Programming of an Autonomous Sailboat

Internship Report

Ewen MELE

FISE26 - Autonomous Robotics

May to September 2025

Supervisor: Prof. Jian Wan, Aston University j.wan3@aston.ac.uk

Acknowledgements

I would like to sincerely thank Prof. Jian Wan for giving me the opportunity to carry out my internship at Aston University, as well as for his valuable help and guidance during the boat's development and testing. I would also like to thank my family, without whom this internship abroad would not have been possible.

Abstract

The following report aims to present the results of my four-month internship at Aston University in Birmingham, under the supervision of Prof. Jian Wan, Lecturer in Mechatronics and Robotics. The goal of this internship was to program a simple sailboat so that it could autonomously travel from one point to another using an already known and tested algorithm. This internship and the corresponding report can be divided into three parts: getting familiar with the hardware, implementing the algorithm, and finally a testing phase.

Résumé

Ce rapport a pour but de présenter les résultats de mon stage de quatre mois réalisé à l'Université d'Aston, à Birmingham, sous la supervision du Pr. Jian Wan, maître de conférences en mécatronique et robotique. L'objectif de ce stage était de programmer un voilier simple afin de lui permettre de se déplacer de manière autonome d'un point à un autre en utilisant un algorithme déjà connu et testé. Ce stage et ce rapport peuvent-être décomposés en trois parties : la prise en main du hardware, l'implémentation de l'algorithme et enfin une phase de tests.

Contents

Acknowledgements **i**

Abstract **ii**

Résumé **ii**

Contents **iii**

List of Figures **iv**

Introduction **1**

1 Hardware Presentation **1**
1.1 Boat . . . 1
1.2 Main board . . . 2
1.3 IMU . . . 2
1.4 GPS . . . 3
1.5 Anemometer . . . 3
1.6 Remote controller . . . 4
1.7 SD Card Adapter . . . 4
1.8 Storage box . . . 5

2 Algorithm **6**
2.1 Global structure . . . 6
2.2 True wind computing . . . 6
2.3 Linefollowing controler . . . 7
2.4 Adding an integral correction to the linefollowing algorithm . . . 8
2.5 Path following . . . 9
2.6 Mission's unfolding . . . 9

3 Tests results **10**
3.1 Introduction . . . 10
3.2 First observations . . . 11
3.3 GPS issues . . . 12
3.4 Anemometer responsiveness . . . 13
3.5 Rudder command saturation . . . 14
3.6 Turning against the wind . . . 15

Conclusion **15**

References **17**

Appendices **18**

List of Figures

1 Sailboat used during the internship . . . 1
2 Arduino Mega 2560 Board . . . 2
3 Grove - Mega shield v1.2 . . . 2
4 Adafruit 16-Channel 12-bit PWM/Servo Shield . . . 2
5 CMPS12 IMU as shown in its documentation . . . 3
6 GPS Grove SEN10752P . . . 3
7 Anemometer . . . 4
8 The Joysway J4C05 Remote controller and its J5C01R receiver . . . 5
9 A micro SD card adapter for Arduino . . . 5
10 Foot of the box . . . 6
11 Main body of the box . . . 6
12 Lid of the box . . . 7
13 Maintaining piece of the IMU . . . 7
14 Code's architecture . . . 7
15 The boat on Bournville's lake . . . 10
16 The four trajectories used to test the boat on Bournville's lake . . . 11
17 Trajectory followed by the boat when trying to follow path 1 . . . 11
18 True wind data computed by the boat while following the line . . . 12
19 Highlight of some points where the GPS did not take good measurements on path 1 . . . 12
20 Highlight of some points where the GPS did not take good measurements on path 2 . . . 13
21 Comparison between the true wind computed with and without the SOG while following path 1 . . . 13
22 Evolution of the SOG calculated using GPS data over time . . . 14
23 Comparison between the true wind's direction and the heading while following path 0 . . . 14
24 Evolution of the servomotor's commands while following path 0 . . . 15

Introduction

I have had the opportunity, from May to September 2025, to complete my second-year internship at Aston University, Birmingham, under the supervision of Prof. Jian Wan. This internship consisted in programming a small and simple autonomous sailboat controlled by an Arduino Mega board so that it could sail from one point to another and follow simple trajectories. Nowadays, the development of autonomous sailboats can be seen as a promising solution for long missions at sea, such as data collection, as they only require little energy to sail. Events such as the Microtransat Challenge [1] also encourage the development of these autonomous sailboats, which present several challenges, the most significant one being that this kind of boat relies heavily on the weather conditions and sailing can be difficult and even dangerous if they are not adapted. No particular use other than teaching was aimed for the boat developed during this internship. It is used for such purpose by Prof. Jian Wan as it is quite modular and easily transportable. However, some of the results obtained during this internship, such as test data or design elements, may still be used

1 Hardware Presentation

1.1 Boat

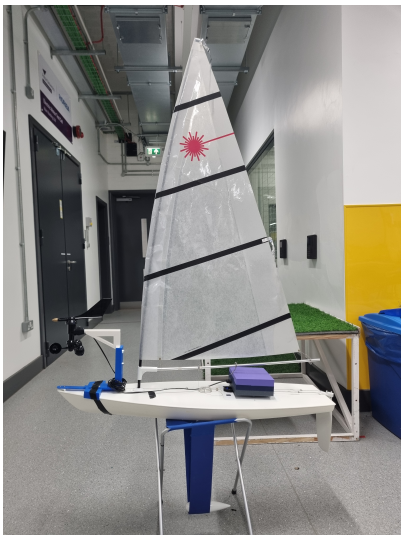

Figure 1: Sailboat used during the internship

The boat used during the internship is a simple sailboat that is modular and easy to carry. It only has a single sail, which while not the most efficient solution, makes it easy to disassemble and assemble back. The boat also has a rudder to control its direction and a keel to prevent it from capsizing. Both of these components can be easily disassembled, too.

