Application Programming Interface for an Autonomous Surface Vessel

by

Sivert Stensrud, Sigurd Aleksander Salomonsen Gresberg, Lars Skogen Johnsen

Report submitted in fulfillment of the requirements for the course ELVE3610 - Introduction to Robotics

Oslo Metropolitan University - OsloMet

Faculty of Technology, Art, and Design

Institute of Mechanical, Electrical, and Chemical Engineering

April 18, 2024

Abstract

This project paper presents the development of an application programming interface for the autonomous surface vessel "Otter". The work primarily focused on data handling between a shore station and the USV through a socket connection, allowing control of the vessels movement, while gathering data on its movement and orientation. A graphical user interface was created with a map and control functions, allowing for data-gathering and interaction with the vessel. The paper discusses the theoretical and technical aspects of the development of the application programming interface, including difficulties in integration of specific functions, while achieving the goal of creating a functional interface accessible for other research purposes.

Group members' contributions will be found in Appendix B

Acronyms: Unmanned surface vessel (USV); Graphical user interface (GUI); Application Programming Interface (API); Guidance, navigation and control systems (GNC); Vehicle control station (VCS); Degrees of freedom (DOF); On-Board system control box (OBS); Inertial Measurement Unit (IMU); Global Positioning System (GPS); Global Navigation Satellite Systems (GNSS); American Standard Code for Information Interchange (ASCII); North, East, Down (NED).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objective

This project details the development of an Application programming interface (API) and a Graphical user interface (GUI) for the unmanned surface vessel Otter (USV), depicted in figure 1.1. This interface will include a developer mode where the user easily can update and view essential values of the vessel kinematics, while also being capable of manually adjusting thruster settings from the connected shore station. In the GUI a real-time map should also be included, displaying the Otter.

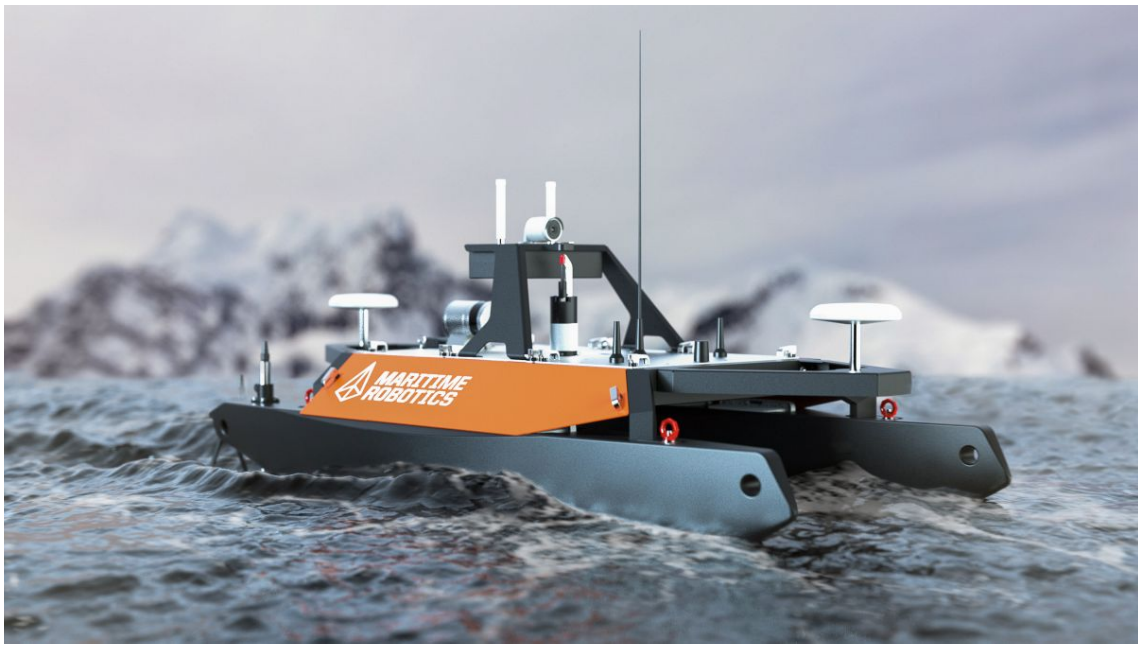


Figure 1.1: Otter USV

To accomplish this a socket communication must be established with the Otter, where the API will process delivered data to and from a shore station.

The objective of this project is a more in-depth understanding of socket communications and the three pillars of Guidance, Navigation and Control (GNC). The aim is to improve on the Otters navigation by incorporating the data achieved through a connection with the Otter, together with the rotation matrix to decide the heading and world coordinates relative to a home position.

1.2 Goals

1.2.1 Real-time wireless communication with the vessel

Sub-goals

- Create a wireless TCP/IP socket comminication link over Wifi/Radio/4G with the Otter for transmitting data between the Otter and shore station.
- Establish an application programming interface that:
  - Initiates and closes the socket communication with the Otter
  - Updates and saves values as variables to be accessed when needed

1.2.2 Manual control of Otter through user interface

Sub-goals

- Create functions for manually controlling thrusters in the API and GUI with:
  - a Manual steering mode with numeric user inputs for thruster forces
  - Drift mode that stops thrusters while continuing towards heading with remaining velocity
- Make a graphical user interface for:
  - Input of changes to thruster controls
  - Use of functions from the API to connect, disconnect and/or update values
  - Displaying tables of relevant values from the API

1.2.3 Real-time map

- Create a live map that continuously updates the position of Otter in the local environment

Chapter 2

Background

The use of USV's has for long been limited the due to technological constraints(Liu et al., 2016). Semi-automated USV's are still more frequent today considering the difficulties in automation with reliable GNC systems, making it challenging to compete with the functionality of manned surface vessels. However big advancements have been made in the last few decades in regards to both military and commercial capabilities (Barrera et al., 2021). There have been significant improvements not only in the precision of which the USV's can be remotely managed, but also in the ranges they can travel, the capacity of payloads and the variety of functions the vessels can perform.

These advancements have led to a range of useful applications including inspections, seabed mapping, research, and search and rescue missions (Beaubouef, 2023). USV's also have the capability of being a cost-efficient way of navigating hazardous and cluttered environments, while improving personnel safety. In 2021, Yara International ASA produced the first fully electric autonomous cargo vessel in the world (Deshayes, 2021), displaying the modern capabilities of USV's to transport heavy payloads, albeit over short distances. The use of USV's has steadily accelerated the last decade due to the improvements done to GNC systems, and it is expected that the amount and quality of unmanned vessels will keep increasing in the future.

Chapter 3

Theory

3.1 Rotation matrices

The SNAME notation, from the Society of Naval Architects and Marine Engineers (SNAME, 1950) as shown in table 3.1, is a standard notation used to define the 6 degrees of freedom (DOF) for marine vessels.

