Internship report

ENSTA Bretagne
2 F. Verny street
29806 BREST Cedex 9
France

Alexandre ARGENTO
alexandre.argento@ensta-bretagne.org

Year 2018-2019
Semester 2

Acknowledgements

I would like to thank my internship tutor Dr Jian Wan for his support and his availability during my work in Plymouth University.

My thanks also go to the researcher Ulysse Vautier who provided me with some useful skills for Arduino coding and helped me dealing with many issues I encountered.

Contents

I. Introduction

In order to complete my second year of studies at ENSTA Bretagne and my international training, I chose to do my internship at the University of Plymouth, in the UK. For 12 weeks, I worked on the development of autonomous sailboats.
This document aims to present my internship as an assistant researcher at the University of Plymouth, from June 10th to August 30th.

Because of my interests in the research work about robotics, I wanted to study a specific area in order to see how I could contribute. The subject of controlling a robot sailboat fits with my training in robotics and practice of coding cards used for such systems. It also presents the specificity of the robot's movement, which, while having a low energy cost, requires a more complex decision in choosing a path.

The research axis, wanted by my internship tutor Jian Wan, was to design an autonomous sailboat with basic moving functions. I had to work, along with other interns, with the goal of making it usable for testing more complex control command, including a fleet control. The "fleet" component involves designing a sailboat so as to be able to reproduce it easily.

Despite the practical oriented aspect of the work of the other interns, I chose to direct my work towards the project's analysis that could help knowing how to use, test or improve it.
In particular, I started with the aggregation of data allowing the establishment of the budget and the equipment used. The coding analysis is about the architecture and the way codes operate in the sailboat. The following sections deal with the system's use once his main code has been set up.

II. Pre-coding discussion

II.1. Project requirements

II.1.a) How autonomous ?

To begin with, the subject of developing an autonomous sailboat might need some clarification about what is called “autonomous”.
In fact, speaking about autonomy is to be understood “at a certain level” because we won’t leave full autonomy to the boats we are sending to the sea. Still, from a user’s point of view who has simple requests, autonomy is the fact of dealing with problems that doesn’t require the user’s attention for solving it. These problems may or may not occur depending on the environment the system is facing.

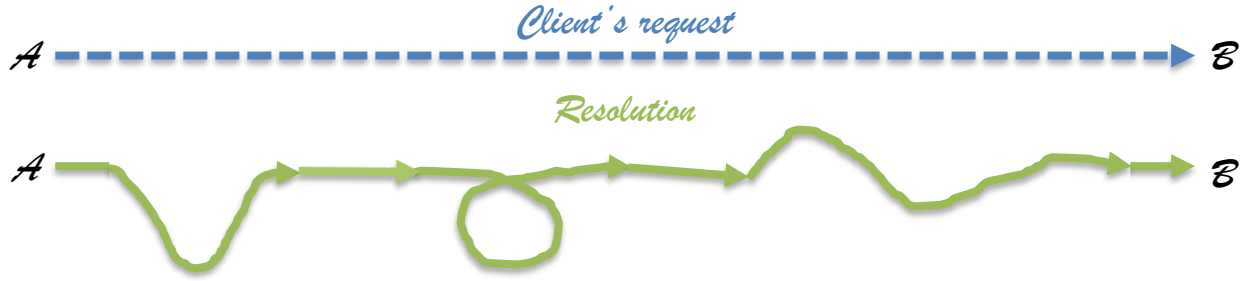


Figure 1 – Autonomy illustration

In other words, autonomy can be considered as the resolution of sub-problems for which the resolution doesn’t need the user’s consultation and has been previously implemented by the designer.
For example, while sailing from a point A to a point B, we might have to:

- adapt according to the wind, which is rather obvious,
- avoid unexpected obstacles,
- deal with internal problems such as the lack of battery or a system crash, …

The identification of such features comes from analyzing the environment in which the robot has to act. This environment generates problems if it contains elements that may slow down or jeopardize the accomplishment of the robot’s task.
These features will be added in a FAST (Function Analysis System Technique) diagram, after considering another utility when we use requirements, which is the work progression.

II.1.b) Work progression assessment

We can assess a work progression by defining goals reachable in terms of condition and performance. The condition is the “if” of “if it works, then we can move on (…)” and the performance is the “how”.
Outside of the requirements of the business world and the industry, rather in an opening field, a research work tend more to reach the “if” than the “how”. In other words, it is better to evaluate progress with conditions rather than performance. One might find wiser sometimes to use both condition and performance for one task in order to divide it.
On the other hand, more in a business context, we might be looking into some specific conditions already fulfilled and improve their performances.

We can now set up a FAST diagram of the project, by dividing the main objective into intermediate tasks, from the left to the right. Features that are related to the autonomy viewpoint are represented in green boxes. Requirements that are detailed in another FAST are represented in purple boxes and technical solutions, in red circles.

Starting with the largest target:

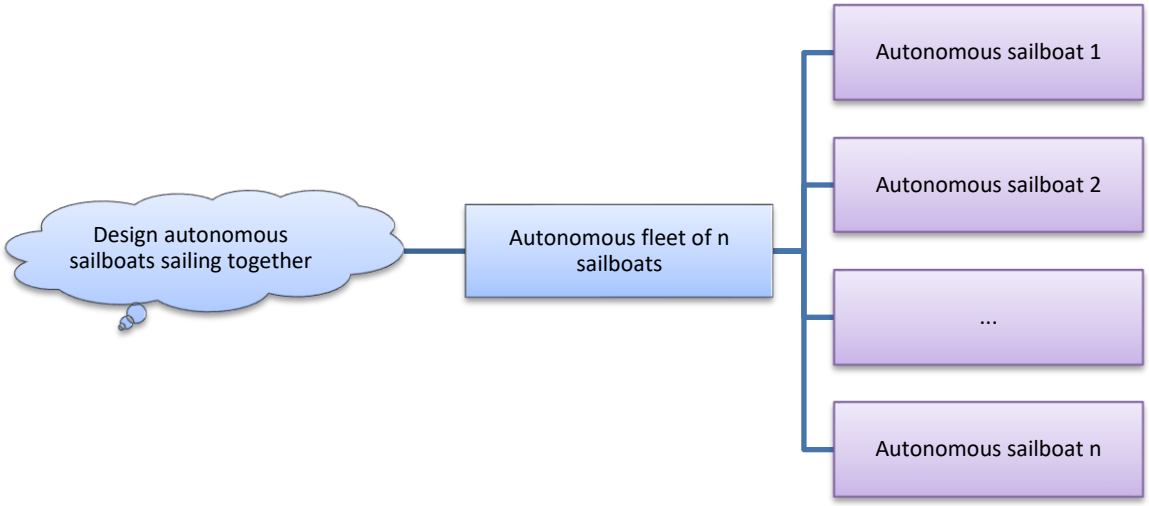


Figure 2 – General FAST diagram

Then, for i in $[\![2, n]\!]$:

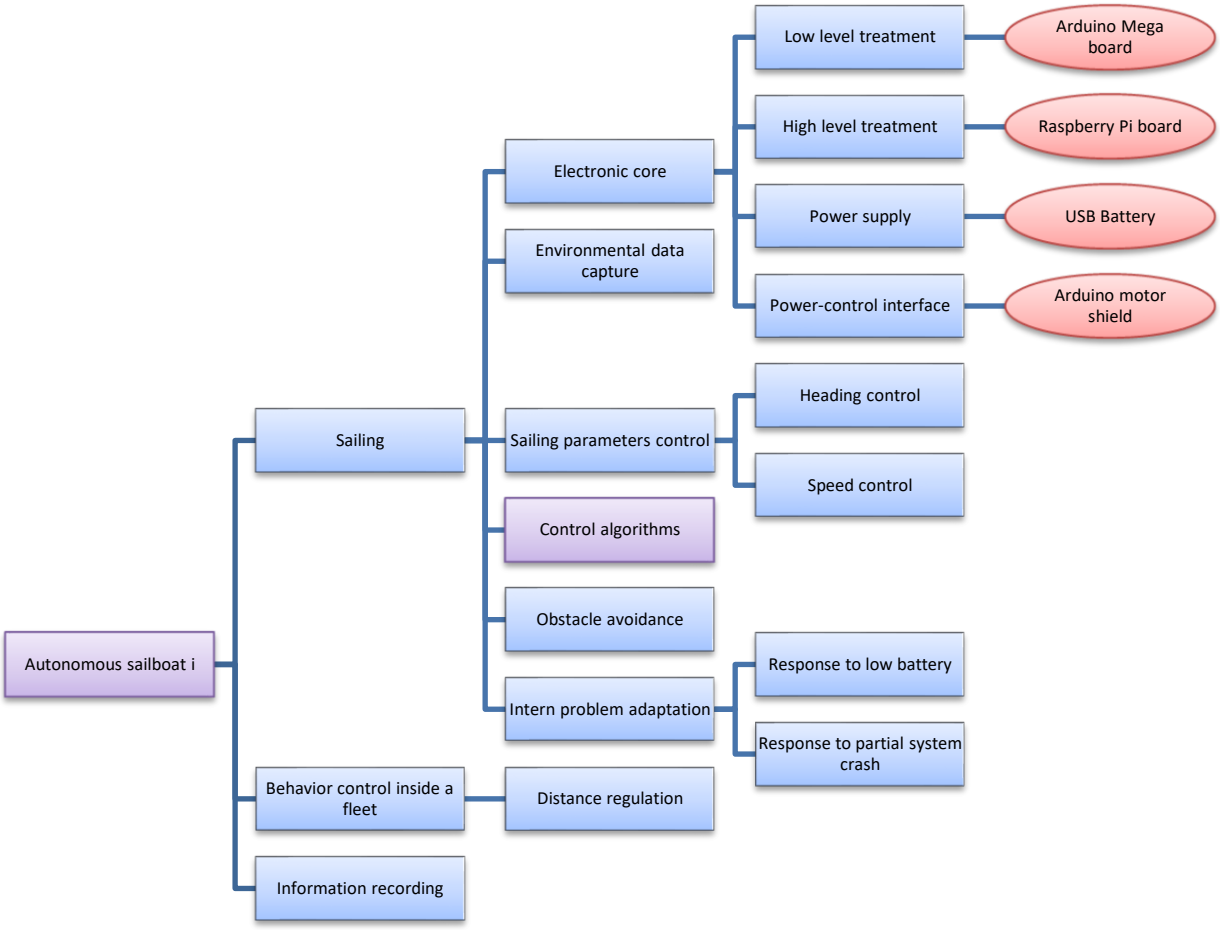


Figure 3 – Sailboat FAST diagram

With, for the detailed control algorithms requirements:

