Develop learning control of autonomous sailboats for oceanography

MUSELLEC Yann

UNIVERSITY OF
PLYMOUTH

Supervisor: Wan Jian
Tutor: Jaulin Luc
Address: Plymouth/Home

August 29, 2020

Contents

Acknowledgements

I express my deepest thanks to **Dr. Jian Wan** *Lecturer in Control Systems Engineering* at the University of Plymouth for the opportunity of internship, his availability and for the help all along the internship.

I would also like to thank **Mr Jaulin** for introducing me to this internship in Plymouth and presenting me to the Dr. Wan.

And finally thanks to my parents for all the support during this peculiar period. All the internship has been done from home due to the surge of the pandemic of Corona virus.

Abstract

In nowadays society, questions about energy reduction, time saving, having more accurate and more autonomous machines are becoming a hot topic. Thus, autonomous driving is more and more studied. Self driving vehicules allow to save time for other activities and be a helpful tool to increase the driver's security. Autonomous driving is present in a lot of field such as aerial exploration (with autonomous drones), for the railway (with autonomous train), for aerial transport (with autonomous taxi). This is a very exploited area for research. The progress in this field are very important since the computers are more and more sophisticated and effective.

Even in the maritime domain, machines and boats are created to make autonomous travels, namely boats with a motor such as container carrier.

It is in this environment that my internship took place, to see if we can create a sailing boat which could travel by itself and learn on its own how to do so.

This work is the continuation of other internships on the same sailboat.

Chapter 1

Introduction

1.1 Global health situation

The internship was a project of the University of Plymouth in England. There was theoretically a competition for autonomous sailing in which the university should have competed. However, due to the situation with the coronavirus, the event has been canceled. Moreover, with the several lockdowns, I could't go to the University of Plymouth, so I did all the internship from home. This situation was strange and difficult at the beginning, but when I received the components of the boat sent by the postal way : the real internship began.

Due to the situation, a weekly meeting has been planned with the Dr. Wan on the software Zoom. The goals of these meetings were to check my advancements and to solve if they were some difficulties.

1.2 The goal

This project was the continuation of a project made by students of the ENSTA Bretagne last year. They implemented algorithms for line following and station keeping. My mission was to use what they did, to improve what needed to be changed, test new things on the sensors, namely the calibration for some of them. Furthermore, to comment what I did in order to have an easy to understand file and taking charge of the project.

The documentation was essential for future people who will continue the project after me. They will have a better introduction on the project that I had. Therefore, all the code is opensource and free to use on my github [MUS 2020].

Then, I had to implement a machine learning algorithm and more specially a reinforcement learning algorithm to make the boat learn by himself to sail for a given mission.

1.3 The hardware architecture

The model of the sailboat used is the Ragazza Proboat (see the Figure 1.1).

Figure 1.1: Ragazza Proboat

Inside this boat:

- Sensors used are : an anenometer, a weather vane, an IMU and a GPS.
- Actuators are : 2 servomotors, one for the rudder and the other for the sail.
- Communication part was handle by a RCmodule.
- An Arduino Mega acquires the values of the sensors and a Raspberry PI 3b+ manages the control of the boat. For an easier data acquisition and servomotor control, two Arduino shields were used, Module Grove Base Shield and Adafruit 16-Channel 12-bit PWM/Servo Shield.

Since the access to the boat was not possible, I worked on the system below:

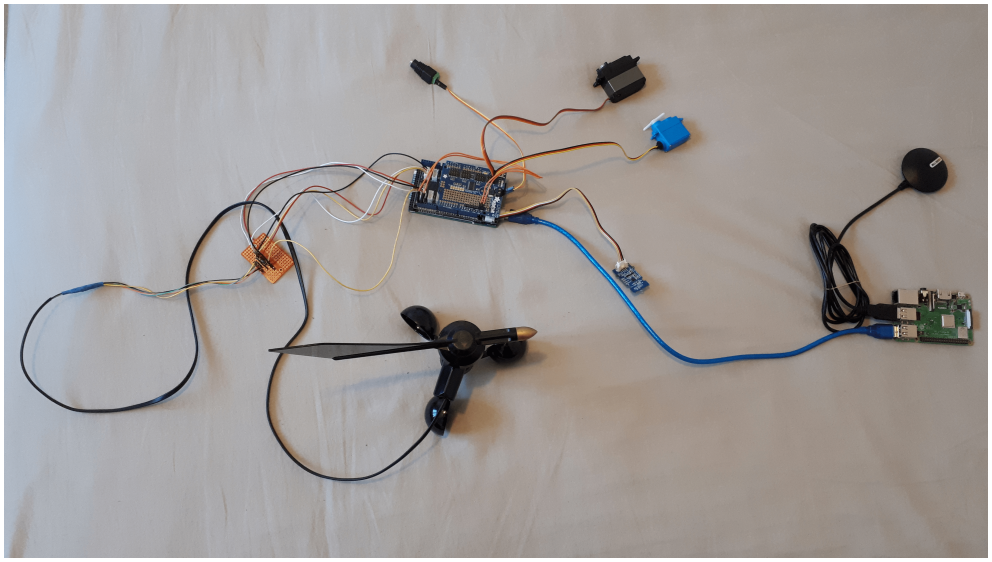

Figure 1.2: Physical system

As you may have seen on the previous picture, the Xbee module is not on the picture since the outward communication wasn't needed (it is also why we will not develop the communication part in this report). Between each component, there are different communications as you can see on the hardward architecture (Figure 1.3).

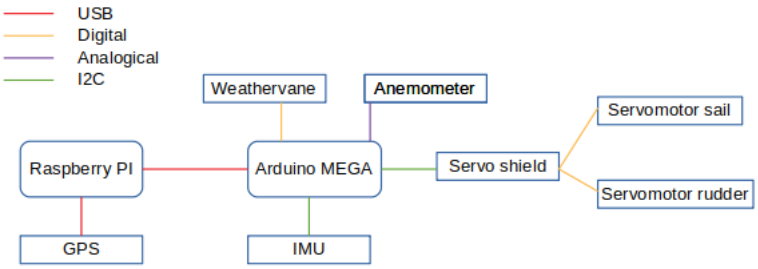

Figure 1.3: Hardware architecture

1.4 Software architecture

As you can see on the Figure 1.4, the low level composed of the sensors and the actuators is handled by the Arduino board in order to have the best possible frequency on the sensors. Then, the data are sent to the Raspberry PI for the filtering and controlling part. The Raspberry PI and the Arduino board 1.5 are connected via USB, the messages between the two boards are coded by ROS. The Arduino board has two shields plugged in, one shield 1.6 for an easy I2C communication and the other one 1.7 is there for the control of the servomotors. Finally, the Arduino board is an Arduino MEGA to have a better dynamic memory for ROS.

