Preliminary Design and Control of a Sailing Robot

Yoann Sola

Supervisor: Jian Wan

Tutor: Luc Jaulin

Address: University of Pymouth, School of Marine Science, England.

October 10, 2016

Abstract

English Version

Sailing robots or autonomous sailboats are among the latest developments of marine robotics. The report of this internship in the University of Plymouth is divided in two parts.

The first part deals with a feasibility study concerning the prototype of an autonomous sailboat. The architecture of this prototype and the calibration methods of all its sensors will be described.

The second part deals with the construction of two simulations about the control of a sailing robot : the first one in order to make it follow a path, and the second one in order to make it stay inside a zone and avoid obstacles.

Key Words : sailboat ; robotics ; calibration ; path following ; obstacle avoidance ; virtual anchoring.

French Version

Les robots voiliers ou voiliers autonomes sont parmi les derniers développements de la robotique marine. Le compte-rendu de ce stage à l'Université de Plymouth est divisé en deux parties.

La première partie traite d'une étude de faisabilité concernant le prototype d'un voilier autonome. L'architecture de ce prototype ainsi que les méthodes de calibration de tous ses capteurs seront décrits.

La seconde partie traite de la construction de deux simulations à propos du contrôle d'un robot voilier : la première afin de lui faire suivre un chemin, et la seconde afin de le faire rester dans une zone et éviter des obstacles.

Mots Clés : voilier ; robotique ; calibration ; suivi de chemin ; évitement obstacle ; ancrage virtuelle.

Contents

Chapter 1

Introduction

1.1 The Need of Autonomous Sailboat

Marine robotics : a world to explore

Compared to the latest technical and scientific advances in ground robotics (autonomous cars, humanoid robots, etc) and aerial robotics (autonomous plane, quadcopter, etc), the sector of marine robotics is yet to be explored further. Indeed, the two-thirds of the Earth is covered by oceans, seas and lakes, but the general public is mainly focused on space exploration or lands applications of robotics.

But since the beginning of the twenty first century, we have observed a huge development of marine robotics. Indeed, more and more universities and companies, associated with the field of marine sciences, are developing and using autonomous marine systems in various projects. In this way, Autonomous Underwater Vehicle (AUV) and Unmanned Surface Vehicle (USV) can be used alone or in flleet in various context : during scientific explorations, for the protection of the ocean resources, during environmental disaster, ...

Why autonomous sailboat are needed ?

One of the latest type of autonomous marine system to appear is the autonomous sailboat. Over the past few years, there have been growing interests in sailing robotics, with the apparition of major events such as the World Robotic Sailing Championship (WRSC) or the International Robotic Sailing Conference (IRSC).

Autonomous sailboats offer strong advantages such as its low-cost design or its ecological functioning, and can be used in a variety of activities : during ecological studies of oceans without electromagnetic interference, for supplying equipments in remote islands or regions, for generating energy thanks to its low consumption of electricity and the use of solar panels or windmills, ...

1.2 The University of Plymouth

Presentation of the University of Plymouth

In order to complete my second year of engineering school, I made a three months internship in the University of Plymouth, in England. The university was divided in different schools, each one of them operating in a different field of activity.

My internship took place in the School of Marine Science and Engineering, and my supervisor was Dr Jian WAN, lecturer in Control Systems Engineering and member of the Autonomous Marine Systems (AMS) research group.

The goal of my internship was doing a feasibility study concerning the sensors used by autonomous sailboats and trying to explore new control algorithms for obstacle avoidance, anchoring and path following..

The Mayflower Autonomous Research Ship

The University of Plymouth is currently involved in a pioneering project of sailing robotics : the Mayflower Autonomous Research Ship (MARS) project. This project is made in collaboration with two british companies, MSubs and Shuttleworth Design, and its goal is to build a big autonomous sailboat (over 100 feet in length) in order to entirely cross the Atlantic ocean, without any human assistance. The autonomous sailboat will carry on board several aerial drones in oder to make experimentations during the journey.

The crossing is planned to be done in 2020, in order to celebrate the 400^{th} anniversary of the original Mayflower sailings from Plymouth, England to Plymouth, Massachusetts, USA.

3D models of the future sailboat can be seen on Figure 1.1.

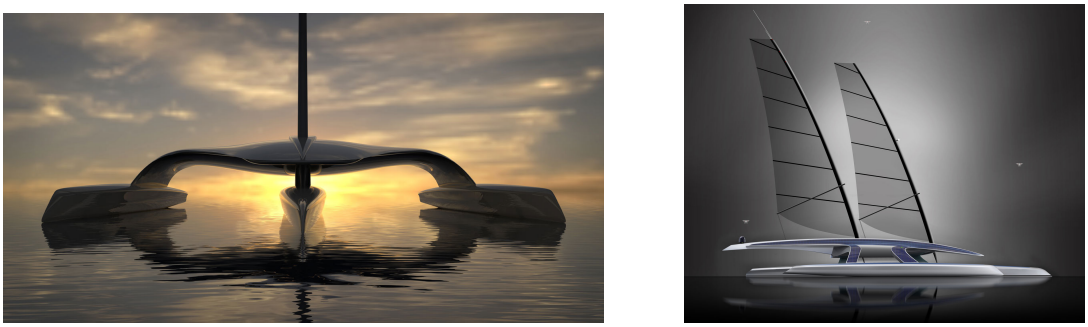


Figure 1.1: The Mayflower Autonomous Research Ship project

My internship was made in the context of the preparation of the MARS project, and I met some representatives of the company MSubs during my stay in Plymouth.

Chapter 2

Building a Prototype of a Sailboat Robot

The first part of my internship consisted in building a preliminary prototype of an autonomous sailboat. Once finished, the prototype had not to be fully operational, since the aim of my internship was doing a feasibility study. Indeed, the robot had to be cheap and all parts had to be reusable for future student projects, which means that solderings were not allowed (except in some cases such as the power supply coming from the Motor Shield or the I2C module of the LCD Screen) : the whole device could not be resistant enough to face the harsh environment of the ocean, so experimentations in water were not planned during this internship.

In order to test easily the calibration of sensors, I built a fully equiped buggy (or a ground robotic platform) with all sensors and electronic cards required for an autonomous sailboat. Indeed it is easier to test a small buggy than a sailboat, because we just have to put the buggy outdoor, wherever we want, whereas a sailboat has to be put in an aquatic environment, like in a pool for example.

In this section, I'm going to describe chronologically the construction of this prototype, part by part. Moreover, in order to help potential builders who would be interested in building this kind of robot, I will explain the knowledge I acquired at each step of the construction in detail, including the theoretical background, calibration methods, useful tools or even just practical advices.

The final version of the buggy can be seen on Figure 2.1, and its electronic architecture is shown on the Figure 2.2.

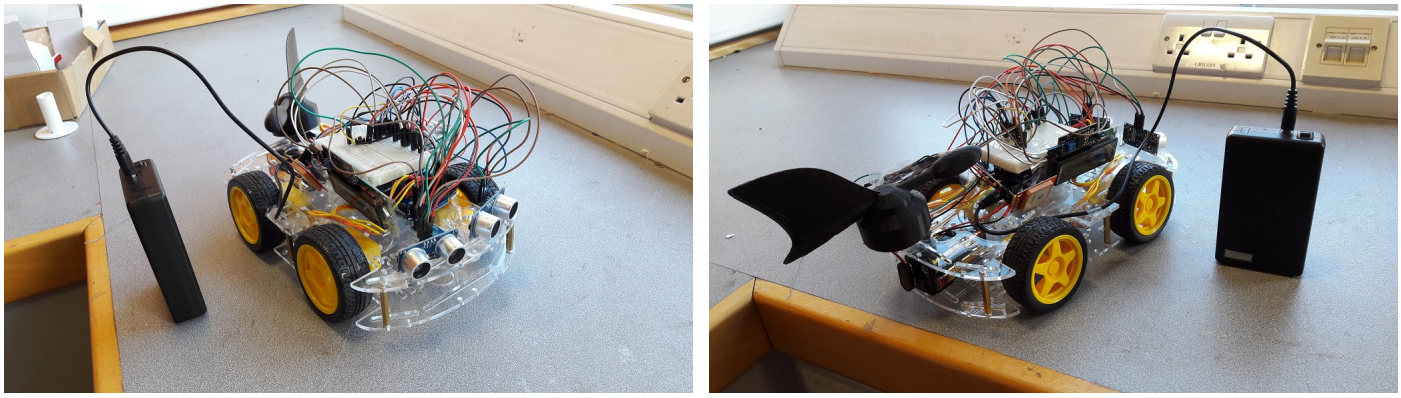


Figure 2.1: Final version of the buggy

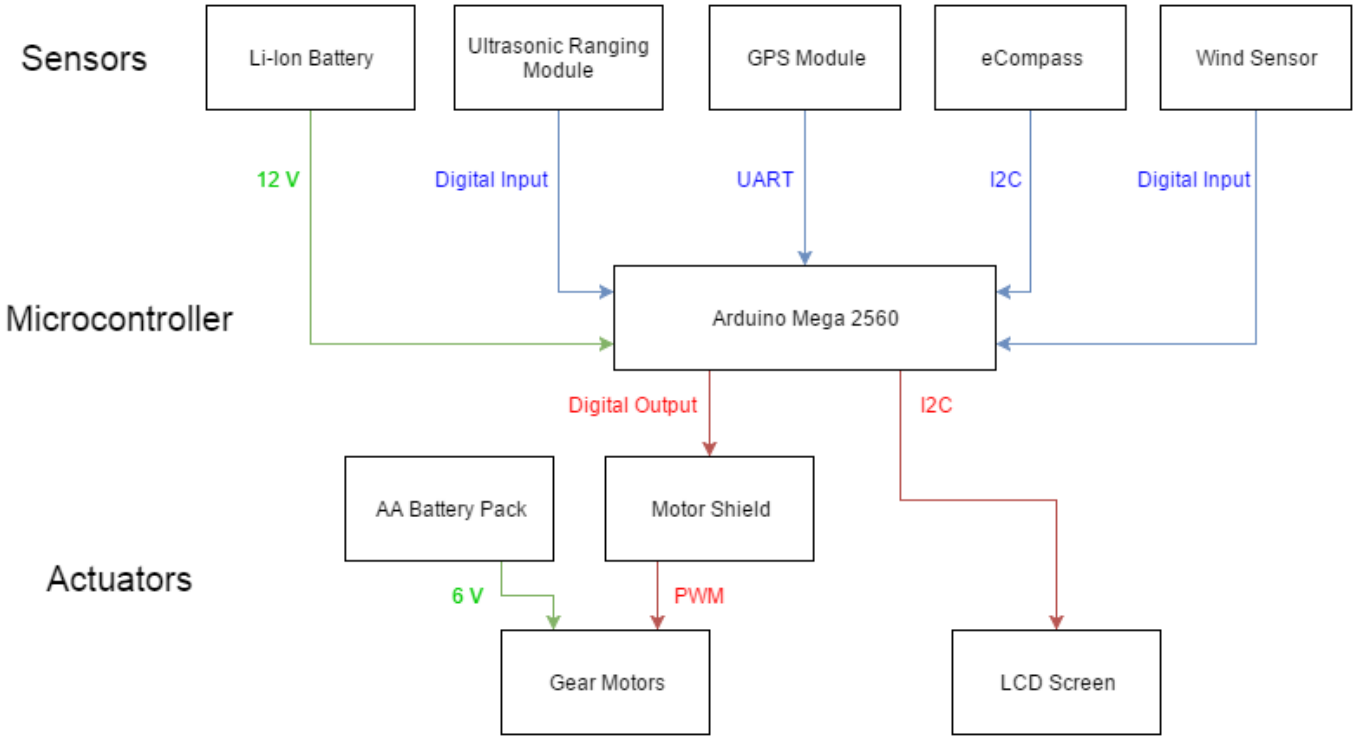


Figure 2.2: Electronic architecture of the buggy

2.1 The Body of the Buggy

Firstly, I'm going to talk about the body of the buggy, which includes the chassis, the microcontroller and motors.

2.1.1 The Chassis

The whole architecture of the buggy is based on a four-wheel drive chassis (a 4WD chassis or 4x4 chassis), which can be seen on the left part of the Figure 2.3. The chassis is made of acrylic material and is equiped with four gear motors, each one directly connected with a wheel, as shown on the right part of the Figure 2.3.

Gear motors have a power supply range from 3V to 12V, and a nominal voltage of 6V. Their speed and their direction of rotation can easily be controlled with a Pulse-Width Modulation signal (PWM).

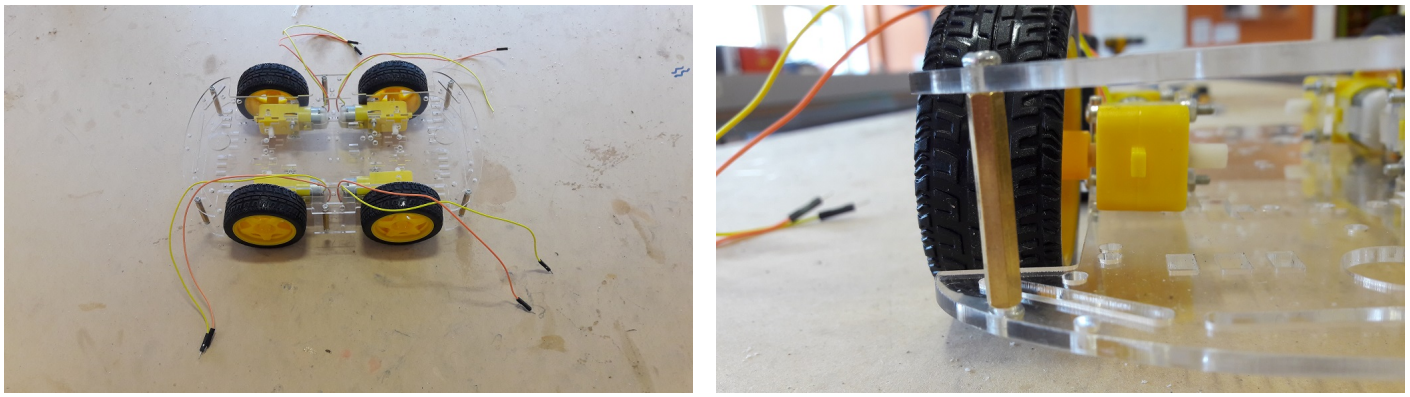


Figure 2.3: The chassis of the buggy

2.1.2 The Microcontroller

The microcontroller board which was choosen in order to read sensors, host programs and operate actuators is an Arduino Mega 2560 (see Figure 2.4, left part).

This PCB (Printed Circuit Board) is based on an ATmega2560 microcontroller and is designed for project involving a large numbers of sensors and actuators, since it includes 54 digital Input/Output Pins (of which 15 can deliver PWM signals, 6 can be used for Interrupt functions, 8 can allow serial communications through UARTs and 2 can allow I2C communications) and 16 analog Input pins. Digital output pins deliver an operating voltage of 5V, but the board needs a power supply that lies between 7V and 12V in order to work correctly.

Moreover, thanks to the open-source Integrated Development Environment (IDE) provided by Arduino (see Figure 2.4, right part), this microcontroller board is easy to program with the use of language C++ and various useful libraries. Arduino has also a huge community of “Makers” using these kind of boards in various open-source projects and developping new unofficial libraries. I will enumerate all libraries I used in each of the following sections of this report.