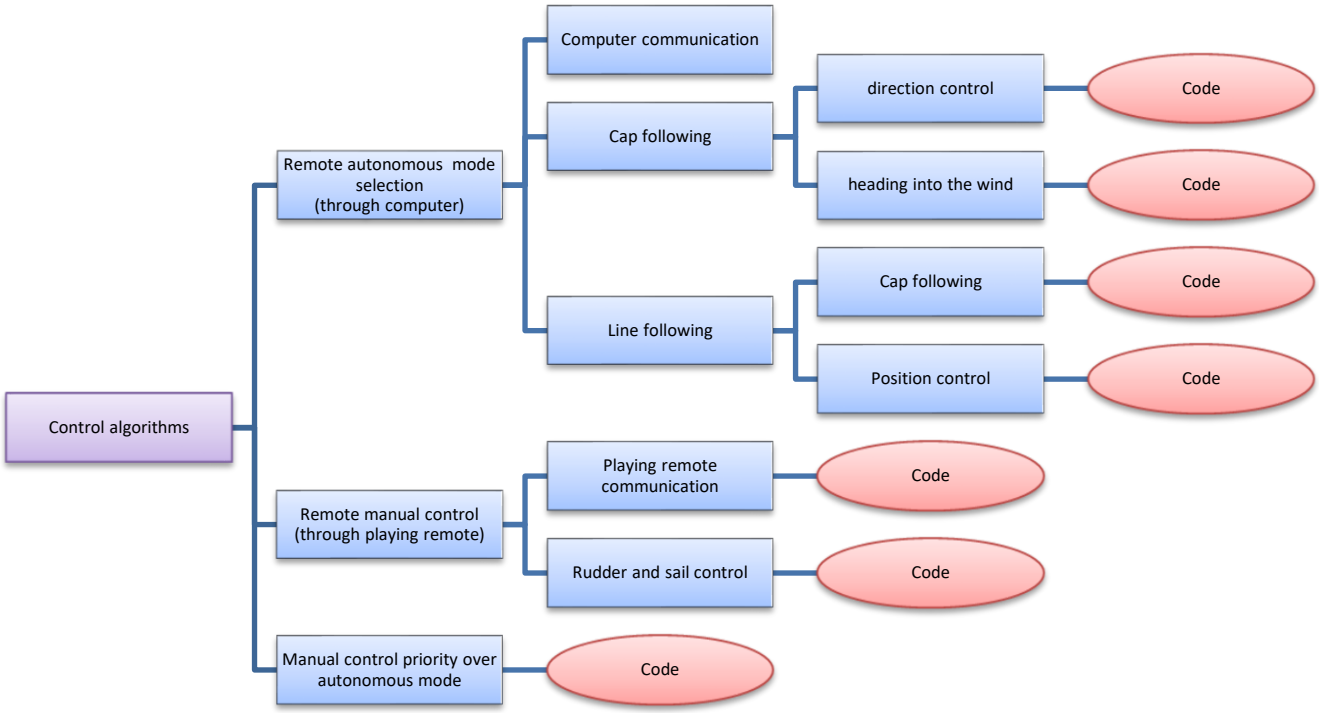


Figure 4 – Control algorithms FAST diagram

The remote manual control is a security measure also convenient for testing, the idea is to be able to regain control of the sailboat if there is a problem with its autonomous behavior. Hence the manual control priority over autonomous mode, to be coded as well.
We will not treat the "Computer communication" requirement in this report, it is part of the work of Matthieu Bouveron, another trainee from ENSTA Bretagne.

We haven't yet completed the Sailboat FAST diagram with all the technical solutions to our requirements. In order to do so, we need to analyze the inputs needed for our sailboat system in order to fulfill its sailing function.

II.2. Environment and existing technology analysis

II.2.a) Use of sensors

We consider in the sailboat's environment all physical signals that can be collected through sensors and that can be significant for its sailing behavior.
Such sailing behavior can be defined by the state vector $\vec{X}$, which contains all parameters that characterize the sailboat's state. This state vector is important for giving any information for a specific controller and can be used in order to implement a Kalman filter, tool used for helping the sailboat locate itself.
Our state vector here will be the following:

$$\vec{X} = \begin{pmatrix} x \\ y \\ \theta \\ v \\ \omega \end{pmatrix}$$

Where x and y are the coordinates, either in a 2-dimensional plane, if we can consider the earth flat in the sailing region, or as latitude and longitude. θ is the angle corresponding to the ship's heading relative to a fixed reference, v is its velocity and ω, its rotation speed.

Sensors can be used in order to measure the state vector's components and other information needed such as the presence of obstacles and their distance or the apparent wind for path planning. These sensors are presented in the table below:

Characteristic	Symbol	Sensor
Position	(x, y)	GPS
Heading	θ	IMU (Inertial Measurement Unit)
Velocity	v	(no direct measurement)
Rotation speed	ω	IMU
Wind direction (apparent wind)	ψ_{ap}	Weathercock
Wind force (apparent wind)	a_{ap}	Anemometer
Presence of obstacles or fleet sailboats		Camera

Table 1 – Sensors for measurements needed

Knowing the wind direction and strength is not only crucial to know how to sail against the wind, but gives information about the sailboat's behavior and how its state vector is affected.
The equations that characterize the evolution of the state vector are the following:

$$\begin{cases} \dot{x} = v\cos\theta + p_1 a\cos\psi \\ \dot{y} = v\sin\theta + p_1 a\sin\psi \\ \dot{\theta} = \omega \\ \dot{v} = \dfrac{f_r\sin\delta_s - f_r\sin u_1 - p_2 v^2}{p_9} \\ \dot{\omega} = \dfrac{f_s(p_6 - p_7\cos\delta_s) - p_8 f_r\cos u_1 - p_3\omega v}{p_{10}} \\ \delta_s = F_1(a,\psi,\theta,v,u_2) \\ f_s = F_2(v,u_1) \\ f_r = F_3(a,\psi,\theta,v,u_2) \end{cases}$$

u_1 and u_2 are respectively the rudder angle and the mainsheet length that regulates the sail's opening.
The p_i are coefficients related to the mechanical characteristics of the sailboat, f_s is the force of the wind on the sail, f_r is the force of the water on the rudder,
ψ is the true wind angle, which is defined here because of the difference between the apparent wind on the boat ψ_{ap} and the wind felt when we are not moving (ψ). A relation $\psi = F_4(\psi_{ap}, a, \theta, v)$ allows us to find ψ from available or measurable data. Likewise, a is the true wind speed, obtained from ψ_{ap}, θ, v and the apparent wind speed a_{ap}.
Without detailing the functions F_1 , F_2 and F_3, we know that measuring the apparent wind speed and direction will not only allow us to decide how to follow a cap, but also help localizing the sailboat.

We can note that the IMU sensor uses a magnet to measure the sailboat's heading. It will require a regular calibration phase before

II.2.b) FAST

From here, we can complete the Sailboat FAST diagram with appropriate technical solutions:

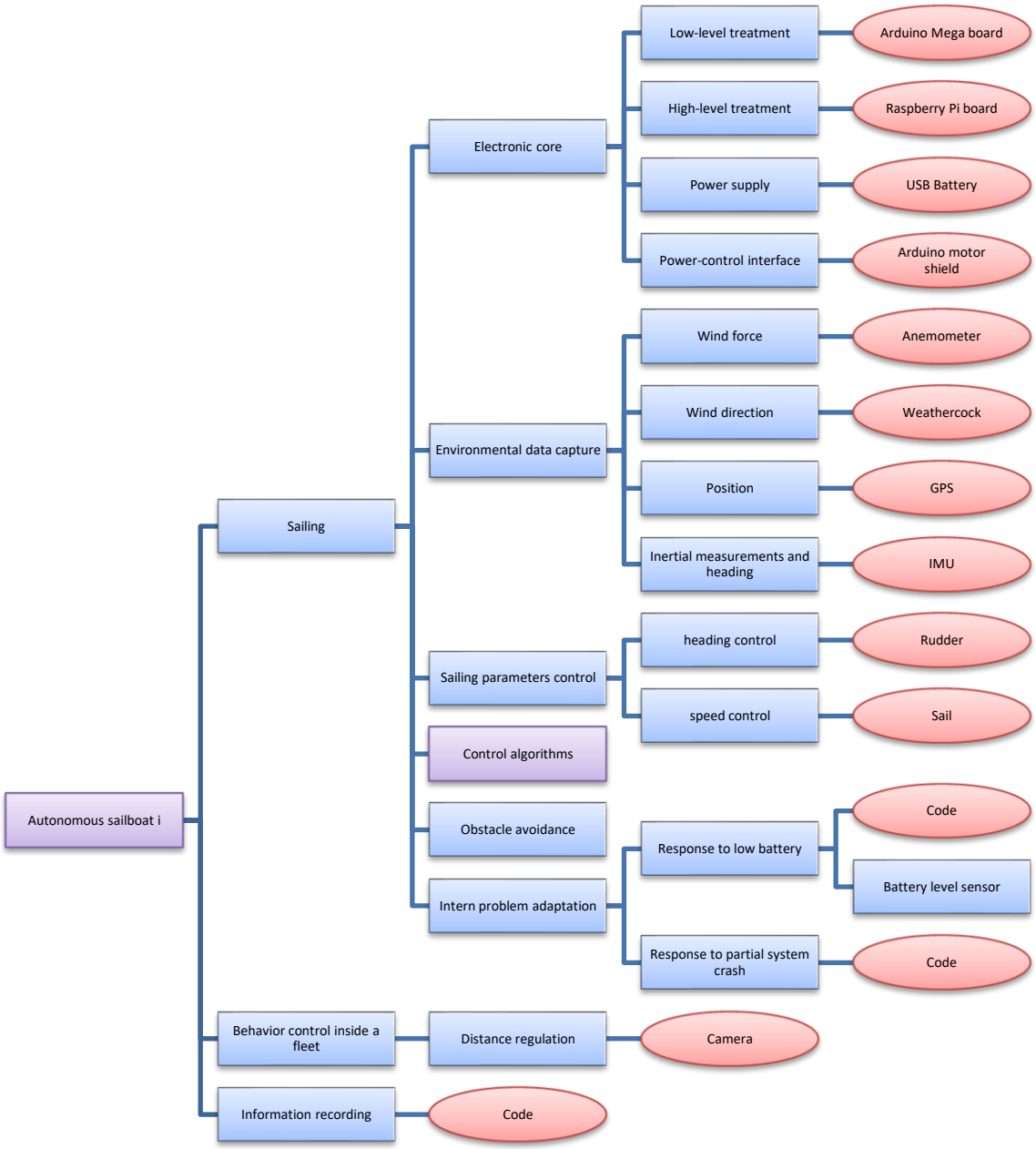


Figure 5 – Final sailboat FAST diagram

The battery level sensor was a requirement I left aside because I didn't find any easy solution for measuring the level of an USB battery.

II.3. Work tools

II.3.a) Code organization

The choice of using an Arduino board connected to a Raspberry Pi board has already been made the past year by a researcher in the University of Plymouth and also other trainees. On the one hand, the Arduino board has a low-level library to get sensor data the most effective way. On the other hand, the Raspberry Pi board allows us to have more computing power in order to use a middleware, ROS (Robot Operating System), and code more advanced control laws with multithreading.

Communication between Arduino and Raspberry Pi can be done with the rosserial protocol, wich will allow the Arduino board, once wired to the Raspberry Pi, to publish data on a specific topic and to subscribe to other topics.

As a computer, I used the Raspberry Pi by connecting it to a screen, a mouse and a keyboard. The code developed was stored in the Raspberry Pi and uploaded on an internet platform, Github.
All the codes were contained in the following two main folders, one for Arduino and the other to be "launched" through the Raspberry Pi board:

- Arduino folder
  - Main code (.ino)
  - Low-level controller 1 (.cpp)
  - Low-level controller 2 (.cpp)
  - …
  - Arduino libraries
    - ros_lib (library for rosserial)
    - Other Arduino library
    - …
    - Arduino class 1
    - Arduino class 2
    - …
- ROS workspace
  - src
    - ROS node 1 (.cpp)
    - ROS node 2 (.cpp)
    - …
    - Launch file (.launch)
  - CMakeLists.txt
  - Package.xml

The folder Arduino libraries was only used for storage, as the Arduino software uses a specific folder named "libraries". When updating our local Arduino libraries for the first time, we need to put the folders contained in the "Arduino libraries" folder inside the "libraries" folder used by Arduino. However, to change code on any library, we can modify the local library files from the "libraries" folder and save them later, on the "Arduino libraries" folder for storage and uploading it on Github.

Arduino (.ino files) code can be edited in the Arduino software along with the libraries and low-level controllers in c++. When the code is ready for testing, the Upload function updates the Arduino board with the main code (.ino) selected and the libraries in the local "libraries" folder.

ROS files such as nodes and launch files are edited in a local pre-created ROS workspace. Then, all ROS features for launching the nodes, view topic information with rqt or visualize data with rviz are command line to enter through command prompts.

II.3.b) Modularity

Something is modular if can fit several uses. Here, in our sailboat system, modularity can be used in order to save time assembling the boat, changing its electronic components or programming it.
In order to build a fleet, the best guideline for choosing our components is to use ones that has connections, that group the most functions we need and that we know to be reliable in terms of availability. This way, the sailboat will be easier to build and less likely to have construction defects.

Here is how we can represent our system, from a point of view of the connection between three groups of elements:

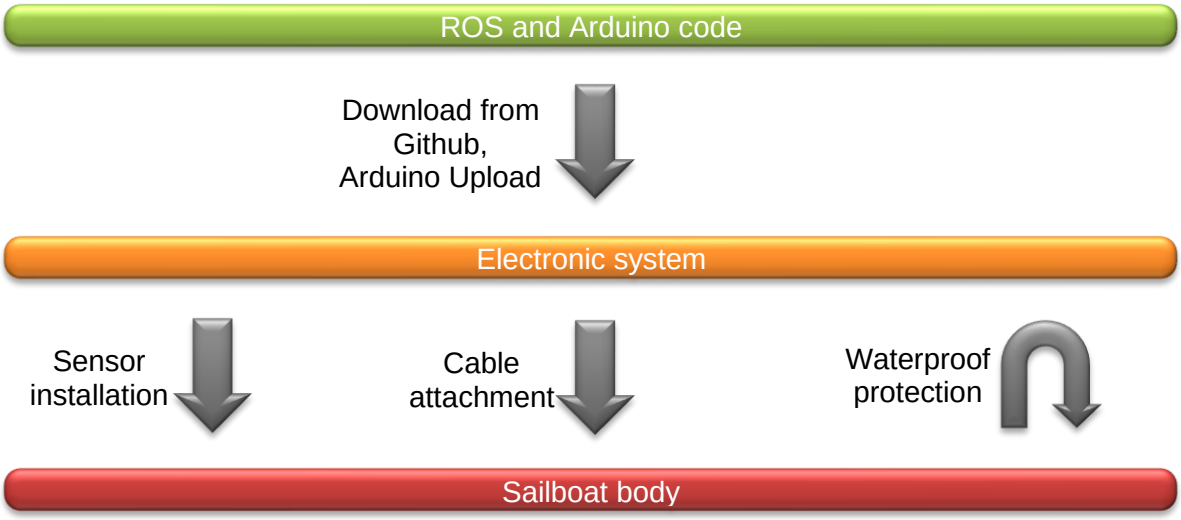


Figure 6 – Modularity and compatibility issues

The Electronic system groups all boards, electronic cables and sensors except for the sail and rudder actuators which will already be included in the sailboat body on purchase.
What needs to be done is the supports that will allow some of the sensors to be secured to the sailboat body, with the necessary protection against oxidation.
The compatibility between the code and electronic need to be established in a way that the ROS workspace needs to be prepared and the Arduino local files filed.

II.3.c) Bill of material

The bill of material allows us to anticipate the project costs and theoretically follows from the considerations made previously. However, me might need to compromise between the equipment we already have and the ideal one. In my case, my internship tutor already knew a large part of the material I was going to use, because of his experience. Hence, I gained time on the order of components.

Component	Model	Price	Availability	Link
Sailboat	Pro Boat Ragazza 1 Meter Sailboat V2	£285.99	Currently out of stock	https://www.elitemodelsonline.co.uk/Boats/By-Manufacturer/Pro-Boat/22231-/Pro-Boat-Ragazza-1-Meter-Sailboat-V2-RTR
GPS	GlobalSat BU-353-S4 USB GPS Receiver	$26.08	OK	https://www.amazon.com/GlobalSat-BU-353-S4-USB-Receiver-Black/dp/B008200LHW
Wind sensor	Anemometer for MicroLogger Weather Station	£190.00	OK	https://www.measurementsystems.co.uk/sensors_and_meters/meteorological-sensors/anemometer-for-micrologger-weather-station-analog?gclid=EAIaIQobChMIqvv0kcXA4wIVh0PTCh2uvQp-EAkYAyABEgIK0_D_BwE
Camera	Raspberry Pi camera v2.1	£24.00	OK	https://shop.pimoroni.com/products/raspberry-pi-camera-module-v2-1-with-mount?variant=19833929735&currency=GBP&utm_source=google&utm_medium=cpc&utm_campaign=google+shopping&gclid=EAIaIQobChMI--vU26fB4wIViZ3VCh1dXgMtEAkYASABE

