported sensor readings.
7. Actuator: An interface to the vehicle hardware.

### 2.3.2 Neptus

Neptus is the user interface in the LSTS toolchain, used through the complete life-cycle of a mission. This begins with visually planing and then simulating the mission. During the mission, it allows for changes on the fly, as well as vehicle monitoring. When the mission is completed, the MRA (Mission Review and Analysis) tool can read the vehicles log, and either export it to a commonly readable format, or display visualizations facilitating further analysis. A more thorough description is given in [41].

Neptus is developed in Java, and compiled with the Apache Ant system. This makes it compatible with both Linux based operating systems, and with Windows<sup>14</sup>.

For operators, documentation is available at [42], while developers are directed to the GitHub repository with its wiki at [43].

#### The Console

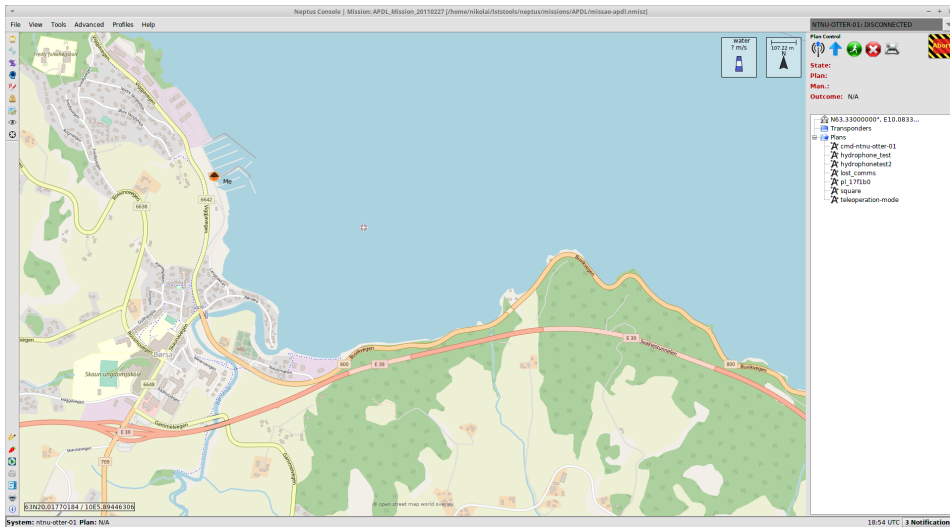
To address the needs of the different types of vehicles, the Neptus console is customizable through XML files that tells what plugins to load, with the filename ending in *.ncon*. Starting from scratch, the console is just a solid gray window, but after adding a map and some more plugins, it turns into something like figure 2.4.

#### Mission Review and Analysis

In the Mission Review and Analysis (MRA) interface, mission data can be inspected and analyzed. The file format used for this purpose is suffixed by *.lst*, often compressed with Gzip with suffix *.lsf.gz*.

---

<sup>14</sup>And potentially other operating systems like macOS or BSD



**Figure 2.4:** The console window of Neptune.

A screenshot of the MRA is provided in figure 2.5. In addition to the listing messages, graphing and other visualizations of messages are available.

### 2.3.3 DUNE

DUNE is the software running onboard the vehicle. It is written in C++, and is compatible with multiple operating systems, including Microsoft Windows and GNU/Linux systems. To facilitate the building process, CMake<sup>15</sup> is used. In addition to running onboard the vehicle, it can also be used to simulate the vehicle, where for most tasks the exact code can be used in both hardware and simulations.

DUNE is designed as a single process with multiple threads, where each thread contains a single logical operation, such as interacting with vehicle hardware, running controllers, navigation, maneuvers as well as supervisors and system monitors. All inter-thread communication is done through a bus where the tasks can publish or subscribe to IMC messages. Using standard IMC messages for all communication makes modularity possible. As an example of this, selecting which path controller to use is as simple as changing what task is activated in a running DUNE instance.

#### Configuration Files

Configuring what tasks will be running in an instance is done through a configuration file, containing information about what tasks are to be active, as well as what parameters to run them with. An example of this is given in figure 2.2, showing how the task written for the *Thelma Biotel TBR 700 RT* hydrophone is configured, as well as comments describing the

<sup>15</sup>Creates the makefile that is used when compiling DUNE. It makes cross-compiling and adding source files to the project easier. CMake homepage: <https://cmake.org/>

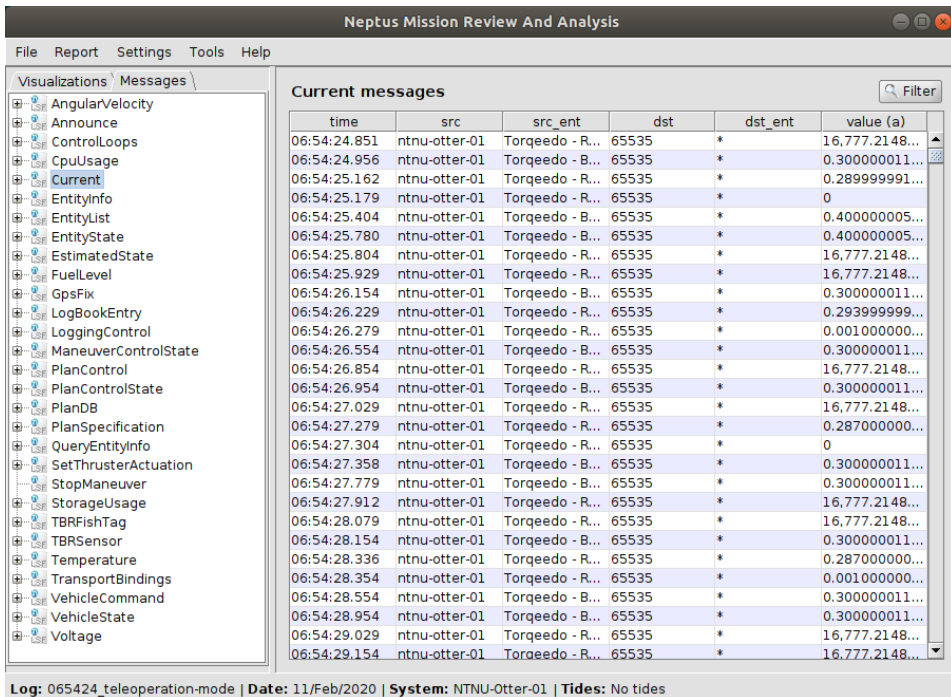


Figure 2.5: The Neptus MRA.

role of each line. The profiles gives the flexibility of using a single configuration file for a vehicle through many scenarios, such as simulation, hardware, developing using hardware-in-the-loop or other custom profiles, like the `StratoPi` profile. The configuration files can be exported to Neptus, which allows for them to be changed while a mission is in progress, handy for tuning controller parameters.

**Code 2.2:** Dune configuration file example.

```

1 [Sensors.TBR700RT]
2 # Common parameters for all task
3 Enabled           = Hardware, StratoPi # Profiles
4 Debug Level       = Spew # Level printed to console
5 Entity Label      = Hydrophone # Task instance name
6 # Task specific parameters
7 Serial Port - Device = /dev/ttyAMA0
8 Serial Port - Baud Rate = 115200

```

Line 3 of code listing 2.2 shows how profiles are used in configuration files. The catch-all statement `Always` makes a task active for all profiles, while specifying the profile names makes it only enable when DUNE is run with the parameter.

Configurations spanning multiple files is supported by the `include` and `require` statements.

## Tasks

The life cycle of a task is defined in a set of methods that it inherits from a base class, see [44] and [45]. There are two types of base classes: one for continually running tasks, and another for periodically running tasks. Both provide a common set of functions;

```

onUpdateParameters(), onEntityReservation(), onEntityResolution(),
onResourceAquisition(), onResourceInitialization(), onResourceRelease
(),
onActivation(), onDeactivation(), as well as onMain() in continually running
tasks, and task() running on periodically running tasks.

```

### 2.3.4 IMCProxy

IMC networks are able to automatically discover and connect to nodes through using broadcasted announce and discovery messages. When using IMC over UDP, these broadcasted messages are often not routed to other networks, and thus they are limited to a local network. In solutions where traffic across different networks is necessary, a proxy solution is necessary to bridge IMC networks, with bridging over the Internet being one example. The IMCProxy has been created for this purpose. Its based on a `WebSocket`<sup>16</sup> server/client architecture written in Java, where the server has a separate connection for each connected client. The proxy server receives all IMC messages from connected clients, and relay them to all other clients. An example is given in figure 2.6, showing how DUNE instances and a Neptus console can be connected across different local networks.

<sup>16</sup>WebSocket: Standardized in RFC 6455. TCP based, HTTP compatible protocol.

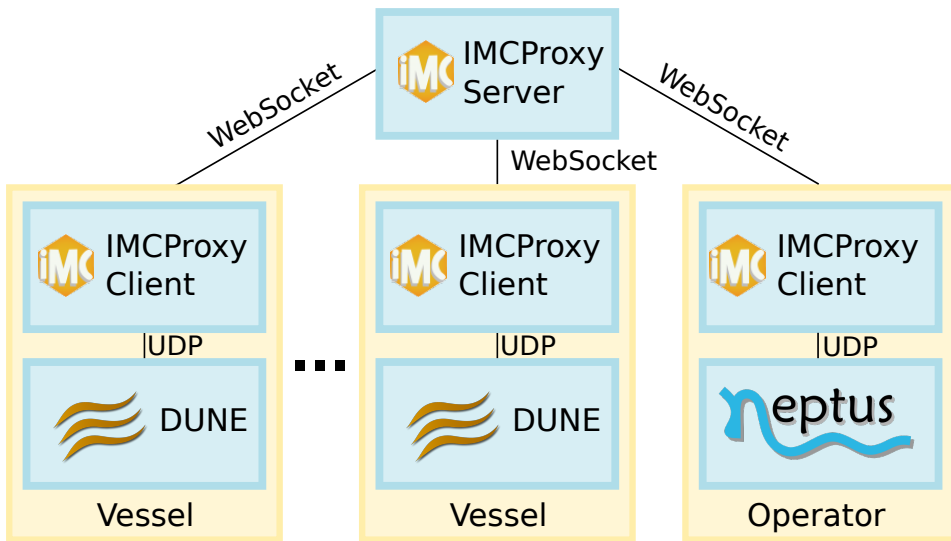


Figure 2.6: IMCProxy network overview.

### 2.3.5 GLUED

GLUED is a minimal GNU/Linux distribution targeted at embedded systems, developed as a part of the LSTS toolchain. It's not currently used by the NTNU Otters, but contains a cross-compiling tool that is used to compile DUNE for the ARM-processor in the RPI4.

## 2.4 Geoinformatics and Cartography

In this section, a selection of concepts from the field of geoinformatics and cartography that are used later on in this thesis are presented.

### 2.4.1 Geodetic Datums and Projections

A geodetic datum is a reference surface for the earth, on which points of reference are defined. Relative to these points, coordinates can be used to describe any position on the surface. Multiple reference systems have been created over time, with newer being created as better measurements become available. For small areas, using a plane surface is possible, but to increase precision over large areas, a surface more similar to the ellipsoidal shape of the earth is used [46].

#### Maps by Kartverket

Kartverket, the Norwegian mapping authorities (NMAP), often use Gauss-Krüger projections<sup>17</sup> and coordinate systems in its available datasets. These projects the earth surface to

<sup>17</sup>Also named Transverse Mercator Projection.



a cylindrical surface. This surface can then be folded to a two-dimensional plane, resulting in a Cartesian coordinate system. A consequence of using the cylindrical surface is that there is a large deviation for the areas farthest away from the middle meridian. To solve this problem, the earth surface is divided into multiple smaller zones [47].

The standard used by NMAP is called the Universal Transverse Mercator (UTM), and divides the surface into zones in latitude and longitude. In latitude, the zones are divided into 20 letters [c-x], spanning from 84° south to 80° north where x is the northern extreme. In longitude, there are 60 numbered zones where most are spanning at most 3° longitude on either side the zones central meridian. The only exceptions on the entire earth is over the areas of Norway, where zone 32V is extended to cover the entire southern Norway, as well as the zones 31X, 33X, 35X and 37X spanning areas around Svalbard [48]. The UTM projection achieves a worst case deviation of 4cm/100m or 400 parts per million (ppm) between the actual terrain measurements and the map [49]. The worst case accuracy is achieved in the areas farthest away from the central meridian.

Inside a zone, the coordinates are given as Cartesian coordinates with (easting, northing) pairs, with the origin set in the intersection between the numbered zone central meridian and equator. Both are measures of distance, contrary to the angles used in latitudes and longitudes.

The reference system used by NMAP to position the zones is called EUREF89/ETRS89. According to NMAP, this is a regional reference frame fixed to the stable parts of the Eurasian tectonic plate, with 11000 points of reference in Norway [50]. This comes with the benefit that the coordinates are fixed to the surface, and not drifting.

## **WGS84**

The LSTS software used by the NTNU Fish Otter uses the World Geodetic System (WGS84) with coordinates for latitude, longitude and height. WGS84 is widely adopted, due to its use in geo-positioning devices. It is also the horizontal datum used in S-57 electronic navigational chart [13]. In contrast with ETRS89, WGS84 is not fixed to the earths surface. Due to changes on the earths surface such as continental drift, there is an error of  $\pm 40$  cm[51].

## **EPSG Identifiers**

EPSG is an abbreviation for the European Petroleum Survey Group which has released, and maintains a geodetic parameter database with standard codes. EPSG codes are given for coordinate systems, datums, spheroids, units and such. In this thesis, three identifiers will be used: EPSG:4326 is the identifier for WGS84 datum, EPSG:3857 is a projected coordinate system often used in online maps and EPSG:25833 is the EUREF89 UTM zone 33 coordinate system.

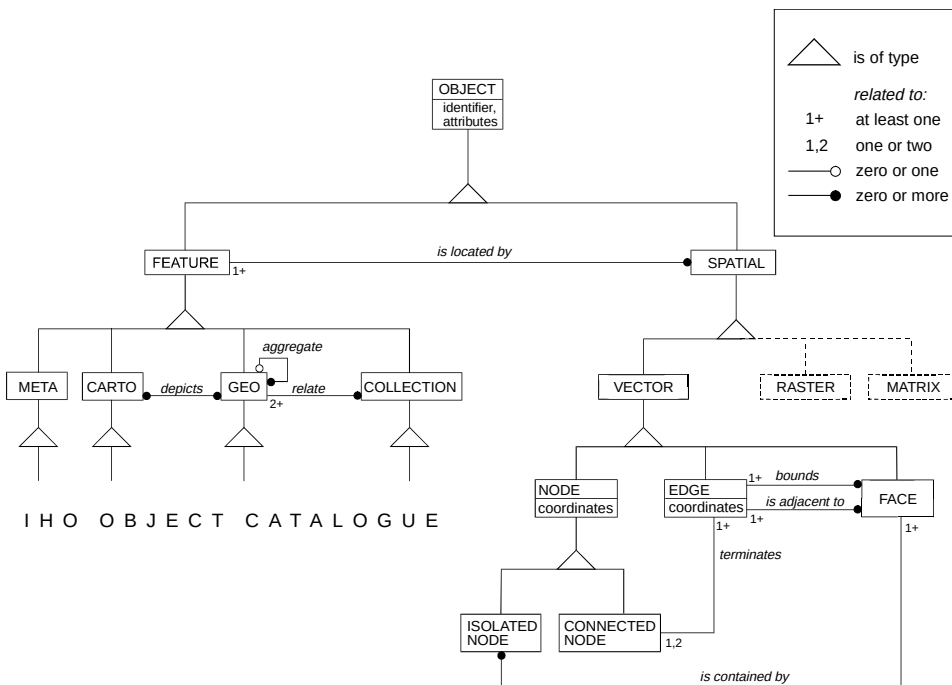
### **2.4.2 The S-57 ENC format**

The International Hydrographic Organization, which is *an intergovernmental organization that works to ensure all the world's seas, oceans and navigable waters are sur-*

veyed and charted<sup>18</sup> has through its Hydrographic Services and Standards Programme (HSSC) provided a series of technical standards for managing electronic navigational charts (ENC)[53]. The S-57 standard defines the IHO Transfer Standard for Digital Hydrographic Data, which is *applied by all hydrographic offices world-wide for the production of corresponding electronic navigational charts for their waters*[54].

The data is stored in objects, as described in figure 2.7. As is seen, spatial information is stored separated from the descriptive features, with relations connecting the feature with spatial objects used to describe where the feature is valid. Combinations of the vector object types is used to describe three two-dimensional geometric primitives: Points, lines or areas. When a third dimension needs to be described, this is stored as a feature.

The naming convention for objects are acronyms. An example of this is DEPARE, which is a feature object that contains depth bounds for areas. The depth range limits are defined by the attributes DRVAL1 and DRVAL2, where DRVAL1 is the shallowest depth found in the area, and DRVAL2 is the deepest water found in the area. A complete catalogue of the objects is given in [55], while a small subset is included in table A.1 of this text.



**Figure 2.7:** IHO S-57 theoretical data model (Source: Figure 1.1 and 2.1 from [13] combined).

<sup>18</sup>Cited from [52]

### The files of the S-57 standard

S-57 files are suffixed with *.xxx*, where *xxx* is three digits with values from 0 to 9. The *.000* files are the initial file for a section of land, while *.001* and up stores updates to the initial *.000* ENC. The different sections of the earth are stored in a folder structure, with a separate folder for each section. On the upper level, there is a catalogue file, which has the suffix *.030*.

The objects stored in the files are encapsulated in accordance with the ISO 8211 specification to allow transferring between computer systems.

### The Norwegian Mapping Authorities Organization of the S-57 data

Upon request, NMAP provided S-57 ENCs for Trøndelag (Norway) for this thesis. The received data is placed in a root folder named ENC\_ROOT, which contains the catalogue file, a readme file and a folder named NO. This structure is part of the S-57 standard [13], where the NO is a code for the mapping authority releasing the ENC. Opening this folder reveals the actual data of the ENC, as multiple folders giving information about different areas. NMAP has named these folders in a format like NO[p][s][xx][yy], which is further described in table 2.2.

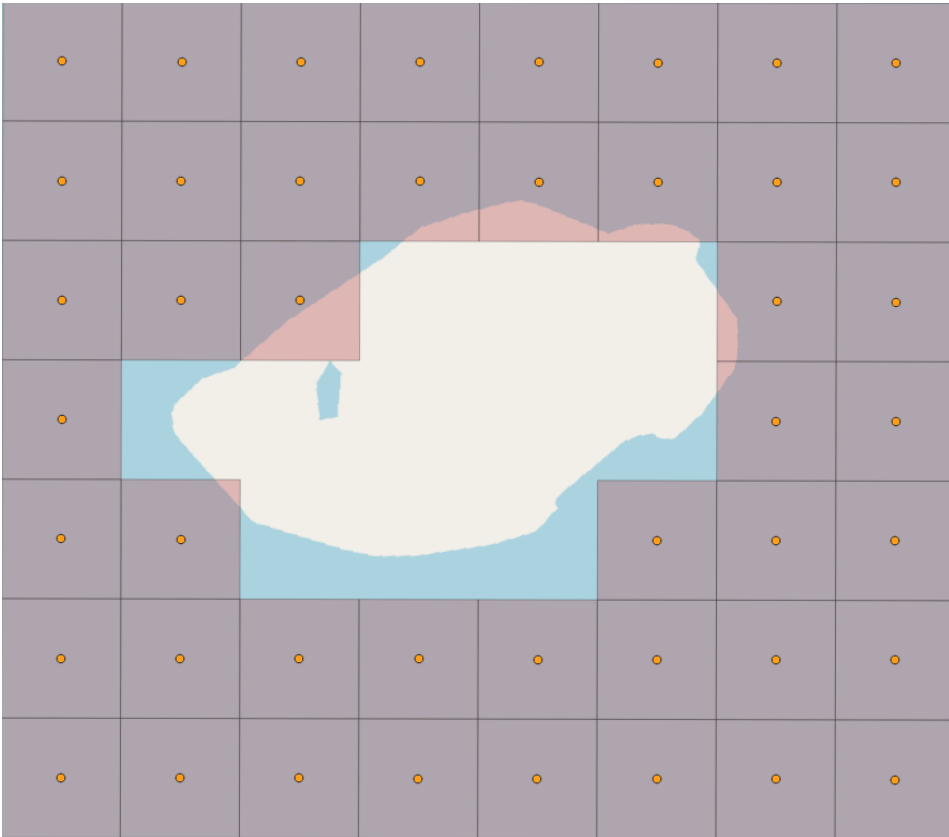
Country Code	Purpose	Size	Position	
[cc]	[p]	[s]	[xx]	[yy]
Two capitalized letters given according to IHO Codes for Producing Agencies	Purpose enumeration, formatted like the DSID_INTU (Intended Usage) S-57 object. Possible values:  1 = Overview 2 = general 3 = coastal 4 = approach 5 = harbour 6 = berthing	Single capitalized letter where 'A' is the largest area, and 'Z' is smallest	Specifies the latitude and longitude position of the south west corner of the ENC. The cell representation has its origin at (55° N, 0° E.), and increments in a 0.5° resolution.	

**Table 2.2:** S-57 naming convention used by Kartverket.

### 2.4.3 Grid-based Decomposition

In grid-based decomposition, a continuous surface is discretized to a grid of points. The information about a point is valid in an area around the point, which for some cases may be overlapping. There are many possible shapes that can be used, but squares will be used

in this thesis. To save storage space, only the points over the ocean is stored in the later works of this thesis. An example is shown in figure 2.8.



**Figure 2.8:** Grid-based decomposition with squares.

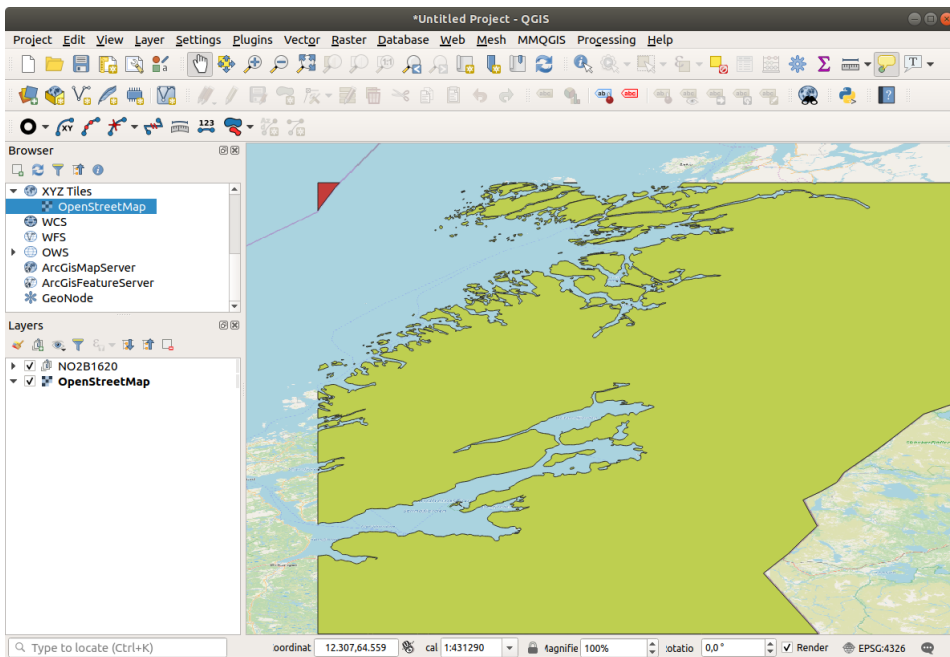
## 2.4.4 Visualizing Spatial Data in QGIS

When working with spatial information, visualizing is a helpful aid in understanding the results. For this purpose, the free and open source Geographic Information System (GIS) QGIS has been used throughout this thesis to create figures with maps that describe the results. As a base map, the free OpenStreetMap is used. Because the base map uses the EPSG:3857 projection of the WGS84 datum, and some of the data used in this thesis is based on ETRS89 referenced features, some small deviations (below 1m) between the base map and the drawn points is expected<sup>19</sup>.

A screenshot of a S-57 ENC opened in QGIS is shown in figure 2.9.

---

<sup>19</sup>A datum-transformation would mitigate this.



**Figure 2.9:** QGIS main window showing a S-57 ENC on top of the OpenStreetMap.

### 2.4.5 Manipulating Spatial Data in FME Desktop

The FME software suite by Safe Software is used for data integration, with support for spatial data manipulation. FME is an acronym for *Feature Manipulation Engine*, where a feature describes a spatial object.

The part of the FME suite used for data extraction/transformation is named the FME Desktop Workbench. The interface works by setting up a workspace with reader, writer and transformer nodes that each have inputs and outputs that have connections routed in between them. Examples of workbenches can be seen in figure B.1, B.2, B.3 and B.4. A description the nodes relevant for this thesis is given in table 2.3.

#### Batch Deployment

The FME workspaces can be applied to multiple datasets at a time by using batch deployment. This is practical, because extracting data from multiple S-57 files can be done all at once. When selecting Run->Batch Deploy, whole folders can be scoured for relevant files. As an example, `/dataset location/ENC_ROOT/NO/NO4**/**/*.*.000` is used in this thesis. This selects all S-57 base files in the selected folders. The `NO4**` restricts the deployment to only folders starting with `NO4`.

Node Name	Purpose	Purpose in this thesis
Reader	The data source in a workbench. Numerous formats are available.	Read S-57 and CSV files.
Writer	The data sink in a workbench. Numerous formats are available.	Write SQLite3 files.
AngleConverter	Converts a field with angles to another format for angles.	Used for converting degrees to radians for use with the LSTS toolchain.
AttributeFilter	Filter objects based on attributes.	Used in figure B.4 to remove areas with no DRVAL1, which means removing areas that are not in DEPARE.
AttributeRenamer	Gives a feature attribute a new name.	Give coordinates the names <i>lat</i> and <i>lon</i> .
CoordinateExtractor	Adds spatial coordinates as attributes to a feature.	Makes it possible to store spatial data and feature data together in the database.
Densifier	Adds vertices by interpolating at given intervals.	Used in figure B.3 to ensure a lower bound on distance between vertices.
Reprojector	Changes projection from one to another.	EUREF89 UTM 33 to WGS84
SpatialRelator	Performs a join operation on spatial relationships.	Used in figure B.4 to add DRVAL1 and DRVAL2 to the grid.
VertexExtractor	Extracts the vertices of an object.	Make a table of the vertices defining DEPARE in figure B.3.
2DGridAccumulator	Creates a square two-dimensional grid in the size of a given object. Resolution is given as parameters.	Used in figure B.4 to create a two-dimensional grid over DEPARE area.

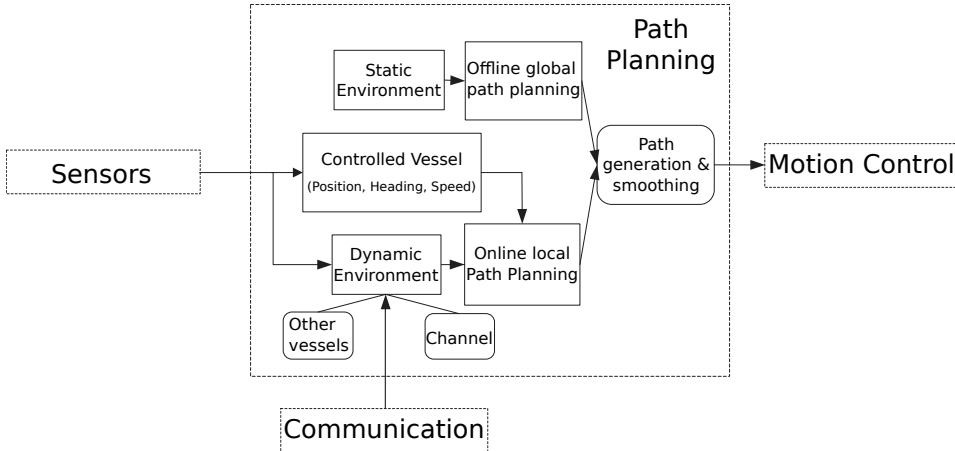
**Table 2.3:** Description of the FME Workbench nodes used in this thesis.

## 2.5 Path Planning for Marine Vessels

An important sub-system of an ASV, is the path planning implemented to provide the vessel with the ability to make decisions autonomously. The task of a path planner, is to find a safe and valid sequence of states for bringing the vessel from a state towards its desired state. An additional goal is to find an acceptable balance between optimizing

energy consumption, path length and processing time.

The general path planning framework for intelligent marine vessels is given in figure 2.10, as drawn out in [6].



**Figure 2.10:** The basic path planning framework of an intelligent marine surface vessel (Based on figure 1 from [6]).

As is laid out in the figure, offline path planning tries to find a global solution based on only information about static parts of an environment. Based on this global plan, it's left to the local path planner to manage dynamic elements such as path deviations caused by currents, waves or wind or adapting to unexpected hazards like other moving objects.

For the NTNU Fish Otter, the environmental sensors part of the system is limited to the Hemisphere GNSS V104s, the communication is with the operator and the other vessels in the system, and the motion control is the DUNE pathController task in combination with the CourseAndSpeed controller.

In this thesis, a system for offline global path planning is developed, so some related topics are given in the next sections.

### 2.5.1 Complexity Measures for Algorithms

To give a measure of the efficiency of an algorithm, two methods will be used: Benchmarks and the  $O()$  asymptotic analysis, both which are described in [2]. The concept behind benchmarks is simple, measure an aspect of the algorithm by running it, like execution time or states searched. This is ultimately what matters, but has the drawback that the results are influenced by the hardware used to achieve them.

To be able to say something about an algorithm's complexity no matter what system it is using, an asymptotic analysis is performed, describing how many operations have to be performed, or how many elements have to be stored. It is also useful for analyzing how the size of the search domain is reflected in resource usage. To make this relationship clear, the  $O()$ -measure abstracts over constant factors, so if say an algorithm runs for a maximum

of  $n + 100$  operations, the measure for it is  $O(n)$ , where  $n$  is the number of nodes in the searched graph/tree.

## 2.5.2 The A\* Graph Search Algorithm

When searching in a graph in order to find a path from one state to another, not having to consider all states in each operation is very much a desired property in an algorithm. The best-first search algorithm variation called A\* achieves this by calculating the total estimated solution cost for each node  $f(n) = g(n) + h(n)$ , where  $g(n)$  is the calculated path cost from start to  $n$ , and  $h(n)$  is a heuristic function estimating the path cost from node  $n$  to the goal.

By expanding the node which has lowest value  $f(n)$  value at each iteration, the nodes needed to be searched is often dramatically decreased[2] because the iterations finish when a goal condition is met. The pseudocode for the algorithm is given in code listing 2.3, while the pseudocode function for expanding a node is given in 2.4.

**Code 2.3:** Pseudocode for the graph searching A\*-algorithm (Source: [2]).

```
1 function A-STAR-SEARCH(problem) returns a solution, or failure
2   node ← a node with STATE=problem.INITIAL-STATE, PATH-COST = 0
3   frontier ← a priority queue ordered by PATH-COST, with node as
4     the only element
5   explored ← an empty set
6   loop do
7     if EMPTY?(frontier) then return failure
8     node ← POP( frontier ) /* chooses the lowest-cost node in
9       frontier */
10    if problem.GOAL-TEST (node.STATE) then return SOLUTION(node)
11    add node.STATE to explored
12    for each action in problem.ACTIONS(node.STATE) do
13      child ← CHILD-NODE(problem, node, action)
14      if child.STATE is not in explored or frontier then
15        frontier ← INSERT(child, frontier)
16      else if child.STATE is in frontier with higher PATH-COST
17        then
18          replace that frontier node with child
```

**Code 2.4:** Pseudocode for the function expanding a child (Source: [2]).

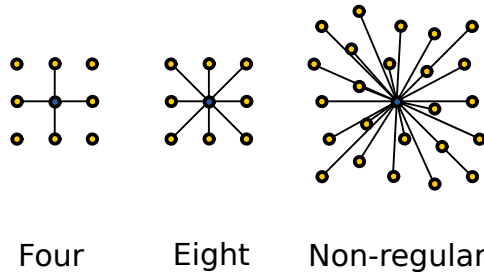
```
1 function CHILD-NODE(problem, parent , action) /*returns a node*/
2   return a node with
3     STATE ← problem.RESULT(parent.STATE, action),
4     PARENT ← parent, ACTION ← action,
5     PATH-COST ← parent.PATH-COST + problem.STEP-COST(parent.
6     STATE ,action) + HEURISTIC-COST(STATE)
```



## Heuristics

The heuristic function in the A\* algorithm is responsible for estimating the cost from one node to the end node. Multiple heuristics have been developed that are optimized for different graphs. A small selection, based on [56] is mentioned below, with examples of grid movement in figure 2.11:

- **Manhattan distance:** The distance between two points measured along axes at right angles. Suited for grids with four directions of movement.
- **Diagonal distance:** The diagonal distance measure is used when there are eight directions of movement allowed. The cost of each these directions is calculated by the formula:  $h(node, end) = straightCost \cdot \max(|node.x - end.x|, |node.y - end.y|) + (diagonalCost - straightCost) \cdot \min(|node.x - end.x|, |node.y - end.y|)$
- **Euclidean distance:** The distance of a straight line between two points. Used on non-regular search graphs. Calculated with the formula:  $h(node, end) = weight * \sqrt{|node.x - end.x|^2 + |node.y - end.y|^2}$ . The formula in some cases need to be modified, such as when the distance is on the earths surface.



**Figure 2.11:** Grid movement examples for searching algorithm.

In addition to the choice of heuristic, the scaling weight used in the heuristic plays an important part in optimizing the A\* algorithm. The scale of the heuristic function  $h()$  should be equal to the scale of the path cost function  $g()$ .

## Optimality Condition

The paths produced by the A\* algorithm on graphs are optimal under the condition that the heuristic path cost estimator is consistent. In [2], this is defined as:

$$h(n) \leq c(n, a, n') + h(n') \quad (2.6)$$

where  $n$  is the node representing the current state,  $h(n)$  is the heuristics function,  $c(n, a, n')$  is the step cost from taking action  $a$  resulting in changing current state from  $n$  to the next state node  $n'$ .

### **Balancing Performance at the Cost Of Optimality**

By slight increases in the heuristic scaling weights, some of the accuracy of the algorithm can be sacrificed for performance gains. This results in a more greedy approach, where fewer nodes have to be evaluated.

It may also be a benefit to slightly increase weighting to handle tie breaking when multiple paths of same length is expected. This leads to only one of the paths being expanded, leading to a solution being returned faster, possibly at the expense of optimality[56].

## **2.6 Online Services**

This section gives background information about software services used for distributing information from the Otter server.

### **2.6.1 Wiki**

To document the NTNU Fish Otter project, a wiki has been created. According to [57], a *Wiki is a simple web authoring tool. A wiki page is edited using a plain text format, which can be automatically translated to formatted HTML.*

Often, wikis are editable by all, while the Otter wiki only allows personnel involved with the project to edit. Most of the wiki is available to the public, while some sections have been locked off.

The wiki can be found by accessing <http://otter.itk.ntnu.no/doku.php> on the web, and a screenshot of the front page is shown in Figure 2.12.

#### **DokuWiki**

The software used to run the Otter wiki is called DokuWiki, and allows for easy creation and editing of wiki-pages through an online content management system. The layout and look of the wiki is the only setting changed from a default installation by using a layout template named "Bootstrap", adding a sidebar and putting the NTNU logo in the header.

The DokuWiki stores content in files, rather than in a database, making it easy to set up, and easy to backup (by just copying the data folder).

#### **Apache Webserver**

To serve the php<sup>20</sup> scripts used in DokuWiki on the web, the Apache HTTP server has been configured. It's released as free and open-source software, and is currently licensed under the Apache 2.0 License.

### **2.6.2 Grafana**

Grafana is an open source analytics and monitoring solution that provides a web based interface that lets users create multiple dashboards containing data visualizations<sup>21</sup>. The

---

<sup>20</sup>PHP: Hypertext Preprocessor

<sup>21</sup>Live demo available at: <https://play.grafana.org>.

NTNU Fish Otter Project Documentation

Search

Log In

Trace / start
[ start ]

**Homepage**

**Introduction**

**Academic Literature**

**The Otter**

- Hardware Architecture
- Software Architecture
- Network Architecture

**Sensors**

- Navigation Sensors

**Power budget**

- Mission Logs

**The Server**


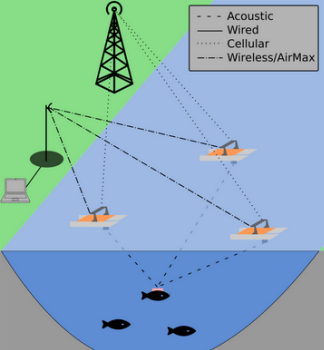
- Online Data Visualizing
- VPN
- Documentation

edit

## The NTNU Fish Otter Project - A fish tracking system

The NTNU Fish Otter is a small unmanned catamaran propelled by two electrical fixed thrusters. The hull is based on the [Maritime Robotics Otter](#), along with an interface card for thrusters, batteries and power management. A custom hardware and software design has been integrated for the vessel in order to allow full system control based on the [LSTS toolchain](#).


The goal for the vehicle is to provide a platform that can autonomously carry a hydrophone to wherever it is needed and collaborate with other Otters performing multilateration to position the tag. On the way to reach this goal, the Otter project will have to solve multiple problems like collision avoidance, multi-vehicle autonomous coordination, covering search algorithms etc. all while operating in an unpredictable ocean environment.


This wiki is maintained by Nikolai Lauvås, Master student at the Department of Engineering Cybernetics, NTNU.

start.txt Last modified: 2020/07/09 18:13 by nikolai

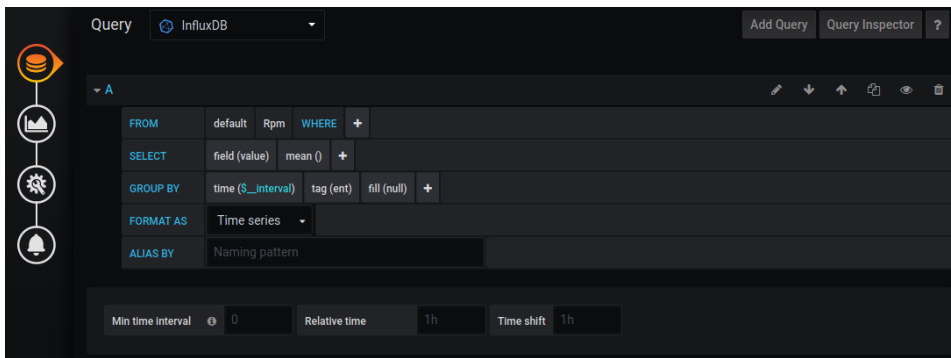
NTNU Fish Otter Project Documentation



Except where otherwise noted, content on this wiki is licensed under the following license:  
CC Attribution-Share Alike 4.0 International



**Figure 2.12:** A screenshot of the DokuWiki made for the Otter, accessed 09/07/2020.



**Figure 2.13:** A screenshot of the Grafana querying interface.

time range, visualizations and appearance is entirely up to the user, and is configured in the web interface.[58]

Grafana supports multiple data sources, such as InfluxDB, Prometheus, MySQL and PostgreSQL, which are queried through an online interface, as shown in figure 2.13. The result can then be presented in visualizations such as graphs, maps, gauges, tables etc. As the platform is open source, custom visualizations can also be implemented. Exporting the queried data is available in CSV format for further analysis.

Once the dashboards are configured, users can browse through the data without needing any knowledge of the underlying database or querying happening in the background. Grafana can therefore also be described as an online visual querying solution.

Examples of dashboards are given in figure 7.4 and figure 7.5.

## 2.7 Database Systems

To store large large amounts of data, like the results of grid-based decomposition, in a way that balances searching performance and overhead storage, databases are used. Through a Database Management System (DBMS), a database is created by defining its structure, constructed by filling it with data, and stored in some fitting data-structure. After creation, an application program can access the DBMS to manipulate the data, or run queries to receive the constructed data. An important aspect is that the DBMS allows multiple-users to access the database simultaneously in a way that maintains the database in an atomic state [59].