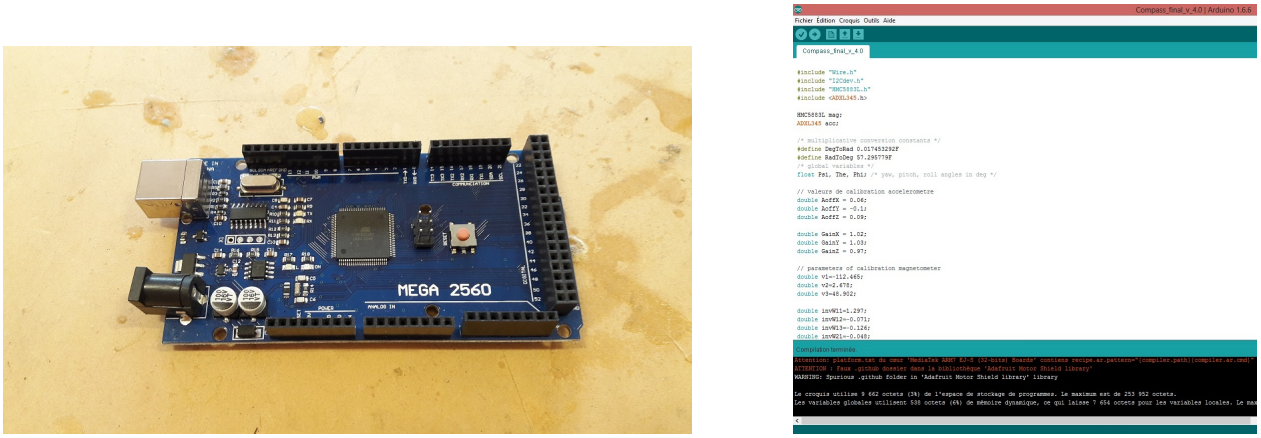


Figure 2.4: The Arduino Mega 2560 board (left) and Arduino IDE (right)

2.1.3 The Motor Shield

There are different ways of controling DC motors or gear motors : by using PWM pins of the Arduino Mega 2560, or with an H-bridge motor driver integrated circuit such as the L293D or even by using additional electronic boards. I chose the last option.

Indeed, I used a “clone version” of the Adafruit Motor Shield (see Figure 2.5, left part). In the vocabulary associated with Arduino boards, a “shield” means an additional card that can be directly connected on the main Arduino board, as shown on the right part of the Figure 2.5. This way of connecting boards one on top of the other allows to gain a lot of space, adds all new features to the main board and no soldering is needed (except power supply pins provided by the main Arduino card, in order to use them with other electronic components in the future).

The Adafruit Motor Shield is able to controll simultaneously several configurations of motors, depending on their type : 4 DC or gear motors + 2 servomotors, or 2 stepper motors + 2 servomotors, or even 2 DC or gear motors + 1 stepper motor + 2 servomotors. Moreover, the company Adafruit provides a library called *“AFMotor”* which adds all the functions needed for the use of this shield.

I used this Motor Shield in order to simultaneously controll the four gear motors of the previous chassis, which allows me to adjust their respective speed and to change their direction of rotation.

Finally, this Motor Shield can also be used in order to controll the future autonomous sailboat. Indeed, it is able to simultaneously controll the two servomotors used in that system : one in order to adjust the rudder and the other to regulate the length of the sheet (which will allow to adjust the sail).

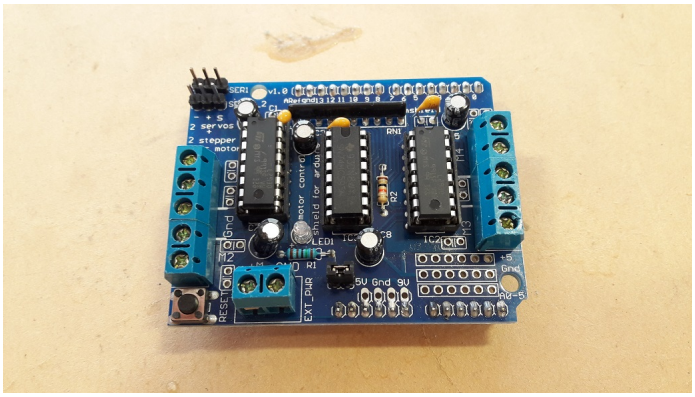
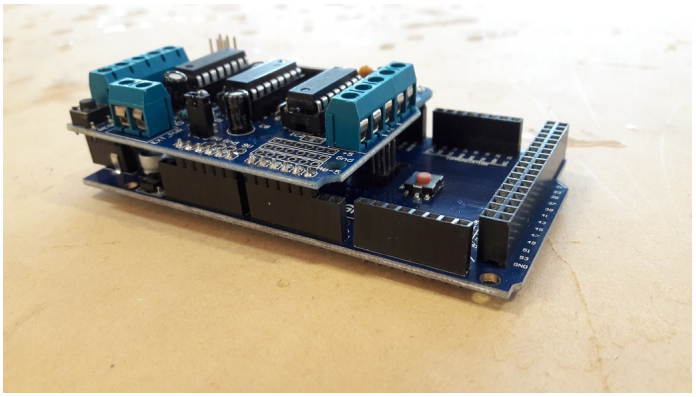



Figure 2.5: The Adafruit Motor Shield board (left)

2.1.4 Power Supply

Concerning the power supply, we need two sources of power : one which lies between 7V and 12V for the main Arduino board (the microcontroller board), and an other one of 6V for the four gear motors of the chassis.

Concerning the microcontroller board, I chose a 12V Li-ion battery, which is rechargeable and has a good capacity of 6800 mAh. For gear motors, I chose a battery pack made of four AA battery. Given that each AA battery is able to provide 1.5V, the battery pack manages to provide the nominal voltage of the gear motors (6V).

2.2 LCD Screen

In order to have an instant and direct feedback from the buggy, I decided to add a LCD screen to the structure. Indeed it helps a lot to have a display like that when you have to calibrate sensors, test an algorithm or debug the software part of your project.

The problem is that such a device possesses 16 digital pins, and at least 12 of them are necessary in order to display anything on the screen. An electrical circuit based on a LCD screen uses a lot of wires and space, as seen on the left part of the Figure 2.6.

So in order to save space and to reduce electrical connections, I decided to use an I2C module in order to easily control the LCD screen. Indeed, once this module is soldered to the LCD screen (see Figure 2.6, right part), the whole device possesses only 4 digital pins, instead of 16, while offering the same possibilities as before.

This module use the I2C (Inter-Integrated Circuit) communication technology in order to control the screen. It is a multi-master, multi-slave, single-ended, serial computer bus which could be used for making several microcontrollers comunicate with each other, or like here for sending and receiving information from sensors or actuators. Some electronic components also needs to be configured through this way before being used.

An I2C communication is composed of two pins : the SCL (Serial Clock Line) pin for synchronizing the communication and the SDA (Serial Data Line) pin for exchanging data.

With only one I2C bus, a microcontroller is able to communicate with a great number of components, since each device involved in an I2C communication has a unique 7-bit adress. Moreover, “ground” pins of all devices involved in the I2C bus have to be connected all together.

In order to being able to control the LCD screen and its I2C module, I needed to use a library called *“New LiquidCrystal”*,which is based upon the official *“LiquidCrystal”* Arduino library and adds new types of connections for the use of LCD screens.

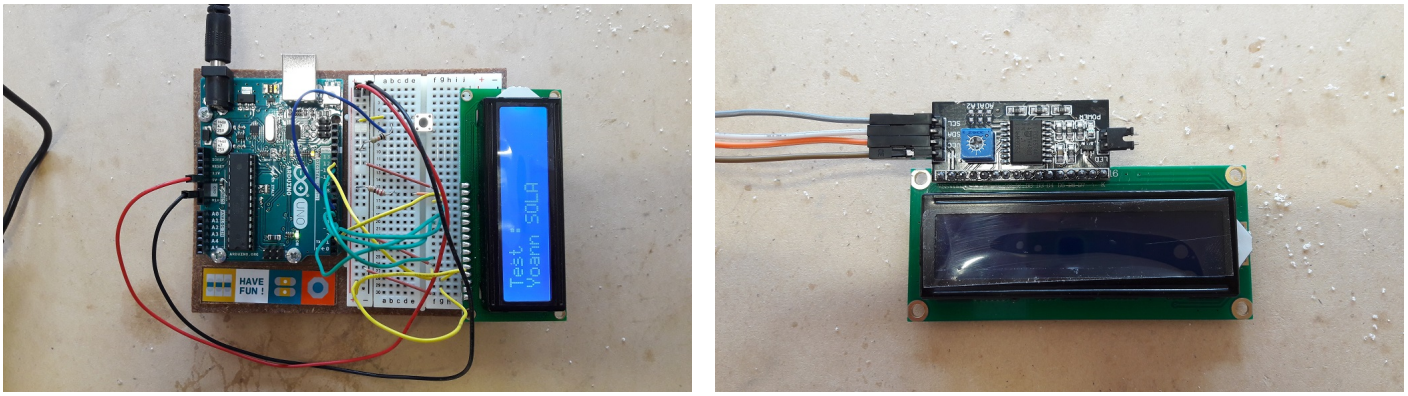


Figure 2.6: The LCD screen before (left) and after (right) the use of the I2C module

2.3 Ultrasonic Ranging Module

An ultrasonic ranging module (or ultrasonic distance sensor) is not necessary within an autonomous sailboat, but it could bring more security. Indeed it allows the future sailboat robot to detect and avoid obstacles that could suddenly appear and which were not expected during the planning of the mission. In this way, I decided to add two of these sensors to my project, one for the left side of the buggy, and the other one for its right side.

I chose the Ultrasonic Ranging Module HC-SR04 (see Figure 2.7), which has a working voltage of 5V (compatible with the Arduino Mega board). This sensor is very easy to use and is able to detect obstacles with a range from 5cm to 3m, thanks to ultrasonic waves.

Indeed, the sensor have an ultrasonic wave transmitter and a receiver that is able to detect ultrasonic wave beams. This sensor is going to emit ultrasonic wave beams and will return the time taken by these waves for being reflected by an obstacle and returning to the sensor. Once this time is measured, we can easily find the distance between the ultrasonic ranging module and the obstacle.

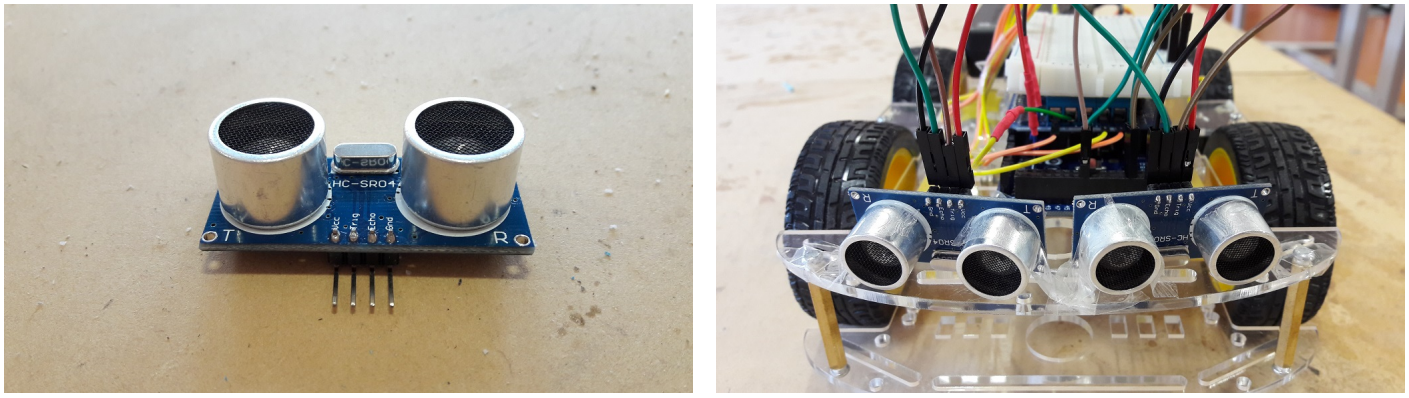


Figure 2.7: The Ultrasonic Ranging Module HC-SR04

The speed of an ultrasonic (and even sonic) wave in a perfect gas is given by the formula :

$$v = \sqrt{\gamma R_s T} \tag{2.1}$$

where v is the speed of the ultrasonic wave in m/s, γ is the heat capacity ratio or adiabatic index of the perfect gas, R_s is the specific gas constant of the perfect gas, and T is the temperature of the perfect gas in Kelvin.

We know that in the air : $\gamma = 1.4$ and $R = 287\, J.kg^{-1}.K^{-1}$.

According to 2.1, we are able to compute that the speed of ultrasonic waves in the air is : $v = 343\,\mathrm{m.s^{-1}}$ for $T = 20°\mathrm{C}$ and $v = 346\,\mathrm{m.s^{-1}}$ for $T = 25°\mathrm{C}$.
Since the variation of the speed is very weak in regards to the variation of the temperature (less than 1% here), the Ultrasonic Ranging Module HC-SR04 has an intern value for the speed of ultrasonic waves in the air set to 340 m/s.

Knowing that, we are now able to compute the distance between the sensor and the obstacle.

Indeed, we have :

$$t = \frac{2d}{v} \tag{2.2}$$

where t is the time between the emission and the reception of the reflected ultrasonic wave (which is measured by the sensor) in seconds, d is the distance between the sensor and the obstacle in meters, and v is the speed of the ultrasonic wave in the air in m/s.

Since the speed v is known (340 m/s) and the time t is returned by the ultrasonic ranging module, we can now compute the distance d as follows :

$$d = \frac{vt}{2} \tag{2.3}$$

The obstacle is now detected and its distance with respect to the robot is measured.

Moreover, some limitations have to be taken into account with the use of ultrasonic distance sensors :

- The sensor can only detect obstacles that are within a 30° cone angle. This is why I took two sensors in my project, one for each side of the robot.

- The sensor has a "blind zone" which makes that no obstacle can be detected within the first 5cm. The reason for this is that the receiver of the sensor is disabled during approximately 100 microseconds after the emission of an ultrasonic wave. Indeed since the transmitter and the receiver are very close (on the PCB of the sensor), if the receiver were not disabled after the emission of an ultrasonic wave, the wave would be instantly detected by the receiver as an "echo" before reaching any obstacles, and the robot would detect an obstacle that does not exist.

- The range of detection of the sensor can vary depending of the shape and the material of the obstacle. Indeed, these two factors can modify the way ultrasonic waves will be reflected by the obstacle. For exemple, a plane surface is able to reflect more waves than a round surface. Moreover, some materials tend to absorb waves rather than reflect them.

All these limitations are not influential in the context of our present project, since obstacles such as rocks or other boats are easy to detect for an ultrasonic range module.

Additional recommandations or characteristics can easily be found in the data sheet of the HC-SR04 module, made by *ElecFreaks*.

2.4 GPS Module

The GPS Module is only used in order to know the position of the robot, so in our project we are not interested in other information such a device could provide, such as the date, the altitude or the speed of the robot for example.

I used a cheap version of a Ublox NEO-6M GPS Module (see Figure 2.8). This device is suitable for my feasibility study, but it would not be accurate enough for a real robot, since its measuring errors concerning the GPS position can reach up to 20 meters.

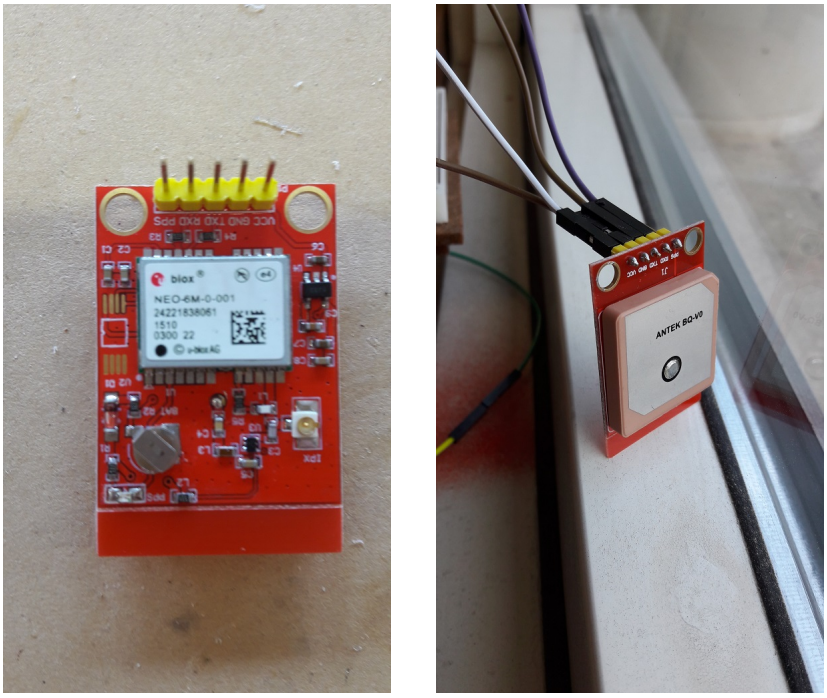


Figure 2.8: The Ublox NEO-6M GPS Module

This module is able to detect GPS satellite signals thanks to its built-in GPS antenna and to transmit data in the form of NMEA sentences thanks to a serial communication (UART).