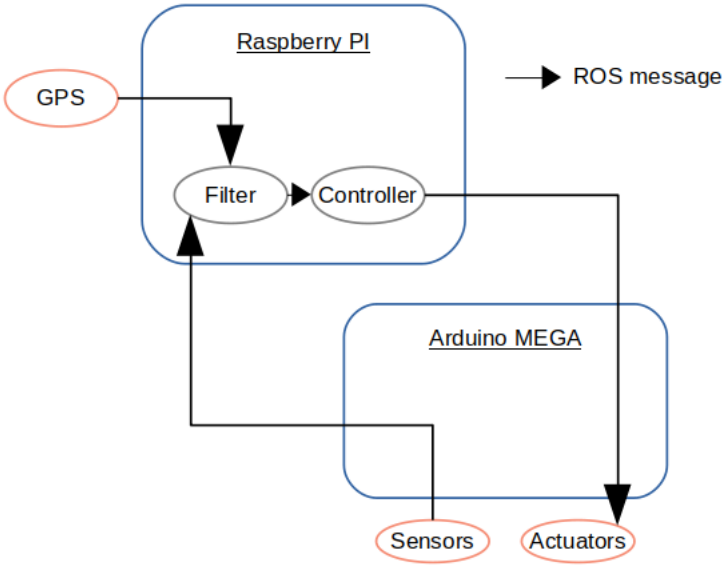

Figure 1.4: Software architecture

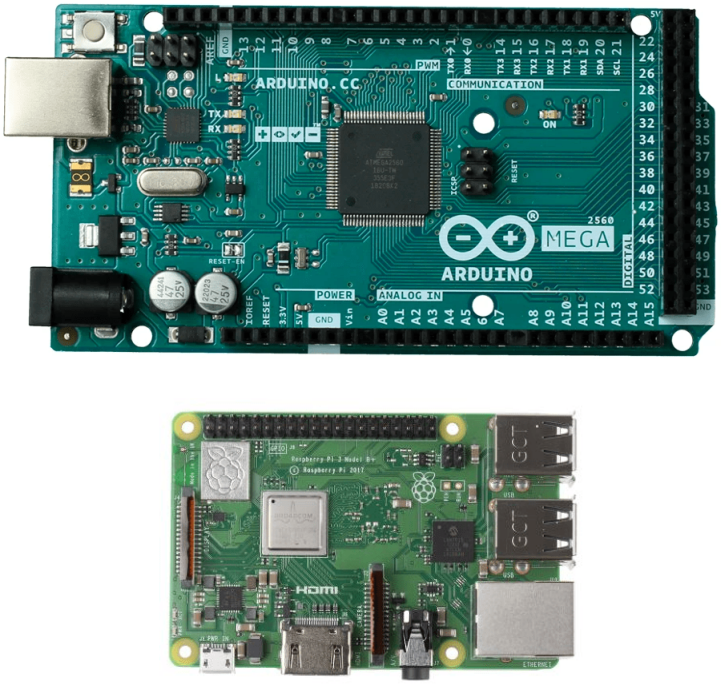

Figure 1.5: Arduino MEGA board on the top and Raspberry PI below

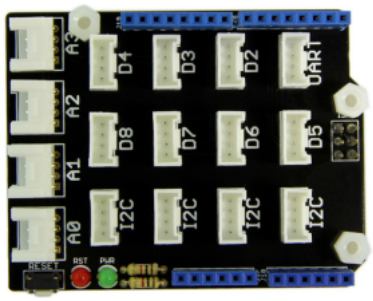

Figure 1.6: Module grove base shield

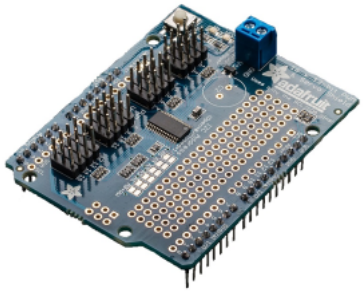

Figure 1.7: Adafruit servo shield

Chapter 2

Components

This chapter is the presentation of the components (sensors and actuators) and how they were used. Then, is the calibration part for the components that needed to be corrected. Finally, there is the filtering part of the data with the Kalman filter.

2.1 The sensors

The IMU

The IMU used is the IMU 9DOF v2.2 (Figure 2.1). The information given are the angular velocity, linear acceleration and the magnetic field. The euler angle of the boat can be estimated with the filtering part that is explain later.

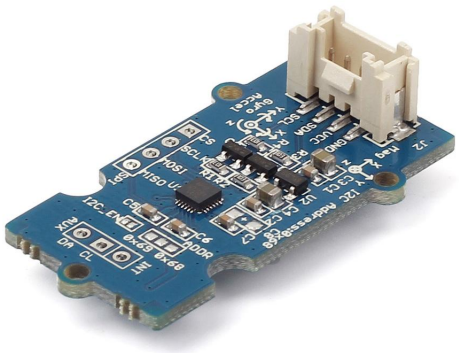

Figure 2.1: Picture of the IMU 9DOF v2.2

Type of message

The ROS message used from the Arduino to the Raspberry PI is the ROS sensor_msgs Imu message to communicate the values of the angular velocity. Also, the acceleration and the acquisition time are on a single topic. The value of the magnetometer is inside the ROS sensor_msgs MagneticField message.

The wind sensor

The wind sensor is composed of two sensors : a weathervane and an anemometer. Then, we have information about the wind direction and intensity. The wind sensor data were also filtered in order to define the good wind direction and a more consistant wind speed. The sensor used is the Standard Weathervane Anemometer – 7911 – Davis Instruments sensor (Figure 2.2).

Figure 2.2: the Standard Weathervane Anemometer – 7911 – Davis Instruments

Type of message

The wind sensors data are put in two ROS messages Float32.

The GPS

The GPS's model used is the BU-535-S4 (Figure 2.3). The GPS is directly connected to the Raspberry PI via USB. The values from this sensor are quite good and steady, thus no filtering action were needed.

Figure 2.3: GPS BU-535-S4

Type of message

The ROS GPSfix message from GPS_common is use for the communication part. This message takes into account all the data that can be recovered from the frames (here the GPGGA and GPRMC frames). The message type is able to take more information to improve the acquisition code.

2.2 The actuators

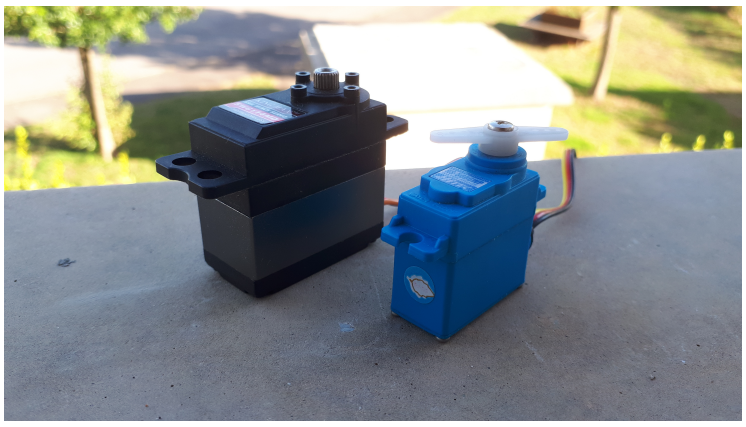

Figure 2.4: On the right the servomotor for

The actuators are two servomotors (Figure 2.4), one for the rudder and the second one is for the sail. The one for the rudder has a range of values from $-\frac{\Pi}{4}$ to $\frac{\Pi}{4}$, and is directly connected

to the rudder. The possible values for the angle of the sail sent by the controller are from 0 to $\frac{\Pi}{2}$, thanks to the wind we can have positive or negative values for the sail angle according to the wind direction.