### 2.7.1 Relational Databases

In relational databases, the content is modelled as tables having attributes stored in tuples. Each row of the table is a tuple, and contains related data. The structure description of such a database is called a schema, and also includes the datatype of the attributes.

	<b>Attribute1</b>	<b>Attribute2</b>	<b>Attribute3</b>
<b>Tuple1</b>	Value11	Value12	Value13
<b>Tuple2</b>	Value21	Value22	Value23

**Table 2.4:** Relational database schema example with values.

## 2.7.2 Indexing and Keys

For each database, there is a key that uniquely identifies every tuple [59]. To be able to efficiently search a database on keys, auxiliary data structures called indexes are used. How these are implemented varies between database systems, but hash-tables and some binary-heap implementations are widely used. For some types of data, like spatial data or time-varying data, specialized indexes exist. Additional indexes can also be created to increase searching performance on the other attributes of the table.

## 2.7.3 SQL

The main interface to access the data stored in a database is through writing queries that precisely describe what portion of the data is wanted. For this purpose, multiple languages have been invented. A desirable characteristic of such a language is that it provides independence from the underlying data structure.

The structured querying language (SQL) is a standard language for relational DBMSs that provides a high-level declarative way of querying [59].

## 2.7.4 SQLite

SQLite is a light weight DBMS that stores its database in a single self-contained file. This makes it widely used for storing data in applications<sup>22</sup>, and also as the base for a lot of file formats[60]. Unlike many other DMBSes, no server is required to use SQLite, everything is done from a single library written in the C programming language.

### SQLite in the LSTS toolchain

Both DUNE and Neptus use SQLite to store data, like the vehicle plans stored in DUNE. This means that the library is already integrated in both software packages, making it a natural choice when developing extensions for the LSTS toolchain.

## 2.7.5 Time-Series Database Management Systems

A time-series database management system (TSDBMS), is a DBMS that is optimized for time-stamped or time series data [61]. The primary key of a TSDBMS is always a measure of time, with performance optimization for features such as time-aggregation, down-sampling and data monitoring.

<sup>22</sup>High-profile examples of users: Apple, Microsoft, Google, etc. See <https://www.sqlite.org/famous.html> (accessed 25/06/2020).

## 2.7.6 InfluxDB

InfluxDB is an open-source time-series DBMS, and has been optimized for storing and serving time series through associative pairs between time(s) and value(s). InfluxDB further divides the values into tags and fields. The difference between them, is that tags are automatically indexed, while fields are not being indexed at all. Choosing between them is a part of the database design, and is done by considering which queries will be performed most frequently.

In this thesis, it will be used for storing converted IMC messages. Examples of how the fields and tags are implemented for this, is having the message source entity and vehicle as tags, while temperature, position<sup>23</sup>, current and voltage is examples of fields. This is based on that most often, the the queries will be restricted to a single entity.

Accessing InfluxDB can be done through a SQL based language called InfluxQL, or through a HTTP based API. The documentation for both is available at Influxdata's web pages at [62] and [63]. Because parts of the HTTP API is used in section 7.2.2, a short introduction is given below:

The HTTP POST request method for ingesting data to the database, has the URL format:

```
1 http://[server]:[port]/write?db=[dbname]&u=[user]&p=[password]&precision=[time_precision]
```

The [dbname] variable is the only required variable, while the others are optional.

The contents of the HTTP POST request is then added according to the Influxdata line protocol [64], with the syntax:

```
1 <measurement>[,<tag_key>=<tag_value>[,<tag_key>=<tag_value>]] <field_key>=<field_value>[,<field_key>=<field_value>] [<timestamp>]
```

The <measurement> is the InfluxDB equivalent to tables in general DBMSes.

---

<sup>23</sup>There is experimental support for geo-temporal data, but as this was still in beta testing at the time of writing, it is not used

# Design of a Robotic Fish Tracking Vehicle System

In this chapter, the concept design of a Robotic Fish Tracking Vehicle System called the NTNU Fish Otter System is presented. Along with the concept description in chapter 3.1, the current design is described, with hardware in section 3.2 and software in section 3.3.

## 3.1 The NTNU Fish Otter System for Fish Tracking

The motivation behind the NTNU Otter Project is the research presented in [21] and [22], where the concept of having multiple surface vessels with acoustic receivers track an acoustic transmitter was showed to be viable. The prior masters thesis by Ekanger [65] and Kristiansen [66] are directly relevant to the fish tracking aspects of the project, while Steindal in [67] creates a model for the Maritime Robotics Otter along with a bumpless linear-quadratic-controller that changes linearizations based on velocity.

In Ekanger [65], the initial design of the NTNU Fish Otter project is laid out. Since this was written, major hardware changes has been done to the ASV, like the decision to have the hydrophone fixed instead of mounting it to a flexible cable.

On a different note, Kristiansen [66] presents strategies for searching after acoustic transmitters using an unmanned surface vehicle, with the NTNU Fish Otter as the intended user.

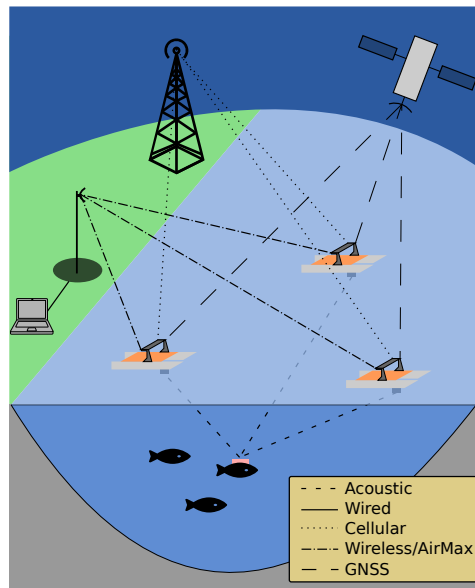
### 3.1.1 Design Problem Definition

To aid in the design process of the NTNU Fish Otter Project, the scenario presented in table 1.1 has been used as a base. A further definition of the conditions surrounding a mission based on the scenario is given in table 3.1. These should be considered as initial values, and may be subject to change as more experience with the system is gained.

A visual representation of the most important phase of the scenario, the fish tracking, is shown in figure 3.1. Here, the three Fish Otters have gained contact with a tagged fish, and is using multilateration to estimate its position. Based on the estimate, decisions are then made regarding the next position of the formation. The formation also has to try to minimize GDOP, while at the same time avoiding surrounding hazards and reducing the thruster usage. Reducing motor usage is beneficial because it conserves the stored energy and reduces noise in the medium (water). Increased noise makes it harder for the acoustic receiver to discern tag transmissions and may also influence the fish behavior.

The communication between the operator and the vessels, can, as shown in figure 3.1 go through two channels: the public cellular network, and the private Ubiquiti AirMax network. Either one, or both at once is to be used, with the expectation that bandwidth and increased network control will be benefits of the AirMax channel, while range is the benefit of the cellular communications.

Positioning of the vessels is based on GNSS systems, which also is a multilateration system. The GNSS can be used as an inverted analogy for the fish tracking performed by the Otters. In the GNSS, the satellites transmit signals, with the Otters receiving them. In the fish tracking scenario, the Otters have the same role as the satellites, but instead receives the message sent from the acoustic tag for surveillance purposes.



**Figure 3.1:** Fish Tracking Scenario (The additional satellites needed in a GNSS is omitted for simplicity).

### 3.1.2 System Requirements for the NTNU Otter Project

Based on the design problem definition given in section 3.1.1, concrete requirements have to be developed, along with ways of fulfilling them. This list has been created based



<b>Description</b>	<b>Value</b>
Maximum operational workspace	10km · 10km, resulting in an area of 100Km <sup>2</sup> .
Minimum navigable depth	1m
Minimum depth limit during autonomous operation	5m
Maximum wave height	0.5m
Maximum sea current	Not defined, less than maximum operational speed.
Maximum wind	Not defined

**Table 3.1:** Operational Limits of the Fish Tracking System.

on the experience gained in the TTK4550 specialization project [1], as well as through discussions with the supervisor Jo Arve Alfredsen.

### **Creating an Equipment List**

To ensure that all necessary equipment is transported to the deployment area and that no equipment is left behind afterwards, a list should be specified. This is not a technical problem, but rather an organizational part of the project.

### **Specify and Procure Necessary Hardware and Accessories**

The specifying and procurement of hardware has in large part been completed, with the exception of cellular equipment for both the operator and vessels.

Spare parts and tools may also be beneficial on missions, to solve problems where a component may break or need to be modified. For this purpose the tools needed to perform rudimentary tasks on the hardware has to be brought. From sea trials in [1], a set of screwdrivers and socket wrenches for opening and closing the control box was identified as needed. In addition to this, a rope and a boat hook simplified deploying the vessel.

### **Deployment Procedures**

In aviation, checklists are used to ensure that all systems are working as intended before the ensuing flight. A similar concept should be employed before deployment on missions for the Otters. Important aspects to check would be communication with the vessel, being able to its thrusters and ensuring that vessel position is obtained. Further checklist items should be added as more experience is gained.

### **Hardware Integration**

All the vessel hardware needs be integrated to the DUNE system, and external components need to be configured. This was in large part completed in [1], with the remaining cellular communications system and hydrophone synchronization being subjects treated in this thesis.

### **Guidance, Navigation and Control**

Guidance, Navigation and Control is in large part handled by DUNE, but the parameters of the controller functions need to be tuned according to the vessels behavior. A kinematic model with known constants fitting the vessel should be developed, which can be used in finding initial controller parameters.

### **Communication**

Communication between vessels and the operator has to be available, at least in the vicinity of the deployment area. This is to be accomplished by a locally controlled Ubiquiti AirMax 5GHz network and through the public cellular infrastructure. Both these subjects are treated in chapter 7.

### **Anti-Collision**

A system shall be developed that monitors the vessels locations, and prevents collisions. As the Otter has no external sensors to detect other vessels, accessing an online AIS data source is desired. This subject has not been explored further in this thesis.

### **Anti-Grounding and Hazard Avoidance**

A system for avoiding entering areas deemed to shallow shall be implemented. This system shall also be able to verify the depths of paths. As the Otter has sensors for detecting external hazards and depths, this system will be based on a priori spatial information. This is the subject of chapter 4 and section 5.3.

### **Formation Control**

To minimize GDOP when tracking a fish, a formation controller shall be implemented. This has not been explored in this thesis, but a possible design is presented in [21]. A non-functional requirement is that the acoustic disturbance should be minimized, to avoid affecting the fish. This is accomplished by minimizing thruster usage, and locating the vessels at a reasonable distance from the estimated tag position.

To make traveling from the deployment area to the search area simpler, a formation control for this purpose may also be implemented. An option would be the application developed for the LSTS toolchain in [68].

### **Searching Strategy**

For the phase when the vessels are trying to make initial contact with a tagged animal, a search strategy is needed. [66] explores multiple strategies that can be used for this purpose. Other options include using a multi-robot coverage path planner algorithm. The algorithm introduced in [69] accomplishes this goal, and has the benefit of (under reasonable restrictions on the configuration space), returning the paths with lowest distance possible.

#### **Mission Coordination Task**

When the vessel is performing autonomous fish tracking, a DUNE task that changes operation mode according to the scenario phases must be implemented. This topic has not been further explored in this thesis.

#### **Hydrophone Integration**

An integration of a hydrophone from Thelma Biotel shall be made, including time synchronizing and clock disciplining with millisecond precision. An initial integration of the TBR700RT was made in [1], but as shown in section 6.1, did not meet the synchronization requirement. The work described in section 6.1.2 aims to improve on this, but has yet to be verified.

#### **Single Vessel Path Planner**

A single vessel path planner shall be implemented that provides a safe path. The implementation should balance energy consumption, speed and algorithm run time in a manner suitable for the vessel. Initial work is provided in section 5.4, along with an extended discussion and assessment of alternatives to the implementation.

#### **Optional: Single Vessel Fish Tracking**

When only a single vessel has made contact with the acoustic tag, it would be beneficial if it could start the tracking immediately. An implementation based on the [19] is proposed as a solution to accomplish this goal.

#### **Estimator for Fish Position**

When the Fish Otters have made contact with the fish and assembled in formation, the position of the fish has to be estimated. Methods such as the eXogenous Kalman filter presented in [22] can be used as the starting point for this.

#### **Low Noise Listening Maneuver**

Another scenario for the Fish Otter is that they could be used as "mobile buoys", keeping a position with as little noise polluting thruster action as possible. Experiences from the project work showed that thruster usage had a severe impact on the measured signal-to-noise ratio.

#### **Battery Monitoring**

Monitoring of the available energy stored in the battery shall be made available to the vessel and its operator. This was implemented in the specialization project [1].

When the battery charge level sinks to below a set threshold, a planned action should be taken, such as executing a pre determined plan and alerting the operator. This is left for future works.

## Manual control

In situations such as berthing and deploying the vessel, operations with narrow safety margins is expected. For these cases, the vessel should be able to be controlled manually. This functionality is already implemented in the LSTS toolchain, and has been verified in sea-trials in the specialization project [1].

## 3.2 Hardware Design

### 3.2.1 The Fish Otter ASV

The NTNU fish Otter ASV, as shown in figure 3.2, is based on the Maritime Robotics(MR) Otter USV<sup>1</sup>, but with communication and control systems being designed and developed at NTNU. It's constructed as a catamaran multihull, with two pontoons spanning 200cm in length, combined with a structure in between that carries control, communication and payload hardware. At the stern of each pontoon, there is a fixed thruster to actuate the vessel, providing steering by differential thrust, also called skid-steering. A useful feature of the NTNU Otters that it has inherited from the MR Otter, is that it's easy to both assemble and disassemble into part weighting less than 20Kg. This makes it very portable, and also ensures that it fits into most normal cargo vans. It also makes it manageable for a single person to deploy the vessel alone, even from harbors with no slipway or shiplift, a simple floating docks or some fitting terrain will suffice.

The reasoning behind calling the NTNU Otter an autonomous surface vehicle, if oounded on the greater degree of independence from a operator/autonomy being envisioned.

An overview of the components of the NTNU Fish Otter ASV is given in table 3.2, with the control box further detailed in table 3.3.

## Navigational Sensors

For positioning, the Otter relies upon a single Hemisphere GNSS V104s. By using its dual integrated GPS antennas, both GPS positioning and compass functions are achieved. It also supports Space Based Augmentation Systems (SBAS) in addition to a single axis gyro and tilt sensors for x- and y-axis. This results in an positional accuracy of better than 1.0 m available 95% of the time<sup>2</sup>. The v104s is interfaced through two full-duplex RS-232, out off which the Fish Otter only uses one. An other relevant feature is the dedicated PPS<sup>3</sup> output that can be used for time synchronization.

The RS-232 protocol of the v104s adheres to the NMEA<sup>4</sup> 0183 standard, but is extended by some Hemisphere GNSS specific messages.

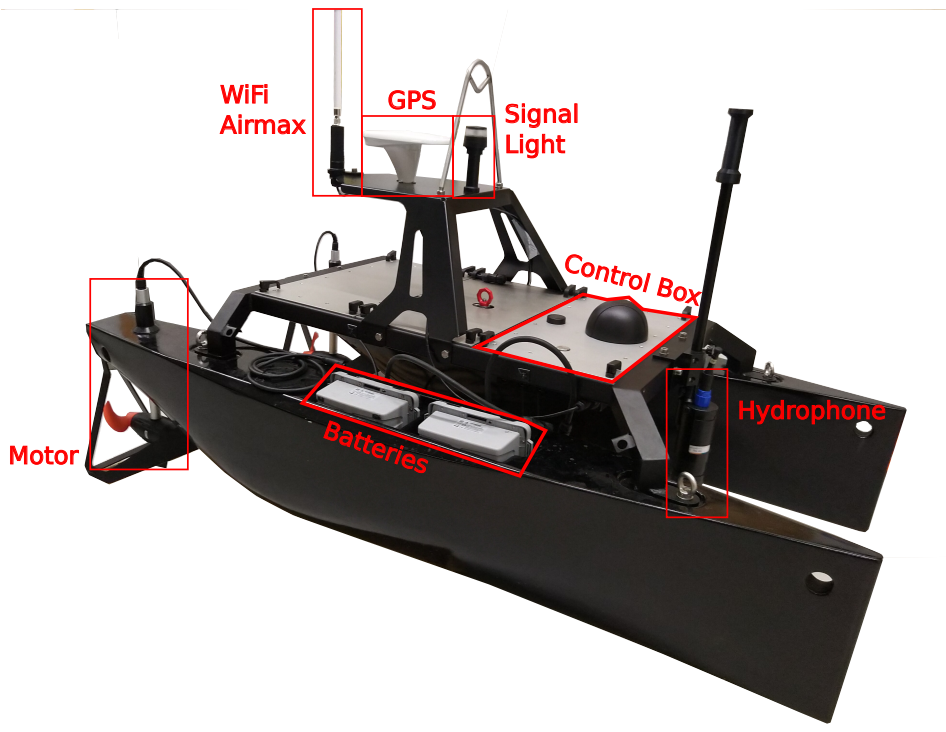
---

<sup>1</sup>Since the procurement of the NTNU Otters, the hull of the Maritime Robotics Otter has been redesigned. The old design has more of a straight keel, making it more direction bound at speed, and thus makes yaw motion harder to achieve.

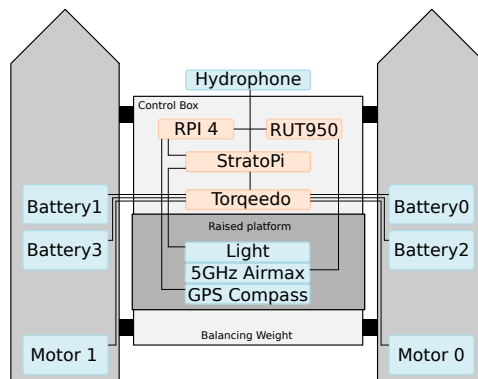
<sup>2</sup>According to the datasheet, when using SBAS.

<sup>3</sup>Pulse-Per-Second

<sup>4</sup>National Marine Electronics Association



**Figure 3.2:** A picture of the Fish Otter with component labels (Before the hydrophone upgrade mentioned in section 6.1).



**Figure 3.3:** NTNU Fish Otter Hardware.

### Powertrain

For the powertrain, the NTNU Otter uses a MR designed system based around Torqeedo Ultralight products. The two Torqeedo Ultralight 403 trolling motors, originally designed

<b>Component</b>	<b>Model</b>	<b>Description</b>
Thrusters	Torqueedo Ultra-light 403	One at the stern of each hull, 180W propulsive power, max 1200 RPM. [70]
Batteries	Torqueedo Battery 915 Wh Ultra-light 403	Two in each hull, total capacity 3660Wh(or 124Ah). [71]
Hulls	Maritime Robotics Otter	Catamaran, 200cm X 108cm, disassembly into parts with weights under 20Kg possible.
Signaling Light	Hella Marine NaviLED 360	1W white.
5GHz Access Point	Ubiquiti Bullet AC IP67	AirMax, IP67 rated, Passive PoE,
5GHz Antenna	Delock 802.11 ac/a/h/b/g/n Antenna	50Ω, N-type connector, SISO (1x1 MIMO)
Positioning	Hemisphere v104s	GPS compass, RS-232 NMEA0183 interface, SBAS, gyro and tilt sensor
Hydrophone	Thelma Biotel TBR700RT/TBRLive	Detects DPPM modulated 63-77 kHz acoustic tags, see section 6.1.
Control Box	NTNU	See table 3.3.

**Table 3.2:** Overview of the hardware components of the NTNU Fish Otter ASV.

for use on Kayaks, is coupled with four Torqueedo Ultralight 403 915Wh batteries through a MR designed interface board. A benefit of basing the design around the commonly available Torqueedo components, is that spare parts can be acquired anywhere international shipping is available.

Communicating with the interface board is done using a CAN interface implementing a MR defined protocol. This allows for controlling the RPM of the individual thusters, as well as receiving telemetry from both motors and batteries.

### **Payload Sensor**

The payload of the vessel, is a digital hydrophone, which lets the Otter detect aquatic animals that acoustic tags have been attached to.

The digital hydrophones used on the NTNU Fish Otters are produced by the Trondheim based company Thelma Biotel. The company's web-page states that its roots are in the technology developed in the experiments of Belchen, Holand and Mohus (one of which was described in section 1.1), and as such, it still has ties with the NTNU DEC [28]. These ties has been a benefit in the development of the NTNU Fish Otters through direct communication, and also as early adopters of the newly released TB Live.

Originally, the TBR700RT hydrophone was to be used, but was replaced by the TB Live hydrophone as it became available. For the motivation behind the replacement, see

section 6.1.

### Communication

For letting the Fish Otter communicate with supporting systems, and also convey telemetry, two systems have been integrated: One based on Ubiquiti AirMax 5GHz equipment and another based on LTE cellular communication. From experiences with the NTNU AutoNaut<sup>5</sup>, cellular network coverage in Norway provides a quality of service sufficient for use on ASVs in coastal areas. To provision for cases when this is not true, the AirMax equipment is included as a backup system. An overview of all communication protocols used, and connected devices is given in figure 7.6.

Internally, the components of the NTNU Otter communicate through wired Ethernet cables, managed by a Teltonika RUT950 LTE router. Two of its four connections are then used, one for the controlling computer, and the other for the AirMax AC.

At a sea trial performed by the NTNU Otter in the autumn of 2019, the range of the AirMax equipment was found to be 1000m in near optimal conditions. The trial was performed with a sector antenna used at the base station.

### The Control Box

The control box is where all the peripherals is connected, and is based around the Raspberry Pi 4 single board computer. To support the necessary interfaces of all the peripherals, the daughter board *StratoPi CAN* by Sferea Labs has been used along with a FTDI USB to RS-232 adapter. A table of the components in the control box along with short descriptions is included in table 3.2. The exact wiring in the control box is documented in figure 3.4. To get more of an overview, figure 3.3 is included.

### System classification

Assuming that roll, pitch and heave motion is negligible, the system is operating in three degrees of freedom: surge, sway and yaw. Having fewer independent actuators than degrees of freedom, the Otter is classified as an underactuated marine vessel, as defined in [74]. The result of this underactuation is the addition of a second degree non-holonomic constraint. Controlling Surge is done by outputting equal thrust from the two thrusters, while motion in yaw is accomplished by outputting differential thrust. The constraint lays on sway motion, which there is no way to directly control.

### Multiple Fish Otter ASVs

NTNU has procured a total of four Otters from Maritime Robotics, where at least three are available for use as fish tracking Otters. For now, hardware has only been created for a single fish Otter because the control box design had yet to be completed. With the addition of the cellular modem in 7.1.2 and the change of hydrophone model described in section

---

<sup>5</sup>Currently managed by the thesis co-supervisor, and documented on <http://autonaut.itk.ntnu.no/doku.php>

<b>Component</b>	<b>Model</b>	<b>Description</b>
Single Board Computer	Raspberry Pi 4B with 4GB RAM	CPU: Broadcom BCM2711, RAM: 4GB LPDDR4, Storage: Micro-SD, 2xUSB3, 2xUSB2, 28xGPIO, 1x Gigabit Ethernet port, for complete overview, see [72].
Interface board for SBC	Sfera Labs StratoPi CAN (SPBC12X)	Connects to RPI4 GPIO pins, Power supply, CAN, RS-485, relay, RTC, buzzer, hardware watchdog. For complete overview, see [73]
Passive PoE injector	Unknown	Adds power to the ethernet cable going to the Bullet AC IP67.
RS232 to USB	FTDI USB-RS232	Connects v104s GPS to RPI4
DIN rail 12-24v booster	Phoenix Contacts CUI Inc. PDQE10-Q24-S24-DIN	PTC thermistors on positive supply leads, Used to provide 24v to the Ubiquiti Bullet AC IP67 through the PoE injector, 10W max.
Power distribution and thruster control	Maritime Robotics Otter Torqeedo interface board Rev.D	Provides a CAN interface to the Torqeedo hardware, as well as power distribution.
Cellular Modem	Teltonika RUT950	4G LTE router, see section 7.1.2.
Cellular Antenna	Teoglas Limited MA741.A.BI.001	2x2 MIMO

**Table 3.3:** Overview of the hardware components used in the control box of the NTNU Fish Otter ASV.

6.1, the hardware design phase is finished, and can be duplicated for the remaining two fish Otters.

### 3.2.2 Supporting Hardware

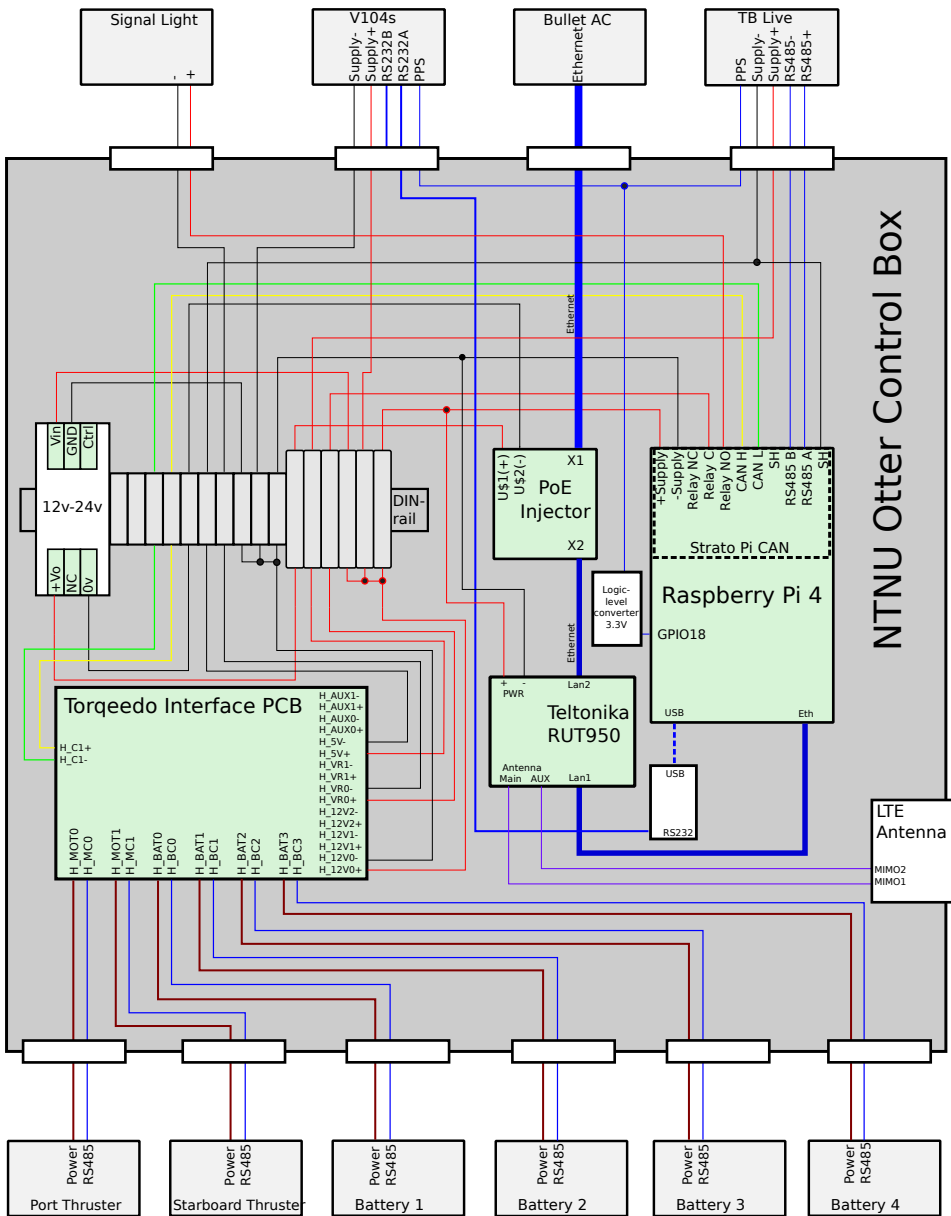
In addition to the ASVs, additional hardware used when performing missions with the Otters are:

- **Communication mast:** In locations where the cellular reception is not suitable for controlling the Otters, a communication mast has been made, consisting of the a broad but shorter range 120° sector antenna and a long range and concentrated dish antenna<sup>6</sup>. The mast is approximately three meters high, with a weight near the ground.
- **Console PC:** A tough laptop, like one of the Dell Latitude Rugged Extremes will be

---

<sup>6</sup>Sector: An Ubiquiti Rocket R5AC-lite with the antenna model Ubiquiti AM-5G19-120. Dish: Ubiquiti Power Beam AC gen2



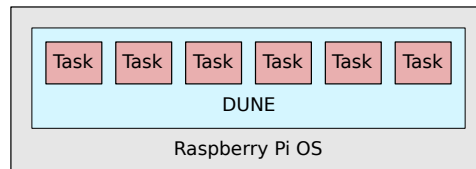


**Figure 3.4:** Figure showing the updated wiring in the NTNU Fish Otter control box.

procured for use on missions with the Otters. Any computer able to run Neptus, and with a wired Ethernet connection (USB to Ethernet can be used) will suffice.

- Controller: A Playstation 4 DUALSHOCK 4 is used to manually control the fish Otter when needed (Berthing, embarking).

### 3.3 Software Design On the NTNU Fish Otter ASV



**Figure 3.5:** NTNU Fish Otter Software.

#### 3.3.1 Operating System

The operating system on the RPI4 used in the NTNU Fish Otter is called Raspberry Pi OS<sup>7</sup>, and is a Debian based GNU/Linux distribution specialized for use on the RPI hardware. In addition to a full desktop variant, there is also a headless<sup>8</sup> version available, named lite, referring to it being lightweigh/less resource intensive alternative. As the use on the vessel will be headless anyway, this option was selected.

#### Configuration

From a stock Raspberry Pi OS install, some configuration has been performed. The first step is activating SSH and changing the super user<sup>9</sup> password.

To control the peripherals through the GPIO header on the RPI4, some customizations have been made. For communication with the MCP2515 CANbus controller on the StratoPi CAN, the SPI bus of the RPI4 has to be activated, the SocketCAN driver enabled, and configured with the correct bit rate for communicating with the Torqeedo interface card. This makes the CAN network interface available to the system as a socket.

If PPS time disciplining is used, the Linux kernel has to be compiled with the additional options activated: `CONFIG_PPS` and `CONFIG_NTP_PPS`. After completion, a PPS source can configured through a GPIO pin, making it an available device in `/dev/pps0`<sup>10</sup>.

More detailed descriptions are available on the Fish Otters public Wiki.

#### 3.3.2 DUNE on the NTNU Otter

The DUNE instance on the NTNU Fish Otter is using a custom configuration, with some purpose built tasks and classes being developed for hardware not supported by the official LSTS build. An selection of the most important tasks are shown in figure 3.6, as well as an indication of message content sent between them.

The `Hardware/Torqeedo` task bridges messages between IMC and the CAN based protocol used by the Torqeedo interface card. To facilitate this, the Otter utilizes a custom IO handle, which is the software part of DUNE that interfaces OS drivers.

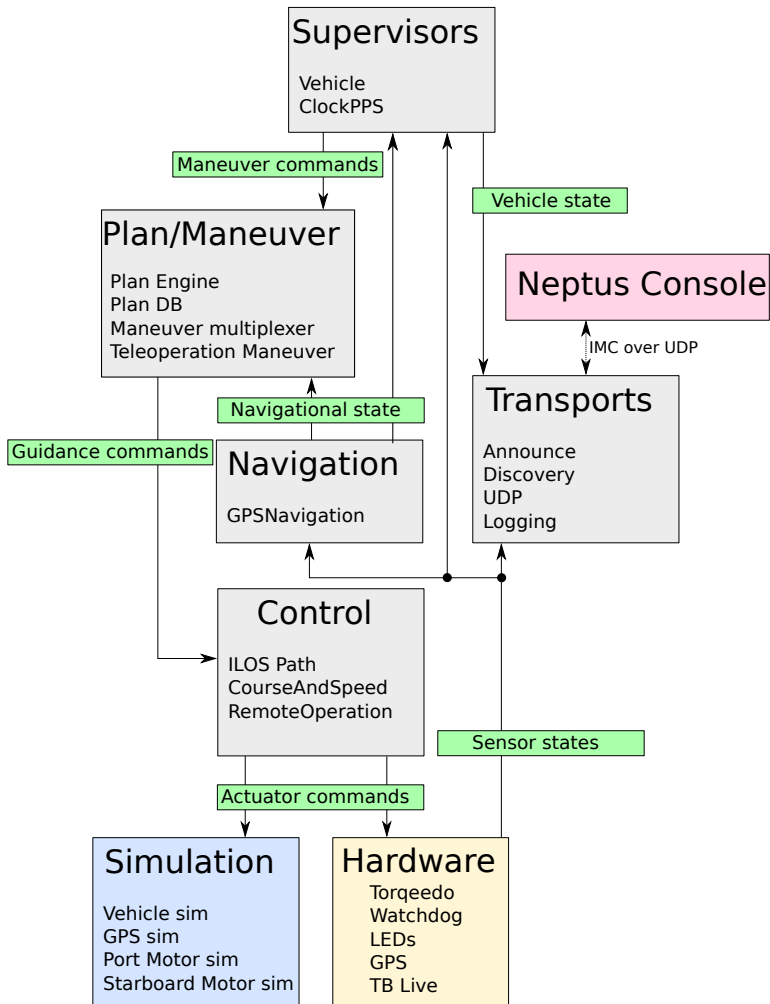
---

<sup>7</sup>Previously named Raspian

<sup>8</sup>Use without local peripherals, often through remote console.

<sup>9</sup>By default named pi.

<sup>10</sup>Device name is configurable



**Figure 3.6:** Overview of the most important tasks used in the DUNE configuration of the Otter, as well as an (incomplete) view of communications between them.

The guidance, navigation and control systems of the configuration uses the GPS to navigate. For control, the CourseAndSpeed<sup>11</sup> task implements PID controllers with optional integral limiters for keeping desired course at a set speed. For guidance, the integrated line-of-sight path controller is used. The benefits of the ILOS controller for underactuated systems such as the Otter, is described in [74], while the controller itself is described by the developer of the task in [75].

To be able to use the configuration both with hardware, and during simulations, the tasks running in the hardware and simulation boxes at the bottom of figure 3.6 can be switched through activating different profiles when executing DUNE. During development, the RPI4 with the StratoPi CAN interface was available, so a third profile called StratoPi has been created that enables some of hardware, while still simulating most of the hardware.

The Plan/Maneuver, Supervisors and Transports tasks are all default from the official LSTS DUNE, with no major modifications done before being used on the NTNU Fish Otter. Of interest is perhaps the Teleoperation Maneuver that can be activated, which switches control to RemoteOperation, allowing the console to control the actuators directly.

### 3.4 The Otter Server

To perform various supporting roles, a public server for the project has been obtained. Provided by the NTNU technical support, it's what they called a semi-administrated server running the GNU/Linux based Ubuntu server. These servers are kept updated and backed up by the IT-department, but can be customized to a certain degree by the user in order to provide the desired services. The customization allowed are:

- Configuring the firewall to allow traffic on specified ports.
- Configuring the tool *pkgsync*, which decides what .deb or .rpm software packages are installed on the server.
- Storing software files.

Access to perform the customisation's can only be had through SSH<sup>12</sup>, so the command line interface has to be utilized.

From the work performed in [1], a documentation wiki had already been deployed, while the deployment of a IMC web-visualizing tool, VPN and IMC Proxy services are described in section 7.2, 7.1.3 and 7.1.4.

---

<sup>11</sup> Somehow misleadingly called HeadingAndSpeed in the LSTS DUNE, but uses the course over ground from the GPS.

<sup>12</sup>Secure Shell: Encrypted access to network services over unsecured networks.

# Designing a Database for Providing A Priori Environmental Information

Knowledge of the vessel surroundings is necessary during autonomous operation, and may also be beneficial when deciding where to prioritize when searching for fish. In its current configuration, the NTNU Fish Otter has no sensors to survey its surroundings, and thus must rely only on a priori knowledge of its environment that can be used in combination with the GPS position<sup>1</sup>.

This section describes the design of a database containing a priori environmental information, how the data is extracted from the S-57 ENC standard data format, and potential additional sources of data. The use of this database is described in chapter 5.

## 4.1 Available data

The Otter will mostly operate in the Norwegian fjords, large rivers and lakes in mild weather. Because of this, the scope of the data search was limited to data that are available for Norway.

To avoid grounding, bathymetric data is needed. The Norwegian mapping authority *Kartverket* has developed a webservice called *Geonorge*, available at <https://geonorge.no>, that describes itself as being the *national website for map data and other location information in Norway*. Here, multiple datasets with bathymetric data for the Norwegian coast can be found.

Due to Norwegian Law (prop. 116 L (2016-2017)), bathymetric data with higher resolution than 50 meters between every sounding included in a dataset are considered confidential in accordance to the national security act. In order to access the most detailed datasets, with bathymetric data available in a 5m-5m grid, the Norwegian armed forces must be consulted. According to discussions with *Kartverket*, they are not likely to release

---

<sup>1</sup>The Hemisphere GNSS v104s has a gyroscope and inertial tilt sensors for pitch and roll in addition to GPS. They could possibly be utilized in some way, but has not been considered in this thesis

the data for use in autonomous vehicles, or probably not for the extent that is desired. For non-military use, data with 50m resolution is probably the best available within the Norwegian coast<sup>2</sup>.

For bathymetry in 50-50m grids, there are two datasets available. One that covers the whole country, which is based on some soundings and interpolation. During testing, it was discovered that some areas that were right by land, had a depth of 100m or above, which is obviously wrong. According to contacts at Kartverket, it is a known issue with this dataset, due to it not using depth contour data in the interpolation. However, another dataset containing data from actual soundings was available, but only covering most of the Norwegian coast, and not all. One of the areas that it did not have data for, was around Børsa, where the Fish Otter is being tested.

After requesting more data, Kartverket gave us access to the ENC's in S-57 format, which unlike the bathymetric data, is meant to be used for navigational purposes, and therefore the author assumes it to be more reliable. It also contains other data relevant for navigation, of which some are interesting to the Otter.

## 4.2 Storing and Making Data Accessible

To effectively store and use the a priori geographical information data on the vessel, a local SQLite3 database will be stored in each vessel. The SQLite database was chosen because it doesn't need connection to a server, and also because DUNE already came with the necessary libraries. Another benefit is that it's a very portable database because it stores the entire database in a single file.

As the SQLite3 is a relational database, a solution for storing geometric objects must be found. In this thesis, this is the approach taken:

- Objects that are points by nature, like buoys and piers, are stored as (Latitude, Longitude) pairs in separate tables according to type.
- Objects that represent areas by polygons are either converted into two-dimensional grids, or by extracting the vertices describing the polygons.

## 4.3 Data Extraction

Working with the data from S-57 ENC's can be done in various ways, but Kartverket recommended using the FME suite, which is introduced in 2.4.5. Besides supporting S-57 ENC's, the suite supports multiple other formats, and even has SQLite3 support, which is used in this thesis.

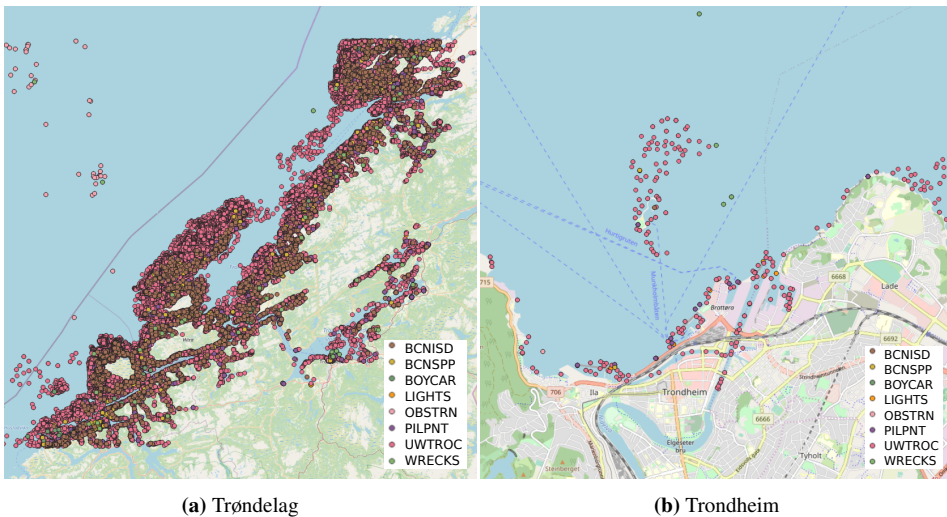
An alternative would be to use the GDAL library, as is done by [15] for S-57 objects. The benefit of using FME, is that it provides a GUI that allows novices like the author to quickly perform advanced transforms on the data. The drawback is that it's closed-source and not free.