NMEA 0183 is a both electrical and data specification used in navigation by many instruments such as sonars, gyrocompasses or GPS receivers. NMEA data are encapsulated within "NMEA sentences", which are standard data representations. There are many different type of NMEA sentences, each one containing various sort of data. We are only going to use GPGGA sentences, which is one of the most common used NMEA sentences, since it contains only basic data. Here is an example of a GPGGA sentence :

$GPGGA,194530.000,3051.8007,N,10035.9989,W,1,4,2.18,746.4,M,-22.2,M,,*6B

The type of the NMEA sentence is always specified at the beginning of the sentence. Here it is *$GPGGA*.

194530.000 is the UTC time of reception : 19 hours, 45 minutes and 30.000 seconds.

3051.8007,N is the latitude : 30 degrees and 51.8007 arc minutes, North.

10035.9989,W is the latitude : 100 degrees and 35.9989 arc minutes, West.

1 is the "Fix Quality" of the GPGGA sentence. It is an indicator of the quality of the NMEA sentence : 0=no fix, 1=GPS fix, 2=Differential GPS fix.

4 is the number of GPS satellites that are currently detected.

2.18 is the Horizontal Dilution of Precision (HDOP). It corresponds to the relative accuracy of the horizontal position.

746.4,M is the altitude of the antenna that receives the GPS signal : 746.4 meters above the mean sea level.

-22.2,M is the geoidal separation : the mean sea level is 22.2 meters below the WGS-84 earth ellipsoid.

The last two blanks are supposed to be the time since last DGPS update for the first one, and the DGPS reference station id for the second one. These information only appeared when the Fix Quality is set to 2 (Differential GPS fix), but it is not the case in this example.

**6B* is the "checksum" : it is used by all softwares in order to check if transmission errors are present in the NMEA sentence.

Before connecting the GPS module to the Arduino Mega board, I tested it by connecting it to my computer, whith a "USB to TTL" Serial cable.

In order to vizualize GPS data received by the antenna, I used *"u-center"*, a software provided by the company *u-block* which allows to configure GPS receivers and to simultaneously display various GPS data. During the first utilizations of the GPS module, GPS satellites could take a long time to be detected (up to 30 minutes in my case), but the more the module is used the faster GPS data are received.

Once I was sure that the GPS module was working correctly, I connected it to one of the Serial ports (or UARTs) of the microcontroller board.

In order to read GPS data from the module, I used a library called *"TinyGPS++"*, which allows to extract directly the latitude and the longitude of the current position from the output of the GPS module.

2.5 Electronic Compass

The electronic compass (or eCompass) was the most sensible sensor I used in my project, and required a particularly accurate calibration. I'm going to explained in detail its principle and its calibration, but you could find additional explanations in [AND10], [NXP1], [NXP2] and [NXP3].

2.5.1 Principle of the Sensor

An electronic compass is a device composed of two sensors : a 3-axis accelerometer and a 3-axis magnetometer. Its function is to indicate the heading of the robot on which it is attached. In order to do this, it has to measure the earth's magnetic field $\overrightarrow{B}$ (or the geomagnetic field) with its magnetometer, to extract the horizontal component $\overrightarrow{B_h}$ of that field (which gives us the direction of the North Magnetic Pole of the Earth) and to measure the angle between this horizontal component $\overrightarrow{B_h}$ and the axis of the robot. This angle will correspond to the heading ψ of the robot (see Figure 2.9, left part). The accelerometer allows to extract the horizontal component of the geomagnetic field by measuring the gravity vector

$\overrightarrow{G}$ of the earth and by computing the roll angle and the pitch angle of the robot (see Figure 2.9, right part).

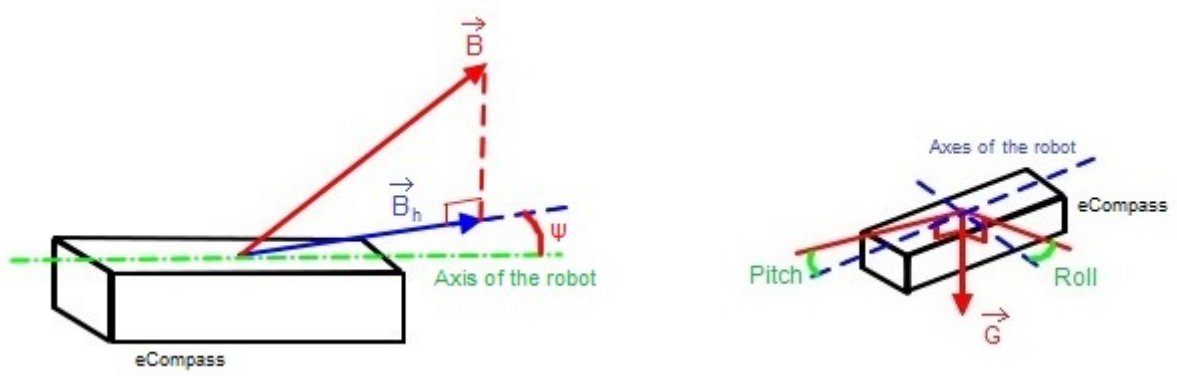


Figure 2.9: Function of the magnometer (left) and function of the accelerometer (right)

We need to calibrate separately the accelerometer and the magnetometer, then we need to implement a tilt-compensated eCompass from the raw data of these two sensors.

2.5.2 The Aerospace Coordinate System

The eCompass uses the aerospace coordinate system. It means that when the PCB of the eCompass is facing the north (when the heading is equal to 0°) and is parallel to the ground (when the roll angle and the pitch angle are equal to 0°), the eCompass is in the following configuration : the x-axis points north, the y-axis points east and the z-axis points downwards (see Figure 2.10). So the heading will be the angle measured between the x-axis of the eCompass and the horizontal component of the geomagnetic field.

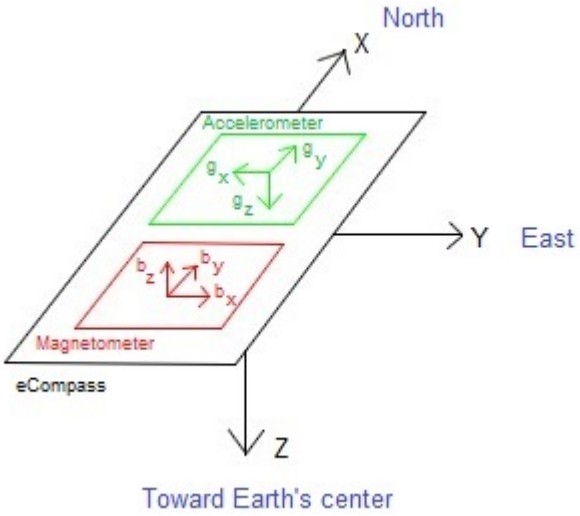


Figure 2.10: The aerospace coordinate system

Most of the time, the 3 axis of the magnetometer and the accelerometer are not oriented using this coordinate system. Indeed the sensors are arbitrarily put on the PCB of the eCompass and we have to apply corrections to their output axes in order to align them with respect to the aerospace coordinate system.

In the example shown on the Figure 2.10, we have to redefine output axes :

$$\begin{cases} G_x = g_y & B_x = b_y \\ G_y = -g_x & B_y = b_x \\ G_z = g_z & B_z = -b_z \end{cases} \tag{2.4}$$

where :
g_x, g_y and g_z are the output axes of the accelerometer in g
b_x, b_y and $b_z$are the output axes of the magnetometer in Tesla
G_x, G_y and $G_z$are the redefined axes of the accelerometer in g
B_x, B_y and $B_z$are the redefined axes of the magnetometer in Tesla

2.5.3 Calibration of the Accelerometer

The accelerometer needs to be calibrated in order to remove two sources of erroneous data : the 0g offset and the sensitivity mismatch between axes.

Indeed, each axis of the accelerometer could possibly have a constant offset (the 0g offset) and a wrong sensitivity (a gain that is not equal to 1 on this axis). So the output of each of the three axis of the accelerometer can be written as follows :

$$G_{nc} = G_{off} + (Gain \times G_c) \tag{2.5}$$

where :
G_{nc} is the non-calibrated output of one of the axes of the accelerometer in g
G_{off} is the 0g offset present on this axis in g
$Gain$ corresponds to the sensitivity of the axis
G_c is the calibrated output of the axis and corresponds to the real acceleration acting on this axis in g

The aim of the calibration is to compute G_c from G_{nc} , for each of the three axis. In order to do that, we need to compute the $Gain$ and G_{off} for the three axis.

A simple and efficient method consists in measuring outputs of each axis when the axis is placed into a +1g field and a -1g field. It means that we have to place the PCB of the eCompass in two orientations, completely immobile : one in order to put the axis in a vertical position and pointing upwards (in a -1g field), and an other one in order to put it in a vertical position and pointing downwards (in a +1g field). Using 2.5 and since the gravity is the only acceleration acting on the eCompass , we have the following relations for each of the accelerometer's axis :

$$G_{+1g} = G_{off} + (Gain \times 1g) \tag{2.6}$$

$$G_{-1g} = G_{off} - (Gain \times 1g) \tag{2.7}$$

where G_{+1g} is the output (in g) of the axis placed in a +1g field, and G_{-1g} is the output (in g) of the axis placed in a -1g field.

Using 2.6 and 2.7, we can now compute the 0g offset and the gain of each axis as follows :

$$G_{off} = 0.5 \times (G_{+1g} + G_{-1g}) \tag{2.8}$$

$$Gain = 0.5 \times \left(\frac{G_{+1g} - G_{-1g}}{1g} \right) \tag{2.9}$$

Once the 0g offset and the gain are computed for each axis, we can find the real acceleration acting on them.

For the three axes of the accelerometer, wa have :

$$G_c = \frac{G_{nc} - G_{off}}{Gain} \tag{2.10}$$

2.5.4 Calibration of the Magnetometer

A magnetometer is able to detect the geomagnetic field but also various other magnetic fields, which could result in interferences. In this project, we only want to measure the geomagnetic field, in order to know the direction of the North Magnetic Pole. The calibration of the magnetometer consists in removing interferences from its data, which are mainly induced by two sources : the hard-iron effect and the soft-iron effect.

In the absence of any hard-iron and soft-iron interferences and after rotations in yaw ψ, pitch θ and roll ϕ applied to the magnetometer, the magnetic field $\mathbf{B_m}$ measured by the magnetometer is :

$$\mathbf{B_m} = \mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\,\mathbf{R_z}(\psi) \begin{pmatrix} B\cos\delta \\ 0 \\ B\sin\delta \end{pmatrix} \tag{2.11}$$

where B is the geomagnetic field strength in Tesla and δ is the angle of inclination of the geomagnetic field in degrees.

These two characteristics vary with with the position of the robot on the earth's surface. Their value could be found on the World Data Center for Geomagnetism's website. It is important to note that we do not need their value for computing the heading of the robot (see the section 2.5.5).

Let us recall that the rotation matrices $\mathbf{R_x}(\phi)$, $\mathbf{R_y}(\theta)$ and $\mathbf{R_z}(\psi)$ are defined as follows :

$$\mathbf{R_x}(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\,\phi & sin\,\phi \\ 0 & -sin\,\phi & cos\,\phi \end{pmatrix}, \mathbf{R_y}(\theta) = \begin{pmatrix} cos\,\theta & 0 & -sin\,\theta \\ 0 & 1 & 0 \\ sin\,\theta & 0 & cos\,\theta \end{pmatrix}, \mathbf{R_z}(\psi) = \begin{pmatrix} cos\,\psi & sin\,\psi & 0 \\ -sin\,\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.12}$$

The hard-iron effect is due to the presence of permanently magnetized ferromagnetic components that are near the eCompass. For example in my project, magnets which are within ultrasonic range modules are an important source of hard-iron interferences. Indeed, permanently magnetized ferromagnetic components add an important magnetic field to measurements.

Since these components and the magnetometer are all fixed on the platform (the buggy or the saiboat), they are moving and rotating together. It means that hard-iron interferences will always result in a constant vector $\mathbf{V_{hard}}$ that will be added to measurements, independently of the position or the orientation of the robot.

Moreover, like for the accelerometer, the magnetometer has a constant zero field offset on each of his axes, which results in a constant additive vector $\mathbf{V_{0-field}}$.

So we can write the final constant vector $\mathbf{V}$ which is going to be added to the magnetic field defined in 2.11, as :

$$\mathbf{V} = \mathbf{V_{hard}} + \mathbf{V_{0-field}} \tag{2.13}$$

The soft-iron effect is due to the magnetic field induced by the geomagnetic field into normally unmagnetized ferromagnetic components. According to [NXP2], this interfering field is assumed to be linear and can be modelled by a 3 by 3 matrix $\mathbf{W_{soft}}$.

Moreover, like for the accelerometer, there is a sensitivity mismatch between the three axes of the magnetometer which can be modelled by a 3 by 3 matrix $\mathbf{W_{gain}}$.

Finally, a third 3 by 3 matrix $\mathbf{W_{NonOrthog}}$ can be defined in order to model the lack of perfect orthogonality between the axes of the magnetometer.

We can defined a 3 by 3 matrix $\mathbf{W}$ which is the combination of the previous sources of interference and which is going to multiply the magnetic field defined in 2.11 :

$$\mathbf{W} = \mathbf{W_{NonOrthog}}\mathbf{W_{gain}}\mathbf{W_{soft}} \tag{2.14}$$

We can now write the magnetic field mesured by the magnetometer and defined in 2.11 , as follows :

$$\mathbf{B_m} = \mathbf{W}\,\mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\,\mathbf{R_z}(\psi)\begin{pmatrix} B\cos\delta \\ 0 \\ B\sin\delta \end{pmatrix} + \mathbf{V} \tag{2.15}$$

where $\mathbf{W}$ is the matrix defined in 2.14 and $\mathbf{V}$ is the vector defined in 2.13.

The calibration of the magnetometer consists in computing $\mathbf{W}$ and $\mathbf{V}$, in order to remove them from $\mathbf{B_m}$.

This computation can be made thanks to the Method of Least Squares, based on a great number of samples taken from magnetometer's measurements.

In order to save time, I used a software called *"MagMaster"*, which allows to compute $\mathbf{W}$ and $\mathbf{V}$, from magnetometer's measurements (see Figure 2.11).

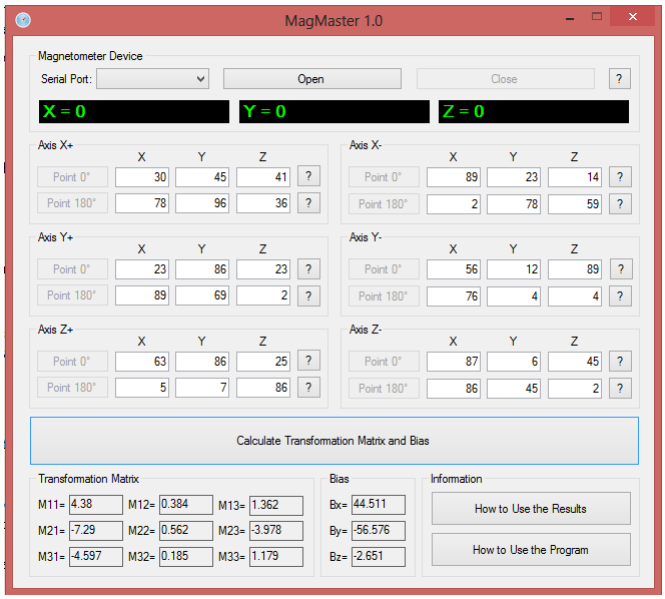


Figure 2.11: MagMaster

2.5.5 Computation of the Heading

Once both accelerometer and magnetometer are calibrated, we can implement a tilt-compensated eCompass.

As explained in the section 2.5.1, we need to compute the direction of the gravitational acceleration's vector, in order to know where the horizontal plane is with respect to the robot. In order to know that, we need to compute the pitch angle θ and the roll angle ϕ of the robot.

The magnetometer will measure the geomagnetic field's vector. Thanks to the computation of the pitch angle and the roll angle, we could extract the horizontal component of the geomagnetic field's vector.

The final will consist in measuring the angle between this horizontal component and the axis of the robot, which will correspond to the heading (or the yaw) ψ of the robot.