Table 3.1: Marine vessel motion notation

DOF	Motion	Force & Moment	Linear & Angular Velocity	Position & Euler Angle
1	Surge	X	u	x
2	Sway	Y	v	y
3	Yaw	N	r	ψ
4	Roll	K	P	ϕ
5	Pitch	M	q	θ
6	Heave	Z	w	z

As discussed by Fossen, 2011, p.15-56, Fossen and Sagatun, 1991, p.19, and Setiawan et al., 2022, the dynamic forces working on a small marine craft can be modelled in 6 DOF by Equation 3.1, where $\boldsymbol{M}$, $\boldsymbol{C(v)}$ and $\boldsymbol{D(v)}$ sequentially denote the inertia, Coriolis and damping matrices of the system, with $\boldsymbol{\tau}$ as torque and $\boldsymbol{g(\eta)}$ as the vector of gravitational and buoyant forces.

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) + g_0 = \tau + \tau_{\text{wind}} + \tau_{\text{wave}} \tag{3.1}$$

While neglecting wind and wave loads, hydrodynamic drag and ocean current, the state vector of the USV's kinematics can be written as $\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{x} & \boldsymbol{y} & \boldsymbol{z} & \boldsymbol{\phi} & \boldsymbol{\theta} & \boldsymbol{\psi} \end{bmatrix}^{\mathrm{T}}$, visualized for 6 DOF in figure 3.1. Simplified for the vessel in 3 DOF this is rewritten to $\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{x} & \boldsymbol{y} & \boldsymbol{\psi} \end{bmatrix}^{\mathrm{T}}$, which represents the position $\boldsymbol{x}$ and $\boldsymbol{y}$, and heading $\boldsymbol{\psi}$, for the USV in reference to the Earth frame.

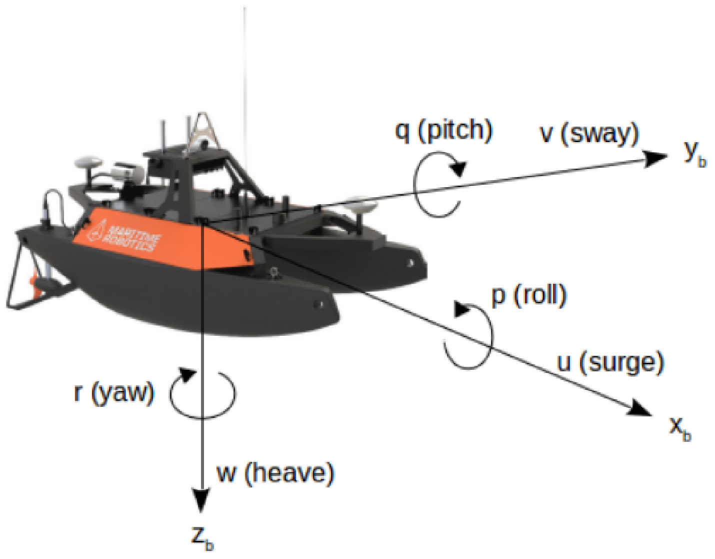


Figure 3.1: Otter USV with 6 degrees of freedom

With the heading $\boldsymbol{\psi}$, the rotation matrix for the body-frame to the Earth-fixed frame is defined by Equation (3.2) (Saksvik et al., 2022).

$$\boldsymbol{R}(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \in SO(3) \tag{3.2}$$

Continuing Fossen, 2011's notation, solving this equation in regards to the linear and angular velocity displays the movement in the x and y directions as shown in Equation (3.3)-(3.6)

$$\begin{bmatrix} x \\ y \\ \psi \end{bmatrix} = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \tag{3.3}$$

$$x = u\cos(\psi) - v\sin(\psi) \tag{3.4}$$

$$y = u\sin(\psi) + v\cos(\psi) \tag{3.5}$$

$$\psi = r \tag{3.6}$$

Where x and y is the respective movement in north and east direction, and the resultant velocity can be found by Equation (3.7).

$$V = \sqrt{u^2 + v^2} \tag{3.7}$$

3.2 Control Allocation

Control allocation refers to determining the distribution of control signals to relevant actuators, which for the Otter USV refers to two non-rotatable stern thrusters. Following Fossen, 2011, p.405-413 and Bjering Strand, 2020, p.17-18, while neglecting wind $\boldsymbol{\tau}_{\text{wind}}$ and waves $\boldsymbol{\tau}_{\text{wave}}$ the control forces can be calculated with Equation (3.8).

$$\tau = TKu \tag{3.8}$$

For the control forces, **K** is a diagonal matrix of thrust coefficients and **T** the actuator configuration matrix, which are crucial in translating desired motion to the physical propeller speed. Furthermore, **u** is the control variable with **n** as the propeller revolutions per minute in Equation (3.9).

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} n_1 & |n_1| \\ n_2 & |n_2| \end{bmatrix} \tag{3.9}$$

As discussed by Bjering Strand, 2020, as the thrusters of Otter only act on the surge and heading of the USV, the matrix for $\boldsymbol{\tau}$ can be written as Equation (3.8), with the control inputs $\boldsymbol{\tau}_1$ for surge and $\boldsymbol{\tau}_3$ for yaw.

$$\tau = \begin{bmatrix} \tau_1 & 0 & 0 & 0 & 0 & \tau_6 \end{bmatrix}^{\mathrm{T}} \tag{3.10}$$

By using Equation (3.9)-(3.10) in conjunction with Equation (3.8) the control forces, with $\boldsymbol{l}$ as the moment arms in yaw, can be calculated as Equation (3.11).

$$\begin{bmatrix} \tau_1 \\ \tau_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -l_1 & -l_2 \end{bmatrix} \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{3.11}$$

as the propellers are evenly placed in regards to the USV's center line the moment arms will have the relation $\boldsymbol{l_1} = -\boldsymbol{l_2}$. Furthermore by solving this for $\boldsymbol{u}$ it yields Equation (3.12).

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 \\ -l_1 & -l_2 \end{bmatrix}^{-1} \begin{bmatrix} \tau_1 \\ \tau_6 \end{bmatrix} \tag{3.12}$$

The Otters thrusters are modeled as quadratic thrusters, meaning the produced force will be proportional to the square of the propeller revolutions per minute, scaled by a thruster coefficient, which is a multiplier adjusting the force output based on the propeller speeds. By solving for $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ in Equation (3.9), Equation (3.13) is found:

$$\begin{bmatrix} n_1 \\ n_2 \end{bmatrix} = \begin{bmatrix} \operatorname{sgn}(u_1)\sqrt{|u_1|} \\ \operatorname{sgn}(u_2)\sqrt{|u_2|} \end{bmatrix} \tag{3.13}$$

Where the sign function ($\boldsymbol{sgn}$), defined in Equation (3.14), returns either a -1, 0 or 1 to decide the direction of the thruster forces (Fossen, 2011, p.414):

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{3.14}$$

And $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ will return the thruster values from controller inputs.

Chapter 4

Materials & Equipment

Table 4.1 contains the equipment used throughout project (*OTTER USV User Manual*, 2020).