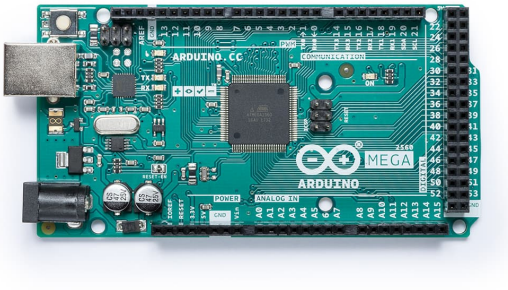

Figure 2: Arduino Mega 2560 Board

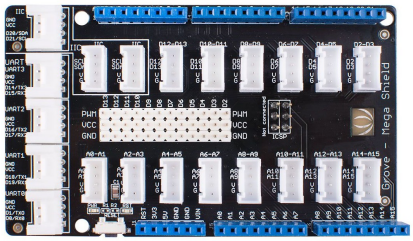

Figure 3: Grove - Mega shield v1.2

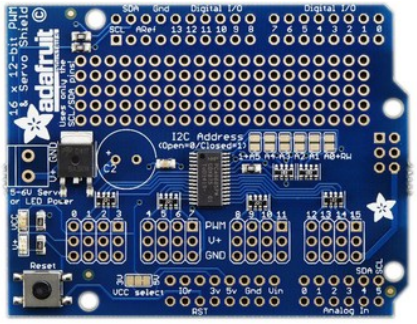

Figure 4: Adafruit 16-Channel 12-bit PWM/Servo Shield

1.2 Main board

The boat is controlled by an Arduino Mega plugged to two shields: a Grove Mega shield, and an Adafruit PWM and Servo shield. This configuration allows us to easily use all the sensors we need to make the boat work and to use its two servomotors simultaneously with simple code. It also allows for some modularity, with the possibility of easily adding new components or changing the current ones.

1.3 IMU

The IMU used on the boat is a CMPS12 from *Robot Electronics*. As explained on the documentation[2], communication with the IMU can either be established via I2C or Serial. The IMU is self-calibrating and only requires some simple movements to complete its calibration. The IMU has 3-axis magnetometer, accelerometer and gyrometer, giving it 9 degrees of freedom. During this project, only the magnetometer data have been used. Please note that the code used for the IMU calibration has been taken and adapted from [3].

[1] `https://www.microtransat.org/`
[2] `https://www.robot-electronics.co.uk/files/cmps12.pdf`

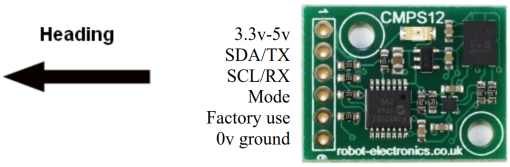

Figure 5: CMPS12 IMU as shown in its documentation

1.4 GPS

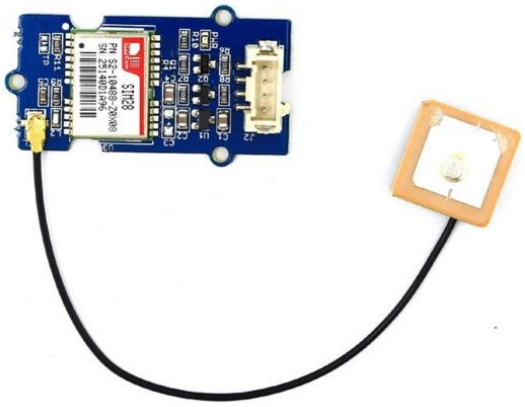

Figure 6: GPS Grove SEN10752P

In order to determine the boat position, as well as its Speed Over Ground (SOG) and Course Over Ground (COG), we used a GPS Grove SEN10752P module from RedOhm. This module, composed of a board and an antenna, sends GPS data following the NMEA norm via a serial connection. As we will see later in this report, while this module proved to be sufficient to know the boat's position, it lacked the efficiency required to send exploitable SOG and COG values.