We know that we can write the calibrated output $\mathbf{G_c}$ of the accelerometer as follows (after applying the gain and the 0g offset corrections to the raw data, as explained in the section 2.5.3) :

$$\mathbf{G_c}= \begin{pmatrix} G_{cx} \\ G_{cy} \\ G_{cz} \end{pmatrix} = \mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\,\mathbf{R_z}(\psi) \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \tag{2.16}$$

where $\mathbf{R_x}(\phi)$, $\mathbf{R_y}(\theta)$ and $\mathbf{R_z}(\psi)$ are the rotation matrices defined in 2.12, in g is the gravitational acceleration equal to $9.81\,\mathrm{m.s^{-2}}$.

Since we assume than no other acceleration than the gravitation acceleration is acting on the axes of the accelerometer, we can say that the rotation of the robot in yaw ψ does not affect the accelerometer's measurement. Using 2.16 and since rotation matrices are orthogonal, we can write :

$$\mathbf{R_y}(-\theta)\,\mathbf{R_x}(-\phi) \begin{pmatrix} G_{cx} \\ G_{cy} \\ G_{cz} \end{pmatrix} = \mathbf{R_z}(\psi) \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \tag{2.17}$$

By using 2.12, we can develop 2.17 as follows :

$$\begin{pmatrix} cos\,\theta & 0 & sin\,\theta \\ 0 & 1 & 0 \\ -sin\,\theta & 0 & cos\,\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\,\phi & -sin\,\phi \\ 0 & sin\,\phi & cos\,\phi \end{pmatrix} \begin{pmatrix} G_{cx} \\ G_{cy} \\ G_{cz} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \tag{2.18}$$

$$\begin{pmatrix} cos\,\theta & sin\,\theta sin\,\phi & sin\,\theta cos\,\phi \\ 0 & cos\,\phi & -sin\,\phi \\ -sin\,\theta & cos\,\theta sin\,\phi & cos\,\theta cos\,\phi \end{pmatrix} \begin{pmatrix} G_{cx} \\ G_{cy} \\ G_{cz} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \tag{2.19}$$

By using the second line of the linear system defined in 2.19, we can compute the roll angle ϕ:

$$G_{cy}cos\,\phi - G_{cz}sin\,\phi = 0 \tag{2.20}$$

$$\phi = atan2\left(G_{cy}\,,\,G_{cz}\right) \tag{2.21}$$

By using the first line of the linear system defined in 2.19 and 2.21, we can compute the pitch angle θ:

$$G_{cx}cos\,\theta + G_{cy}sin\,\theta sin\,\phi + G_{cz}sin\,\theta cos\,\phi = 0 \tag{2.22}$$

$$\theta = atan2\left(-G_{cx}\,,\,G_{cy}sin\,\phi + G_{cz}cos\,\phi\right) \tag{2.23}$$

Now that we have computed the roll angle and the pitch angle, we can use them with the magnetometer's mesurement in order to compute the heading (or yaw) angle ψ.

Using 2.15, we can write :

$$\mathbf{R_z}\left(\psi\right)\begin{pmatrix} B\,cos\,\delta \\ 0 \\ B\,sin\,\delta \end{pmatrix} = \mathbf{R_y}\left(-\theta\right)\mathbf{R_x}\left(-\phi\right)\mathbf{W^{-1}}\left(\mathbf{B_m} - \mathbf{V}\right) \tag{2.24}$$

where $\mathbf{B_m}$ is the raw data vector of the magnetometer in Tesla.

Since $\mathbf{R_x}\left(\phi\right)$, $\mathbf{R_y}\left(\theta\right)$, $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{B_m}$ are now known, we can note the right part of the equation 2.24 as :

$$\mathbf{B_c} = \begin{pmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{pmatrix} = \mathbf{R_y}\left(-\theta\right)\mathbf{R_x}\left(-\phi\right)\mathbf{W^{-1}}\left(\mathbf{B_m} - \mathbf{V}\right) \tag{2.25}$$

where $\mathbf{B_c}$ represents magnetometer's measurements which have been calibrated with respect to hard-iron and soft-iron effects, and for which the rotations in pitch and roll have been removed.

Using 2.24 , 2.12 and 2.25 , we have :

$$\begin{pmatrix} cos\,\psi & sin\,\psi & 0 \\ -sin\,\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} B\,cos\,\delta \\ 0 \\ B\,sin\,\delta \end{pmatrix} = \begin{pmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{pmatrix} \tag{2.26}$$

After developping 2.26, the first and second lines give us :

$$cos\,\psi\,B\,cos\,\delta = B_{cx} \tag{2.27}$$

$$sin\,\psi\,B\,cos\,\delta = B_{cy} \tag{2.28}$$

Finally thanks to 2.27 , 2.28 and 2.25 , we can compute the heading (or yaw) angle ψ as follows :

$$\psi = atan2\,(-B_{cy}\,,\,B_{cx}) \tag{2.29}$$

2.5.6 Components Involved

In order to implement the tilt-compensated eCompass of my buggy, I used a PCB which contains a 3-axis accelerometer ADXL345 and a 3-axis magnetometer HMC5883L (see Figure 2.12). Both of these sensors were connected to my Arduino Mega board thanks to I2C communications.

In order to communicate with these sensors, I used three unofficial Arduino libraries : *"I2Cdev.h"*, *"HMC5883L.h"* and *"ADXL345.h"*.

These cheap sensors give pretty accurate measurements with this whole calibration, but the robot must not move with high accelerations during the computation of the heading, since it would distort accelerometer's measurements.

In order to fix this problem, we could add a 3-axis gyroscope and use an advanced sensor fusion algorithm in order to have more stable measurements of the roll and the pitch angles.

Figure 2.12: eCompass composed of a 3-axis accelerometer ADXL345 and a 3-axis magnetometer HMC5883L

2.6 Wind Sensor

As explained in [SAU09], there are three types of wind sensors : the mechanical sensor, the contactless mechanical sensor and the ultrasonic sensor. For my project, I chose the cheepest one : the mechanical wind sensor.

It is composed of two parts : a rotary encoder and a wind vane.

2.6.1 The Rotary Encoder

The rotary encoder I used is a Keyes KY-040 Rotary Encoder Module (see Figure 2.13, left part). This device is a simple potentiometer (or knob) which can rotate continuously, without never being blocked. It also allows to know for how much the knob has been rotated and its direction of rotation.

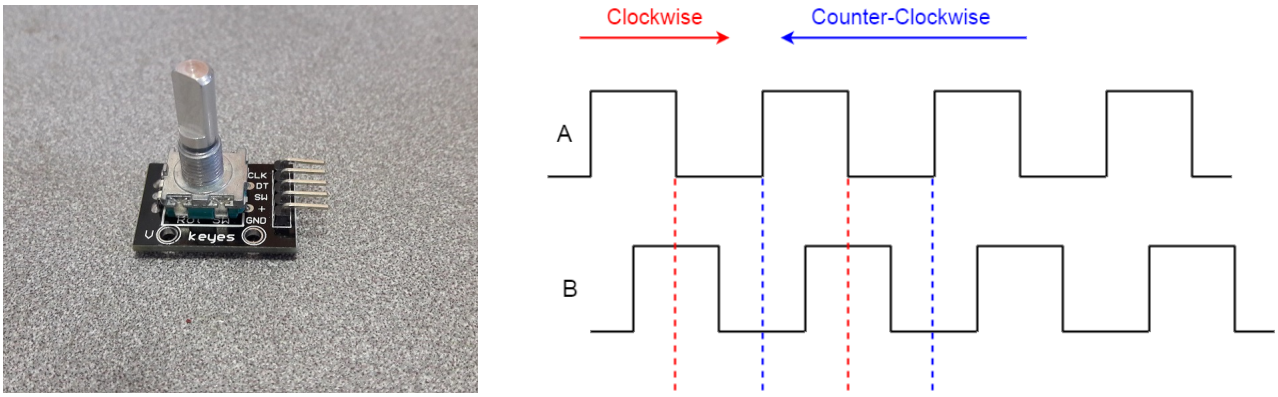


Figure 2.13: The Keyes KY-040 Rotary Encoder Module (left) and the two signals used in this module (right)

A rotary encoder uses two square wave outputs A and B which are in quadrature (they have a phase offset equal to 90 degrees), as represented on the right part of the Figure 2.13.

The rotary encoder has several steps which can be felt when you turn the knob. Each time a step is crossed, one of the two signals A or B changes its value (0 or 1).

Moreover, the microcontroller board will read the value of the signal B each time that the value of the signal A is dropping from 1 to 0.

As shown in the Figure 2.13, if the value of the signal B is 1, it means that the knob is being turned clockwise. At the contrary if its value is 0, it means that knob is being turned counter-clockwise.

Finally, the number of crossed steps allows us to know how much the knob has rotated. For example, there are 60 steps within the device I used, which means that each time a step is crossed, the knob has done a rotation of 6 degrees (360 degrees divided by 60). So if we are able to count the number of steps crossed from the initial position of the knob, we can know the rotation completed by the knob with an accuracy of 6 degrees.

In order to count the number of steps crossed at any moment (during the execution of the program), we have to use "interruption functions".

Indeed, as explained in the section 2.1.2, the Arduino Mega has 6 digital inputs that could be used for this purpose. We just have to attach an interruption function to two digital inputs (one for each square wave outputs A and B), and each time that the value of A or B will change, the function will interrupt the program and the change of value will be taken into account.

2.6.2 The Wind Vane

In order to detect the current direction of the wind, a wind vane is needed. Indeed, this wind vane will make the rotary encoder rotate according to the direction of the wind.

Figure 2.14: The 3D printed wind vane

I used the software *"SolidWorks"* in order to create a 3D model of my wind vane, and to print it with a 3D Printer (see Figure 2.14).

The 3D Printer used was a "MakerBot Replicator 2", using ABS filaments. This material (Acrylonitrile Butadiene Styrene) is very light and resistant, which is perfect for a wind vane that has to be moved by the wind.

I paid attention to make a hole in the wind vane with the same shape as the knob of the rotary encoder : a circle with a cutout part. The knob will be placed in this hole (see Figure 2.15).

This wind vane is pretty accurate in most cases, but does not work well with a weak wind. Indeed, the wind is then too weak to make the knob cross a step.

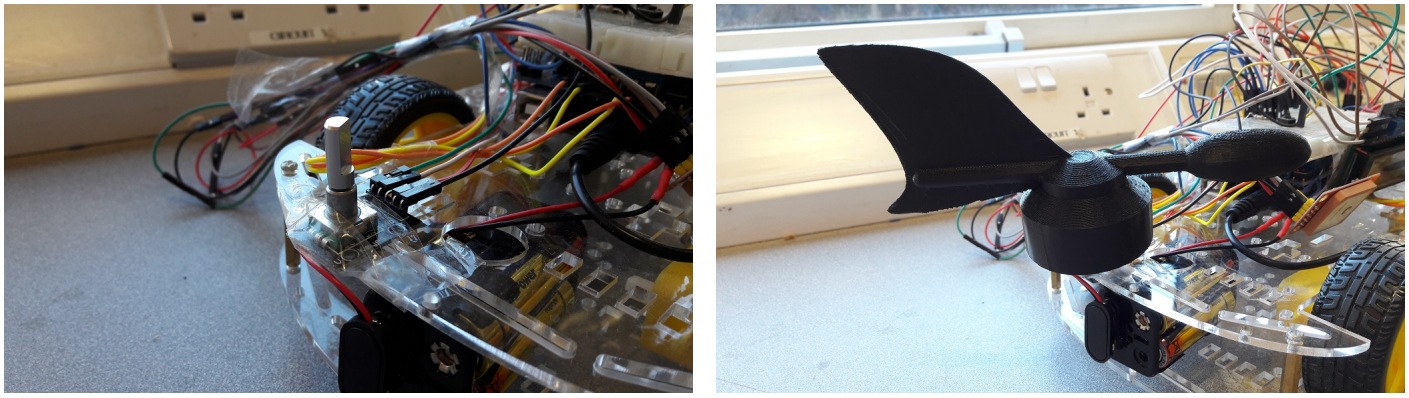


Figure 2.15: Positioning of the wind vane

Chapter 3

Simulation of an autonomous sailboat

The second part of my internship consisted in simulate the future autonomous sailboat during two situations : following a path, and staying in a predefined zone while avoiding several obstacles.

I will describe the model of the autonomous sailboat used in the two simulations, then I will describe the algorithms used in both simulations.

3.1 Modelling an autonomous sailboat

In order to test my algorithms, I used the following model in order to simulate the behaviour of a sailboat robot, taken in [JAU15] :

$$\begin{cases} \dot{x} & = v\,cos\,\theta + p_1 a\,cos\,\psi \\ \dot{y} & = v\,sin\,\theta + p_1 a\,sin\,\psi \\ \dot{\theta} & = \omega \\ \dot{v} & = \frac{f_s sin\,\delta_s - f_r sin\,u_1 - p_2 v^2}{p_9} \\ \dot{\omega} & = \frac{f_s(p_6 - p_7 cos\,\delta_s) - p_8 f_r cos\,u_1 - p_3 \omega v}{p_{10}} \\ f_s & = p_4 \left\|\mathbf{w_{ap}}\right\| sin\left(\delta_s - \psi_{ap}\right) \\ f_r & = p_5 v\,sin\,u_1 \\ \sigma & = cos\,\psi_{ap} + cos\,u_2 \\ \delta_s & = \begin{cases} \pi + \psi_{ap} & if\ \sigma \leq 0 \\ -sign\left(sin\,\psi_{ap}\right) \cdot u_2 & otherwise \end{cases} \\ \mathbf{w_{ap}} & = \begin{pmatrix} a\,cos\left(\psi - \theta\right) - v \\ a\,sin\left(\psi - \theta\right) \end{pmatrix} \\ \psi_{ap} & = angle\left(\mathbf{w_{ap}}\right) \end{cases} \tag{3.1}$$

The establishment of these state equations is explained in [JAU04] and [JAU11].

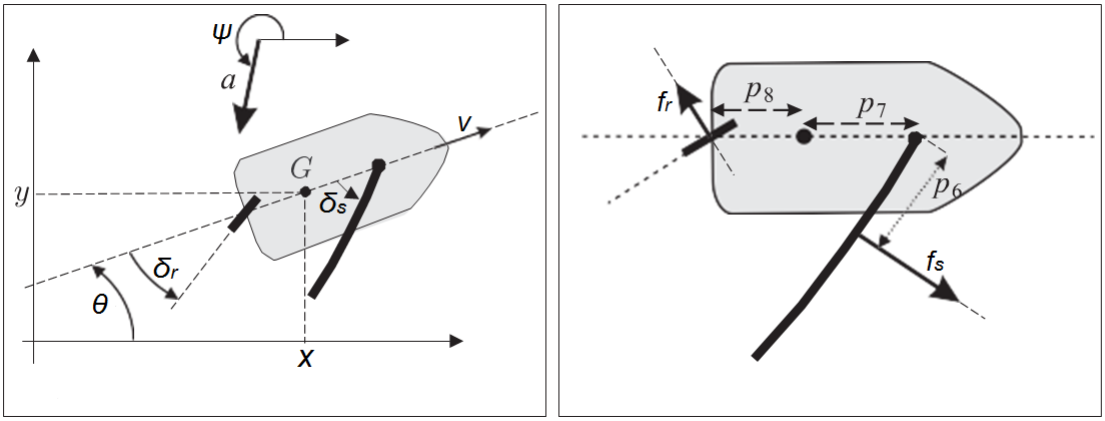


Figure 3.1: The model of the autonomous sailboat

The state vector $x = (x, y, \theta, v, \omega)$ has a dimension equal to 5.

x and y represent the GPS coordinates of sailing robot's centre of gravity, and θ is its heading.

v is its linear acceleration, while $\omega$is its angular acceleration.

The two inputs u_1 and u_2 represent respectively the angle δ_r of the rudder and the maximal length δ_s^{max} of the sheet that allows to adjust the sail.

The last six equations are not state equations, but are just written in order to simplify the previous state equations.

σ is a parameter that indicates if the sheet of the sailboat is stretched or not, and δ_s is angle of the sail witch respect to the hull of the sailboat.

a is the speed of the wind and ψ is its direction.

$\mathbf{w_{app}}$ is the apparent vector of the wind, from the point of view of the sailboat, and ψ_{app} is its direction or angle.

f_s is the force applied by the wind on the sail of the sailboat and f_r is the force applied by waves on its rudder.

p_1 , p_2 , . . . , p_{10} are parameters related to the construction the sailboat itself :

- p_1 is the drift coefficient
- p_2 is the tangential friction coefficient between the boat and water, and p_3 is the angular friction coefficient
- p_4 is the sail lift coefficient and p_5 is rudder lift coefficient
- p_6, p_7 and p_8 are distances shown on Figure 3.1
- p_9 is the mass of the sailboat and p_{10} is its angular inertia

According to [JAU04] and [JAU11], the model 3.1 is known to reproduce very well the behaviour of a real sailboat, and it will be suitable for our next simulations.