				gI0GfD_BwE
IMU	Groove IMU 10DOF v2.0	£14.00	OK	https://www.unmannedtechshop.co.uk/product/grove-imu-10dof-v2-0-mpu-9250-and-bmp280/?gclid=EAIaIQobChMI_pqHk9DA4wIVRvhRCh2lzQRiEAkYASABEgIKC_D_BwE
Raspberry Pi board	Raspberry Pi 3 Model B+	£29.89	OK	https://www.amazon.co.uk/Raspberry-Pi-Model-64-Bit-Processor/dp/B07BDR5PDW/ref=asc_df_B07BDR5PDW/?tag=googshopuk-21&linkCode=df0&hvadid=310818960639&hvpos=1o1&hvnetw=g&hvrand=11215478279115832376&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9045313&hvtargid=pla-433437388235&psc=1
Arduino board	Arduino Mega 2560 R3	£36.00	OK	https://coolcomponents.co.uk/products/arduino-mega-2560-r3?variant=45223081486&currency=GBP&gclid=EAIaIQobChMIy4Cvvs_A4wIVB1XTCh3cAAL_EAkYASABEgLBiPD_BwE
Arduino shield sensor	Grove Mega Shield V1.1	£5.95	OK	https://www.ebay.co.uk/i/332587862928?chn=ps&norover=1&mkevt=1&mkrid=710-134428-41853-0&mkcid=2&itemid=332587862928&targetid=594652109720&device=c&adtype=pla&googleloc=9045313&poi=&campaignid=1700604133&adgroupid=68042089162&rlsatarget=pla-594652109720&abcId=1140496&merchantid=7388724&gclid=EAIaIQobChMIvuiRr8_A4wIVk_dRCh3oWQ4kEAkYBSABEgI8PfD_BwE
Arduino shield motor	Adafruit 1411 16-Channel 12-bit PWM Servo Shield	£19.00	OK	https://www.rapidonline.com/Adafruit-1411-Servo-PWM-Shield-16-Channel-12-bit-I2C-Interface-for-Arduino-75-0558?IncVat=1&pdg=pla-337654354019:kwd-337654354019:cmp-757438067:adg-44804851896:crv-207912323492:pid-75-0558:dev-c&gclid=EAIaIQobChMIgIjt9szA4wIVxeFRCh00iguGEAkYAiABEgLqfvD_BwE

Table 2 – Bill of material

The total expenses, without counting the assembling costs or 3D printing costs, amounts to £630.91, which is, with the current conversion rate of £1=1.10€, equal to 694€.
The remote control is included in the sailboat price as it is part of the product.

We might face a problem with the sailboat's availability and have to see if the company will resume its distribution or if we can use another similar sailboat.

III. Coding structure

III.1. Information flow

In order to manage the communication between the different processes and devices, we need to define or identify the different data types to use.
All different processes or actors that will transform data are:

- ⇨ The remote control computer: from **user request** to **behavior command**.
- ⇨ The ROS middleware including:
  - The state machine node: from **filtered sensor data** and **behavior command** to **simple orders** and . The behavior command will be the state machine choice and the filtered sensor data, the parameters for each state machine.
  - Filters nodes: from **sensor data** to **filtered sensor data**. Their function is generally to give more reliable data from sensor data by replacing outliers or smoothing variations.
  - Arduino serial node: from **simple order** to **simple order** and **sensor data** to **sensor data**. The serial node is basically an information smuggler, it distributes sensor data to their adapted filter.
  - The camera treatment node: from **raw data** to **filtered sensor data**.
  - The rosbag function in ROS that allows to store data and replay their progress once the mission is over.
- ⇨ The sensors: from the physical environment to **raw data**.
- ⇨ The remote control: from **user request** to **simple order**. A simple remote control with rudder and sail control.
- ⇨ The Arduino board: Collects data from all the sensors it is connected to and transmits it via the serial node. Applies order that comes from the state machine node or, in priority, the command from the remote controller.

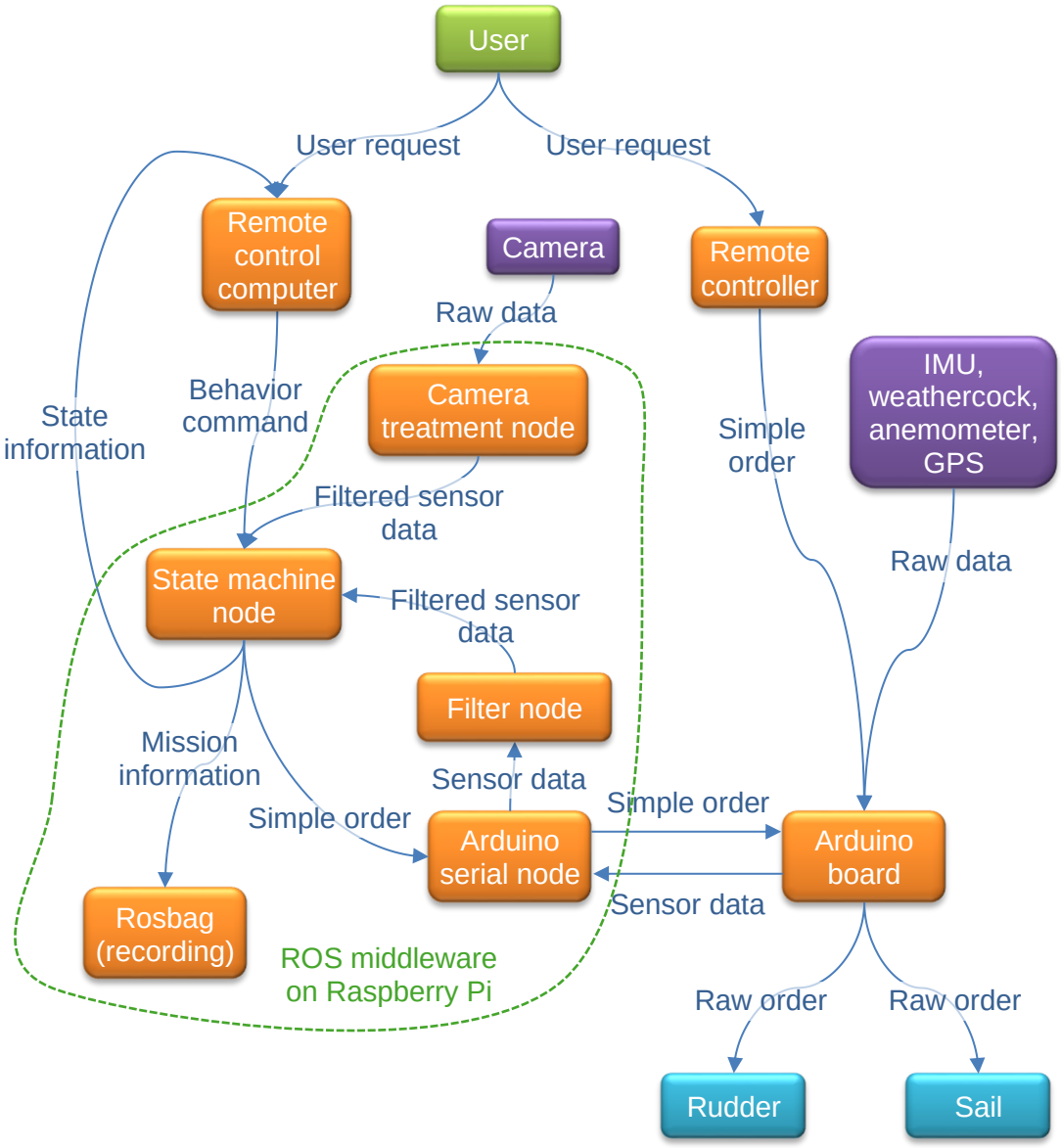


Figure 7 – Information flow graph

The following figure presents the different repartition of orders and the hierarchy between the tasks to be accomplished.

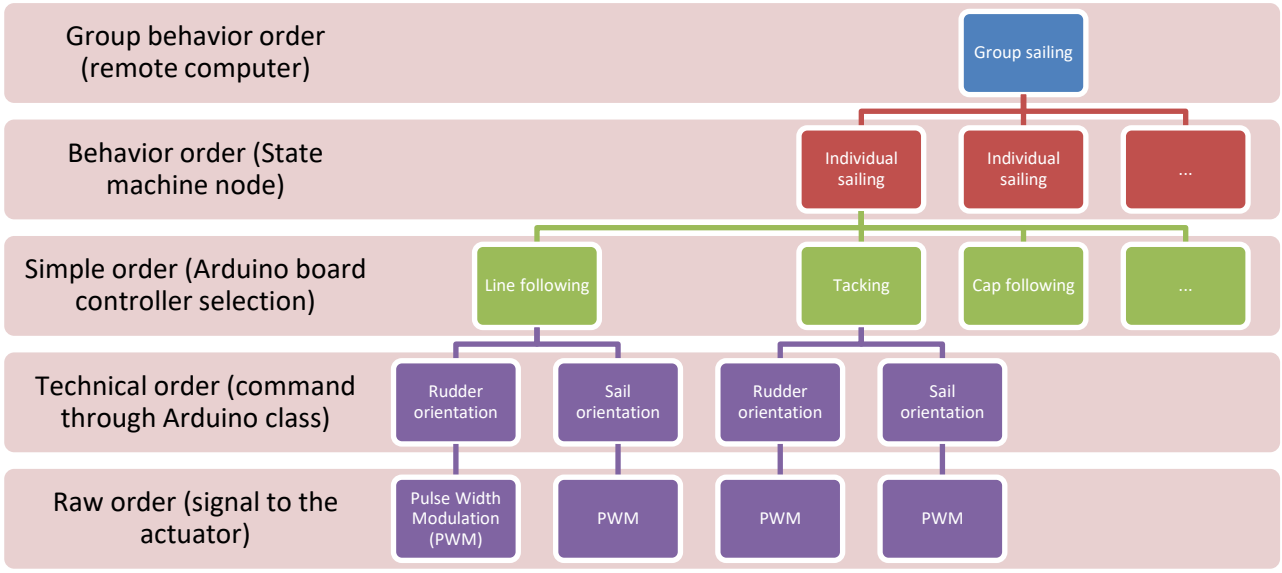


Figure 8 – Orders and hierarchy

In that way, the fleet commander can have a choice between different group sailing mode and select them through the remote computer. For example, either we ask the fleet to go pick up rubbish, or to come back home, to stand still or to transport goods to another port.
The sailboat itself, for a specific group behavior, follows a specific state machine which uses the simple orders it can give to the Arduino board such as a cap following, a line following between two points or staying in a given zone.
Each simple order can be declined in technical order given to the actuator classes, by demanding different orientations for the rudder and the sail.
These classes traduce these orders by activating the right PWM to the servo motor activating the mechanical function.

