Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in information engineering

modelling of remotely operated vehicles: assessing the accuracy of existing ODEs-based simulators against field experiments

Master Candidate

Riccardo Fusari

Student ID 1198977

Supervisor

Prof. Damiano Varagnolo

University of Padova

Co-supervisor

Roberto Francescon, Filippo Campagnaro,Antonio Montanari

University of Padova

Academic Year
2022/2023

Abstract

The task for this project was to build the hardware for the BlueRov2 and implement an autopilot in a way that emulated a simulator. This was done to test how well the simulator matched reality and to evaluate its coherence
This is an important project because it is the first practical project for the DEI of Padova University involving this hardware. The project will rely on a set of foundations that can be used for future projects
This was the first time I approached a project like this, and I found it constructive to see how the knowledge I gained through my studies can be applied to a practical project
Unfortunately, the program didn't work very well, but this experience taught me the standard procedure for approaching a project like this, and how to see progress despite setbacks. It also helped me improve my problem-solving skills for future projects of this nature

Contents

List of Figures

List of Code Snippets

1 introduction

1.1 Background

Underwater vehicles (UV) are today used in a variety of marine operations, such as industrial, military and research applications. Specific tasks may be inspections, seafloor mapping and surveillance, as well as military specific missions like mine hunting and sabotage.
They perform tasks that earlier have been done by humans, as well as task humans can't do. They are in general highly maneuverable, as well as cost efficient, reliable and delivers high quality services
Based on their level of autonomy, UUVs may be divided into two categories:

- Remotely operated vehicles (ROV)
  Underwater vehicles that are controlled by an operator on the surface or on a nearby vessel. ROVs are typically tethered to a support vessel or platform, which provides power, control signals, and video feedback to the operator

- Autonomous underwater vehicles (AUV)
  Underwater vehicles that are capable of operating without direct human control. Unlike remotely operated vehicles (ROVs), which are controlled by an operator on the surface, AUVs are self-contained and can operate independently.
  AUVs are typically pre-programmed with a mission plan that outlines the vehicle's intended path, speed, and actions. Once deployed, the AUV will follow this plan and perform its tasks autonomously, without the need for human intervention
  With advances in artificial intelligence and machine learning, AUVs are

becoming even more autonomous, able to make decisions based on real-time data and adapt their behavior to changing conditions. This allows them to operate more efficiently and effectively, while also reducing the risk of human error and increasing the safety for human operators relative to hostile environments

1.2 TASKS TACKLED

- The primary objective was to program the ROV in such a way that its behavior would precisely mirror that of the simulator [7]. This was carried out to evaluate the disparities between the simulation and actual-world testing.
- Since Python was not part of my study program, a significant portion of the task was to become familiar with and learn to utilize this environment.
- While it is possible to program the BlueRov2 in Python, there may not be a comprehensive guide available for it. Therefore, one of the tasks involved in this project was to conduct thorough research by sifting through various forums and online resources, such as the BlueRoboticsforum [2], to learn how to write a script for controlling the ROV using Python.
- The simulator [7] was not intended for practical use, and as a result, the ability to manually set the path was not implemented.

1.3 CHAPTERS DESCRIPTION

- Chapter 1-Introduction
  - This is the chapter that you are currently reading. It provides an overview of the importance of subsea automation, highlights the challenges encountered during the initial phase of the project, and provides a brief description of the different chapters of this thesis.
- Chapther 2- Theory
  - This chapter is dedicated to explaining the fundamental aspects of the knowledge that I employed in this project. Specifically, the topics covered include coordinate frames, slide mode control, and the model of the ROV.

- Champter 3- The physical ROV we considered
  - "This chapter is dedicated to providing an overview of the main hardware and software components used in the project. This includes a detailed description of the different components, such as the sensors, controllers, and other hardware, as well as the software. The goal is to provide the reader with a clear understanding of the technology and tools used throughout the project and how they relate to each other.

- Chapther 4- The Simulator
  - This section provides an overview of the primary components of the Matlab simulator [7] and explains how it works. The discussion covers the fundamental aspects of the simulator and provides a detailed explanation of its functionality.

- Chapther 5- Program's developmet
  - In this section I will explain the logic behind the script and its components

- Chapther 6-Test and evaluation
- Chapther 7-conclusion

2 Theory

In this chapter, I will acknowledge the theoretical framework that has been used in this project. To this end, I have drawn upon various sources, including the simulator's documentation [7] and a thesis from NUTU university [6]. By considering the relevant literature, I aim to provide a comprehensive understanding of the underlying principles that inform the development and implementation of the project.

2.1 Coordinate Frames

Coordinate frames are essential for defining the orientation and position of objects with respect to each other. In the case of controlling a moving vehicle, having a reference point that remains constant is crucial. The body frame is a frame of reference that remains constant when viewed from the perspective of a moving vehicle, even though it is actually moving in relation to the world frame. For this project, I have chosen to use the North-East-Up (NEU) representation as the coordinate frame.

The ship motion convention outlines standard movement directions in relation to a vehicle's body frame. In order to properly apply attitude and position set points to a vehicle, it is crucial to understand how they are presented within the body frame, as this knowledge is necessary to properly manage the behavior of the vehicle's thrusters.

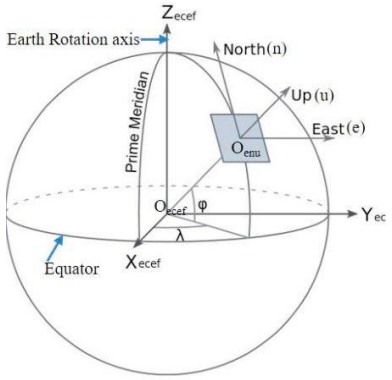


Figure 2.1: A image that describe the North-East-Up frame in relation with the earth

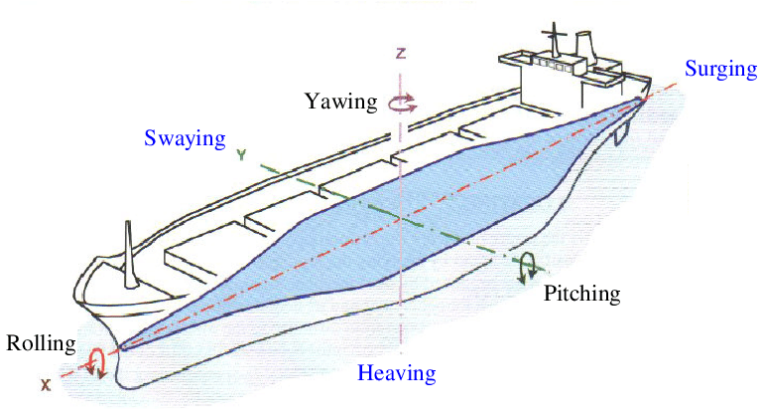


Figure 2.2: A diagram that shows the convention for ship motion on a nautical vessel

2.2 Sliding Mode controller (SMC)

The Sliding Mode controller is a popular choice for controlling non-linear systems in this environment due to its low computational cost and robustness to disturbances. Although this topic was not covered in my studies, my understanding is that it is a feedback control technique that forces the system to slide along a prescribed path by sending a discontinuous control signal. While I do not have a complete understanding of its inner workings, its effectiveness in controlling non-linear systems makes it a viable option for this project.