3.2 A Line-Following Algorithm

For the first simulation, I decided to implement on Matlab a well-known line-following algorithm taken from [JAU12] and [JAU15].

3.2.1 Principle of the Algorithm

This controller is a "model-free controller", which means that the controller is not based on the state equations of the sailing robot. Indeed, instead of being constructed from a model that could not perfectly reproduce the reality of a complex system, this kind of controller tries to mimic natural behaviours : this is biomimetics.

In our case, the previous model is quite realistic, but can not take into account all aspects of a real sailboat, such as the elasticity of the sail or its curvature for example. The algorithm used in this section try to reproduce the real behaviour of a human navigator, without taking into account any state equation.

The model 3.1 will only help to test the controller, just for the needs of the simulation.

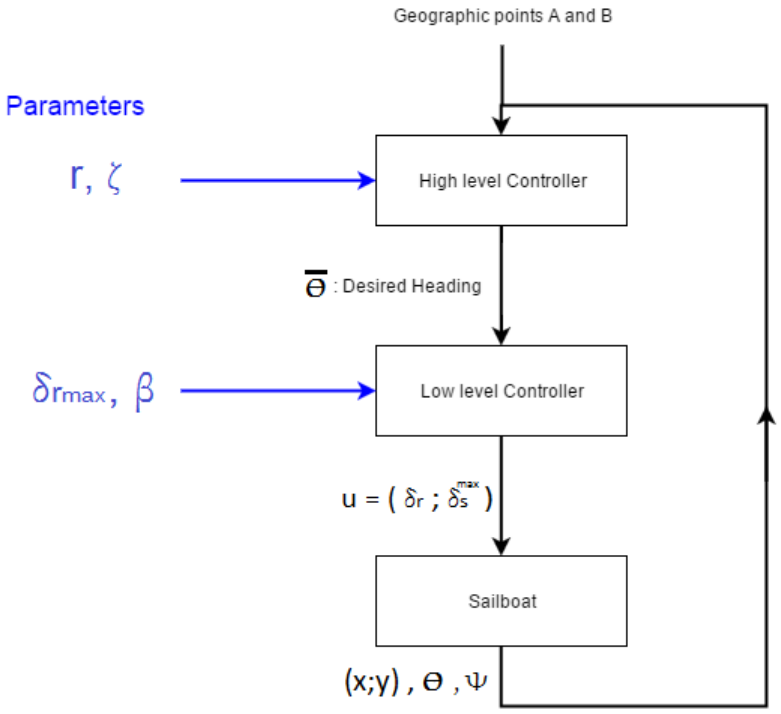


Figure 3.2: The Line Following Algorithm

As shown in the Figure 3.2, this controller has two levels : a high level controller that generates lines that have to be followed by the sailboat, and a low level controller that acts on the rudder and the sail of the sailboat in order to follow these lines.

Indeed, the high level controller takes two points **a** and **b** as inputs, generates an attractive line that crosses these two points, and gives a desired heading $\bar{\theta}$ (that tends to follow the attractive line) as an output. Moreover, this controller has two parameters : the width $2r$ of the "safety corridor" and the characteristic angle ζ of the "no-go zone" (see section 3.2.2).

The low level controller takes the desired heading $\bar{\theta}$ as an input and generates u_1 (the angle of the rudder) and u_2 (the length of the sheet that adjust the sail) as outputs, which will make the sailboat follow $\bar{\theta}$. This controller has two parameters : the maximal angle δ_r^{max} of the rudder and the angle β of the sail in a "beam reach" configuration (see section 3.2.2).

The sensors of the sailboat allow us to measure the current geographic position (x, y) of the sailing robot, its heading θ and the direction of the wind ψ. Then, these measurements are taken into account by the high level controller in order to generate a new desired heading $\bar{\theta}$, and a new loop is launched by this line-following controller.

3.2.2 Detailled Algorithm

I'm going to describe more technically the line-following algorithm, but further mathematical explanations could be found in [JAU12] and [JAU15].

Here is the mathematical implementation of the algorithm :

Algorithm 3.1 Line-Following Algorithm

1 $e = det\left(\frac{\mathbf{b}-\mathbf{a}}{\|\mathbf{b}-\mathbf{a}\|}, \mathbf{m}-\mathbf{a}\right)$
2 if $\mid e \mid > r$ then $q = sign\,(e)$
3 $\varphi = angle\,(\mathbf{b}-\mathbf{a})$
4 $\bar{\theta} = \varphi - atan\left(\frac{e}{r}\right)$
5 if $cos\left(\psi - \bar{\theta}\right) + cos\,\zeta < 0$
6 or $(\mid e \mid < r$ and $(\cos(\psi - \varphi) + \cos\zeta < 0))$
7 then $\bar{\theta} = \pi + \psi - \mathrm{q}\zeta$
8 $\delta_r = \frac{\delta_r^{max}}{\pi}\, sawtooth\left(\theta - \bar{\theta}\right)$
9 $\delta_s^{max} = \frac{\pi}{2}\left(\frac{cos\left(\psi-\bar{\theta}\right)+1}{2}\right)^{\frac{log\left(\frac{\pi}{2\beta}\right)}{log(2)}}$

Lines from 1 to 7 correspond to the high level controller and lines 8 and 9 correspond to the low level controller.

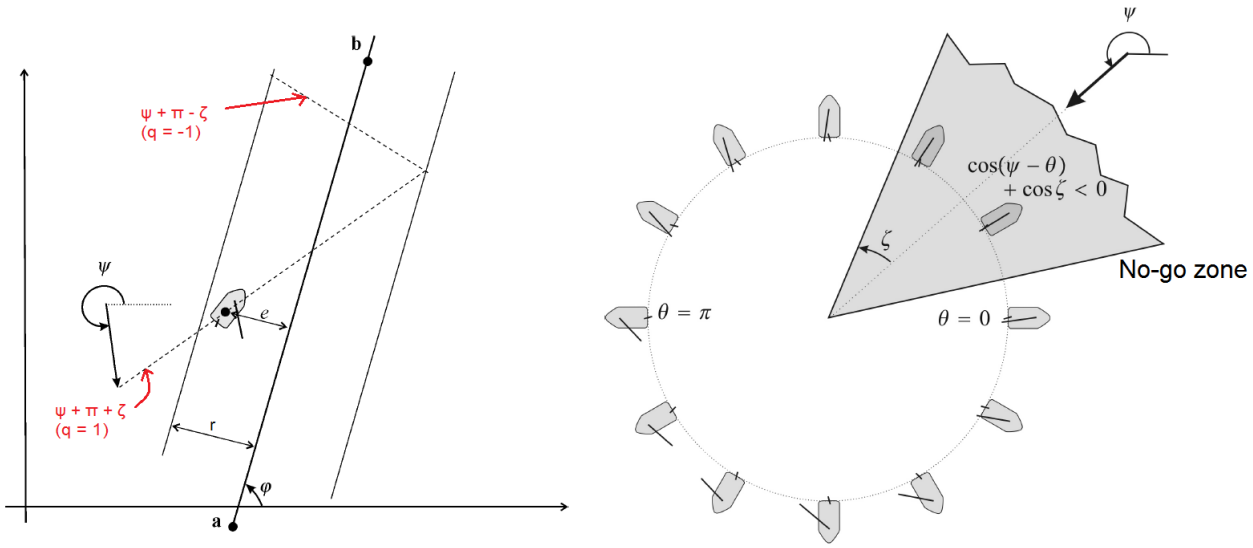


Figure 3.3: The safety corridor (left part) and the no-go zone (right part)

The line 1 allows to compute the algebraic error e between the sailboat and the line that must be followed. If $e < 0$, the sailboat is at the left of the line ; if $e = 0$, the center of gravity of the sailboat is exactly on the line ; if $e > 0$, the sailboat is at the right of the line.

The line 2 allows to check if the sailboat is out of the "safety corridor" in which the sailboat must absolutely stay. The parameter r represents half of the width of this corridor. If the sailboat is beyond a distance of r from the line that must be followed, the variable q is updated. The aim of the variable q is to direct the sailboat toward the inside of the "safety corridor" (see Figure 3.3, left part) . The variable q will be used in the line 7, during the computation of the desired heading $\bar{\theta}$.

In line 3, the algorithm basically computes the angle between the x-axis of the local plane and the direction of the line crossing the points **a** and **b** (which is the line that must be followed).

The line 4 computes the desired heading $\bar{\theta}$ in order to bring the sailboat closer to the line that must be followed, without taking into account the direction of the wind. The desired heading $\bar{\theta}$ lies between $-\frac{\pi}{2}$ and $+\frac{\pi}{2}$.

Lines 5, 6 and 7 allows to adjust the desired heading $\bar{\theta}$ in taking into account the direction ψ of the wind. Indeed, the sailboat can not navigate while facing the wind, because the sail would not be inflated, and would flap like a flag. In order to avoid this situation, the sailboat must not enter in what we called a "no-go zone". The no-go zone is a set of heading that the sailboat must not take and is characterized by the angle ζ. Indeed, the no-go zone is an angle which has a width of 2ζ and which is centered on the direction ψ of the wind (see Figure 3.3, right part). The line 7 allows the sailboat to enter in a "close hauled" configuration, instead of entering in the no-go zone (see Figure 3.3, left part).

The line 8 computes the angle δ_r of the rudder, which corresponds to the entry u_1 of the autonomous sailboat.

The function $sawtooth\,(x)$ is a function that allows to "filter" the angle x and to return its value into the interval $[-\pi, \pi]$. A good expression of this function could be :

$$sawtooth\,(x) = 2\,atan\,(tan\,(0,5 \times x)) \tag{3.2}$$

The parameter δ_r^{max} is the maximal angle of the rudder and can be adjusted by the user depending on the structure of the hull of the sailboat. The angle δ_r computed in the line 8 lies between $-\delta_r^{max}$ and $+\delta_r^{max}$ and is proportional to the difference between the current heading of the sailboat θ and the desired heading $\bar{\theta}$.

The line 9 computes the length δ_s^{max} of the sheet that allows to adjust the angle of the sail, which corresponds to the entry u_2 of the autonomous sailboat.

δ_s^{max} is proportionnal to the difference between the direction ψ of the wind and the desired heading $\bar{\theta}$, and lies between 0 and $\frac{\pi}{2}$. In the following explanations we are going to assume that $\delta_s = \delta_s^{max}$, because the angle of the sail δ_s has the same behaviour as the length of the sheet δ_s^{max}.

The expression of δ_s^{max} was built from a cardioid model, which allows to have the following values :

$$\delta_s^{max} = \begin{cases} 0 & when\ \bar{\theta} = \psi \pm \pi \ \ (the\ sailboat\ is\ facing\ the\ wind) \\ \beta & when\ \bar{\theta} = \psi \pm \frac{\pi}{2} \ \ (both\ directions\ of\ the\ sailboat\ and\ the\ wind\ are\ perpendicular) \\ \frac{\pi}{2} & when\ \bar{\theta} = \psi \ \ (the\ sailboat\ is\ in\ the\ same\ direction\ of\ the\ wind) \end{cases} \tag{3.3}$$

where β is a parameter adjustable by the user, used during a "beam reach" configuration.

These values allows to reproduce the real navigation of a sailboat.

Results of the simulation

Results of the simulation can be seen in the Annex 2.

3.3 Anchoring Zone and Obstacle Avoidance

The second simulation that I made during my internship had two goals : the autonomous sailboat had to stay in a predefined zone while avoiding several obstacles.

Indeed, the user of this simulation can defined a rectangular zone in which the sailboat must absolutely stay, and can also place on the local plane (or the map) several points that must be avoided, independently of the anchoring zone. We could for example imagine a navigator who wants to place his sailboat in a port, without using a real anchor. The sailboat would have to stay in a precise zone in this port and would have to avoid various obstacles such as other boats, rocks or port structures.

In this section, the zone in which the sailboat must stay will be called the "anchoring zone" and the points to avoid will be called "obstacles".

3.3.1 Principle of the Simulation

This simulation is more complex than the first one from the section 3.2, and I decided to use two robust and effective tools : Voronoi Diagrams and the Line-Following Algorithm from the section 3.2.2.

The Voronoi Diagram is a powerful tool that allows to easily compute safe paths to follow, in order to go between obstacles on the map and avoid them (see section 3.3.2 for further explanations). Moreover, as seen in the section 3.2.3, the line-following algorithm allows to follow lines very efficiently, independantly of the direction of the wind.

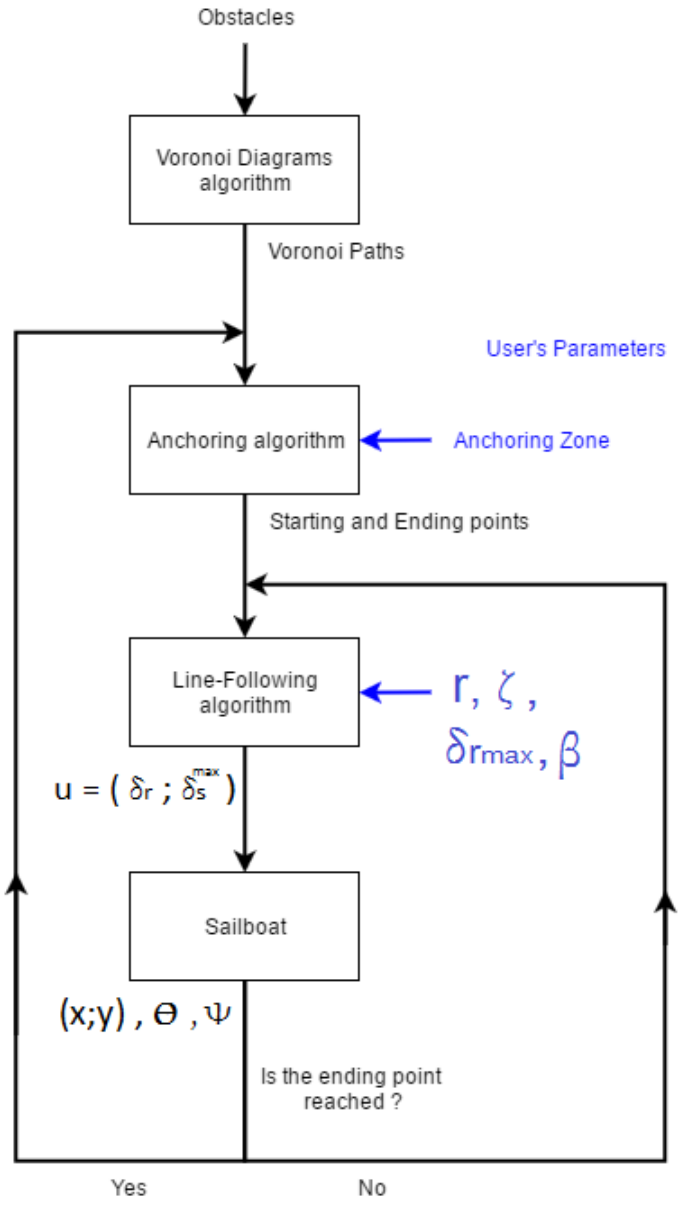


Figure 3.4: The anchoring zone and obstacle avoidance simulation

As seen on the Figure 3.4, the whole simulation is composed of three distinct algorithms : the "Voronoi Diagrams algorithm", the "Anchoring algorithm" and the "Line-Following algorithm".

The input of the Voronoi Diagram algorithm is the list of obstacles' coordinates. This algorithm will take into account these obstacles and will compute the safest paths that go between each of the obstacles (see section 3.3.2). These safe paths are called "Voronoi paths". Then, all Voronoi paths are put into a list and return as an output.

The Anchoring algorithm has two inputs : Voronoi paths computed by the Voronoi Diagram algorithm and the anchoring zone in which the sailboat must stay. Depending on the position of the sailboat, the Anchoring algorithm will choose the most appropriate path from the list of Voronoi paths (in order to avoid obstacles) and will also be able to shorten it in order to stay in the Anchoring zone (see section 3.3.3). Once the path to follow is determined, its starting point and its ending point are returned by the Anchoring algorithm as an output.