1.5 Anemometer

In order to measure the apparent wind (i.e. the wind in the boat's frame) speed and direction, the boat was equipped with a mechanical anemometer from Davis Instrument. The anemometer is connected to a custom board provided by Pr. Wan, which is connected to an analog pin and an interrupt pin on the Arduino Mega. The wind direction is measured through a wind vane on the top of the anemometer that is connected to a potentiometer. The wind direction is therefore obtained by reading the voltage on the analog pin. The wind speed is measured using a solid-state magnetic sensor according to the documentation[3]. This sensor can be considered as a simple odometer that sends a single signal each time the lower part of the anemometer makes a full turn. Therefore, the wind speed can be measured by

[3] `https://cdn.shopify.com/s/files/1/0515/5992/3873/files/7911_SS.pdf`

Figure 7: Anemometer

counting how many signals are received on the interrupt pin during a given amount of time and by then applying the following formula given in the anemometer's documentation :

$$V = 0.447 \cdot P * \frac{2.25}{T} \tag{1}$$

where V is the wind speed in meters per second, P is the number of pulses sent during the sampling period, and T is the sampling period given in seconds. The 0.447 factor is used to convert the speed, initially given in miles per hour, to meters by second, and the 2.25 factor is due to the anemometer's shape. By measuring the wind speed with a 2.25 period, as suggested in the documentation, the formula can therefore be simplified to :

$$V = 0.447 \cdot P \tag{2}$$

The resolution of the wind speed sensor therefore is 0.447 meters per second, or approximately 1.6 kilometers per hour.

1.6 Remote controller

In order to be able to manually control the boat, for example to regain control in case a problem occurs during tests, we need a remote controller. The one chosen here is a Joysway J4C05 that communicates with a J5C01R receiver using a 2.4GHz frequency. The receiver then sends PWM signals corresponding to the controller's inputs to the Arduino. Even though the controller's switches weren't designed for such purpose, we can also use them to change the scenario the boat follows without having to change its code, as they would otherwise be useless to control the boat.

1.7 SD Card Adapter

In order to collect data during tests, a micro SD card adapter is used. This device communicates with the Arduino board following the SPI protocol.

Figure 8: The Joysway J4C05 Remote controller and its J5C01R receiver

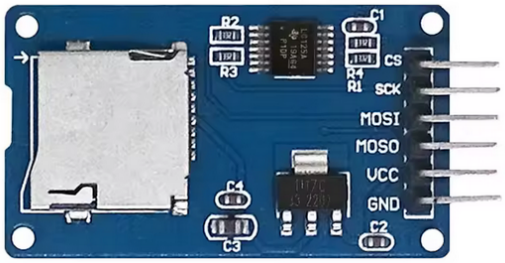

Figure 9: A micro SD card adapter for Arduino

1.8 Storage box

In order to be able to easily fix the component on the boat and not have to rearrange them between each test, it was decided to produce boxes that would fit on the boat and be able to store every required component. Several successive prototypes led to the following box design. It is made of four parts and has been designed to be easy to print, assemble and fix on the boat while protecting the components from water and holding them in their designated place, especially the IMU that needs to be completely still and flat on the boat. The four parts of the box are a foot [Figure 10] that keeps the box well fixed on the boat, a main body [Figure 11] where all the components go, a lid [Figure 12] to close the box, and finally a small piece [Figure 13] designed to be put on top of the IMU to hold it still. It has been decided to separate the foot from the main body of the box to make the whole faster to produce with 3D printers. Once the parts are printed, the foot is glued in a groove at the bottom of the main body. Once all components are put in the box, it can easily be "plugged" on the boat. Waterproof tape is then used to secure the lid on the box and prevent water from entering, to ensure that the components are safe from water. All designs have been created using Autodesk Inventor Pro 2025 and can be found in both .stl and .ipt formats on the project's GitHub repository in annexes. For future projects led by Pr. Wan, similar designs of a box intended to store a Raspberry Pi 4 board have been created using Solidworks. These designs can also be found on the project's GitHub repository.

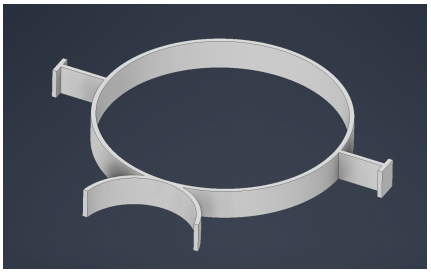

Figure 10: Foot of the box

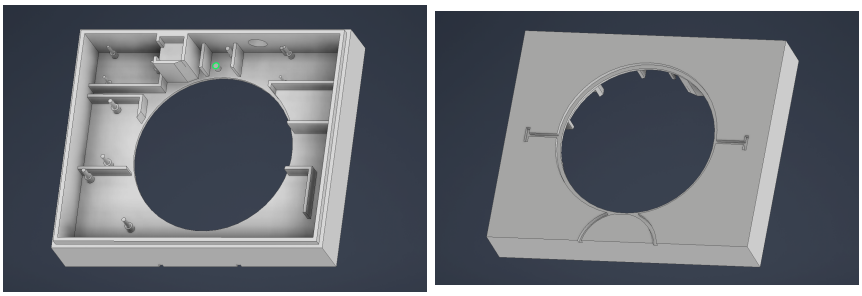

Figure 11: Main body of the box

2 Algorithm

2.1 Global structure

In order to make the best use of the C++ language, it has been decided to program the boat by creating a class to control each component. Those classes are then used by a central class, "nav", which collects data from each sensors to compute the proper command to send to the servomotors depending on the mission scenario chosen. The main.ino file then creates an instance of the nav class and calls the function corresponding to the mission that has to be done. A diagram of this architecture can be seen on Figure 14.

