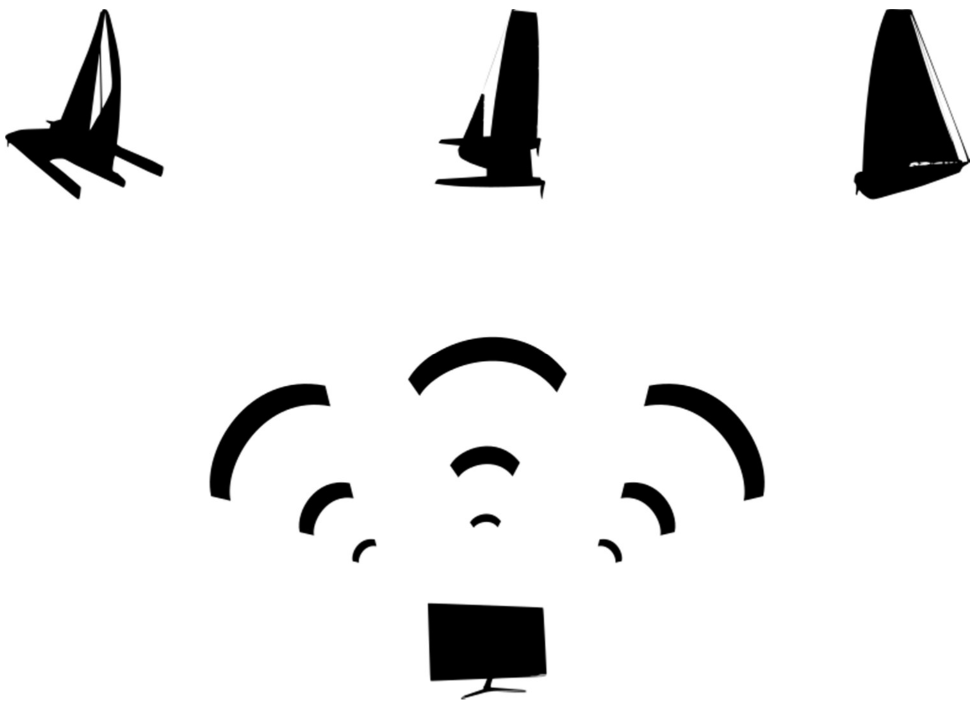
Robotics Internship Report



Development of autonomous sailboats

ENSTA Bretagne
2 rue François Verny
29806 Brest Cedex 9, France

Bouveron Matthieu,
matthieu.bouveron@ensta-bretagne.org

Tutor: Professor Luc JAULIN,
Professor in Robotics, ENSTA
Bretagne -LabSTICC

Supervisor: Doctor Jian WAN, Lecturer
in Control Systems Engineering,
University of Plymouth -AMS

08/2019

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Doctor Jian Wan, for welcoming me to such an interesting project, for his responsiveness to my requests and for trusting me so much.

My tasks in this project would have never been completed without the lessons provided at ENSTA Bretagne, especially without the aid of Professor Luc Jaulin. I am deeply thankful to all the ENSTA Robotics teachers.

Eventually, my thanks also go to the other members of the project for their team spirit and great work. I would specifically like to thank Ulysse Vautier for his many suggestions and his good humour, which contributed to make the work very pleasant every day. His keen sense of Physics was a great help.

Abstract

This report deals with the development and commissioning of a fleet of autonomous sailboats as part of the French second year engineering school *assistant engineer internship*, focusing in particular on the project's communications and artificial vision axes. The assembly, integration of the electronic elements, software and commissioning of the sailboats were carried out by a team of three people over three months at the University of Plymouth. The boats are equipped with a low-cost system, easily reproducible and adaptable to different types of sailboats and are able to carry out autonomous missions. These missions may require coordination between several sailboats, under the control of an operator.

Résumé

Le présent rapport traite du développement et de la mise en fonctionnement d’une flotte de voiliers autonomes dans le cadre du stage dit *Assistant-Ingénieur* de deuxième année d’école d’ingénieurs, en s’attachant notamment aux axes *communications* et *vision artificielle* du projet. Le montage, l’intégration des éléments électroniques, les logiciels et la mise en service des voiliers ont été réalisés par une équipe de trois personnes pendant trois mois à l’Université de Plymouth. Les bateaux sont équipés d’un système peu cher, facilement reproductible et adaptable à différents types de voiliers et sont capable d’assurer des missions en autonomie. Ces missions nécessitent éventuellement une coordination entre plusieurs voiliers, sous le contrôle d’un opérateur.

Table of contents

The University of Plymouth

The University of Plymouth is a vast complex hosting a wide variety of academic profiles, such as law, psychology, geographical sciences, computing, art, health, business and marine science. It is also a reputable research centre in these fields, with a specific attention paid to Marine and Ocean Engineering.

Much work and advertisement is done around the ocean issues in the University of Plymouth, highlighting the fact that this is a core preoccupation nowadays. In line with this approach, climate activist Greta Thunberg was welcomed in Plymouth earlier this year and met with members of the University's International Marine Litter Research Unit.

Figure 1: University of Plymouth

Introduction

Not so long ago, sailboats used to dominate maritime trade. However, the industrial revolution quickly relegated them to recreational use, as their lack of speed and their dependency upon the weather conditions made them unable to compete with powerful motorised ships.

New preoccupations such as cost reduction and public pressure around climate change are nevertheless beginning to bring sailboats back into fashion. Indeed, as opposed to their motorised counterparts, sailboats require only little energy to be stored on board to power the actuators, as the main energy source is the wind. Most advanced sailboats can even be energy self-sufficient using wind turbines, solar panels, etc.

Moreover, 19th - 20th centuries commercial sailing ships were globally inefficient against the wind, making them intolerably dependent upon weather conditions. Much progress has been done about this, which now allow sailboats to be effective in most directions. Another drawback of sailboats used to be that they required a large crew to move the sails. However, automation now allows to easily control the sailing configuration of any boat, with a reduced crew. See the example of the *Maltese Falcon*, which can technically be operated by a single person.

Figure 2: Maltese Falcon

This efficiency concerning power consumption and the fact that they do not have so much drawbacks anymore, along with the miniaturisation of electronic devices such as computers and GPS, make sailboats suitable for long missions in the ocean. In the same time, scientists always seek reliable means to get data on the ocean on a regular basis. Organizing scientific missions with big motorized boats quickly becomes expensive as a crew is also needed for a long time in this case. In order to be able to provide research such a stream of data, an alternative is to use a fleet of autonomous sailboats which, as we saw above, offer both a good manoeuvrability and a very low energy dependence.

The purpose of this internship report is to show the steps of the creation of such a fleet, from the expectations of the supervisor to the tests. After the description of the tasks carried out by the working team, this report will mainly focus on communications and artificial vision.

1. Expectations

The purpose of the internship was to be able to operate a fleet of different kinds of autonomous sailboats, with the choice for the operator of either giving them a specific mission or remote-controlling them.