III.2. Arduino structure

The Arduino structure I used was a simplification and adaptation of the one the researcher Ulysse Vautier let me start from, with slightly different sensors. Along with the practice of coding C++ classes, the idea of using classes was to use polymorphism in order to organize the processes by functions which are:

- ⇨ **update()**: the updating function for all sensor measures inside their respective class. Low-level calculations are done at this level to measure the incoming signal and to store it into a value that makes sense.
- ⇨ **communicateData()**: the function that triggers the publication of the sensor data through the serial node. The same applies for the current actuator commands.
- ⇨ **ControlTime()**: the actuator function that triggers the application of the actuator's command, by checking if its refresh period has passed.

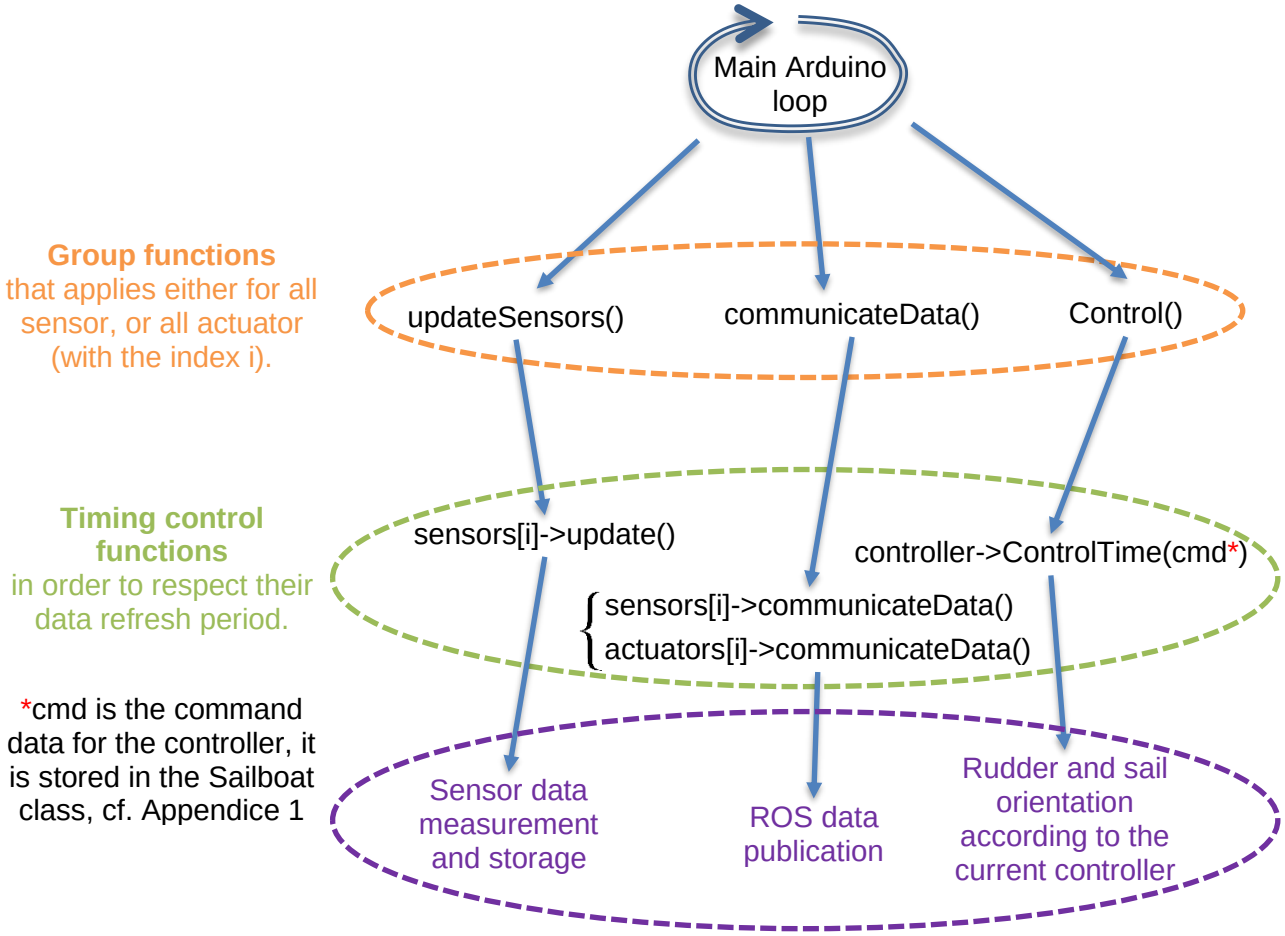


Figure 9 – Arduino code process

The orange zone is the code in the main Arduino loop where the functions are called from the Sailboat class.
The green zone is the code in the Sailboat class where the functions are called from the Sensor, Controller or Actuator classes stored in the Sailboat class. There, we find all the virtual functions to be used regardless of the sensor, controller or actuator.
The timing control functions are just a control layer that is passed if the refresh period is respected. In that case, we end up in the violet section, which is the application of the Sensor, Controller or Actuator main function.
The Appendice 1 details the complete Arduino class architecture, with the initialization functions and the data stored in each classes.

III.3. Protocols

Protocols are the practical actions that are repeated with a potentially high frequency. I chose to identify many of the ones I used because they meet the following benefits:

- ⇨ Identification of the good working method: Firstly, because of the repetitiveness of these actions, we could someday forget some steps without realizing it. Then, it is the method that usually works until now, which means that it helps debugging if we needed to use a new protocol that could be better.
- ⇨ Narrowing a problem's origin: Checking the protocol tracking is the first verification to do before debugging a program. It also helps clarifying the problem when asking for help.
- ⇨ Reference for beginners or project takers who are not familiar with the way of working on the project.

In this process, I detailed in the Appendices the following protocols:

- ⇨ Code test protocol
- ⇨ Code modification protocol
- ⇨ Sensor integration protocol
- ⇨ Actuator integration protocol
- ⇨ ROS node integration protocol
- ⇨ Maintenance protocol

We can say that the protocols for sensor and actuator integration are quite heavy. Fortunately, the sailboat output structure will never grow much and adding more sensors are not for today. The Arduino code might also be improved so as to detect automatically the presence of sensors and better adapt itself to it a little more.

IV. System intern security

IV.1. Robustness

The sailboat can face different types of problem:
- Mechanical problems such as breakage, oxidation, disconnections, …
- Algorithmic problems which concerns all types of program crash, but also wrong decisions when avoiding an obstacle for example.

Mechanical problems can be avoided through dimensioning given the operating environment, with the choice of safety coefficients or repeated tests on the structure.
For flotation tests, we let the sailboat in a water tank long enough to detect water leaks and to make it waterproof.
However, if something happens in the middle of a mission, we want to know if the sailboat can finish its mission or come back without finishing it, or not coming back at all, even with manual control.
Often, the closer to hardware we are, the more critical it is. If we lose actuators such as the rudder or the sail, the system stops working properly and no on-board decision can be taken in order to save it.

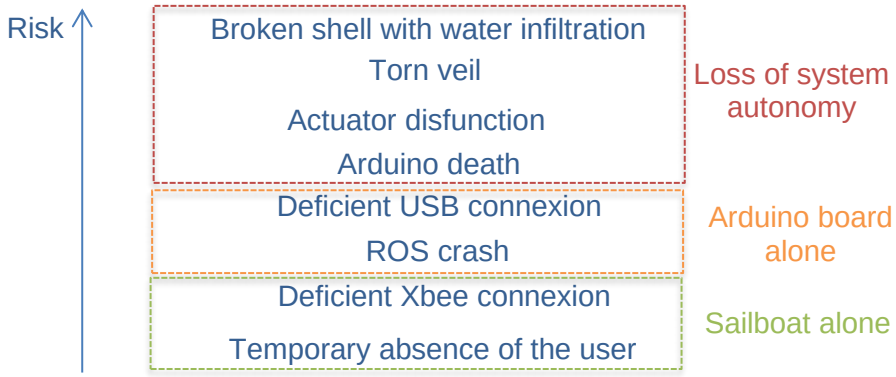


Figure 10 – System robustness

When the system has lost his autonomy, it means no on-board solution can be found in order to come back to the user.