2.2 True wind computing

The first step to properly control an autonomous sailboat is to be able to compute the true wind. This requires to be able to measure the wind speed and heading in the boat's frame of reference as well as the boat Course Over Ground (COG) and Speed Over Ground (SOG). According to [1], considering the currents as negligible, the true wind in the earth frame of reference can be calculated using the formula:

$$TW = \begin{pmatrix} SOG \cdot \sin(COG) - AWS \cdot \sin(AWD) \\ SOG \cdot \cos(COG) - AWS \cdot \cos(AWD) \end{pmatrix}$$

Where AWS is the Apparent Wind Speed which corresponds to the wind speed measured by the anemometer and AWD is the Apparent Wind Direction which corresponds to the sum of the boat's heading and the wind direction in the boat's frame as measured by the anemometer. Unfortunately, it has been observed during tests during this project as well as during previous internships [4] that the GPS used on the boat is unable to provide us

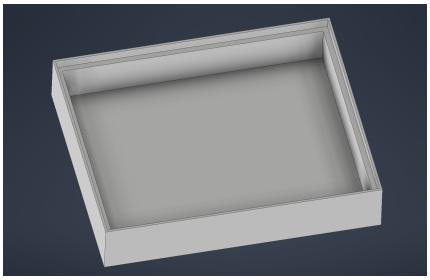

Figure 12: Lid of the box

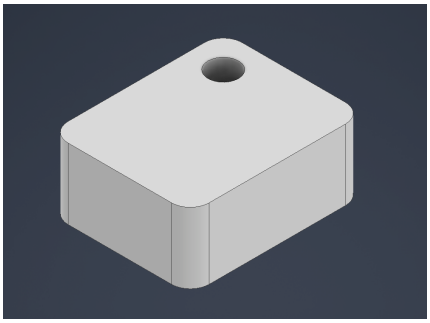

Figure 13: Maintaining piece of the IMU

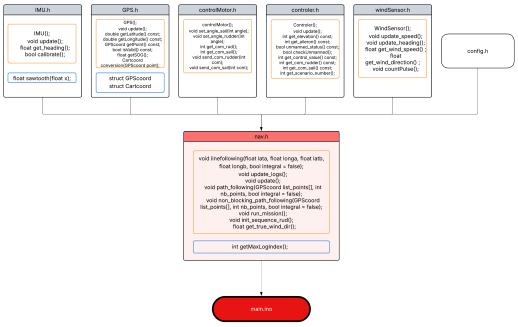

Figure 14: Code's architecture

with accurate measurements of the speed over ground and course over ground. Considering tests are made on an artificial lake where there are no currents, the course over ground can be approximated as equal to the heading of the boat. An approximation of the speed over ground could have been manually computed using the successive GPS positions of the boat, but it has not been made during this project as the boat seemed to work well without it, and the problem has therefore been discovered too late during the project.

2.3 Linefollowing controler

The main goal is for the boat to be able to join and follow a line between two points. This is mostly done thanks to the *linefollowing* method from the *nav* class. Using the algorithm given in [2], the method computes the command to send to the boat's servomotor depending on its current state and the line to follow. It is therefore possible to have the boat follow a line by repetitively updating its sensors and calling the *linefollowing* method.

After updating all the sensors and making sure the boat is in autonomous mode, the method first defines some useful variables :

- The cutoff distance r, which corresponds to the maximum distance the boat can travel perpendicularly to the followed line when tacking. This means the boat should not get to a distance higher than $\frac{r}{2}$ when tacking.
- The incidence angle γ, which corresponds to the desired difference between the heading

of the boat and the direction of the followed line when the boat is far from it.

- The clause-hauled angle ϕ, which is the minimal angle between the direction of the wind and the heading of the boat so that the boat can move forward.
- *angle_ruddermax* which is the maximal angle the rudder can reach each side of the boat.

The method then computes the error e using the determinant between the vector linking the start of the line to the boat and the direction vector of the followed line. The considered error therefore is the signed distance to the line of the boat. Then, if this distance is superior to half the cutoff distance, the program sets the sign of the tacking variable q, which defines which way the boat should be going when tacking, to the sign of the error. The method then computes the angle *angle_nominal* that the boat should follow to reach and follow the line, using the following formula:

$$angle\_nominal = angle\_target - 2 \cdot \gamma \cdot \frac{\arctan(\frac{e}{r})}{\pi} \tag{3}$$

where angle_target is the direction of the line. If both the difference between this angle and the wind direction is greater than the clause-hauled angle, and if the difference between the line direction and the wind direction if greater than the clause-hauled angle or the distance of the boat to the line is less than the cutoff distance, then the boat will not be tacking and therefore set its aimed angle to *angle_nominal*. Otherwise the boat will be tacking and moving in a clause-hauled configuration by computing its aimed angle using the following formula :

$$aimed\_angle = \pi + \Psi - q \cdot \phi \tag{4}$$

where Ψ is the true wind direction. The tacking condition given previously is mathematically expressed as :

$$\cos(\Psi - angle\_nominal) + \cos(\phi) < 0 \text{ or } (\cos(\Psi - line\_direction) + \cos(\phi) < 0 \text{ and } abs(e) < r) \tag{5}$$

Finally, the method sets the rudder position and the sail position using the following formulas :

$$angle\_rudder = angle\_ruddermax \cdot \sin(heading - aimed\_angle) \tag{6}$$

$$angle\_sail = 90 \cdot \frac{\cos(\Psi - aimed\_angle) + 1}{2} \tag{7}$$

Both angles are given in degrees. If the aimed angle is behind the boat, i.e. if $\cos(\theta - aimed\_angle) < 0$, the angle of the rudder is set to $\pm angle\_ruddermax$.