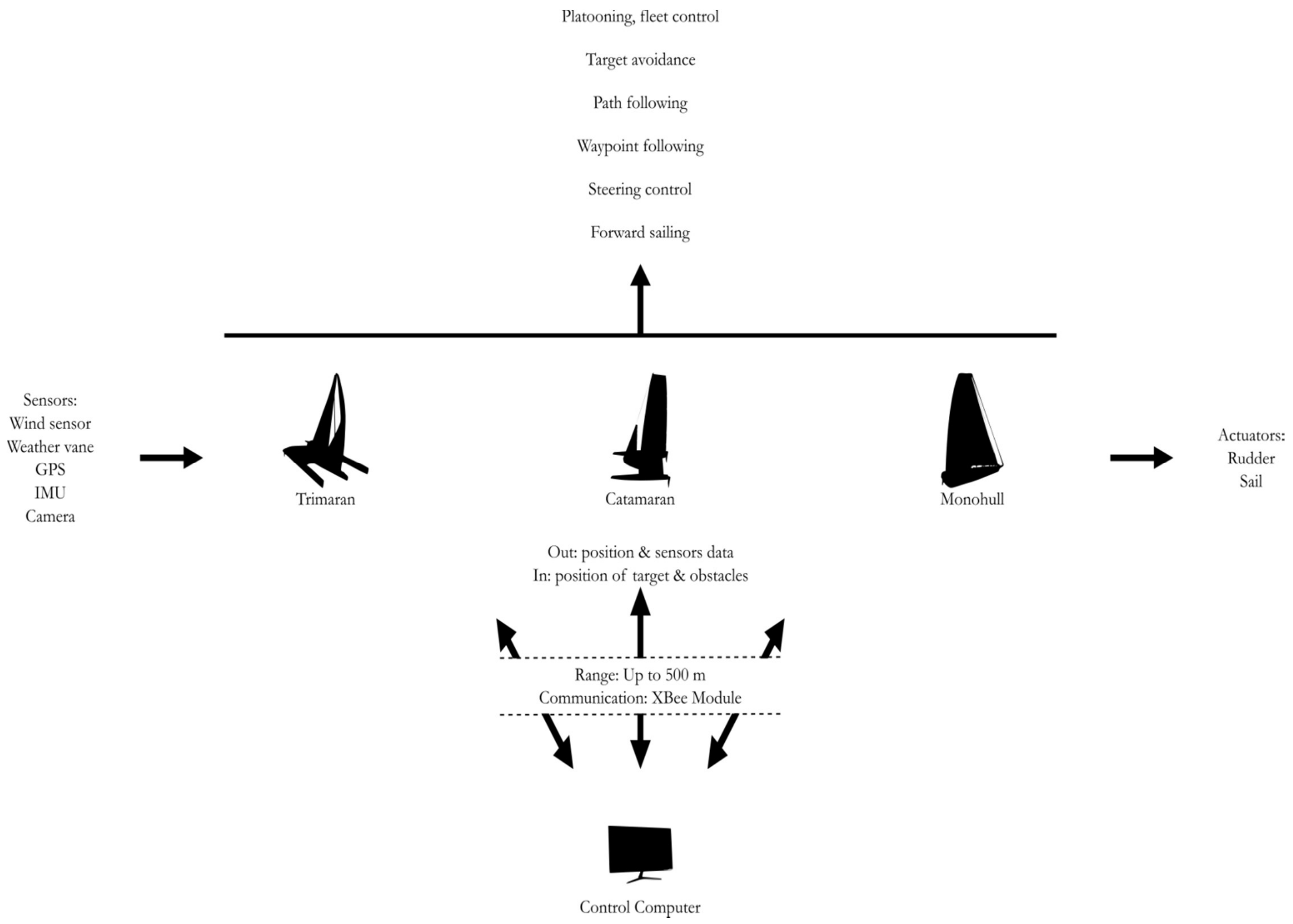


Figure 3: Synthesis of the expectations

To do this, three radio-controlled sailboats were available: a monohull, a catamaran and a trimaran. Each one was approximately one meter long. The idea was to equip them with sensors, communications modules and electronic boards (Raspberry Pi 3B+ and Arduino Mega) to make them able to sail by themselves and realise various tasks.

The supervisor put some constraints in order to assure the reproducibility of the design: it was necessary to avoid expensive products, and as much as possible avoid modifying the original architecture of the boats. These constraints allow to transfer or copy the project to implement it on other boats, so that it is easy to lead test performances on different hulls or quickly create a controllable fleet. Indeed, such a fleet with simple architecture would be nice to train students on control algorithms or for research purpose.

2. Architecture and work distribution

Controlling an autonomous sailboat involves the realisation of multiple tasks and the use of multiple devices. The first step of this project was therefore to agree on the architecture to be used, as simple as possible, and on the role of each teammate.

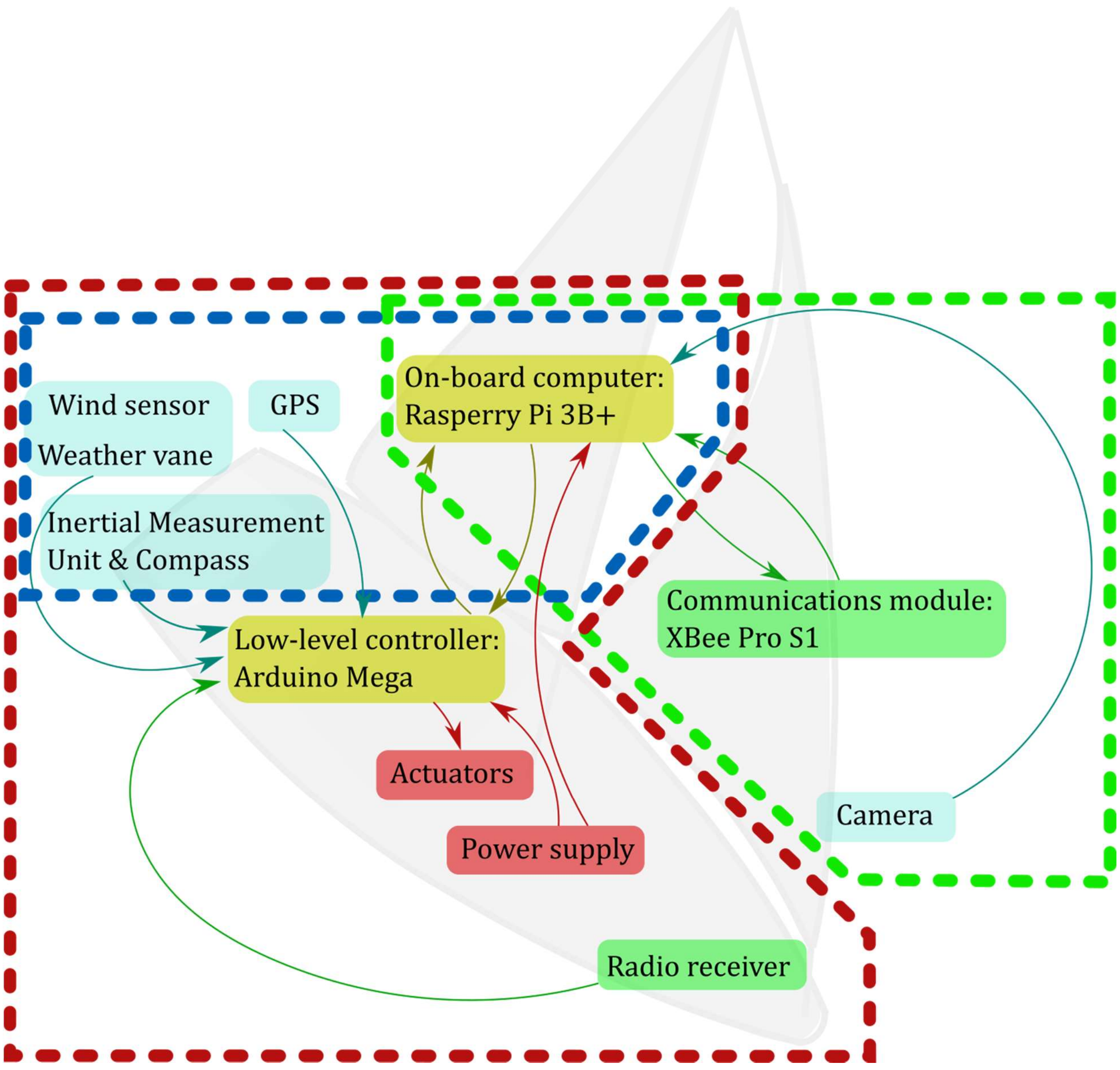


Figure 4: Architecture and work distribution

The first task to be addressed is to make all low-level components work together. This includes powering the electronic boards (Arduino, Raspberry Pi), collect data from the sensors and send commands to the actuators, along with creating a reliable path to exchange data with the Raspberry Pi board (mentioned as RPi hereinafter). For more security, this part also includes the integration of the original radio receiver in order to be able to take direct control of the actuators if needed, using the remote.

The second task is to improve the data coming from the sensors. After having chosen the sensors, this is done by inserting a filter layer between the raw data inputs and the controllers. The interest of this layer is to provide more reliable and stable data than using only the sensors. Each filter must be generated in accordance with the sensor to which it is attached and the reactivity wanted for the corresponding data.

One of the purposes of the project is to trigger different reactions of the fleet accordingly with what happens in its environment. The first task allows to react to wind changes and position changes, but it is also needed to detect external object that may require an action, either acquiring data or avoiding an obstacle. Thus, a camera was added to the sailboat to do this with the adequate software.

To perform fleet operations and allow complete remote-controlling, data exchange is needed between the boats of the fleet and an operator on the shore. On the one hand, depending on the mission, a boat may indeed require the position of the other boats or the data from their sensors. On the other hand, the operator may prefer to control the boats from a single keyboard rather than from a distinct remote for each boat.

Lastly, the coordination of all the tasks above is at the heart of the project in the controllers. These controllers must make the sailboats able to lead any mission the operator wants, from the direct control of the actuators with the keyboard to coordinated manoeuvres around specific location, using the different capabilities of the boats.