---

<sup>2</sup>With some small areas as an exception, where the higher resolution grids have been declassified.

### 4.3.1 Extracting Points of Interest from S-57 ENC

The ENCs contain multiple features as point data that are of interest for an autonomous vessel. For now, we have chosen to extract the objects BOYCAR, BCNSPP, BCNISD, LIGHTS, OBSTRN, PILPNT, UWTROC and WRECKS, but others may be interesting to add at a later point in the Otter development. A description of all objects is given in table A.1. A visualization of the extracted data is shown in figure 4.1.

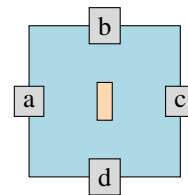


**Figure 4.1:** A selection of points of interest from NO ENCs.

### 4.3.2 Two-dimensional Batymetric Grids

With a two-dimensional grid, getting information about the area around the vehicle, can be reduced to a simple SQL statement: "SELECT \* FROM TABLE WHERE Lat BETWEEN a and c Lon BETWEEN d and b;". This is visualized in figure 4.2, and returns all values in a square around the orange vehicle in the center.

A drawback with creating grids of data from polygons, is that the dataset becomes much larger than the original polygons, with a lot of redundant data. The alternative, storing the polygons in the DB is not practicably possible in vanilla SQLite if searching on locations is desired. This may be atoned for by switching to a spatial database, but the effects of this has not been researched further.



**Figure 4.2:** Square limits.

### Assessment of Data Sources

While the data from GeoNorge provides better depth resolution, there are a multitude of reasons why using data from the S-57 ENC would be a better match for navigational

purposes:

- Reliable data for all areas where the Otter might like to go.
- Not being limited to a grid with 50 meter resolution, because it could result in smaller ground areas in between points going unnoticed. This problem can be mitigated if the grid has higher resolution.
- The S-57 data is considered more trustworthy, as it's intended use is navigation and charts for electrical chart plotters.

A pros/cons table is given in table 4.1, which summarizes the available datasets for two-dimensional grids with depth data.

<b>Dataset</b>	<b>Resolution</b>	<b>Pro</b>	<b>Con</b>
Geonorge Raw Data [76]	Varying	The most detailed depth soundings available along the Norwegian coastline	Restricted access, Not suited for navigation.
Interpolated depth grid Geonorge[77]	50	Complete coverage of the Norwegian coast	Very inaccurate both in space and depth.
MAREANO depth soundings Geonorge [78]	50, 25, 5	Depth resolution	Better spatial resolution than 50m restricted for most of the Norwegian coast. Incomplete coverage of the Norwegian coast.
Grids made from S-57 DEPARE	User defined	User defined grid resolution, coverage of all areas where S-57 DEPARE is available, including non-Norwegian territories	Low depth Resolution.

**Table 4.1:** Datasets available for two-dimensional grids.

### Reprojecting ETRS89 utm33 .xyz point clouds to WGS84 radians

The depth soundings from Kartverket, comes in .xyz<sup>3</sup> point clouds, and are referenced with the ETRS89 datum projected in UTM zone 33 coordinate system. The depth is given as z in meters below the surface.

Pre-processing this data to simplify working with it in its target, the LSTS toolchain, is done by transforming the datum to WGS84 with the coordinate angles given as radians.

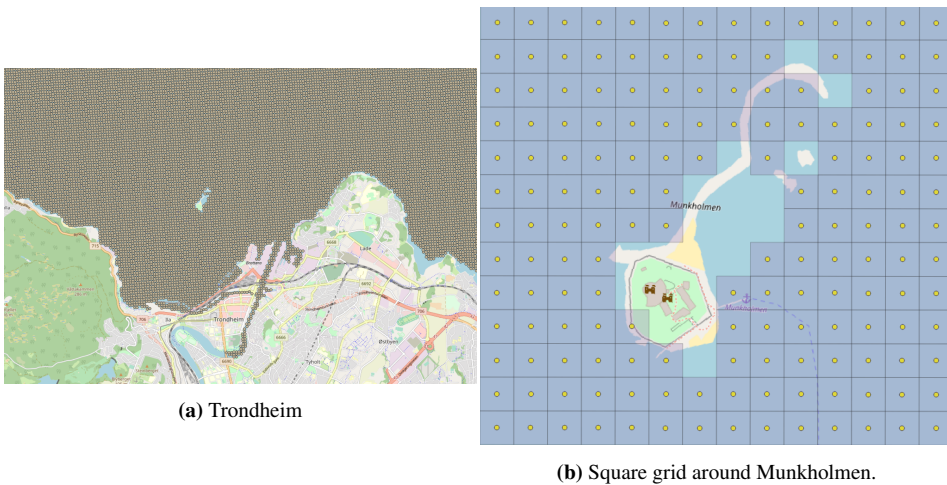
---

<sup>3</sup>Similar to comma separated values, but only for three columns formatted with space as the delimiting character.



The FME workbench given in B.1 shows how this was done. The resulting data is visualized in figure 4.3a. The projecting introduces a small precision loss in the coordinates, probably caused by drift between WGS84 and ETRS89. In QGIS, the measured error was below 20cm in each direction, which may also just be an errors in the visualization (caused by using fast but inaccurate transformations before visualizing). Either way, they will be considered as negligible for the rest of this thesis, which is probably true for the use made of them. The spatial accuracy of the original data is also unknown.

Figure 4.3b shows how a grid looks around the points, and also shows that depths are known for all tiles that are not land.



**Figure 4.3:** The bathymetry readings from GeoNorge.

### Grid from S-57 DEPARE

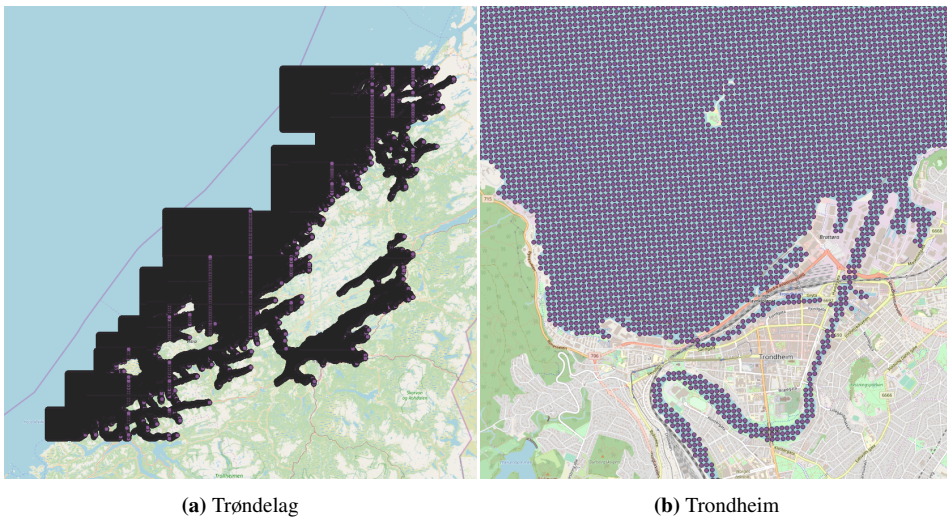
In the S-57 ENC, there are a soundings object SOUNDG and a depth area DEPARE object that contains information about the bathymetry. While the depth resolution is greater in SOUNDG, the aggregated data in DEPARE is considered sufficient for navigational use. It also has the benefit of describing an area, instead of discrete points, making it easier to attach depth values to a two-dimensional grid without any interpolation between soundings being needed.

The DEPARE object describes areas where depth is between an upper and lower limit, DRVAL2 and DRVAL1. For navigational use, DRVAL1 can therefore be used as the shallowest depth expected at a certain point.

To create two-dimensional grids with the desired attributes from polygons in the DEPARE ENC object, the FME workbench shown in B.4 has been created. It creates a square grid in a given resolution, that is given values from attributes from DEPARE. A filter then removes areas that are not described by DEPARE, the coordinates are extracted and named *Lat* and *Lon*, and finally, the result is saved as a table in a SQLite DB.

This workbench is then batch deployed on all ENCS with the name "NO4", with the

'4' indicating its purpose is navigation (as described in table 2.2). The result of this can be seen in figure 4.4.



**Figure 4.4:** Two-dimensional array of points based on DEPARE from NO4 ENC. The purple points all have DRVAL1 and DRVAL2 defined.

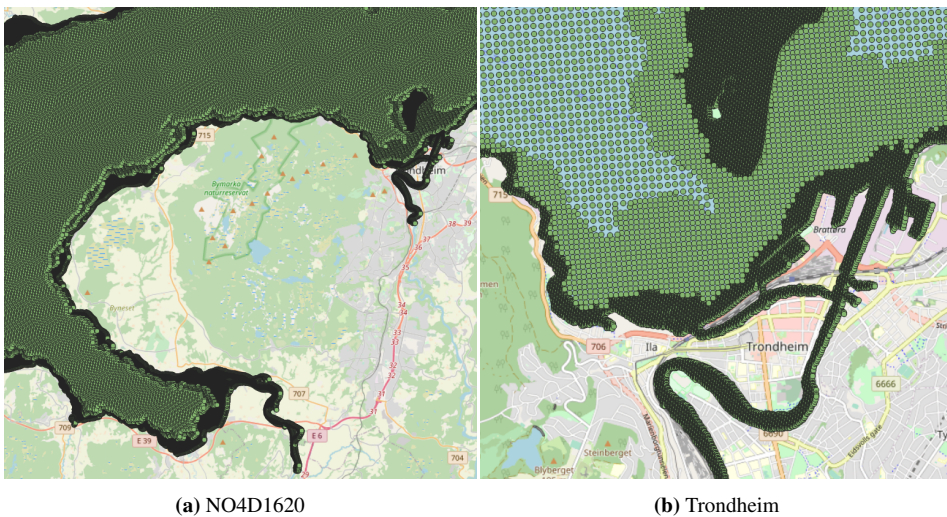
### Two-dimensional DEPARE bathymetric grids with varying resolution

A possible extension to this way of storing data, is to use different resolutions based on depth. This is possible in FME by using an attribute filter transformer, which allow us to filter out shallower areas, and have a higher resolution in these areas than in the deep areas where there is no danger of grounding. The example shown in figure 4.5 has 10m between points in areas with  $DRVAL1 \leq 20.0m$ , and have 50m between points with  $100m > DRVAL1 > 20.0m$ . For areas with  $DRVAL1 \geq 100.0m$ , there is 75m between points.

### 4.3.3 Extracting Vertices from S-57 Polygons

The two-dimensional grids uses a lot of storage, and searching for the desired area in the data also becomes computationally expensive. An alternative way to handle the DEPARE polygons has therefore been explored; Extracting the vertices of the polygon. This would mean that the vessel only gets to know the information of an area when it crosses between polygons, and thus handle less data. For the DEPARE object, this means that the vessel only gets to know the depth as it explores an area with different depth, which is what is wanted for implementing anti-grounding.

For this to work, there must be some known minimum distance between vertices, which there is not in the S-57 ENC. Because of this, redundant vertices have been made, so there is at least one vertex per  $n$  meters.



**Figure 4.5:** Two-dimensional array of points based on DEPARE from NO4D1620 with higher resolution in shallow areas. 10m-10m in shallower than 20m, 50m-50m for depth between 20m and 100m, and 75m-75m for depths deeper than 100m.

For further reducing the amount of data handled with this method, just including the depth ranges that are interesting can be done. For the purpose of anti-grounding, depths above a certain threshold can be ignored, because the vessel is only interested in knowing if it's entering a shallow area.

The workbench to extract vertices from S-57 polygons is shown in figure B.3. The vertex extractor is doing the actual work in this bench, while the other transformers filter the the data, makes vertices denser, gives the coordinated the names Lat and Lon; and converts the coordinate angles from degrees to radians.

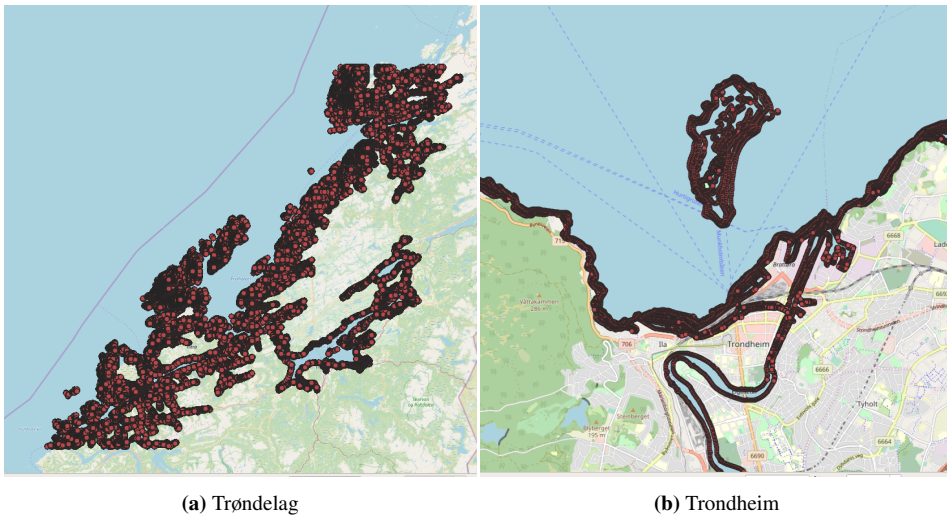
## 4.4 Discussion

While this chapter has considered data covering the ocean, the FME workbenches for working on DEPARE objects from S-57 ENC's is also functional on shapefiles<sup>4</sup>. The Norwegian Water Resources and Energy Directorate uses shapefiles in their database of Norwegian lakes larger than 2500 m<sup>2</sup>, with some 600 of the lakes having available depth information [79]. Operating the Otters in lakes is a possible future scenarion, where these shapefiles may be used to create a DB for the Otter.

### 4.4.1 Choosing the Resolution

For the two dimensional grids and for the DEPARE vertices, finding the balance between spatial resolution and the amount of data is an important topic. On the one hand, better

<sup>4</sup>By exchanging the S-57 reader for a shapefile reader



**Figure 4.6:** Vertices on contour around depth areas (DEPARE) from NO4 ENC data. DRVAL1 and DRVAL2 is defined for all the red points.

resolution will lead to smaller features such as narrow portions of the navigable area being detected. On the other side, larger resolutions will make searching the data slower, and for motion planning on the two-dimensional grid, path searching is made more computationally expensive. The 50m·50m grid shown in figure 4.4 was made more as a demonstration, and is not considered feasible for operations close ( $< 50\text{m}$ ) to the land.

Another aspect of choosing the resolution should be based around the expected accuracy of the GPS/GNSS. According to the producer, the estimated accuracy of the v104s is 1m at 95% of the time with SBAS augmentation[80], which is supported over Europe and most of Northern America<sup>5</sup>. Based on this, the grid resolution should be above 1m.

A definite conclusion of the most suited resolution for the Otter is not made in this thesis, but a 2m grid is experimented with in section 5.4.1 for path planning purposes.

## 4.4.2 Optimizing Database Performance

In order to increase the querying speed, Latitude and Longitude have been defined as the primary key in all extracted data. In addition to this, an index has been added to depth information fields to allow fast queries that restrict depth. Adding the extra indexes makes the database file larger, and a decision therefore has to be made at a later point if this is merited or not.

More specialized ways of optimizing the database performance is to change the SQLite3 file parameters PARAMS. For instance, SQLite3 databases can be used as in-memory databases, potentially increasing its performance. Storing the database in memory could also be possible through `ramfs` or `tmpfs` in the OS. None of these suggestions have been implemented in this thesis, but is worth exploring in future works.

<sup>5</sup>The v104s SBAS supports operation in the EGNOS and WAAS areas [80].

An entirely different approach is to change to a database format that is optimized for storing and querying spatial data. An interesting candidate is SpatialLite, which is an extended version of SQLite but with spatial indexes and spatial querying algorithms already implemented. SpatialLite also has the benefit of supporting storing and searching through polygon objects. In hindsight, using this might have provided a better solution than the one presented in this thesis.

A final measure for optimizing the database performance is that only the data relevant for a mission should be included in the vessel. For the NTNU Otter, restricting the area of the grid is one example of this.

### 4.4.3 Weather Forecast Services

As a part of searching for available data, publicly available APIs for weather forecasts was explored. The desired forecasts were for wave, wind and current in services covering Norway. As the services may prove to be useful in the future of the Otter project, they are included here:

- Barentwatch API: <https://code.barentswatch.net/wiki/display/BWOPEN/API-Documentation>
- Ocean Forecast from The Norwegian Meteorological Institute: <https://api.met.no/weatherapi/oceanforecast/0.9/documentation#/>
- Weather forecast from Yr, with a list of more APIs from The Norwegian Meteorological Institute: <https://developer.yr.no/>

A potential use for these in the fish tracking system, is as a way to contextualize the weather conditions while a fish is being tracked. While not used in this thesis, they may be integrated to the online data visualizing solution explored in 7.2.



# Utilizing the A Priori Data in LSTS Toolchain

To make use of the a priori data gathered in chapter 4, the DB has to be available on the vessel, and DUNE needs to be able to read and make use of the data. The libraries written to accomplish this is described in section 5.2. Tasks that uses these libraries are then created, consisting of two methods of anti-grounding in section 5.3 and a global path planner in section 5.4.

In addition, a short introduction to a Neptus implementation developed by Alberto Dallolio is given in section 5.5, showing how the same database and SQL queries has been used to increase the situational awareness for the operator.

## 5.1 Implementation Design Choices in DUNE

When implementing new functionality for DUNE, a decision has to be made if it should be as a task, or as a class that is used by tasks. Two example designs of implementing it as a task is: 1) that all functions using the DB, is cobbled together in a single task, or 2) that other task can access the information through IMC messages being requested from a reader task. For use cases requiring many queries, like the path planning described in section 5.4, this would add a lot of overhead, and come with a significant performance penalty. Instead, the functionality has been implemented as a set of classes in the DUNE library<sup>1</sup>.

This comes with the challenge of possibly having multiple threads accessing the same DB file at once. While SQLite has multi-user support, performance could suffer if multiple users/threads commits changes at the same time. This is not considered a problem in the work presented here, because none of the classes using the database needs to permanently alter the DB, which means that the DB connection mode can be set to read-only.

---

<sup>1</sup>Source folder: src/DUNE/SituationalAwareness



## 5.2 Situational Awareness Classes in DUNE

An overview of the C++ classes created for accessing the DB created in section 4 is given in table 5.1, and a class diagram is given in figure 5.1. The base class *LocationData* opens a DB connection in the constructor, and closes the DB in the destructor. It also provides a function that makes a SQL clause condition for use when a square around a point is wanted, as shown in figure 4.2.

Class name	Description	Data used
DensifiedVertices	Used with table of vertices	Depare vertices that have been densified
TwoDGrid	Used with two-dimensional grids	Kartverket soundings and grid made from S-57 object DE-PARE
PointOfInterest	Used for point data	Points extracted from S-57 objects.
PathPlanner	Implements the A*-algorithm to find paths between two locations from the depth grids.	See TwoDGrid
LocationData	Base class used for treating data with locations	All of the above

**Table 5.1:** C++ classes implemented in DUNE to get data from the DB.

### 5.2.1 Data Containers

To store the spatial data, a superclass storing coordinates was created, and then inherited from to provide functionality for additional attributes. These points could then be stored in C++ STL containers to keep series of data.

The superclass is named `LocationData::LocationDataContainer_t`, and stores latitudes and longitudes. The `<operator2` is overloaded, in order to allow a special sorting method to used in containers. The implemented sorting method works by first sorting on latitude, and for values with equal latitude, sort on longitude. This makes searching in sorted STL containers, such as the keys in `std::map` possible.

Two basic subclasses has been implemented, named `DensifiedVertices::DensifiedVerticesContainer_t` and `TwoDGrid::DepthDataContainer_t`. Additionally, the `STD::vector` container type with each has been defined as `DEPAREVector` and `DepthVector` to simplify code reading.

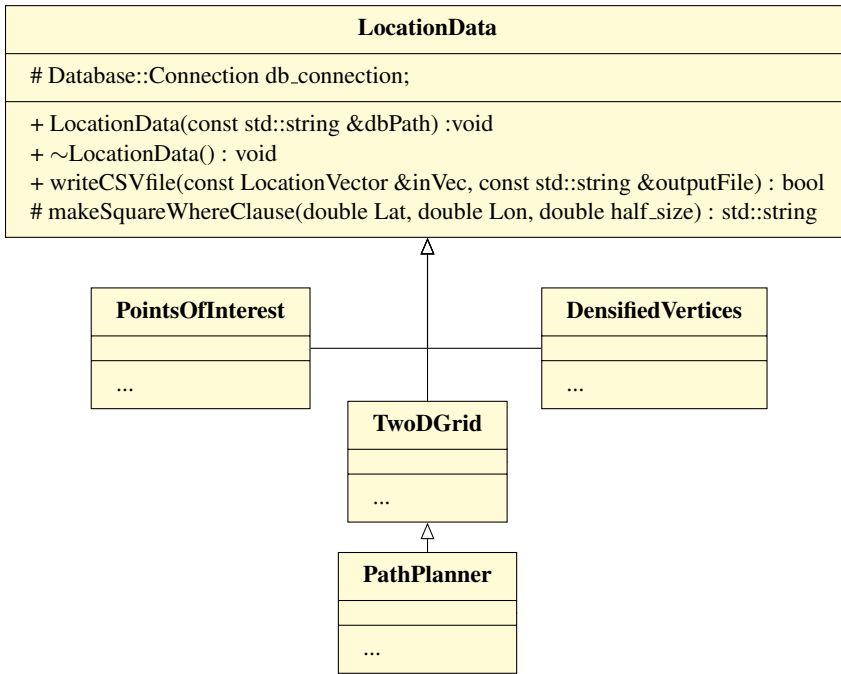
### 5.2.2 PointOfInterest

The points of interest (POI) class contains a single function that gets all POI of an Object within a square, and returns a `std::vector` with locations as `LocationData_t`. For de-

---

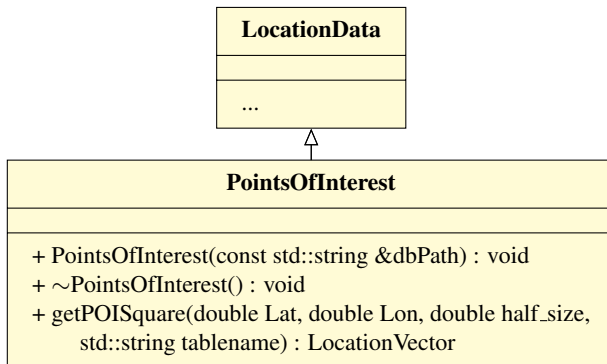
<sup>2</sup>The less-than operator.





**Figure 5.1:** Class diagram showing all the C++ classes implemented in DUNE to get data from the DB.

bugging, the inherited `LocationData::writeCSVfile` is used without modifications. For more details, see the class diagram in figure 5.2.



**Figure 5.2:** Class diagram showing the PointsOfInterest C++ class developed for DUNE.

### 5.2.3 DensifiedVertices

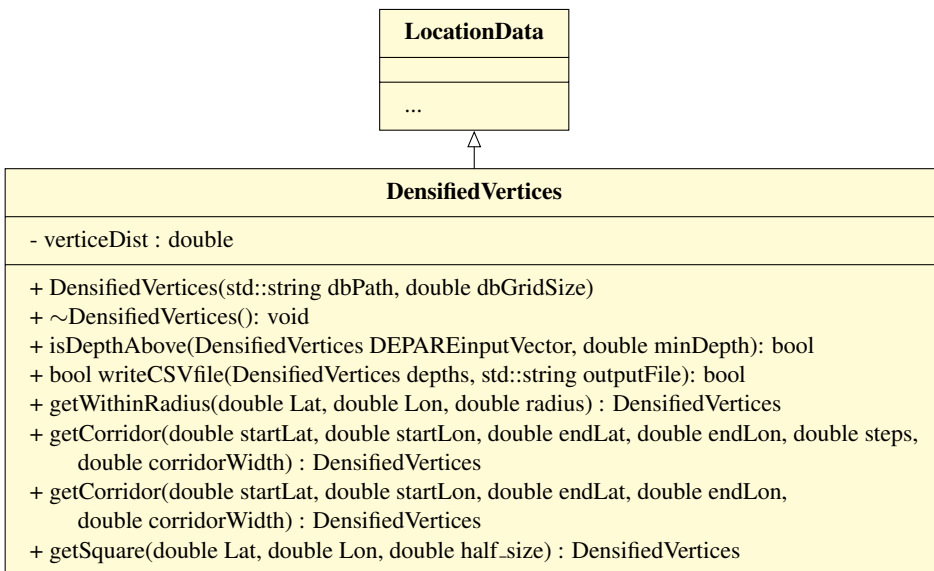
The class for using the vertices extracted in section 4.3.3 is called `DensifiedVertices`, and its class diagram is given in figure 5.3, with code included in the appendix D.3.2.

`getSquare` returns all vertices in a square area around a location, which is useful for checking the surroundings of a position.

`getWithinRadius` does exactly the same, but removes vertices that are farther away than the radius before returning, effectively returning a circle. This requires some additional computation, because the distance from (Lat, Lon) is calculated for each vertices returned from `getSquare`, so use should be restricted to small radiuses.

The `getCorridor` function works by making multiple points along the line between (startLat, StartLon) and (endLat, endLon), and then running `getSquare` with `half_size = corridorWidth`. The results from all `getSquare` function calls is merged together and returned. The result ends up being a corridor, hence the functions name.

The `isDepthAbove` iterates through a `DensifiedVertices` vector checking for `DRVAL1 < minDepth`. This can be combined with `getCorridor` to check a transect for crossings into areas that are shallower than a minimum depth. This is one of the solutions developed in section 5.3.



**Figure 5.3:** Class diagram showing the `DensifiedVertices` C++ class developed for DUNE.

### 5.2.4 TwoDGrid

To use the grids with bathymetric data created in section 4.3.2, the class `TwoDGrid` has been created in DUNE. The class diagram is available in figure 5.4, while the code is included in appendix D.3.3.

The functions `getSquare`, `getWithinRadius` and `getCorridor` performs the same querying as in the `DensifiedVertices` class, but in the two-dimensional grids.

The function `getClosestDepths` gets a square that reaches two times the grid resolution in each direction. In this square, the closest value in each quadrant is selected and returned. The result is always a list of four grid positions when not at the coastline.

Checking a transect in the two-dimensional grids for grounding is complicated, because areas with land are not represented in the dataset. The question then becomes how does one detect if parts of a transect is land? This is where the `getClosestDepths` functions comes in handy, because land is identified whenever less than four grid positions are returned. This is used in `checkTransect` to check points at regular intervals along a transect, and returning locations that are land, have no data or that is closer to land than the gridsize, meaning the depth cannot be guaranteed.

A side effect of the method used in `checkTransect`, is that a safety buffer around the coast is created. This can either be seen as positive, in that it deems uncertain areas not navigable, but can also be negative, because it reduces the navigable area of the vessel.

If the transect is deemed not to go through land areas, the returned coordinates come with depth measurements. Checking a depth threshold for the transect can then be done by iterating through the results, and verifying that the depths are acceptable.

### 5.2.5 SQL Queries

All the queries that have been written, uses a square selection to accomplish their purpose. In order to avoid writing the same code in every class, the function `makeSquareWhereClause` was created in the superclass. It creates a `where` clause that limits the range of returned values from a query. The SQL formulation for this is given in code listing 5.1, with the letters in the square brackets representing the four edges of the queried area, as shown in figure 4.2.

**Code 5.1:** SQL "where" clause created by `makeSquareWhereClause`.

```
1 Lat between [d] and [b] and Lon between [a] and [c]
```

The simplest example of a complete SQL query using this `makeSquareWhereClause` is given in code listing 5.2, which is the query used in the `getSquare` and `getWithinRadius` functions.

**Code 5.2:** SQL query that returns a square.

```
1 select Lat, Lon, Depth from table where Lat between [d] and [b]
   and Lon between [a] and [c];
```

The SQL query used in `getClosestDepths` to get the four closest depth readings from a point also uses the same query, as shown in line 2, 5, 8 and 11 of code listing 5.3. The results are then used in four separate queries that each get the closest depth reading of each quadrant around (`[vesselLon]`, `[vesselLat]`). Finally the union command makes the four queries return only a single result.

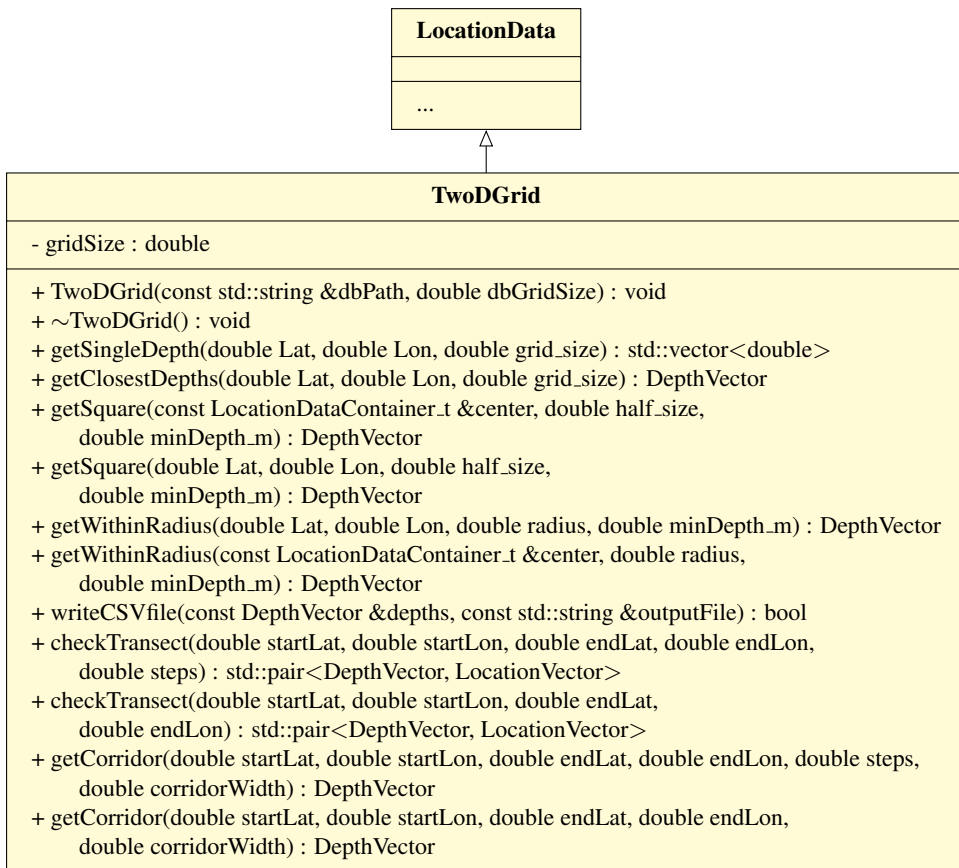


Figure 5.4: Class diagram showing the TwoDGrid C++ class developed for DUNE.

Code 5.3: SQL query that returns the four closest depths around the vessel.

```

1  select min(Lat+Lon), Lat, Lon, Depth from
2      (select Lat, Lon, Depth from table where Lat between [d] and [
3          b] and Lon between [a] and [c])
4  where Lat >= [vesselLat] and Lon >= [vesselLon]
5  union select max(Lat+Lon), Lat, Lon, Depth from
6      (select Lat, Lon, Depth from table where Lat between [d] and [
7          b] and Lon between [a] and [c])
8  where Lat <= [vesselLat] and Lon <= [vesselLon]
9  union select min(Lat-Lon), Lat, Lon, Depth from
10     (select Lat, Lon, Depth from table where Lat between [d] and [
11         b] and Lon between [a] and [c])
  
```

```
12 | where Lat <= [vesselLat] and Lon >= [vesselLon];
```

## 5.2.6 Discussion

The observant reader may have noticed that the functions in the subclasses were relatively similar. Adding these functions to the superclass would be a better solution, avoiding having to make one function in each class performing almost identical task (such as the get square functions). The problem with such an approach, is that the data is returned in `std::vector` containers. The vector container is dependent on having a fixed and known object size at compilation time, which it would not have if it stored objects of different types. The multi-class approach is therefore a compromise which makes the code less elegant, but should not affect performance. A proposed solution is to store pointers to objects instead of the actual objects in the container. This option has not been further explored.

## 5.3 Anti-Grounding Task in DUNE

To keep the vessel in the navigable area of the sea, a system for checking plans and monitoring the depths in the vicinity of the vessel has been developed, in the form of an anti-grounding task in DUNE. As the functionality of the task is *supervising* the depth, it has been placed in `dune/src/Supervisors/Grounding/`.

Two alternatives has been developed to accomplish anti-grounding, one based on the two-dimensional grid, and the other based on the DEPARE contour vertices.

Checking plans in DUNE, is accomplished by subscribing to `IMC::PlanSpecification`. Each message contains one or more maneuvers which have to be checked. The maneuvers may be of different types, each with their own speciality that the grounding task can handle separately.

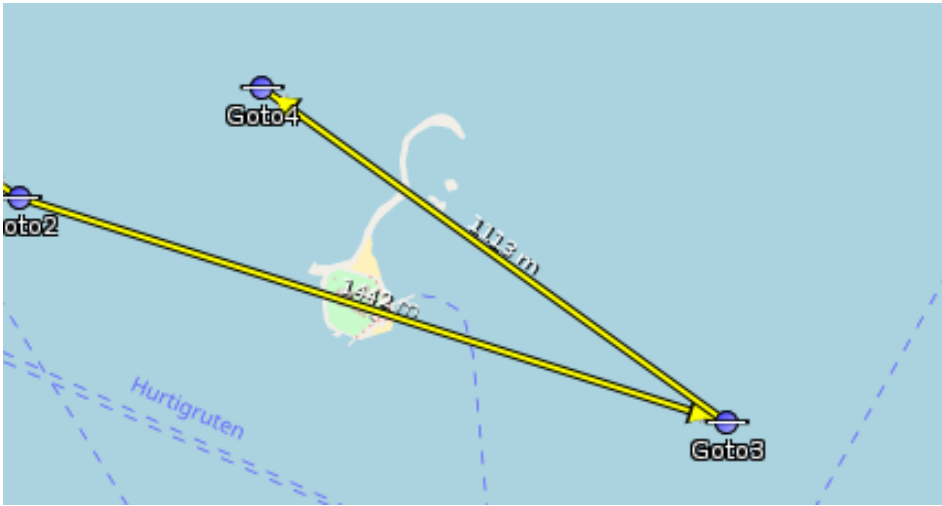
The only maneuver currently supported by this task, is the GoTo maneuver. This maneuver creates a straight line path from one point to another that is to be traveled at a given speed.

Acknowledgement: Alberto Dallolio wrote the code in the DUNE task that iterates through a received plan, while the author developed the `checkTransect` code. Alberto Dallolio also wrote the periodical grounding check on the vessels surroundings in the task.

### 5.3.1 By two-dimensional grids

The two-dimensional grids created in section 4.3.2 only contains points that are on the ocean, and has no representation of land. To be able to detect land, one either has to add data for land<sup>3</sup>, or use the absence of points in an area to deduce that the area is land. The

<sup>3</sup>Readily available from the S-57 object LNDARE.



**Figure 5.5:** Plan used to demonstrate anti-grounding transect checking.

latter is chosen in this implementation, which creates an instance of the `TwoDGrid` class, and uses the `checktransect` function.

For this to be effective, the square has to be  $n_{grid} * \sqrt{2}$ m in width and length, where  $n_{grid}$  is the grids resolution. Due to rounding errors and small possible position shifts when projecting ETRS UTM33 grids to WGS84, some small margin  $m_e$  should be added  $n_{grid} * \sqrt{2} + m_e$ m.

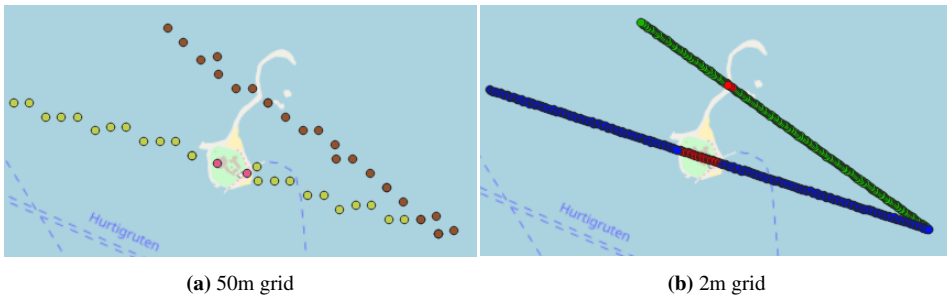
Iterating through the maneuvers of the plan, a `checkTransect` is called on each `goTo` maneuver.

A special case of the `GoTo` maneuver is when it is starting in a different location than where the vessel is located. Then the vessel first travels to the starting point of the maneuver. To check the transect between the current vessel position and the starting point of the maneuver is solved by having the task subscribe to the IMC message `IMC::GpsFix`, and storing the most recent location. This can then be used as starting position of the transect.

If grounding is detected in an activated plan, an `IMC::Abort` message is sent, which makes the vehicle supervisor end the plan execution, and stopping the vessel. An alternative to this, would be searching for a valid path with the path planner described in section 5.4.

In addition to checking received plans, the task can also perform regular depth data queries for the current location, giving the operator warning if the vessel is nearing or entering shallow water. For this, using the already received `IMC::GpsFix` is used. With the `getWithinRadius` function, the information about the surroundings are queried, and the result can be checked for shallow depth.

Based on the path shown in figure 5.5, the results from this anti-grounding approach is shown in figure 5.6.



**Figure 5.6:** The results of running `checkTransect` on Munkholmen, Trondheim with grids of depth. Red points are detected land area, other colors show the navigable waters of the transects (with no depth limit set).

### 5.3.2 By DEPARE Contour Vertices

For checking plans, the DEPARE contour vertices can also be used. By checking a corridor around a transect, all depth changes along the transect is detected.

In order to guarantee that a vertex is detected, the corridor width has to be larger than largest gap expected between the vertices of any contour in the data. When the vertex distance of the data is set to 5m, as used in the data shown in figure 4.6, a result such as figure 5.7 are returned.

As each crossing of a line of vertices signify entering an area with different depth limits by a different DRVAL1 and DRVAL2, how is the vessel to know if it is entering a shallow area or leaving it? The proposed solution to this, is to store the previous registered DRVAL1. When meeting a new vertex with DRVAL2 equal to the previously recorded DRVAL1, the detected vertex signifies that one is entering the area with the detected DRVAL1, else, one is leaving the area.

### 5.3.3 Discussion

The queries used for the anti-grounding examples use the square queries with the vessel at the center of the square. Alternative, this square could be skewed in the direction of movement to get more relevant data in a longer horizon. Ideally, the area checked should be cone shaped around the vessel, with the broadest part of the cone sitting in the direction of movement, but making those kinds of queries efficiently in SQL would be difficult.

The depth soundings of the ENC's is referenced to NMAP's *sea chart zero*<sup>4</sup> reference, as is defined in [81]. This level is placed below or at the lowest astronomic level reference, which is the the lowest expected tidal depth without accounting for the effects of the weather. Lower depth than those read from the ENC data is therefore not expected to occur frequently.