The signals sent by the controller to the motors are in PWM (Pulse With Modulation). The value of the wanted angle is converted in PWM by the controller.

Type of message

The communication of the actuators' values are made by the ROS message Float32.

2.3 Calibration of the components

The components that needed some calibrations were, the IMU, the wind sensor (more precisely the wind direction part) and the servomotors.

The IMU calibration

In this calibration, there are 3 sensors to adjust : the accelerometer, the gyroscope and the magnetometer.

The accelerometer

The accelerometer detects a force (often called Inertial Force) that is directed in the opposite direction of the acceleration vector. Moreover, the acceleration value at the Earth's surface of the magnetic field is about 1g. An offset is often presents and will need to be estimated. Thus, to calibrate the sensor, a sample of acceleration values is taken when the IMU is not moving. The offset value is equal to the difference between the mean of the norm of the magnetic field and 1g.

The gyroscope

A gyroscope measures the rotation around the axis (here 3 axis). It gives a value that is linearly related to the rate of change of the angles around its axis. As for the calibration of the accelerometer, a sample of values is taken when there is no rotation. On each axis, there is an offset estimated by the mean of the values theoretically equal to zero.

The magnetometer

The calibration of the magnetometer is more difficult since it determines a heading relative to the Earth's magnetic North. This sensor is very sensible to every nearby magnetic field. When the magnetometer is correctly calibrated, rotating the sensor around 360° and plotting

the resulting data will create a sphere centered around (0,0,0). There are two types of distorsion possible :

- Hard iron distorsion : It is when something (a magnet for instance) creates a additive constant magnetic field to the Earth's field. This may cause an offset from location (0,0,0).
- Soft iron distorsion : This distorsion is not additive to the Earth's field. It is due to some materials (nickel and iron for instance) that disturbs the Earth's field. This distortion is not constant regardless of the orientation, soft iron' effects deform the sphere to produce an ellipsoid.

In order to calibrate the magnetometer, the values of the hard iron (the b value) and soft iron (the matrix A) distorsion must be found. The correct values of the magnetometer (h_{cal}) will be defined by the following equation (2.1).

$$h_{cal} = A^{-1} * (h - b) \tag{2.1}$$

To calculate A and b, the use of the software Magneto which uses the "Li's ellipsoid specific fitting algorithm" to find the best ellipsoid was needed. A sample of data from the magnetometer in every direction possible has been taken and was put as the input of the software.

After having the values for A and b the magnetometer was calibrated. There was no offset from the localisation (0,0,0) and the shape was a sphere as you can see on the figure 2.5.

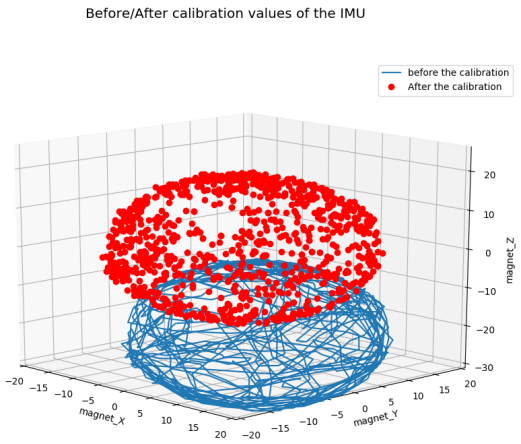

Figure 2.5: Before/After calibration values of the IMU

The wind sensor calibration

The calibration of the wind sensor was meant for the weathervane. We had the wind direction compared to the boat and not to the frame of the earth (this angle is called the true wind direction). To do so, the calibrated and filtered values from the IMU were needed (more specificaly the heading of the boat). Thanks to the right heading, the true wind direction and intensity were known.

The servomotors

The two servomotors didn't have the same specificities so the values found on one were not usable for the other one. The command from the Arduino to the actuator are PWM signals, so the extreme PWM values had to be found. For each motor, empirically this value was found by sending a value then the value+1 with a delay of few seconds and so on. Once the extreme values were found, functions were created given a wanted angle, with cross-multiplication. The function gives us the good PWM value.

Then the constrains were encoded, $\delta_r \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ and $\delta_s \in \left[0, \frac{\pi}{2}\right]$ where δ_r is the rudder angle and δ_s is the sail angle.

For the actuators, the values of the angles of the rudder and the sail are limited in order to protect the mechanical part of the boat.

2.4 Filtering the data

2.4.1 Finding the Euler angles

For the sail of the boat, we wanted to have the Euler angles. There are the three angles of the boat with respect to a fixed coordinate system. These angles are the roll (around $\vec{x}$), the pitch (around $\overrightarrow{y}$) and the yall (around $\overrightarrow{z}$). You can see it on the figure 2.6.

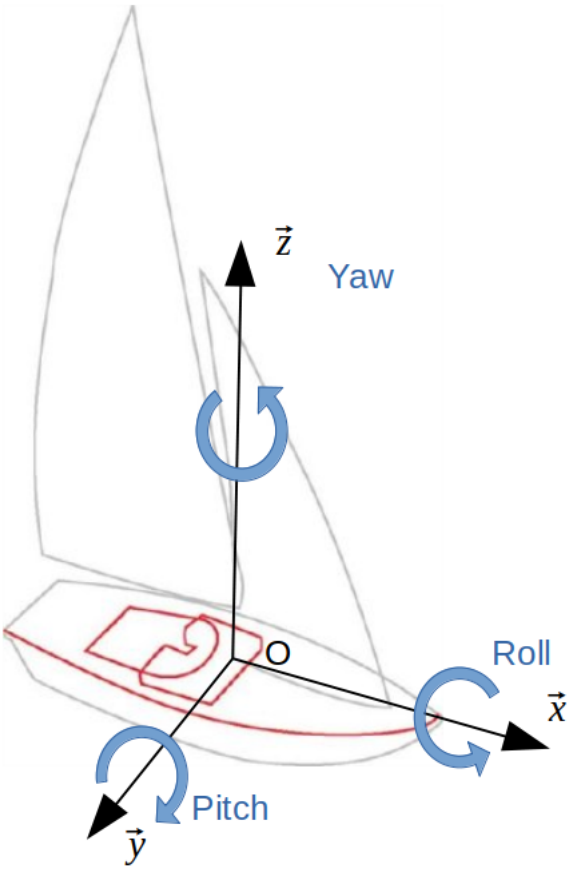

Figure 2.6: Euler angles on a sailboat

The three sensors in the IMU are able to give information about at least one angle:

- The accelerometer: It gives the acceleration vector pointing to the Earth's center when the sensor is not moving to fast. Thus it can give the values of the pitch (θ) and the roll (ψ). This is doable by projecting the gravity vector $\overrightarrow{g} = \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix}$ on plan xOz and yOz. The equations are :

$$\psi = arctan(\frac{g_y}{g_z}), \text{ and } \theta = arctan(\frac{g_x}{g_z})$$

- The gyroscope: It gives information about all the angles since the angular speed is given. The relation between the angular velocity and the angle is defined below:

$$angle = \int angular\_velocity * dt$$

- The magnetometer: It indicates the position of the magnetic North, by projecting this direction in the xOy plan, the yaw is known.