To carry out all these tasks, the team was made of three members, as shown in the previous illustration with the dotted lines. One allowed the low-levels controls (first task) and realised some of the controllers. The second took care of the filter layer and created the other high-level controllers. The third created the communications software accordingly with the hardware and took charge of the artificial vision capacities of the boats.

3. Communications

Sharing data is important when designing an autonomous sailboat because it allows you to keep an eye on its decisions. But it becomes crucial when you decide to implement fleet coordination because the decisions of all boats are then linked to the data relative to the others.

This necessity to have all boats share data, the implementation of computer-based remote-control and the wish to be able to launch specific commands without having to use Wi-Fi explains why so much time was dedicated to communications during this project.

a. Shared data

Each boat needs the data from the others, as mentioned before. But the more each boat communicates data, the more data must be processed and thus more time and power is consumed. Even if this is not a real issue when the fleet only contains a few boats, it is still better to choose wisely which data is relevant to be communicated and which data can be kept internally.

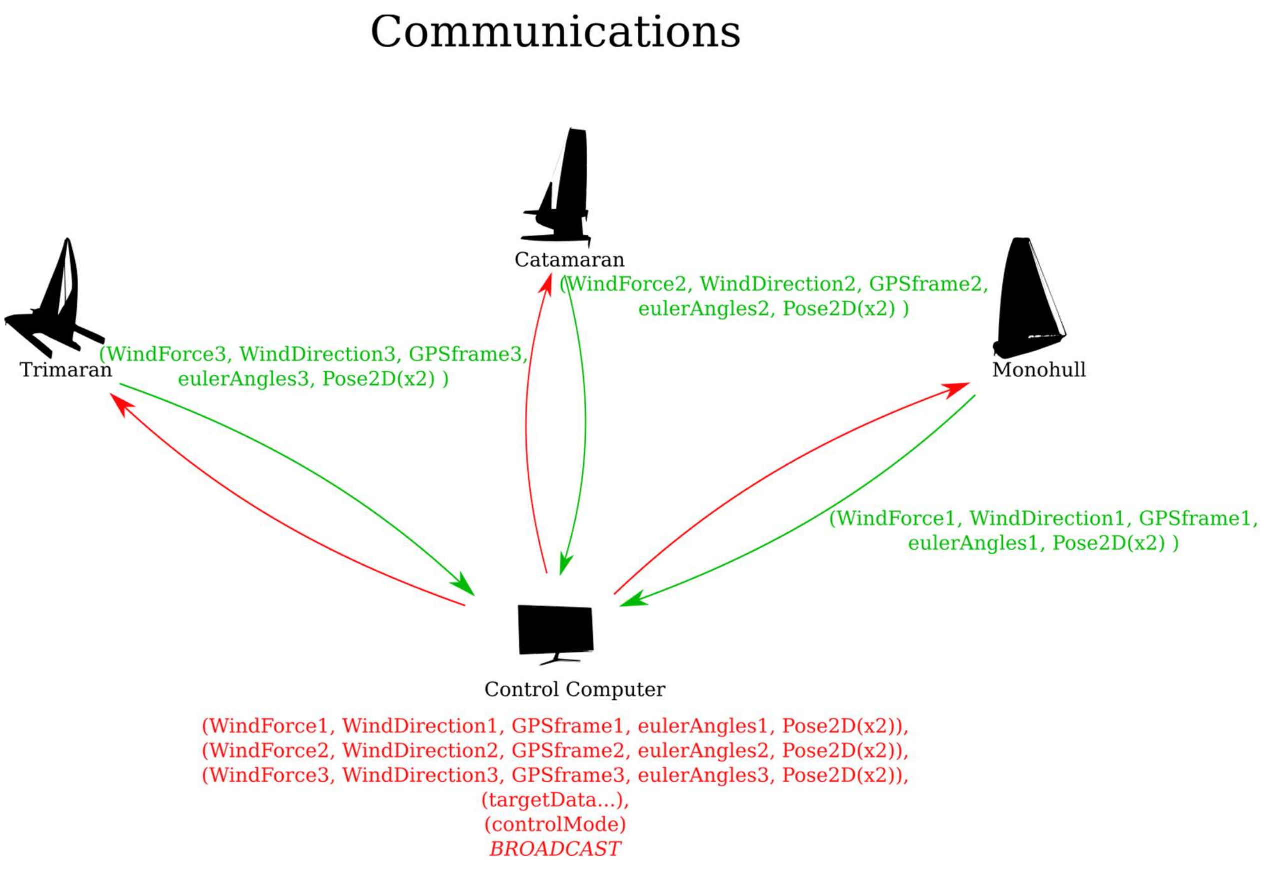

Communications
Catamaran
(WindForce2, WindDirection2, GPSframe2, eulerAngles2, Pose2D(x2))
Trimaran
(WindForce3, WindDirection3, GPSframe3, eulerAngles3, Pose2D(x2))
Monohull
(WindForce1, WindDirection1, GPSframe1, eulerAngles1, Pose2D(x2))
Control Computer
(WindForce1, WindDirection1, GPSframe1, eulerAngles1, Pose2D(x2)),
(WindForce2, WindDirection2, GPSframe2, eulerAngles2, Pose2D(x2)),
(WindForce3, WindDirection3, GPSframe3, eulerAngles3, Pose2D(x2)),
(targetData...),
(controlMode)
BROADCAST

Figure 5: Communications overview

As shown in the picture above, we chose to share six sets of data per boat.
The two first sets give measures of the wind so that if the fleet operates in a small area, each boat can improve the measures from its sensors or even keep sailing properly with broken sensors. This might also be an interesting piece of data for scientific research and it is better to have it stored on a computer on the shore rather than in the SD card of a sailboat.
Next set is the GPS frame right as the GPS outputs it. This is used for many purposes: it allows the operator to keep an eye on the fleet *via* a control window, it allows the other boats to possibly join the emitter and permits further study after the mission by replaying it.
Then, the Euler angles (especially the heading component) allow the other boats to predict movement of the emitter and facilitates for instance *platooning* manoeuvres. It might also be useful for the operator to detect issues. The idea is quite the same with the waypoints that are published by the boats: they correspond to its current objectives.
Moreover, the operator can himself send data to the boats through a variable that sets the control mode (autonomous, remote control, etc.…) and another that indicates things to be done (command for actuators, waypoints… in accordance with the control mode).

b. XBee devices

Radio devices XBee Pro S1 were used for this project. These are highly configurable long-range radio modules that can be plugged on an adapter so that they can be controlled with the USB ports of the RPi. It is otherwise possible to use them directly plugged on the Arduino board using an Arduino serial library, but we preferred the first option to avoid overloading the Arduino board and keep more control on the XBee devices.

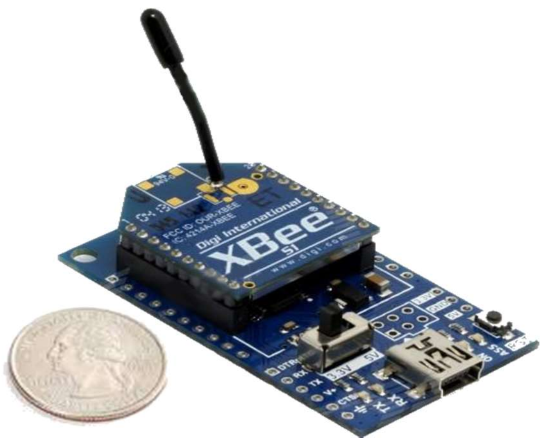


Figure 6: XBee module with adapter

These radio modules come with a user-friendly application (XCTU) that allow to reconfigure them. There are many options that may be changed, especially the type of network and the role of each device in this network. This setup can also be done directly in lines of commands using serial port, but this is much less intuitive.
According to their technical sheet[1], XBee modules have an operative range of about 1km, which make them perfectly suitable for controlling a fleet with an operator on the shore.

Using the USB adapter, the XBee device can be used with serial libraries included in *Python*. From there, every character string sent through serial port will be sent by the device to the other devices of the network *via* radio (except a few specific commands, which will be exposed later in this report). All that is to be done is therefore to create the character strings for emission and reception.