A benefit of using the DEPARE vertices rather than the `twoDgrid`, is that it requires less data to be stored to accomplish the same effective resolution for anti-grounding purposes. It would also make it easier to make maneuvers that follows the contours of sub-aquatic

<sup>4</sup>Translation from Norwegian word *Sjøkartnull*.



**Figure 5.7:** The results of running `checkTransect` on Munkholmen, Trondheim with contour vertices. Points show returned vertices with known DRVAL1 and DRVAL2 for each transect.

structures, which might be an useful feature for searching along the coastline. The drawback is that it does not contain the amount of information that the MAREANO grid has. In the used dataset, as shown in figure 4.6, a filter has been applied to remove areas where DRVAL1 is above 20m, in order to reduce the amount of vertices in the database. Keeping this filter would also lead to having no information about depths in deeper areas

### Grid Areas Considered Grounding

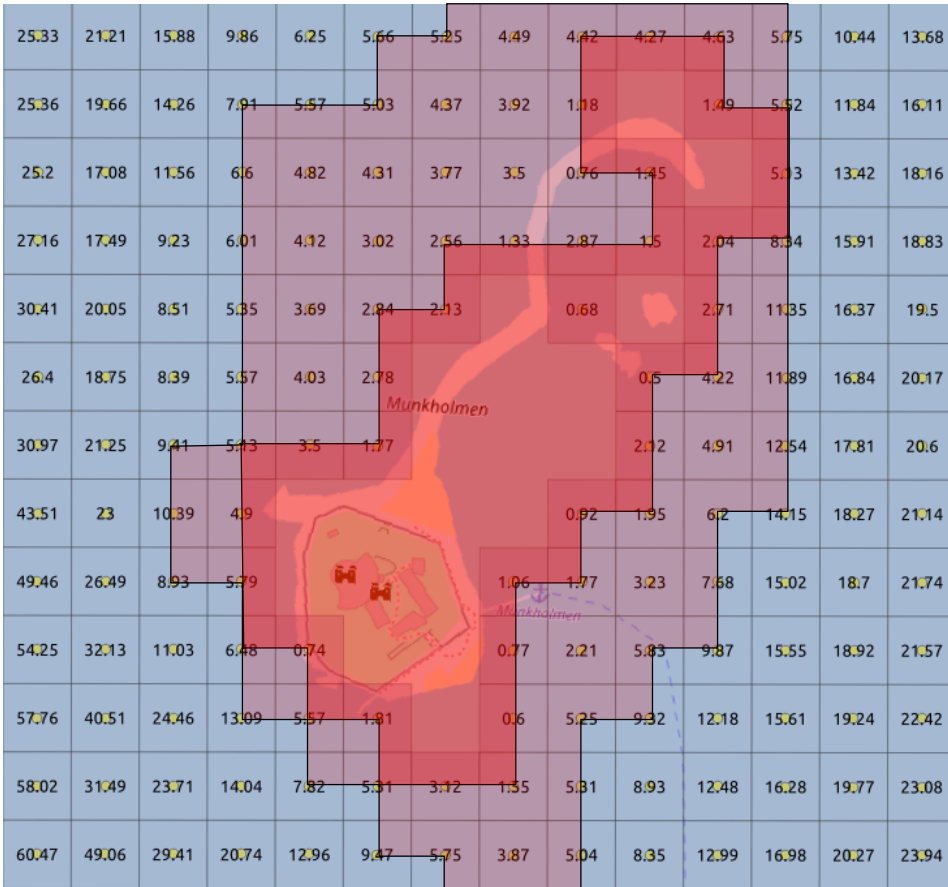
An example of what areas the grid implementation deem not navigable is shown in figure 5.8. This is the same area previously shown in figure 4.3b. From the figure, the need to put restrictions on depth is visible by areas on land being allowed. The outer area of the figure has a 5m depth limit, which results in a larger safety buffer around land. The figure shows depth soundings from the MAREANO dataset, in a 50m-50m grid.

## 5.4 Path Planning in DUNE

Online path planning will be required in multiple scenarios of autonomous collaborative fish tracking. An example is when one vessel has made a tag detection, the other vessels should stop searching, and instead congregate in desired positions in the vicinity of the vessel that made initial tag detection.

A benefit of having the two-dimensional grid, is that with small modifications, it can be used as a graph with all locations being in the navigable area, with a known minimum depth. This means that the algorithms developed for graph searching can be used to create valid paths between any two points in the navigable area. For this purpose, the class





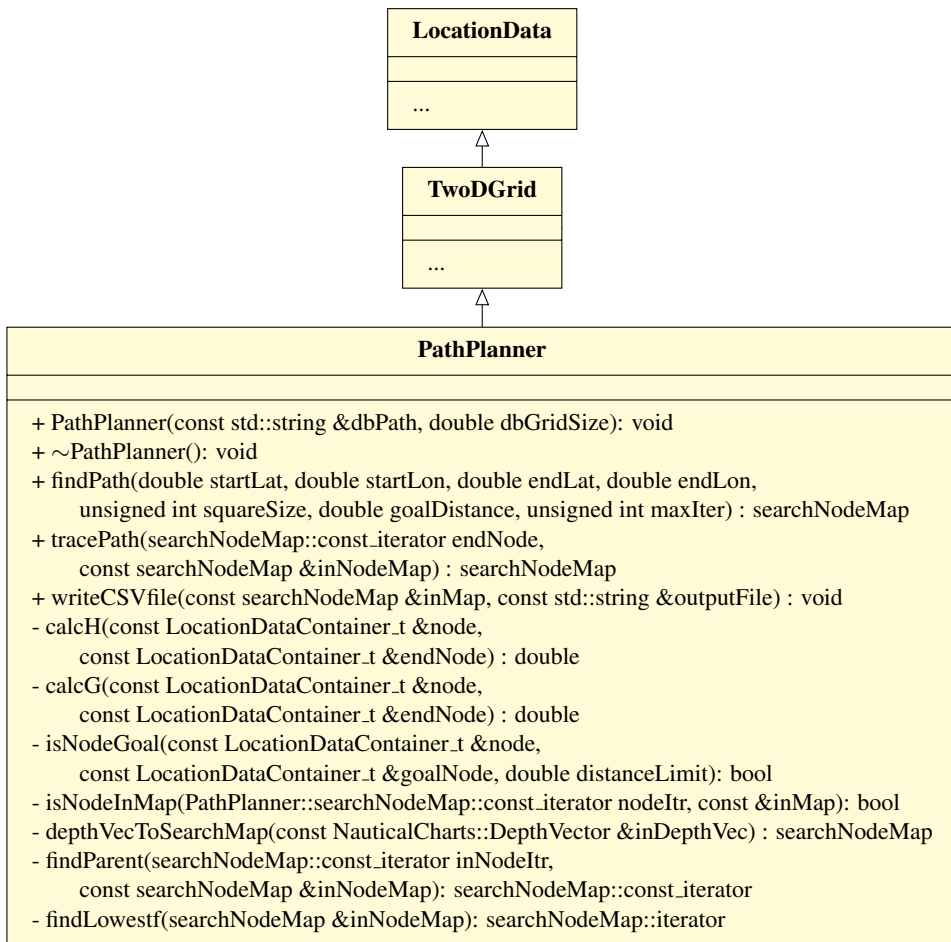
**Figure 5.8:** Area the Anti-Grounding implementation considers grounding. Inner area: No data. Outer area, depth below 5m.

`PathPlanner` has been created. A class diagram is available in figure 5.9. As can be seen in the diagram, the `TwoDGrid` class is inherited from, making all of its functions available.

`PathPlanner` implements the best-first search variant  $A^*$ , as is described in section 2.5.2, with the actual algorithm being run in the `findPath` function.

### Creating Edges

As the grid does not contain connections between the nodes needed to consider it a graph, the edges has to be defined in some way. Using the spatial relationship between them has been chosen in the implementation of the `PathPlanner` class. This works by considering every node in some parameter around the searched nodes as direct neighbours. This has been implemented with the `getSquare` function from the `TwoDGrid` class, but the `getWithinRadius` function should also work. The current node being expanded by the algorithm is then set as the parent for the returned neighbours.



**Figure 5.9:** Class diagram showing the PathPlanner C++ class developed for DUNE.

A benefit of this approach is that the edges does not need to be stored in the database, but are instead implicitly defined from the spatial relationship. The drawback is that the complete dataset has to be searched for every node. By using the indexed database, the efficiency of this operation is significantly improved.

### Storing Nodes and Algorithm Costs

To store the  $f$ ,  $g$  and  $parent$  data needed by the A\*-algorithm, the datatype `PathPlanner::searchData_t` is created. It is further used as the value in a STL map, with `LocationData::LocationDataContainer_t` as key. This container is defined with the name `PathPlanner::searchNodeMap` to make code reading easier. The exact definition is found in line 56-69 of code listing D.10.

## Heuristic Function

For the heuristic function `calcH`, the three functions mentioned in section 2.5.2 were implemented. As the vessel is not limited to moving in four or eight directions, the Manhattan distance and the diagonal distance was deemed unfitting. Instead, the Euclidean distance was used.

## Distance Calculations in the Map

Because the map has coordinates in WGS84, the formula for Euclidean distance mentioned in 2.5.2 could not be used directly. This is because the distance of one degree movement in latitude is not equal to the distance of moving one degree in longitude, and the distances are also varying depending on where on the earth one considers. To solve this, the WGS84 distance equation already implemented in DUNE was utilized, which accounts for the curvature in the earth. This function was also used to calculate the distance between a node and its neighbour in `calcG`.

## Considering Start and End Nodes not in the Grid

The start and end locations will often not be defined precisely on a node in the grid. For the start node, this solves itself by defining the point as a node and running the first iteration of the algorithm. This finds the closest node in the grid, solving the problem.

For the goal condition, the function `isNodeGoal` solves this by accepting nodes within a distance limit from the goal, instead of demanding that a certain node is met.

## Returning a Path

The `tracePath` function backtracks from the goal node to the start node by going to each nodes parent. When the start node is reached, the nodes in the path is returned, along with  $f(n)$ ,  $g(n)$  and parent for debugging.

## Remaining Functions

The remainder of functions in `pathPlanner` are either supporting functions for `findPath` or `tracePath`, or used for debugging, and is not further expanded upon here, other than referencing the reader to the code listing D.11.

## Algorithm Parameters for Tuning

Beside the normal tuning of the heuristics function in all A\*-implementations, the implemented approach gives rise to further customization's. By selecting the area shape and size around a node considered neighbours, and the distance limit in the goal condition, the operation is changed. For optimal results, setting the distance limit to slightly above the maximum distance expected between any position and a node. Quantified, this is  $gridDistance \cdot \sqrt{2}/2$ . The area considered neighbours should be set slightly above  $gridDistance \cdot \sqrt{2}$  to avoid any land areas traversed.

The reason for setting the value slightly above the calculated values are to account for rounding errors.

### 5.4.1 Results

This section shows the results returned from the path planner.

The laptop used for the benchmarks is a Dell Inspiron 7590 CN759021SR with an Intel Core i5-9300H (8MB Cache, up to 4.1 GHz, 4 cores) processor and 16GB of DDR4 2-channel RAM clocked at 2666MHz.

#### Laptop Compared to RPI4 Performance Example

To compare the performance of the laptop to the RPI4, the same paths were computed on the laptop as well as the RPI4. The grid used has a resolution of 50m·50m and the parameters used in the path planner was `SquareSize=100`, `GoalDistance=37` and heuristic scaling `D=1.01`. The dataset was created from the DEPARE ENC's as shown in figure 4.4 with a depth limit of 0.0m. This database contains 5'899'806 positions.

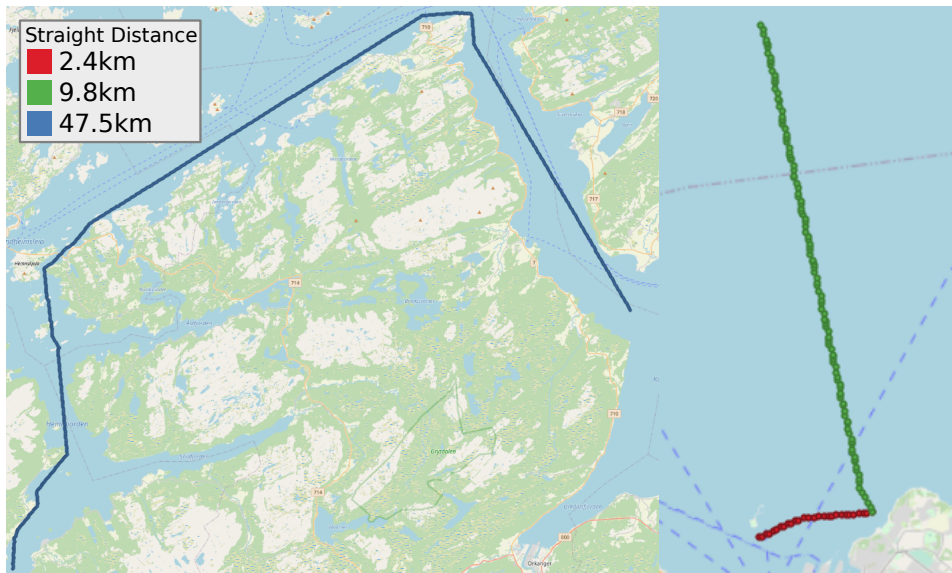


Figure 5.10: Benchmark paths found by path planner.

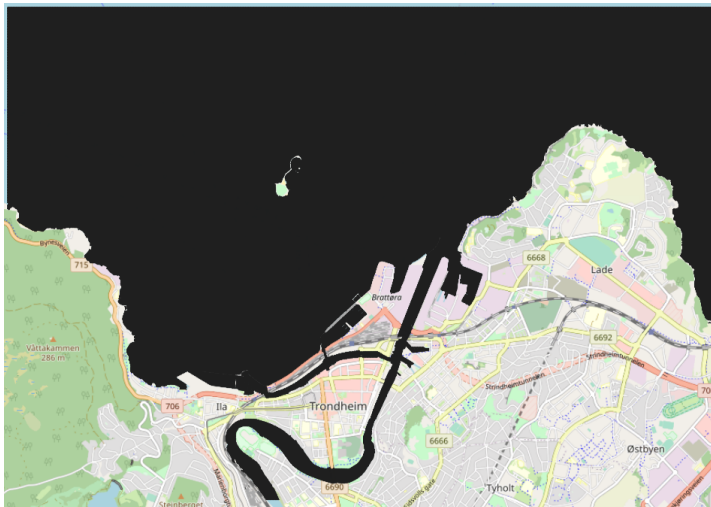
#### High-Resolution Performance Example

For the Otter, a higher grid resolution than 50m·50m is desirable. Therefore, a new 2m·2m grid was created from the 5H1620 DEPARE ENC's containing 5'342'661 positions. The new dataset is visualized in figure 5.11 where the black area contains the grid. The two paths created are shown in figure 5.12, while the details about each path is given in table

Path	Distance [m]	RPI4 [s]	Laptop [s]	Path [nodes]	Open [nodes]	Closed [nodes]
Red	2508	4.3	1.0	28	215	325
Green	10264	51.3	12.1	124	856	4085
Blue	81771	2985.5	599.2	854	2758	241516

**Table 5.2:** PathFinder benchmark results.

5.3. The algorithm parameters used was `SquareSize=4`, `GoalDistance=3` and heuristic scaling `D=1.01`.



**Figure 5.11:** A 2x2m grid made of 5H1620 ENC DEPARE object.

### Optimality versus Performance Example

To demonstrate the effect of different  $D$  values in the heuristic, the blue path from figure 5.2 was calculated with different values (with the other parameters as before being `SquareSize=100` and `GoalDistance=37`). The resulting paths are visualized in figure 5.13, and figure 5.14 shows the areas covered by the searched nodes by each value. Note that  $green \subset pink \subset brown \subset yellow$ . Details about the results are shown in table 5.4.

### 5.4.2 Discussion

The paths chosen for the RPI4 versus laptop comparisons were chosen to represent how the algorithm can be used on different lengths and amount of nodes. The blue path distance is far beyond what is expected to be a realistic scenario for the Otter, but is meant as a pushing the limits of the implementation. The green path is more in line with how a worst



**Figure 5.12:** Paths created by PathPlanner on high-resolution grid.

case path length for the Otter. The compute time it needed on the RPI4 is by the author considered serviceable for a global path planner.

Using a 50m grid is however not considered acceptable, so another grid with a 2m resolution was created along with paths for benchmarking. Initial tests ran on the laptop indicate that the algorithm is not feasible on the RPI4 for such resolutions, as shown in table 5.3. It was not considered necessary to run the benchmarks on the RPI4, because the laptops results were already worse than is considered acceptable.

To improve this, trying to make the algorithm more greedy is a possible solution. This would in most cases mean that a significantly lower number of nodes has to be expanded. As is seen in figure 5.13 the results visually look similar, and in figure 5.14 the dramatic reduction in the amount of areas of nodes can be seen. This is reflected in the execution times shown in table 5.4, which is more than halved by setting  $D=5$ . The downside is that the path becomes much longer than an optimal path would have been, with an increase of

Path	Distance [m]	Path [nodes]	Laptop [s]	Open [nodes]	Closed [nodes]
Cruising in Nidelva (Red)	5733.1	1576	627.0	2182	197845
Around the Pier (Green)	2146.9	511	599.2	2569	183449

**Table 5.3:** PathFinder benchmark results on 2x2 grid based on 5H1620 DEPARE object.

D	Laptop [s]	Distance [m]	Open [nodes]	Closed [nodes]
1.0	815.7	81740.3	2706	248143
1.2	513.3	82392.4	3982	144279
2.5	388.1	86582.6	4818	102367
5.0	349.1	89297.0	5164	92352

**Table 5.4:** Performance comparison with differently scaled heuristic function.

7557 meters in the worst case.

By comparing the results of  $D=1$  from table 5.4 with the blue path from 5.2, which was ran with  $D=1.01$  but otherwise had identical parameters, the tie-breaking effect can be seen. By only resulting in a slightly less optimal path (30m), a 217 second performance improvement is achieved. This is because that in such an open environment, multiple almost identical paths exists, with only one of them being the absolute most optimal. The tie-breaking avoids this by slightly skewing the algorithm towards being more greedy, resulting in fewer nodes being expanded<sup>5</sup>.

### 5.4.3 Further Work

The performance of the A\* implementation in `pathPlanner` has not been optimized, and was developed more as an proof of the usefulness of the created database, rather than a viable solution. But in the field of grid based path planning/motion planning, there is already several alternative algorithms that could be experimented with.

#### Accounting for the Kinematic constraints of the vessel

The kinematic constraints of the vehicle is not accounted for in the implemented A\* algorithm. An interesting alternative is the hybrid A\*, which accounts for a vessels non-holonomic constraints during the search. In [11], this algorithm is applied to a mobile robot with similar constraints as the Otter, with the resulting paths that are guaranteed to be navigable by the vehicle.

<sup>5</sup>The closed nodes have been expanded while the open nodes have only had their costs calculated, but not their neighbours explored.



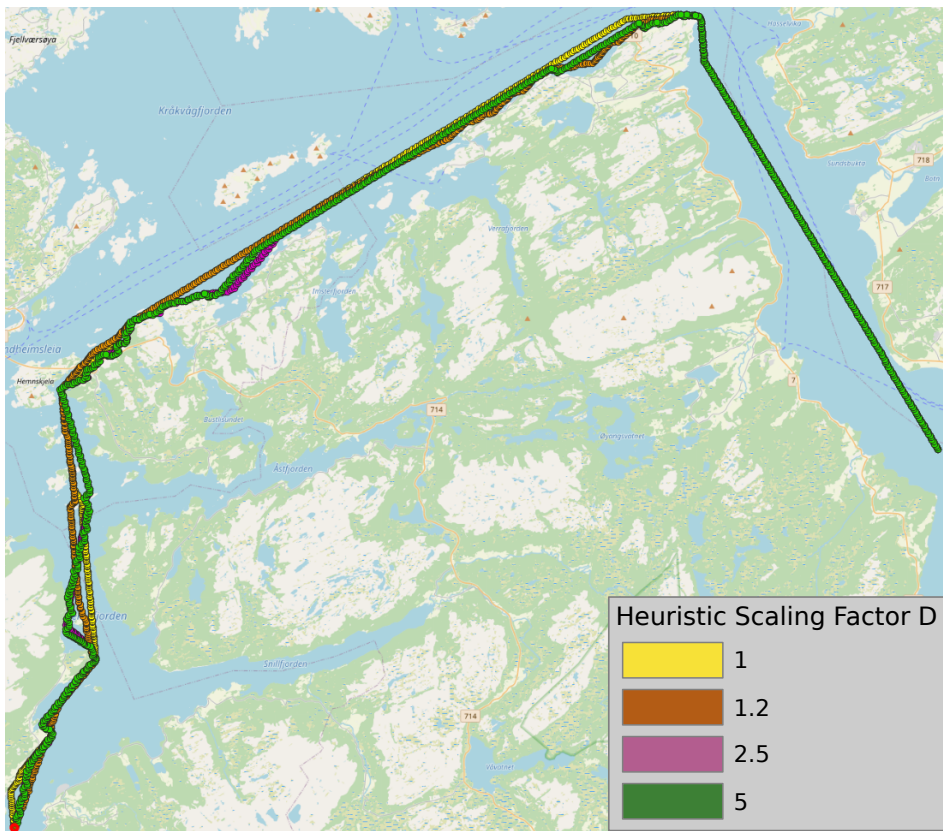


Figure 5.13: Paths showing the effect of increasing the heuristic scaling D.

### Changing to an All-Pairs Shortest Paths Algorithm

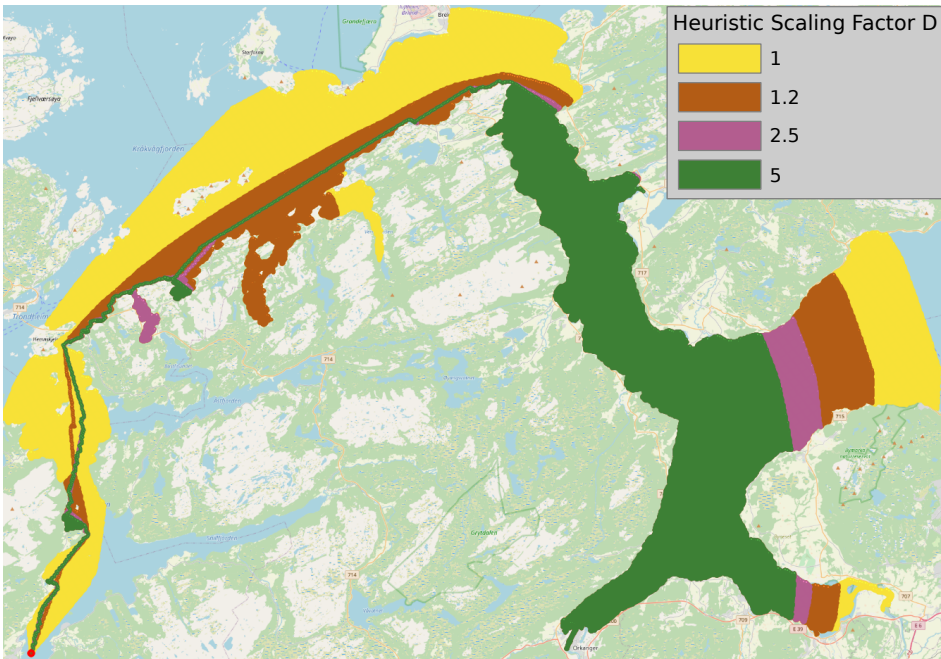
Instead of processing paths on-board, the two-dimensional grids could be pre-processed with an All-Pairs Shortest Paths Algorithm, like the Floyd-Warshall Algorithm (See [82], section 25.2). The result of the algorithm is a  $n^2$  predecessor matrix  $P_i$  that could be stored on the vessel. To find the shortest path to any location would then be as simple as looking up the next waypoint in the predecessor matrix.

For the 50m-50m grid based on all Trøndelag shown in figure 4.4a, there are 5899806 data entries. Each of these would have to store at least 3 fields at 32bit=4bytes, resulting in a predecessor matrix size of  $3 \cdot 5899806^2 \text{ entries} \cdot 4 \text{ Bytes/node} = 417.7 \text{ TB}^6$ . At the time of writing, adding that much storage to the Otter would not be feasible given the space and budget constrains of the Otter.

Reducing the scope of the 50m grid to the operational area of the NTNU Fish Otter (10Km · 10Km) results in a more feasible amount of nodes  $(10000m/50m)^2 = 40000 \text{ nodes}$

<sup>6</sup>This assumes that the node identifier is a 32-Bit address linking each node to a database where the latitude, longitude is stored. In reality,  $5899806 \cdot 8 \text{ Bytes}$  extra space would be needed to store the actual data.





**Figure 5.14:** Effect on searched area by increasing the heuristic scaling D.

that requires  $40000^2 \cdot 4 \cdot 3 = 19.2\text{GB}$  of storage. For the Otter, which is to operate in coastal areas, a 50m grid resolution is most likely too small, with 5m being a more realistic minimum. Doing the same calculations for a 5m grid results in a dataset requiring 6.4TB. This is closer to being realizable, but probably still not a practical solution.

### Sampling Based Approaches

The sampling based RRT\* and PRM\* introduced in [83] takes a wholly different approach by pseudo-randomly exploring points in a continuous space. Points that pass an acceptance test and compared to other nearby nodes have a better cost value are then added to the grid. For watercrafts, the acceptance test would be avoiding land, shallow areas and other hazardous points of interest. From the random points, the RRT\* algorithm then connects the nodes to a tree in a manner that makes it asymptotically optimal.

The PRM\* algorithm is based much on the same concept, but does not include actually finding the path. Instead it creates a graph of the configuration space formed as a multiple clusters along with some cost metric. This graph can then be searched by other graph traversal algorithms such as the A\*.

Many of the functions needed to implement these algorithms have already been provided in the library of classes presented in this thesis, such as land and grounding checking for obstacle detection, and `checkTransect` for considering if an edge can safely be added between two of the randomly generated vertices. The algorithms should also be able to operate directly on the polygons extracted from the S-57, which would reduce the stor-

age requirements significantly. These algorithms should therefore be further explored for potential use in the NTNU Fish Otters.

### Performance gains from data structures

The most time-demanding operations in the A\* `while` loop are searching in either the database, or in the open and closed data-structures. In the developed implementation, the C++ STL container `STD::map` is used, which performs insertion, erasing and search on key in  $O(\log n)$  complexity. Having defined the key as the location, searching for specific location therefore becomes logarithmic in complexity. The downside is that extracting the vertex with minimum `fScore`, which are not sorted on, results in a complexity of  $O(n)$ .

Optimally, both location and `fScore` should be optimized for. A solution is to use a container with multiple indexes. An example of such a container is the `Boost.MultiIndex`<sup>7</sup> provided by the open-source Boost C++ libraries.

An entirely different approach is to store all data in the database, with multiple indexes. This is possible in SQLite through temporary tables, which are stored in memory. By the use of foreign keys<sup>8</sup> instead of locations, this could be implemented in a memory efficient way. This would mean using SQLite as a data container, which according to its homepage is considered one of the appropriate uses[84].

### PathPlanner On Varying-Resolution Database

By using a dataset with varying resolution, two benefits to the algorithm would be realized: For the same amount of storage/points, areas with obstructions would have larger resolution, while in deep areas, a larger distance between vertices could be achieved. This would then result in a reduced number of nodes needing to be expanded.

Directly applying the written algorithm on the dataset shown in figure 4.5 does return a result, but there are some problems with the paths found: The area around a vertex considered neighbours have to be set to work with the largest resolution used, because a possibility of no nodes being in the area. The benefit of the additional resolution in shallow areas then disappear, because a large amount of vertices are just skipped, missing potential problematic areas.

Deciding a goal condition is also problematic, Because the distance has to be set according to the largest resolution.

A potential solution for overcoming these problems, would be to count neighbours when a node is expanded, and then change parameters if more neighbours than expected are returned. The development of such an algorithm is left for future works.

## 5.5 Situational Awareness in Neptus

As a closing remark on the use of the created databases, a Neptus plugin has been created by Alberto Dallolio, which uses the database created in chapter 4 to increase the level

---

<sup>7</sup>[https://www.boost.org/doc/libs/1\\_67\\_0/libs/multi\\_index/doc/tutorial/basics.html](https://www.boost.org/doc/libs/1_67_0/libs/multi_index/doc/tutorial/basics.html)

<sup>8</sup>The database equivalent of a memory pointer.

of situational awareness for the operator. As Neptus is written in Java, the code of the situational awareness classes is rewritten for that language, with some additional queries added for use when visualizing. This creates a visual interface for checking transects for groundings, and also shows the POI, depth contours, and depth measurements.



# Software Integration

## 6.1 Hydrophone Integration

To detect the transmitted signals from acoustic tags in water, a hydrophone is mounted on the Otter, as shown in figure 3.2. In the figure, the Thelma Biotel TBR700RT hydrophone is mounted, which originally was planned to be used on the NTNU Fish Otter. During the period when this thesis work was completed, Thelma Biotel released a new hydrophone called the *TB Live*. A comparison of the two are given in table 6.1. For use on the Otter, there are many benefits to be gained by doing this upgrade, such as the built in PPS support and the smaller footprint<sup>1</sup>. Because of this, the upgrade was made.

Feature	TBR700RT	TB Live
Battery operation	Yes	No
External power supply	Yes	Yes
Outer diameter [mm]	75	42
Length [mm]	230	80
Time synchronization	RS-485	RS-485 time stamps and PPS
Timestamp Resolution [ms]	1	1
Message sending	No	Shared Open Network protocol

**Table 6.1:** TB Live comparison to TBR700RT.

To perform the upgrade, no software changes was needed. This was confirmed on missions with the NTNU AutoNaut, which has had the TB Live integrated into its hull, and reused the task created for the Otter. The hardware changes needed were also minimal, and are included in figure 3.4<sup>2</sup>. In the current firmware of the TB Live, the PPS related functionality is yet to be implemented and so the PPS connection to the RPI4 is still in-

<sup>1</sup>Reducing the drag of the hydrophone while moving through the water

<sup>2</sup>The hardware changes relating to the TB Live have not actually been made yet due to the author not having access to the hardware.

cluded in the diagram. If a logic level-shifter is needed for the PPS signal going to the TB Live is still unclear, so it has been omitted from the figure.

### 6.1.1 Assessing the Implemented Hydrophone Synchronization

Before changing the hydrophone model to the better suited TB Live, time-synchronization through the Thelma Biotel RS-485 protocol was developed and tested. Having correct time-synchronization on the hydrophones is paramount, because they are used in the estimators TDOA measurements.

The results of the experiment showed that the synchronization differed from the SLIM PCB developed at NTNU DEC, which was used as a known good implementation. This section describes the experiment, while section 6.1.2 describes improvements made to the task in trying to mitigate the identified problems.

#### The Developed Time-Synchronization Method

To synchronize and discipline the internal clock of the Thelma Biotel hydrophones, a RS-485 based protocol is used that sends either a complete Unix timestamp messages like (+) 123456789X where X is Luhn's verification number, or a clock discipline message (+) that rounds the clock to the nearest 10.5 seconds. The timestamp of tag readings are given with a 1ms accuracy.

In order for all the vessels in the system to have synchronized and disciplined clocks, the timestamps and PPS from the GPS system is used. To enable PPS support in Linux, the kernel has been compiled with the options `CONFIG_PPS` and `CONFIG_NTP_PPS` as is described in the Wiki<sup>3</sup>. This lets a PPS device be created, which is accessed by a monitor in DUNE that disciplines the clock when it drifts beyond some threshold.

#### Experiment setup

The experiment was performed using a single tag, and two TBR700RT hydrophones, one connected to the NTNU Fish Otter control box, and the other to a SLIM PCB. A picture showing the hardware setup is given in figure 6.1. The trolley with the equipment was brought outside for optimal GPS reception on a day with clear skies. After GPS fix was achieved on both the Otter system and the SLIM system, the acoustic tag was activated and transmissions logged.

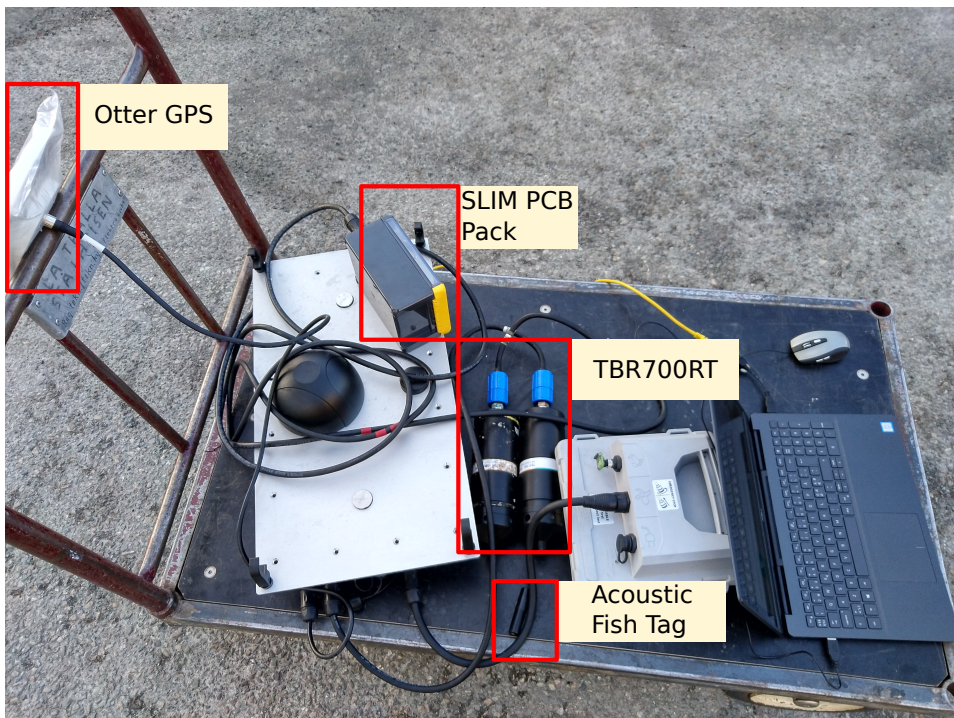
The result from the two hydrophones can then be compared by joining the tag registrations on sensor data measured by the tag. Having joined up the tags, the timestamps can be compared to spot deviations.

#### Experiment Results

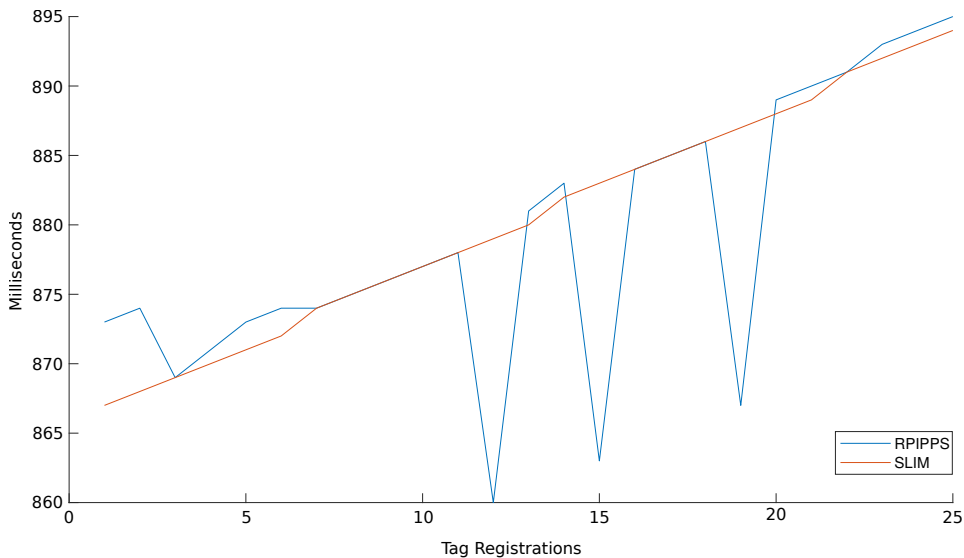
The logged tag registrations of the experiment is shown in figure 6.2, where the registrations made by the NTNU Otter hardware is called RPIPPS, and the registrations made with the SLIM PCB is called SLIM. As can be seen, there are three registrations where

---

<sup>3</sup>[http://otter.itk.ntnu.no/doku.php?id=enable\\_pps\\_support\\_raspian](http://otter.itk.ntnu.no/doku.php?id=enable_pps_support_raspian), Accessed 07/07/2020



**Figure 6.1:** The setup used in the PPS experiment.



**Figure 6.2:** Comparison between SLIM and RPIPPS synchronization of TBR700RT (389ms had to be added to RPIPPS to account for constant offset).

the RPIPPS lags behind by about 15ms. In addition to this, a constant offset of 389ms had to be applied to the RPIPPS match the registrations. The expected result is a steady drift caused by temperature changes in the internal clock of the fish tag, which is exactly what is seen in the SLIM results.

## Discussion

In saline water, the propagation speed of acoustic waves is approximately  $1500\text{m/s}$  [25]. The large deviations in the RPIPPS disciplined system would therefore result in an error of  $1500\text{m/s} \cdot 15 \cdot 10^{-3}\text{s} = 22.5\text{m}$ , which is unsuited for use in acoustic fish tracking systems where position accuracy is desired. The constant offset is also a problem that must be solved.

Possible reasons for the variation shown in RPIPPS is that the Linux kernel is not a real-time operating system, making stable periodic tasks difficult. To solve this, the suggested actions given below may help:

- Applying the `CONFIG_PREEMPT_RT` to the kernel.
- Adding a micro-kernel below the Linux kernel.
- Switching to a real-time operating system (RTOS). According to [44], DUNE runs on the QNX RTOS.
- Isolate a dedicated CPU kernel for the purpose with the `isolcpus` kernel option.

Due to switching to the TB Live with dedicated PPS input, none of these solutions was explored further.



Assuming this solves the problem, the constant offset must nevertheless be mitigated because it will still be used for the TB Live. By examining the documentation from Thelma Biotel, this turned out to be an oversight from the developer (the author) who had failed to notice that the messages had to be sent at timestamps ending with a zero, basically at 10 second segments (with an added 500ms delay).

### Conclusion

The RPIPPS clock disciplining is not working sufficiently well to be used in the system. As this will solve itself by upgrading to the TB Live, no further action will be taken. As for the timestamp synchronization that does not work either, this should be fixed by changes to the hydrophone DUNE task.

### 6.1.2 Improvements to the DUNE Hydrophone Task

Due to the errors identified in the experiment, a redesign of the hydrophone task was performed to fix the synchronizations. Instead of being designed as a periodically running task with a separate thread monitoring the RS-485 serial communication, it was changed to a continuously running task with timers that periodically sent timestamp and disciplining messages (still having a separate reading thread).

While not having repeated the experiment to confirm it working<sup>4</sup>, printing the serial timestamp messages from the redesigned task showed the desired behavior.

In anticipation of the use of the tag registrations in an estimator, the most recent GPS position is added to the tags ensuing IMC message. At a later point, this functionality should also include a buffer of the ten most recent GPS positions, as it may need to account for some small propagation delays and such.

With the documentation for the TB Live, there came a range of commands that could be used to configure the device, such as receiving channel and tag transmission protocols. To make configuring the device simple, sending these commands were made available in task parameters. These can then be set in the DUNE configuration file, which also means that changes can be made remotely through Neptus without needing to recompile DUNE.

## 6.2 Preparation for Multi-Vessel Operations in the LSTS Toolchain

The Fish Otter project will use at least three Otters to gather information and perform multilateration on. In preparation for operations with multiple vessels, some changes had to be made.

To have multiple Otters in the same IMC network, new IMC messages were made. New IMC bindings with the updated addresses were created for Java, and added to IMCProxy and Neptus. New C++ bindings were added to DUNE as well.

For DUNE, one configuration file for each vessel was created. In order to not have many almost identical configuration files to maintain, a common configuration file was

---

<sup>4</sup>Due to the author not having access to the hardware.

created that included the hardware on all Otters. This file was then included in the configuration files to be used on the Otters.