In fact, we can even specify alternative problems with a tree graph:

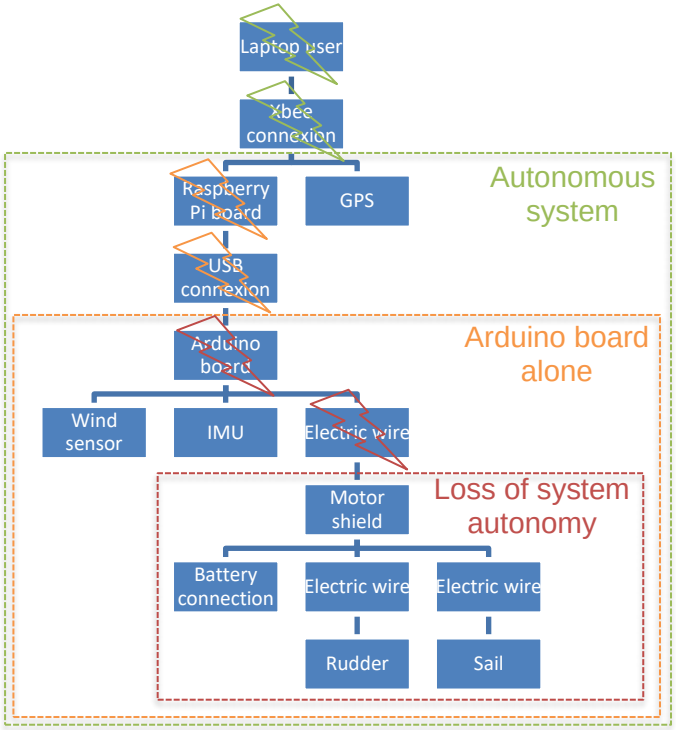


Figure 11 – System robustness tree

If one box is deficient, it will damage dataflow between the entire "leaf" part and only the "branches" that go up to the tree's "root".
For example, if the IMU gives wrong data, say because of calibration problem, then neither the wind sensor nor the motor shield (and its wire) will be affected. However, data coming from the Arduino board, the USB connection, the Raspberry Pi board, the Xbee connection and will be corrupted.

IV.2. Risk list

A list can be drawn up in order to identify system threats and to implement an appropriate protection. Depending on the recurrence of the problem during the use of the sailboat, a possible update of the risk consideration may lead to a system revision.

The risk identification can either damage the system in a short-term perspective, or in a long-term way, limited to a single mission.
Risk level:

0: Not a risk.
1: Damages quality (precision, quality of mission data, …)
2: Damages mission's chances of success, but not the boat itself (lower precision, wrong behavior, …).
3: Damages the boat itself to a lesser extent.
4: Damages the boat itself to a higher extent, including expensive material (possibly more than £50).
5: Puts the user at risk.

We can, then, identify the probability of the problem happening, depending on the system implementation. In doing so, we shall use an estimated percentage of failure, or risk happening, in order to specify it. We will keep at a 5% precision since it is roughly estimated, and we will assume that a 0% failure might be the result of a demonstration (for codes), but still means 0.01% instead.

Tree level	Risk	Risk level (/5)	Current probability	Main causes

			level (%)	
Any	Oxidation	4	10	Water infiltration, poor permeability
	Wire disconnection	3	10	Hanging, bad welding
IMU	Inexact compass calibration	1	80	Ferrous materials nearby
Wind sensor	External blocking of rotation	1	10	External environment, lack of protection
???	Boat crash	4	Unknown	Faulty obstacle avoidance in triggering, method or precision
Any	Component catching fire	5	10	Bad sizing, unwanted connection, bad welding
Raspberry Pi	Bug	2	90	A large range of causes
Arduino	Bug	2	90	A large range of causes

Table 3 – Risk list and analysis

The higher the risk level is, the more important is taking it into consideration for implementing a robust solution.
The main criteria we can have an impact on is the probability level of a problem occurring, as long as the main cause is the system itself. This probability level can be lowered through solution design. This corresponds to the “virtuous circle” described below:

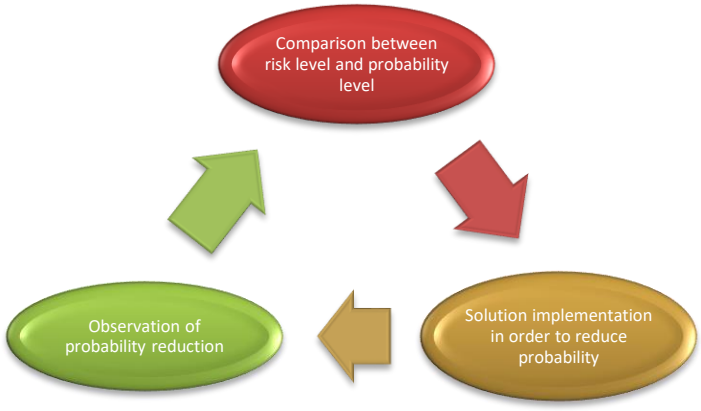


Figure 12 – System protection virtuous circle

It is also possible to design a solution that reduces the risk level of a specific risk, but in all cases, a solution implementation implies the emergence of a new line in the previous table.

For example, if we consider the external blocking of rotation of the wind sensor. We can design a physical protection in order to prevent such problem from happening. With a good design, the probability has been reduced to almost 0%, but we are facing a new risk which is the solution's failure. This means we are adding a new line to our table:

Tree level	Risk	Risk level (/5)	Current probability level (%)	Main causes
Wind sensor	External blocking of rotation	1	0	External environment, lack of protection
Protection structure	Boat crash	1	10	

Table 4 – Risk protection example

V. Fleet building

V.1. Assembling

In order to visualize how the sailboat pieces are assembled, the figure below shows the physical links between all the sailboat parts. The yellow links are the connections between the electronic parts and the mechanical structure.

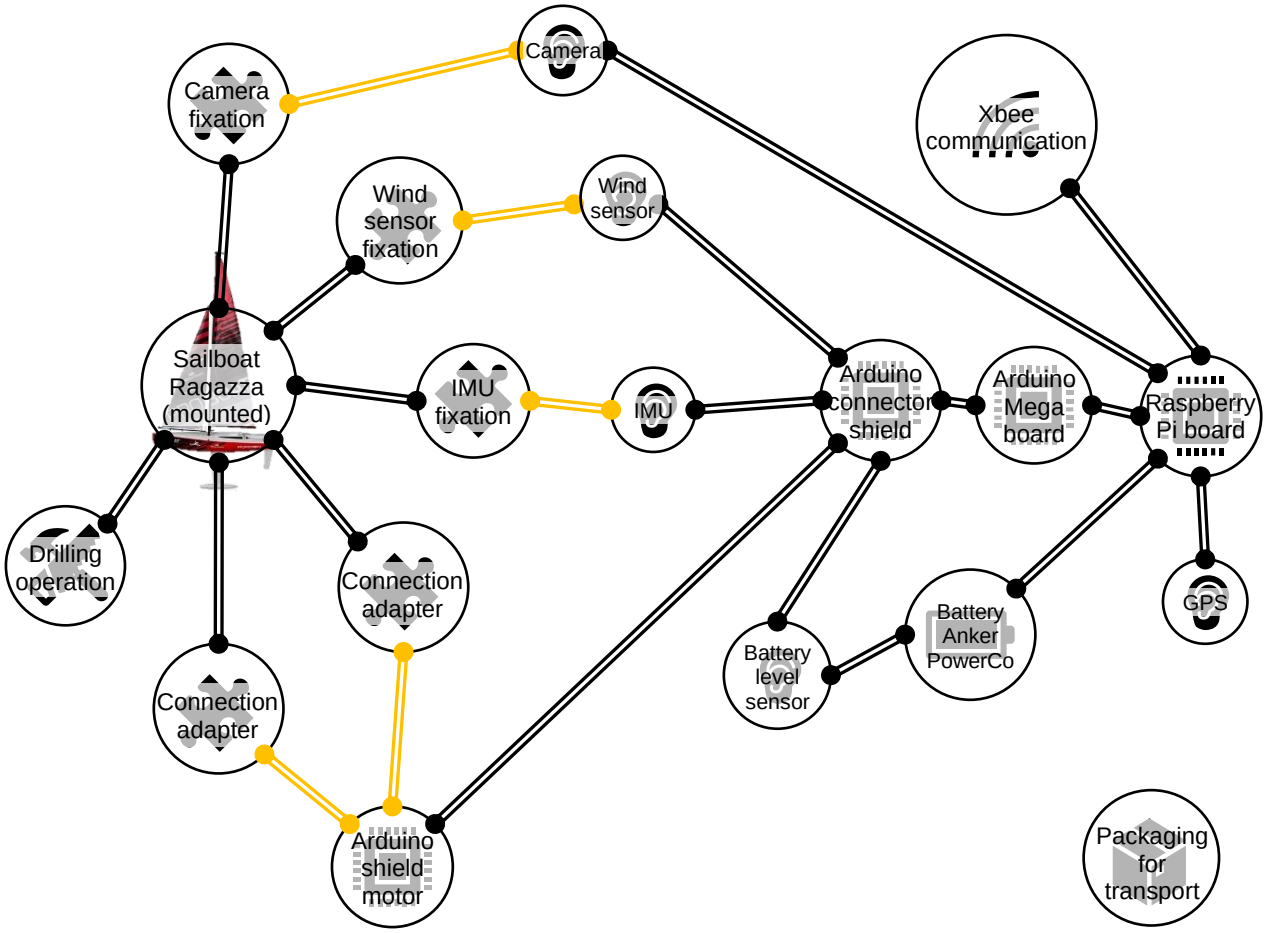


Figure 13 – Links between the system's parts

We see that the assembling can be done by realizing the black links, with the two groups on the left and right of the yellow links simultaneously. The last steps are the electronic device fixation on the sailboat, and the Arduino and Raspberry Pi boards can be placed inside the hull.

A more detailed, step by step diagram can be drawn in order see more precisely the sailboat's sequence of assembly.