2.4 Adding an integral correction to the linefollowing algorithm

The paper cited in [2] also suggests to implement an integral correction in the linefollowing algorithm in order to avoid constant error. Such correction has been implemented in the algorithm used during the internship and can be optionally turned on or off using the optional boolean argument *integral* of the method.

This correction is made thanks to a variable z. At each iteration of the *linefollowing* method, if the integral correction is enabled, the value of z is set to $z + \alpha \cdot dt \cdot e$, where α is an arbitrarily chosen value, preferably less than 1, and dt is the time between each execution given of the method given in seconds.. The nominal angle is then computed using the formula :

$$angle\_nominal = angle\_target - 2 \cdot \gamma \cdot \frac{\arctan(\frac{e+z}{r})}{\pi} \tag{8}$$

The correction variable z is set to zero when the boat is tacking. If the boat has to follow several consecutive lines, it must also be set to zero between each line. Finally, z is also set to 0 if the distance between the boat and the followed line exceed a certain value, here chosen as 50 meters as suggested in [2], in order to avoid overcorrection.

2.5 Path following

Using the previously described *linefollowing* method, it is easy to have the boat follow a path made of several points by having him follow the lines linking to consecutive points. Such a method has been programmed in the *non_blocking_path_following* and *path_following* methods.

These methods take a list of GPS coordinates and the number of elements in this list as arguments, as well as the same optional boolean argument used to enable or disable integral correction in the *linefollowing* method. When called, for each pair of consecutive GPS points in the list, the program will call the *linefollowing* method between these two points, with or without integral correction enabled depending on the value of the boolean argument, until the boat goes past the second point. It also set the value of the z variable described previously to 0. The "went past" condition is computed simply by first computing the Cartesian coordinates of the two points and the boat in a local frame. If the points corresponding to the first point, the second point and the position of the boat are respectively noted A, B and M, then we can check whether the boat went past the second point by simply checking the sign of the scalar product between BA and BM : it should be negative if the boat went past the second point and positive otherwise.

We could also have decided to switch lines once the boat reached a certain distance to the end of the first line. However, if the boat for some reason had trouble following the line closely, it could go past the point at the end of the line without getting close enough to it to start following the next line, and would therefore keep going forward indefinitely. On the contrary, the method chosen here ensures the boat will be able to change the line it is following even if it has trouble staying close to the line it is following.

While the *path_following* method only stops once the boat went past all the points, the *non_blocking_path_following* stops if the boat switches to manual mode, which allows to easily change the followed path using the remote during tests.

2.6 Mission's unfolding

The main program only creates an instance of the *nav* class and then calls the *run_mission* method. This method sends commands to the servomotors depending on the controller's input while the boat is in manual mode, and then executes the *non_blocking_path_following* with a path depending on the controller's input while the boat is in autonomous mode. This allows to easily test the boat on different missions without having to access the Arduino board between each mission.

3 Tests results

3.1 Introduction

Several tests have been conducted during the internship to help identify mistakes and errors in the code and check the boat's performance. In this report, only the results of the latest tests -labeled number 51 in the list of log files- will be studied, as they correspond to the latest version of the boat's code. You might notice that there is also however a log file labelled number 52. This log file corresponds to a test that failed, probably due to a problem with the GPS.

Figure 15: The boat on Bournville's lake

The tests presented here have been made at *Bournville Lake, England*[4], which is a small and shallow artificial lake, which allows us to make sure that the boat will not be lost even if a major problem occurs during the tests. Several trajectories for the boat to follow have been implemented in its code prior to the tests. Those trajectories are, in order :

- A long line along the East-West direction.
- A triangular loop trajectory in the widest part of the lake
- A shorter line along the North-South direction
- A loop that goes around the lake

These trajectories aim to test the performance of the boat in various situations, by mixing long and short distances to cover along different directions and with various turns to make.

Each loop starts at the southernmost point and runs clockwise. Once the boat is set in autonomous mode with a loop to follow, it will follow it indefinitely until it is switched back to manual mode. For the trajectories corresponding to a single line, the loop consists in going back and forth along this line.