The filenames for the configurations, as well as the IMC addresses are given in table 6.2. The address segment where the IMC addresses are placed, is for non-LSTS ASV.

<b>Configuration Filename</b>	<b>IMC Address</b>
ntnu-otter-01.ini	0x2810
ntnu-otter-02.ini	0x2811
ntnu-otter-03.ini	0x2812
ntnu-otter-04.ini	0x2813

**Table 6.2:** The IMC addresses of the NTNU Fish Otters.

# Communications Design

## 7.1 Computer Networking

To establish communication between the operator and the Otters, two wireless systems have been implemented: One using the Ubiquiti AirMax 5GHz wireless system, and the other using a 4G LTE modem. The reasoning behind this, is that the 4G modem can be used as primary communication, while the 5GHz is available for scenarios in areas where 4G LTE is not available.

### 7.1.1 5GHz Wireless

AirMax wireless equipment from Ubiquiti provides communication for the Otter and land station. On the Otter, a Ubiquiti Bullet AC IP67 has been used, because of its IP rating.

On the land station, two access points are available. One is a 120° sector antenna connected to a Ubiquiti Rocket AC, which was configured and used in the specialization project [1]. During sea trials with clear skies and no waves, a solid connection was maintained in ranges up to 700 meters, while intermittent connection was achieved as far as 1000 meters from the communication mast. The other device, an Ubiquiti PowerBeam Gen2 dish antenna/access point was configured as a part of this thesis work, with the hope being that it will provide better range than the sector, at the expense of covering a smaller area.

Power to the AirMax equipment used on the land station is, like on the Otter, provided through passive 24V Power over Ethernet (PoE), with injectors from Ubiquiti. This injector is added between the computer and the access point, which has the benefit that only one cable per device is needed on the mast.

Managing AirMax equipment is done through either a web based interface, or a command line interface accessed with SSH. Access to these interfaces can be had through wired Ethernet or via a dedicated WiFi management network the device enables. This separate configuration network is needed because the AirMax protocol can not be interfaced by standard WiFi devices.

Multiple topologies are available for AirMax networks. For the work done in this thesis, the devices at the base station have been set up as point-to-multipoint devices, which acts like access points (APs) in the networks, while the Otters are points that all communicate through the AP.

### 7.1.2 4G LTE Wireless

From the design phase of the NTNU Fish Otters, mobile broadband was a planned feature. To meet that end, the MIMO antenna *Teoglas Limited MA741.A.BI.001* with IP67 rating was installed on the top of the control box, with its two SMA connectors on the inside of the box. Originally, a USB modem from *Huawei* had been procured to be used along with the antenna, but low confidence in whether it could withstand rough treatment or not led to it being discarded.

A modem that could connect through a wired Ethernet cable was proposed as a better solution. To achieve this, either a 4G router or a separate router in combination with a 4G modem had to be found, since the RPI4B<sup>1</sup> is equipped with only one wired Ethernet connection.

Because of the space constraints in the control box, and that one less component means one less component to manage, the 4G router solution was deemed the better fit. A component search commenced, trying to fulfill the following criterions:

- Appropriate size: Being able to fit in the control box.
- Solid mounting options, preferably din-rail.
- Two or more Ethernet ports, more is a plus.
- SMA external antenna connectors, or adaptable to SMA.
- VPN support preferable.
- Affordability.



**Figure 7.1:** The Teltonika RUT950 4G router used in the fish Otter (Picture from Teltonika).

In the end, these alternatives were considered:

- Westermo MRD-415
- Teltonika RUT240 or RUT950
- Welotech TK812L
- Advantech BB-SR303 with Ethernet power delivery and 5 ports.

---

<sup>1</sup>Raspberry Pi 4 Model B

From the alternatives, the Teltonika RUT950 (see fig. 7.1) was selected, because it fitted all criteria and came at a reasonable price point. It provides routing between four Ethernet ports, a built-in WiFi access point (not used for now) and a 4G LTE Cat 4 rated cellular modem with dual-sim support.

Powering the RUT950 can be done either through a 2x2 Molex Micro-Fit 3.0 connector, or through passive PoE. As the PoE injector in the control box only has one output, the Molex connector was wired up along with the two SMA 50Ω antenna connectors for the cellular modem. How the connections were made in the control box is shown in figure 3.4.

A benefit of the RUT950 is that the firmware is based on OpenWRT, a *highly extensible GNU/Linux distribution for embedded devices, typically wireless routers*<sup>2</sup>, giving ample possibilities for customisations in addition to command line access through SSH.

### RUT950 Software Configuration

Configuring the router for use in the Otter has been done through the web interface, though access through SSH has also been experimented with for debugging. The following changes have been made compared to default:

- OpenVPN Client configuration, see figure C.2.
- Port Forwarding, see figure C.1.
- Disabled integrated WiFi AP.
- Root/admin user setup: Changed password for the *root* user through SSH, as well as the *admin* used in web interface.
- LAN setup: Changed the IP subnet. Different subnets are assigned to each RUT950, with values as shown in table 7.1.

Network	Base Address	Netmask
VPN	10.8.0.1	255.255.255.0
RUT950 Otter1 Local	10.10.0.1	255.255.255.0
RUT950 Otter2 Local	10.11.0.1	255.255.255.0
RUT950 Otter3 Local	10.12.0.1	255.255.255.0
RUT950 Otter4 Local	10.13.0.1	255.255.255.0

**Table 7.1:** Planned subnetworks for the NTNU Fish Otter System.

### 7.1.3 VPN Setup and Deployment

To simplify the communication between the networked devices on the Otter ASVs and the console through the Internet, a VPN solution has been installed. The benefits of this is that all devices are on the same network, making it possible to reach all devices connected to the RUT950<sup>3</sup>. The topology of the VPN, is given in figure 7.2.

<sup>2</sup>Description from their web page: [openwrt.org/about](http://openwrt.org/about)

<sup>3</sup>An alternative would be to report the IP addresses of the routers to a DNS service.

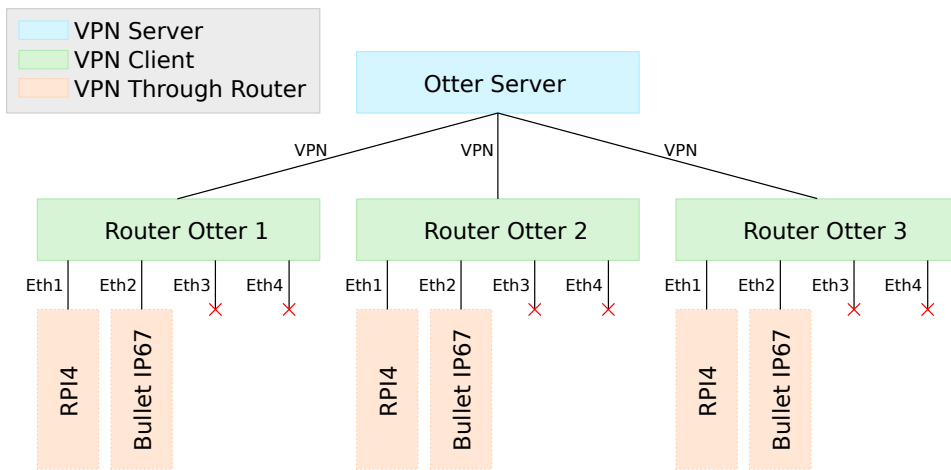


Figure 7.2: VPN topology.

To realize the VPN, the open-sourced OpenVPN solution was chosen (a short introduction to VPNs and OpenVPN is given in section A.1.1). The deployment included installing and configuring a server located at the otter.itk.ntnu.no server. The RUT950 has both server and client functionality support from the factory, but only the client functionality is used in this thesis work.

The configuration process that was used is described in [85], which includes setting up the a public cryptography key infrastructure with `easy-rsa` along with the basic server/client setup for an ISO/OSI 3rd layer tunnel VPN. For the RUT950 client, the configuration shown in figure C.2 was made. The setup on the operators computer can be performed according to [85], but as the author was using Ubuntu 18.04, the client that was included by default in the desktop environment was used.

#### 7.1.4 IMCProxy

To use the IMCProxy in the NTNU Fish Otter System, the proxy server has been set to run at the Otter server described in 3.4 an instance of the client is ran at the vessel when needed, as shown in figure 2.6.

#### 7.1.5 Discussion

The original purpose of introducing the VPN, was to have all IMC networks operate directly over the VPN, and not through the IMCProxy<sup>4</sup>. This would have required configuring the VPN to operate on the data link layer of the ISO/OSI stack<sup>5</sup>. Configuring OpenVPN to do this required configuring a virtual network bridge (TAP<sup>6</sup> device) on the NTNU Otter server. As mentioned in section 3.4, the server is managed by the IT-services

<sup>4</sup>Which is not dependent on the VPN to function.

<sup>5</sup>As described in appendix A.1.1

<sup>6</sup>Terminal Access Point

at NTNU, and is only reachable through remote access. During the setup process of the network bridge, communication with the server was lost. This complicated the configuration of the VPN to the point where it was decided to simply settle on letting the VPN operate in the network layer with a virtual TUN<sup>7</sup> device. As the UDP IMC broadcast used for discovering new devices is stopped at the routers, only using the VPN was not a solution any longer, and thus the overhead of the IMCProxy had to be introduced.

It is generally considered a drawback to increase the amount of communication in robotic systems because it tends to reduce the robustness. Because the TUN VPN is operating on the network layer, IP packages are sent, which comes with a smaller overhead than the Ethernet frames used in the intended link layer TAP VPN<sup>8</sup>. The comparison of network overhead between the implemented IMCProxy+TUN VPN system versus only having the TAP VPN has not been performed, so the author can not conclude the question about whether the compromise was beneficial or detrimental to the networks performance.

As one could remove the VPN, and still have a functioning system, this should also be further investigated. For this, the importance of loosing the other benefits of having the VPN, such as simplifying connecting to remote devices, better control over network traffic to and from the vessel and increased security due to the implemented encryption should be considered. Increasing the security is a potential gain, because the collected data may be used by adversaries to either sway the results or exploit the results as described in [86].

The topology of the AirMax network should also be further investigated. A suggested option is to configure one of the Otters as the access point and the others communicating through it. This would allow for faster and more efficient inter-vessel communication. The caveat in choosing such a topology, is that the vessels could not operate at great distances from one another. This is not expected to be a problem in the intended use-cases the Otter system.

Regardless of the network design which ends up being implemented in the deployed system, the ability to fail gracefully at communication loss will have to be implemented. A possibility is having the vessel performing a maneuver that is considered safe, like loitering or traveling back to the deployment area.

## 7.2 Website for sharing and analysing data

It is a goal of the NTNU Otter ASVs to be able to execute missions that will gather data that is relevant to other scientists. Having every interested party learn to use Neptus can not be expected, and it's also not desirable to let others access the IMC network during missions.

Instead, the design of a system that makes the telemetry available in a password protected online environment has been developed and deployed.

---

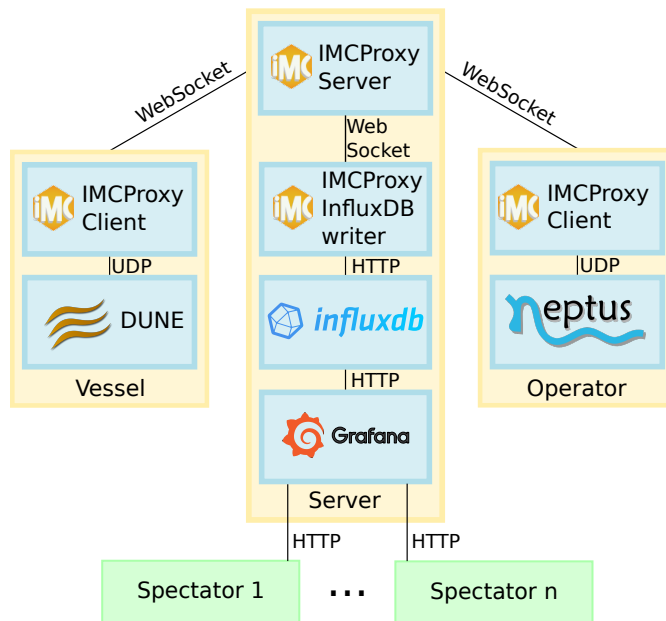
<sup>7</sup>Short for Tunnel.

<sup>8</sup>Because the Ethernet frames would still need to encapsulate the IP Packet.

## 7.2.1 System Design

The design of the system is based around Grafana<sup>9</sup> and InfluxDB<sup>10</sup>, along with a custom ingester developed for reading IMC messages from the IMCProxy already being used, and writing the data to the InfluxDB. By using Grafana as a visual querying solution for the InfluxDB, both data monitoring during the mission, as well as sharing, analyzing and exporting the data after the mission is made possible.

An overview of this system is shown in figure 7.3, which is simply an extension of figure 2.6. Extracting the data in the server has the benefit of needing no additional data from the vessels, as the data passes through the IMCProxy server anyways.



**Figure 7.3:** IMCProxy network with spectator functionality.

## 7.2.2 The IMC to InfluxDB Ingester Design

To ingest the IMC messages to the database, a simple Java application has been developed. It is derived from the IMCProxy client<sup>11</sup>, but instead of relaying messages, it ingest them to a DB. The written code is available in appendix D.2, with the parts of interest being the function `httpPostImcToInfluxDB` that creates a HTTP POST request to the Otter InfluxDB server that correctly formats the URL and parameters according to [63]. This function then uses `imcToInfluxLineProtocol` to convert the IMC message fields to a

<sup>9</sup>Introduced in section 2.6.2.

<sup>10</sup>Introduced in section 2.7.6.

<sup>11</sup>Original code available at: <https://github.com/LSTS/imcproxy>



line protocol string<sup>12</sup>.

The `imcToInfluxLineProtocol` function also filters the IMC messages by type, so only certain messages are ingested to the DB. By basing the conversion on the `Map<String, Object>` representation of the IMC message, all IMC messages<sup>13</sup> are supported without needing to specialize for each one. Adding new message types for ingestion therefore is as simple as adding another case in a switch statement, as shown in line 62 to 71 of code listing D.1.

In anticipation of using the data in map panels in Grafana, IMC message attributes containing coordinates are converted to degrees before ingesting.

To optimize sorting by message source and entity in the DB, these have been stored as tags, while all other attributes are stored as fields.

If in some future design update, the IMCProxy is removed from the system, the parts of this implementation could be reused in a Neptus plugin. Because both are written in Java, and both use the same IMC bindings, the changes needed are minimal.

### 7.2.3 Result

The resulting system is presented in the two following screenshots: The dashboard shown in figure 7.4 visualizes data from the InfluxDB in suited panels, such as a track map that shows the path traversed by the vessel in a selected time period along with graphs of other important telemetry. A practical feature for analysis shown in the figure, is that the data is joined on timestamps. By hovering the cursor over the RPM metric, the corresponding value for `setThrustersActuation` is shown, as well as a blue dot on the map signifying where the vessel made the measurement.

Another dashboard was made with the purpose of showing tag registrations on a map, as shown in figure 7.5. This map panel also supports spatial aggregating, which means that if you zoom out, it will reduce multiple registrations to a single point with color indicating the amount of tags in a location. This panel, as all panels in Grafana does custom time ranges, which makes it possible to zoom the graphs for closer inspection.

### 7.2.4 Discussion

Because the system is placed on the same physical server as the IMCProxy server, the delay between when the IMC message is sent from the vessel and it being available in the InfluxDB through Grafana should be on par with or better than using Neptus for telemetry<sup>14</sup>. This means that the delay between the message being sent and it being available in the visualization is only limited by the performance of the cellular communications network, the public Internet infrastructure and how fast Grafana is allowed to query the data.

The choice of using Grafana is motivated by getting most of the needed functionality for minimal additional work, and also getting free functionality updates as updates become available. If at some point in the future it is decided that the visualizations available does

---

<sup>12</sup>Described in section 2.7.6

<sup>13</sup>Some IMC message types are nested, support for these is not verified

<sup>14</sup>This holds only when IMCProxy is used, and not when vessel and operator is on the same network without needing the IMCProxy.

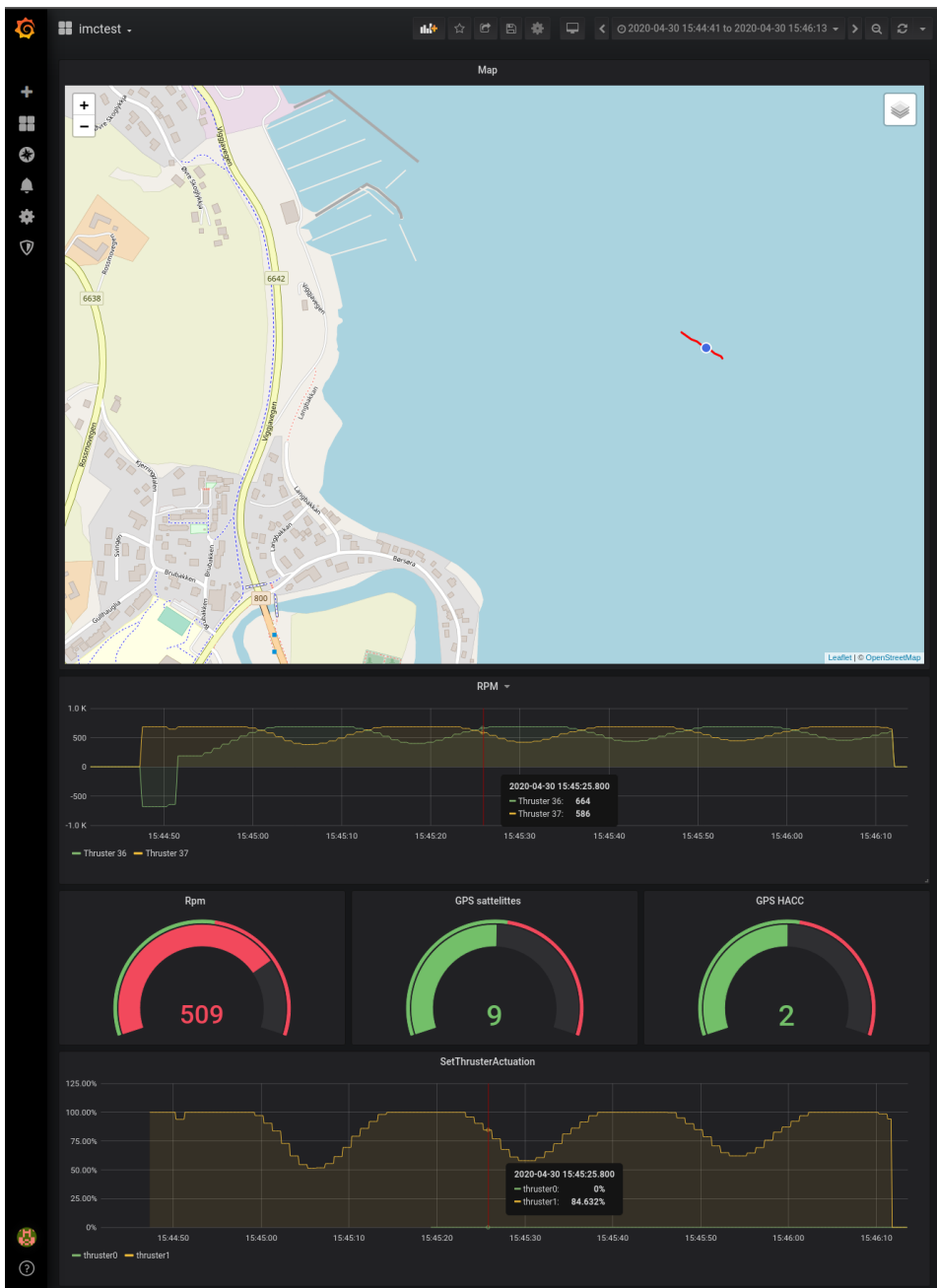
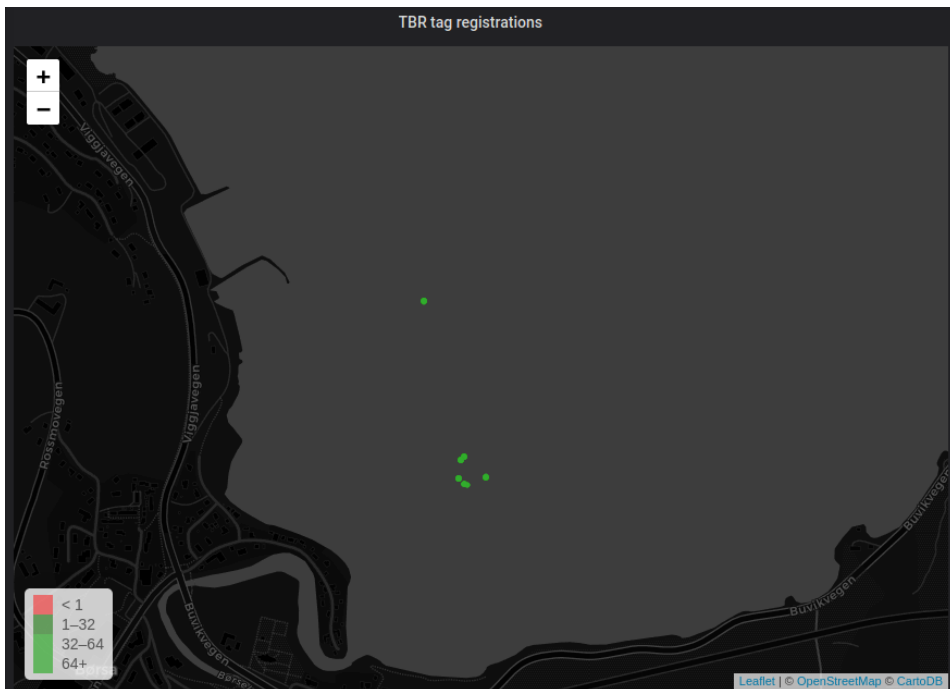


Figure 7.4: An example screenshot of the Grafana Dashboard.



**Figure 7.5:** A Grafana Dashboard showing fish tag detections at the locations the registrations were made.

not fit the projects purposes, the open-sourced nature of the project makes it possible to develop new ones.

A benefit of using Grafana, is that it can pull data from various sources, making extending it to fetch publicly available weather data an interesting prospect. Another possible extension imagined, is that a more computationally expensive estimator setup can be implemented on the server, while the estimator running in the vessels only need compute with enough accuracy to hold the vessels in formation around the tracked acoustic tag.

## 7.3 Summary

With the addition of the RUT950 and the Dish AirMax access point, as well as the upgrade to the TB Live hydrophone, all planned devices for a single NTNU Fish Otter has been installed and integrated. To document how they all communicate, figure 7.6 has been created. For a more detailed diagram over all devices of the NTNU Fish Otter, see figure 3.4 where the discussed modifications has also been included.

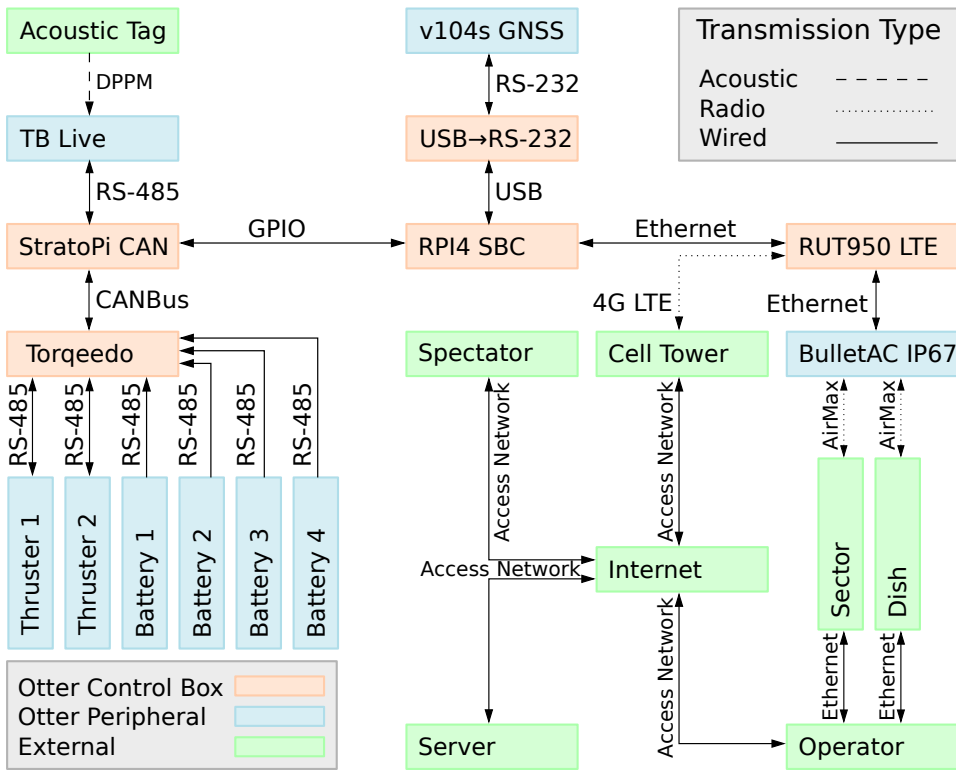


Figure 7.6: Communication overview after thesis work completion.

## Conclusion and Further Work

This masters thesis has presented the NTNU Fish Otter System in its current state, and the work that has been performed for it to achieve its ultimate goal; to become an autonomous multi-vessel system that collaboratively can search for and track acoustic fish tags. The contributions made towards this goal is: Designing a system for providing a priori spatial data on the vessel, implementing a global path planner, implementing an anti-grounding monitor, implementing the network infrastructure, implementing an online interface for visualizing vessel telemetry, and upgrading the hydrophone hardware and software.

The spatial a priori data system is based around S-57 formatted ENC's and publicly available depth soundings. A database is designed to store this data, along with a software library to make it accessible in DUNE, the on-board middleware. Use of this library is then demonstrated by the implementations of an anti-grounding supervisor and a global path planner for the vessel.

The network infrastructure is completed with the integration of a cellular modem. Along with the already integrated 5GHz network, this will let the operator communicate with the vessels to command and monitor its progress. To simplify device management, a VPN is configured on the server for the system.

The servers function has also been extended with a system that stores and visualizes received telemetry messages through an online interface. It is envisioned that this system will be used to both assist the operator, and share data with collaborating researchers.

Finally, the previously written software implementation is given an assessment, and found to have serious flaws. Updates is made accordingly, but the result has yet to be put to the test. The implementation has also been made ready for use with an updated hydrophone model, the TB Live from Thelma Biotel

### 8.1 Further Work

Throughout the sections of this thesis, multiple suggestions on how to improve and extend the implemented solutions have been made. If the author was to select just one to focus on, it would be further researching how changing to the SpatialLite database would impact

---

storage efficiency. This would need to go along with changes done to both the path planner and the anti-grounding functionality.

Options on how IMC messages are used through the Internet should also be explored.

The most important task to be performed in the project is performing more sea trials in order to find appropriate parameter values to use in the path controller and the course and speed controller. Along with this, the now completed design of the control box has to be duplicated in order to get all Fish Otters operational. Once this has been completed, the formation control and fish tag estimator described in [21] should be implemented, and used as the base for further developments.

# Bibliography

- [1] Nikolai Lauvås. *TTK4550 - Design and development of a robotic fish tracking vehicle*. NTNU DEC, 2020.
- [2] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. 3rd global. Pearson Education, 2016.
- [3] Gard Paulsen. *Alltid rabiatt (Jens Glad Balchen og den kybernetiske tenkemåten)*. Fagbokforlaget, 2019. ISBN: 9788245025460.
- [4] J. E. Manley. “Unmanned surface vehicles, 15 years of development”. In: *OCEANS 2008*. 2008, pp. 1–4.
- [5] Zhixiang Liu, Youmin Zhang, Xiang Yu, et al. “Unmanned surface vehicles: An overview of developments and challenges”. In: *Annual Reviews in Control* 41 (May 2016). DOI: 10.1016/j.arcontrol.2016.04.018.
- [6] H. Zheng, R. R. Negenborn, and G. Lodewijks. “Survey of approaches for improving the intelligence of marine Surface Vehicles”. In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. 2013, pp. 1217–1223.
- [7] Hanguen Kim, Donghoon Kim, Jae-Uk Shin, et al. “Angular rate-constrained path planning algorithm for unmanned surface vehicles”. In: *Ocean Engineering* 84 (July 2014), pp. 37–44. DOI: 10.1016/j.oceaneng.2014.03.034.
- [8] Kenny Daniel, Alex Nash, Sven Koenig, et al. “Theta\*: Any-Angle Path Planning on Grids”. In: *J. Artif. Intell. Res. (JAIR)* 39 (Jan. 2014). DOI: 10.1613/jair.2994.
- [9] Yanlong Wang, Xu Liang, Baoan Li, et al. “Research and Implementation of Global Path Planning for Unmanned Surface Vehicle Based on Electronic Chart”. In: *Recent Developments in Mechatronics and Intelligent Robotics*. Ed. by Feng Qiao, Srikanta Patnaik, and John Wang. Cham: Springer International Publishing, 2018, pp. 534–539. ISBN: 978-3-319-65978-7.
- [10] Chenguang Liu, Qingzhou Mao, Xiumin Chu, et al. “An Improved A-Star Algorithm Considering Water Current, Traffic Separation and Berthing for Vessel Path Planning”. In: *Applied Sciences* (Mar. 2019). DOI: <https://doi.org/10.3390/app9061057>.

- 
- [11] J. Petereit, T. Emter, C. W. Frey, et al. “Application of Hybrid A\* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments”. In: *ROBOTIK 2012; 7th German Conference on Robotics*. May 2012, pp. 1–6.
- [12] Hanguen Kim, Donghoon Kim, Jae-Uk Shin, et al. “Angular rate-constrained path planning algorithm for unmanned surface vehicles”. In: *Ocean Engineering* 84 (July 2014), pp. 37–44. DOI: 10.1016/j.oceaneng.2014.03.034.
- [13] *INTERNATIONAL HYDROGRAPHIC ORGANIZATION IHO TRANSFER STANDARD for DIGITAL HYDROGRAPHIC DATA: Publication S-57*. 3.1. International Hydrographic Bureau MONACO, Nov. 2000.
- [14] Janusz Magaj Marcin Maka. “Data extraction from an electronic S-57 standard chart for navigational decision systems”. In: *Akademia Morska w Szczecinie* 30 (2012), pp. 83–87.
- [15] Ole Sivert Otterholm. *Extracting Mapped Hazards from Electronic Navigational Charts for ASV Collision Avoidance*. Master’s thesis – NTNU, Department of Engineering Cybernetics, 2019.
- [16] S. Reed and V. E. Schmidt. “Providing Nautical Chart awareness to autonomous surface vessel operations”. In: *OCEANS 2016 MTS/IEEE Monterey*. 2016, pp. 1–8.
- [17] A. Aguiary, J. Almeida, M. Bayaty, et al. “Cooperative Autonomous Marine Vehicle motion control in the scope of the EU GREX Project: Theory and Practice”. In: *OCEANS 2009-EUROPE*. 2009, pp. 1–10.
- [18] T. M. Grothues, J. Dobarro, and J. Eiler. “Collecting, interpreting, and merging fish telemetry data from an AUV: Remote sensing from an already remote platform”. In: *IEEE/OES Autonomous Underwater Vehicles* 1695-1702 (2010). URL: <https://ieeexplore.ieee.org/document/5779658>.
- [19] S. S. Løvskar. *Positioning of periodic acoustic emitters using an omnidirectional hydrophone on an AUV platform*. Master’s thesis – NTNU, Department of Engineering Cybernetics, 2017.
- [20] D. Bhadauria, V. Isler, A. Studenski, et al. “A Robotic Sensor Network for monitoring carp in Minnesota lakes”. In: *2010 IEEE International Conference on Robotics and Automation*. May 2010, pp. 3837–3842. DOI: 10.1109/ROBOT.2010.5509499.
- [21] A. Zolich, T. A. Johansen, J. A. Alfredsen, et al. “A formation of unmanned vehicles for tracking of an acoustic fish-tag”. In: *OCEANS 2017 - Anchorage*. 2017, pp. 1–6.
- [22] R. P. Jain, A. P. Aguiar, J. B. de Sousa, et al. “Localization of an Acoustic Fish-Tag using the Time-of-Arrival Measurements: Preliminary results using eXogenous Kalman Filter”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1695–1702.
- [23] C. Forney, E. Manii, M. Farris, et al. “Tracking of a tagged leopard shark with an AUV: Sensor calibration and state estimation”. In: *2012 IEEE International Conference on Robotics and Automation* (2012). URL: <https://ieeexplore.ieee.org/document/6224991>.
-



- 
- [24] D. Shinzaki, C. Gage, S. Tang, et al. “A multi-AUV system for cooperative tracking and following of leopard sharks”. In: *2013 IEEE International Conference on Robotics and Automation*. May 2013, pp. 4153–4158. DOI: 10.1109/ICRA.2013.6631163. URL: <https://ieeexplore.ieee.org/document/6631163>.
- [25] Inge Mohus and Bård Holand. *Fish telemetry manual*. Report NO. STF48 A83040. SINTEF Automatic Control. Dec. 1983.
- [26] Nigel E. Hussey, Steven T. Kessel, Kim Aarestrup, et al. “Aquatic animal telemetry: A panoramic window into the underwater world”. In: *Science 12 Jun 2015: Vol. 348, Issue 6240, 1255642* (2015). URL: <https://doi.org/10.1126/science.1255642>.
- [27] Claude Leroy, Stephen Robinson, and M. Goldsmith. “A new equation for the accurate calculation of sound speed in all oceans”. In: *The Journal of the Acoustical Society of America* 124 (Dec. 2008), pp. 2774–82. DOI: 10.1121/1.2988296.
- [28] Thelma Biotel. *About*. Accessed: 2020-06-23. URL: <https://www.thelmabiotel.com/about/>.
- [29] Steven Cooke, Scott Hinch, Martin Wikelski, et al. “Biotelemetry: A mechanistic approach to ecology”. In: *Trends in ecology & evolution* 19 (July 2004), pp. 334–43. DOI: 10.1016/j.tree.2004.04.003.
- [30] Douglas Pincock and Sam Johnson. “Acoustic Telemetry Review in Telemetry Techniques.” In: *A User Guide for Fisheries Research*. American Fisheries Society, 2012. ISBN: 9781934874264.
- [31] Jan Reubens, Pieterjan Verhelst, Inge van der Knaap, et al. “The need for aquatic tracking networks: the Permanent Belgian Acoustic Receiver Network”. In: *Animal Biotelemetry* 7.1 (Jan. 2019), p. 2. ISSN: 2050-3385. DOI: 10.1186/s40317-019-0164-8. URL: <https://doi.org/10.1186/s40317-019-0164-8>.
- [32] Jijie Zhu. “Calculation of geometric dilution of precision”. In: *IEEE Transactions on Aerospace and Electronic Systems* 28.3 (1992), pp. 893–895.
- [33] Shuqiang Xue and Yuanxi Yang. “Positioning Configurations with the Lowest GDOP and their Classification”. In: *Journal of Geodesy* 89 (Oct. 2014). DOI: 10.1007/s00190-014-0760-6.
- [34] Tor A. Johansen and Thor I. Fossen. “The eXogenous Kalman Filter (XKF)”. In: *International Journal of Control* 90.2 (2017), pp. 161–167. DOI: 10.1080/00207179.2016.1172390. eprint: <https://doi.org/10.1080/00207179.2016.1172390>. URL: <https://doi.org/10.1080/00207179.2016.1172390>.
- [35] Gergely Magyar, Peter Sincak, and Zoltán Krizsán. “Comparison Study of Robotic Middleware for Robotic Applications”. In: *Advances in Intelligent Systems and Computing* 316 (Jan. 2015), pp. 121–128. DOI: 10.1007/978-3-319-10783-7\_13.
-

- 
- [36] Ayssam Elkady and Tarek Sobh. “Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography”. In: *Journal of Robotics* 2012 (May 2012). DOI: 10.1155/2012/959013.
- [37] Sigurd Andreas Holsen. “DUNE: Unified Navigation Environment for the REMUS 100 AUV - Implementation, Simulator Development, and Field Experiments”. 2015. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2350792>.
- [38] Sølve Dahlin Sæter. “COLREGS compliant Collision Avoidance System for a Wave and Solar Powered USV”. 2018. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2562570>.
- [39] José Pinto, Pedro Calado, José Braga, et al. “Implementation of a control architecture for networked vehicle systems”. In: *IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles (NGCUV'2012)* 270180 (2012). URL: <https://goo.gl/swCB3D>.
- [40] R. Martins, P.S. Dias, E. Marques, et al. “IMC: A communication protocol for networked vehicles and sensors”. In: *OCEANS 2009 - EUROPE*. 2009, pp. 1–6. DOI: 10.1109/OCEANSE.2009.5278245.
- [41] Paulo Sousa Dias, Sergio Loureiro Fraga, Rui MF Gomes, et al. “NEPTUS- A framework to support multiple vehicle operation”. In: *Proc Oceans MTS/IEEE Conference*. Brest, France, 2005, pp. 963–968.
- [42] *Neptus Operator Manual*. Accessed: 2020-05-08. URL: <https://www.lsts.pt/neptus/manual/trunk/>.
- [43] *Public LSTS Neptus repository*. Accessed: 2020-05-08. URL: <https://github.com/LSTS/neptus>.
- [44] J. Pinto, P. S. Dias, R. Martins, et al. “The LSTS toolchain for networked vehicle systems”. In: *MTS/IEEE Oceans*. IEEE. 2013, pp. 1–9.
- [45] *LSTS GitHub DUNE repository*. Accessed: 2020-05-02. URL: <https://github.com/LSTS/DUNE/>.
- [46] Zhiping Lu and Yunying QuShubo Qiao. *Geodesy: Introduction to Geodetic Datum and Geodetic Systems*. Springer, Berlin, Heidelberg, 2014. ISBN: 978-3-642-41245-5.
- [47] Øystein B. Dick. *Gauss-Krüger projeksjon*. Accessed: 2020-06-25. URL: [https://snl.no/Gauss-Kr%C3%BCger\\_projeksjon](https://snl.no/Gauss-Kr%C3%BCger_projeksjon).
- [48] Knut Grunderud, Haakon Rassmussen, Steinar Nilsen, et al. *GIS: The geographic language of our age*. 2nd ed. Tapir academic press, 2017. ISBN: 9788245020113.
- [49] NMAP/Kartverket. *Kartprojeksjonar*. Accessed: 2020-06-25. URL: <https://www.kartverket.no/Posisjonstjenester/Kartprojeksjoner/>.
- [50] NMAP/Kartverket. *Referanserammer for Noreg*. Accessed: 2020-06-25. URL: <https://www.kartverket.no/en/Posisjonstjenester/bruke-referanserammer/Referanserammer-for-Noreg/>.
-

- 
- [51] Geodata. *Datumtransformasjon mellom WGS84 og ETRS89 (Euref89) i ArcGIS for Desktop*. Accessed: 2020-07-10. URL: <https://geodata.no/guider/datumtransformasjon-mellom-wgs84-og-etrs89-euref89-i-arcgis-for-desktop>.
- [52] IHO. *About the IHO*. Accessed: 2020-06-25. URL: <https://iho.int/en/about-the-iho>.
- [53] IHO. *HSSC*. Accessed: 2020-06-25. URL: <https://iho.int/en/hssc>.
- [54] NMAP/Kartverket. *Miscellaneous Notices to Mariners: 5 Electronic Navigational Charts (ENC)*. Accessed: 2020-06-25. URL: <https://www.kartverket.no/en/EFS/Miscellaneous-Notices-to-Mariners/5-Electronic-Navigational-Charts-ENC/>.
- [55] *S-57 Appendix A IHO Object Catalogue*. 3.1. International Hydrographic Bureau MONACO, Nov. 2000.
- [56] Amit Patel. *Amit's A\* Pages*. Accessed: 2020-06-29. URL: <http://theory.stanford.edu/~amitp/GameProgramming/>.
- [57] A. van Deursen and E. Visser. "The Reengineering Wiki". In: *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*. 2002, pp. 217–220.
- [58] Grafana. *The analytics platform for all your metrics*. Accessed: 2020-07-08. URL: <https://grafana.com/grafana/>.
- [59] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, Global Edition*. 7th ed. Pearson, Aug. 2016. ISBN: 1292097612.
- [60] SQLite. *About SQLite*. Accessed: 2020-06-25. URL: <https://www.sqlite.org/about.html>.
- [61] Influxdata. *Time series database (TSDB) explained*. Accessed: 2020-07-14. URL: <https://www.influxdata.com/time-series-database/>.
- [62] InfluxData InfluxQL Documentation. *Influx Query Language (InfluxQL)*. Accessed: 2020-04-30. URL: [https://docs.influxdata.com/influxdb/v1.8/query\\_language/](https://docs.influxdata.com/influxdb/v1.8/query_language/).
- [63] InfluxData InfluxDB API Documentation. *Write data with the InfluxDB API*. Accessed: 2020-05-02. URL: [https://docs.influxdata.com/influxdb/v1.8/guides/write\\_data/](https://docs.influxdata.com/influxdb/v1.8/guides/write_data/).
- [64] InfluxData InfluxDB Line Protocol Documentation. *InfluxDB line protocol reference*. Accessed: 2020-05-02. URL: [https://docs.influxdata.com/influxdb/v1.8/write\\_protocols/line\\_protocol\\_reference/](https://docs.influxdata.com/influxdb/v1.8/write_protocols/line_protocol_reference/).
- [65] August Ekanger. "Developing an Autonomous Tracking System for the Atlantic Salmon". NTNU, Department of Marine Technology, 2018.
- [66] Eva Kristiansen. "Strategies for search and detection of acoustic transmitters using unmanned surface vehicle". NTNU, Department of Engineering Cybernetics, 2018.