When an XBee device emits a character string, the corresponding receiver(s) receive one character after another without any distinction. This means that the beginning and end of the original character string are not known *a priori*. This also means that if two XBees are emitting at the same time, their messages will be mixed up. These issues require a robust message structure and a good synchronisation to be solved.

c. First version: four communication slots

For the setup part, we elected a basic architecture using one module called *Coordinator*, which is connected to the operator's computer, and one module per boat called *End Point*. This architecture is the simplest to use: *coordinator* can broadcast data to all connected *end points*, while these *end points* can only talk to *coordinator* and not between themselves. If the fleet were to grow in size, this architecture would quickly encounter speed issues as it uses a single communication channel and therefore requires a synchronisation of the modules to avoid mixing up several messages. As the size of the fleet is not expected to be higher than 5 boats, we did not consider the implementation of a more efficient but more complex network.

Given that the team had three sailboats available for tests, the first version of the communications system was built as a quick functional prototype. The main purposes with this prototype were to communicate data through XBees, receive valid data through XBees and connect to the global system.

[1] www.digi.com, « User Guide - XBee/XBee-PRO S1 radio modules », *https://www.digi.com/resources/documentation/digidocs/pdfs/90000982.pdf*, p. 121.

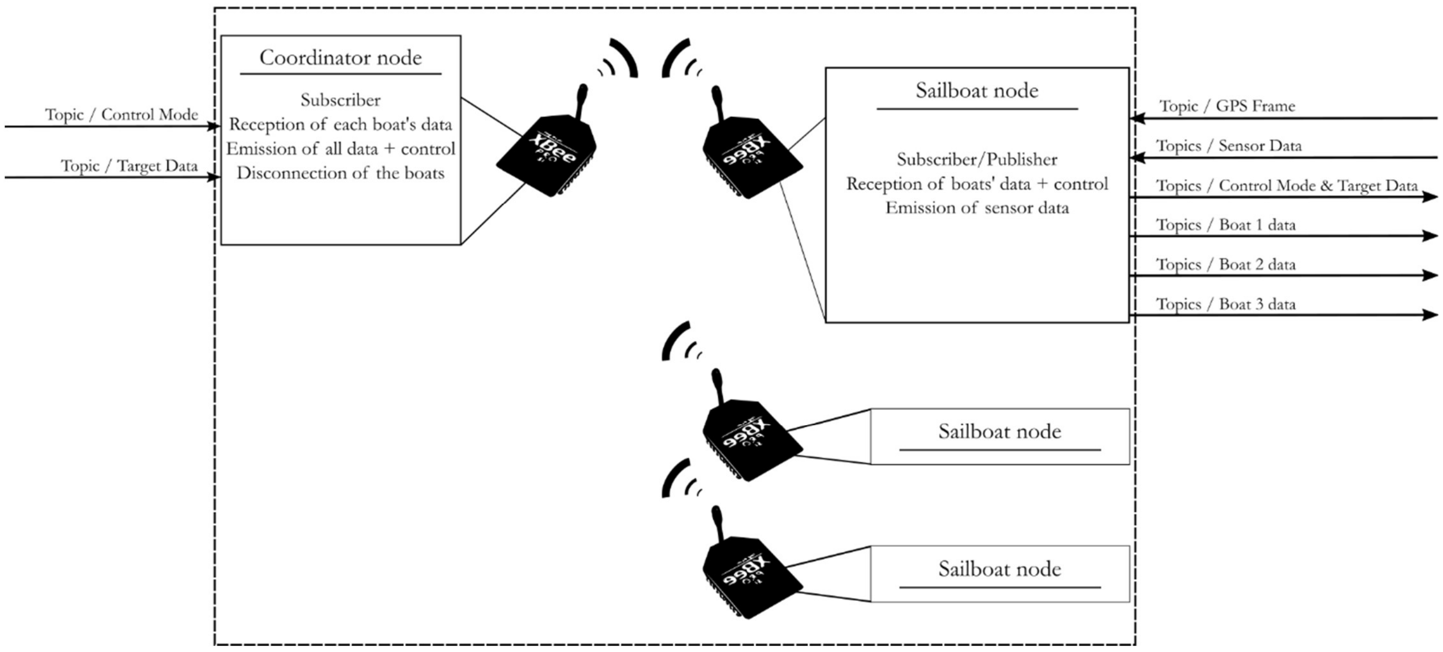


Figure 7: Communications architecture for first version

i. Connecting to the rest of the system

As this project is based on the use of *Robot Operating System* (ROS) with three packages[2][3][4] corresponding to the work of each of the three teammates, all data pass through channels called *topics*. The programs executed to complete specific actions (*nodes*) can either publish data in these topics, or subscribe to these topics to access the data.
Thus, connecting the communications software in the global system simply means:
- picking up the data to share in the topics where it can be accessed
- giving access to the external data in new topics, where controllers can access it.

The concerned topics are shown in Figure 7 above. Notice that the communications part of a sailboat (with an *end point* module connected) has six input sets of data, as detailed in Figure 5 (clustered in Figure 7 for readability purpose), and *3 * 5 + 2* = 17 output topics (5 for each *possible* boat and two for the operator commands).

As mentioned before, this was only a prototype built to perform remote communications, with no performance nor adaptation goals yet. This is why output channels are hard-coded and exist for three sailboats even if less boats are actually connected. Thus, the connection part was more developed in the second version.

[2] Matthieu Bouveron, *GitHub/Matthix7/plymouth_internship_2019*, s.l., 2019 [2019].
[3] Corentin Jegat, *GitHub/corentin-j/wrsc_plymouth_2019*, s.l., 2019 [2019].
[4] Alexandre Courjaud, *GitHub/AlexandreCourjaud/Stage2APlymouth*, s.l., 2019 [2019].

ii. Communicating data

As mentioned before, a robust message structure is required to ensure good remote communications with the XBee devices. This structure must allow to reconstruct the character string and check transmission errors.

The first step in creating the character string before sending it is to get the interesting data from the *topics* and convert them into character strings (using comas as separators if the data is a set of variables). This is easily done with *call-back function*s and global variables under *Python* ROS architecture. Then, theses character strings are assembled together with separators to create the core of the message, which contains the interesting data.
The case of the *Coordinator* is slightly more complicated as most of the interesting data are what the sailboats share, thus this data come from the XBee device and not from *topics*. The *Coordinator* first reconstructs the character strings coming from the sailboats (see Synchronisation below) and then assemble them in a single character string containing the data from all sailboats and from the operator. If nothing comes from the sailboats (connection issue, sailboat not connected …), the software is not blocked and fills the sets with default values.
Once the core message has been created with a separator between each set of data, we implement a security check: counting the number of characters in the message. This number is put at the beginning of the message with a separator. The last step before sending the message is to add start and end symbols to it.

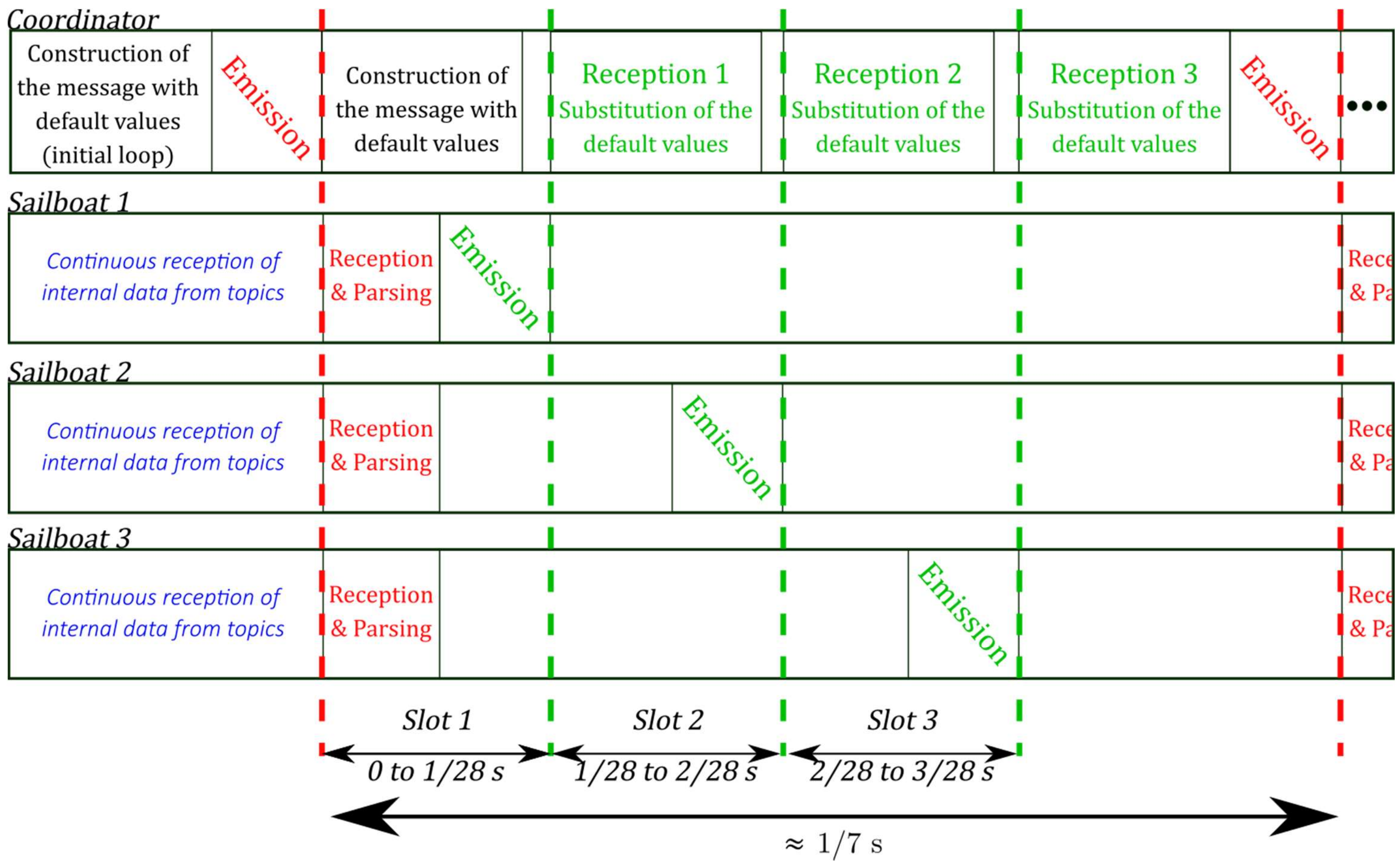
Receiving and decoding the messages logically follows the same steps in reverse order. The receiver, *coordinator* or *end point*, reads characters from its serial port until it finds a start symbol. From there, it assembles incoming characters in a character string until an end symbol. Then it removes these start and end characters and finds the checksum. After having checked if the message length is the same as indicated by the checksum, the remaining character string is parsed to publish the data sets in their corresponding *topics*.