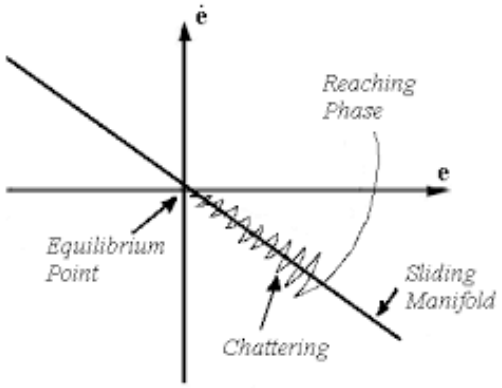


Figure 2.3: an example of how a Sliding Mode Controller interact with a system

2.3 The ROV's model

To study his behaviour we have to consider not only the kinematic of the vehicle but the kinetics too; For that we used the Fossen notation Which takes both into account

$$\dot{\eta} = J(\eta)\nu \tag{2.1}$$

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \tau \tag{2.2}$$

- The first equation use the matrix J(q) to pass from the body frame movements V=[u,v,w,p,q,r] (surge, sway, heave, roll, pitch and yaw) in to the world frame movements n=[x,y,z,ϕ,θ,ψ]

$$J(\eta) = \begin{bmatrix} J_1(\eta) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & J_2(\eta) \end{bmatrix} \tag{2.3}$$

$$J_1(\eta) = \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\phi\sin\theta\sin\psi & -\cos\psi\sin\phi + \sin\theta\sin\psi\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{2.4}$$

$$J_2(\eta) = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \tag{2.5}$$

where let us recall that J2 is not defined when θ=$-\pi$,π)

- The second one is about UUVs motion, derived from the Newton-Euler fomulation (T. I. Fossen 2021), in particular:
  - M is the system inertia matrix with:
    - $X_{\dot{u}}, Y_{\dot{v}}, Z_{\dot{w}}, K_{\dot{p}}, M_{\dot{q}}, N_{\dot{r}}$ are the added mass components
    - I_x, I_y, I_z are the Inertia moment components

$$M = \begin{bmatrix} m - X_{\dot{u}} & 0 & 0 & 0 & mZ_g & 0 \\ 0 & m - Y_{\dot{v}} & 0 & -mZ_g & 0 & 0 \\ 0 & 0 & m - Z_{\dot{w}} & 0 & 0 & 0 \\ 0 & -mZ_g & 0 & I_x - K_{\dot{p}} & 0 & 0 \\ mZ_g & 0 & 0 & 0 & I_y - M_{\dot{q}} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z - N_{\dot{r}} \end{bmatrix} \tag{2.6}$$

  - C is the Coriolis and centripetal matrix

$$C(\nu) = \begin{bmatrix} 0 & 0 & 0 & 0 & mw + Z_{\dot{w}}w & 0 \\ 0 & 0 & 0 & -mw - Z_{\dot{w}}w & 0 & -X_{\dot{u}}u \\ 0 & 0 & 0 & mv - Y_{\dot{v}}v & -mu + X_{\dot{u}}u & 0 \\ 0 & mw - Z_{\dot{w}}w & -mv + Y_{\dot{v}}v & 0 & I_z r - N_{\dot{r}}r & -I_y q + M_{\dot{q}}q \\ -mw + Z_{\dot{w}}w & 0 & -mu + X_{\dot{u}}u & -I_z r + N_{\dot{r}}r & 0 & I_x p - K_{\dot{p}}p \\ mv - Y_{\dot{v}}v & -mu + X_{\dot{u}}u & 0 & I_y q - M_{\dot{q}}q & -I_x p + K_{\dot{p}}p & 0 \end{bmatrix} \tag{2.7}$$

  - D(v) is the hydrodynamic dampening matrix
    - $X_u, Y_v, Z_w, K_p, M_q, N_r$ are the linear dumping components
    - $X_{u|\dot{u}|}, Y_{v|\dot{v}|}, Z_{w|\dot{w}|}, K_{p|\dot{p}|}, M_{q|\dot{q}|}, N_{r|\dot{r}|}$ are the quadratic damping components

$$D(\nu) = -\begin{bmatrix} X_u + X_{u|\dot{u}|}|\dot{u}| & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_v + Y_{v|\dot{v}|}|\dot{v}| & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_w + Z_{w|\dot{w}|}|\dot{w}| & 0 & 0 & 0 \\ 0 & 0 & 0 & K_p + K_{p|\dot{p}|}|\dot{p}| & 0 & 0 \\ 0 & 0 & 0 & 0 & M_q + M_{q|\dot{q}|}|\dot{q}| & 0 \\ 0 & 0 & 0 & 0 & 0 & N_r + N_{r|\dot{r}|}|\dot{r}| \end{bmatrix} \tag{2.8}$$

  - g(q) is the vector of restoring forces and moments

$$g(\eta) = \begin{bmatrix} (W - B)\sin\theta \\ -(W - B)\cos\theta\sin\phi \\ -(W - B)\cos\theta\cos\phi \\ y_b B\cos\theta\cos\phi - z_b B\cos\theta\sin\phi \\ -z_b B\sin\theta - x_b B\cos\theta\cos\phi \\ x_b B\cos\theta\sin\phi + y_b B\sin\theta \end{bmatrix} \tag{2.9}$$

    where W=mg and B=ρg∇ are respectively the gravity and buoyancy forces (ρ the water density and ∇ the volume of fluid displaced by the vehicle), and (xb,yb,zb) are the coordinates of the center of buoyancy expressed in the body frame
  - T=[X,Y,Z,K,M,N] is the vector that contains the forces and moments applied to the vehicle

3

The physical ROV we considered

The ROV that i used for the experience is a BlueRov2 bought from the university from BlueRobotics [**BlueRobotics**]; a company that specializes in designing and manufacturing underwater robotics and related components.
The BlueROV2 is designed to be modular and customizable, with a variety of add-ons and accessories available, including sonar systems, manipulators, and scientific sensors. It is also open-source, which means that users can modify and customize the ROV's software and hardware to suit their needs.
Precisely i use the heavy configuration variant: a modified version of the base one with additional components that increase its payload capacity and endurance.