- 
- [67] Jenny Aurora Frøland Steindal. “Gain Scheduled Controller with Bumpless Transfer for an Unmanned Surface Vehicle”. NTNU, Department of Marine Technology, 2018.
- [68] S. Buadu, I. Schjøberg, and T. Mo-Bjørkelund. “Mission planner for multiple AUVs: Verification procedures combining simulations and experiments”. In: *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*. 2018, pp. 1–6.
- [69] Athanasios Kapoutsis, Savvas Chatzichristofis, and Elias Kosmatopoulos. “DARP: Divide Areas Algorithm for Optimal Multi-Robot Coverage Path Planning”. In: *Journal of Intelligent & Robotic Systems* 86 (Jan. 2017). DOI: 10.1007/s10846-016-0461-x.
- [70] Torqeedo. *Ultralight 403 A*. Accessed: 2020-05-02. URL: <https://www.torqeedo.com/en/products/outboards/ultralight/ultralight-403-a/1405-00.html>.
- [71] Torqeedo. *Battery 915 Wh Ultralight 403*. Accessed: 2020-05-02. URL: <https://www.torqeedo.com/en/products/accessories/spare-batteries/battery-915-wh-ultralight-403/1417-00.html>.
- [72] *Raspberry Pi 4 Model B Product Brief*. Raspberry Pi Trading Ltd. June 2019.
- [73] *Strato Pi CAN User Guide*. Rev. 008. Sfera Labs S.r.l. Aug. 2019.
- [74] Kristin Y. Pettersen. “Underactuated Marine Control Systems”. In: *Encyclopedia of Systems and Control*. Ed. by John Baillieul and Tariq Samad. London: Springer London, 2015, pp. 1499–1503. ISBN: 978-1-4471-5058-9. DOI: 10.1007/978-1-4471-5058-9\_125. URL: [https://doi.org/10.1007/978-1-4471-5058-9\\_125](https://doi.org/10.1007/978-1-4471-5058-9_125).
- [75] Walter Caharija, Mauro Candeloro, Kristin Y. Pettersen, et al. “Relative Velocity Control and Integral LOS for Path Following of Underactuated Surface Vessels”. In: *IFAC Proceedings Volumes 45.27 (2012)*. 9th IFAC Conference on Manoeuvring and Control of Marine Craft, pp. 380–385. ISSN: 1474-6670. DOI: <https://doi.org/10.3182/20120919-3-IT-2046.00065>. URL: <http://www.sciencedirect.com/science/article/pii/S1474667016312599>.
- [76] *Dybdedata - rådata*. Accessed: 2020-05-07. URL: <https://kartkatalog.geonorge.no/metadata/dybdedata-raadata/2fe7b56c-334d-4660-ac50-6fcf973a0f70>.
- [77] *Dybdedata - terrengmodeller 50 meters grid landsdekkende*. Accessed: 2020-05-07. URL: <https://kartkatalog.geonorge.no/metadata/dybdedata-terrengmodeller-50-meters-grid-landsdekkende/bbd687d0-d34f-4d95-9e60-27e330e0f76e>.
- [78] *Dybdedata terrengmodeller DTM WCS*. Accessed: 2020-05-07. URL: <https://kartkatalog.geonorge.no/metadata/dybdedata-terrengmodeller-dtm-wcs/8896c751-f3e0-4e4c-86e2-7579cealf111>.
- [79] Geonorge. *Innsjødatabase*. Accessed: 2020-06-23. URL: <https://kartkatalog.geonorge.no/metadata/innsjoedatabase/823b8639-9a49-41bf-8571-3608435eb149>.
-

- 
- [80] *V104s GPS Compass User Guide A3*. Hemisphere GNSS, 2017.
- [81] Kartverket. *Vannstandsni a*. Accessed: 2020-07-14. URL: <https://www.kartverket.no/sehavniva/data-pa-se-havniva/viktige-vannstandsniiva/>.
- [82] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, et al. *Introduction to Algorithms*. 3rd. The MIT Press, 2009.
- [83] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761. eprint: <https://doi.org/10.1177/0278364911406761>. URL: <https://doi.org/10.1177/0278364911406761>.
- [84] SQLite. *Appropriate Uses For SQLite*. Accessed: 2020-07-14. URL: <https://www.sqlite.org/whentouse.html#container>.
- [85] Ubuntu 18.04 Documentation. *OpenVPN*. Accessed: 2020-03-02. URL: <https://help.ubuntu.com/18.04/serverguide/openvpn.html>.
- [86] Steven J. Cooke, Vivian M. Nguyen, Steven T. Kessel, et al. “Troubling issues at the frontier of animal tracking for conservation and management”. In: *Conservation Biology* 31.5 (2017), pp. 1205–1207. DOI: 10.1111/cobi.12895. eprint: <https://conbio.onlinelibrary.wiley.com/doi/pdf/10.1111/cobi.12895>. URL: <https://conbio.onlinelibrary.wiley.com/doi/abs/10.1111/cobi.12895>.
- [87] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. 6th International. Pearson, 2013. ISBN: 9780273768968.

---

# Additional Background and Theory

## A.1 Networked Remote Device Access

This section gives a brief description of computer networking technologies that have been used in order to gain access to and control networked devices without physical access. In the NTNU Fish Otter Project, this is relevant both for the server, and for the vessels during missions.

### A.1.1 VPN

A virtual private network creates an IPsec<sup>1</sup> tunnel to allow the creation of private sub-networks over public or shared networks like the Internet. By encapsulating, and also often encrypting the network traffic between multiple points, data integrity and privacy is accomplished. Only allowing data packets coming through the VPN tunnel also gives control over who can send packets to a receiver.

#### VPN OSI/ISO Layer

Network protocols are often divided into a hierarchy of seven layers, according to the ISO/OSI model for communication systems [87]. VPNs can operate either in the link<sup>2</sup> layer or in the network<sup>3</sup> layer. The operation done at the different layers by the VPN server, is called routing for the network layer or bridging for the link layer.

A routing VPN causes less bandwidth overhead than a bridging VPN, due to an extra Ethernet header being added to data units in bridged mode, and additional link layer protocols going through the VPN tunnel<sup>4</sup>, versus staying on the local network. In some cases, where tunneling all link layer frames are desired, bridging VPNs are still needed.

---

<sup>1</sup>The IP (Internet Protocol) security protocol. Described in RFC4301 and RFC6071, RFCs being the formal standard documents published by the Internet Engineering Task Force (IETF).

<sup>2</sup>The second layer in the ISO/OSI model.

<sup>3</sup>The third layer in the ISO/OSI model.

<sup>4</sup>Such as ARP requests.

---

## OpenVPN

OpenVPN is an open-source VPN solution released under the GPL licence. To authenticate users, it supports the use of pre-shared secret keys, certificates or username/password combinations. In this report, pre-shared keys has been used, created with the help of the `easy-rsa` utility.

Configuring both the server and the clients is done through configuration files, with support for both routed and bridged operating level.

### A.1.2 Network Device Management

Physical access to the Otters while deployed is not practical, and even when physical access is available, only the console computer has peripherals that a human operator can interface with. The only way to manage the hardware and software of these devices are through network interfaces. Secure Shell (SSH) and secure copy (SCP) are used extensively in this work, so a brief introduction to the commands are given in this section.

#### Secure Shell

The secure shell command (SSH) is used to provide remote access to the host computers command line, and also other secure services<sup>5</sup>. The basic command syntax is `ssh [remote_user]@[remote_host]`, while a plethora of options can be added, like changing remote port with `-p [remote_port]`.

#### Secure Copy

The secure copy command (SCP) is a part of SSH that is used to transfer files between two computers. The command syntax is: `scp [from] [to]`, where `[from]` and `to` is either a path in the local file system, or a remote path formatted like `[remote_user]@[remote_host]:[remote_path]`. This is very similar to the UNIX command for local copying, `cp`, but with options that are relevant for communication with remote hosts, like `-P [remote_port]` for selecting which remote port to connect to.

## A.2 Summary of TTK4550 Project Work on the NTNU Fish Otter

During the second half of 2019, the author was assigned the task of performing a system integration for the NTNU Fish Otter ASV. At that point, none of the software integration had been performed, and although most of the hardware was mounted in the control box, a lot of wiring had to be made. As the system never had been powered before in this configuration, this involved also ensuring that power distribution of the vessel was sorted out. For the PoE powered Ubiquiti Bullet IP67, a 12V to 24V converter had to be added to avoid a voltage drop in a PTC that had been specified according to its current use at 24v.

---

<sup>5</sup>The Debian wiki: <https://wiki.debian.org/SSH>



---

Originally, the SLIM synchronization board was to be used to synchronize the hydrophone clock. The author was encouraged to investigate the possibility of doing the synchronization on the RPI4. The functionality for this was implemented, but not verified.

The software integration which was performed began with choosing an operating system to use on the RPI4. The choice fell on the Debian based Raspian Lite distribution, which was purposely made for the RPI SBCs. With an OS in place, an configuration was made for DUNE, along with newly developed tasks to control the hydrophone, the Torqeedo interface card and the other features on the StratoPi daughter board. To support communicating with the Torqeedo Interface card, CAN bus support also had to be implemented for DUNE to interact with the SocketCAN drivers in the Linux Kernel.

By implementing software and connecting the hardware one component at a time, all current components were successfully integrated. This was validated through two sea trials where the first demonstrated basic vessel functions, and the second validated that acoustic tags were detected and sent to the operator. The clock synchronization was implemented and used, but could not be verified due to not having a known good implementation to compare against.

The further work section of the report stated these tasks:

1. Verify PPS synchronization for system.
2. Make the TBR700RT DUNE task send periodic sync messages.
3. Make a custom PCB for the logic-level-shifter used on the PPS signal.
4. Hardware and software support for 4G communication.
5. Setting up and testing the Ubiquiti PowerBeam AC Gen2.
6. Getting IMC addresses for the fish Otters in the official IMC repository.
7. Properly tune the PID controllers of the course and speed controller, as well as the parameters for the ILOS path controller.

All but task 3, 6 and 7 is addressed in this thesis. Task 3 and 6 is not considered necessary any longer, but 7 remains highly relevant.

### **A.3 S-57 ENC Object Quick Reference**

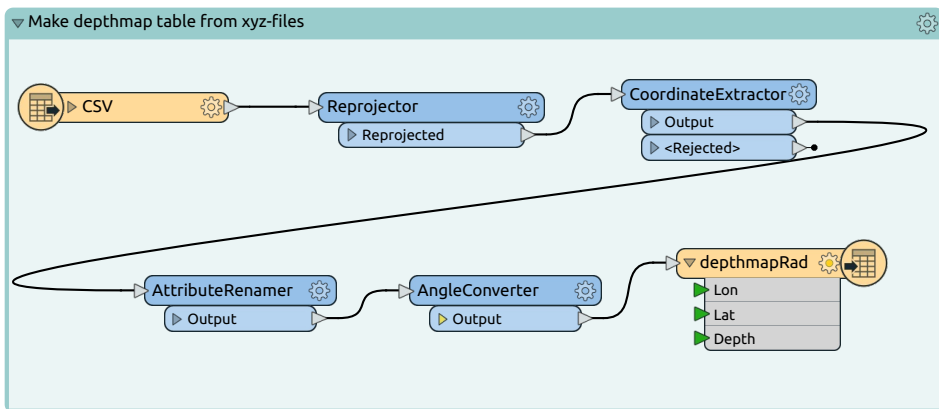
<b>Acronym</b>	<b>Object</b>	<b>Description</b>
BCNISD	Isolated Danger Beacon	A beacon erected on an isolated danger of limited extent, which has navigable water all around it.
BCNLAT	Lateral Beacon	Used to indicate the port or starboard hand side of the route to be followed.
BCNSPP	Beacon, special purpose /general	
BOYCAR	Cardinal Buoy	Used in conjunction with the compass to indicate where the mariner may find the best navigable water.
BOYINB	Installation Buoy	Used for loading tankers with gas or oil.
BOYISD	Isolated Danger Buoy	A buoy moored on or above an isolated danger of limited extent, which has navigable water all around it.
BOYLAT	Lateral Buoy	Used to indicate the port or starboard hand side of the route to be followed.
BOYSAW	Safe Water Buoy	Used to indicate that there is navigable water around the mark.
BOYSPP	Special Purpose Buoy	Primarily used to indicate an area or feature, the nature of which is apparent from reference to a chart, Sailing Directions or Notices to Mariners.
COALNE	Coastline	The line where shore and water meet
DEPARE	Depth Area	Area of water within a defined range of values.
LIGHTS	Light	A luminous or lighted aid to navigation.
OBSTRN	Obstruction	In marine navigation, anything that hinders or prevents movement.
PILPNT	Pile	long heavy timber or section of steel, wood, concrete, etc.. forced into the earth which may serve as a support,
UWTROC	Underwater /awash rock	A concreted mass of stony material or coral which dries, is awash or is below the water surface. The ruined remains of a stranded or sunken vessel which has been rendered useless.
WRECKS	Wreck	

**Table A.1:** Overview of the S-57 objects used in this report (Source: [55]).

# Appendix B

## FME Workbenches

For a description of the nodes in the workbench, see table 2.3.



**Figure B.1:** FME flow for reading .xyz files in utm33 projection, and reprojecting them to WGS84, then saving in database.

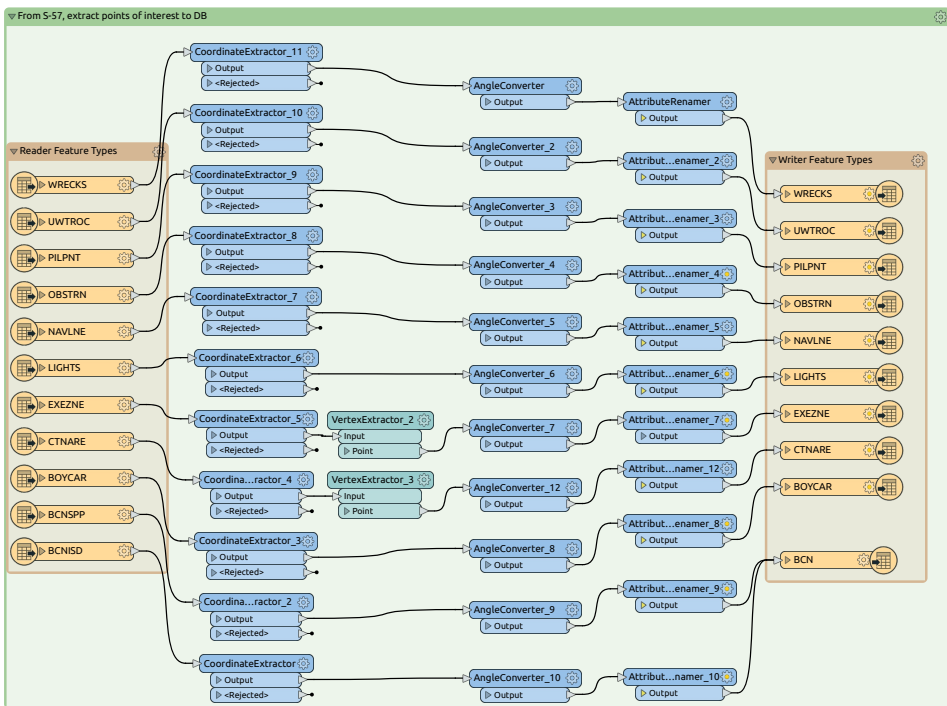


Figure B.2: FME Workbench for extracting points of interest from POI.

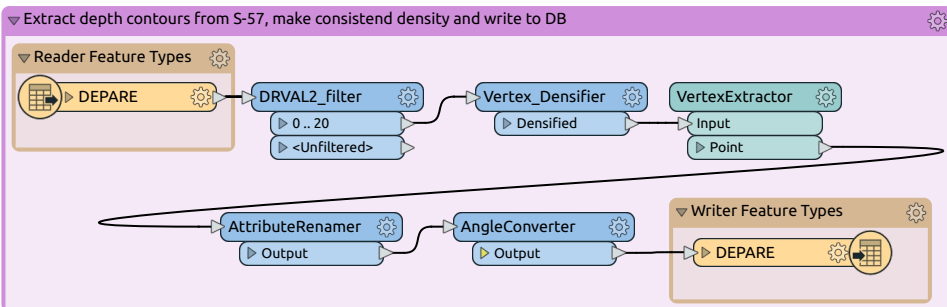
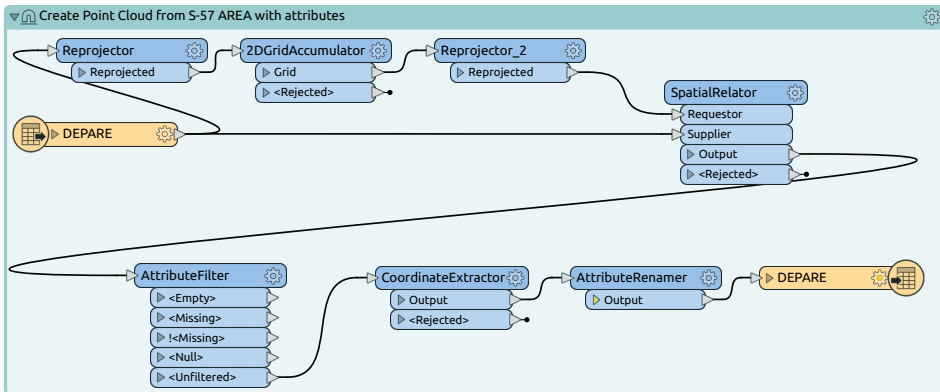


Figure B.3: FME workbench that extracts vertices from a polygon, makes more vertices at uniform intervals around the polygon, and writes the result to a SQLite3 DB.



**Figure B.4:** FME workbench that creates point cloud with values from S-57 polygon areas.



# Appendix C

## RUT950 Configuration

The screenshot shows the RUT950 web interface. At the top, there is a navigation bar with the Teltonika logo and menu items: Status, Network, Services, and System. A Logout button is on the right. Below the navigation bar, the current profile is 'default' and the firmware version is 'RUT9XX\_R\_00.06.05.3'. A secondary navigation bar contains tabs for General Settings, Port Forwarding (selected), Traffic Rules, Custom Rules, DDOS Prevention, Port Scan Prevention, and Helpers. The main content area is titled 'Firewall - Port Forwarding' and includes a brief description: 'Port forwarding allows remote computers on the Internet to connect to a specific computer or service within the private LAN.' Below this is a table of 'Port Forwarding Rules' with two entries: 'DUNE http' and 'ntnu-otter-01-ssh'. Each entry has columns for Name, Protocol, Source, Via, Destination, and Enable, along with Edit and Delete buttons.

Name	Protocol	Source	Via	Destination	Enable	Sort
DUNE http	TCP, UDP	From any host in vpn	To any router IP at port 48080	Forward to IP 192.168.1.224, port 8080 in lan	<input checked="" type="checkbox"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
ntnu-otter-01-ssh	TCP, UDP	From any host in vpn	To any router IP at port 40022	Forward to IP 192.168.1.224, port 22 in lan	<input checked="" type="checkbox"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Figure C.1: RUT950 port forwarding configuration.

## OpenVPN Instance: Client\_otter

### Main Settings

Enable OpenVPN config from file

Enable

TUN/TAP

Protocol

Port

LZO

Authentication

Encryption

TLS cipher

Remote host/IP address

Resolve retry

Keep alive

Remote network IP address

Remote network IP netmask

HMAC authentication algorithm

Additional HMAC authentication

User name

Password

Extra options

Use PKCS #12 format

Certificate authority

Client certificate

Client key

[Back to Overview](#)
[Save](#)

**Figure C.2:** RUT950 OpenVPN configuration.



# Appendix **D**

## Source Code

### D.1 GitHub Repositories

The source of the software developed for this project is available in these GitHub repositories, which are based on forks from the original LSTS repositories. The files that have been created or changed is included in the remaining sections of this appendix.

- Main Otter DUNE branch with the new hydrophone task: <https://github.com/nikkone/dune/tree/ntnuOtterASV/>
- Otter Situational Awareness branch of DUNE: [https://github.com/nikkone/dune/tree/nautical\\_charts\\_radians](https://github.com/nikkone/dune/tree/nautical_charts_radians)
- Otter branch of IMC: <https://github.com/nikkone/imc/tree/ntnuOtterASV>
- The Otter branch of IMCProxy and the IMC to InfluxDB ingester: <https://github.com/nikkone/imcproxy/tree/ntnuOtterASV>
- The Otter branch of Neptus: <https://github.com/nikkone/neptus/tree/ntnuOtterASV>

### D.2 IMCProxy Data Ingester

Code D.1: ImcToInfluxDB.java

```
1 package pt.lsts.imc;
2
3 import java.net.HttpURLConnection;
4 import java.net.URI;
5 import java.net.URL;
6 import java.net.URLConnection;
7
```

```

8 import java.text.SimpleDateFormat;
9 import java.util.Date;
10 import java.util.Map;
11 import java.io.OutputStream;
12 import java.nio.charset.StandardCharsets;
13
14 import org.eclipse.jetty.websocket.api.annotations.WebSocket;
15
16 import pt.lsts.imc.IMCMessage;
17
18 @WebSocket
19 public class ImcToInfluxDB extends ImcClientSocket {
20     protected static SimpleDateFormat format = new
21         SimpleDateFormat ("[YYYY-MM-dd, HH:mm:ss] ");
22     protected static String influxhost = "http://localhost";
23     protected static int influxport = 8086;
24     protected static String dbname = "imc";
25
26     public void httpPostImcToInfluxDB(IMCMessage message) {
27         String messageLineProtocol = ImcToInfluxDB.
28             imcToInfluxLineProtocol(message);
29         if(messageLineProtocol != "") {
30             try {
31                 System.out.println("Writing to DB: " + message.
32                     getAbbrev());
33                 // Connect
34                 URL url = new URL(influxhost + ":" + influxport +
35                     "/write?db=" + dbname + "&precision=ms");
36                 URLConnection con = url.openConnection();
37                 HttpURLConnection http = (HttpURLConnection)con;
38                 http.setRequestMethod("POST"); // PUT is another
39                     valid option
40                 http.setDoOutput(true);
41                 // make
42                 byte[] out = messageLineProtocol.getBytes(
43                     StandardCharsets.UTF_8);
44                 int length = out.length;
45                 // send
46                 http.setFixedLengthStreamingMode(length);
47                 http.setRequestProperty("Content-Type", "
48                     application/x-www-form-urlencoded; charset=UTF
49                     -8");
50                 http.connect();
51                 try(OutputStream os = http.getOutputStream()) {
52                     os.write(out);
53                 } catch (Exception e) {
54                     System.out.println("err while writing db first
55                         ");
56                     return;
57                 }
58             }
59         }
60     }
61 }

```

```

49         } catch (Exception e) {
50             System.out.println("err while writing db");
51             return;
52         }
53     }/* else {
54         System.out.println("Empty");
55     }*/
56 }
57
58 public static String imcToInfluxLineProtocol(IMCMessage
59     message) {
60     String out;
61     switch(message.getAbbrev()) {
62         // Add case case for messages to be stored
63         case "Rpm":
64         case "Voltage":
65         case "Current":
66         case "Temperature":
67         case "FuelLevel":
68         case "SetThrusterActuation":
69         case "GpsFix":
70         case "TBRFishTag":
71         case "TBRSensor":
72         case "EstimatedState":
73             Boolean first = true;
74             out = message.getAbbrev() + ",src="+ message.
75                 getSrc() + ",ent=" + message.getSrcEnt() + " "
76                 ;
77             Map<String, Object> mp = message.getValues();
78             for (Map.Entry<String, Object> entry : mp.entrySet
79                 ()) {
80                 if(first) {
81                     first = false;
82                 } else {
83                     out += ",";
84                 }
85                 out += entry.getKey() + "=" + entry.getValue()
86                 ;
87                 if(entry.getKey().equals("lat") || entry.
88                     getKey().equals("lon")) { // Radians to
89                     Degrees for lat and lon
90                     System.out.println("Lat or Lon added");
91                     out += "," + entry.getKey() + "deg=" + (
92                         double) (entry.getValue())*(180/Math.PI
93                         );
94                 }
95             }
96             out += " " + message.getTimestampMillis();

```

```

89         //System.out.println("Wrote to InfluxDB: " +
90         message.getAbbrev());
91         break;
92     default:
93         out = "";
94     }
95     return out;
96 }
97
98
99 @Override
100 public void onMessage(IMCMessage message) {
101     httpPostImcToInfluxDB(message);
102 }
103
104 public ImcToInfluxDB(String serverHost, int serverPort) throws
105     Exception {
106     connect(new URI("ws://" + serverHost + ":" + serverPort));
107 }
108
109 public static void console(String text) {
110     System.out.println(format.format(new Date()) + text);
111 }
112 public static void main(String[] args) throws Exception {
113     String host = "otter.itk.ntnu.no";
114     int port = 9090;
115
116     if (args.length == 5) {
117         try {
118             port = Integer.parseInt(args[1]);
119             host = args[0];
120             influxport = Integer.parseInt(args[3]);
121             influxhost = args[2];
122             influxhost = args[4];
123         }
124         catch (Exception e) {
125             System.out.println("Usage: ./imcplot <host> <port>
126                 or ");
127             System.out.println("Usage: ./imcplot <host> <port>
128                 <influxhost> <influxport> <dbname>");
129             return;
130         }
131     } else if (args.length == 2) {
132         try {
133             port = Integer.parseInt(args[1]);
134             host = args[0];
135         }
136         catch (Exception e) {

```

```

134         System.out.println("Usage: ./imcplot <host> <port>
           or ");
135         System.out.println("Usage: ./imcplot <host> <port>
           <influxhost> <influxport> <dbname>");
136         return;
137     }
138 }
139 ImcToInfluxDB.console("Connecting to IMCProxy server at "+
           host+": "+port);
140 ImcToInfluxDB.console("Using InfluxDB server at "+
           influxhost+": "+influxport + ",db:" + dbname);
141 new ImcToInfluxDB(host, port);
142 }
143 }

```

## D.3 DUNE

The DUNE files contain a preamble commenting the licensing of the LSTS software, but has been omitted in this appendix. The line numbers on the side of the code represents the line numbers in the original files, which is found in the GitHub repository.

### D.3.1 LocationData Class

Code D.2: LocationData.hpp

```

30 #ifndef DUNE_SITUATIONALAWARENESS_LOCATIONDATA_HPP_INCLUDED_
31 #define DUNE_SITUATIONALAWARENESS_LOCATIONDATA_HPP_INCLUDED_
32
33 // DUNE headers.
34 // #include <DUNE/Config.hpp>
35 #include <DUNE/System.hpp>
36 #include <DUNE/Database.hpp>
37
38
39 namespace DUNE
40 {
41     namespace SituationalAwareness
42     {
43
44         //! The LocationData class
45         class LocationData
46         {
47         public:
48             class Error: public std::runtime_error
49             {
50             public:
51                 Error(std::string op, std::string msg):

```

```

52     std::runtime_error("Error (" + op + "): " + msg)
53     { }
54 };
55 //! Data structure for storing location data
56 struct LocationDataContainer_t
57 { // Could this just be a std::pair<double,double>?
58     double Lat;
59     double Lon;
60     LocationDataContainer_t() {
61         Lat = 0.0;
62         Lon = 0.0;
63     }
64     LocationDataContainer_t(double inLat, double inLon) {
65         Lat = inLat;
66         Lon = inLon;
67     }
68     bool operator<(const LocationDataContainer_t& rhs) const {
69         if (this->Lat < rhs.Lat) return true;
70         if (rhs.Lat < this->Lat) return false;
71
72         // a==b for primary condition, go to secondary
73         if (this->Lon < rhs.Lon) return true;
74         if (rhs.Lon < this->Lon) return false;
75         return false;
76     }
77 };
78 typedef std::vector<LocationDataContainer_t> LocationVector;
79
80 LocationData(const std::string &dbPath);
81 ~LocationData(void);
82 bool writeCSVfile(const LocationVector &inVec, const std::
83     string &outputFile);
84 protected:
85     std::string makeSquareWhereClause(double Lat, double Lon,
86         double half_size);
87     Database::Connection* db_connection;
88 };
89
90 #endif // END define
91     DUNE_SITUATIONALAWARENESS_LOCATIONDATA_HPP_INCLUDED_

```

### Code D.3: LocationData.cpp

```

30 // DUNE headers.
31 // #include <DUNE/DUNE.hpp>
32 #include <DUNE/SituationalAwareness/LocationData.hpp>
33 #include <DUNE/Coordinates.hpp>

```

```

34 #include <DUNE/Math.hpp>
35 // Other C++
36 #include <fstream>
37 #include <iostream>
38 #include <iomanip>
39
40 namespace DUNE
41 {
42     namespace SituationalAwareness
43     {
44         LocationData::LocationData(const std::string &dbPath) {
45             db_connection = NULL;
46             try {
47                 db_connection = new Database::Connection(dbPath.c_str(),
48                 Database::Connection::CF_RDONLY);
49             } catch(std::runtime_error& e) {
50                 throw Error("Could not make DB connection: ", System::
51                 Error::getLastMessage());
52             }
53         }
54
55         LocationData::~LocationData(void) {
56             if (db_connection!=NULL) {
57                 delete db_connection;
58             }
59         }
60
61         std::string LocationData::makeSquareWhereClause(double Lat,
62         double Lon, double half_size) {
63             double lat_minus_displaced = Lat;
64             DUNE::Coordinates::WGS84::displace(-half_size,0.0,&
65             lat_minus_displaced, &Lon);
66
67             double lat_plus_displaced = Lat;
68             DUNE::Coordinates::WGS84::displace(half_size,0.0,&
69             lat_plus_displaced, &Lon);
70
71             double lon_minus_displaced = Lon;
72             DUNE::Coordinates::WGS84::displace(0.0,-half_size,&Lat, &
73             lon_minus_displaced);
74
75             double lon_plus_displaced = Lon;
76             DUNE::Coordinates::WGS84::displace(0.0,half_size,&Lat, &
77             lon_plus_displaced);
78
79             return std::string("Lat between " + std::to_string(
80             lat_minus_displaced) + " and " + std::to_string(
81             lat_plus_displaced) + " and Lon between " + std::
82             to_string(lon_minus_displaced) + " and " + std::
83             to_string(lon_plus_displaced));
84         }
85     }
86 }

```

```

73     }
74
75     bool LocationData::writeCSVfile(const LocationData::
        LocationVector &inVec, const std::string &outputFile) {
76         std::ofstream file_(outputFile);
77         file_ << "Lat,Lon" << "\r\n";
78         for (LocationData::LocationVector::const_iterator itr =
            inVec.begin(); itr != inVec.end(); ++itr)
79             {
80                 file_ << std::setprecision(15) << DUNE::Math::Angles::
                    degrees(itr->Lat) << "," << DUNE::Math::Angles::
                    degrees(itr->Lon) << "\r\n";
81             }
82         return false;
83     }
84
85 } // End of namespace SituationalAwareness
86 } // End of namespace DUNE

```

### D.3.2 DensifiedVertices Class

Code D.4: DensifiedVertices.hpp

```

30 #ifndef DUNE_SITUATIONALAWARENESS_DENSIFIEDVERTICES_HPP_INCLUDED_
31 #define DUNE_SITUATIONALAWARENESS_DENSIFIEDVERTICES_HPP_INCLUDED_
32
33 // DUNE headers.
34 #include <DUNE/SituationalAwareness/LocationData.hpp>
35
36
37 namespace DUNE
38 {
39     namespace SituationalAwareness
40     {
41
42         //! The DensifiedVertices class
43         class DensifiedVertices: public LocationData
44         {
45         public:
46             //! Data structure for storing depth range data, DRVAL1=
                mindepth, DRVAL2=maxdepth
47             struct DensifiedVerticesContainer_t: LocationDataContainer_t
48             {
49                 double DRVAL1;
50                 double DRVAL2;
51                 DensifiedVerticesContainer_t(): LocationDataContainer_t
                    (0.0,0.0) {
52                     DRVAL1 = 0.0;

```



```

53         DRVAL2 = 0.0;
54     }
55     DensifiedVerticesContainer_t(double inLat, double inLon,
        double inDRVAL1, double inDRVAL2):
        LocationDataContainer_t(inLat,inLon) {
56         DRVAL1 = inDRVAL1;
57         DRVAL2 = inDRVAL2;
58     }
59 };
60 typedef std::vector<DensifiedVerticesContainer_t>
        DEPAREVector;
61
62 DensifiedVertices(std::string dbPath, double verticeDist);
63 ~DensifiedVertices(void);
64
65 bool isDepthAbove(DEPAREVector DEPAREinputVector, double
        minDepth);
66
67 bool writeCSVfile(DEPAREVector depths, std::string
        outputFile);
68
69 DEPAREVector getWithinRadius(double Lat, double Lon, double
        radius);
70
71 DEPAREVector getCorridor(double startLat, double startLon,
        double endLat, double endLon, double steps, double
        corridorWidth);
72
73 DEPAREVector getCorridor(double startLat, double startLon,
        double endLat, double endLon, double corridorWidth);
74
75 DEPAREVector getSquare(double Lat, double Lon, double
        half_size);
76 private:
77     double verticeDist;
78 };
79 }
80 }
81
82
83 #endif // END define
        DUNE_SITUATIONALAWARENESS_DENSIFIEDVERTICES_HPP_INCLUDED_

```

#### Code D.5: DensifiedVertices.cpp

```

30 // DUNE headers.
31 #include <DUNE/SituationalAwareness/DensifiedVertices.hpp>
32 #include <DUNE/Coordinates.hpp>
33 #include <DUNE/Math.hpp>
34 #include <DUNE/System/Error.hpp>

```

---

```

35
36 // Other C++
37 #include <fstream>
38 #include <iostream>
39
40 namespace DUNE
41 {
42     namespace SituationalAwareness
43     {
44         DensifiedVertices::DensifiedVertices(std::string dbPath,
45             double dbVerticeDist):DUNE::SituationalAwareness::
46             LocationData(dbPath), verticeDist(dbVerticeDist){
47
48     }
49
50     DensifiedVertices::~DensifiedVertices(void) {
51
52     }
53
54     /// DEPARE functions ///
55     DensifiedVertices::DEPAREVector DensifiedVertices::
56     getWithinRadius(double Lat, double Lon, double radius)
57     {
58         DensifiedVertices::DEPAREVector returnVector =
59         DensifiedVertices::DEPAREVector();
60         DensifiedVertices::DEPAREVector square = this->getSquare(Lat
61             ,Lon, radius);
62         for (DensifiedVertices::DEPAREVector::iterator itr = square.
63             begin(); itr != square.end(); ++itr)
64         {
65             double distance = Coordinates::WGS84::distance(Lat, Lon,0,
66                 itr->Lat, itr->Lon,0);
67             if(distance<=radius) {
68                 returnVector.push_back(DensifiedVertices::
69                     DensifiedVerticesContainer_t(itr->Lat, itr->Lon,itr
70                         ->DRVAL1,itr->DRVAL2));
71             }
72         }
73         return returnVector;
74     }
75
76     bool
77     DensifiedVertices::isDepthAbove(DensifiedVertices::
78         DEPAREVector DEPAREinputVector, double minDepth) {
79         for (DensifiedVertices::DEPAREVector::iterator itr =
80             DEPAREinputVector.begin(); itr != DEPAREinputVector.end
81             (); ++itr)

```

---

```

73     {
74         if(itr->DRVAL1<minDepth)
75             return false;
76     }
77     return true;
78 }
79
80 DensifiedVertices::DEPAREVector DensifiedVertices::getSquare(
    double Lat, double Lon, double half_size)
81 {
82     std::string tablename = "DEPARE";
83     std::string c_stmt = "select Lat, Lon, DRVAL1, DRVAL2 from "
        + tablename + " where " + makeSquareWhereClause(Lat,
        Lon, half_size) + ";";
84     //std::cout << c_stmt;
85     DensifiedVertices::DEPAREVector returnMap;
86     try{
87         Database::Statement* iterator_stmt = new Database::
            Statement(c_stmt.c_str(), *db_connection);
88         std::pair<bool, double> SQLLat;
89         std::pair<bool, double> SQLLon;
90         std::pair<bool, double> SQLDRVAL1;
91         std::pair<bool, double> SQLDRVAL2;
92         unsigned int cntr=0;
93         while (iterator_stmt->execute())
94             {
95                 *iterator_stmt >> SQLLat >> SQLLon >> SQLDRVAL1 >>
                    SQLDRVAL2;
96                 if(std::get<0>(SQLLat)) {
97                     returnMap.push_back(DensifiedVertices::
                        DensifiedVerticesContainer_t(std::get<1>(SQLLat),
                        std::get<1>(SQLLon), std::get<1>(SQLDRVAL1), std::
                        get<1>(SQLDRVAL2)));
98                     cntr++;
99                 }
100             }
101         iterator_stmt->reset();
102         delete iterator_stmt;
103     } catch(std::runtime_error& e) {
104         throw Error("Problem while executing statement \"" +
            c_stmt + "\": ", System::Error::getLastMessage());
105     }
106     return returnMap;
107 }
108
109 bool DensifiedVertices::writeCSVfile(DensifiedVertices::
    DEPAREVector depths, std::string outputFile)
110 {
111     std::ofstream file_(outputFile);
112     for (DensifiedVertices::DEPAREVector::iterator itr =

```

```

113     depths.begin(); itr != depths.end(); ++itr)
114     {
115         file_ << Math::Angles::degrees(itr->Lat) << "," << Math
116             ::Angles::degrees(itr->Lon) << "," << itr->DRVAL1 <<
117             "," << itr->DRVAL2 << "\r\n";
118     }
119     return false;
120 }
121
122 DensifiedVertices::DEPAREVector DensifiedVertices::getCorridor
123     (double startLat, double startLon, double endLat, double
124     endLon, double steps, double corridorWidth) {
125     float stepLat= (endLat-startLat)/steps;
126     float stepLon= (endLon-startLon)/steps;
127
128     DensifiedVertices::DEPAREVector returnDEPAREVector = this->
129     getSquare(startLat, startLon, corridorWidth);
130     for(unsigned int step=1; step<steps;step++) {
131         DensifiedVertices::DEPAREVector squareAroundCurrentStep =
132         this->getSquare(startLat+step*stepLat, startLon+step*
133         stepLon, corridorWidth);
134
135         returnDEPAREVector.insert(returnDEPAREVector.end(),
136         squareAroundCurrentStep.begin(),
137         squareAroundCurrentStep.end());
138     }
139     return returnDEPAREVector;
140 }
141
142 DensifiedVertices::DEPAREVector DensifiedVertices::getCorridor
143     (double startLat, double startLon, double endLat, double
144     endLon, double corridorWidth) {
145     double distance = Coordinates::WGS84::distance(startLat,
146     startLon,0, endLat, endLon,0);
147     return getCorridor(startLat, startLon, endLat, endLon,
148     distance/(verticeDist/2*std::sqrt(2)), corridorWidth);
149 }
150 }
151 }

```

### D.3.3 TwoDGrid Class

Code D.6: TwoDGrid.hpp

```

30 #ifndef DUNE_SITUATIONALAWARENESS_TwoDGrid_HPP_INCLUDED_
31 #define DUNE_SITUATIONALAWARENESS_TwoDGrid_HPP_INCLUDED_
32
33 // DUNE headers.