iii. Synchronisation

The previous steps are executed multiple times per second (≈ 7 Hz). But as mentioned before, the messages sent by the XBee devices can be mixed up and therefore become unreadable. To avoid this, it is necessary to implement robust synchronisation so that the sailboats and the *coordinator* never try to talk at the same time. In this version, no connection phase was implemented so the process begins as soon as the coordinator software is launched.

Each transmission loop begins with a broadcast emission from the *coordinator*, as described in the previous section. This emission occurs whether or not data is received from the sailboats. This emission not only shares the common data to the whole fleet, but also serves as a clock signal to synchronise speaking times.

The reception of the message from the *coordinator* triggers a timer on sailboats' side. Before emitting its message, each sailboat waits for the number of speaking slots corresponding to its identifier. Speaking slots correspond to a quarter of the period of the transmission loop ($\approx 1/7$ s). XBee devices on sailboats are given IDs from 1 to 3 (remember, we only used three sailboats for tests). At the end of each slot, the *coordinator* overwrites the default values contained in the message in the spaces allocated to the corresponding sailboat. At the end of the third slot, it sends the message containing the data from all boats and a new loop begins. Depending on the time needed to parse the messages, the time for each slot may vary. This is taken into account by measuring this processing time and giving some margin.



Coordinator
Construction of the message with default values (initial loop)
Emission
Construction of the message with default values
Reception 1 Substitution of the default values
Reception 2 Substitution of the default values
Reception 3 Substitution of the default values
Emission
•••
Sailboat 1
Continuous reception of internal data from topics
Reception & Parsing
Emission
Rece & Pa
Sailboat 2
Continuous reception of internal data from topics
Reception & Parsing
Emission
Rece & Pa
Sailboat 3
Continuous reception of internal data from topics
Reception & Parsing
Emission
Rece & Pa
Slot 1
0 to 1/28 s
Slot 2
1/28 to 2/28 s
Slot 3
2/28 to 3/28 s
≈ 1/7 s

Figure 8: Transmission loop

d. Second version: n+1 communication slots

The second version of the communications software relies on the same principles as the first one. The network architecture is the same, with a *coordinator* and one or several *end points*. The aim with this second version was to make the communications scheme designed in the first version more adaptive, in order to be able to use any reasonable number of sailboats. Secondary goals include communicating more data and partial replacement of *SSH* protocol to launch commands.

i. Adaptability

In the first version, the ID of the boats that sent messages were known to be between 1 and 3, with no more than three connected boats at a time. Thus the message structure on *coordinator* side was fixed and was like:

[startCharacter][checksum]_1_dataBoat1_2_dataBoat2_3_dataBoat3_[endCharacter]

Knowing the ID of the sailboat that sent data during the current slot (ID sent by the boat), it was simple to fill the structure with the data.

With the second version, IDs could be any integer and more than three boats could be connected at a time (not too much though, as it slows the communication down). To manage this, the idea was to come back to a case similar to the previous one. So we implemented a connection phase during which each sailboat of fleet sends its ID to *coordinator* so that it can build a list of the connected sailboats. Once this list is built, the *coordinator* builds a linking dictionary to create relative IDs it will use for next steps and sends the list to all the sailboats so that they can do the same operation.

Example: Consider a fleet of two sailboats (IDs: 3 and 7) and a coordinator (ID: 0 by convention). The first step is to launch the coordinator software without omitting to tell it how much sailboats it should expect to connect. The connection phase begins. Launch the software n the two sailboats: they will start to spam the coordinator with their IDs. For each new ID the coordinator receives, it adds it to a list. Once this list contains two elements, the coordinator sends it to the two sailboats, therefore telling them that the connection phase has ended. Then, both the coordinator and the sailboats create a linking dictionary: {0: 3, 1: 7}. 0 and 1 become the relative IDs of the connected sailboats for software purpose. These IDs will be used to create lists dedicated to these boats and then store and access data in them.

The *coordinator* and the sailboats then create lists of publishers in which the publishers will be accessed using the relative IDs. As opposed to the previous version, we now have only the required publishers: six for each *connected* boat. Then the transmission loop begins just like in the previous version, except that each loop is divided into *n+1* slots instead of four (three sailboats and the *coordinator*) and there is a bit more data.

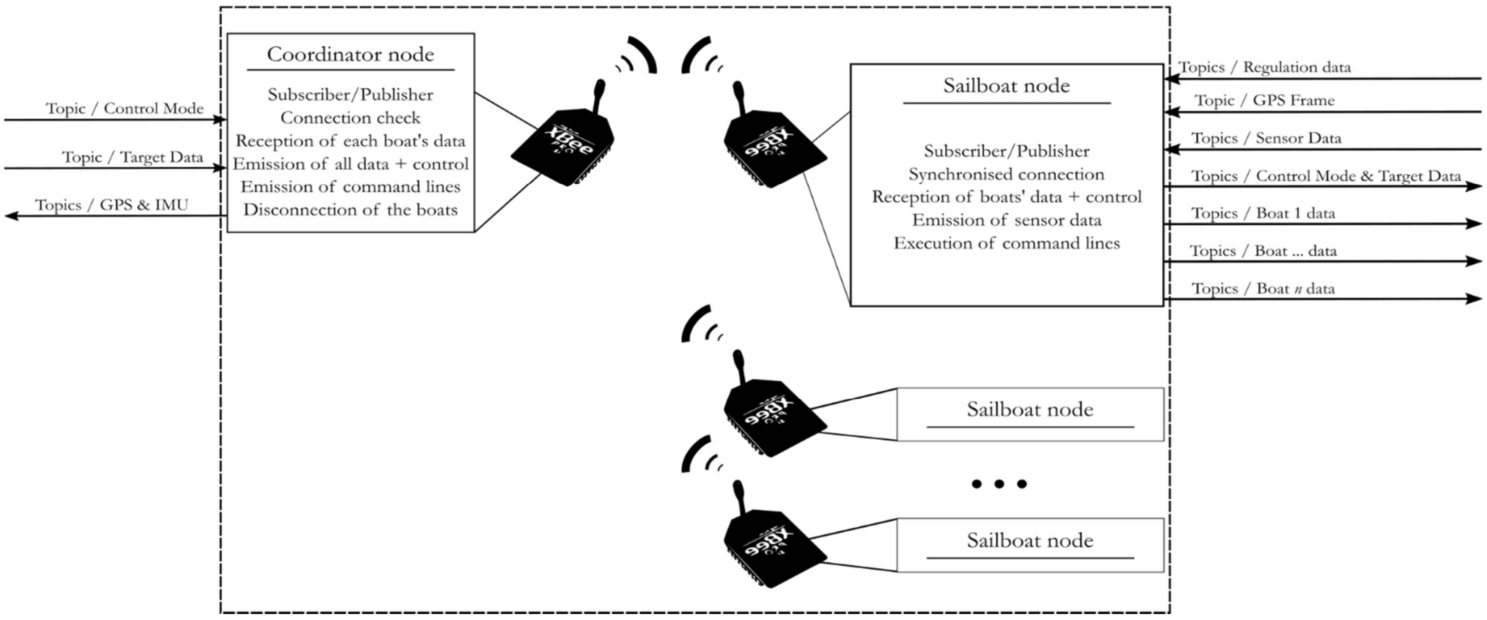


Figure 9: Communications architecture for second version

ii. Remote control

Remote control was implemented in the first version of the communications software, but faced limitations. Firstly, the commands chosen by the operator for the actuators were sent to all the sailboats of the fleet (meaning that they were all remote-controlled at the same time with the same commands). Secondly, the operator had to make sure he had clicked in a specific window for his keyboard interactions to be taken into account. The second version improved this part too with the creation of a separate node for human interaction.