The Line-Following algorithm is exactly the same that the one from the section 3.2, so it will not be explained in detail in this section. This algorithm will make the sailboat successively follow all paths determined by the Anchoring algorithm. Its input is the starting point and the ending point which composed the line to follow choosen by the Anchoring algorithm. Its output is the vector u composed of the angle of the rudder δ_r and the maximal length of the sheet δ_s^{max} , which allows to controll the sailboat. As in the section 3.2, the parameters r, ζ, β and δ_r^{max} can still be adjusted, depending on the configuration of obstacles, the size of the anchoring zone and the sailboat itself.

Finally, the current geographic position (x, y) of the sailing robot, its heading θ and the direction of the wind ψ measured by sensors will be used in order to know the situation of the sailboat. If the ending point of the line to follow has been reached, a new path is computed by the Anchoring algorithm in order to stay in the anchoring zone. If not, the sailboat keep following the current line thanks to the Line-Following algorithm.

The Voronoi Diagrams algorithm and the Anchoring algorithm are now going to be explained in detail.

3.3.2 Voronoi Diagrams algorithm

The computation of a Voronoi diagram can be implemented in various way. The following method was taken from [JAU15].

Definition of Voronoi Diagrams

As explained in the previous section, the Voronoi Diagrams algorithm takes all obstacles as inputs and computes the safest paths for avoiding them.

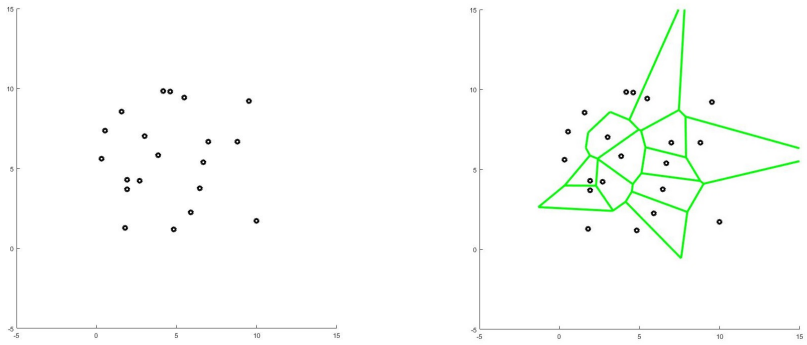


Figure 3.5: Obstacles to avoid before (left) and after the computation of Voronoi paths (right)

The Figure 3.5 shows us an illustrative example of the Voronoi Diagrams algorithm. On the left side of the Figure 3.5, black points represent the obstacles to avoid ; on the right side of the Figure 3.5, green lines represent "Voronoi paths", which are at equal distance of the two obstacles between which they go. We are going to describe more precisely how these Voronoi paths are computed.

Let us call O_1 , ... , O_m the obstacles to avoid, where m is the number of obstacles.

On the right side of the Figure 3.5, we can see that each obstacle O_i is inside a polygon $\mathbb{P}_i$ delimited by green lines.

A Voronoi Diagram is basically the collection of all the polygons $\mathbb{P}_i$, each one associated to its obstacle O_i, and defined by :

$$\mathbb{P}_i = \left\{ x \, \epsilon \, \mathbb{R}^2 \, / \, \forall j \, \epsilon \, \{1, \ldots, m\} \, , \|x - O_i\| \leq \|x - O_j\| \right\} \tag{3.4}$$

The lines that compose the boundaries of all these polygons $\mathbb{P}_i$ are called "Voronoi paths". Thanks to the definition 3.4, we can easily see that each Voronoi path is at equal distance of the two obstacles that lie on both side of this Voronoi path.

This is why we can say that Voronoi paths are the safest paths for avoiding obstacles, and this is why I chose this method of guidance. Moreover, Voronoi paths are just basic lines that can be easily followed by the sailboat thanks to the Line-Following algorithm.

We are now going to see the algorithm I chose to use in order to compute Voronoi Diagrams, but note that other algorithms that are able to compute Voronoi Diagrams exist, based on other mathematical concepts.

Computation of Voronoi Diagrams

Algorithm 3.2 Voronoi Diagrams Algorithm

Input : $L_O = \{O_1\,,\,\ldots\,,\,O_m\}$

```
1  V_T = ∅
2  L_VP = ∅
3  for O_i in L_O
4     for O_j in L_O
5        for O_k in L_O
6           T = (O_i, O_j, O_k)
7           C = circumscribed circle (T)
8           Bool = False
9           for O_x in L_O
10             if  (O_x is inside C)  AND  (O_x ≠ O_i  AND  O_x ≠ O_j  AND  O_x ≠ O_k)
11                Bool = True
12             end if
13          end for
14          if  Bool = True
15             discard T
16          else
17             push T into V_T
18          end if
19       end for
20    end for
21 end for
22 for T_u in V_T
23    for T_v in V_T
24       if  (T_u and T_v have exactly two vertices in common)
25          C_u = center of the circumscribed circle (T_u)
26          C_v = center of the circumscribed circle (T_v)
27          push  (C_u ; C_v)  into L_VP
28       end if
29    end for
30 end for

return L_VP
```

The input of this algorithm is the list L_O of all the obstacles $O_1\,,\,\ldots\,,\,O_m$ that lie on the map (or the local plane).

We define two other lists in lines 1 and 2 : the list V_T of all "valid triangles" and the list L_{VP} of all the "Voronoi Paths".

From lines 3 to 6, the Voronoi Diagrams algorithm goes through the list L_O (thanks to three for loops), in order to go through all the possible triangle T that can be formed by the obstacles O_1 , ... , O_m.

In line 7, the algorithm computes the circumscribed circle C of the selected triangle T (formed by the obstacles O_i, O_j and O_k). See Annex 1 in order to have more mathematical explanations about the computation of the circumscribed circle C.

From lines 8 to 13, the algorithm checks if any obstacles are inside the circle C, apart from the three vertices O_i, O_j and $O_k$of the triangle T. If one or more obstacles (appart from O_i, O_j and O_k) are inside the circle C, the variable $Bool$ is set to $True$; if none of the obstacles (appart from O_i, O_j and O_k) are inside the circle C, the value of the variable $Bool$ stays at $False$. See Annex 1 in order to have more mathematical explanations about how to check if an obstacle is inside C.

From lines 14 to 18, the algorithm discard or delete the triangle T if $Bool$ is set to $True$ (if at least one obstacle is inside C), or keep T in memory if $Bool$ is set to $False$, by storing it inside the list of all "valid triangles" V_T.

From lines 22 to 30, the algorithm goes through the list of all "valid triangles" V_T, figures out those which are side by side (those which share exactly two vertices), and stores the centers of their two respective circumscribed circles in the list of all the "Voronoi Paths" L_{VP}. Indeed, these two centers are going to form together a Voronoi Path.

Finally, the output of this algorithm is L_{VP}, the list of the points that formed all Voronoi Paths.

In a Voronoi diagram, the circle circumscribed to each triangle does not enclose any other obstacles than the vertices of its triangle.

3.3.3 Anchoring Algorithm

In order to make a robust simulation, I created the Anchoring algorithm. This algorithm is the most important part of the simulation because it allows to ensure the safety of the navigation and to provide the effectiveness of the simulation.

Principle of the Anchoring algorithm

As explained in the section 3.3.1, the Anchoring algorithm allows the autonomous sailboat to choose the best line to follow in order to achieve the two objectives of the simulation : stay in the anchoring zone and avoid obstacles.

Indeed the Anchoring algorithm is going to receive the coordinates of the four points that form the rectangular anchoring zone, and the list of all the Voronoi pahs computed by the Voronoi Diagrams algorithm. Then the algorithm will choose the starting point and the ending

point of the line that must be followed by the sailboat, and will transmit this line to the Line-Following algorithm.

The Figure 3.6 shows the way the Anchoring algorithm works.

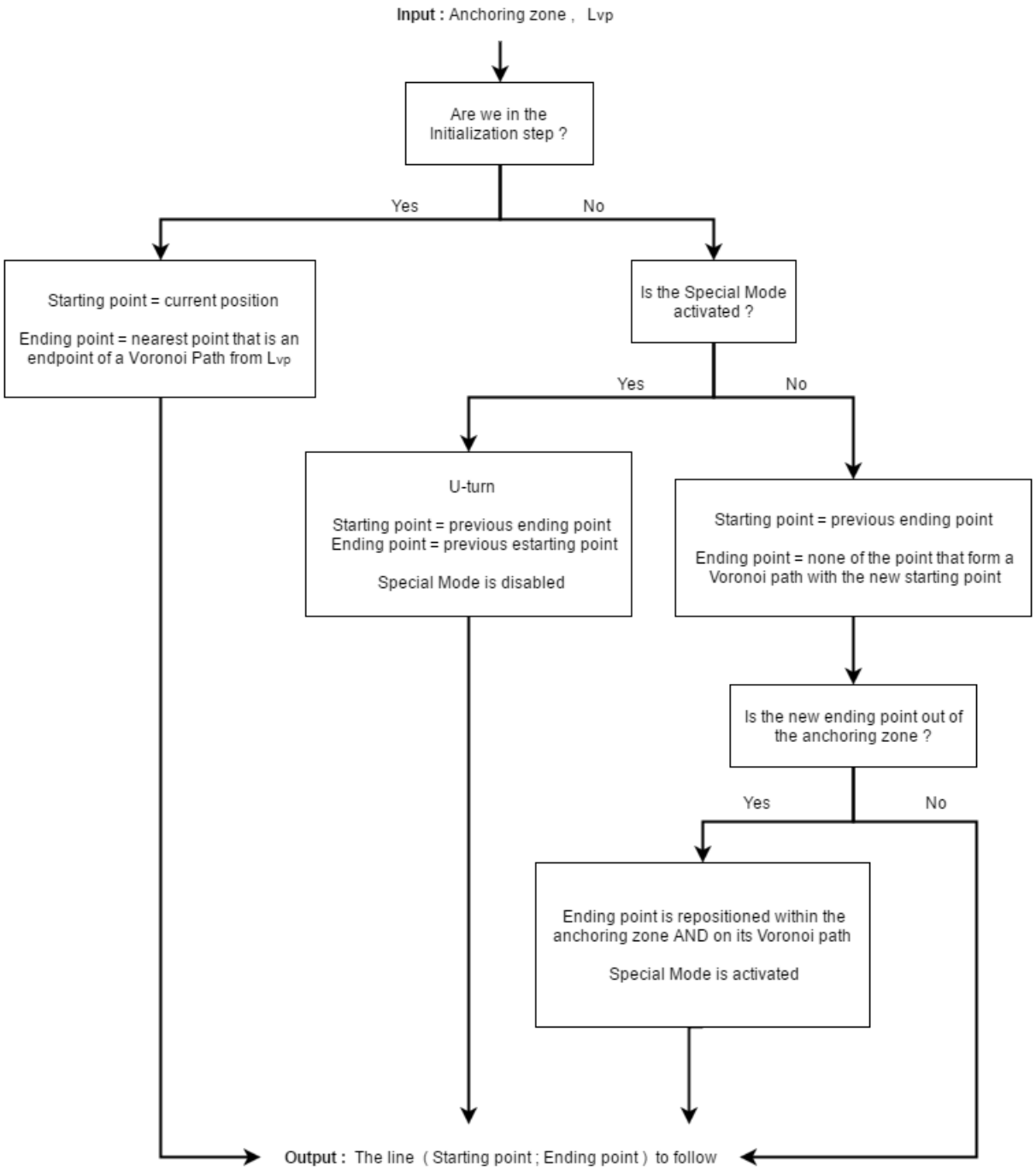


Figure 3.6: The Anchoring algorithm

The Anchoring algorithm has two inputs : the rectangular Anchoring zone (choosen by the user of the simulation) and the list of all the Voronoi Paths L_{VP} (computed by the Voronoi Diagrams algorithm). Moreover, the output of the algorithm is the line that is going to be followed by the sailboat and which is represented by a "starting point" and an "ending point".

Firstly, the algorithm checks if the simulation is in its "Initialization" step : when the simulation has just begun and when the sailboat is in its initial position and could be anywhere on the map. If the simulation is in this Initialization step, the output of the Anchoring algorithm is the line with the initial position of the sailboat as its starting point, and which has its ending point represented by the nearest point to the sailboat that is also an endpoint of a Voronoi Path stored in L_{VP}.

Then, the algorithm checks if the "Special Mode" is activated. This mode is initially disabled, but it will be activated further in the simulation, when the sailboat will be about to leave the Anchoring zone. Indeed, when the Special Mode is activated, the algorithm allows the sailboat to make a U-turn by reversing the starting point and the ending point of the previous line that has been followed. The sailboat will follow the same line, but in the opposite direction. Thanks to this Special mode, the autonomous sailboat will never leave the Anchoring zone. The Special mode is then immediately disabled and will be activated again, each time the sailboat is about to leave the Anchoring zone.

If the Special Mode is disabled and if the simulation is not in its Initialization step, it means that the sailboat has finished to follow a Voronoi Path stored in L_{VP} and is now exactly on the corresponding ending point. In this case, the algorithm will basically gives a new line to follow, by taking the ending point of the previous Voronoi Path as the "new starting point", and by defining the "new ending point" thanks to Voronoi Paths stored in L_{VP}. Indeed, the algorithm will search in L_{VP} a Voronoi Path which has the "new starting point" as one of its endpoints, and will take the other endpoint of this Voronoi Path as the "new ending point".

Finally, if the "new ending point" that has been selected by the Anchoring zone is out of the Anchoring zone, the "new ending point" is repositioned within the Anchoring zone, in order to be still placed on the corresponding Voronoi Path. In order to have further explanations about this repositionning, see Annex 3. Moreover, the Special Mode is activated for the next call of the Anchoring algorithm, in prevision of the fact that the sailboat is going to reach the "new ending point" and will then have to make a U-turn in order to stay within the Anchoring zone.

Illustrative Example

The Figure 3.7 shows an illustrative example of the Anchoring algorithm. The sailboat is represented in red. O_1, O_2, O_3 and O_4 (in black) are the obstacles that must be avoided by the sailboat. The blue rectangle is the anchoring zone in which the sailboat must stay. Green lines represent the Voronoi paths that have been generated by the Voronoi Diagrams algorithm in order to avoid the obstacles O_i. A, B, C, D, E and F are the endpoints of Voronoi paths.

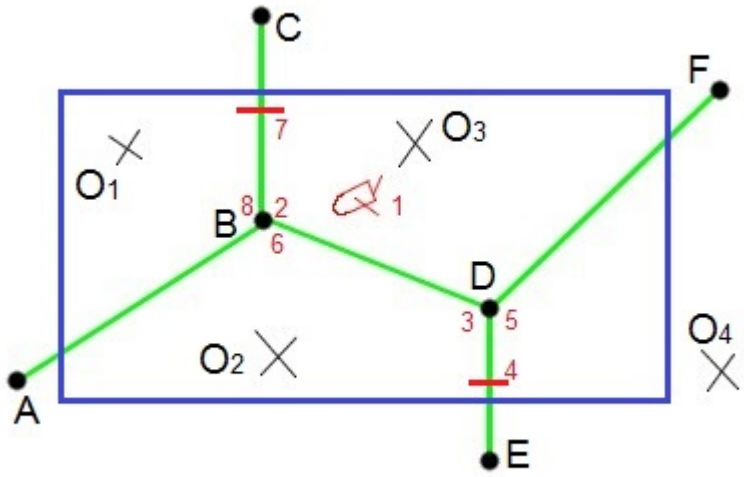


Figure 3.7: Illustrative example of the Anchoring algorithm

The Figure 3.7 shows the path followed by the autonomous sailboat :

- *1 :* The sailboat is in a given initial position. The Anchoring algorithm is in its Initialization step, so the sailboat is going to follow the line formed by its initial position (starting point) and the point B (ending point), since the point B is the nearest point that is an endpoint of a Voronoi Path.

- *2 :* Once the sailboat has reached the point B, the Anchoring algorithm is going to select a Voronoi Path to follow. The special mode is not activated, so the algorithm has to choose one of the three Voronoi Path that are connected to the point B. The point D is selected, so the new starting point of the line to follow is the point B and its ending point is the point D, which is inside the anchoring zone.

- *3 :* Once the sailboat has reached the point D, the Anchoring algorithm has to select a new Voronoi Path to follow, as in the previous step. The algorithm choose the point E, which is not within the anchoring zone : the special mode is activated and the ending point of the new line to follow is repositioned.