```

---

```

34 #include <DUNE/SituationalAwareness/LocationData.hpp>
35
36
37 namespace DUNE
38 {
39     namespace SituationalAwareness
40     {
41         // Export DLL Symbol.
42         class DUNE_DLL_SYM TwoDGrid;
43
44         //! The TwoDGrid class
45         class TwoDGrid: public LocationData
46         {
47         public:
48
49             //! Data structure for storing depth data
50             struct DepthDataContainer_t: LocationDataContainer_t
51             {
52                 double Depth;
53                 DepthDataContainer_t(): LocationDataContainer_t(0.0,0.0) {
54                     Depth = 0.0;
55                 }
56                 DepthDataContainer_t(double inLat, double inLon, double
57                     inDepth): LocationDataContainer_t(inLat,inLon) {
58                     Depth = inDepth;
59                 }
60             };
61
62             typedef std::vector<DepthDataContainer_t> DepthVector;
63
64             //! TwoDGrid constructor.
65             TwoDGrid(const std::string &dbPath, double dbGridSize);
66
67             //! TwoDGrid destructor.
68             ~TwoDGrid(void);
69
70             //////////////////////////////////// DepthmapRad functions
71
72             //! Returns the depth of the queried location if it exists.
73             //! Returns the depth of the location closest to the queried
74             one, their distance and bearing.
75             std::vector<double>
76             getSingleDepth(double Lat, double Lon, double grid_size);
77
78             DepthVector
79             getClosestDepths(double Lat, double Lon, double grid_size);
80
81             DepthVector
82             getSquare(const LocationDataContainer_t &center, double
83                 half_size, double minDepth_m = 0.0);

```

---

```

81
82     DepthVector
83     getSquare(double Lat, double Lon, double half_size, double
            minDepth_m = 0.0);
84
85     DepthVector
86     getWithinRadius(double Lat, double Lon, double radius,
            double minDepth_m = 0.0);
87
88     DepthVector
89     getWithinRadius(const LocationDataContainer_t &center,
            double radius, double minDepth_m = 0.0);
90
91     bool writeCSVfile(const DepthVector &depths, const std::
            string &outputFile);
92
93     std::pair<DepthVector, LocationVector>
94     checkTransect(double startLat, double startLon, double
            endLat, double endLon, double steps);
95
96     std::pair<DepthVector, LocationVector>
97     checkTransect(double startLat, double startLon, double
            endLat, double endLon);
98
99     DepthVector
100    getCorridor(double startLat, double startLon, double endLat,
            double endLon, double steps, double corridorWidth);
101
102    DepthVector
103    getCorridor(double startLat, double startLon, double endLat,
            double endLon, double corridorWidth);
104    private:
105        double gridSize;
106    };
107 }
108 }
109
110
111 #endif

```

#### Code D.7: TwoDGrid.cpp

```

30 // DUNE headers.
31 #include <DUNE/Coordinates.hpp>
32 #include <DUNE/Math.hpp>
33 #include <DUNE/System/Error.hpp>
34 #include <DUNE/SituationalAwareness/TwoDGrid.hpp>
35
36 // Other C++
37 #include <fstream>

```

```

38 #include <iostream>
39 #include <iomanip>
40
41 namespace DUNE
42 {
43     namespace SituationalAwareness
44     {
45         TwoDGrid::TwoDGrid(const std::string &dbPath, double
            dbGridSize):DUNE::SituationalAwareness::LocationData(
            dbPath), gridSize(dbGridSize) {
46         }
47         TwoDGrid::~TwoDGrid(void){
48         }
49
50         std::vector<double> TwoDGrid::getSingleDepth(double Lat,
            double Lon, double grid_size)
51         {
52             std::vector<double> ranges, bearings, depths;
53             std::vector<double> return_vec;
54             double range, bearing;
55
56             //std::string db_statement
57             std::string c_stmt = "select Depth from 'depthmapRad' where
                Lat = " + std::to_string(Lat) + " and Lon = " + std::
                to_string(Lon) + ";";
58
59             try
60             {
61                 Database::Statement* iterator_stmt = new Database::
                    Statement(c_stmt.c_str(), *db_connection);
62                 std::pair<bool, double> DBDepth;
63
64                 while(iterator_stmt->execute())
65                 {
66                     *iterator_stmt >> DBDepth;
67
68                     if(std::get<0>(DBDepth))
69                     {
70                         iterator_stmt->reset();
71                         delete iterator_stmt;
72                         return_vec.push_back(std::get<1>(DBDepth));
73                         return return_vec;
74                     }
75                 }
76             } catch(std::runtime_error& e)
77             {
78                 throw Error("Problem while executing statement \"" +
                    c_stmt + "\": ", System::Error::getLastMessage());
79             }
80

```

```

81     printf("This location does not exist in the map, retrieving
82           the closest . . . \n");
83     TwoDGrid::DepthVector four_closest = getClosestDepths(Lat,
84                 Lon, grid_size);
85     for(TwoDGrid::DepthVector::iterator itr = four_closest.begin
86         (); itr != four_closest.end(); ++itr)
87     {
88         //std::cout << itr->Lat << " " << itr->Lon << "\n";
89         Coordinates::WGS84::getNEBearingAndRange(Lat, Lon, itr->Lat
90             , itr->Lon, &bearing, &range);
91         ranges.push_back(std::fabs(range));
92         bearings.push_back(bearing);
93         depths.push_back(itr->Depth);
94     }
95     int min_index = std::min_element(ranges.begin(), ranges.end()
96         ) - ranges.begin();
97     double min_range = *std::min_element(ranges.begin(), ranges.
98         end());
99     return_vec.push_back(depths[min_index]);
100    return_vec.push_back(min_range);
101    return_vec.push_back(bearings[min_index]);
102
103    return return_vec;
104 }
105
106 TwoDGrid::DepthVector TwoDGrid::getClosestDepths(double Lat,
107     double Lon, double grid_size)
108 {
109     double disp = 2*grid_size;
110     std::string c_stmt = "select min(Lat+Lon), Lat, Lon, Depth
111         from (select Lat, Lon, Depth from depthmapRad where " +
112         makeSquareWhereClause(Lat, Lon, disp) + ") where Lat >=
113         " + std::to_string(Lat) + " and Lon >= " + std:::
114         to_string(Lon) +
115     " union select max(Lat+Lon), Lat, Lon, Depth from (select
116         Lat, Lon, Depth from depthmapRad where " +
117         makeSquareWhereClause(Lat, Lon, disp) + ") where Lat <=
118         " + std::to_string(Lat) + " and Lon <= " + std:::
119         to_string(Lon) +
120     " union select min(Lat-Lon), Lat, Lon, Depth from (select
121         Lat, Lon, Depth from depthmapRad where " +
122         makeSquareWhereClause(Lat, Lon, disp) + ") where Lat >=
123         " + std::to_string(Lat) + " and Lon <= " + std:::
124         to_string(Lon) +
125     " union select max(Lat-Lon), Lat, Lon, Depth from (select
126         Lat, Lon, Depth from depthmapRad where " +
127         makeSquareWhereClause(Lat, Lon, disp) + ") where Lat <=
128         " + std::to_string(Lat) + " and Lon >= " + std:::
129         to_string(Lon) + ";";

```



```

108
109 //std::cout << c_stmt;
110 TwoDGrid::DepthVector returnMap;
111 try{
112     Database::Statement* iterator_stmt = new Database::
113         Statement(c_stmt.c_str(), *db_connection);
114     std::pair<bool, double> DBLat;
115     std::pair<bool, double> DBLon;
116     std::pair<bool, double> DBDepth;
117     while (iterator_stmt->execute())
118     {
119         *iterator_stmt >> DBLat >> DBLat >> DBLon >> DBDepth; //
120         Four because max/min column gets discarded
121         if(std::get<0>(DBLat)) { // If row not empty (only
122             checks first element)
123             returnMap.push_back(TwoDGrid::DepthDataContainer_t(std
124                 ::get<1>(DBLat), std::get<1>(DBLon), std::get<1>(
125                     DBDepth));
126         }
127     }
128     iterator_stmt->reset();
129     delete iterator_stmt;
130 } catch(std::runtime_error& e) {
131     throw Error("Problem while executing statement \"" +
132         c_stmt + "\": ", System::Error::getLastMessage());
133 }
134 return returnMap;
135 }
136
137 TwoDGrid::DepthVector TwoDGrid::getSquare(const TwoDGrid::
138     LocationDataContainer_t &center, double half_size, double
139     minDepth_m) {
140     return this->getSquare(center.Lat, center.Lon, half_size,
141         minDepth_m);
142 }
143
144 TwoDGrid::DepthVector TwoDGrid::getSquare(double Lat, double
145     Lon, double half_size, double minDepth_m)
146 {
147     //select * from depthmapRad where depth > 20 and Lat between
148     1.1 and 1.101
149     std::string c_stmt = "select Lat, Lon, Depth from depthmapRad
150         where Depth > " +std::to_string(minDepth_m)+ " and " +
151         makeSquareWhereClause(Lat, Lon, half_size) + ";";
152     TwoDGrid::DepthVector returnMap;
153     try{
154         Database::Statement* iterator_stmt = new Database::
155             Statement(c_stmt.c_str(), *db_connection);
156         std::pair<bool, double> DBLat;

```

```

144     std::pair<bool, double> DBLon;
145     std::pair<bool, double> DBDepth;
146     while (iterator_stmt->execute())
147     {
148         *iterator_stmt >> DBLat >> DBLon >> DBDepth;
149         if(std::get<0>(DBLat)) { // If row not NULL
150             returnMap.push_back(TwoDGrid::DepthDataContainer_t(std
                ::get<1>(DBLat), std::get<1>(DBLon), std::get<1>(
                DBDepth)));
151         }
152     }
153     iterator_stmt->reset();
154     delete iterator_stmt;
155 } catch(std::runtime_error& e) {
156     throw Error("Problem while executing statement \"" +
        c_stmt + "\": ", System::Error::getLastMessage());
157 }
158 return returnMap;
159 }
160
161 TwoDGrid::DepthVector TwoDGrid::getWithinRadius(const TwoDGrid
    ::LocationDataContainer_t &center, double radius, double
    minDepth_m) {
162     TwoDGrid::DepthVector returnVector = TwoDGrid::DepthVector()
        ;
163     TwoDGrid::DepthVector square = this->getSquare(center,
        radius, minDepth_m);
164     for (TwoDGrid::DepthVector::iterator itr = square.begin();
        itr != square.end(); ++itr)
165     {
166         double distance = Coordinates::WGS84::distance(center.Lat,
            center.Lon, 0, itr->Lat, itr->Lon, 0);
167         if(distance<=radius) {
168             returnVector.push_back(TwoDGrid::DepthDataContainer_t(
                itr->Lat, itr->Lon, itr->Depth));
169         }
170     }
171     return returnVector;
172 }
173
174 TwoDGrid::DepthVector TwoDGrid::getWithinRadius(double Lat,
    double Lon, double radius, double minDepth_m)
175 {
176     TwoDGrid::LocationDataContainer_t center = TwoDGrid::
        LocationDataContainer_t(Lat, Lon);
177     return this->getWithinRadius(center, radius, minDepth_m);
178 }
179
180
181 bool TwoDGrid::writeCSVfile(const TwoDGrid::DepthVector &

```

```

    depths, const std::string &outputFile)
182 {
183     std::ofstream file_(outputFile);
184     file_ << "Lat,Lon,Depth" << "\r\n";
185     for (TwoDGrid::DepthVector::const_iterator itr = depths.
        begin(); itr != depths.end(); ++itr)
186     {
187         file_ << std::setprecision(15) << Math::Angles::degrees(
            itr->Lat) << "," << Math::Angles::degrees(itr->Lon) <<
            "," << Math::Angles::degrees(itr->Depth) << "\r\n";
188     }
189     return false;
190 }
191
192 std::pair<TwoDGrid::DepthVector, TwoDGrid::LocationVector>
193 TwoDGrid::checkTransect(double startLat, double startLon,
    double endLat, double endLon, double steps) { // Add depth
    limit
194     float stepLat= (endLat-startLat)/steps;
195     float stepLon= (endLon-startLon)/steps;
196     TwoDGrid::LocationVector groundingPositions=TwoDGrid::
        LocationVector();
197     TwoDGrid::DepthVector returnVector = TwoDGrid::DepthVector()
        ;
198
199     // Iterate through points on line between start- and end-
        position
200     for(unsigned int step=0; step<steps;step++) {
201
202         // Get four closest depth soundings
203         TwoDGrid::DepthVector fourClosest = this->getClosestDepths
            (startLat+step*stepLat, startLon+step*stepLon,
                gridSize);
204
205         // If less than four depth soundings available, it's
            probably ground
206         if(fourClosest.size() < 4) {
207             groundingPositions.push_back(TwoDGrid::
                LocationDataContainer_t(startLat+step*stepLat,
                    startLon+step*stepLon));
208         } else {
209
210             // Select the closest of the four depth soundings
211             double closest_distance=-1;
212             TwoDGrid::DepthDataContainer_t closest= TwoDGrid::
                DepthDataContainer_t();
213             for (TwoDGrid::DepthVector::iterator itr = fourClosest.
                begin(); itr != fourClosest.end(); ++itr) {
214                 double distance = Coordinates::WGS84::distance(
                    startLat+step*stepLat, startLon+step*stepLon,0,

```

```

        itr->Lat, itr->Lon,0);
215     if (distance < closest_distance or closest_distance <
        0.0) {
216         closest_distance=distance;
217         closest = *itr;
218     }
219 }
220 returnVector.push_back(closest);
221 }
222 }
223 return std::pair<TwoDGrid::DepthVector, TwoDGrid::
        LocationVector>(returnVector,groundingPositions);
224 }
225
226
227 std::pair<TwoDGrid::DepthVector, TwoDGrid::LocationVector>
228 TwoDGrid::checkTransect(double startLat, double startLon,
        double endLat, double endLon) {
229     double distance = Coordinates::WGS84::distance(startLat,
        startLon,0, endLat, endLon,0);
230     return TwoDGrid::checkTransect(startLat, startLon, endLat,
        endLon, distance/gridSize);
231 }
232
233 TwoDGrid::DepthVector TwoDGrid::getCorridor(double startLat,
        double startLon, double endLat, double endLon, double
        steps, double corridorWidth) {
234     float stepLat= (endLat-startLat)/steps;
235     float stepLon= (endLon-startLon)/steps;
236
237     TwoDGrid::DepthVector returnDepthVector = this->getSquare(
        startLat, startLon, corridorWidth);
238     for(unsigned int step=1; step<steps;step++) {
239         TwoDGrid::DepthVector squareAroundCurrentStep = this->
            getSquare(startLat+step*stepLat, startLon+step*stepLon
                , corridorWidth);
240
241         returnDepthVector.insert(returnDepthVector.end(),
            squareAroundCurrentStep.begin(),
            squareAroundCurrentStep.end());
242     }
243     return returnDepthVector;
244 }
245
246 TwoDGrid::DepthVector TwoDGrid::getCorridor(double startLat,
        double startLon, double endLat, double endLon, double
        corridorWidth) {
247     double distance = Coordinates::WGS84::distance(startLat,
        startLon,0, endLat, endLon,0);
248     return getCorridor(startLat, startLon, endLat, endLon,

```

```

249         distance/(gridSize/2*std::sqrt(2)), corridorWidth);
250     }
251 } // End of namespace SituationalAwareness
    } // End of namespace DUNE

```

### D.3.4 PointsOfInterest Class

**Code D.8:** PointsOfInterest.hpp

```

30 #ifndef DUNE_SITUATIONALAWARENESS_POINTSOFINTEREST_HPP_INCLUDED_
31 #define DUNE_SITUATIONALAWARENESS_POINTSOFINTEREST_HPP_INCLUDED_
32
33 // DUNE headers.
34 // #include <DUNE/Config.hpp>
35 // #include <DUNE/DUNE.hpp>
36 #include <DUNE/SituationalAwareness/LocationData.hpp>
37
38
39 namespace DUNE
40 {
41     namespace SituationalAwareness
42     {
43         // Export DLL Symbol.
44         class DUNE_DLL_SYM PointsOfInterest;
45
46         //! The PointsOfInterest class
47         class PointsOfInterest: public LocationData
48         {
49         public:
50             //! PointsOfInterest constructor.
51             PointsOfInterest(std::string dbPath);
52
53             //! PointsOfInterest destructor.
54             ~PointsOfInterest(void);
55             LocationVector getPOISquare(double Lat, double Lon, double
                    half_size, std::string tablename);
56         };
57     }
58 }
59
60
61 #endif // END
    DUNE_SITUATIONALAWARENESS_POINTSOFINTEREST_HPP_INCLUDED_

```

**Code D.9:** PointsOfInterest.cpp

```

30 // DUNE headers.
31 #include <DUNE/DUNE.hpp>

```

---

```

32 #include <DUNE/System/Error.hpp>
33 #include <DUNE/SituationalAwareness/PointsOfInterest.hpp>
34
35 // Other C++
36     #include <fstream>
37 #include <iostream>
38
39 namespace DUNE
40 {
41     namespace SituationalAwareness
42     {
43         PointsOfInterest::PointsOfInterest(std::string dbPath):DUNE::
            SituationalAwareness::LocationData(dbPath) {
44         }
45
46         PointsOfInterest::~PointsOfInterest(void) {
47         }
48
49         PointsOfInterest::LocationVector PointsOfInterest::
            getPOISquare(double Lat, double Lon, double half_size, std
                ::string tablename)
50         {
51
52             std::string c_stmt = "select Lat, Lon from " + tablename + "
                where " + makeSquareWhereClause(Lat, Lon, half_size);
53
54             PointsOfInterest::LocationVector returnVector;
55             try{
56                 Database::Statement* iterator_stmt = new Database::
                    Statement(c_stmt.c_str(), *db_connection);
57                 std::pair<bool, double> SQLLat;
58                 std::pair<bool, double> SQLLon;
59                 unsigned int cntr=0;
60                 while (iterator_stmt->execute())
61                 {
62                     *iterator_stmt >> SQLLat >> SQLLon;
63                     if(std::get<0>(SQLLat)) {
64                         returnVector.push_back(PointsOfInterest::
                            LocationDataContainer_t(std::get<1>(SQLLat), std::
                                get<1>(SQLLon)));
65                         cntr++;
66                     }
67                 }
68                 iterator_stmt->reset();
69                 delete iterator_stmt;
70             } catch(std::runtime_error& e) {
71                 throw Error("Problem while executing statement \"" +
                    c_stmt + "\": ", System::Error::getLastMessage());
72             }
73             return returnVector;

```

---

```

74     }
75 } // End of namespace SituationalAwareness
76 } // End of namespace DUNE

```

### D.3.5 PathPlanner Class

Code D.10: PathPlanner.hpp

```

30 #ifndef DUNE_SITUATIONALAWARENESS_PATHPLANNER_HPP_INCLUDED_
31 #define DUNE_SITUATIONALAWARENESS_PATHPLANNER_HPP_INCLUDED_
32
33 // DUNE headers.
34 #include <DUNE/SituationalAwareness/TwoDGrid.hpp>
35 #include <map>
36
37 namespace DUNE
38 {
39     namespace SituationalAwareness
40     {
41         // Export DLL Symbol.
42         class DUNE_DLL_SYM PathPlanner;
43
44         //! The TwoDGrid class
45         class PathPlanner: private DUNE::SituationalAwareness::
            TwoDGrid
46         {
47         public:
48             class Error: public std::runtime_error
49             {
50             public:
51                 Error(std::string op, std::string msg):
52                     std::runtime_error("PathPlanner error (" + op + "):
53                                     " + msg)
54             { }
55             };
56         //! Data structure for search
57         class searchData_t
58         {
59         public:
60             double f;
61             double g;
62             LocationDataContainer_t parent;
63             searchData_t(): f(0.0), g(0.0), parent() {
64             }
65             searchData_t(double inF, double inG, double
66                 inLatParent, double inLonParent): f(inF), g(inG),
67                 parent(inLatParent, inLonParent) {

```

```

66     };
67
68     typedef std::pair<LocationDataContainer_t, searchData_t>
        searchNode;
69     typedef std::map<LocationDataContainer_t, searchData_t>
        searchNodeMap;
70
71     PathPlanner(const std::string &dbPath, double dbGridSize);
72     ~PathPlanner(void);
73
74     searchNodeMap findPath(double startLat, double startLon,
        double endLat, double endLon, double squareSize = 75,
        double goalDistance = 37.0, unsigned int maxIter =
        1000000);
75
76     searchNodeMap tracePath(PathPlanner::searchNodeMap::
        const_iterator endNode, const searchNodeMap &inNodeMap
        );
77     void writeCSVfile(const searchNodeMap &inMap, const std::
        string &outputFile);
78 private:
79     double calcH(const LocationDataContainer_t &node, const
        LocationDataContainer_t &endNode);
80     double calcG(const LocationDataContainer_t &node, const
        LocationDataContainer_t &endNode);
81
82     bool isNodeGoal(const LocationDataContainer_t &node, const
        LocationDataContainer_t &goalNode, double
        distanceLimit);
83     bool isNodeInMap(PathPlanner::searchNodeMap::
        const_iterator nodeItr, const searchNodeMap &inMap);
84
85     searchNodeMap depthVecToSearchMap(const TwoDGrid::
        DepthVector &inDepthVec);
86
87     searchNodeMap::const_iterator findParent(searchNodeMap::
        const_iterator inNodeItr, const searchNodeMap &
        inNodeMap);
88     searchNodeMap::iterator findLowestf(searchNodeMap &
        inNodeMap);
89 };
90 }
91 }
92
93
94 #endif

```

#### Code D.11: PathPlanner.cpp

```

30 // DUNE headers.

```



---

```

31 #include <DUNE/SituationalAwareness/PathPlanner.hpp>
32 #include <DUNE/Coordinates.hpp>
33 #include <DUNE/Math.hpp>
34 // Other C++
35 #include <fstream>
36 #include <iostream>
37 #include <iomanip>
38
39 namespace DUNE
40 {
41     namespace SituationalAwareness
42     {
43         PathPlanner::PathPlanner(const std::string &dbPath, double
            dbGridSize):TwoDGrid(dbPath, dbGridSize){
44
45         }
46         PathPlanner::~PathPlanner(void) {
47
48         }
49         /// Path planning ///
50
51
52         double PathPlanner::calcH(const PathPlanner::
            LocationDataContainer_t &node, const PathPlanner::
            LocationDataContainer_t &endNode) {
53             /*double D=50;
54             double D2=75;
55             double d_max = std::max( std::abs(node.Lat-endNode.Lat),
                std::abs(node.Lon-endNode.Lon) );
56             double d_min = std::min( std::abs(node.Lat-endNode.Lat),
                std::abs(node.Lon-endNode.Lon) );
57             return D*d_max + (D2-D)*d_min;*/
58
59             // uniform Diagonal Distance heuristic WGS84
60             // return std::max(WGS84::distance(nodeLat, nodeLon,0,
                nodeLat, endLon,0), WGS84::distance(nodeLat, nodeLon,0,
                endLat, nodeLon,0));
61
62             // Euclidian distance heuristic
63             return 1*Coordinates::WGS84::distance(node.Lat, node.Lon,0,
                endNode.Lat, endNode.Lon,0);
64
65             // Manhattan distance heuristic WGS84
66             // return WGS84::distance(nodeLat, nodeLon,0, nodeLat,
                endLon,0+WGS84::distance(nodeLat, nodeLon,0, endLat,
                nodeLon,0);
67         }
68
69         double PathPlanner::calcG(const PathPlanner::
            LocationDataContainer_t &fromNode, const PathPlanner::

```

---

```

70     LocationDataContainer_t &toNode) {
71     // Euclidian distance
72     return Coordinates::WGS84::distance(fromNode.Lat, fromNode.
73         Lon,0, toNode.Lat, toNode.Lon,0);
74 }
75
76 PathPlanner::searchNodeMap::iterator PathPlanner::findLowestf(
77     PathPlanner::searchNodeMap &inNodeMap) {
78     PathPlanner::searchNodeMap::iterator lowestFNode = inNodeMap
79         .end();
80     for (PathPlanner::searchNodeMap::iterator itr = inNodeMap.
81         begin(); itr != inNodeMap.end(); ++itr) {
82         if(lowestFNode==inNodeMap.end() || itr->second.f <
83             lowestFNode->second.f) {
84             lowestFNode = itr;
85         }
86     }
87     return lowestFNode;
88 }
89
90 bool PathPlanner::isNodeGoal(const PathPlanner::
91     LocationDataContainer_t &node, const PathPlanner::
92     LocationDataContainer_t &goalNode, double distanceLimit){
93
94     if( distanceLimit < Coordinates::WGS84::distance(node.Lat,
95         node.Lon,0, goalNode.Lat, goalNode.Lon,0) ) {
96         return false;
97     } else {
98         return true;
99     }
100 }
101
102 bool PathPlanner::isNodeInMap(PathPlanner::searchNodeMap::
103     const_iterator nodeItr, const PathPlanner::searchNodeMap &
104     inMap) {
105     PathPlanner::searchNodeMap::const_iterator match = inMap.
106         find(nodeItr->first);
107     if(match != inMap.end())
108         return true;
109
110     return false;
111 }
112
113 PathPlanner::searchNodeMap PathPlanner::depthVecToSearchMap(
114     const TwoDGrid::DepthVector &inDepthVec) {
115     PathPlanner::searchNodeMap returnNodes;
116     for (TwoDGrid::DepthVector::const_iterator itr = inDepthVec.
117         begin(); itr != inDepthVec.end(); ++itr) {
118         returnNodes.emplace(LocationDataContainer_t(itr->Lat, itr
119             ->Lon), PathPlanner::searchData_t(0.0, -1.0, 0.0, 0.0)

```

```

        );
105     }
106     return returnNodes;
107 }
108
109 PathPlanner::searchNodeMap::const_iterator PathPlanner::
    findParent(PathPlanner::searchNodeMap::const_iterator
    inNodeItr, const PathPlanner::searchNodeMap &inNodeMap) {
110     return inNodeMap.find(inNodeItr->second.parent);
111 }
112
113 PathPlanner::searchNodeMap PathPlanner::tracePath(PathPlanner
    ::searchNodeMap::const_iterator endNode, const PathPlanner
    ::searchNodeMap &inNodeMap) {
114     PathPlanner::searchNodeMap returnMap;
115     unsigned int maxIter = 10000;
116     unsigned int iter = 0;
117     PathPlanner::searchNodeMap::const_iterator node = findParent
        (endNode, inNodeMap);
118     while(!inNodeMap.empty()) {
119         returnMap.emplace(*node);
120         if(iter >= maxIter) {
121             printf("maxIter reached");
122             break;
123         }
124         if(node->second.f== -1) {
125             printf("Start reached");
126             break;
127         }
128         if(node == inNodeMap.end()) {
129             printf("Parent not found");
130             break;
131         }
132         node = findParent(node, inNodeMap);
133         iter++;
134     }
135     return returnMap;
136 }
137
138 //! A* algorithm using depthmapRad
139 PathPlanner::searchNodeMap PathPlanner::findPath(double
    startLat, double startLon, double endLat, double endLon,
    double squareSize, double goalDistance, unsigned int
    maxIter) {
140     unsigned int iterations = 0;
141     double minDepth_m = 0.0;
142     PathPlanner::searchNodeMap openNodes;
143     PathPlanner::searchNodeMap closedNodes;
144     LocationDataContainer_t endLocation =
        LocationDataContainer_t(endLat, endLon);

```

---

```

145     openNodes.emplace(LocationDataContainer_t(startLat, startLon
146         ),PathPlanner::searchData_t(-1.0, 0.0, 0.0, 0.0));
147
148     // Allocating loop-variables
149     PathPlanner::searchNodeMap::iterator currentNodeItr;
150     std::pair<PathPlanner::searchNodeMap::iterator, bool>
151         currentPair;
152     PathPlanner::searchNode currentNode;
153     double tentativeG;
154     PathPlanner::searchNodeMap::iterator currentSuccessorNodeItr
155         ;
156     while(!openNodes.empty()) {
157         if(iterations >= maxIter) {
158             break;
159         }
160         // Get element with smallest f, remove from openNodes and
161         // add to closedNodes
162         currentNodeItr = findLowestf(openNodes);
163         currentPair = closedNodes.emplace(*currentNodeItr);
164         currentNode = *(currentPair.first);
165
166         openNodes.erase(currentNodeItr);
167         // Check if current node is goal
168         if(isNodeGoal(currentNode.first, endLocation, goalDistance
169             )) {
170             printf("FoundDest, writing open\n");
171             writeCSVfile(openNodes, "/home/nikolai/debugdune/ppopen.
172                 csv");
173             printf("FoundDest, writing closed\n");
174             writeCSVfile(closedNodes, "/home/nikolai/debugdune/
175                 ppclosed.csv");
176             printf("Tracing path\n");
177             return tracePath(currentPair.first,closedNodes);
178         }
179
180         // Get successors(in a square) and set parent location
181         //searchNodeMap successor = depthVecToSearchMap(this->
182             getWithinRadius(currentNode.first, squareSize,
183                 minDepth_m));
184         searchNodeMap successor = depthVecToSearchMap(this->
185             getSquare(currentNode.first, squareSize, minDepth_m));
186
187         // Process successors
188         for (PathPlanner::searchNodeMap::iterator itr = successor.
189             begin(); itr != successor.end(); ++itr) {
190             tentativeG = currentNode.second.g + calcG(currentNode.
191                 first, itr->first);
192             currentSuccessorNodeItr = openNodes.find(itr->first);
193             if(currentSuccessorNodeItr != openNodes.end()) { //Is

```

---

```

node already in openNodes
183     if(tentativeG < currentSuccessorNodeItr->second.g) { //
        Has the g value improved?
184         currentSuccessorNodeItr->second.parent = currentNode
            .first;
185         currentSuccessorNodeItr->second.g = tentativeG;
186         currentSuccessorNodeItr->second.f =
            currentSuccessorNodeItr->second.g + calcH(
                currentSuccessorNodeItr->first, endLocation);
187         if(!isNodeInMap(currentSuccessorNodeItr, closedNodes
            )) { //Has the node neighbour of the node been
                expanded before?
188             openNodes.emplace(*currentSuccessorNodeItr);
189         }
190     }
191 } else { // Node not in Opennodes
192     if(!isNodeInMap(itr, closedNodes)) {
193         itr->second.parent = currentNode.first;
194         itr->second.g = tentativeG;
195         itr->second.f = itr->second.g + calcH(itr->first,
            endLocation);
196         openNodes.emplace(*itr);
197     }
198 }
199 }
200 if(openNodes.empty()) { // Just for debugging
201     printf("Empty Opennodes!\n");
202     break;
203 }
204 iterations++;
205 }
206 return closedNodes;
207 }
208
209 void PathPlanner::writeCSVfile(const PathPlanner::
    searchNodeMap &inMap, const std::string &outputFile)
210 {
211     std::ofstream file_(outputFile);
212     file_ << "Lat,Lon,LatParent,LonParent,f,g" << "\r\n";
213     for (PathPlanner::searchNodeMap::const_iterator itr = inMap.
        begin(); itr != inMap.end(); ++itr)
214     {
215         file_ << std::setprecision(15) << Math::Angles::degrees(
            itr->first.Lat) << "," << Math::Angles::degrees(itr->
            first.Lon) << "," << Math::Angles::degrees(itr->second
            .parent.Lat) << "," << Math::Angles::degrees(itr->
            second.parent.Lon) << "," << itr->second.f << "," <<
            itr->second.g << "\r\n"; // << "," << itr->second.h <<
            "\r\n";
216     }

```

```
217     }
218   }
219 }
```

## D.3.6 Grounding Task

Code D.12: Task.cpp Grounding Supervisor.

```
30 // ISO C++ 98 headers.
31 #include <cstring>
32 #include <cmath>
33 #include <vector>
34 #include <string>
35 #include <fstream>
36 #include <sstream>
37
38 #include <chrono> // For timing only, can be removed
39
40 // DUNE headers.
41 #include <DUNE/DUNE.hpp>
42
43 namespace Supervisors
44 {
45     //! Insert short task description here.
46     //!
47     //! Insert explanation on task behaviour here.
48     //! @author Nikolai Lauv s & Alberto Dallolio
49     namespace Grounding
50     {
51         using DUNE_NAMESPACES;
52         struct Arguments
53         {
54             //! Path to DB file
55             std::string db_path;
56             //! Path to debug folder
57             std::string debug_path;
58             //! Map resolution.
59             double map_res;
60             //! Range around location of interest.
61             double range;
62             //! Grid size.
63             double grid_size;
64             //! Surroundings check frequency.
65             double surr_check;
66             //! GPS entity label.
67             std::string elabel_gps;
68         };
69     }
```

```

70 struct Task: public DUNE::Tasks::Task
71 {
72     //!< Debug variable.
73     int m_surr_num;
74     //!< Current Lat and Lon of vehicle.
75     double m_current_lat, m_current_lon;
76     //!< Timer.
77     Time::Counter<float> m_timer;
78     //!< GPS entity eid.
79     int m_gps_eid;
80     // Task arguments
81     Arguments m_args;
82     // Database handle.
83     SituationalAwareness::TwoDGrid* m_nc;
84
85     //!< Constructor.
86     //!< @param[in] name task name.
87     //!< @param[in] ctx context.
88     Task(const std::string& name, Tasks::Context& ctx):
89         DUNE::Tasks::Task(name, ctx),
90         m_surr_num(0),
91         m_current_lat(0.0),
92         m_current_lon(0.0),
93         m_nc(NULL)
94     {
95         param("Digital Map Path", m_args.db_path)
96             .defaultValue("")
97             .description("Path to digital map DB file");
98
99         param("Debug Path", m_args.debug_path)
100            .defaultValue("")
101            .description("Path to where debugging files are saved");
102
103         param("Digital Map Resolution", m_args.map_res)
104            .units(Units::Meter)
105            .defaultValue("")
106            .description("Digital Map resolution in meters");
107
108         param("Range Around Location", m_args.range)
109            .units(Units::Meter)
110            .defaultValue("")
111            .description("Radius [m] of circle containing queried
112                locations, around location of interest");
113
114         param("Squared Grid Size", m_args.grid_size)
115            .units(Units::Meter)
116            .defaultValue("")
117            .description("Grid Size");
118         param("Surroundings Check Frequency", m_args.surr_check)

```

---

```

119     .units(Units::Second)
120     .defaultValue("180.0")
121     .minimumValue("0.0")
122     .description("Frequency at which current vehicles
        surroundings are observed");
123
124     param("Entity Label - GPS", m_args.elabel_gps)
125     .description("Entity label of 'GpsFix' message");
126
127     bind<IMC::Abort>(this);
128     bind<IMC::PlanSpecification>(this);
129     bind<IMC::GpsFix>(this);
130
131     }
132
133     //! Update internal state with new parameter values.
134     void
135     onUpdateParameters(void)
136     {
137         if(paramChanged(m_args.surr_check))
138             m_timer.setTop(m_args.surr_check);
139     }
140
141     //! Reserve entity identifiers.
142     void
143     onEntityReservation(void)
144     {
145     }
146
147     void
148     onEntityResolution(void)
149     {
150         try
151         {
152             m_gps_eid = resolveEntity(m_args.elabel_gps);
153         }
154         catch (...)
155         {
156             m_gps_eid = 0;
157         }
158     }
159
160     //! Acquire resources.
161     void
162     onResourceAcquisition(void)
163     {
164         try {
165             m_nc = new SituationalAwareness::TwoDGrid(m_args.db_path
                , m_args.map_res);
166         } catch(std::runtime_error& e) {

```

---



```

167         inf(DTR("Problem opening charts: %s"), e.what());
168     }
169 }
170
171 ///! Initialize resources.
172 void
173 onResourceInitialization(void)
174 {
175     ///! Set timer for periodic check of surroundings.
176     m_timer.setTop(m_args.surr_check);
177 }
178
179 ///! Release resources.
180 void
181 onResourceRelease(void)
182 {
183
184     try {
185         Memory::clear(m_nc);
186     }
187     catch(std::runtime_error& e) {
188         err(DTR("Could not clear Nautical charts class: %s"), e.
189             what());
189     }
190 }
191
192
193 void
194 consume(const IMC::Abort* msg)
195 {
196     if (msg->getDestination() != getSystemId())
197         return;
198
199     if (isActive())
200         requestDeactivation();
201 }
202
203 void
204 consume(const IMC::GpsFix* msg)
205 {
206     if(msg->getSource() != getSystemId() || msg->
207         getSourceEntity() != m_gps_eid)
208         return;
209     m_current_lat=msg->lat;
210     m_current_lon=msg->lon;
211
212     if(m_timer.overflow())
213     {
214         ///! Check vehicle surroundings.