The use of the *Python* library *pynput* allowed to monitor all the keyboard interactions, without any constraints for the operator. With this library, we can monitor all the keys of the keyboard and assign actions to them, whereas the first version used the ROS package *key_teleop* which limits the interactions to the arrows. From there, it became possible to send distinct commands to each boat. More precisely, each boat receives the commands for all boats, but these commands are organised in a dictionary using the IDs of the boats.

This new nodes allow much more flexibility for the operator. Indeed, he is able to control two distinct boats at a time with distinct commands using two pads (arrows and WASD) on his keyboard. Specific keys are dedicated to system actions such as putting a sailboat in remote-control or autonomous mode (one key per boat) or sending terminal-style command lines (see below).

To communicate the commands, to sets of data are used:

- The first consists in a dictionary linking boats' IDs with their control mode (0: autonomous, 1: remote-controlled, 2: command line)
- The second is also a dictionary, linking boats' IDs with the commands for their actuators (rudder and winch). These are taken into account only of the corresponding control mode is at 1.

Once the commands are sent through the XBee devices by the *coordinator* and if the operator wants the sailboat to be remote-controlled, they are received by the communications part in the sailboat, which then sends them to a control layer that directly acts on the actuators instead of calling a controller.

iii. Command launcher

A new functionality implemented in the second version is to partially replace the *SSH* protocol. *Partially* only, because we wanted the operator to be able to launch terminal-style command lines through the radio devices, but could not assign time enough on this part to be able to monitor what really happens in the on-board terminal.

This functionality is realised using the libraries *pynput* and *pyautogui* on the *coordinator* side and the library *subprocess* on the sailboat side. The first one monitors any actions on the *insert* button on the keyboard, the second one displays a window where the operator can type the command he wants to be executed. This is done in the same node as the remote control and uses the same messages.

Pressing the *insert* button displays the prompting window. In this window, the operator types first the ID of the receiver (broadcast is also available) and then the terminal-style command as he would write it in a normal terminal. Some additional features are available to kill running nodes.

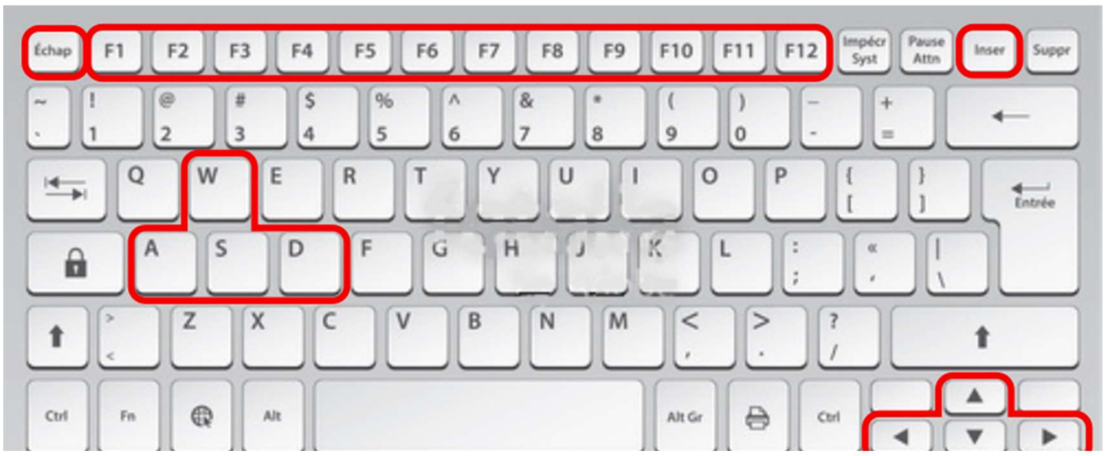


Figure 10: Keys used for remote control and command launcher

Example: As before, consider a fleet of two sailboats (IDs: 3 and 7) and a coordinator (ID: 0 by convention). Considering that both boats are already running autonomous controllers, the operator wishes to change the behaviour of boat 7. The figure below explains what happens when he launches a command to do so by changing the controller (simplified version).

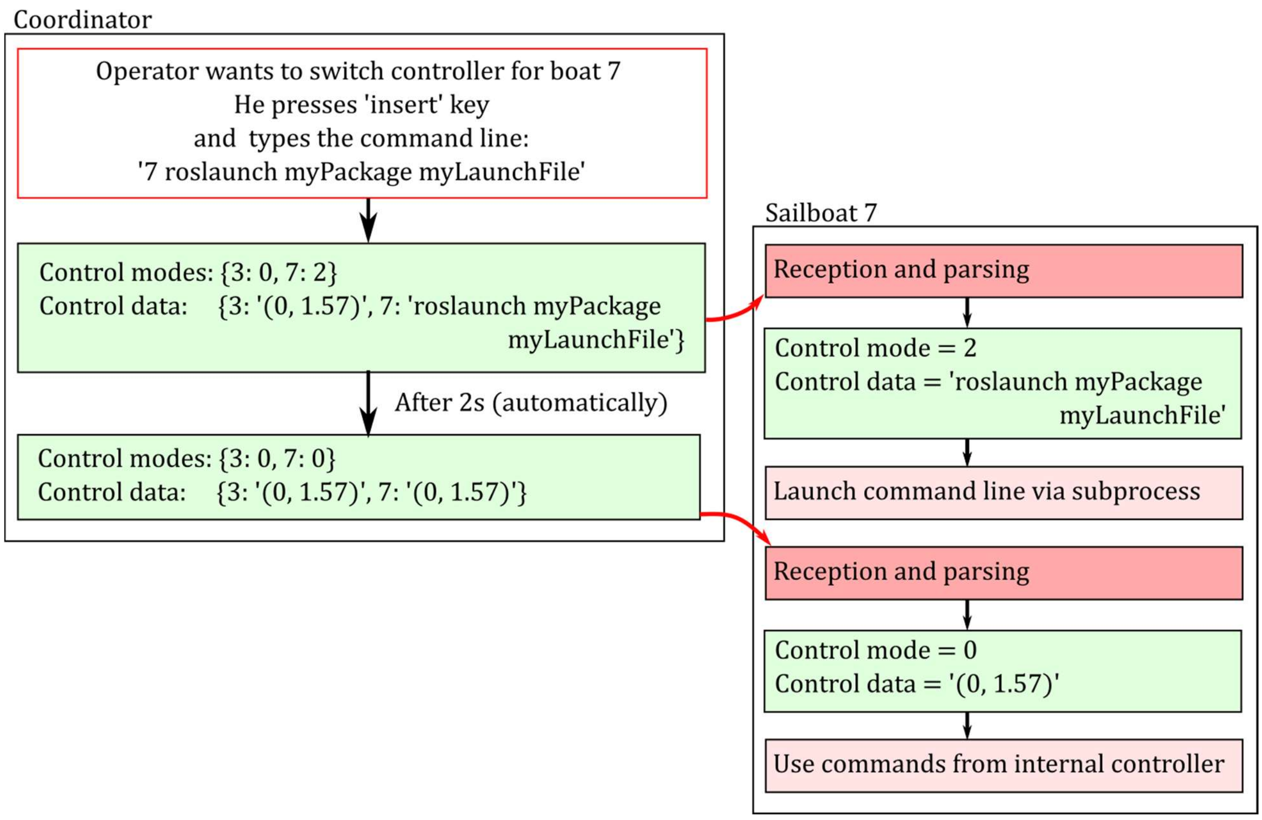

Coordinator
Operator wants to switch controller for boat 7
He presses 'insert' key
and types the command line:
'7 roslaunch myPackage myLaunchFile'
Control modes: {3: 0, 7: 2}
Control data: {3: '(0, 1.57)', 7: 'roslaunch myPackage myLaunchFile'}
After 2s (automatically)
Control modes: {3: 0, 7: 0}
Control data: {3: '(0, 1.57)', 7: '(0, 1.57)'}
Sailboat 7
Reception and parsing
Control mode = 2
Control data = 'roslaunch myPackage myLaunchFile'
Launch command line via subprocess
Reception and parsing
Control mode = 0
Control data = '(0, 1.57)'
Use commands from internal controller

Figure 11: commands transmission

The work described in this part resulted in the creation of a platform that makes possible fleet coordinated actions and allows the operator to monitor/control the behaviour of each sailboat. Even though no real graphical user interface was designed, the use remains quite intuitive as the user is guided in the terminal for each of his actions. Moreover, a 3D representation of the fleet makes monitoring easier (made by a teammate).