Table 4.1: Component list: Otter USV

Parts	Description
On-Board system control box	Central unit of electronic systems
Frame	Structural framework of the Otter
Pontoons	Floating structures for buoyancy
Torqeedo Motor	2 electric motors up to 80HP
Torqeedo Battery	2-4 batteries
Vehicle Control Station	Oceanlab ThinkPad with installed VCS
Side panels	Protective covers on side of vessel
Sensor payload box	Container housing sensors

4.1 Otter USV

The Otter frame with two antenna mounts, on the front and the back, is connected on top of the two pontoons. These have integrated Torqeedo motors and room for two batteries each. On the side of the frame the side panels are connected, with an On-Board Computer on the top with sensor hardware. The disassembled items of the Otter are shown in Figure 4.1.

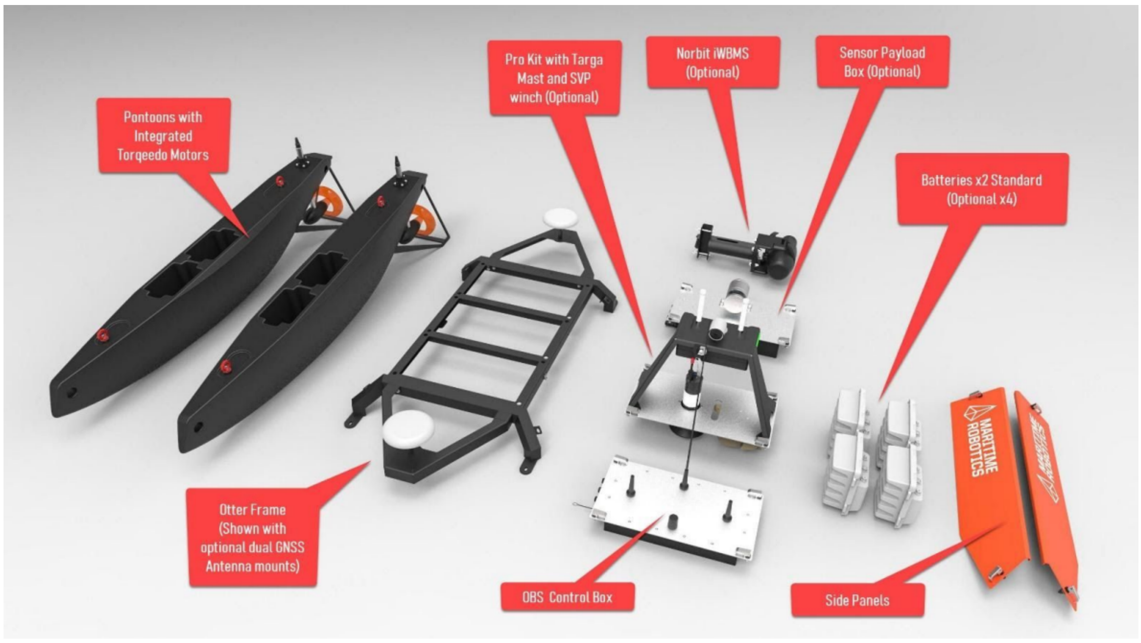

Pontoons with Integrated Torqeedo Motors
Pro Kit with Targa Mast and SVP winch (Optional)
Norbit iWBMS (Optional)
Sensor Payload Box (Optional)
Batteries x2 Standard (Optional x4)
Otter Frame (Shown with optional dual GNSS Antenna mounts)
OBS Control Box
Side Panels
MARITIME ROBOTICS

Figure 4.1: Otter USV components

4.1.1 On-Board System Control Box

Most of the essential electronics are housed and protected by the OBS, and are connected to the Torqeedo batteries and propulsion systems. It is an integral part in the computational and communicative parts of the Otter. A SBG inertial measurement unit (IMU) is also housed in the control box as depicted in Figure 4.2.

Custom electronics contained in the OBS includes:

- An On-Board Computer with an interface for the torqeedos
- Automatic identification system and a high frequency whip antenna
- A Huawei E3372H-153 4G LTE USB dongle and a 4G antenna
- 2.4GHz antenna for the WiFi interface

The automatic identification system (AIS) is an automatic tracking system used to view marine traffic (Yang et al., 2019), conveying data regarding position, heading and velocity. With the original intent of assisting watchkeepers maintain safe course it has become an integral part in the automation of waypoint planning in USV's.

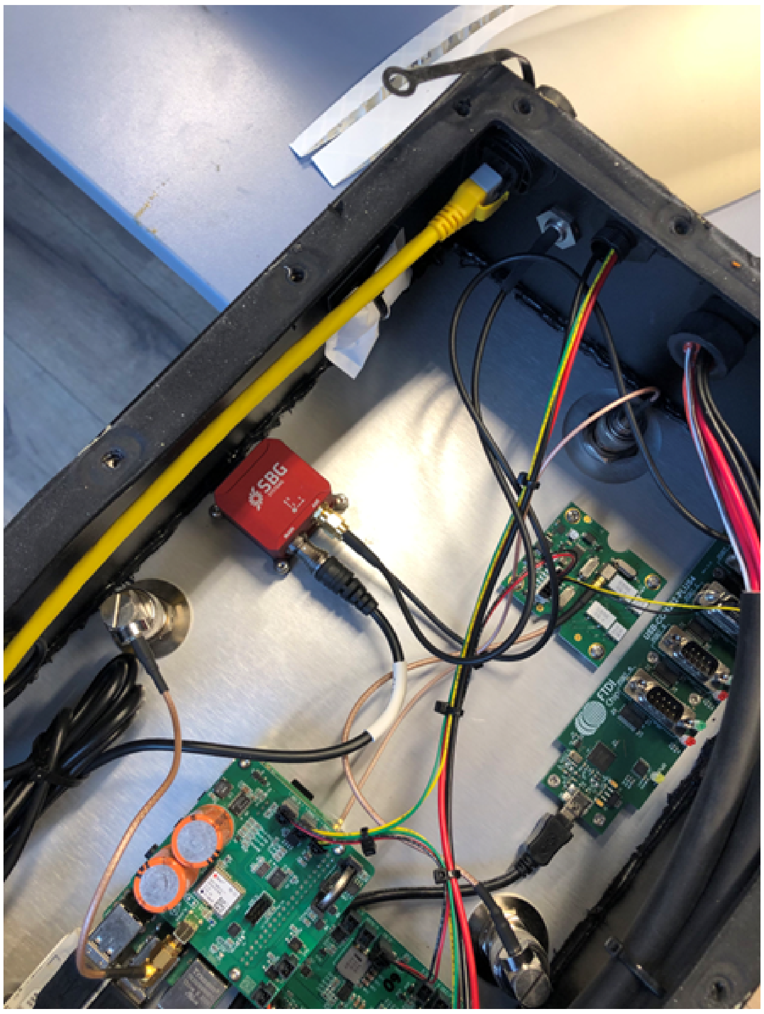


Figure 4.2: OBS Control Box with IMU

4.2 Sensors

The Otter is by default equipped with an IMU, as well as GPS, GNSS and RTK antennas. It is possible to mount different sensors depending on user needs such as LIDAR or sonar systems.

4.2.1 Positioning systems

The Otter uses a dual antenna setup for the GNSS, GPS as well as RTK correction data. This gives the Otter a high accuracy way of positioning itself, where optimal conditions allows the Otter sub-centimeter precision (Mæhlum, 2023).