Two values for the angles are now known. One of these values may be false due to a drift of a sensor value through time or a big acceleration that need to be compensated. A fusion algorithm was needed to correct the errors, in this case a Kalman filter was applied.

The aim of a Kalman filter is to estimate an unknown value with a serie of measurements. It is used a linear problem and when the variables follows a Gaussian distribution. In this project, the unknown value is an angle following a Gaussian distribution however, an angle is none linear that is why an extended Kalman filter was implemented.

This algorithm is based on two equations:

- An evolution equation : $x_{k+1} = f(x_k, u_k)$, after the linearization $x_{k+1} = F_k * x_k + v_k + \alpha_k$ and $v_k = f(\hat{x}_k, u_k) - F_k * \hat{x}_k$ where F_k is the state transition model,u_k is the control vector and α_k is a white noise.

- A measurement equation : $y_k = h(x_k)$, after the linearization $z_k = H_k * x_k + \beta_k$ where H_k is the observation matrix and β_k is the observation white Gaussian noise.

Here, $F = \frac{\partial f}{\partial x}$ and $H = \frac{\partial h}{\partial x}$.

The filter is used as a fusion algorithm.

You can find more information in the ENSTA Bretagne's course on Kalman filter [JAU 19].

Here are all the expressions of the extended Kalman filter.

Prediction step:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$$

$$P_{k|k-1} = F_k * P_{k-1|k-1} * F_k^T + Q_k$$

Update step:

$$y_k = z_k - h(\hat{x}_{k|k-1})$$

$$S_k = H_k * P_{k|k-1} * H_k^T + R_k$$

$$K_k = P_{k|k-1} * H_k^T * S_k^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k * y_k$$

$$P_{k|k} = (I - K_k * H_k) * P_{k|k-1}$$

Next:

$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} cos(\psi) \\ sin(\psi) \end{pmatrix}$ Or, $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} cos(\psi) \\ sin(\psi) \\ cos(\psi)^2 + sin(\psi)^2 \end{pmatrix}$

The cosinus and sinus functions are linear so the extended Kalman filter works. At the beginning, a Kalman filter was already implemented with $x \in \mathbb{R}^3$ (see above) but after running

some tests of the $x \in \mathbb{R}^3$ and $x \in \mathbb{R}^2$ (that you can find here 2.7), the $x \in \mathbb{R}^2$ was found sufficient and the value of the angle didn't recquire to force the convergence of the angle.

The matrix f, h, F, H are set to:

$f(x,u) = \begin{pmatrix} x_1 - x_2 * dt * u \\ x_1 * dt * u + x_2 \end{pmatrix}$,$F(x,u) = \begin{pmatrix} 1 & -dt * u \\ dt * u & 1 \end{pmatrix}$ where dt is the sample period of the IMU.

$h(x,u) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $H(x,u) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

A mesurement value is defined as below every time new values are published.

$$z = \begin{pmatrix} cos(\psi_{raw}) \\ sin(\psi_{raw}) \end{pmatrix}$$

Finally, the value of the angle is $\psi = arctan(\frac{x_2}{x_1})$

The same process is applied to find ϕ and θ .

Once the value is determined, the ROS geometry_msgs Vector3 message is used to contain the value of the angles for future algorithms.

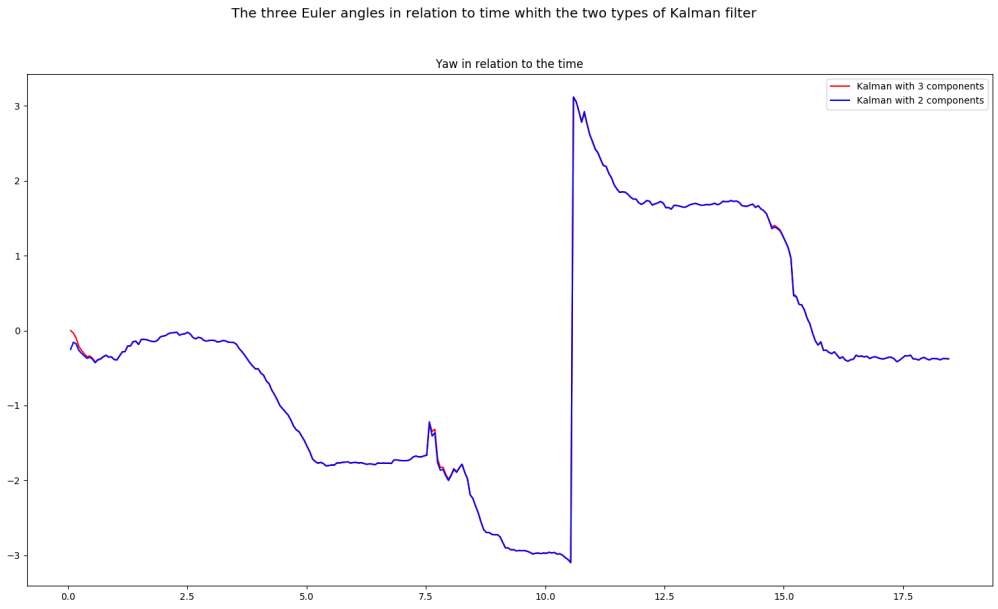

Figure 2.7: Comparation of extended Kalman Filter with $x \in \mathbb{R}^2 or \in \mathbb{R}^3$

2.4.2 Filtering the wind sensor values

For the wind sensor, the wind direction calculates an angle so the extended Kalman filter is once again applied. For the wind speed a low-pass filter has been chosen to avoid huge fluctuations. From the previous values of the wind's speed ($v_{previous}$) and the new one (v_{new}) a value is calculated with the formula 2.2 by putting more weight on one of the values.

$$v = \gamma * v_{previous} + (1 - \gamma) * v_{new} \tag{2.2}$$

γ is the gain which puts more weight on one of the values.

Chapter 3

Model

In order to test the behavior of the boat, the use of a model was compulsory because the physical access to the boat was not possible. The model used is from a paper made by Luc Jaulin [Jau 19].

$$
\begin{cases}
\dot{x} &= v\cos\theta + p_1 a\cos\psi \\
\dot{y} &= v\sin\theta + p_1 a\sin\psi \\
\dot{\theta} &= \omega \\
\dot{v} &= \frac{f_s\sin\delta_s - f_r\sin u_1 - p_2 v^2}{p_9} \\
\dot{\omega} &= \frac{f_s(p_6 - p_7\cos\delta_s) - p_8 f_r\cos u_1 - p_3\omega v}{p_{10}} \\
f_s &= p_4||W_{ap}||\sin\delta_s - \psi_{ap} \\
f_r &= p_5 v\sin u_1 \\
\sigma &= \cos\psi_{ap} + \cos u_2 \\
\delta_s &= \begin{cases} \pi + \psi_{ap} & \text{if } \sigma \leq 0 \\ -\text{sign}(\sin\psi_{ap})\cdot u_2 & \text{otherwise} \end{cases} \\
W_{ap} &= \begin{pmatrix} a\cos(\psi-\theta) - v \\ a\sin\psi - \theta \end{pmatrix} \\
\psi_{ap} &= \text{angle } W_{ap}
\end{cases}
$$

Figure 3.1: Sailboat model

Where, (x, y, θ) is the position and the heading of the boat, v is its forward speed, ω is the angular speed, f_s is the force's intensity of the wind's speed in the sail, f_r is the force's intensity of the water on the rudder, δ_s is the angle of the sail, a is the true wind speed, ψ is the true wind vector's direction and ω_{ap} is the apparent wind vector.