The next part of this report tackles a totally different topic: artificial vision. Giving a sense of vision to a sailboat is a very interesting thing as it opens fields for new measurement or even improve measurement from other sensors.

4. Image processing

a. Horizon detection

An auxiliary goal for this part was to make it as independent as possible from the other sensors. This makes it more easily reusable for future projects, more secure in case of a failure in the rest of the system, and makes tests a lot easier. However, it requires more work and does not work in all situations (see *Tests* chapter).

Basically, the only sensor on which the artificial vision relied was the IMU. Indeed, the main goal for a camera is to find headings to interesting objects. However, a sailboat has a lot of roll and this fact affects the heading estimated by the vision software. To compute the true heading leading to a target, the roll angle is needed. This angle may be obtained from the IMU, but another solution is to compute it from the horizon line. To achieve the auxiliary goal, the second solution was chosen in a first approach.

The figure below details the main steps that lead to finding the horizon line and therefore the roll angle of the sailboat from an initial picture taken by the camera. The base image here is of course artificial, created specifically to show all aspects of artificial vision tackled during the internship, but was processed by the same algorithm as during real missions.

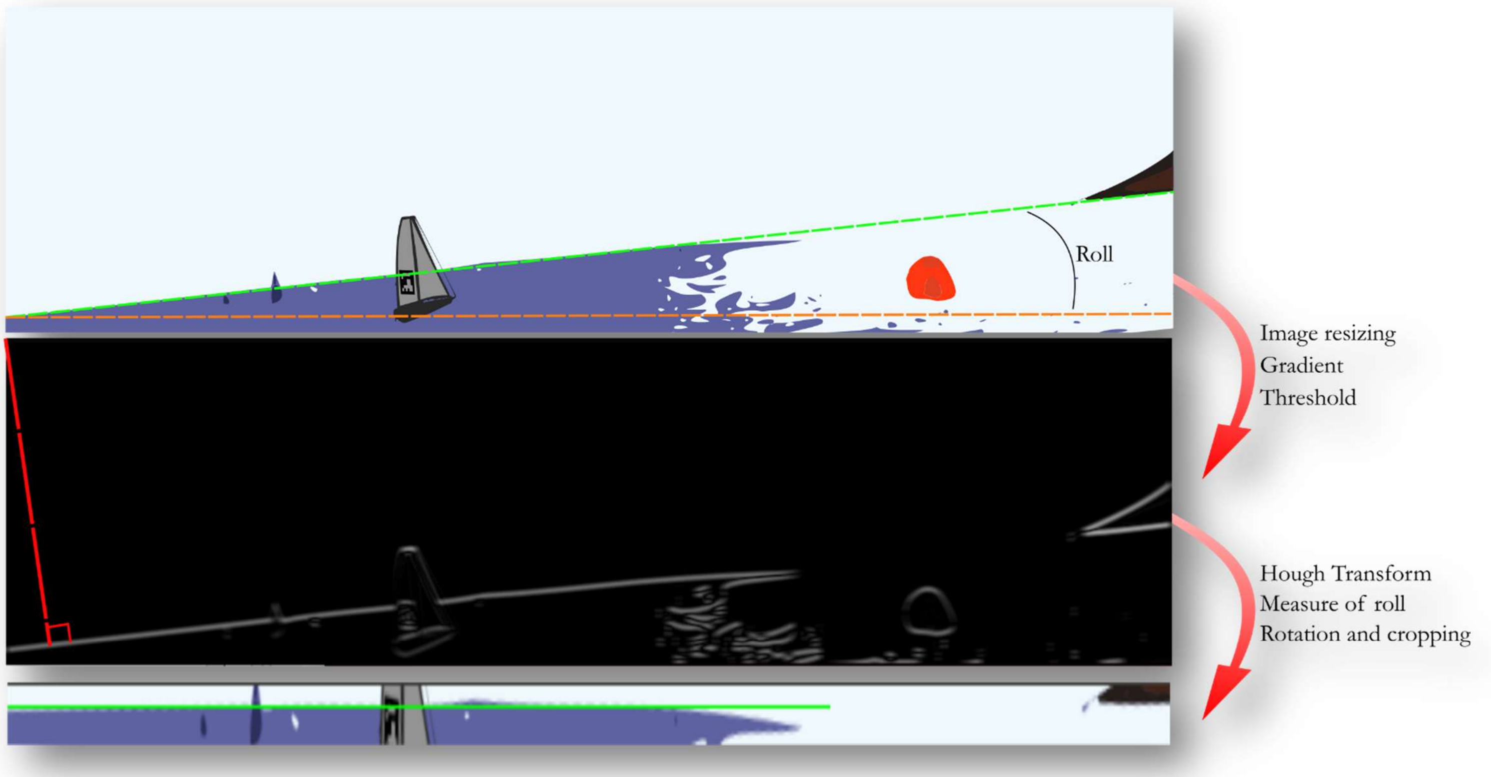


Figure 12: Looking for the horizon line

The top picture of Figure 12 represents the image as it is provided by the camera. As the camera is fixed on the bow of the sailboat, 10cm above the deck, the image has the same roll angle as the sailboat. This angle is highlighted on the picture, with the horizon line in green.
Using a Sobel-style custom-tailored gradient filter, the vertical lines are discarded and we get the middle picture. This step assumes that the sailboat keeps a reasonable roll angle (still up to ≈60°).
Then a Hough transform is applied on this binary picture. Hough transform for lines measures the number of white pixels in a line-shaped mask, iterating until the mask has been on all possible positions with all angles. It can therefore build a chart where numbers of white pixels in the mask are linked with the position and the angle of the mask. More information about this algorithm can be found on OpenCV online page[5]. The interesting thing is that this algorithm directly outputs the angle of the line (in fact the angle of the red dotted line), meaning that if it can be assured that this line is actually the horizon then the algorithm succeeds. In the example, even if the horizon line is not perfectly clear (mast, lighthouse, reflections, mountains …) the algorithm is robust enough to detect it and crop the image around it. The green line was drawn in the algorithm too, showing how accurate it is.

However, the reliability of the software is not enough to ensure the success of the mission: it also has to be efficient. The Hough algorithm is quite efficient as implemented in OpenCV library, but still takes too much time to deliver enough real-time data. Indeed, early tests using Raspberry board and camera showed that the maximum processing speed was at two images per second, which is quite slow. An efficient way to improve speed is to make the assumption that the roll angle of the sailboat cannot change heavily in the meantime between two pictures. Therefore, it is possible to search the horizon line close to where it was found during the previous iteration of the algorithm. This is done by saving the data related to the line at time *t,* then crop the image shot at $t+dt$ using the stored data and use this cropped image instead of the original image in the algorithm described above. Doing this reduces the number of pixels to be considered in Hough algorithm and therefore increases significantly the speed (≈ 50% faster).

The outputs of this module of artificial vision are:

- Horizon line localisation in the image (for next iteration)
- Image cropped around the horizon line (for next section: masts detection)
- Roll angle for the computing of true headings in next steps

[5] *Hough Line Transform — OpenCV-Python Tutorials 1 documentation*, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html, (consulté le 24 septembre 2019).

b. Masts detection

A possible interesting thing to detect for an autonomous fleet is the presence of other boats in the navigation area. Considering that the camera is almost at water level, any floating object should be seen at the horizon line level. This module is called *masts detection* because its first goal is to detect other sailboats, but in fact it should be able to detect anything that crosses the horizon line.
Using the cropped image delivered by the previous module, a gradient filter can be used to discard horizontal lines and keep only vertical lines. From there, the Hough line transform allows to identify the main lines that remains in the binary image obtained with the gradient filter, and the intersections of these lines with the horizon line give headings that lead to floating objects. This heading is computed from the number of pixels that separate the points from the centre of the image, multiplied by a resolution factor calculated before in a calibration part.