4.2.2 Inertial Measurement Unit

The IMU shown in figure 4.2 measures the rate of rotation around the three spatial axes (Bjering Strand, 2020, p.20), which is a critical component in guidance, navigation and control of USV's. The IMU is composed of a magnetometer, gyroscope and an accelerometer to measure the angular velocity of the pitch, yaw and roll around the respective x-, y- and z-axis, as well as the linear acceleration. As discussed by Bjering Strand, 2020 and Fossen, 2011, a stand-alone IMU will drift due to sensor biases, which can be offset by combining GNSS/GPS with the IMU.

4.3 Vehicle Control Station

The VCS is a computer running on Ubuntu Linux, installed with software including a GUI produced by Maritime Robotics, as seen in Figure 4.3 (*OTTER USV User Manual*, 2020). Each label corresponds to a core function of the GUI.

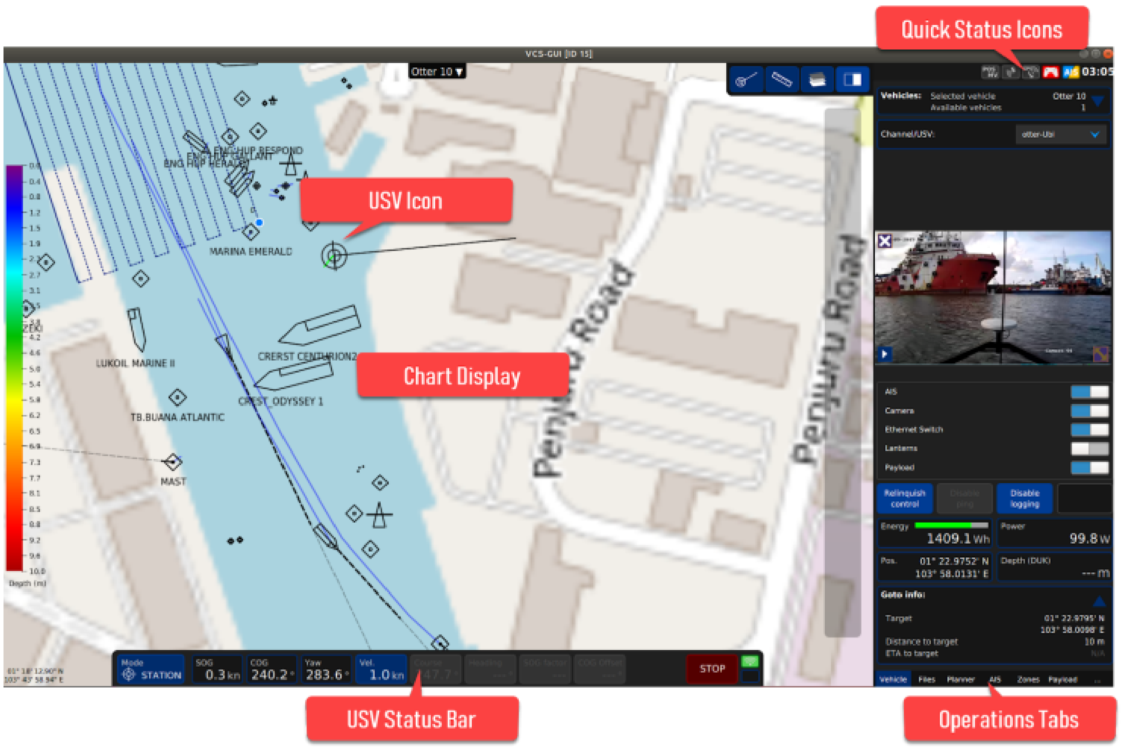


Figure 4.3: GUI for Vehicle Control Station

Core functions of the VCS also include verification of sensors and AIS functionality. Additionally it includes a map-based GUI for the surveyor to plan, monitor and acquire data along a designated route. A connection via separate devices is facilitated through the VCS user interface, supporting WiFi, Bluetooth and 4G.

Chapter 5

Methods & implementation details

5.1 Python code implementation

The complete GitHub repository with instructions and comments is published for viewing in Appendix A.

5.1.1 Application programming interface

The API consists of three main files, ”Otter_API.py”, ”Connector.py” and ”Control.py”.

Connector.py handles all the communication and message handling to and from the Otter. This includes establishing a socket connection while applying and decoding checksum on transmissions to and from the Otter.

Control.py handles all the logic and functions that is used to control the Otter and its thrusters.

Otter_API.py is the main API, where both Connector.py and Controller.py are imported and their functions used to create an Otter object, with an array of callable functions. Therefore when using the API, the Otter_API.py is the only file that needs to be imported, although the Connector.py and Control.py must be in the same directory as the main API.

Communication

In the API an ”establish_connection” function is called with the Otter's IP address and port to connect to the Otter, establishing a streaming socket connection to the Otter. If the connection is unsuccessful an error message will be posted in the console. Listing 5.1 shows the functions that are responsible for opening and closing the socket connection.

Listing 5.1: Connector: Socket communication

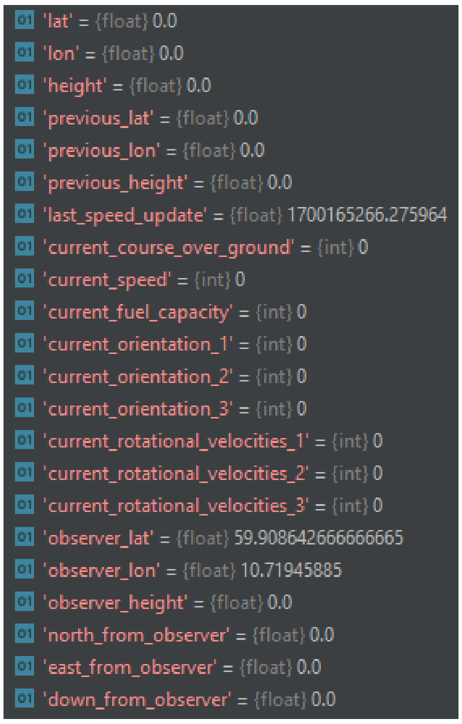
```
def establish_connection(self, ip, port):
    try:
        # Creates a socket object.
        self.sock = socket.socket(socket.AF_INET,
            socket.SOCK_STREAM)
        self.sock.connect((ip, port))
        self.connection_status = True
        print("connected")
        return True
    except:
        print("Could␣not␣connect␣to␣Otter")
        return False

def close_connection(self):
    try:
        self.sock.close()
        self.connection_status = False
        return True
    except:
        print("Error␣when␣disconnecting␣from␣Otter")
        return False
```

Receiving data

An "update_values" function sorts all the last received data from the Otter into a dictionary named "sorted_values", seen in Figure 5.1, which is then available to the Otter object.