Figure 3.1: To the left a BlueRov2 in normal configuration, to the right a BlueRov2 in heavy configuration

3.1 Hardware

The structure have eight T200 thrusters that allow the vehicle to perform movements with 6 DOF

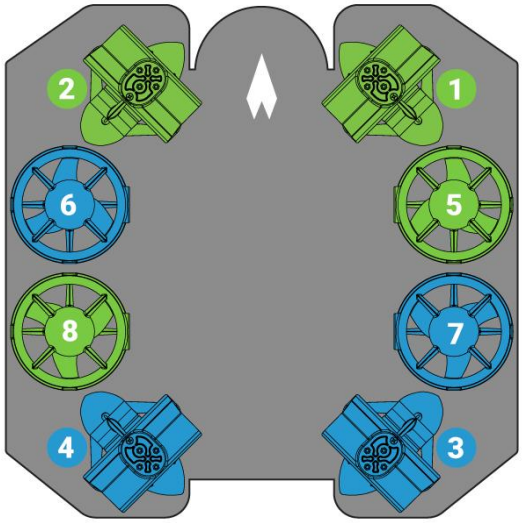


Figure 3.2: A diagram that show how the thrusters are mounted in a BlueRov2 in heavy configuration

The ROV already has several sensors, including:

3.1.1 3 DOF gyroscope

Device that measures the rate of rotation or angular velocity of an object in three dimensions. The 3 DOF gyroscope on the BlueRov2 is used to measure the rotational velocity of the vehicle in three axes - pitch, roll, and yaw. The gyroscope is used to provide information to the vehicle's control system, which adjusts the thrust of the ROV's thrusters to maintain stability and orientation

3.1.2 3 DOF accelerometer

The BlueRov2 is equipped with a 3 degrees of freedom (DOF) accelerometer that is used to measure the vehicle's acceleration in three axes - X, Y, and Z. An accelerometer is a device that measures changes in velocity or acceleration

of an object. In the case of the BlueRov2, the accelerometer is used to provide information about the ROV's movement and orientation.

3.1.3 MAGNETOMETER

Sensor that measures the strength and direction of a magnetic field. The magnetometer on the BlueRov2 is used to provide information about the vehicle's orientation with respect to the Earth's magnetic field. The magnetometer measures the strength and direction of the magnetic field and uses this information to calculate the vehicle's heading or direction of travel. The magnetometer is used in conjunction with other sensors, such as the 3 DOF accelerometer and 3 DOF gyroscope, to provide accurate positioning and orientation control for the ROV

3.1.4 PRESSURE SENSOR

sensor that measures the pressure of the surrounding environment. The pressure sensor on the BlueRov2 is used to measure the depth of the ROV in water.
The pressure sensor measures the pressure of the water surrounding the ROV and converts it into a depth measurement. This information is used by the vehicle's control system to adjust the thrusters and maintain the desired depth. For issue about acquire the data sensor this instrument isn't used during the project

3.1.5 HIGH-RESOLUTION CAMERA

The BlueRov2 is equipped with a high-resolution camera that is used for underwater imaging and video recording. The camera is designed to capture high-quality images and videos of the underwater environment and can be used for a wide range of applications such as marine research, underwater inspection, and exploration.
The camera is mounted on a tilt mechanism, which allows the operator to adjust the camera angle to capture images and videos of different areas of interest. The camera is also equipped with various features such as autofocus and image stabilization, which help to ensure that the captured images and videos are sharp and clear.

The camera is connected to the operator's control station on the surface through a tether cable, which provides power and data transmission. The video feed from the camera is displayed on the operator's monitor in real-time, allowing the operator to remotely control the vehicle and navigate through the underwater environment while simultaneously capturing high-quality footage
Unfortunately there is a bug in the software dedicated to the Ubuntu (the S.O. of the laboratory computer) operating system, which is preventing the recording of video from the BlueRov2

3.1.6 2 LED LIGHTS

Lights use to illuminate the environment and can be controlled through a Pulse Width Modulation (PWM) signal. The intensity of the lights can be adjusted by changing the duty cycle of the PWM signal

3.2 Software

The software is composed by 3 main component:

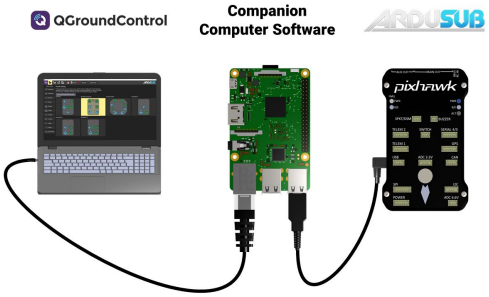


Figure 3.3: A image that show the main software's components and where they are loaded

3.2.1 QGController [5]

The QGController is a software that run on the surface computer; it provides a user-friendly interface for controlling the ROV's movements, viewing live video feeds, and monitoring sensor data.

3.2.2 ArduSub [1]

Is loaded on to the internal memory of the autopilot board and contain all the logical process necessary to control the vehicle
Ardusub communicate with a protocol called Mavlink and by a python implementation called Pymavlink is possible write a script to read sensor data and send commands
It is an open-source software system that is designed specifically for remotely operated underwater vehicles (ROVs), including the BlueRov2. ArduSub is based on the popular ArduPilot software platform, which is widely used for unmanned aerial vehicles (UAVs) and other robotic systems.
ArduSub provides a range of features and capabilities that are tailored to the requirements of underwater applications, including precise control and navigation (if the hardware is implemented with the necessary sensors), depth and altitude hold, and support for various sensors and instruments such as sonar, cameras, and manipulators

3.2.3 Blue Robotics Companion

Blue Robotics Companion is a software package developed by Blue Robotics specifically for the BlueROV2 remotely operated underwater vehicle (ROV); it is a modified version of Raspbian that is written onto a microSD card and installed in the Companion Computer
Can perform the following function:

- relays communication from the autopilot and the QGroundControl via ethernet comunication
- Streams HD video to QGroundControl
- allow the implementation of additional peripherals

4

The simulator

For the experience i thought to use the simulator that i found on the BlueRobotics site [7]; it is a simulator, implemented on Simulink, designed recently by a group of researchers of the University of Southern Denmark; it is develop to model the environment and the BlueRov's behaviour controlled by a SM controller

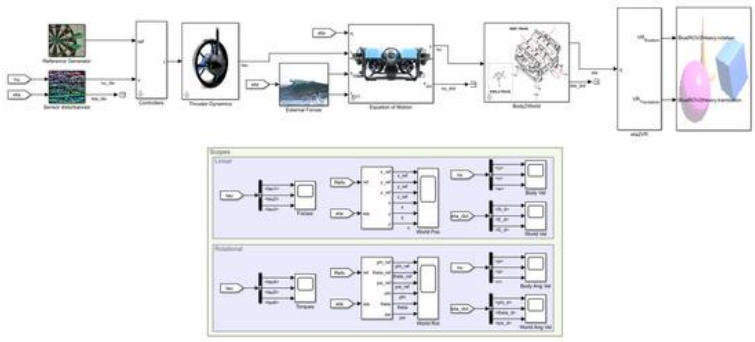


Figure 4.1: the simulink scheme of the simulator used for the experience that represents the main component and some visualize data tools

From the software is possible to set some parameters like:

- Starting position and speed

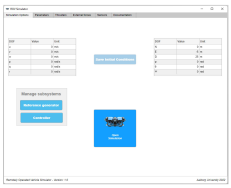


Figure 4.2: window of setting initial position and speed

- Vehicle's and environment's parameters

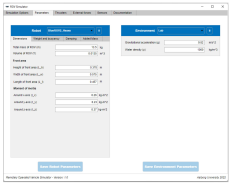


Figure 4.3: window of setting Vehicle's and environment's parameters

- The number of thrusters, their position and the output limit

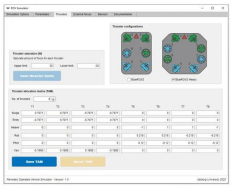


Figure 4.4: window of setting of the number of thrusters, their position and the output limit

- The external forces: ocean current, tether and manual force

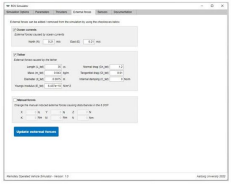


Figure 4.5: window of setting of the external forces

- The sensors used

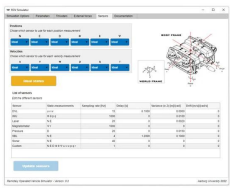


Figure 4.6: window of setting of the sensors used

4.1 CONTROLLERS

The simulator use a SMC to calculate the necessary force for each DOF of the body frame to apply and after it convert the value in the world frame

4.2 THRUSTER DYNAMICS

This block has the task of calculate the force to apply to each thruster in order to obtain the force desired in the appropriate DOF

According to the documentation [7] in order to link the vector of forces and moments applied to the vehicle t to the voltages Vi applied to each thruster, they use the expression:

$$\tau = T_K(s)F(V) \tag{4.1}$$

where:

- T
  is the so-called thrust configuration matrix; that change in base of the hardware setup chosen
- K(s)
  Is a diagonal matrix that contains unity DC-gain transfer functions accounting for the dynamic relation between Vi and the force F
- F(v)
  Is the vector "i cui elementi rappresentano" the force of each thruster "in base" of the voltage input; It is obtained by a nonlinear regression using data from experiments conducted in prior work.
  This block is dedicate to calculate the values of this vector

4.3 External Force

That component simulate the behaviour of the external force like the ocean current ("settabile" from the starting setting) and how the tether interact with the rov

The tether disturbs the ROV due to the forces generated by the drag and underwater currents so in the simulator to "ricreare" this effect "è stato implementato" a model of the tether from

4.4 Equations of motion

Here the simulator emulate the body frame's behaviour to the application of the thrusters and external force

This block use the equation [2.1] and [2.2] describe in the "ROV's model" section

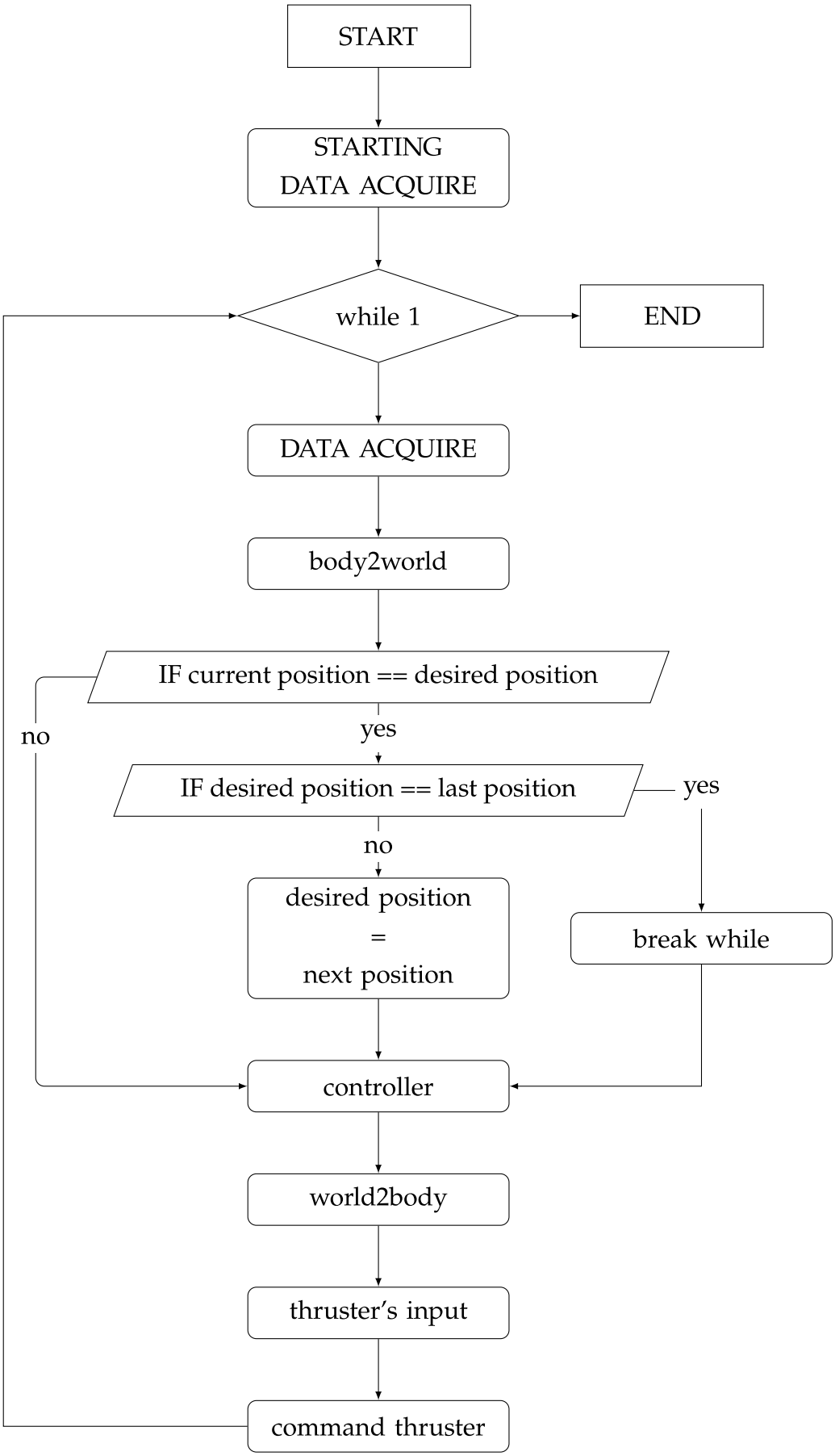
4.5 Body2World

At last the body frame behaviour are converted in the world frame

5

Program development

The program is written in Python, and its structure is depicted in the following block diagram. Acknowledgements from the Ardusub site [1] and the BlueRobotics forum [2] were utilized in its development. The code was structured into sub-blocks to facilitate future modifications.


START
STARTING
DATA ACQUIRE
while 1
END
DATA ACQUIRE
body2world
IF current position == desired position
no
yes
IF desired position == last position
yes
no
desired position
=
next position
break while
controller
world2body
thruster's input
command thruster

5.1 Check position

The position targets for the vehicle are set in a matrix format, where the number of rows corresponds to the number of desired set points. The first six columns represent the x, y, and z positions from the starting point, as well as the pitch, roll, and yaw positions. The last value in each row represents the time at which the vehicle should reach the set point.
If the current position of the vehicle matches the desired set point, the program will check if it is the last set point. If not, it will move on to the next set point and continue navigating. If it is the last set point, the program will disarm the vehicle and end the navigation.