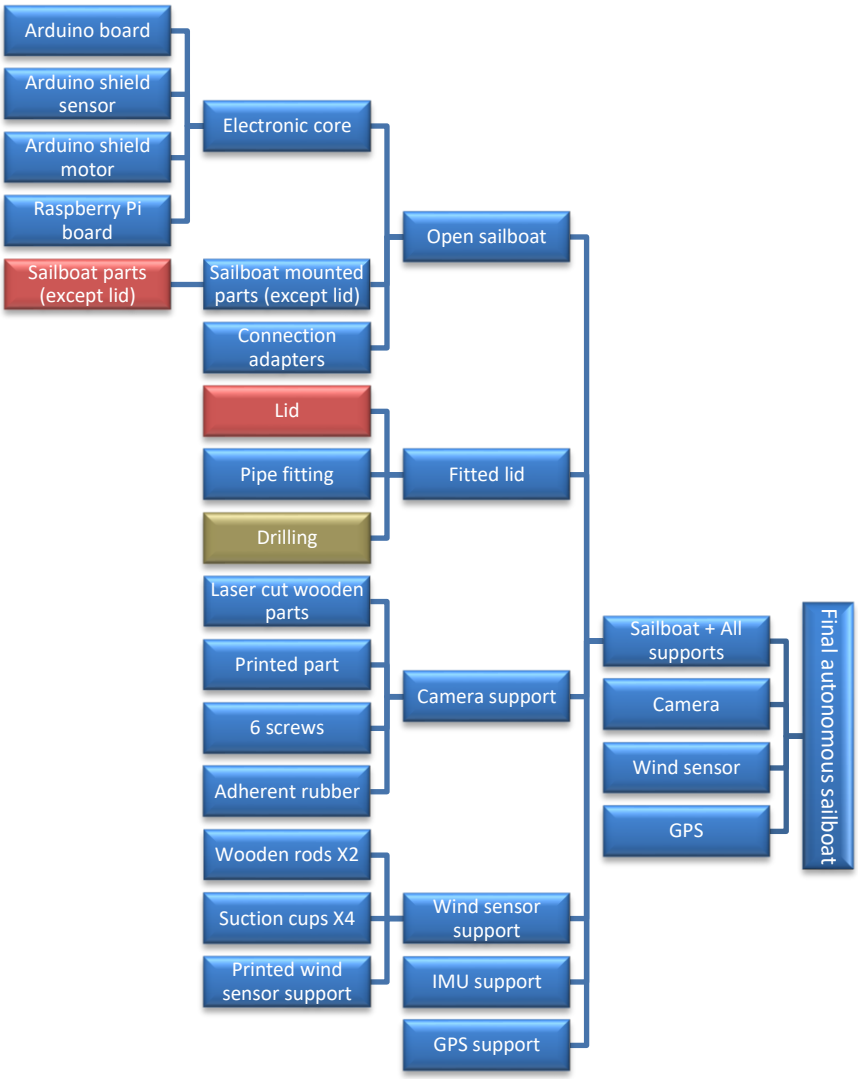


Figure 14 – General assembling scheme

This is the final view of the wind sensor support:

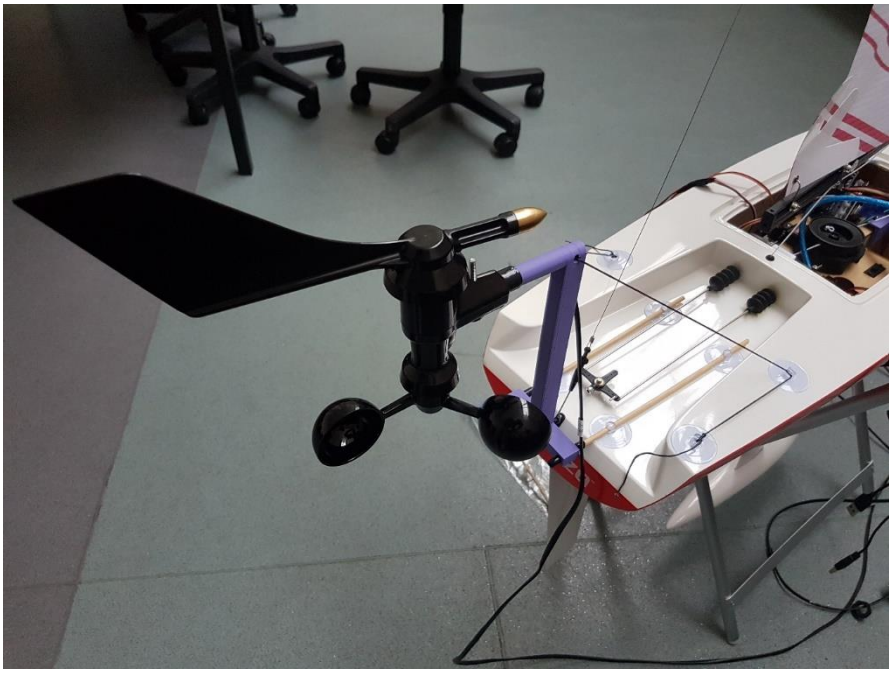


Figure 15 – Wind sensor photograph

V.2. Fleet behavior

The constitution of a fleet opens a possible practice of programming a group behavior for sailboats. Some examples can show new areas of performance:

- ⇨ Marine waste collection:

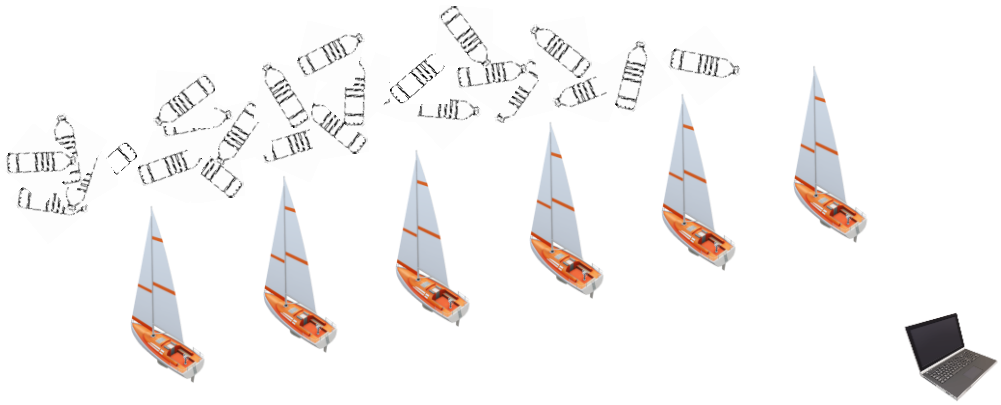


Figure 16 – Marine waste collection

Either by localizing, regrouping or collecting waste from the sea, autonomous sailboats can be a solution where we could avoid making go-rounds ourselves.

- ⇨ Wind force mapping:

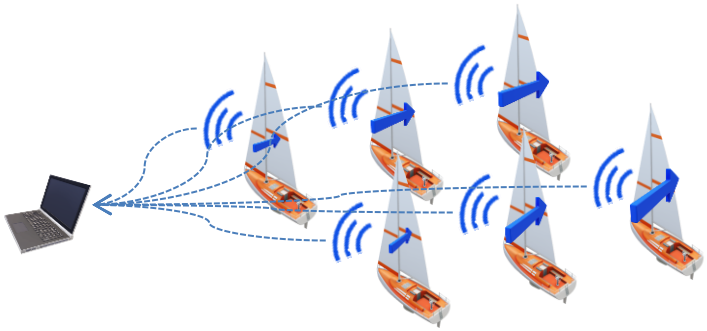


Figure 17 – Wind force mapping

With the use of a supervisor that collects all data from the sailboats, it can find out which sailboat of the fleet receives more wind in order to guide the fleet towards more favorable areas.

- ⇨ Faulty sensor compensation:

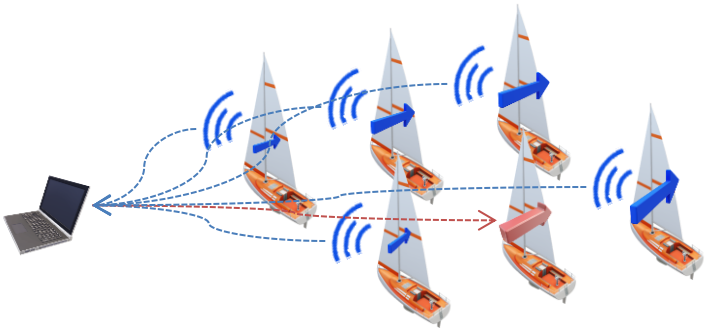


Figure 18 – Faulty sensor compensation

When supervising the fleet, the remote computer can detect if a sailboat gives outliers and filter its data with field approximations.

- ⇨ Obstacle avoidance for the group:

Figure 18 – Group obstacle avoidance

The obstacle avoidance for a fleet is more complex if we want to prevent the sailboats from colliding with each other. It requires a group reaction where the solutions are open.