The API also features a geodetic to north, east and down (NED) conversion, where the observer coordinates can be manually entered. The API will calculate the NED distances from the observer coordinates to the Otter and store these in the "sorted_values" dictionary together with



```
'lat' = {float} 0.0
'lon' = {float} 0.0
'height' = {float} 0.0
'previous_lat' = {float} 0.0
'previous_lon' = {float} 0.0
'previous_height' = {float} 0.0
'last_speed_update' = {float} 1700165266.275964
'current_course_over_ground' = {int} 0
'current_speed' = {int} 0
'current_fuel_capacity' = {int} 0
'current_orientation_1' = {int} 0
'current_orientation_2' = {int} 0
'current_orientation_3' = {int} 0
'current_rotational_velocities_1' = {int} 0
'current_rotational_velocities_2' = {int} 0
'current_rotational_velocities_3' = {int} 0
'observer_lat' = {float} 59.908642666666665
'observer_lon' = {float} 10.71945885
'observer_height' = {float} 0.0
'north_from_observer' = {float} 0.0
'east_from_observer' = {float} 0.0
'down_from_observer' = {float} 0.0
```

Figure 5.1: sorted_values dictionary

all the other information from the Otter. The "geodetic2ned" function in Listing 5.2 applies the Equations (3.2)-(3.5) and is automatically called every time the values in "sorted_values" is updated.

Listing 5.2: Connector: Geodetic2ned

```
def geodetic2ned_position(self):
    n, e, d = pm.geodetic2ned(self.sorted_values["lat"],
        self.sorted_values["lon"],
        self.sorted_values["height"],
        self.sorted_values["observer_lat"],
        self.sorted_values["observer_lon"],
        self.sorted_values["observer_height"])
    self.geo2ned_from_observer = [n, e, d]
```

The geodetic2ned function is also used to calculate the speed of the Otter by dividing the change in movement, seen in Equation (3.7), by the time since the previous update in Listing 5.3.

Listing 5.3: Connector: Geodetic2ned speed

```
    # Running geodetic2ned again in the,
    # Connector "Update_values" function

        s = np.hypot(n, e)
        v = s / (time.time() - self.last_speed_update)
        self.current_speed = v
        self.last_speed_update = time.time()
        self.previous_position = copy(self.current_position)
```

Checksum

The Otter uses the NMEA checksum standard to ensure that messages sent to and from the Otter are complete and not corrupted. NMEA checksum is a standard method of confirming the integrity of data in marine navigation systems (Shoab et al., 2013). As seen in Listing 5.4 the API employs a checksum function that takes messages from the Otter, transforming them into their numeric ASCII values while performing a XOR (exclusive or) operation to calculate a checksum value and converting it to a hexadecimal. When receiving a message the checksum is compared to a transmitted checksum, where a match indicates the message has been correctly transmitted.

Listing 5.4: Connector: Checksum

```
def checksum(message):
    checksum = 0
    for character in message:
        checksum ^= ord(character)
    checksum = hex(checksum)
    checksum = checksum[2:]
    if len(checksum) == 1:
        checksum = "0" + checksum
    return checksum

# From Connected "update_values" Function.
```

```
gps_message = ""
        imu_message = ""
        mod_message = ""
        for message in list:
            if message[:8] == "$PMARGPS":
                gps_message = message
            elif message[:8] == "$PMARIMU":
                imu_message = message
            elif message[:8] == "$PMARMOD":
                mod_message = message

        # Check for checksum error
        if checksum(gps_message[1:-3]) != gps_message[-2:].lower():
            print("Checksum␣error␣in␣$PMARGPS␣message")
            return
```

Most commands sent to and from the Otter needs checksum, while some like "drift" do not. This is a safety measure implemented by Maritime Robotics to reduce the likelihood of accidents as Otter is often used in close proximity with the shore. The checksum system reduces the chances of a corrupted command to enable thrusters being transmitted, increasing the chances of safer commands, such as "drift", being accepted.

Control allocation

The API also features a function for handling the control allocation, applying Equations (3.8)-(3.13). The function takes inputs $\boldsymbol{\tau}_\mathrm{x}$ and $\boldsymbol{\tau}_\mathrm{n}$ from the control system and then returns the thruster values $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ in a list.

5.1.2 Graphical user interface

The GUI is created using a python library named PyQT. This library hosts multiple features including its own graphical designer, QT designer, where windows can be created in real time. The library then creates .UI files which are converted to python and imported to other scripts.

This is used in all parts of GUI, and are set up with multiple types of widgets for different functions.

Login window

The login window (Figure 5.2) is the first window the user sees. It consists of two fields where values can be entered and a login button. When values are entered in the fields and the button is pressed, the GUI will store these and try to establish a connection to the Otter using the establish_connection function in the API, calling the function for opening the main window if successful.

Main window

When a connection to the Otter is established, the main window shown in Figure 5.3 opens. It hosts a map, a checkbox for enabling a gamepad and a drop down menu to open the developer mode window. The gamepad checkbox calls a function that continuously reads the inputs from a gamepad using the "get_gamepad" function from the inputs library. The data is transmitted to the Otter through the set manual mode function in the API. The developer mode button calls a function within the GUI itself to bring up the developer mode window as an entirely independent window.

The main feature of the window is a map created using the "Folium" library in python, displayed in Listing 5.5. The GUI initially checks for Otter connection and centers the map around the current longitudinal and latitudinal coordinates. If not connected, a set of default coordinates are used. When the map is called it returns a HTML map which is stored within a local variable before being displayed in the main window using a "QWebEngineviewer", which acts as a browser displaying the HTML code.

If the function detects that a map is already created, it will instead create a new marker at the current coordinates being received from the Otter. The main window also has a thread running a loop in the background which repeatedly updates the marker.

Listing 5.5: Main GUI: show_map

```
def show_map(self):
    self.otter.update_values()
    if not hasattr(self, 'main_map_ob'):
```

```
    self.otter.update_values()
    if self.otter.sorted_values["lat"] != 0 and
        self.otter.sorted_values["lon"] != 0:
        if self.verbose:
            print("Using␣GPS␣to␣setup␣map")
    else:
        if self.verbose:
            print("Using␣default␣values␣to␣set␣up␣map")

    self.main_map_ob =
        folium.Map(location=(self.otter.sorted_values["lat"],
            self.otter.sorted_values["lon"]),
            zoom_start=15)
            self.marker = None

if hasattr(self, 'main_map_ob'):
    if self.marker:
        self.main_map_ob = folium.Map(
            location=(self.otter.sorted_values["lat"],
                self.otter.sorted_values["lon"]),
                zoom_start=15)

    self.marker = folium.Marker(location=,
        [self.otter.sorted_values["lat"],
        self.otter.sorted_values["lon"]],
        icon=folium.Icon(color="green"))
    self.marker.add_to(self.main_map_ob)

data = io.BytesIO()
self.main_map_ob.save(data, close_file=False)
self.main_screen_ob.main_map_widget,
    .setHtml(data.getvalue().decode())
```

```
def thread_1_loop(self):
    self.otter.check_connection()
    self.otter.update_values()
    if self.main_window.isHidden():
        if self.verbose:
            print("is␣hidden")
        self.close_event()

    if self.developer_window.isHidden(),
        and self.developer_mode_enable:
            self.developer_window.destroy()
            self.developer_mode_enable = False

    self.show_map()
```

Developer window

The developer window (Figure 5.4) hosts most functions of the GUI. Each of the buttons and input fields in the developer window is directly connected to a function in the API. This enables efficient troubleshooting and testing of the Otter. The developer window also features a table on the right hand side, displaying all known values received from the Otter. A checkbox is supposed to be enabled to continuously update with live data, however due to limitations in the Folium library, the table will not update in real time without being interacted with.