[4] `https://maps.app.goo.gl/28hqY4XkqS4Yib9R7`

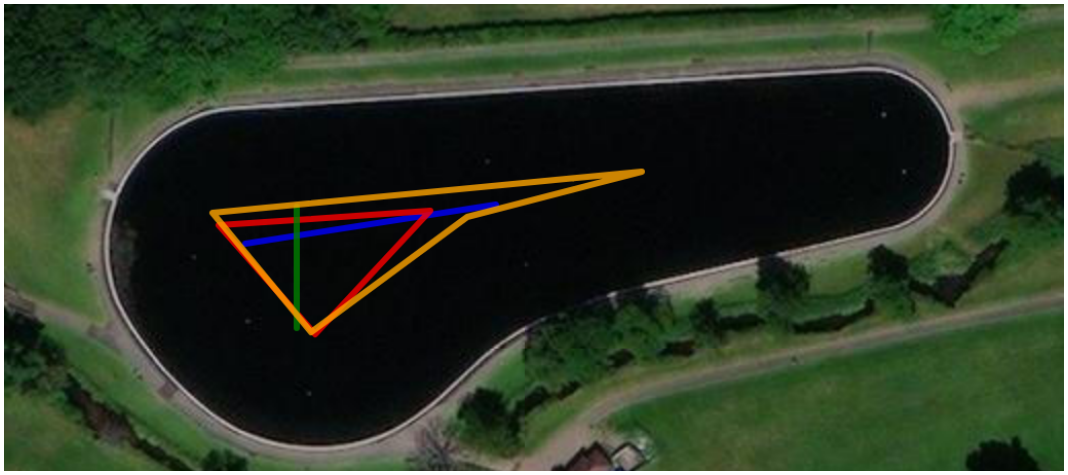

Figure 16: The four trajectories used to test the boat on Bournville's lake

This section aims to present the results of the latest test realized and to try and identify potential problems revealed by these tests and their causes. Videos of some of the tests are also available in appendices.

3.2 First observations

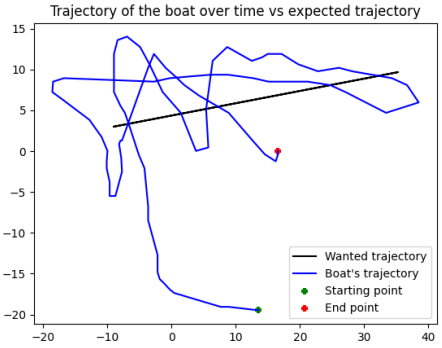

Figure 17: Trajectory followed by the boat when trying to follow path 1

Figure 17 shows the trajectory followed by the boat when trying to follow the first path. The true wind as calculated by the boat during the test was flowing toward the south-east, as shown by figure **??**, where angle 0° corresponds to the north and the angles increase counterclockwise. We can see that after getting closer to the line the boat started tacking as expected to go west and then turned around at the end of the line and sailed west without tacking.

We can also notice that the boat has trouble staying exactly on the line, especially when not tacking. This was also expected since according to [2], the algorithm without integral correction - which was not enabled here - can sometimes have a gap up to 10 meters with

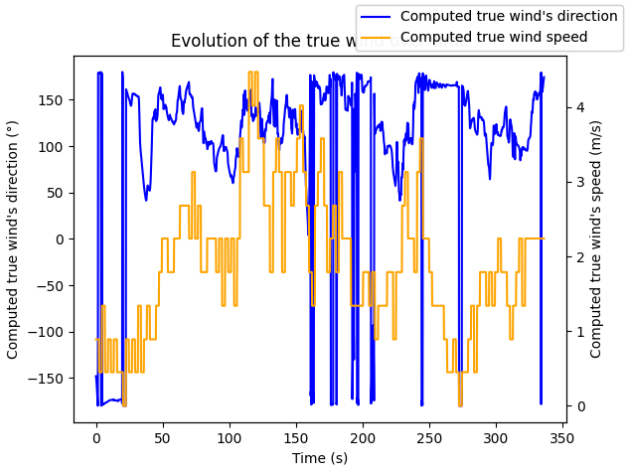

Figure 18: True wind data computed by the boat while following the line

the followed line. Still, this test, as well as tests on other paths as can be seen in the annex, show that the boat is overall able to follow a line or a succession of lines.

3.3 GPS issues

The GPS used on the boat quickly turned out to have some problems in taking frequent and valid measurements. As seen in Figures 19 and 20, the GPS only makes good measurements every few seconds, and sometimes no exploitable measurements are made for more than 30 seconds. Since the boat needs update about its position to change the heading it is following, this means the boat will sometimes go forward for a long time even if it causes it to move away from the line. This can cause a high inaccuracy in the boat's trajectory, especially if the wind speed is high, since this means that the boat will sail a longer distance before a new GPS position is available.

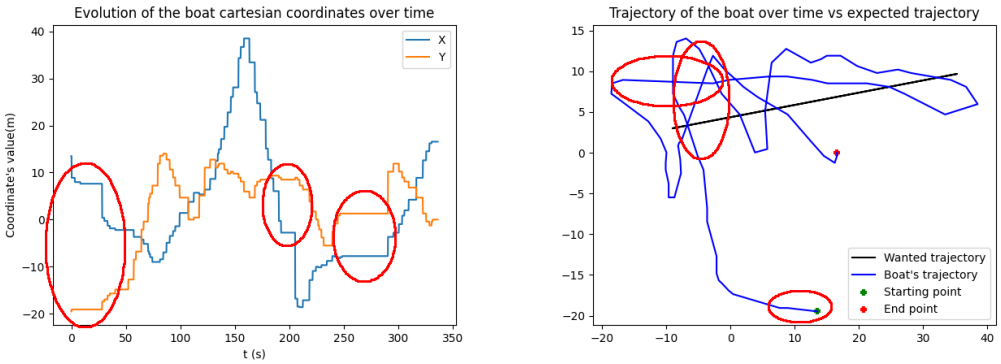

Figure 19: Highlight of some points where the GPS did not take good measurements on path 1