5.2 Data acquire

This block is dedicated to acquiring data from the sensor, specifically the acceleration in the body frame, time from system boot, and angular position (roll, pitch, and yaw) and speed. The position of the data is recorded on the Ardusub site [1]. Subsequently, the program calculates the world frame position from the sensor data and the previous acquisition time, with the origin being the starting point.

```
import time
import sys
from world_body import w2b,b2w

# Import mavutil
from pymavlink import mavutil

V=[0,0,0,0,0,0]   #posizione
Vd=[0,0,0,0,0,0]  #velocita
Vaccw=[0,0,0]   #acc world frame
Vaccb=[0,0,0]   #acc body frame
Vxyzm=[0,0,0]   #acc xyz data
Ti=[0]

def acq_sensor(V,Vd,Ti):

  # Create the connection
  master = mavutil.mavlink_connection('udpin:0.0.0.0:14550')
```

```
19
20  # Wait a heartbeat before sending commands
21  master.wait_heartbeat()
22
23  # Request all parameters
24  master.mav.param_request_list_send(master.target_system, master.
     target_component)
25
26  V[3]=round(master.messages['ATTITUDE'].roll,2)                  #
     assegn roll in rad
27  V[4]=round(master.messages['ATTITUDE'].pitch,2)                 #
     assegn pitch
28  V[5]=round(master.messages['ATTITUDE'].yaw,2)                 #assegn
      yaw
29  Vd[3]=master.messages['ATTITUDE'].rollspeed              #assegn
     rollspeed
30  Vd[4]=master.messages['ATTITUDE'].pitchspeed             #assegn
     pitchspeed
31  Vd[5]=master.messages['ATTITUDE'].yawspeed             #assegno
     yawspeed
32
33  Tn=master.messages['SYSTEM_TIME'].time_boot_ms*pow(10,-3)
       #acq time now [ms]+ conv [ms]>[s]
34
35
36  while True:
37    msg = master.recv_match()
38    if not msg:
39      continue
40    if msg.get_type() == 'RAW_IMU':
41      data = str(msg)                          #acq stringa whit sensor
     data
42
43      data = data.split(":")
44
45      Xacc = -int(data[2].split(",")[0])             #acq Xacc body
     frame
46
47      Yacc = int(data[3].split(",")[0])            #acq Yacc body
     frame
48
49      Zacc = int(data[4].split(",")[0])            #acq Zacc body
     frame
```

```
50
51      break
52
53  Vaccb[0]=(Xacc)*(9.80665/1000)                          #conversion in
     m/s/s
54  Vaccb[1]=(Yacc)*(9.80665/1000)                          #conversion in
     m/s/s
55  Vaccb[2]=(Zacc)*(9.80665/1000)                          #conversion in
     m/s/s
56
57  Vaccw=b2w(V,Vaccb)                                  #from body frame to
     world frame
58
59  Vaccw[2]=Vaccw[2]+9.80665                             #cut gravity
     component
60
61
62
63  DT=Tn-Ti[0]                                     #calc delta t
64
65  Ti[0]=Tn
66
67  Vd[0]=Vd[0]+Vaccw[0]*DT                               #calc speed x
68  Vd[1]=Vd[1]+Vaccw[1]*DT                               #calc speed y
69  Vd[2]=Vd[2]+Vaccw[2]*DT                               #calc speed z
70
71  V[0]=round(V[0]+Vd[0]*DT+0.5*Vaccw[0]*pow(DT,2),2)
        #calc position x
72  V[1]=round(V[1]+Vd[1]*DT+0.5*Vaccw[1]*pow(DT,2),2)
        #calc position y
73  V[2]=round(V[2]+Vd[2]*DT+0.5*Vaccw[2]*pow(DT,2),2)
        #calc position z
74
75  return (Xacc,Yacc,Zacc)
```

Code 5.1: acquisition data

Finally, all of the collected data is printed into a text file for further post-processing analysis.

```
Acc_xyz=acq_sensor(V,Vd,Ti)

Acc_xyz=str(Acc_xyz)
Vtx=str(V)
Vdtx=str(Vd)
Titx=str(Ti)
with open('data_sensor.txt', 'a') as f:
    f.write('Acc_xyz:')
    for line in Acc_xyz:
        f.write(line)
    f.write('V:')
    for line in Vtx:
        f.write(line)
    f.write('Vd:')
    for line in Vdtx:
        f.write(line)
    f.write('Ti:')
        f.write(Titx)
    f.write('\n')
```