5.2 Program launch & Testing procedure

5.2.1 Launching the graphical user interface

When first launching the GUI the user is brought to a login window, displayed in Figure 5.2. By either entering the IP-address and Port to the Otter, or running the test server with the ”localhost” IP, access is granted to the GUI.

The main window, as shown in Figure 5.3, presents a map with a marker that continuously

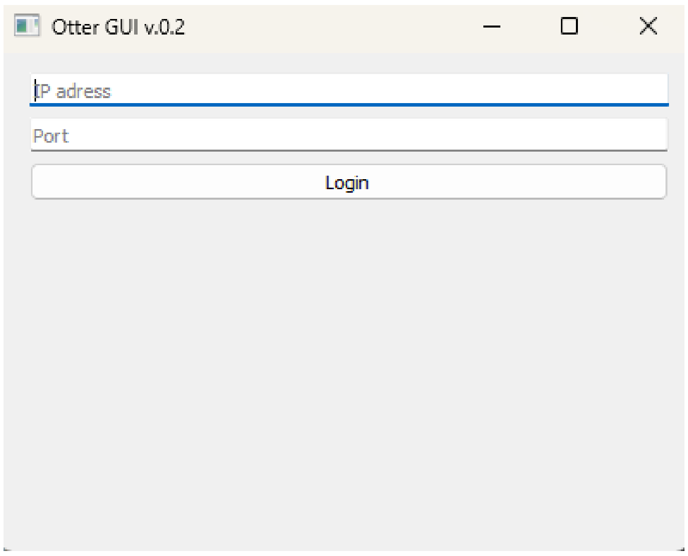


Figure 5.2: GUI login window

updates to reflect the current longitudinal and latitudinal coordinates. Through the main window the user can either access the developer window through the drop-down menu, or enable the gamepad, allowing the user to control the thrusters through the left analog stick of a controller connected to the shore station. This will work as long as the controller is recognized as a game controller by windows.

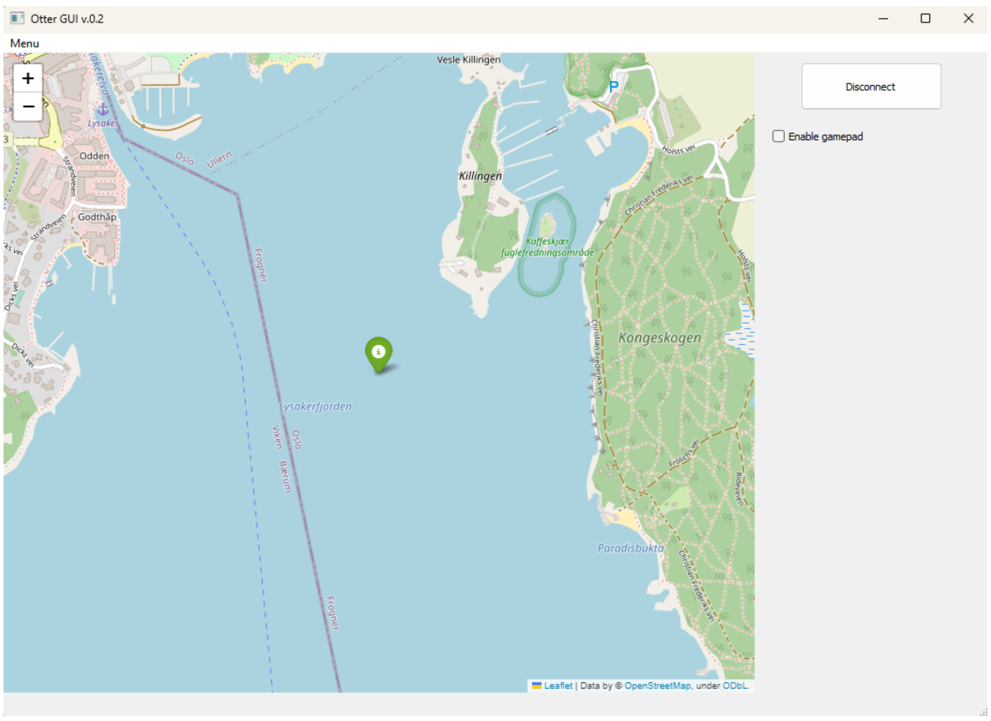
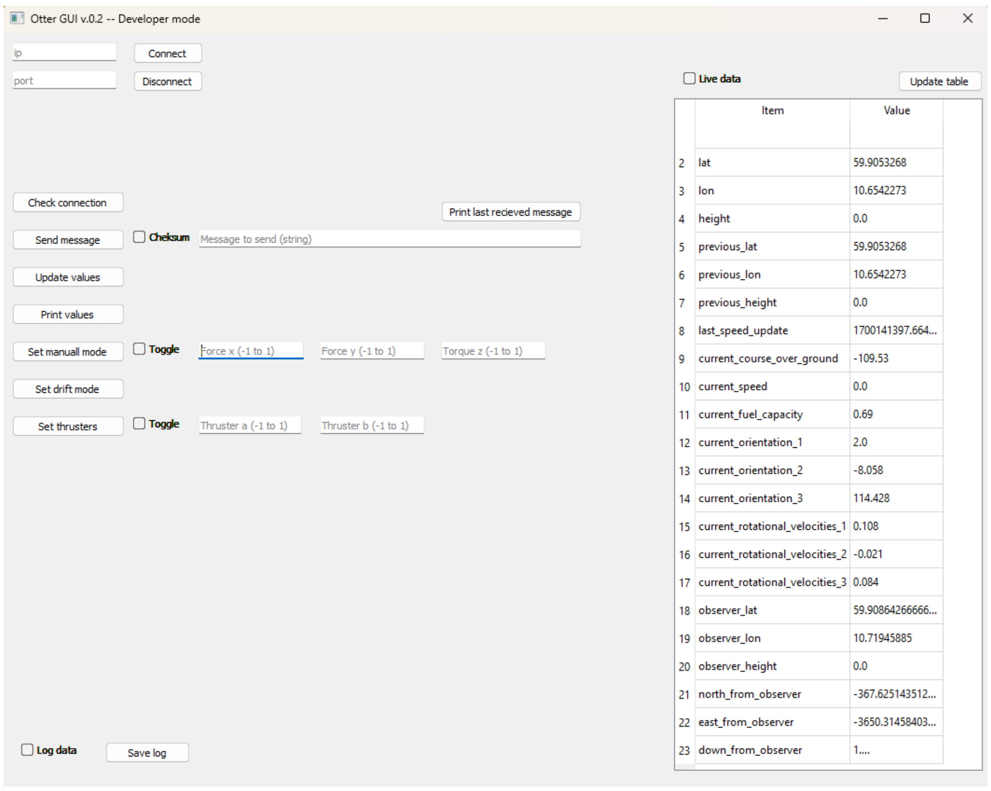


Figure 5.3: GUI main window

In the developer window the user is granted different options for operating Otter and viewing key values. Presented on the left hand of Figure 5.4, it allows for checking the connection and sending messages with the option to enable or disable checksum in the message being sent. The ”Set manual mode” changes the thruster output in the x and z (x and y in NED coordinates) directions by entering a value between -1.0 and 1.0 as output either reversed or forward, where 0.0 would be no thruster output. ”Set thrusters” works similarly by granting control of each individual thruster rather than the desired axis movement.