The values of the coefficients $(p_1, p_2, ..., p_{10})$ are design parameters of the sailboat. The values of these coefficients were taken from another similar boat of the University of Plymouth .

This state machine representation is a key part of the project since the physical model couldn't be use, then all the future comments about the algorithms are only based on this representation.

3.1 Vizualisation

In the first place, to see how the behavior of the boat with the state machine works. The python's library matplotlib was used (as you can see on the Figure 3.2), the real values of the sensors are plotted on screen .

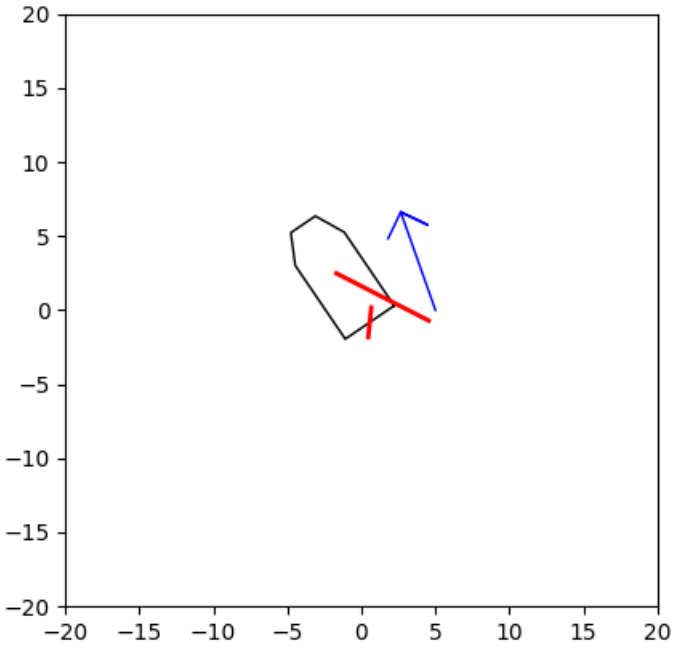

Figure 3.2: Real time values print on screen

With this library, the line following and station keeping were also tested. An exemple of a mission is on the Figure 3.3, the mission is to follow three lines.

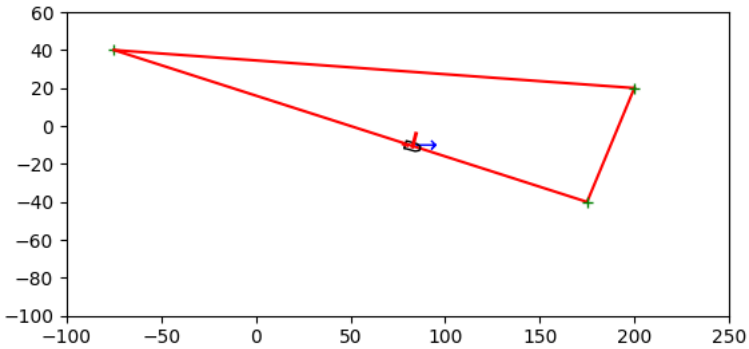

Figure 3.3: Vizualisation of three line followings

Then, the use of a ROS tool was more appropriate and faster to implement, this tool is Rviz. It allows to have a 3D representation of the boat (see the Figure 3.4). The visual aspect allows to see how the boat react to his environment, we can focus and zoom on a special part of the boat. All these operations are a lot easier to use since it is directly linked with the ROS environment.

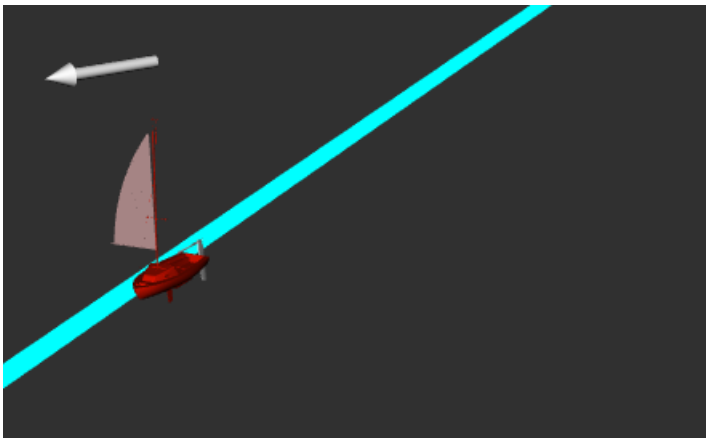

Figure 3.4: Vizualisation using Rviz

Chapter 4

Control algorithm

The control of a sailboat is quite compliquated because the behavior of the sailboat on the sea can change really fast and they are some areas of the boat position where the boat can't move (the no-sail-zone, to see the Figure 4.1). In order to have a good control for the boat, two main algorithms were implemented using a central algorithm.

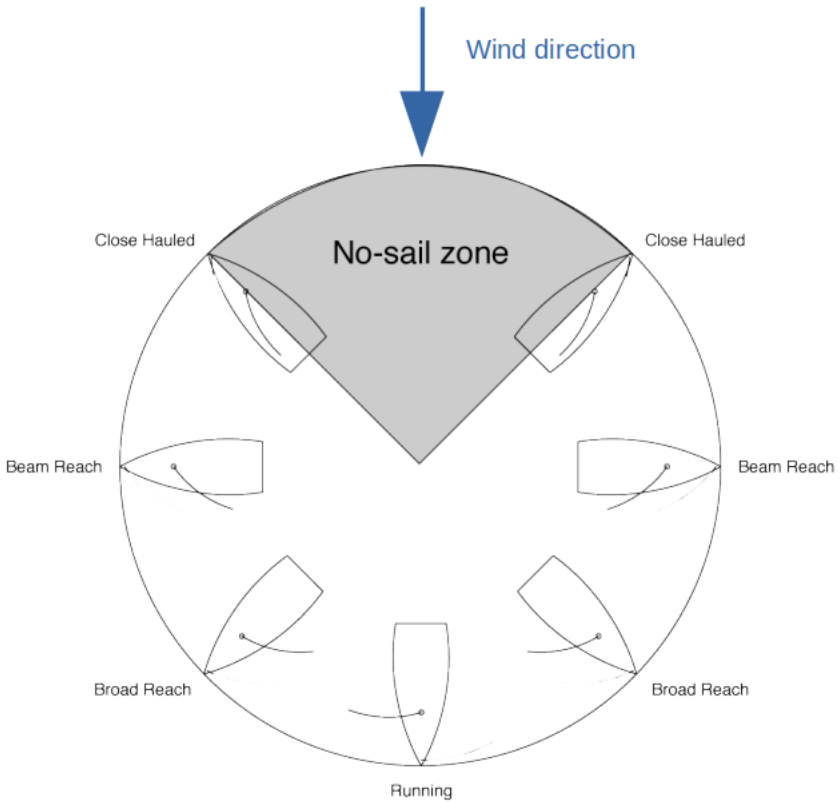

Figure 4.1: Sailing zones

In this chapter, the wanted heading is represented by $\bar{\theta}$, the real heading is θ, the wind direction is ψ, α are constants and u is the command.