A potential solution to test to solve this problem without changing the GPS would be to

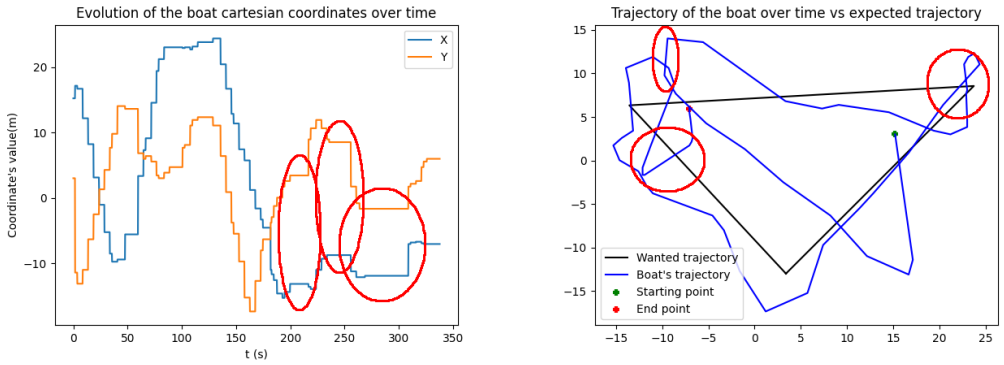

Figure 20: Highlight of some points where the GPS did not take good measurements on path 2

implement the sailboat's state equations in its code so that it can evaluate how its position evolved since it was last able to measure it.

Another problem detected too late to be fixed during the internship is that the GPS fails to measure the SOG properly. Because of that, the value of the SOG's value remained zero during the whole test. This problem is easy to fix since an approximation of the SOG can be easily computed manually. However, as shown in Figure 21, using the manually computed SOG to compute the true wind seems to have little impact on the wind's direction compared to computing it with a SOG equal to 0. This also explains why the problem has been detected so lately, since it had no visible effect on the behavior of the boat.

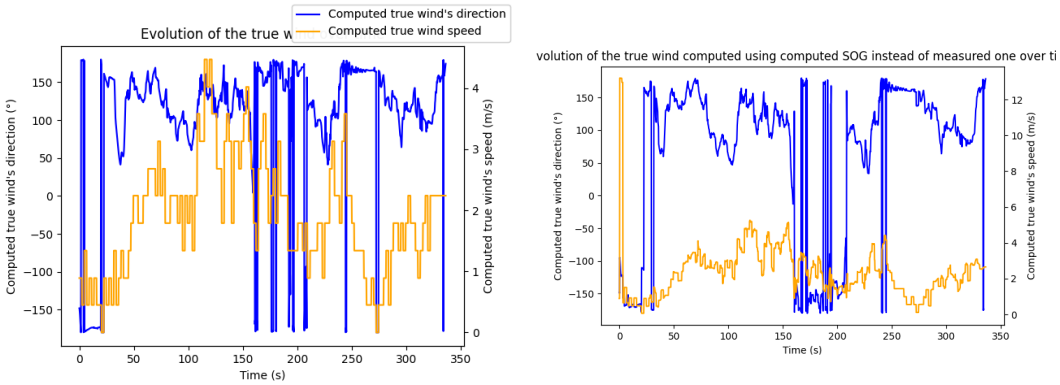

Figure 21: Comparison between the true wind computed with and without the SOG while following path 1

3.4 Anemometer responsiveness

The anemometer used on the boat is a mechanical one. This means that this sensors does not immediately perceive changes in the wind direction it is measuring, especially if the wind is low. Since the wind direction is initially measured in the boat's frame, this means that the anemometer takes some time to turn when the boat is turning, which creates a

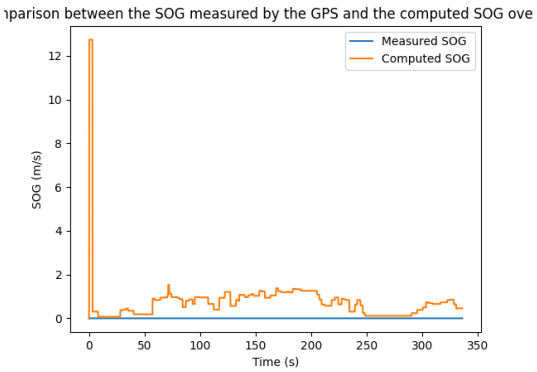

Figure 22: Evolution of the SOG calculated using GPS data over time

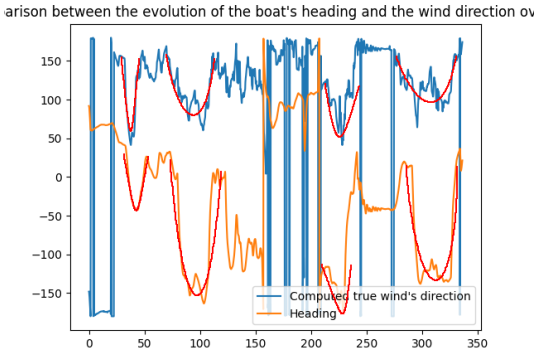

Figure 23: Comparison between the true wind's direction and the heading while following path 0

visible effect on the final calculated direction of the true wind as shown on Figure 23 This causes the boat to be a bit slower after some turns, since for some time after a turn it might calculate that the current wind does not require it to tack, or on the contrary that it does require it to tack, even if it is not true. It does not seem however that this effect causes any major issue on the boat's behavior, since the anemometer still ends up moving to the right position, and therefore it is likely not required to try to fix this problem.