Code 5.2: Data writer

5.3 CONTROLLERS

To replicate the behavior of the simulator accurately, I employed the same controller, namely a Sliding Mode controller, as used in the simulator.

```
import math
from world_body import w2b,b2w

V=[0,0,0,0,0,0]
Vd=V
Pref=V
Pstart=V

def Xcontroller(V,Vd,Pref,Pstart):

  #Xref(xposition_ref,xspeed_ref,xacc_ref)
  XRef=[Pref[0],(Pref[0]-Pstart[0])/Pref[6],(Pref[0]-Pstart[0])/pow(
    Pref[6],2)]


  #power limiter
  Pl=85

  #costant controller
  L=1.17
  epsilon=0.08
  c0=1
  alpha=0.1

  # ASMC
  f=1.62093244040397621574811637740158*Vd[5]*Vd
    [1]-0.0503609016144395127412280999124 05*Vd
    [0]*(32.77999999999883584678173065 1855*abs(Vd[0])
    +26.09999999999854480847716331 4819)
    -0.0642907270009942521937773562967 78*math.sin(V[4])
    -1.0384720227252728363964706659317*Vd[4]*Vd[2]
  gt=0.0503609016144395127412280999124 05*1

  # Control error(s)
  e=V[0]-XRef[0]
  e_d=Vd[0]-XRef[1]

  # Original constraint function(s)
```

```
  sigma = e_d+c0*e

  # % Switching gain
  rho=alpha/math.sqrt(2)+L
  v=-rho*(sigma)/(abs(sigma)+epsilon)

  # Control input signal
  u=pow(gt,-1)*(XRef[2]-c0*e_d+v)

  #power limiter
  if (u<(-Pl)):
    u=-Pl

  if (u>Pl):
    u=Pl

  return u

def Ycontroller(V,Vd,Pref,Pstart):

  #Yref(yposition_ref,yspeed_ref,yacc_ref)
  YRef=[Pref[1],(Pref[1]-Pstart[1])/Pref[6],(Pref[1]-Pstart[1])/pow(
   Pref[6],2)]

  #power limiter
  Pl=85

  #costant controllorer
  L=1.04
  epsilon=0.03
  c0=3
  alpha=0.5

  # ASMC
  f=0.06190896393363942706855596043169 5*math.cos(V[4])*math.sin(V[3])
    -0.048495193430703409376292256638408*Vd
    [1]*(50.940000000002328306436538696289*abs(Vd[1])+20.0)
    -1.560882146974108763970434665679 9*Vd[5]*Vd
    [0]+0.962953241027776130067650228738 78*Vd[5]*Vd[2]
  gt=0.048495193430703409376292256638408

  # Control error(s)
  e=V[1]-YRef[0]
```

```
71    e_d=Vd[1]-YRef[1]
72
73    # Original constraint function(s)
74    sigma = e_d+c0*e
75
76    # % Switching gain
77    rho=alpha/math.sqrt(2)+L
78    v=-rho*(sigma)/(abs(sigma)+epsilon)
79
80    # Control input signal
81    u=pow(gt,-1)*(YRef[2]-c0*e_d+v)
82
83    #power limiter
84    if (u<(-Pl)):
85      u=-Pl
86
87    if (u>Pl):
88      u=Pl
89
90    return u
91
92  def Zcontroller(V,Vd,Pref,Pstart):
93
94    #Zref(Zposition_ref,Zspeed_ref,Zacc_ref)
95    ZRef=[Pref[2],(Pref[2]-Pstart[2])/Pref[6],(Pref[2]-Pstart[2])/pow(
       Pref[6],2)]
96
97    #power limiter
98    Pl=120
99
100   #costant controllorer
101   L=0.7
102   epsilon=0.05
103   c0=2
104   alpha=0.1
105
106   # ASMC
107   f = 0.039662804814355467897257767617702*math.cos(V[3])*math.cos(V
       [4]) - 0.031069093540931902452939539216459*Vd
       [1]*(47.330000000001746229827404022217*abs(Vd[2]) +
       36.339999999996507540345191955566) -
       0.61692885839886457688407972455025*Vd[3]*Vd[1] +
       0.64066335945904029358644038438797*Vd[4]*Vd[0]
```

```
  gt =0.031069093540931902452939539216459

  # Control error(s)
  e=V[2]-ZRef[0]
  e_d=Vd[2]-ZRef[1]

  # Original constraint function(s)
  sigma = e_d+c0*e

  # % Switching gain
  rho=alpha/math.sqrt(2)+L
  v=-rho*(sigma)/(abs(sigma)+epsilon)

  # Control input signal
  u=pow(gt,-1)*(ZRef[2]-c0*e_d+v)

  #power limiter
  if (u<(-Pl)):
    u=-Pl

  if (u>Pl):
    u=Pl

  return u

def THETcontroller(V,Vd,Pref,Pstart):

  #THETref(position_ref,speed_ref,acc_ref)
  THETRef=[Pref[4],(Pref[4]-Pstart[4])/Pref[6],(Pref[4]-Pstart[4])/
    pow(Pref[6],2)]

  #power limiter
  Pl=16

  #costant controllorer
  L=1.83
  epsilon=0.1
  c0=0.4
  alpha=0.1

  # ASMC
  f = -0.064291*math.sin(V[4]) - 0.050361*Vd[0]*(32.78*abs(Vd[0]) +
    26.1) + 1.6209*Vd[5]*Vd[1] - 1.0385*Vd[4]*Vd[2]
```

```
149   gt =2.7410526
150
151   # Control error(s)
152   e=V[4]-THETRef[0]
153   e_d=Vd[4]-THETRef[1]
154
155   # Original constraint function(s)
156   sigma = e_d+c0*e
157
158   # % Switching gain
159   rho=alpha/math.sqrt(2)+L
160   v=-rho*(sigma)/(abs(sigma)+epsilon)
161
162   # Control input signal
163   u=pow(gt,-1)*(THETRef[2]-c0*e_d+v)
164
165   #power limiter
166   if (u<(-Pl)):
167     u=-Pl
168
169   if (u>Pl):
170     u=Pl
171
172   return u
173
174 def PSIcontroller(V,Vd,Pref,Pstart):
175
176   #PSIref(position_ref,speed_ref,acc_ref)
177   PSIRef=[Pref[5],(Pref[5]-Pstart[5])/Pref[6],(Pref[5]-Pstart[5])/pow
       (Pref[6],2)]
178
179   #power limiter
180   Pl=22
181
182   #costant controllorer
183   L=2.05
184   epsilon=0.1
185   c0=2
186   alpha=0.1
187
188   # ASMC
189   f = -0.064291*math.sin(V[4]) - 0.050361*Vd[0]*(32.78*abs(Vd[0]) +
       26.1) + 1.6209*Vd[5]*Vd[1] - 1.0385*Vd[4]*Vd[2]
```

```
190   gt =1.690587
191
192   # Control error(s)
193   e=V[5]-PSIRef[0]
194   e_d=Vd[5]-PSIRef[1]
195
196   # Original constraint function(s)
197   sigma = e_d+c0*e
198
199   # % Switching gain
200   rho=alpha/math.sqrt(2)+L
201   v=-rho*(sigma)/(abs(sigma)+epsilon)
202
203   # Control input signal
204   u=pow(gt,-1)*(PSIRef[2]-c0*e_d+v)
205
206   #power limiter
207   if (u<(-Pl)):
208     u=-Pl
209
210   if (u>Pl):
211     u=Pl
212
213   return u
```

Code 5.3: controllers

5.4 WORLD 2 BODY/BODY 2 WORLD

These components are designed to convert the sensor data from the body frame to the world body frame and vice versa for the controller input. It is worth noting that roll, pitch, and yaw do not require body-to-world transformation because they are evaluated from the internal program in the world frame.

```
import numpy
import math

V=[0,0,0,0,0,0]
B=[0,0,0]
C=[0,0,0,0,0,0]

def w2b(V,C):
  J=numpy.array([[math.cos(V[5])*math.cos(V[4]),
         -math.cos(V[4])*math.sin(V[5]),                          math
    .sin(V[4])],
          [math.cos(V[5])*math.sin(V[3])*math.sin(V[4])-math.cos(V
    [3])*math.sin(V[5]),  math.cos(V[3])*math.cos(V[5])+math.sin(V[3])
    *math.sin(V[5])*math.sin(V[4]),   math.cos(V[4])*math.sin(V[3])],
          [math.sin(V[3])*math.sin(V[5])+math.cos(V[3])*math.cos(V
    [5])*math.sin(V[4]),  math.cos(V[3])*math.sin(V[5])*math.sin(V[4])
    -math.cos(V[5])*math.sin(V[3]),   math.cos(V[3])*math.cos(V[4])]])

  T=numpy.array([ [1,   0,          -math.sin(V[4])],
          [0,   math.cos(V[3]),   math.cos(V[4])*math.sin(V[3])],
          [0,   -math.sin(V[3]),  math.cos(V[3])*math.cos(V[4])]])


  Vt1=numpy.array([C[0],C[1],C[2]])
  u1=numpy.dot(J,numpy.transpose(Vt1))

  Vt2=numpy.array([C[3],C[4],C[5]])
  u2=numpy.dot(T,numpy.transpose(Vt2))

  u=numpy.concatenate([u1,u2])

  return u

def b2w(V,B):
  J=numpy.array([[math.cos(V[5])*math.cos(V[4]),
         -math.cos(V[4])*math.sin(V[5]),                          math
```

```
    .sin(V[4])],
30          [math.cos(V[5])*math.sin(V[3])*math.sin(V[4])-math.cos(V
    [3])*math.sin(V[5]),  math.cos(V[3])*math.cos(V[5])+math.sin(V[3])
    *math.sin(V[5])*math.sin(V[4]),   math.cos(V[4])*math.sin(V[3])],
31          [math.sin(V[3])*math.sin(V[5])+math.cos(V[3])*math.cos(V
    [5])*math.sin(V[4]),  math.cos(V[3])*math.sin(V[5])*math.sin(V[4])
    -math.cos(V[5])*math.sin(V[3]),   math.cos(V[3])*math.cos(V[4])]])
32
33  Jinv=numpy.linalg.inv(J)
34
35  Vt=numpy.array([B[0],B[1],B[2]])
36  u=numpy.dot(Jinv,numpy.transpose(Vt))
37
38
39  return u
```