4.1 Heading following

The control of the heading is a proportional command for the rudder and the sail. This algorithm will be use for all the future higher level algorithms. The commands sent to the servomotors are :

$$u_{rudder} = \alpha_{rudder} * arctan(tan(0.5 * (\theta - \bar{\theta})))$$

$$u_{sail} = \alpha_{sail} * \frac{\pi}{4} * (cos(\psi - \bar{\theta}) + 1)$$

The tangent and cosine function are here to avoid any non linear responses. The gains α_{rudder} and α_{sail} are here to have a faster response if needed.

4.2 Line Following

The line following algorithm is taken from Luc Jaulin and Fabrice Le Bars's paper on sailboat [JLB 12]. The algorithm is the following4.2 :

Function in: $\mathbf{m}, \theta, \psi, \mathbf{a}, \mathbf{b}$; out: $\delta_r, \delta_s^{\max}$; inout: q

$e = \det\left(\frac{\mathbf{b}-\mathbf{a}}{\|\mathbf{b}-\mathbf{a}\|}, \mathbf{m}-\mathbf{a}\right)$
if $|e| > \frac{r}{2}$ then $q = \text{sign}(e)$
$\varphi = \text{atan2}(\mathbf{b}-\mathbf{a})$
$\theta^* = \varphi - \frac{2.\gamma_\infty}{\pi}.\text{atan}\left(\frac{e}{r}\right)$
if $\cos(\psi - \theta^*) + \cos\zeta < 0$
  or ($|e| < r$ and $(\cos(\psi - \varphi) + \cos\zeta < 0)$)
    then $\bar{\theta} = \pi + \psi - q.\zeta$.
    else $\bar{\theta} = \theta^*$
end
if $\cos(\theta - \bar{\theta}) \geq 0$ then $\delta_r = \delta_r^{\max}.\sin(\theta - \bar{\theta})$
else $\delta_r = \delta_r^{\max}.\text{sign}(\sin(\theta - \bar{\theta}))$
$\delta_s^{\max} = \frac{\pi}{2}.\left(\frac{\cos(\psi - \bar{\theta}) + 1}{2}\right)$.

Figure 4.2: Line Following algorithm

Where m=(x,y) is the position of the boat, the rest of the input are already defined.

This algorithm is based on an attractive vector field to the line defined by the two points of the beginning.

4.3 Station Keeping

Different policies were possible and tested last year such as:

- Turn around : The boat was turning arround a fixed point (Figure 4.3). The main drawback of this option is that the boat can drift from its position without any correction.

Figure 4.3: Turn around strategy

- The 90 degree line : In order to avoid the no-sail-zone, the boat can follow a line perpendicular to the wind direction (Figure 4.4). Nevertheless, at the end of the line the boat as to do a turn around and may be in the no-sail-zone.

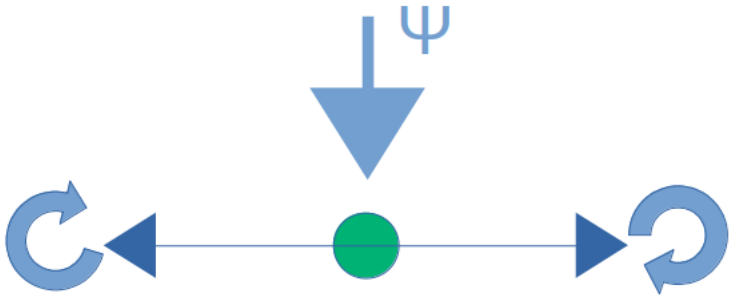

Figure 4.4: 90 degree line strategy

- The infinite symbol : In order to compensate the previous drawback, the infinite symbol that you can see on the Figure 4.5 allows to never be in the no-sail-zone by following lines and half circles in the good direction.

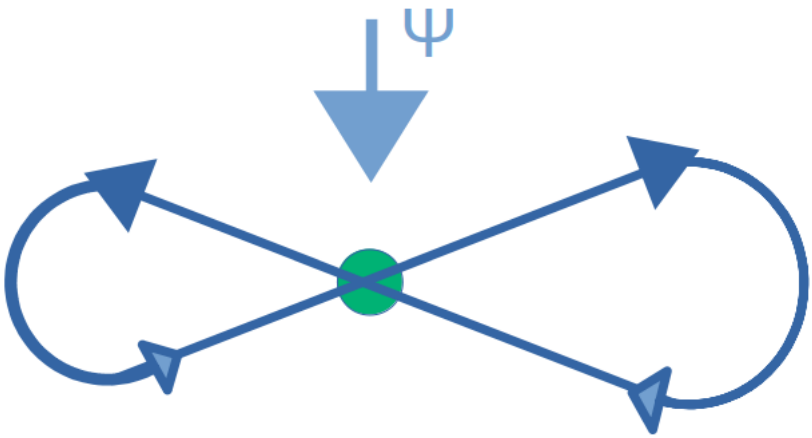

Figure 4.5: Infinite symbol strategy

Chapter 5

Machine learning part

The main algorithms for sailing have been implemented and tested. Now, the question arisen is : would it be possible for a boat to learn autonomous sailing and to be more efficient than the previous control algorithm? With machine learning, the sailboat could take into account more data for the sailing such as all the Euler angles and so on.

5.1 What is machine learning and why using it?

The machine learning is an Artificial Intelligence (AI) technology. It allows a computer to learn by itself with of without the help of a human to supervise a given task.

In this case, the task is to learn how to sail. However, there is no human to give information to the algorithm if its actions are correct or not. This domain of AI is called reinforcement learning.

Doing a supervised machine learning of the line following algorithm would have been easy with a neurone network. But the question of an autonomous program is how the sailboat learn and the result of it.

The aim of this type of algorithm can be described by the figure below 5.1:

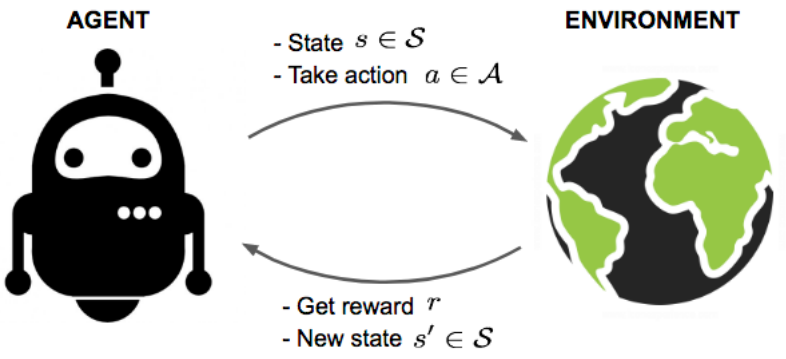

Figure 5.1: Reinforcement learning principle

The main goal for the agent is to maximize its future rewards in an unknown environment. All of this is handled by doing some trials and received feedbacks.