To update the data in the table the button ”update table” is utilized. The data can also be logged as a CSV file, by keeping ”Log data” checked for the duration data should be gathered before using the ”Save log” button when finished.



	Item	Value
2	lat	59.9053268
3	lon	10.6542273
4	height	0.0
5	previous_lat	59.9053268
6	previous_lon	10.6542273
7	previous_height	0.0
8	last_speed_update	1700141397.664...
9	current_course_over_ground	-109.53
10	current_speed	0.0
11	current_fuel_capacity	0.69
12	current_orientation_1	2.0
13	current_orientation_2	-8.058
14	current_orientation_3	114.428
15	current_rotational_velocities_1	0.108
16	current_rotational_velocities_2	-0.021
17	current_rotational_velocities_3	0.084
18	observer_lat	59.90864266666...
19	observer_lon	10.71945885
20	observer_height	0.0
21	north_from_observer	-367.625143512...
22	east_from_observer	-3650.31458403...
23	down_from_observer	1....

Figure 5.4: GUI developer window

5.2.2 Function testing

The testing methodology involved operating the Otter USV on a cart. The cart was brought around Trettenparken, Oslo, while changes in position, velocity, and IMU data were recorded. Additionally, the NED position was tracked relative to the initial starting point on a second-by-second basis. Using the ”Pandas” package for a logging function, all collected data was recorded to a CSV file, where the changes in coordinates were used to plot movement.

Testing of the thruster functions were done on land with Otter at a standstill. By using a controller and manual inputs in the developer window, the thrusters were tested at different values to test response. This was compared to the response of the thrusters when similar inputs were directly input to the VCS.

Chapter 6

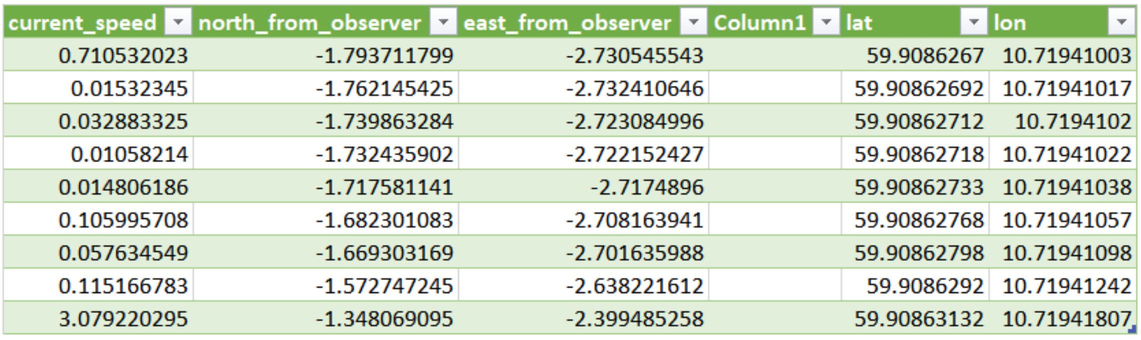
Results

6.1 Function test results

While testing the functionality of the GUI, it was concluded that both manual and drifting mode, as well as the gamepad worked as intended, with both thrusters reacting accordingly to input in either modes. The tracking updated as expected and was correctly logged in a CSV file. The testing was conducted using WiFi as the communication to the USV, causing the live map not to load as it requires an internet connection.

6.2 Experimental data

A CSV file, shown in Figure 6.1, including the coordinates, north and east travel was created. Through the respective arrays it is possible to monitor change between every message from Otter. The complete CSV file and map functions are found in the GitHub repository.



current_speed	north_from_observer	east_from_observer	Column1	lat	lon
0.710532023	-1.793711799	-2.730545543		59.9086267	10.71941003
0.01532345	-1.762145425	-2.732410646		59.90862692	10.71941017
0.032883325	-1.739863284	-2.723084996		59.90862712	10.7194102
0.01058214	-1.732435902	-2.722152427		59.90862718	10.71941022
0.014806186	-1.717581141	-2.7174896		59.90862733	10.71941038
0.105995708	-1.682301083	-2.708163941		59.90862768	10.71941057
0.057634549	-1.669303169	-2.701635988		59.90862798	10.71941098
0.115166783	-1.572747245	-2.638221612		59.9086292	10.71941242
3.079220295	-1.348069095	-2.399485258		59.90863132	10.71941807

Figure 6.1: Logged CSV data (Reduced version)

Using the package "Matplotlib" the different coordinates from the log were plotted in Figure 6.2, creating a graph of the total travel during the testing. Travel from starting location before moving around the park and returning is displayed, and with data being saved every second, the speed at different points of the path is visualized, where a larger distance between each point signifies a higher velocity at that point.

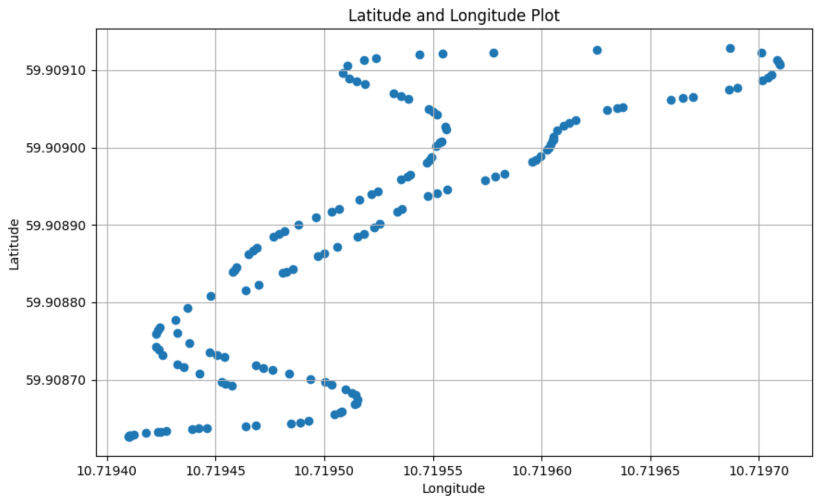


Figure 6.2: Longitudinal and latitudinal travel

Using these test results with Folium an accurate line is drawn on a map of the area surrounding Trettenparken in Figure 6.3.