Code 5.4: world body

5.5 THRUSTER'S INPUT

The thrusters in the system are regulated by a Pulse Width Modulation (PWM) signal. The duty cycle of the PWM signal is utilized to determine the power output of the thrusters.

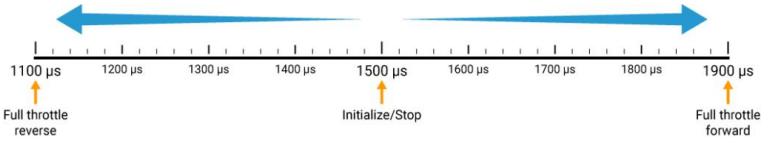


Figure 5.1: A diagram that show how the period of the Pulse-Width-Modulation (PWM) signal command the thruster behaviour found on bluerobotic's site [**BlueRobotics**]

Assuming a linear relationship between the power output and the duty cycle of the Pulse Width Modulation (PWM) signal, I have employed a simple conversion method to obtain the input signal, utilizing the maximum value specified in the simulator's thesis [7].

```
#Input calc
Im[0]=int(1500-N[0]*(400/84))
Im[1]=int(1500-N[1]*(400/84))
Im[2]=int(1500-N[2]*(400/120))
Im[3]=int(1500-N[3]*(400/26))
Im[4]=int(1500-N[4]*(400/14))
Im[5]=int(1500+N[5]*(400/22))

for i in range(5): #security end run
    if Im[i]>1900:
        Im[i]=1900
    if Im[i]<1100:
        Im[i]=1100
```

Code 5.5: thrusters input

5.6 COMMAND THRUSTER

For this section, I employed a program that was developed by a BluRobotic's developer, which can be found in [4].
Within the script, there are two methods for controlling the thruster's behavior:

- The first method involves manually sending the duty cycle value to each thruster individually, but the developer discourages this approach as it turns off all the security systems of the ROV.
- The second method involves specifying how the ROV should behave with respect to the relative degrees of freedom (DOF).

6 Test

Initially, I planned to conduct some basic movement tests to evaluate how the ROV would behave in the physical world following predefined commands. Subsequently, I intended to perform more complex paths such as rectangular trajectories. The objective was to compare the data collected during the field tests with the simulated ones and identify any discrepancies that occurred and when they happened.

6.1 Site of the test

Figure 6.1: Satellite photos that show the site of the test and the near departments of UNIPD

- It was decided to conduct the tests on the nearby Piovego river due to its close proximity to the laboratories, which would be convenient in case of any need for further testing or adjustments.

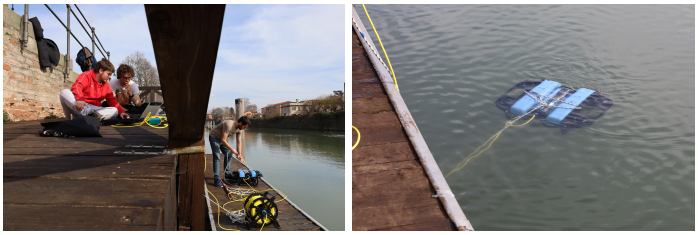


Figure 6.2: To the left, a photo showing how we were arranged on the platform. To the right, a photo of the BlueRov2 in the water

6.2 Preparation

In this section, I will describe the preparation of the equipment needed for the tests

- The BlueROV2's battery was fully charged, as well as the computer's battery. However, due to the high power consumption of the computer programs, we brought along some spare batteries to replace them if needed during the tests
- For the testing, the ROV was secured to a rope to facilitate its immersion and retrieval from the river.
- In addition, prior to the test, the program was modified to enable the thrusters that had been disabled during the setup process outside of the water for safety reasons. Additionally, a separate script was prepared in case it was necessary to halt all of the thrusters.
- The controls for the joystick in QGroundControl [5] were configured to pre-position the vehicle at the center of the river before the test, with the aim of avoiding obstacles during the simulation.
- A Python script was developed to organize the sensor data and generate plots for analysis and visualization purposes.

```
import numpy as np
import matplotlib.pyplot as plt

#read data
with open('data_sensor.txt') as f:
    contents = f.read()
data = str(contents)
data = data.split("\n")
data=data[:len(data)-1]

#time
T=np.zeros(len(data))
for i in range(len(data)):
  data_t=data[i].split('Ti:')
  data_t=data_t[1]
  data_t=data_t[1:len(data_t)-1]
  data_t=float(data_t)
  T[i]=data_t-T[0]
T[0]=0