- *4 :* The sailboat has reached the ending point of the line, which is a new point (different from the point E) that has been placed within the Anchoring zone and on the same Voronoi Path as the point E (see Annex 3). Since the special mode is activated, the sailboat has to make a U-turn and to go again on the point D (which was the previous starting point, in the step 3). The special mode is disabled.

- *5 :* Once the point D is reached, and since the special mode has been disabled, the algorithm choose the point B has the ending point of the new line to follow. The point B is within the anchoring zone, so the special mode is not activated and no repositioning is required.

- *6 :* Once the point B is reached, the Anchoring algorithm choose the point C as the new ending point. As in the step 3, this point is outside the anchoring zone, so the ending point of

the new line to follow is repositioned within the anchoring zone.

- *7 :* As in the step 4, the sailboat as to make a U-turn and to return to the point B.

- *8 :* When the sailboat reach the point B, the algorithm has to choose a new point from a Voronoi Path, and so on.

The Anchoring algorithm will always generated lines that allows the autonomous sailboat to stay in the anchoring zone, while avoiding obstacles.

Results of the simulation

Results of the simulation can be seen in the Annex 4.

Chapter 4

Conclusion

Firstly, the feasibility study showed me a deeper look into some sensors, such as the eCompass for example, and all the calibration methods that I learned during my internship coulb be reusable during my future projects.

Secondly, the "simulation part" of my internship made me discover what "working in the Research" really means. Indeed, I discovered in what consists the work of a researcher. Firstly, I began by reading a lot of scientific publications that were on the same subject than my simulation. Then, I selected what seems to be the best tool in order to reach the goal of my simulation : Voronoi diagrams, which allows to face any configurations of obstacles to avoid, and the Line-Following algorithm which allows to always follow lines efficiently, even when the sailboat has to face the wind. Finally, I even tried to created a new algorithm in order to connect these two tools.

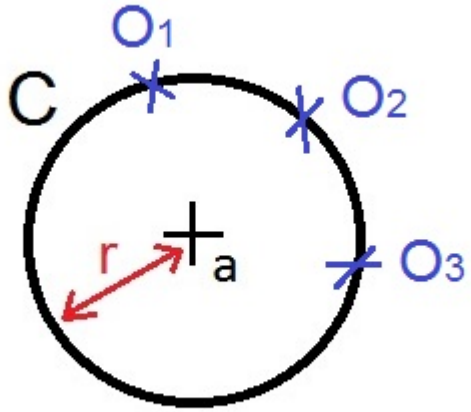
For these reasons, I really liked this internship in the University of Plymouth, and it gives me the desire to work as a researcher after my studies.

Chapter 5

Annex

Annex 1 : Mathematical Details of the Voronoi Diagram Algorithm

How to compute the circle circumscribed to three points ?



We want to compute the center a and the radius r of the circle C circumscribed to three points O_1, O_2 and O_3 (which could be the three vertices of a triangle).

with $a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$ and $O_i = \begin{pmatrix} O_i^1 \\ O_i^2 \end{pmatrix}$, $\forall i \, \epsilon \, \{1, 2, 3\}$.

We have :

$$\forall i \, \epsilon \, \{1, 2, 3\} \, , \quad \|O_i - a\|^2 = r^2 \tag{5.1}$$

Thanks to the polarization identity ($\|x - y\|^2 = \|x\|^2 + \|y\|^2 - 2 < x \, , \, y >$), we have :

$$\forall i \, \epsilon \, \{1, 2, 3\} \, , \quad \|O_i\|^2 + \|a\|^2 - 2 < O_i \, , \, a > -r^2 = 0 \tag{5.2}$$

By defining the variable $a3 = \|a\|^2 - r^2$, we have :

$$\forall i \,\epsilon\, \{1,2,3\}\,, \quad 2 < O_i\,,\, a > -a_3 = \|O_i\|^2 \tag{5.3}$$

Knowing that $< O_i\,,\, a >= O_i^T a$, we can write the equation 5.3 in a matrix format :

$$\forall i \,\epsilon\, \{1,2,3\}\,, \quad \begin{pmatrix} 2O_i^1 & 2O_i^2 & -1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \|O_i\|^2 \tag{5.4}$$

By bringing the three equations (for i = 1, 2 and 3) together, we have :

$$\begin{pmatrix} 2O_1^1 & 2O_1^2 & -1 \\ 2O_2^1 & 2O_2^2 & -1 \\ 2O_3^1 & 2O_3^2 & -1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \|O_1\|^2 \\ \|O_2\|^2 \\ \|O_3\|^2 \end{pmatrix} \tag{5.5}$$

By defining $M = \begin{pmatrix} 2O_1^1 & 2O_1^2 & -1 \\ 2O_2^1 & 2O_2^2 & -1 \\ 2O_3^1 & 2O_3^2 & -1 \end{pmatrix}$, $A' = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$ and $O' = \begin{pmatrix} \|O_1\|^2 \\ \|O_2\|^2 \\ \|O_3\|^2 \end{pmatrix}$, we have :

$$A' = M^{-1}.O' \tag{5.6}$$

which allows us to compute a_1, a_2 and a_3.

Moreover, thanks to the definition of a_3, we have :

$$r = \sqrt{\|a\|^2 - a_3} \tag{5.7}$$

We have managed to compute the center $a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$ and the radius r of the circle C circumscribed to the points O_1, O_2 and O_3.

How to verify if an obstacle is inside the circumscribed circle ?

In order to know if an obstacle O_i is inside a circumscribed circle C, which has a center a and a radius r, we just have to check if :

$$\|O_i - a\| \leq r \tag{5.8}$$

If the inequation 5.8 is verified, the obstacle O_i is inside the circle C.

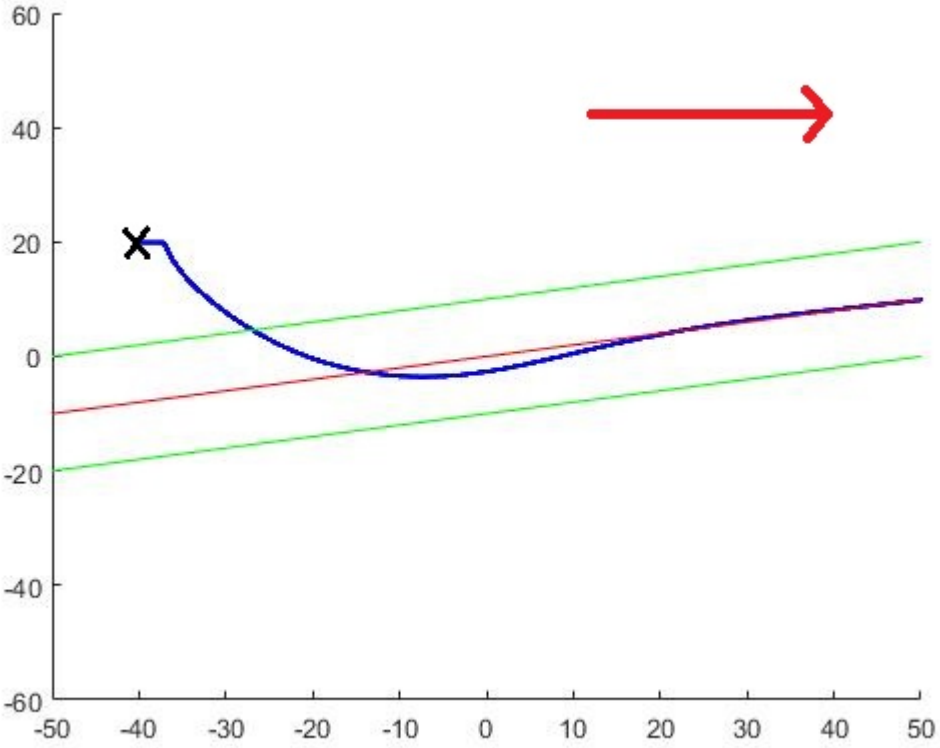
Annex 2 : Results of the First Simulation

Result 1

In the following figure, the black cross represent the initial position of the autonomous sailboat, and the blue curve is the trajectory of its centre of gravity.

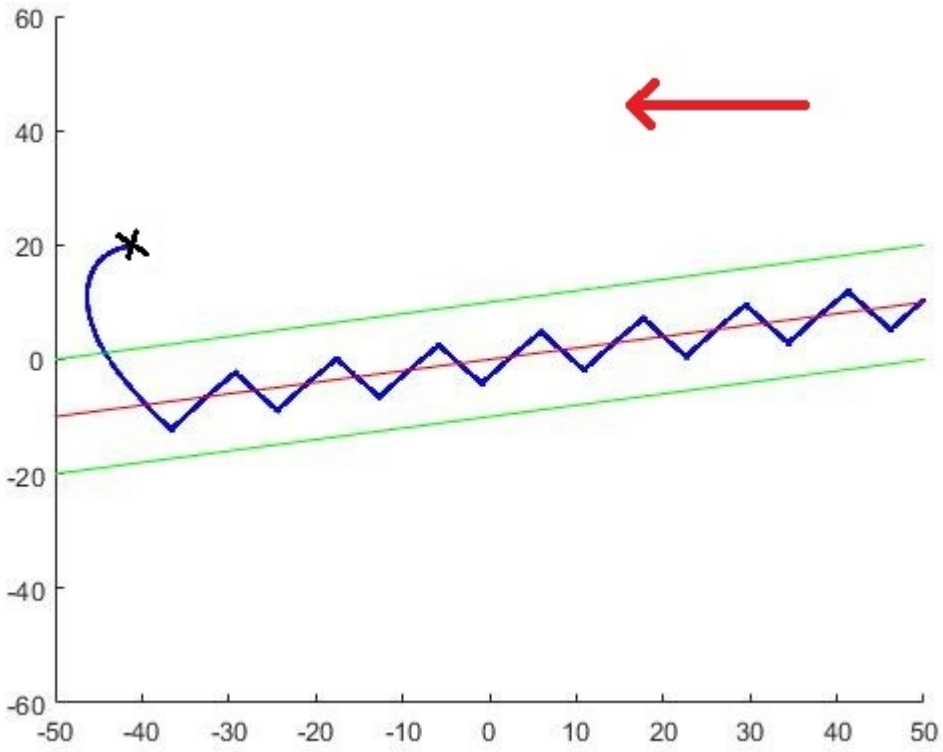
The red line is the line that must be followed by the sailboat, and the two green lines represent the “safety corridor”. The line must be followed from the left to the right.

The red arrow represent the direction of the wind.



We can see that the sailboat manages to follow the line. It is easy to do in this situation, since the sailboat has to navigate toward the same direction than the wind.

Result 2



In this situation, the sailboat has to navigate against the wind.

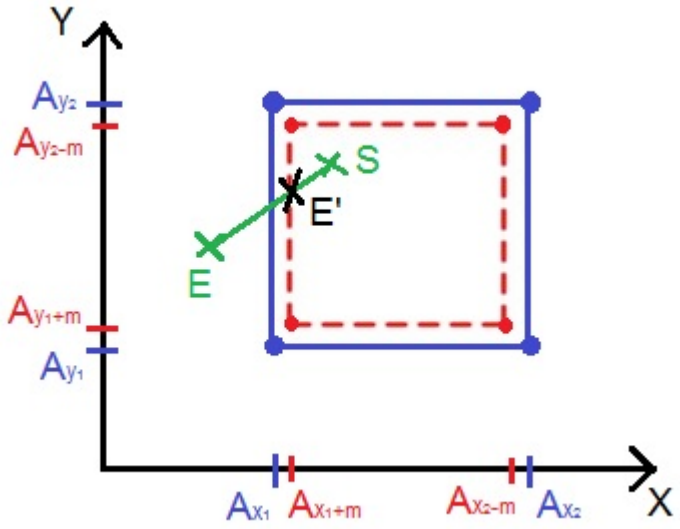
We can see that it still manages to follow the line, but he has to navigate in a “close hauled” configuration.

Annex 3 : Mathematical Details of the Anchoring Algorithm

When the autonomous sailboat is following a Voronoi Path, the ending point of this line can lie outside the anchoring zone. In this situation, the Anchoring algorithm has to repositionned the ending point of the line to follow inside the anchoring zone.

4 cases are possible.

Case 1 : the ending point is at the left of the anchoring zone



The blue rectangle is the anchoring zone in which the sailboat must stay. Its dimensions are : $(A_{x2} - A_{x1}) \times (A_{y2} - A_{y1})$.

The red dotted rectangle corresponds to the "safety margin". It is a rectangle that is at a distance m of all the edges of the anchoring zone. Indeed, in order to provide more security to the navigation, ending points are not repositionned exactly on the boundary of the anchoring zone, but at a distance m of this boundary. This safety distance m can be adjusted by the user of the simulation.

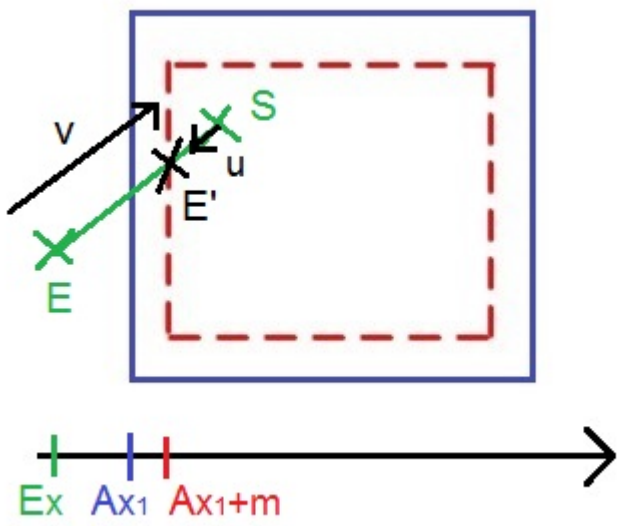
The green line is the line that must be followed by the sailboat.

The point **S** is the starting point of the line to follow. Its coordinates are $\begin{pmatrix} S_x \\ S_y \end{pmatrix}$.

The point **E** is the ending point of the line to follow, which is outside the anchoring zone, at the left side. Its coordinates are $\begin{pmatrix} E_x \\ E_y \end{pmatrix}$.

The point **E′** is the new ending point of the line to follow, which has been repositionned by the Anchoring algorithm inside the anchoring zone, at a distance m of its left boundary. Its

coordinates are $\begin{pmatrix} E'_x \\ E'_y \end{pmatrix}$.



As shown on the Figure, **u** is the unitary vector of the line to follow ;

$$\mathbf{u} = \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \frac{\mathbf{E}-\mathbf{S}}{\|\mathbf{E}-\mathbf{S}\|} = \frac{1}{\sqrt{(E_x-S_x)^2+(E_y-S_y)^2}} \cdot \left(\begin{pmatrix} E_x \\ E_y \end{pmatrix} - \begin{pmatrix} S_x \\ S_y \end{pmatrix} \right) \tag{5.9}$$

We want to find the corrector vector **v** such as :

$$\mathbf{E} + \mathbf{v} = \mathbf{E}' \tag{5.10}$$

with $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$.

We know that **v** is proportional to **u** :

$$\mathbf{v} = k \cdot \mathbf{u} \tag{5.11}$$

where k is a real number.