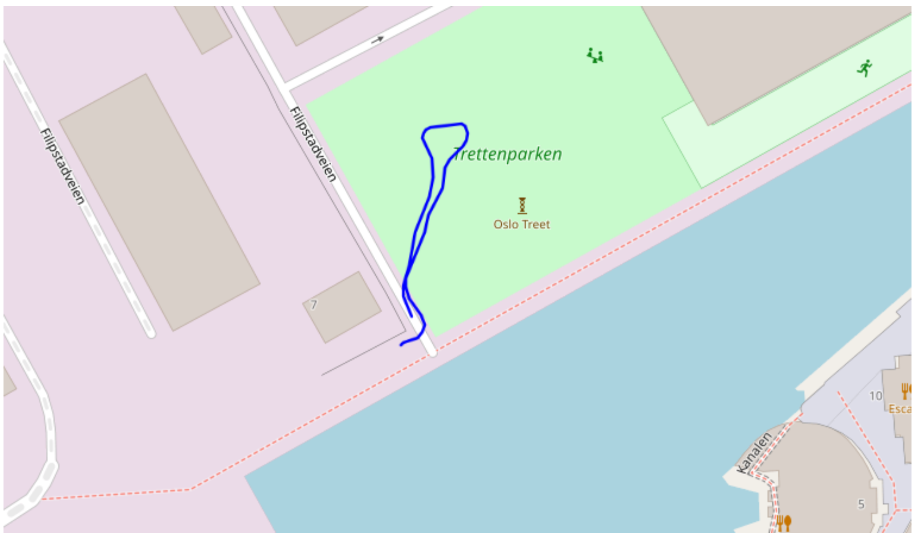


Figure 6.3: Mapped position of USV

Chapter 7

Discussion & conclusion

7.1 Analysis of results and influence of source errors

Since no external tools were used for measuring the speed, it is difficult to confirm if the speed displayed in the GUI is accurate. However, due to the accuracy of the coordinate logs, which were plotted in Figure 6.2, it appears correct as the velocity is a simple function of displacement over time.

Whenever the map is updated for new marker positions the page is completely refreshed. This prevents resizing, moving, and zooming of the map. This appears to be a drawback of the Folium library, and resolving this would likely require changing the entire mapping method.

As discussed in the function test, an issue appeared with the live map not being available during testing, which potentially would be resolved by using another communication protocol, either a 4G or radio, so the shore station can stay connected to the Internet during use. This might be solved by locking the window size to not vary depending on the resolution.

While the features of the GUI are working as intended there is an issue with resizing the window to read the text of all the buttons, which could make it difficult to use all widgets and view values on lower resolution screens.

There was no testing in aquatic conditions, thus it is not possible to conclude that no problems would arise in later testing and use, although the primary test results implies functions work as expected.

7.2 Assessment of theory and other products

The theory behind the project is heavily influenced by Thor I. Fossen's Handbook of Marine Craft. The sources related to the theoretical parts of the paper are also to a varying degree based on Fossen's works, although for the use of vessel in 3 DOF rather than 6 DOF, as discussed by Setiawan et al., 2022, Bjering Strand, 2020 and Saksvik et al., 2022.

The code was loosely based on a part of Magnus Kjelsaas's thesis, found in Appendix A - GitHub foundation repository, which was improved on in terms of functionality, structure and ease of use.

7.3 Concluding remarks & further work

The project with an autonomous surface vessel includes the development of a working application programming interface which establishes a socket communication, and processes data delivered and received between the Otter USV and a shore station in real-time. This includes data of the vessels current coordinates, speed, and total travel in the north, east and down directions.

Furthermore, a graphical user interface was created consisting of a live map with tracking, data monitoring, and manual control options of the Otter USV.

This project has resulted in a more user-friendly way of operating the USV in future research applications, which could enhance the appeal for future research with the Otter.

Future work should focus on refining the mapping system to enhance the USV's detection and tracking capabilities. Furthermore, improving the GUI and carrying out aquatic testing of the USV.

Bibliography

Barrera, C., Padron, I., Luis, F., & Llinas, O. (2021). Trends and challenges in unmanned surface vehicles (usv): From survey to shipping. *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation*, *15*, 135–141.

Beaubouef, B. (2023). Unmanned surface vehicles being deployed for range of applications. *Offshore*. https://www.offshore-mag.com/special-reports/article/14296473/unmanned-surface-vehicles-being-deployed-for-a-range-of-applications

Bjering Strand, H. (2020). *Autonomous docking control system for the otter usv: A machine learning approach* [Master's thesis, NTNU].

Deshayes, P. (2021). First electric autonomous cargo ship launched in norway. *Tech Xplore*. https://techxplore.com/news/2021-11-electric-autonomous-cargo-ship-norway.html

Fossen, T. I. (2011). *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons.

Fossen, T. I., & Sagatun, S. I. (1991). Adaptive control of nonlinear underwater robotic systems.

Liu, Z., Zhang, Y., Yu, X., & Yuan, C. (2016). Unmanned surface vehicles: An overview of developments and challenges. *Annual Reviews in Control*, *41*, 71–93.

Mæhlum, L. (2023). Rtk [Store Norske Leksikon].

Otter usv user manual [Rev. 1.4]. (2020). Maritime Robotics. https://www.maritimerobotics.com/otter

Saksvik, I. B., Alcocer, A., Hassani, V., & Pascoal, A. (2022). Target tracking of an underwater glider using a small unmanned surface vessel. *IFAC-PapersOnLine*, *55*(31), 478–483.

Setiawan, J., Septiawan, M., Ariyanto, M., Caesarendra, W., Munadi, M., Alimi, S., & Sulowicz, M. (2022). Development and performance measurement of an affordable unmanned surface vehicle (usv) [Academic Editors: Jahangir Hossain. Received: 9 November 2021;

Accepted: 29 December 2021; Published: 4 January 2022]. *Automation*, *3*, 27–46. https://doi.org/10.3390/automation3010002

Shoab, M., Jain, K., Anulhaq, M., & Shashi, M. (2013). Development and implementation of nmea interpreter for real time gps data logging. *2013 3rd IEEE International Advance Computing Conference (IACC)*, 143–146.

SNAME, T. (1950). Nomenclature for treating the motion of a submerged body through a fluid. *The Society of Naval Architects and Marine Engineers, Technical and Research Bulletin*, (1950), 1–5.

Yang, D., Wu, L., Wang, S., Jia, H., & Li, K. X. (2019). How big data enriches maritime research–a critical review of automatic identification system (ais) data applications. *Transport Reviews*, *39*(6), 755–773.

Appendix A - GitHub Repositories

GitHub project repository: https://github.com/SigurdGresbe/Otter_Bachelor
figure (5.2)-(5.4) and 6.1 opened by running Main_GUI.py

Figure 6.2 and 6.3 created through plot and plot map.py

GitHub foundation repository: https://github.com/MrKjelsaas/school_projects/blob/master/master_thesis/otter/otter.py

Appendix B - Group Member Contribution

Student	**Written contribution**	**Technical contribution**
Sivert	Ch. 4-5, total 25%	40% of the code, both API and GUI, testing and video editing
Lars	Ch. 1-4, 5.2, 6 and formatting/editorial contributions, total 40%	25% of code, API, Folium map and testing
Sigurd	Abstract, Ch. 6-7, editorial contributions, total 35%	GitHub management, testing and 35% of coding, mainly in API

As several key parts of the document were worked on in conjunction with other group members we attempted to estimate a percentage contribution to the entire document and code.