```

```
#position x
X=np.zeros(len(data))
for i in range(len(data)):
  data_x=data[i].split(':')
  data_x=data_x[2].split(',')
  data_x=data_x[0]
  data_x=data_x[1:]
  X[i]=float(data_x)

#position Y
Y=np.zeros(len(data))
for i in range(len(data)):
  data_y=data[i].split(':')
  data_y=data_y[2].split(',')
  data_y=data_y[1]
  data_y=data_y[1:]
  Y[i]=float(data_y)

#position Z
Z=np.zeros(len(data))
for i in range(len(data)):
  data_z=data[i].split(':')
  data_z=data_z[2].split(',')
  data_z=data_z[2]
  data_z=data_z[1:]
  Z[i]=float(data_z)

#position roll
Roll=np.zeros(len(data))
for i in range(len(data)):
  data_r=data[i].split(':')
  data_r=data_r[2].split(',')
  data_r=data_r[3]
  data_r=data_r[1:]
  Roll[i]=float(data_r)

#position pinch
Pinch=np.zeros(len(data))
for i in range(len(data)):
  data_p=data[i].split(':')
  data_p=data_p[2].split(',')
  data_p=data_p[4]
  data_p=data_p[1:]
  Pinch[i]=float(data_p)

#position yaw
Yaw=np.zeros(len(data))
for i in range(len(data)):
  data_ya=data[i].split(':')
  data_ya=data_ya[2].split(',')
  data_ya=data_ya[5]
  data_ya=data_ya[1:len(data_ya)-3]
  Yaw[i]=float(data_ya)

#acceleration X
Xacc=np.zeros(len(data))
for i in range(len(data)):
  data_xacc=data[i].split(':')
  data_xacc=data_xacc[1].split(',')
  data_xacc=data_xacc[0]
  data_xacc=data_xacc[1:]
  Xacc[i]=float(data_xacc)

#acceleration Y
Yacc=np.zeros(len(data))
for i in range(len(data)):
  data_yacc=data[i].split(':')
  data_yacc=data_yacc[1].split(',')
  data_yacc=data_yacc[1]
  data_yacc=data_yacc[1:]
```

```
  Yacc[i]=float(data_yacc)

#acceleration Z
Zacc=np.zeros(len(data))
for i in range(len(data)):
  data_Zacc=data[i].split(':')
  data_Zacc=data_Zacc[1].split(',')
  data_Zacc=data_Zacc[2]
  data_Zacc=data_Zacc[1:len(data_Zacc)-2]
  Zacc[i]=float(data_Zacc)

#plot data
plt.figure(1)

plt.subplot(611)
plt.plot(T,X)
plt.ylabel('X[m]')
plt.xlabel('Time[s]')
plt.grid(True)

plt.subplot(612)
plt.plot(T,Y)
plt.ylabel('Y[m]')
plt.xlabel('Time[s]')
plt.grid(True)

plt.subplot(613)
plt.plot(T,Z)
plt.ylabel('Z[m]')
plt.xlabel('Time[s]')
plt.grid(True)

plt.subplot(614)
plt.plot(T,Roll)
plt.ylabel('Roll[rad]')
plt.xlabel('Time[s]')
plt.grid(True)

plt.subplot(615)
plt.plot(T,Pinch)
plt.ylabel('Pinch[rad]')
plt.xlabel('Time[s]')
plt.grid(True)

plt.subplot(616)
plt.plot(T,Yaw)
plt.ylabel('Yaw[rad]')
plt.xlabel('Time[s]')
plt.grid(True)

plt.suptitle('Position')


plt.figure(2)

plt.subplot(311)
plt.plot(T,Xacc)
plt.ylabel('Xacc[mg]')
plt.xlabel('Time[s]')
plt.grid(True)

plt.subplot(312)
plt.plot(T,Yacc)
plt.ylabel('Yacc[mg]')
plt.xlabel('Time[s]')
plt.grid(True)

plt.subplot(313)
plt.plot(T,Zacc)
plt.ylabel('Zacc[mg]')
plt.xlabel('Time[s]')
```

```
162 plt.grid(True)
163
164 plt.suptitle('Acceleration measured')
165
166 plt.show()
```

Code 6.1: data vector generator

6.3 SUGGESTIONS FOR INSTRUMENTATION

In this section, I will provide some suggestions regarding the equipment setup for future tests.

- The rope used to tie the ROV wasn't waterproof, so during the water test, it gained weight and caused some buoyancy problems. For future tests, I recommend using a waterproof rope
- To study the ROV's behavior, we did it visually due to a lack of instruments. For future tests, I recommend using reference points such as buoys to have a better understanding of its movement in the real world

7 conclusion

7.1 Summary of conclusions

Unfortunately, the program did not function as expected during the test. Although the thrusters appeared to be spinning in the correct direction, they only activated sporadically, which affected their overall performance. As a result, it appeared that they did not exert enough force to move the vehicle efficiently. Despite my best efforts to troubleshoot and resolve the issue during the test, I was unsuccessful. In hindsight, I realized that my initial assumption about the linear gain of the thrusters was incorrect. Upon conducting further research, I discovered a diagram that clearly represented the thruster output in relation to the input signal.

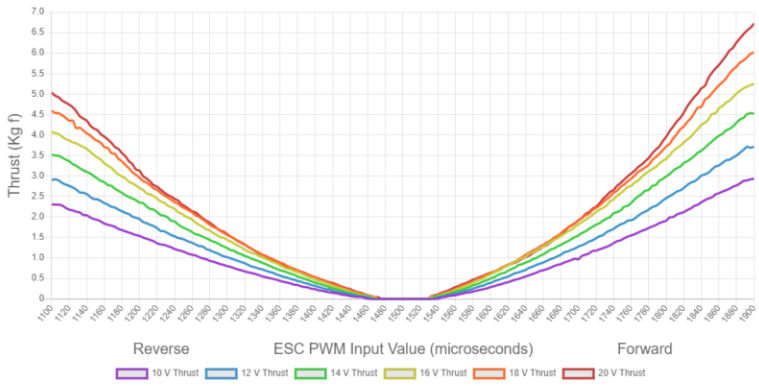


Figure 7.1: Diagram that shows in detail how the period of the PWM signal and the Output force of a thruster T200 are correlate in base of the supply voltage found on the BlueRobotics site [3]

In addition to the issues previously mentioned, it is worth noting that the thrusters appeared to respond correctly to external forces during the test. Furthermore, the script was able to save the sensor data as predicted

7.2 Future projects

Due to time constraints, I am unable to continue working on this project at the moment. However, I suggest the following features as potential areas for future work:

- To improve the performance of the ROV, the block responsible for generating the input signals should be completely rewritten. This is necessary to ensure that the controller receives the correct power output, which will ultimately lead to more efficient and reliable thruster performance.
- A more in-depth study is necessary to determine whether a Sliding Mode Control (SMC) is the optimal controller for this experiment. It may also be beneficial to write a program that is capable of switching between different controllers based on the desired path and environmental conditions.
- It is possible that the sporadic behavior of the thrusters could be due to one of the security systems that the developer mentioned to me. It appears that if the thrusters do not receive an input for a certain period of time, they will stop. To address this issue, it may be necessary to accelerate the data acquisition program in order to reduce the time of the cycle.
- During the testing phase, another issue was identified with the acceleration readings of the body frame. The data obtained showed inaccuracies due to the noise, which in turn affected the position evaluation. To tackle this problem, it would be advisable to implement a Kalman filter, which can improve the accuracy of the readings by incorporating additional sensors. This approach can help enhance the reliability of the data and improve the performance of the ROV in its intended tasks.

Bibliography

[1] ArduSub Homepage. 15/2/2022. URL: https://www.ardusub.com/.

[2] BlueRobotics forum(2022). 15/2/2022). URL: https://discuss.bluerobotics.com/.

[3] BlueROV2 (2022). 15/2/2022. URL: https://bluerobotics.com/store/rov/bluerov2/.

[4] Eliot. rc override example.py. 20/2/2022). URL: https://gist.github.com/ES-Alexander/ee1dd479dd728b7cef1bf936f9114e71.

[5] QGroundControl software. 17/2/2022. URL: www.qgroundcontrol.org.

[6] Erik Rowe et al. "Implementation of a quaternion-based PD controller in ROS2 for a generic underwater vehicle with six degrees of freedom". B.S. thesis. NTNU, 2022.

[7] Malte von Benzon et al. "An Open-Source Benchmark Simulator: Control of a BlueROV2 Underwater Robot". English. In: Journal of Marine Science and Engineering 10.12 (Dec. 2022). ISSN: 2077-1312. DOI: 10.3390/jmse10121898.