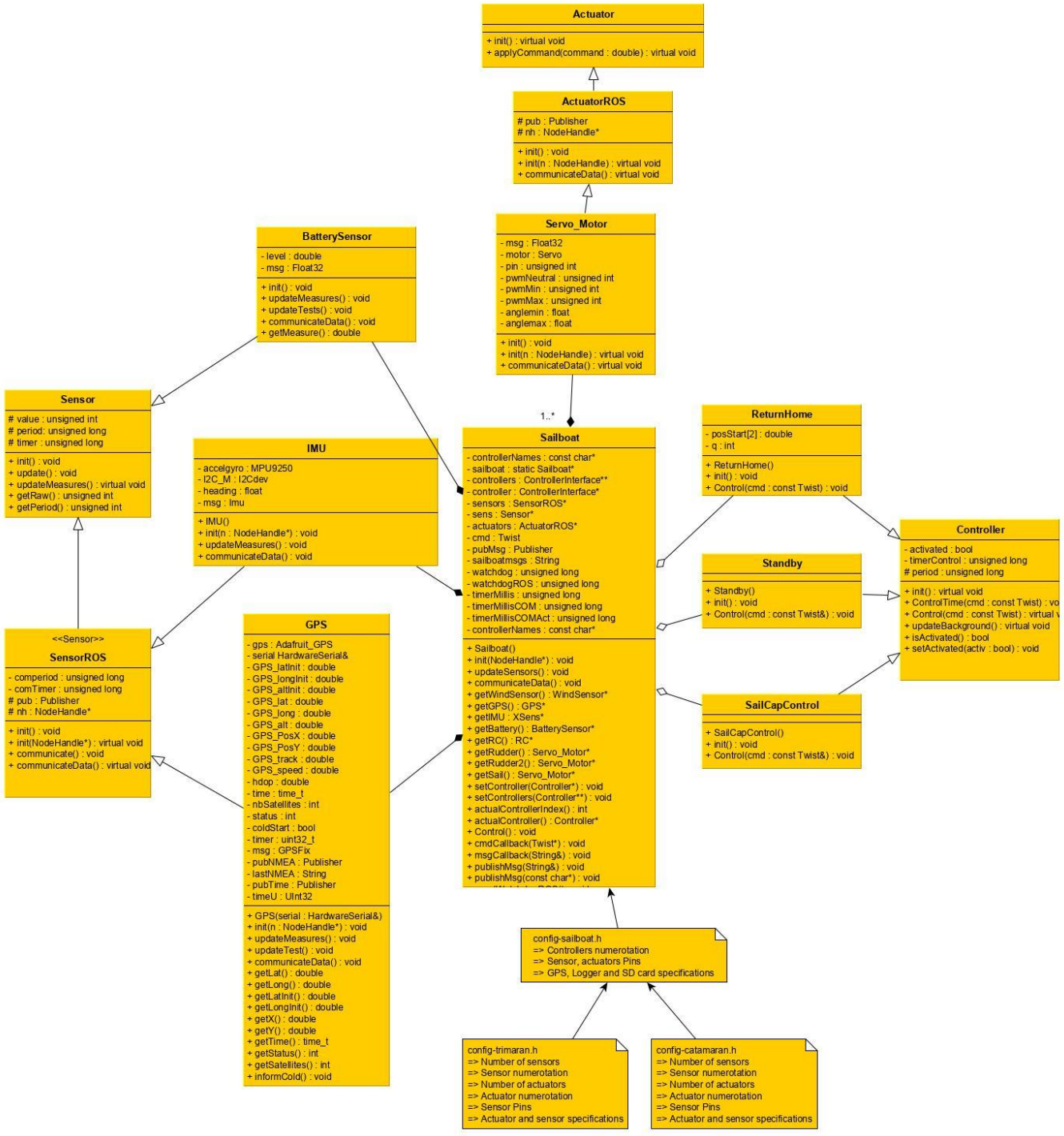
VI. Conclusion

These twelve weeks at the university of Plymouth allowed me to improve myself in coding and to use more complex code structures that can be useful later.
I particularly achieved a real method of code debugging, whether by searching deeper into Arduino libraries or having better methods.

This internship made me apprehend the preparation of a research budget and develop the approach of research topics related to marine robotics. I could formalize different concepts and adapt them to the specific subject.

VII. Appendices

VII.1. Arduino class architecture



VII.2. Code test protocol

⇨ **Activation of the entire system:**

- Open a terminal:
  - Move in the workspaceRos folder;
  - Launch the ROS core:

```
catamaran@catamaran:~$ cd workspaceRos/
catamaran@catamaran:~/workspaceRos$ roscore
```

- Open a second terminal (with ctrl+maj+t):
    - Launch the serial node:

```
catamaran@catamaran:~/workspaceRos$ rosrun rosserial_python serial_node.py
_baud:=115200 _port:=/dev/ttyACM0
```

- Open a third terminal (with ctrl+maj+t):
    - Launch the ROS nodes:

```
catamaran@catamaran:~/workspaceRos$ roslaunch catamaran launch.launch
```

- Open a fourth terminal (with ctrl+maj+t):
    - Run the rqt application:

```
catamaran@catamaran:~/workspaceRos$ rqt
```

VII.3. Code modification protocols

⇨ **Modification of the Arduino program (.ino) or an Arduino library:**

- Save the modified file.
- In the Arduino IDE, upload the .ino file: debug if necessary until upload works.
- Activate the entire system to see modifications.

⇨ **ROS node modification:**

- Save the modified file.
- Open a terminal:
    - Move in the workspaceRos folder;
    - Build the catkin workspace code:

```
catamaran@catamaran:~$ cd workspaceRos
catamaran@catamaran:~/workspaceRos$ catkin_make
```

    - Debug if necessary, until the previous command is successful.
- Activation of the entire system to see modifications.

VII.4. Sensor integration protocol

- Add a folder with the sensor name in the Arduino library
- Complete this folder with the .cpp file and the .h file that include:
    - The inclusion of the library Sensor.h
    - A definition of the class inheriting of Sensor or SensorROS
    - If it inherits from Sensor:
        - Define the class variables that need to be kept between two measures updates.
        - Define an initialization function: init(). It needs the call of “SensorROS::init(n)”.
        - Define the measure function that updates the class variables: updateMeasures().
    - If it inherits from SensorROS:
        - Define the class variables that need to be kept between two measures updates.
        - Define the message to be published to communicate the data.
        - Define an initialization function: init(ros::NodeHandle* n).
        - Define the measure function that updates the class variables: updateMeasures().
        - Define the publishing function: communicateData().
- Modify the file config-catamaran.h by increasing the number of sensors and configuring the number attribution among all sensor variable.
- Include the sensor library (.h) in the file Sailboat.h
- Modify the function Sailboat::init() in file Sailboat.cpp so as to create and store the sensor in the sailboat’s data (sensors[] or sens[]).
- Check that the ROS message has a subscriber node.

⇨ Optionnal: assign in init() the “comperiod” value which specifies the period of sending data through a ROS topic.
⇨ Optionnal: assign a “period” value in the sensor declaration (in Sailboat::init()) in order to specify the measure’s update period.

VII.5. Actuator integration protocol

- Add a folder with the actuator name in the Arduino library
- Complete this folder with the .cpp file and the .h file that include:
  - The inclusion of the library Actuator.h
  - A definition of the class inheriting of Actuator or ActuatorROS.
  - If it inherits from Actuator:
    - Define class variables that need to be kept between two actuations.
    - Define an initialization function: init().
    - Define the “applyCommand(double command)” function which is the execution of the actuator’s function from an order passed as parameter.
  - If it inherits from ActuatorROS:
    - Define class variables that need to be kept between two actuations.
    - Define an initialization function: init(ros::NodeHandle* n).
    - Define the “applyCommand(double command)” function which is the execution of the actuator’s function from an order passed as parameter.
    - Define the publishing function: communicateData().
- Modify the file config-catamaran.h by increasing the number of actuators, adding a variable for the new actuator and configuring the number attribution among all actuator variables.
- Modify the file config-sailboat.h by assigning the Arduino pin used to control the actuator.
- Include the actuator library (.h) in the file Sailboat.h
- Modify the function Sailboat::init() in file Sailboat.cpp so as to create and store the actuator in the sailboat’s data (actuators[]).

VII.6. ROS node integration protocol

- Add a new file .cpp in the workspace: workspaceRos/src/catamaran/src/
- Modify the file CMakeLists.txt in workspaceRos/src/catamaran/:
  - Complete, if necessary, the missing libraries used in the .cpp
  - Add the command to create the executable.
  - Target the linked libraries.
- Build the catkin workspace code:

```
catamaran@catamaran:~/workspaceRos$ catkin_make
```

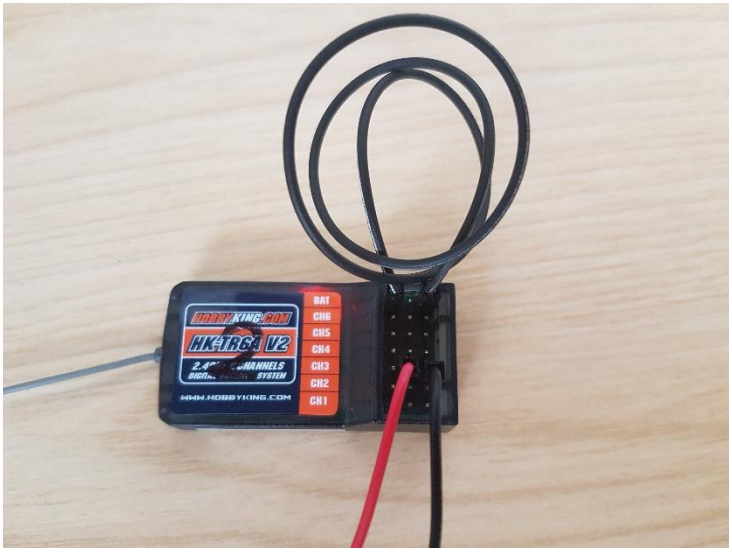
- Debug if necessary, until the previous command is successful.

VII.7. Maintenance protocol

The system needs:
- IMU calibration (before each mission):

Launch a specific procedure made by the trainee Corentin Jegat.

- Remote controller binding:



⇨ Connect pin BAT-1 to BAT-3, CHX-2 to 5V and CHX-3 to GND. The red led on the receiver should be blinking.
⇨ On the transmitter, hold down the bind range test and then, turn the transmitter on. The red led on the receiver should be staying on.
⇨ Disconnect the receiver from its alimentation.
⇨ Turn off the transmitter.