3.5 Rudder command saturation

It has been observed during tests that the rudder sometimes rapidly switched from left to right. Looking at the log data, such as the ones displayed on Figure **??** show that the rudder command indeed saturated and rapidly switched sometimes. Even though it didn't seem to have a major impact on the boat's behavior during tests, it might have just been luck.

It appears that between time 240 and 270 seconds, the rudder's commands rapidly switch between approximately 280 and 395ms. The rudder's PWM command values are all contained between 200 and 430 milliseconds as set in the *config.h* file, so the rudder's command isn't switching between its extreme values, meaning the switch must be caused by strong

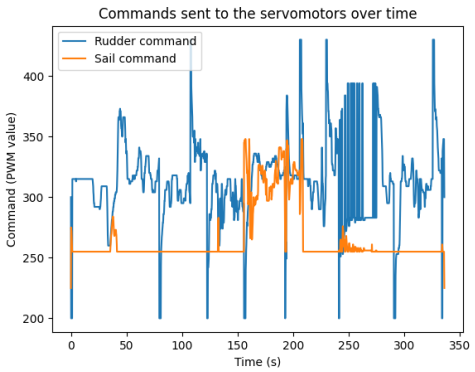

Figure 24: Evolution of the servomotor's commands while following path 0

and rapid variations in the value of $heading - aimed\_angle$ (cf equation 6). Since the heading does not show any such spike at the same time except negligible ones likely caused by the rudder's movement, the problem probably comes from the value of $aimed\_angle$. This means the value of $aimed\_angle$ is likely switching between values separated enough to create these changes in the values of the rudder's commands. This might be due to the boat being quickly changing between tacking and non tacking mode. The problem might therefore be fixed by implementing some kind of hysteresis on the tacking condition so that the boat does not switch back and forth from tacking mode. This might however damage the boat's performance by maintaining it in tacking mode when it could move straight and this solution therefore has to be tested to determine whether it effectively improves overall performance.

3.6 Turning against the wind

It has been observed during tests that the boat sometimes struggled to turn to reach a direction opposite to the wind position. This especially happened when the boat had a low speed during the turn while the wind was strong. In this case, the boat stayed stuck trying to turn against the wind while the wind was pushing it the other way. This did not prevent the boat from completing its mission since the wind eventually changed but this still severely damaged the boat's performance since it was sometimes unable to move for several minutes. A possible solution to this problem, though maybe only partial, might be to have the algorithm change the tacking variable's sign if the boat is stuck, for example if its GPS position stays approximately the same. This solution, however, would be hard to implement considering the current GPS as seen in section 3.3.

Conclusion

During this internship, I have been able to program the boat so that it would follow a given trajectory. Though some issues remain, they have been identified, solutions to these have been suggested and they did not prevent the boat from performing its task during tests. Further improvement might include implementing and testing the solutions suggested in this report to fix the identified issues, and further improving the boat's performance by, for example, implementing some filters on the data collected by the various sensors of the

boat. Finally, this internship helped me strengthen my skills in robotics, particularly in programming, and provided valuable insights into the key challenges to keep in mind when developing and testing a robot. It also allowed me to explore different approaches to working on this type of task. Moreover, I got an overview of how a laboratory operates, especially about the high level of autonomy granted there and the self-discipline required to work effectively in such an environment.

References

[1] Blue Water Sailing. Calculating the true wind and why it matters. https://www.bwsailing.com/cc/2017/05/calculating-the-true-wind-and-why-it-matters/, 2017.

[2] Luc Jaulin and Fabrice Le Bars. A simple controller for line following of sailboats. In *5th International Robotic Sailing Conference*, pages 107–119, Cardiff, Wales, United Kingdom, 2012. Springer. https://www.ensta-bretagne.fr/jaulin/paper_jaulin_irsc12.pdf.

[3] Titouan Leost. Aston autonomous sailboat 2024. https://github.com/TitouanLeost/Aston-Autonomous-Sailboat-2024, 2024.

[4] Titouan Leost. Development of a software architecture for an autonomous sailboat. https://webperso.ensta.fr/jaulin/rapport2024_leost.pdf, 2024.

Appendices

Link to the GitHub repository of the internship : `https://github.com/Ch0c3w/Internship_Ewen_MELE_Sailboat_2025_For_Report.git`

Some videos of the boat on Bournville's lake:

- `https://youtu.be/XxQTStJr2mw`
- `https://youtu.be/dt2sf4GPoYs`
- `https://youtu.be/Wl-aFyvWdZw`