```

```

214         TwoDGrid::DepthVector a = m_nc->getWithinRadius(
                m_current_lat, m_current_lon, m_args.range);
215         for (TwoDGrid::DepthVector::iterator itr = a.begin();
                itr != a.end(); ++itr)
216             {
217                 inf("%f %f %f", itr->Lat, itr->Lon, itr->Depth);
218             }
219         //! Is just a variable for debugging m_surr_num.
220         m_nc->writeCSVfile(a, m_args.debug_path + "surroundings"
                + std::to_string(m_surr_num) + ".csv");
221         m_surr_num++;
222
223         m_timer.reset();
224     }
225 }
226
227 void
228 consume(const IMC::PlanSpecification* msg)
229 {
230     std::vector<IMC::Maneuver> maneuvers_list;
231     //Create waypoints matrix: first row has vehicle current
        position.
232     Math::Matrix waypoints(msg->maneuvers.size()+1, 2);
233     waypoints(0,0) = m_current_lat;
234     waypoints(0,1) = m_current_lon;
235
236     unsigned i=0;
237     // Iterate through plan maneuvers
238     for (std::vector<IMC::PlanManeuver*>::const_iterator itr =
        msg->maneuvers.begin(); itr != msg->maneuvers.end();
        ++itr)
239     {
240         // For now just to GoTos.
241         const IMC::Goto* m = static_cast<const IMC::Goto*>((*itr
        )->data.get());
242         //spew("LAT LON: %0.4f %0.4f", m->lat, m->lon);
243
244         waypoints(i+1,0) = m->lat;
245         waypoints(i+1,1) = m->lon;
246         i=i+1;
247     }
248     //SituationalAwareness::DensifiedVertices* m_dep = new
        SituationalAwareness::DensifiedVertices(m_args.db_path
        , 10);
249
250     for(int j=1; j<waypoints.rows(); j++)
251     {
252
253         // Depthmap based grounding check

```

```

254     spew("LAT LON destLat destLON: %0.4f %0.4f %0.4f %0.4f",
          waypoints(j-1,0), waypoints(j-1,1), waypoints(j,0),
          waypoints(j,1));
255     std::string directory = m_args.debug_path + "
          dune_transect_" + std::to_string(j) + ".csv";
256     std::pair<TwoDGrid::DepthVector, LocationData::
          LocationVector> transectcheck= m_nc->checkTransect(
          waypoints(j-1,0), waypoints(j-1,1), waypoints(j,0),
          waypoints(j,1));
257     m_nc->writeCSVfile(transectcheck.first, directory);
258
259     // For exporting the areas considered land.
260     //SituationalAwareness::LocationData* m_ld = new
          SituationalAwareness::LocationData(m_args.db_path);
261     //directory = m_args.debug_path + "dune_transect_land_"
          + std::to_string(j) + ".csv";
262     //m_ld->writeCSVfile(transectcheck.second, directory);
263
264
265     inf("%u", transectcheck.second.size());
266     if(transectcheck.second.size() !=0) {
267         inf("Aborted because of grounding (depthmap based)");
268         IMC::Abort abort;
269         abort.setDestination(getSystemId());
270         dispatch(abort);
271     } else {
272         inf("No grounding detected, maneuver approved");
273     }
274     /*
275     // DEPRE based grounding check
276     std::string directory = m_args.debug_path + "
          dune_transect_" + std::to_string(j) + ".csv";
277     DensifiedVertices::DEPREVector transectCorridor = m_dep
          ->getCorridor(waypoints(j-1,0), waypoints(j-1,1),
          waypoints(j,0), waypoints(j,1), 20);
278     m_dep->writeCSVfile(transectCorridor, directory);
279     if(!m_dep->isDepthAbove(transectCorridor, 10)) {
280         inf("Aborted because of grounding (depre based)");
281         IMC::Abort abort;
282         abort.setDestination(getSystemId());
283         dispatch(abort);
284     } else {
285         inf("No grounding detected, maneuver approved");
286     }
287     */
288 }
289 }
290
291 void
292 readCoordFromFile(std::string path)

```

```

293     {
294         std::vector<double> lat, lon;
295         std::fstream in(path);
296
297         std::string line;
298         double lat_val, lon_val;
299
300         while(std::getline(in, line))
301         {
302             std::istringstream iss(line);
303             iss >> lat_val >> lon_val;
304             lat.push_back(Angles::radians(lat_val));
305             lon.push_back(Angles::radians(lon_val));
306         }
307
308         in.close();
309     }
310
311
312
313
314     //! Main loop.
315     void
316     onMain(void)
317     {
318         while (!stopping())
319         {
320             waitForMessages(1.0);
321         }
322     }
323 };
324 }
325 }
326
327 DUNE_TASK

```

### D.3.7 ThelmaBiotel Hydrophone Task

In addition to the included code, a separate thread is used for reading the serial input. This is almost identical with the one used in DUNE's GPS task, and because so few modifications have been made to it, it's not included here. It's available in the GitHub repository. There is also a header file containing definitions of the commands supported by the TB Live. As this is not used, it has not been included here.

**Code D.13:** Task.cpp ThelmaBiotel Hydrophone Sensors.

```

30
31 //TODO: Add feature: position at 5s before tag registration in imc
    message.

```

---

```

32 // ISO C++ 98 headers.
33 #include <cstring>
34 #include <algorithm>
35 #include <cstdint>
36 #include <ctime>
37 #include <string>
38 #include <sstream>
39 #include <chrono>
40
41 // DUNE headers.
42 #include <DUNE/DUNE.hpp>
43 // #include <DUNE/Time/Clock.hpp>
44
45 // Local headers.
46 #include "Reader.hpp"
47 #include "commands.hpp"
48
49 namespace Sensors
50 {
51     // ! Device driver for ThelmaHydrophone
52     namespace ThelmaHydrophone
53     {
54         using DUNE_NAMESPACES;
55
56         // ! Maximum number of initialization commands.
57         static const unsigned c_max_init_cmds = 14;
58         // ! Timeout for waitReply() function.
59         static const float c_wait_reply_tout = 4.0;
60         // ! Power on delay.
61         static const double c_pwr_on_delay = 5.0;
62         // ! Number of fields in fish tag message
63         static const unsigned c_tag_fields = 9;
64         // ! Number of fields in TBR-700RT sensor reading
65         static const unsigned c_tbr_sensor_fields = 8;
66         // ! Message used to sync Thelma hydrophones
67         static const std::string syncString = "(+)";
68
69         struct Arguments
70         {
71             // ! Serial port device.
72             std::string uart_dev;
73             // ! Serial port baud rate.
74             unsigned uart_baud;
75             // ! Order of sentences.
76             std::vector<std::string> stn_order;
77             // ! Input timeout in seconds.
78             float inp_tout;
79             // ! Initialization commands.
80             std::string init_cmds[c_max_init_cmds];
81             // ! Initialization replies.

```

---

```

82     std::string init_rpls[c_max_init_cmds];
83     ///! Power channels.
84     std::vector<std::string> pwr_channels;
85     ///! Write full unix timestamp every timestamp_send_divider
        times task is run.
86     unsigned int timestamp_send_divider;
87     ///! Sync Period;
88     double sync_period;
89 };
90
91 struct Task: public DUNE::Tasks::Task
92 {
93     ///! Serial port handle.
94     IO::Handle* m_handle;
95     ///! Task arguments.
96     Arguments m_args;
97     ///! Last initialization line read.
98     std::string m_init_line;
99     ///! TBRReader thread.
100    TBRReader* m_TBRReader;
101    ///! How often the full unix timestamp is sent, in executions
        % counter
102    unsigned int timestamp_send_counter;
103    ///! Timer.
104    Time::Counter<float> m_sync_timer;
105    ///! Current Lat and Lon of vehicle.
106    fp64_t m_current_lat, m_current_lon;
107
108    Task(const std::string& name, Tasks::Context& ctx):
109        DUNE::Tasks::Task(name, ctx),
110        m_handle(NULL),
111        m_TBRReader(NULL),
112        m_current_lat(0.0),
113        m_current_lon(0.0)
114    {
115        // Define configuration parameters.
116        param("Serial Port - Device", m_args.uart_dev)
117            .defaultValue("")
118            .description("Serial port device used to communicate with
                the sensor");
119
120        param("Serial Port - Baud Rate", m_args.uart_baud)
121            .defaultValue("115200")
122            .description("Serial port baud rate");
123
124        param("Sync Period", m_args.sync_period)
125            .units(Units::Second)
126            .defaultValue("10.0")
127            .minimumValue("0.0")
128            .description("Period between sync messages");

```

```

129
130     param("Write full timestamp divider", m_args.
131           timestamp_send_divider)
132         .defaultValue("6")
133         .minimumValue("1")
134         .description("Write full unix timestamp every
135                       timestamp_send_divider times task is run.");
136
137     param("Input Timeout", m_args.inp_tout)
138         .units(Units::Second)
139         .defaultValue("4.0")
140         .minimumValue("0.0")
141         .description("Input timeout");
142
143     param("Power Channel - Names", m_args.pwr_channels)
144         .defaultValue("")
145         .description("Device's power channels");
146
147     param("Sentence Order", m_args.stn_order)
148         .defaultValue("")
149         .description("Sentence order");
150
151     for (unsigned i = 0; i < c_max_init_cmds; ++i)
152     {
153         std::string cmd_label = String::str("Initialization
154                                           String %u - Command", i);
155         param(cmd_label, m_args.init_cmds[i])
156             .defaultValue("");
157
158         std::string rpl_label = String::str("Initialization
159                                           String %u - Reply", i);
160         param(rpl_label, m_args.init_rpls[i])
161             .defaultValue("");
162     }
163
164     bind<IMC::DevDataText>(this);
165     bind<IMC::IoEvent>(this);
166     bind<IMC::GpsFix>(this);
167 }
168
169 //! Update internal state with new parameter values.
170 void
171 onUpdateParameters(void)
172 {
173     if (paramChanged(m_args.sync_period))
174         m_sync_timer.setTop(m_args.sync_period);
175 }
176
177 void
178 onResourceAcquisition(void)
179 {
180     if (m_args.pwr_channels.size() > 0)

```

```

175     {
176         IMC::PowerChannelControl pcc;
177         pcc.op = IMC::PowerChannelControl::PCC_OP_TURN_ON;
178         for (size_t i = 0; i < m_args.pwr_channels.size(); ++i)
179             {
180                 pcc.name = m_args.pwr_channels[i];
181                 dispatch(pcc);
182             }
183     }
184
185     Counter<double> timer(c_pwr_on_delay);
186     while (!stopping() && !timer.overflow())
187         waitForMessages(timer.getRemaining());
188
189     try
190     {
191         if (!openSocket())
192             m_handle = new SerialPort(m_args.uart_dev, m_args.
193                                     uart_baud);
194
195         m_TBRReader = new TBRReader(this, m_handle);
196         m_TBRReader->start();
197     }
198     catch (...)
199     {
200         throw RestartNeeded(DTR("1"), 5);
201     }
202
203     bool
204     openSocket(void)
205     {
206         char addr[128] = {0};
207         unsigned port = 0;
208
209         if (std::sscanf(m_args.uart_dev.c_str(), "tcp://%[^:]:%u",
210                       addr, &port) != 2)
211             return false;
212
213         TCPSocket* sock = new TCPSocket;
214         sock->connect(addr, port);
215         m_handle = sock;
216         return true;
217     }
218
219     void
220     onResourceRelease(void)
221     {
222         if (m_TBRReader != NULL)
223             {

```



```

223     m_TBRRReader->stopAndJoin();
224     delete m_TBRRReader;
225     m_TBRRReader = NULL;
226 }
227
228     Memory::clear(m_handle);
229 }
230
231 void
232 onResourceInitialization(void)
233 {
234     bool configuration_mode = false;
235     for (unsigned i = 0; i < c_max_init_cmds; ++i)
236     {
237         if (m_args.init_cmds[i].empty())
238             continue;
239         if(!configuration_mode) {
240             slowTbrSend(TBCMD_ENTER_COMMAND_MODE); //TODO:Check
                for response
241             configuration_mode = true;
242         }
243         std::string cmd = String::unescape(m_args.init_cmds[i]);
244         m_handle->writeString(cmd.c_str());
245
246         if (!m_args.init_rpls[i].empty())
247         {
248             std::string rpl = String::unescape(m_args.init_rpls[i
                ]);
249             if (!waitInitReply(rpl))
250             {
251                 err("%s: %s", DTR("no reply to command"), m_args.
                    init_cmds[i].c_str());
252                 throw std::runtime_error(DTR("failed to setup device
                    "));
253             }
254         }
255     }
256     if(configuration_mode) {
257         commandTbrSend(TBCMD_EXIT_COMMAND_MODE); //TODO:Check
            for response
258         configuration_mode = true;
259     }
260
261
262     setEntityState(IMC::EntityState::ESTA_NORMAL, Status::
        CODE_ACTIVE);
263     //! Set timer for periodic check of surroundings.
264     debug("Waiting to start timer to dividable by 10.");
265     while(std::time(0) % 10 != 0);
266     m_sync_timer.setTop(m_args.sync_period);

```

```

267     sendTbrTimestampSync();
268     debug("Finished waiting to start timer to dividable by 10.
        ");
269 }
270
271 void
272 consume(const IMC::DevDataText* msg)
273 {
274     if (msg->getDestination() != getSystemId())
275         return;
276
277     if (msg->getDestinationEntity() != getEntityId())
278         return;
279
280     spew("%s", sanitize(msg->value).c_str());
281
282     if (getEntityState() == IMC::EntityState::ESTA_BOOT)
283         m_init_line = msg->value;
284     else
285         processSentence(msg->value);
286 }
287
288 void
289 consume(const IMC::IoEvent* msg)
290 {
291     if (msg->getDestination() != getSystemId())
292         return;
293
294     if (msg->getDestinationEntity() != getEntityId())
295         return;
296
297     if (msg->type == IMC::IoEvent::IOV_TYPE_INPUT_ERROR)
298         throw RestartNeeded(msg->error, 5);
299 }
300
301 void
302 consume(const IMC::GpsFix* msg)
303 {
304     if (msg->getSource() != getSystemId())
305         return;
306     m_current_lat=msg->lat;
307     m_current_lon=msg->lon;
308     //spew("Received position: %f %f \n", m_current_lat,
        m_current_lon);
309 }
310
311 //! Wait reply to initialization command.
312 //! @param[in] stn string to compare.
313 //! @return true on successful match, false otherwise.
314 bool

```

```

315     waitInitReply(const std::string& stn)
316     {
317         Counter<float> counter(c_wait_reply_tout);
318         while (!stopping() && !counter.overflow())
319             {
320                 waitForMessages(counter.getRemaining());
321                 if (m_init_line == stn)
322                     {
323                         m_init_line.clear();
324                         return true;
325                     }
326             }
327
328     return false;
329 }
330
331 uint8_t calcLuhnVerifDigit(uint32_t timestamp) // From TB
        Live datasheet, fw1.0.1 rev.1
332 {
333     uint16_t digitSum = 0;
334     uint32_t digit = 0;
335     for(uint8_t i = 0; i < 9; i ++ ) {
336         timestamp /= 10;
337         digit = timestamp % 10;
338         if( (i % 2) == 0 ) {
339             digit *= 2;
340         }
341         if(digit > 9) {
342             digit -= 9 ;
343         }
344         digitSum += digit;
345     }
346     uint8_t luhnsCheckDigit = ( digitSum * 9) % 10;
347     return luhnsCheckDigit;
348 }
349
350 void sendTbrTimestampSync() {
351     // Get timestamp from system clock
352     std::time_t timestamp = std::time(nullptr);
353     // Remove last digit
354     std::string UTCUnixTimestamp = std::to_string(timestamp
        /10);
355     // Add Luhn verification number
356     UTCUnixTimestamp += std::to_string(calcLuhnVerifDigit(
        timestamp));
357     // Add preamble and send
358     slowTbrSend(syncString + UTCUnixTimestamp);
359 }
360
361 void sendTbrSync() {

```

```

362     slowTbrSend(syncString);
363 }
364 void commandTbrSend(std::string cmd) {
365     m_handle->write(cmd.c_str(), cmd.size());
366 }
367
368 void slowTbrSend(std::string cmd) {
369     // Send to uart slowly, because ThelmaHydrophone processes
370     // max 1 char per millisecond
371     char a[1] = {'0'};
372     for(char& c : cmd) {
373         a[0] = c;
374         m_handle->write(a, 1);
375         Delay::waitMsec(1);
376     }
377     spew(DTR("Sent: \"%s\" at \"%ld\"", cmd.c_str(), std::
378         time(0)));
379 }
380
381 ///! Read int from input string.
382 ///! @param[in] str input string.
383 ///! @param[out] dst number.
384 ///! @return true if successful, false otherwise.
385 bool readIntFromString(const std::string& str, int& dst) {
386     try {
387         dst = std::stoi(str);
388         return true;
389     }
390     catch (const std::invalid_argument& ia) {
391         err(DTR("Invalid argument: %s"), ia.what());
392         return false;
393     }
394     return true;
395 }
396
397 ///! Process sentence.
398 ///! @param[in] line line.
399 void
400 processSentence(const std::string& line)
401 {
402     spew(DTR("Process"));
403     if (line.find("ack01") != std::string::npos) {
404         spew(DTR("Sensor clock diciplined"));
405     }
406     if (line.find("ack02") != std::string::npos) {
407         spew(DTR("Sensor timestamp set"));
408     }
409     if (line.find("$") != std::string::npos) {
410         // Discard leading noise.
411         size_t sidx = 0;
412         for (sidx = 0; sidx < line.size(); ++sidx)

```

```

410     {
411         if (line[sidx] == '$')
412             break;
413     }
414
415     // Split sentence
416     std::vector<std::string> parts;
417     try {
418         spew(DTR("try"));
419         String::split(line.substr(sidx + 1, line.size()), ",",
420             parts);
421     } catch(const std::exception& ex) {
422         err(DTR("Invalid argument: %s"), ex.what());
423         return;
424     }
425     interpretSentence(parts);
426 }
427 }
428
429 /// Interpret given sentence.
430 /// @param[in] parts vector of strings from sentence.
431 void
432 interpretSentence(std::vector<std::string>& parts)
433 {
434     spew(DTR("Interpret"));
435     /*if (parts[0] == m_args.stn_order.front())
436     {
437         // Test if all sentences received, TODO, can probably be
438         // removed
439     }*/
440     if(parts.size() >= 3) {
441         if(parts[2] == "TBR Sensor") {
442             interpretSensorReading(parts);
443         } else {
444             interpretTagDetection(parts);
445         }
446     }
447 }
448 /// Interpret SensorReading sentence.
449 /// @param[in] parts vector of strings from sentence.
450 void
451 interpretSensorReading(const std::vector<std::string>& parts
452     ) {
453     spew(DTR("Interpret SensorReading"));
454     if (parts.size() < c_tbr_sensor_fields)
455     {
456         war(DTR("invalid SensorReading sentence"));
457         return;

```

```

457     }
458
459     int serial_no = 0;
460     int unix_timestamp = 0;
461     int temperature = 0;
462     int avg_noise_level = 0;
463     int peak_noise_level = 0;
464     int rcv_listen_freq = 0;
465     int rcv_mem_addr = 0;
466     float temp_C = 0.0;
467
468     if (readIntFromString(parts[0], serial_no))
469     {
470         // Receiver serial number
471         spew(DTR("Serial number: %u"), serial_no);
472     }
473     if (readIntFromString(parts[1], unix_timestamp))
474     {
475         //UTC UNIX timestamp
476         spew(DTR("UTC UNIX timestamp: %u"), unix_timestamp);
477     }
478     if (readIntFromString(parts[3], temperature))
479     {
480         // Temperature
481         temp_C = float(temperature-50)/10.0;
482         spew(DTR("Temperature (C): %f"), temp_C);
483         IMC::Temperature temp_msg;
484         temp_msg.setSourceEntity(255);
485         temp_msg.value = fp32_t(temp_C);
486         dispatch(temp_msg);
487     }
488     if (readIntFromString(parts[4], avg_noise_level))
489     {
490         // Average Noise Level
491         spew(DTR("Average Noise Level: %u"), avg_noise_level);
492     }
493     if (readIntFromString(parts[5], peak_noise_level))
494     {
495         // Peak noise level
496         spew(DTR("Peak noise level: %u"), peak_noise_level);
497     }
498     if (readIntFromString(parts[6], rcv_listen_freq))
499     {
500         // Noise logging frequency
501         spew(DTR("Noise logging frequency: %u"),
502             rcv_listen_freq);
503     }
504     if (readIntFromString(parts[7], rcv_mem_addr))
505     {
506         // Receiver memory address

```

```

506     spew(DTR("Receiver memory address: %u"), recv_mem_addr);
507 }
508 IMC::TBRSensor sensor_msg;
509 sensor_msg.serial_no = serial_no;
510 sensor_msg.unix_timestamp = unix_timestamp;
511 sensor_msg.temperature = fp32_t(temp_C);
512 sensor_msg.avg_noise_level = avg_noise_level;
513 sensor_msg.peak_noise_level = peak_noise_level;
514 sensor_msg.recv_listen_freq = recv_listen_freq;
515 sensor_msg.recv_mem_addr = recv_mem_addr;
516 dispatch(sensor_msg);
517 }
518 /// Interpret fishtag sentence.
519 /// @param[in] parts vector of strings from sentence.
520 void
521 interpretTagDetection(const std::vector<std::string>& parts)
522 {
523     spew(DTR("Interpret tag"));
524     if (parts.size() < c_tag_fields)
525     {
526         war(DTR("invalid tag sentence"));
527         return;
528     }
529
530     int serial_no = 0;
531     int unix_timestamp = 0;
532     int millis = 0;
533     int trans_protocol = 0;
534     int trans_id = 0;
535     int trans_data = 0;
536     int SNR = 0;
537     int trans_freq = 0;
538     int recv_mem_addr = 0;
539
540     if (readIntFromString(parts[0], serial_no)
541     {
542         // Receiver serial number
543         spew(DTR("Serial number: %u"), serial_no);
544     }
545     if (readIntFromString(parts[1], unix_timestamp)
546     {
547         //UTC UNIX timestamp
548         spew(DTR("UTC UNIX timestamp: %u"), unix_timestamp);
549     }
550     if (readIntFromString(parts[2], millis)
551     {
552         //Millisecond timestamp
553         spew(DTR("Millisecond timestamp: %u"), millis);
554     }
555

```

```

556 //Transmit protocol
557 if(parts[3] == "R256")
558     trans_protocol = IMC::TBRFishTag::TBR_R256;
559 else if(parts[3] == "R04K")
560     trans_protocol = IMC::TBRFishTag::TBR_R04K;
561 else if(parts[3] == "S64K")
562     trans_protocol = IMC::TBRFishTag::TBR_S64K;
563 else if(parts[3] == "R64K")
564     trans_protocol = IMC::TBRFishTag::TBR_R64K;
565 else if(parts[3] == "R01M")
566     trans_protocol = IMC::TBRFishTag::TBR_R01M;
567 else if(parts[3] == "S256")
568     trans_protocol = IMC::TBRFishTag::TBR_S256;
569 else if(parts[3] == "HS256")
570     trans_protocol = IMC::TBRFishTag::TBR_HS256;
571 else if(parts[3] == "DS256")
572     trans_protocol = IMC::TBRFishTag::TBR_DS256;
573 spew(DTR("Transmit protocol: %s, enum: %i"), parts[3].
        c_str(), trans_protocol);

574
575
576 if (readIntFromString(parts[4], trans_id))
577 {
578     // Tag ID number
579     spew(DTR("Tag ID: %u"), trans_id);
580 }
581 if (readIntFromString(parts[5], trans_data))
582 {
583     // Tag raw data
584     spew(DTR("Tag raw data: %u"), trans_data);
585 }
586 if (readIntFromString(parts[6], SNR))
587 {
588     // Signal to noise ratio
589     spew(DTR("SNR: %u"), SNR);
590 }
591 if (readIntFromString(parts[7], trans_freq))
592 {
593     // Signal frequency
594     spew(DTR("Signal frequency: %u"), trans_freq);
595 }
596 if (readIntFromString(parts[8], recv_mem_addr))
597 {
598     // Receiver memory address
599     spew(DTR("Receiver memory address: %u"), recv_mem_addr);
600 }
601 IMC::TBRFishTag tag_msg;
602 tag_msg.serial_no = serial_no;
603 tag_msg.unix_timestamp = unix_timestamp;
604 tag_msg.millis = millis;

```



```

605     tag_msg.trans_protocol = trans_protocol;
606     tag_msg.trans_id = trans_id;
607     tag_msg.trans_data = trans_data;
608     tag_msg.snr = SNR;
609     tag_msg.trans_freq = trans_freq;
610     tag_msg.recv_mem_addr = recv_mem_addr;
611     tag_msg.lat = m_current_lat;
612     tag_msg.lon = m_current_lon;
613     dispatch(tag_msg);
614 }
615
616 void
617 onMain(void)
618 {
619     while(!stopping()) {
620         if(m_sync_timer.overflow())
621         {
622             m_sync_timer.reset();
623
624             if(timestamp_send_counter >= m_args.
625                 timestamp_send_divider) {
626                 sendTbrTimestampSync();
627                 timestamp_send_counter = 0;
628             } else {
629                 sendTbrSync();
630             }
631             //spew("C: %ld", std::time(0));
632             spew("Sending duration: %f", m_sync_timer.getElapsed()
633                 );
634             timestamp_send_counter++;
635         }
636         consumeMessages();
637     }
638 };
639 }
640
641 DUNE_TASK

```

### D.3.8 Specific Otter Configuration File

Code D.14: ntnu-otter-04.ini

```

31
32 [Require basic.ini]
33
34 #####

```

```

35 | # General Parameters. #
36 | #####
37 | [IMC Addresses]
38 | ntnu-otter-01 = 0x2810
39 | ntnu-otter-02 = 0x2811
40 | ntnu-otter-03 = 0x2812
41 | ntnu-otter-04 = 0x2813
42 |
43 | [General]
44 | Vehicle = ntnu-otter-04
45 | Vehicle Type = asv
46 | Speed Conversion -- Actuation = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
    |     0.7, 0.8, 0.9, 1.0
47 | Speed Conversion -- RPM = 0.0, 120, 245, 360, 490, 615, 725, 845,
    |     980, 980, 980
48 | Speed Conversion -- MPS = 0.0, 1.0, 1.3, 1.6, 1.9, 2.2, 2.5, 2.8,
    |     3.09, 3.09, 3.09
49 | Absolute Maximum Depth = 0
50 | Time Of Arrival Factor = 5.0

```

### D.3.9 Common Otter Configuration File

Code D.15: basic.ini

```

31 |
32 | [Require ../common/imc-addresses.ini]
33 | [Require ../common/transport.ini]
34 | [Require ../common/maneuvers.ini]
35 |
36 | [Profiles]
37 | StratoPi = Special simulation mode where only the hardware
    |     features of the StratoPi are active
38 |
39 | #####
40 | # General Parameters. #
41 | #####
42 |
43 | [Transports.Announce]
44 | Enabled = Always
45 | Entity Label = Announce
46 | Announcement Periodicity = 10
47 | Enable Loopback = 1
48 | Enable Multicast = 1
49 | Enable Broadcast = 1
50 | Multicast Address = 224.0.75.69
51 | Ports = 30100, 30101, 30102, 30103, 30104
52 | System Type = USV
53 |

```

---

```

54 [Transports.Discovery]
55 Enabled = Always
56 Entity Label = Discovery
57 Multicast Address = 224.0.75.69
58 Ports = 30100, 30101, 30102, 30103, 30104
59
60 #[Transports.Logging]
61 #Flush Interval = 0.5
62
63 #####
64 # Navigation. #
65 #####
66
67 [Navigation.General.GPSNavigation]
68 Enabled = Always
69 Entity Label = Navigation
70 Entity Label - GPS = GPS
71 Entity Label - Yaw = GPS
72
73 #####
74 # Control. #
75 #####
76
77 [Control.ASV.HeadingAndSpeed]
78 Enabled = Always
79 Entity Label = Course & Speed Controller
80 Debug Level = None
81 Maximum Thrust Actuation = 1.0
82 Maximum Thrust Differential Actuation = 0.4
83 Ramp Actuation Limit = 0.0
84 Hardware RPMs Control = true
85 RPMs at Maximum Thrust = 1100
86 RPMs PID Gains = 0.2e-3, 0.21e-3, 29.0e-6
87 RPMs Feedforward Gain = 0.46e-3
88 MPS PID Gains = 1.0, 25.0, 0.0
89 MPS Integral Limit = 200.0
90 MPS Feedforward Gain = 100.0
91 Minimum RPM Limit = 62
92 Maximum RPM Limit = 1100
93 Maximum RPM Acceleration = 62
94 Yaw PID Gains = 1.5, 0.0, 0.0
95 Maximum Heading Error to Thrust = 30.0
96 Entity Label - Port Motor = Torqeedo - Motor 0
97 Entity Label - Starboard Motor = Torqeedo - Motor 1
98 Share Saturation = true
99 Log PID Parcels = true
100
101 [Control.Path.PurePursuit]
102 Enabled = Never
103 Entity Label = Path Control

```

---

---

```
104 |
105 |
106 | #Integral line of sight
107 | [Control.Path.ILOS]
108 | Enabled = Always
109 | Entity Label = Path Control
110 | Debug Level = None
111 | Control Frequency = 10
112 | Along-track -- Monitor      = true
113 | Along-track -- Check Period = 20
114 | Along-track -- Minimum Speed = 0.05
115 | Along-track -- Minimum Yaw = 2
116 | Cross-track -- Monitor = true
117 | Cross-track -- Nav. Unc. Factor = -1
118 | Cross-track -- Distance Limit = 25
119 | Cross-track -- Time Limit = 20
120 | Position Jump Threshold = 10.0
121 | Position Jump Time Factor = 0.5
122 | ETA Minimum Speed = 0.1
123 | New Reference Timeout = 5.0
124 | Course Control = false
125 | Corridor -- Width = 1.5
126 | Corridor -- Entry Angle = 15.0
127 | Corridor -- Out Vector Field = true
128 | Corridor -- Out LOS = false
129 | ILOS Lookahead Distance = 4.0
130 | ILOS Integrator Gain = 0.5
131 | ILOS Integrator Initial Value = 0.0
132 | Bottom Track -- Enabled = false
133 |
134 | [Control.Path.VectorField]
135 | Enabled = Never
136 | Entity Label = Path Control
137 | Debug Level = None
138 | ETA Minimum Speed = 0.1
139 | Control Frequency = 10
140 | Along-track -- Monitor = false
141 | Along-track -- Check Period = 20
142 | Along-track -- Minimum Speed = 0.05
143 | Along-track -- Minimum Yaw = 2
144 | Cross-track -- Monitor = false
145 | Cross-track -- Nav. Unc. Factor = 1
146 | Cross-track -- Distance Limit = 25
147 | Cross-track -- Time Limit = 20
148 | Position Jump Threshold = 10.0
149 | Position Jump Time Factor = 0.5
150 | ETA Minimum Speed = 0.1
151 | New Reference Timeout = 5.0
152 | Course Control = false
153 | Corridor -- Width = 2.5
```

---

```
154 Corridor -- Entry Angle = 15.0
155 Extended Control -- Enabled = false
156 Extended Control -- Controller Gain = 1.0
157 Extended Control -- Turn Rate Gain = 1.0
158 Bottom Track -- Enabled = false
159 Bottom Track -- Forward Samples = 7
160 Bottom Track -- Safe Pitch = 35.0
161 Bottom Track -- Minimum Range = 4.0
162 Bottom Track -- Slope Hysteresis = 1.5
163 Bottom Track -- Check Trend = false
164 Bottom Track -- Execution Frequency = 5
165 Bottom Track -- Depth Avoidance = true
166 Bottom Track -- Admissible Altitude = 2.5
167
168
169 [Control.ASV.RemoteOperation]
170 Enabled = Always
171 Entity Label = Remote Control
172 Active = true
173 Active - Scope = maneuver
174 Active - Visibility = developer
175 Execution Frequency = 10
176 Connection Timeout = 2.0
177
178 #####
179 # Maneuvers. #
180 #####
181
182 # Can this be removed?
183 [Maneuver.FollowReference.AUV]
184 Enabled = Always
185 Entity Label = Follow Reference Maneuver
186 Horizontal Tolerance = 15.0
187 Vertical Tolerance = 1.0
188 Loitering Radius = 7.5
189 Default Speed = 50
190 Default Speed Units = percent
191 Default Z = 0
192 Default Z Units = DEPTH
193
194 [Maneuver.RowsCoverage]
195 Enabled = Always
196 Entity Label = Rows Coverage Maneuver
197
198
199 #####
200 # Monitors / Supervisors #
201 #####
202
203 [Monitors.Clock]
```

---

```
204 Enabled = Never
205 Entity Label = Clock
206 Minimum GPS Fixes = 30
207 Maximum Clock Offset = 2
208 Boot Synchronization Timeout = 60
209 Hardware Clock Synchronization Command = hwclock -w
210
211 [Monitors.Entities]
212 Enabled = Always
213 Entity Label = Entity Monitor
214 Activation Time = 0
215 Deactivation Time = 0
216 Debug Level = None
217 Execution Priority = 10
218 Report Timeout = 5
219 Transition Time Gap = 4.0
220 Maximum Consecutive Transitions = 3
221 Default Monitoring = Daemon,
222                               GPS,
223                               Navigation,
224                               Path Control
225 Default Monitoring -- Hardware = Torqeedo
226
227 [Supervisors.Vehicle]
228 Enabled = Always
229 Entity Label = Vehicle Supervisor
230 Activation Time = 0
231 Deactivation Time = 0
232 Debug Level = None
233 Execution Priority = 10
234 Execution Frequency = 2
235 Allows External Control = false
236 Maneuver Handling Timeout = 1.0
237
238 [Supervisors.AUV.LostComms]
239 Enabled = Never
240 Entity Label = LostComms Supervisor AUV
241 Plan Name = lost_comms
242 Lost Comms Timeout = 10.0
243 Debug Level = Spew
244
245 #####
246 # Hardware. #
247 #####
248
249 [Actuators.Torqeedo]
250 Enabled = Hardware, StratoPi
251 Execution Frequency = 40
252 Debug Level = Spew
253 Entity Label = Torqeedo
```

```

254 CAN Port - Device = can0
255 Power Channel H_MOT0 - Name = Starboardmotor_pwr
256 Power Channel H_MOT0 - State = 1
257 Power Channel H_MOT1 - Name = Portmotor_pwr
258 Power Channel H_MOT1 - State = 1
259 Power Channel H_VR0 - Name = Signal_Light
260 Power Channel H_VR0 - State = 1
261 Power Channel H_5V - Name = Hydrophone
262 Power Channel H_5V - State = 1
263 Motor write divider = 10
264
265 [Safety.StratoPIWatchdog]
266 Enabled = Hardware, StratoPi
267 Entity Label = Watchdog
268 Execution Frequency = 0.5
269 TimeToggled = 0.25
270 Debug Level = Spew
271
272 # To use with the signal light
273 # The Identifiers are separated by commas, so more can be
    implemented easily
274 # The patterns are given first by on/off(0/1) for each led,
    followed by how long in millis. The pattern loops/repeats.
275
276 [UserInterfaces.LEDs]
277 Enabled = Hardware, StratoPi
278 Entity Label = Signal Light
279 Interface = GPIO
280 Identifiers = 26
281 Critical Entities = Logger
282 Pattern - Normal = 1, 2000, 0, 2000
283 Pattern - Fuel Low = 1, 200, 0, 200, 1, 200, 0, 2000
284 Pattern - Plan Starting = 1, 200, 0, 2000
285 Pattern - Plan Executing = 1, 500, 0, 500
286 Pattern - Error = 1, 200, 0, 2000
287 Pattern - Fatal Error = 1, 200, 0, 2000
288 Pattern - Shutdown = 1, 200, 0, 2000
289
290 [Sensors.GPS]
291 Enabled = Hardware
292 Entity Label = GPS
293 Serial Port - Device = /dev/ttyUSB0
294 Serial Port - Baud Rate = 19200
295 Sentence Order = GPHDT, GPROT, GPHDM, GPGGA, GPVTG, GPZDA
296 Debug Level = Spew
297 Initialization String 0 - Command = $JASC,GPGGA,1\r\n
298 Initialization String 1 - Command = $JASC,GPVTG,1\r\n
299 Initialization String 2 - Command = $JASC,GPZDA,1\r\n
300 Initialization String 3 - Command = $JATT,NMEAHE,0\r\n
301 Initialization String 4 - Command = $JASC,GPROT,1\r\n

```

---

```
302 Initialization String 5 - Command = $JASC,GPHDT,1\r\n
303 Initialization String 6 - Command = $JASC,GPHDM,1\r\n
304 Initialization String 7 - Command = $JSAVE\r\n
305
306 [Sensors.TBR700RT]
307 Enabled = Hardware, StratoPi
308 Debug Level = Spew
309 Entity Label = Hydrophone
310 Serial Port - Device = /dev/ttyAMA0
311 Serial Port - Baud Rate = 115200
312
313 #####
314 # Simulators. #
315 #####
316
317 [Require ../common/vsim-models.ini]
318
319 # Vehicle simulator.
320 [Simulators.VSIM]
321 Enabled = Simulation, StratoPi
322 Entity Label = Simulation Engine
323 Execution Frequency = 25
324
325 # GPS simulator.
326 [Simulators.GPS]
327 Enabled = Simulation, StratoPi
328 Execution Frequency = 1
329 Entity Label = GPS
330 Number of Satellites = 9
331 HACC = 2
332 HDOP = 0.9
333 Activation Depth = 0.2
334 Report Ground Velocity = false
335 Report Yaw = false
336 Initial Position = 63.33, 10.083333
337
338 # Port motor.
339 [Simulators.Motor/Port]
340 Enabled = Simulation, StratoPi
341 Entity Label = Motor 0
342 Execution Frequency = 20
343 Thruster Act to RPM Factor = 50, 1100.0
344 Thruster Id = 0
345
346 # Starboard motor.
347 [Simulators.Motor/Starboard]
348 Enabled = Simulation, StratoPi
349 Entity Label = Motor 1
350 Execution Frequency = 20
351 Thruster Act to RPM Factor = 50, 1100.0
```



---

```

352 Thruster Id = 1
353
354 #####
355 # Transports. #
356 #####
357
358 [Transports.UDP]
359 Enabled = Always
360 Entity Label = UDP
361 Debug Level = None
362 Activation Time = 0
363 Deactivation Time = 0
364 Execution Priority = 10
365 Announce Service = true
366 Contact Refresh Periodicity = 5.0
367 Contact Timeout = 30
368 Dynamic Nodes = true
369 Local Messages Only = false
370 Transports = Acceleration,
371                               AngularVelocity,
372                               AutopilotMode,
373                               ControlParcel,
374                               CpuUsage,
375                               Current,
376                               DesiredPath,
377                               DesiredRoll,
378                               DesiredSpeed,
379                               DesiredVerticalRate,
380                               DesiredZ,
381                               EntityList,
382                               EntityParameters,
383                               EntityState,
384                               EstimatedState,
385                               EstimatedStreamVelocity,
386                               EulerAnglesDelta,
387                               FollowRefState,
388                               FuelLevel,
389                               GpsFix,
390                               GpsNavData,
391                               Heartbeat,
392                               IndicatedSpeed,
393                               LeaderState,
394                               LinkLevel,
395                               LogBookControl,
396                               LoggingControl,
397                               MagneticField,
398                               OperationalLimits,
399                               PathControlState,
400                               PlanControl,
401                               PlanControlState,

```

---

---

```
402 PlanDB,
403 PlanGeneration,
404 PlanSpecification,
405 PowerChannelControl,
406 Pressure,
407 QueryEntityParameters,
408 RemoteActions,
409 RemoteActionsRequest,
410 Rpm,
411 RSSI,
412 SaveEntityParameters,
413 SetEntityParameters,
414 SetServoPosition,
415 SetThrusterActuation,
416 SimulatedState,
417 StorageUsage,
418 Target,
419 TBRFishTag,
420 TBRSensor,
421 Temperature,
422 TrexOperation,
423 TrueSpeed,
424 VehicleMedium,
425 VehicleState,
426 VelocityDelta,
427 Voltage
428 Filtered Entities = CpuUsage:Daemon
429 Local Port = 6002
430 Print Incoming Messages = 0
431 Print Outgoing Messages = 0
432 Rate Limiters = CpuUsage:1,
433 EntityState:1,
434 EstimatedState:10,
435 FuelLevel:0.1,
436 SimulatedState:0.5,
437 StorageUsage:0.05,
438 Acceleration:10,
439 AngularVelocity:10,
440 MagneticField:10,
441 Temperature:10,
442 Pressure:10,
443 EulerAnglesDelta:10,
444 VelocityDelta:10
445
446 [Transports.Logging]
447 Enabled = Always
448 Entity Label = Logger
449 Flush Interval = 5
450 LSF Compression Method = gzip
451 Transports = Acceleration,
```

---

452	AngularVelocity,
453	Announce,
454	AutopilotMode,
455	ControlLoops,
456	ControlParcel,
457	CpuUsage,
458	Current,
459	DesiredHeading,
460	DesiredPath,
461	DesiredRoll,
462	DesiredSpeed,
463	DesiredVerticalRate,
464	DesiredZ,
465	DevCalibrationControl,
466	EntityList,
467	EntityState,
468	EstimatedState,
469	EstimatedStreamVelocity,
470	EulerAnglesDelta,
471	FollowReference,
472	FollowRefState,
473	FuelLevel,
474	GpsFix,
475	GpsNavData,
476	IndicatedSpeed,
477	LeaderState,
478	LinkLevel,
479	LogBookEntry,
480	MagneticField,
481	ManeuverControlState,
482	PathControlState,
483	PlanControl,
484	PlanSpecification,
485	PlanControlState,
486	PlanDB,
487	PowerChannelControl,
488	Pressure,
489	Reference,
490	Rpm,
491	RSSI,
492	SetControlSurfaceDeflection,
493	SetThrusterActuation,
494	SimulatedState,
495	StopManeuver,
496	StorageUsage,
497	Temperature,
498	TBRFishTag,
499	TBRSensor,
500	TrueSpeed,
501	TrexObservation,

---

502	TrexPlan,
503	TrexToken,
504	TrueSpeed,
505	VehicleCommand,
506	VehicleMedium,
507	VehicleState,
508	VelocityDelta,
509	Voltage