So thanks to the equations 5.10 and 5.11, we can write :

$$E'_x = E_x + v_x \tag{5.12}$$

$$E'_x = E_x + k \cdot u_x \tag{5.13}$$

$$k = \frac{E'_x - E_x}{u_x} \tag{5.14}$$

Moreover, since $\mathbf{E}'$ has been repositionned at a distance m of the left boundary of the anchoring zone (see Figure), we know that

$$E'_x = A_{x1} + m \tag{5.15}$$

So, thanks to the equations 5.14 and 5.15, we have

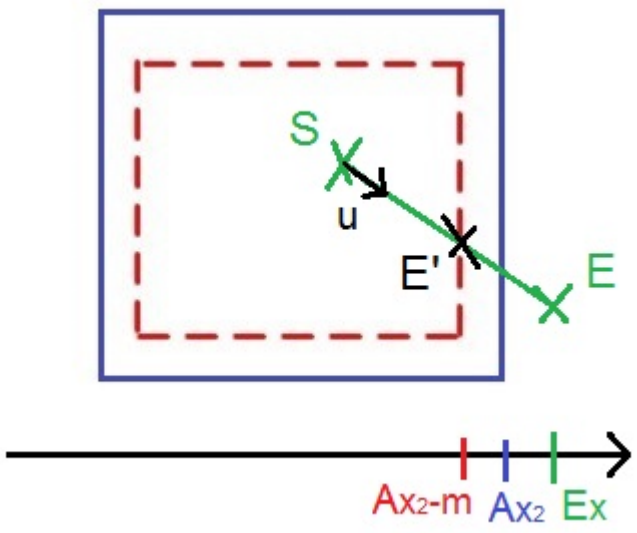
$$k = \frac{A_{x1} + m - E_x}{u_x} \tag{5.16}$$

Finally, thanks to the equations 5.10, 5.11 and 5.16, we have found the coordinates of the new ending point $\mathbf{E}'$:

$$E'_x = E_x + \frac{A_{x1} + m - E_x}{u_x} \cdot u_x \tag{5.17}$$

$$E'_y = E_y + \frac{A_{x1} + m - E_x}{u_x} \cdot u_y \tag{5.18}$$

Case 2 : the ending point is at the right of the anchoring zone

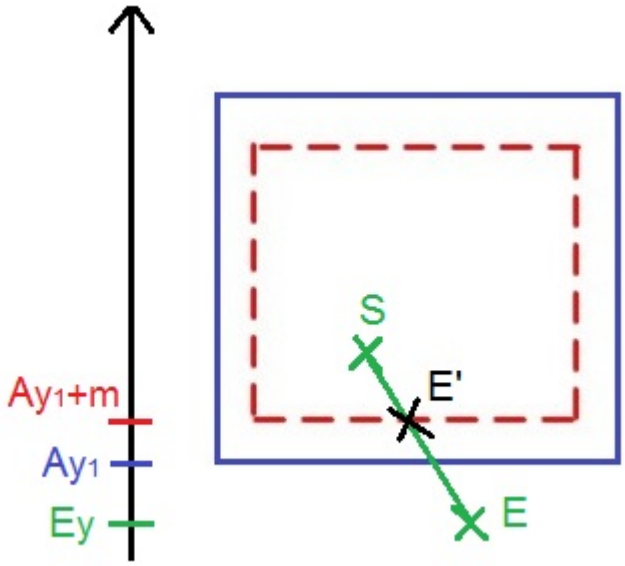


$$\mathbf{u} = \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \frac{\mathbf{E} - \mathbf{S}}{\|\mathbf{E} - \mathbf{S}\|} = \frac{1}{\sqrt{(E_x - S_x)^2 + (E_y - S_y)^2}} \cdot \left(\begin{pmatrix} E_x \\ E_y \end{pmatrix} - \begin{pmatrix} S_x \\ S_y \end{pmatrix} \right) \tag{5.19}$$

$$E'_x = E_x - \frac{E_x - (A_{x2} - m)}{u_x} \cdot u_x \tag{5.20}$$

$$E'_y = E_y - \frac{E_x - (A_{x2} - m)}{u_x} \cdot u_y \tag{5.21}$$

Case 3 : the ending point is below the anchoring zone

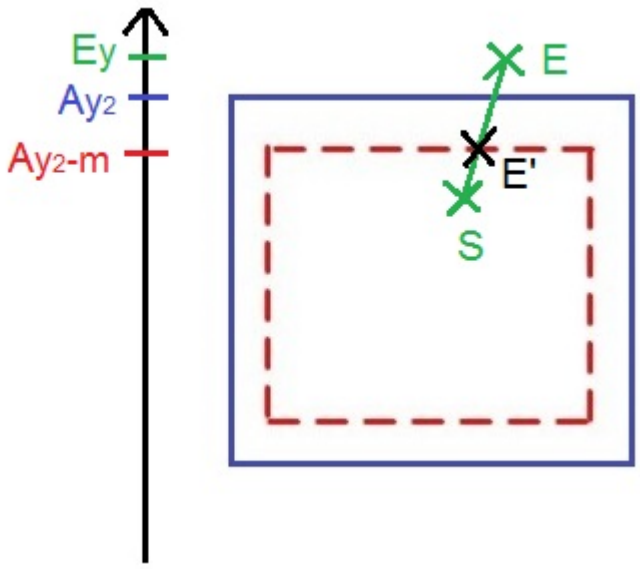


$$\mathbf{u} = \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \frac{\mathbf{E} - \mathbf{S}}{\|\mathbf{E} - \mathbf{S}\|} = \frac{1}{\sqrt{(E_x - S_x)^2 + (E_y - S_y)^2}} \cdot \left(\begin{pmatrix} E_x \\ E_y \end{pmatrix} - \begin{pmatrix} S_x \\ S_y \end{pmatrix} \right) \tag{5.22}$$

$$E'_x = E_x + \frac{A_{y1} + m - E_y}{u_y} \cdot u_x \tag{5.23}$$

$$E'_y = E_y + \frac{A_{x1} + m - E_x}{u_y} \cdot u_y \tag{5.24}$$

Case 4 : the ending point is above the anchoring zone



$$\mathbf{u} = \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \frac{\mathbf{E} - \mathbf{S}}{\|\mathbf{E} - \mathbf{S}\|} = \frac{1}{\sqrt{(E_x - S_x)^2 + (E_y - S_y)^2}} \cdot \left(\begin{pmatrix} E_x \\ E_y \end{pmatrix} - \begin{pmatrix} S_x \\ S_y \end{pmatrix} \right) \tag{5.25}$$

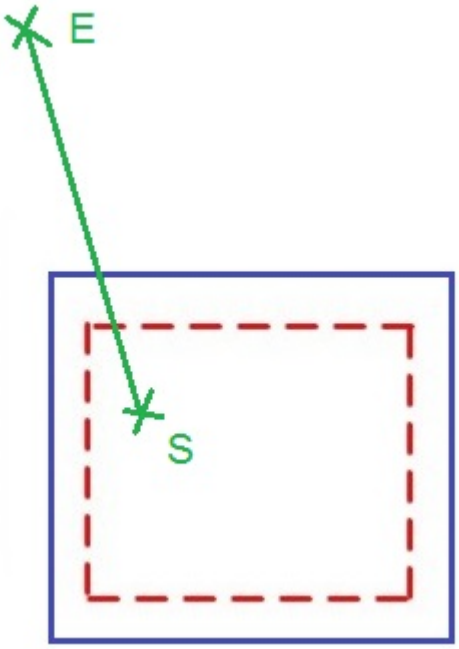
$$E'_x = E_x - \frac{E_y - (A_{y2} - m)}{u_y} \cdot u_x \tag{5.26}$$

$$E'_y = E_y - \frac{E_y - (A_{y2} - m)}{u_y} \cdot u_y \tag{5.27}$$

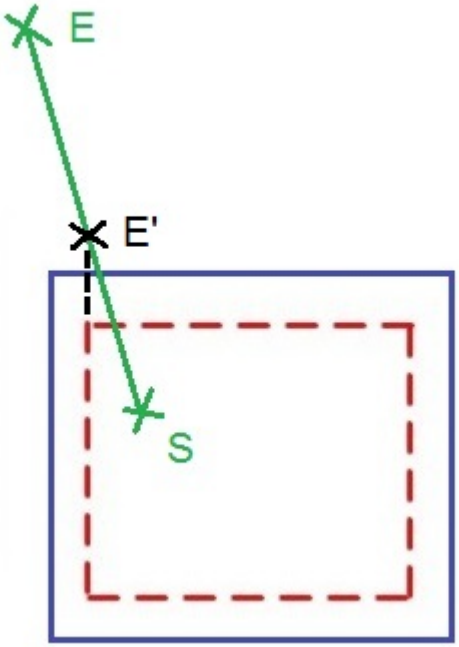
Multi-case

The Anchoring algorithm is able to reposition ending points that are in two different cases at the same time, as shown in the example below :

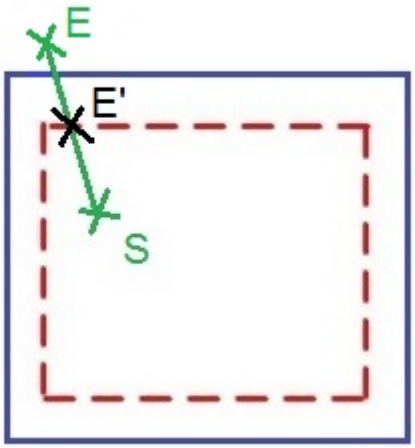
Initial Situation



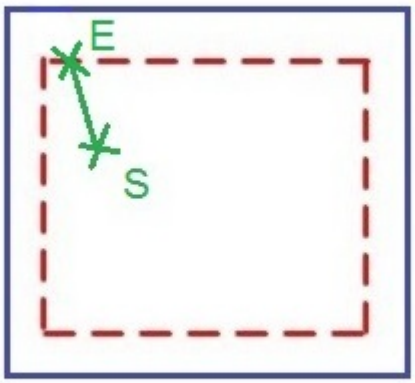
Case 1



Case 4



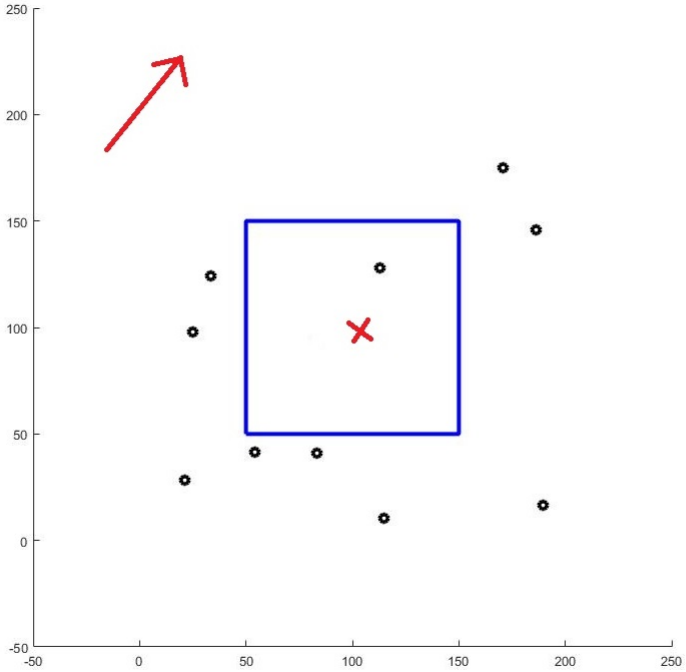
Final Situation



Annex 4 : Results of the Second Simulation

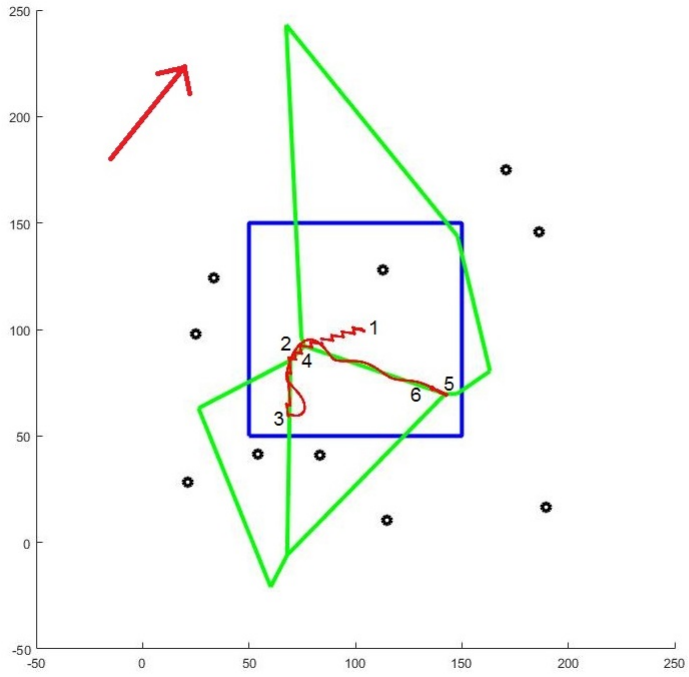
Initial configuration

In the following figure, the blue rectangle is the *anchoring zone*, the black dots are the *obstacles* that must be avoided, the red cross is the *initial position of the autonomous sailboat*, and the red arrow is the *direction of the wind*.



Results of the simulation

The green lines represent the *Voronoi Paths*, the red curve is the *trajectory of the centre of gravity* of the sailboat, and the numbers 1 to 6 is where the sailboat went (chronologically).



We can see that the simulation was a success, since the sailboat stayed inside the anchoring zone, and managed to avoid all obstacles.

Bibliography

[AND10] Analog Devices, *Using an Accelerometer for Inclination Sensing*, Application Note 1057, 2010.

[JAU04] Jaulin L., *Modélisation et commande d'un bateau à voile*, CIFA, Douz (Tunisie), 2004.

[JAU11] Sliwka J., Xiao K., Jaulin L., *A wind-independent control strategy for autonomous sailboats based on Voronoï diagram*, CLAWAR 2011, Paris, 2011.

[JAU12] Jaulin L., Le Bars F., *A simple controller for line following of sailboats*, Proceedings of the 5th International Robotic Sailing Conference, Springer Eds., Cardiff, England, 2012.

[JAU15] Jaulin L., *Mobile robotics*, ISTE WILEY, 2015.

[NXP1] NXP, *Layout Recommendations for PCBs Using a Magnetometer Sensor*, Application Note 4247, 2015.

[NXP2] NXP, *Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference*, Application Note 4246, 2015.

[NXP3] NXP, *Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors*, Application Note 4248, 2015.

[SAU09] Neal M., Sauze C., Thomas B., Alves J., *Technologies for Autonomous Sailing: Wings and Wind Sensors*, Proceedings of the 2nd IRSC, Matosinhos, Portugal, July 6-12, pp. 23–30, 2009.

ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport dès la fin du stage à / *Please return this report at the end of the internship to :*
ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
☎ 00.33 (0) 2.98.34.87.70 - Fax 00.33 (0) 2.98.34.87.90 - stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* Plymouth University

Adresse / *Address* Drake Circus, Devon PL4 8AA UK

Tél / *Phone (with country and area code)* +44 (0) 1752 586137

Fax / *Fax (with country and area code)* +44 (0) 1752 586103

Nom du superviseur / *Name of the person in charge of the placement* Jian Wan

Fonction Lecturer

Adresse e-mail / *E-mail address* jian.wan@plymouth.ac.uk

Nom du stagiaire accueilli / *Name of the trainee* Yoann SOLA

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une cote, en encerclant le chiffre approprié, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please give a mark between **A (very good)** and **F (very weak).***

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? A B C D E F
  Has the task been carried out well ?

- Le stagiaire a-t-il apporté des connaissances nouvelles à l'organisme d'accueil ?
  Did the trainee bring news skills to the host organisation ? ☑ oui/*yes* ☐ non/*no*

  Lesquelles ? *Which ones ?* Sailboat control

- Manquait-il au stagiaire des connaissances ? ☐ oui/yes ☑ non/no
  Was the trainee lacking skills ?

  Si oui, lesquelles ? / *Which ones ?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil / Did the trainee easily integrate into the host organisation?
  (disponible, sérieux, adaptation au travail de groupe)
  (flexible, dedicated, adapts himself (herself) to the team work) A B C D E F

Avez vous des observations ou suggestions à nous faire part / *Do you have any remarks or suggestions to comment ?*_____

_____

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORKS*

Le comportement du stagiaire était-il conforme à vos attentes *(Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances)* ?
Did the trainee come up to expectations (*Punctual, methodical , responsive to management instructions, concerned with quality, concerned about gaining new skills) ?*

A B C D E F

Avez vous des observations ou suggestions à nous faire part / Have you any remarks or suggestions to comment ?_____

_____

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'adaptait vite à de nouvelles situations ? A B C D E F
(Proposer des solutions aux problèmes rencontrés, autonome dans son travail,..)

Did the trainee adapt himself (herself) well to new situations ? A B C D E F
(Suggest solutions to problems met, was independent in his/her job...)

Avez vous des observations ou suggestions à nous faire part / Have you any remarks or suggestions to comment ?_____

_____

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication A B C D E F
Was the trainee open to listening and expressing himself (herself)

Avez vous des observations ou suggestions à nous faire part / Have you any remarks or suggestions to comment ?_____

_____

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : A B C D E F
The technical skills of the trainee were :

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Are you prepared to host another trainee next year ? ☑ oui/*yes* ☐ non/*no*

Fait à Plymouth, UK ,le 01/09/2016 In ,on

Signature Hawan

Merci pour votre coopération
We thank you very much for your cooperation