Basic reinforcement is modeled as a Markov decision process. There are states for the agent, actions possibles, probability of transition $p(s', r|s, a)$ (the probability to go from a state s' by taking the action a in the state s), reward function (the reward after the transition s to s' with action a).

At the beginning of the problem it is possible to have information about the model of the environment such as $p(s', r|s, a)$ if it is true, the problem is called model-based. The policy of the agent is the best action that can be chosen to maximize the future rewards.

The policy is defined as follow $\pi : S \rightarrow A$, it can also be probabilistic. The algorithm is called on-policy, if the policy taken for the evaluation and the improvement is the same one used during the learning phase. Off-policy algorithm are usually longer to be converged but more generalizable.

For more information about the reinforcement learning, you can find useful information here [Wen 18].

5.2 Definition of the problem

The algorithm will solve a station keeping problem. The boat will have to stay as close to a point for the longest time possible.

5.2.1 The chosen algorithm

The choice of the algorithm's type is very important. The more time spent on the choice of the algorithm, the less time will be taken to debbug the program. To chose the algorithm's type, the problem need to be well defined. According to previous definitions of the type of reinforcement learning, this is an off-policy and model-free problem. We don't know what the environment looks like.

There are different algorithms that correspond to these two criteria such as Q-learning, deep Q network, Deep deterministic policy gradient, and so on.

The Q-learning algorithm has been chosen.

5.2.2 Implementation of the algorithm

The Q-learning is an algorithm in which the action space and the state spaces are discrete. The Q-learning consists in creating a funcion $Q(s, a)$ where s is the state and a an action (s and a are the input data for a machine learning system, they are called features). The output value is the possible reward considering the state s and the action a.

Definition of the states and the actions

Since the actions and the states are discrete, the map was in the following Figure 5.2.

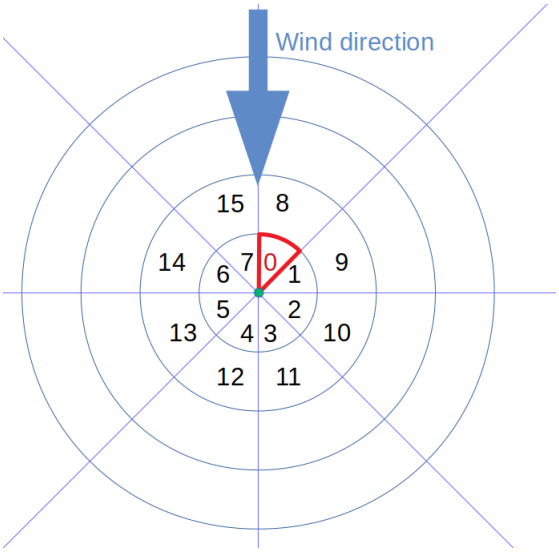

Figure 5.2: Discrete states of the map

At the beginning, the range of a ray, the number of ray and the number of angle parts are defined. In the previous drawing, the range of a ray is 1 the number of ray is 5 and the number of angle parts is 8.

There are also 8 different actions possible, these actions are headings to follow (Figure 5.3).

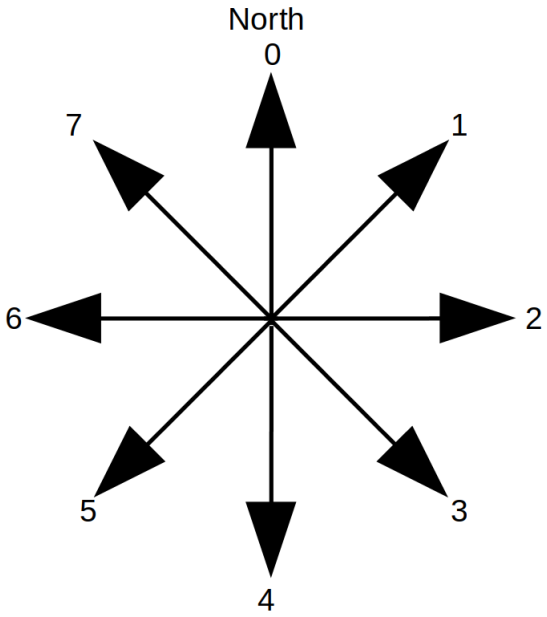

Figure 5.3: The 8 different actions

At the initialization of the algorithm, the number of trials is defined (100 iterations in this case). At each time an action need to be taken. The algorithm has to chose wether or not to explore and chose a random direction or to exploite the Q-table. In this case, the direction taken is the one with the highest reward.

Learning part

At a t time, the equation to update the Q-table is the following 5.4:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$

Figure 5.4: Equation to update the value of the Q-table

Where α is the learning rate or the learning speed, higher this value is, the more important the new value will be. γ is the discount factor, it determines the importance of the future rewards (if it's too close to 1, the solution may diverge).

5.2.2.1 The reward function

The reward function is composed of two elements. One, if the boat goes in the good direction. Two, if it goes there the fastest way possible.

- For the good direction :

    1. if the next ray is closer to the point to stay, then the reward is 1.
    2. If it goes in the opposite direction, the reward is -1.
    3. Finally, if the boat is on the same ray, and just the angle of the area as change, then the reward is 0.01. The system is explain below (Figure 5.5)

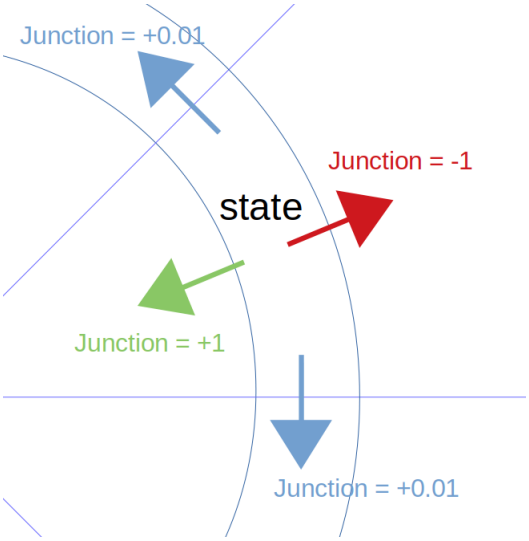

Figure 5.5: Reward function for the junction

- For the fastest way, the boat's speed polars were used.

The boat's speed polars are the representation of the boat's speed according to the wind intensity and speed. For each current boat's speed, there is a different boat's speed polar diagram. This information allows to estimate the sailing time and take the best decision concerning the security and the performance possible.

Since the boat's speed polars have almost the same shape for different speed, the use of a single boat's speed polar has been created (see the Figure 5.6).

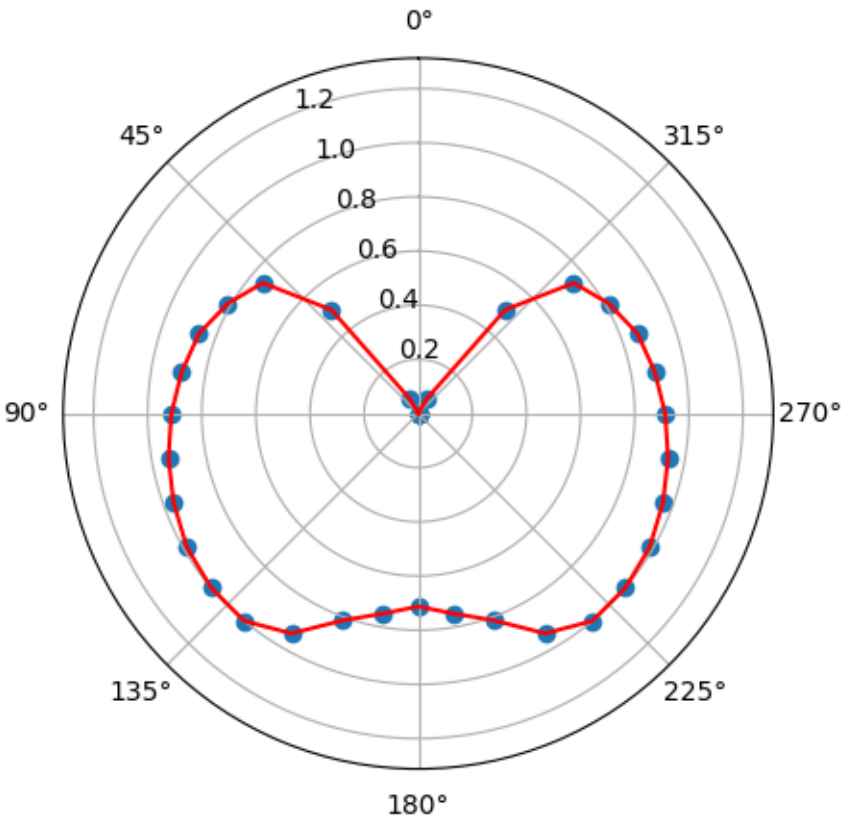

Figure 5.6: The boat's speed polars for the reward function

Then if the direction chosen by the boat is the one with the max speed, the reward will be 1 and the smallest one is 0.

Hence, the reward function is define as below:

$$reward = polar + junction$$

5.3 Results

After half of the iterations, the boat began to behave more and more as an exploiter of the Q-table and the behavior was quite stable. At the end, the sailboat understood that the best way to stay at a point was to be in the no-sail-zone at the point and to go in the good area when it is too far. We already know that this behavior creates a drift of the boat, but for the simulation, it is indeed the best way to stay close to the point. If the Q-table is learnt in the real environment, the behavior may be different. Thus, the conclusion of this part may be different with a training part on the sea. A second improvement for this algorithm will be to have a better reward function namely with the real boat's speed polars and not with a look like one.

Moreover, we may test another type of algorithm such as neurone network or a policy gradient in which the values of s and a can be continuous.

Chapter 6

Conclusion

6.1 What's next?

The next step will be to implement another type of reinforcement learning as the policy gradient since the actions and states may be continuous. To try it on a station keeping mission. Then, a line following algorithm will be needed.

Furthermore, the different machine learning algorithm (station keeping and line folowing) could be assembled in order to have a whole mission.

If these algorithms work, the addition of other features may be needed in order to have a more complex behavior.

6.2 Organization and knowledge

The global health condition during the internship didn't help in its realization. I began the internship fourteen days later and I couldn't go to the University of Plymouth. Nevertheless, with the help of my supervisor, I received every components and documentation needed to work from home. At first, a planning was made for all the tasks but after two weeks, I understood that I would need much more time to understand everything that has been done by the previous students. In these previous works, few algorithms were explained, and some didn't work.

Actions	Receiving all the components	Understanding the components	Installation and initialization of all the devices needed	Doing the low level launch with the filtering part	Implementation of line following algorithm	Implementation of station keeping algorithms	Creating launch for outdoor missions	Machine learning part
Forecasted time (in weeks)	2	2	3	5	1	1	2	5

Table 6.2: First version of the Gantt diagram

Moreover, this project is mainly developed by students. So for the future students whom will be working on it, an explanation of the algorithm was needed (at least a minimum). An organization and an explanation of the codes have been made. You can find all the codes on my github.

Then a second organization of the work as been followed, you can find it on the SADT diagram (Figure 6.1), the arrows with a text below are the skills that I needed to acquire:

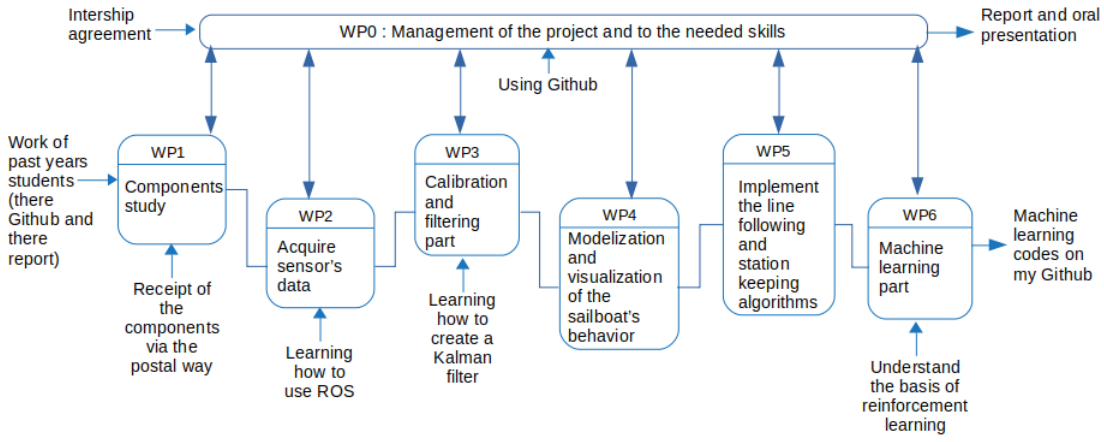

Figure 6.1: SADT diagram

On the diagram there is no concept of time because I needed to learn a lot of things and I had the advantage to schedule my timetable as I wanted during the week.

During this internship, I learnt a lot about robotics and its tools namely ROS, the serial communication bus, the Kalman filter and the use of github. I also improved some of my knowlegde about arduino, the C++ language, python, machine learning with the reinforcement learning, electronics with the low level, understanding and implementing complex algorithms. This was an interesting autonomous internship with new fields of competences and a real improvement of my knowledges in robotics.

Bibliography

[MUS 2020] Musellec Y. Github repository. url: https://github.com/ymusell/plymouth2020.

[JAU 19] Jaulin L. Mobile robotic, KalMOOC. 2019.

[Jau 19] Jaulin L. "Robmooc". In: (Mar. 2019). url: https : / / www . ensta - bretagne . fr / jaulin/robmooc.pdf.

[JLB 12] Luc Jaulin and Fabrice Le Bars, A simple controller for line following of sailboats,2012.

[Wen 18] Weng L.,https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html#key-concepts,Feb 19, 2018