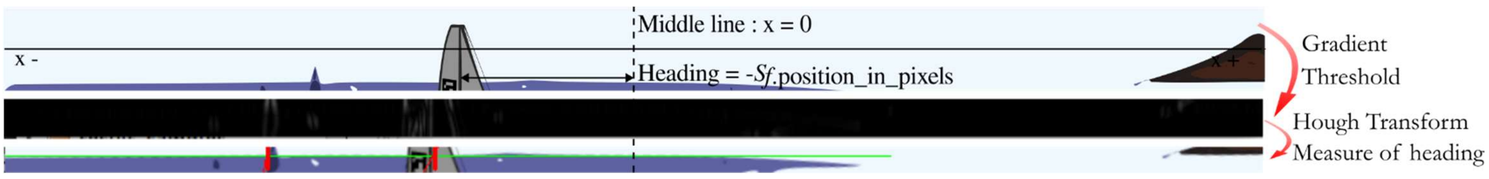


Figure 13: Looking for floating objects

In the example above, the top picture correspond to the cropped image returned by the horizon detection module. One may notice that the sailboat is clearly visible, but is not the only thing that crosses the horizon line. The aim of this algorithm is to be able to distinguish punctual objects (sailboats, lighthouses …) from the landscape (trees and landforms included). Thanks to a reliable chain of operations, the goal can be achieved in a large variety of situations. Here, the identified objects (lighthouse and sailboat) are marked with a red vertical line.

An interesting thing about Hough transform as implemented in OpenCV is that in fact it does not only return one single line but a list of lines ordered by their filling (number of pixels in the mask). In the previous module only the first one, most filled, was considered as being the horizon line. This time, multiple lines must be kept. However, a single object often generate several lines so some precautions are needed to detect each object only once.

c. WRSC vision tasks

Even if it was not the main objective of the internship, the team took part in the *World Robotic Sailing Championship* in China with one of the boats (the monohull). It did not require any changes in the system as this challenge requires abilities that an autonomous fleet requires too. The only exception was in the detection of particular markers, which we implemented in the artificial vision software.

The first type of objects to detect were coloured buoys. As these buoys were used as marks for a station keeping challenge, positioning them with a good precision seemed interesting so we added an estimation of distance.

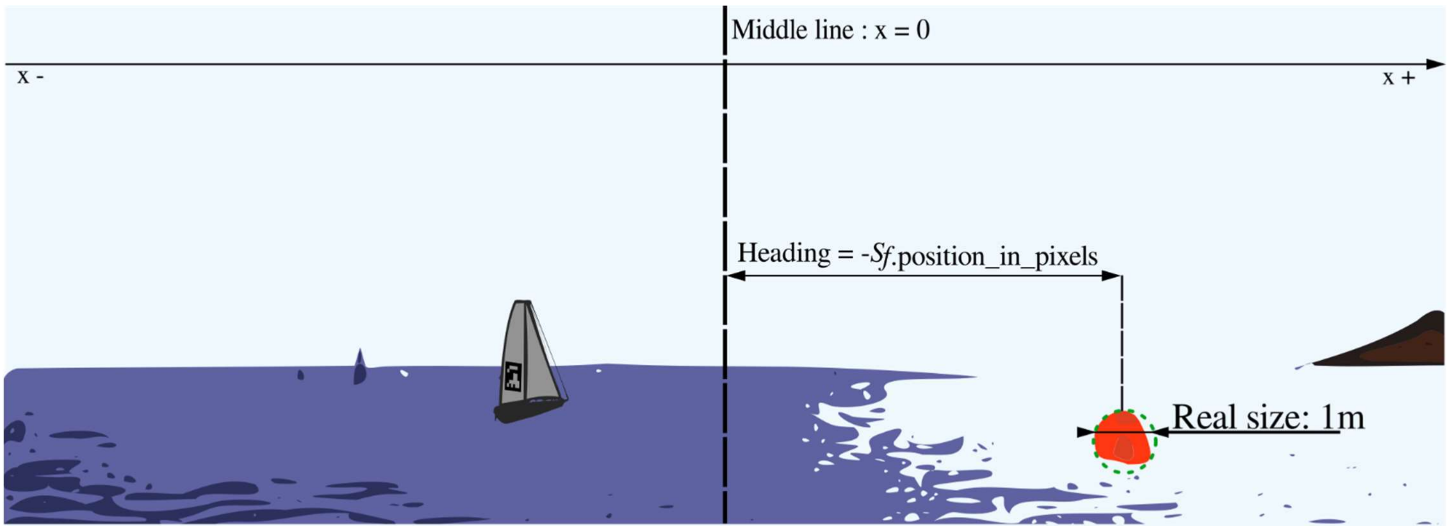


Figure 14: Looking for buoys

Once again, the S_f constant is used for the computation of the heading. This constant is the angle in radians that a pixel represents. Using trigonometric formulas and knowing both the real radius of a buoy and the radius in pixels it takes on the picture, it becomes easy to compute an approximation of the distance that separates the buoy from the camera. However, this approximation relies on the detection of all pixels that are part of the buoy, included in case of reflections, shadows, etc. Thus, the estimation of distance is a lot noisier than the estimation of heading and this must be taken into account for the realisation of the positioning filter that comes downstream (not in this report).

The second type of objects to detect were printed markers (ArUco markers and April Tags). These markers were positioned an all the sailboats in the navigation area during a *hide & seek* challenge. In this challenge, two teams were opposed. Each sailboat had to complete round trips back and forth between two buoys and detect the opponent sailboat's marker to mark points.

Figure 15: Example of ArUco marker

We implemented the two types of detection (ArUco and April Tags) in the machine vision software. A specific OpenCV library is dedicated to the detection of this type of markers, so we used it for the ArUco markers. A few lines are sufficient to detect markers in an image, store the image and identify the position of the marker in relation to the camera. Unluckily, the April Tags detection was implemented in version 4.0 of OpenCV, but version 3.4 was used for this project and it appeared too complicated to change. Thus, a dedicated package[6] was used for this.

5. Modularity and agile methods

From the beginning, this project was designed to be very modular and followed agile development style. Indeed, the versioning of communications software made it possible to focus on essential communications points only so that we could have it working for the early tests with the rest of the system. It is in a second step that non-essential features were added to reach the final communications program. Similarly, the machine vision software is actually built on five P*ython* files: four specialised modules (each of them being independently testable) for the four tasks detailed previously and the image acquisition, and one synchroniser that calls functions of these modules when needed. Calling or not the different modules can be done with setup variables, making it very adaptable to the real mission needs. Moreover, with this method each module can be improved independently without putting the whole software at risk.

6. Tests

Each development stage was tested in both the communications and machine vision programs. The communications part was quite easy to test inside the working room as all devices were available and it was easy to simulate the data coming from the rest of the system. The only thing that could not be tested inside was the range of communications. However, outside tests

[6] *GitHub/AprilRobotics/apriltag*, s.l., AprilRobotics, 2019 [2018].

with the entire system connected showed that some additional security were necessary to avoid occasional crashes.
Testing the artificial vision software was a little more complicated and required to adapt the program so that it could be switched from camera to video input. Even though some tests were still conducted outdoors especially to check the acclimatisation to light conditions, the majority if tests for this part were carried out with videos of maritime landscapes. These videos were chosen to give an overview of what a sailboat could observe during it missions.
Before the *World Robotic Sailing Championship*, a video was issued showing the navigation area. The test carried out using this video showed that the software would not be able to locate the horizon line, putting at risk all headings computations. This inability is due to the fact that the competition took place in small lake in a city centre with walls and buildings all around the water, which is a landscape too far from what was expected for the sailboat. We therefore decided to replace the original roll computation with the data provided by the IMU.

During the internship we had time for five tests in real conditions with the full system. No major issues occurred and we never lost any of the boats because we were always able to remote-control them *via* XBees or remote. Concerning the parts described in this report, one issue occurred that was hard to solve; some hardly visible damages in the camera cable made the images abnormally bright. We therefore protected it and it worked for next tests. The last of them was the *World Robotic Sailing Championship* itself, where our system proved to be reliable enough to complete all the challenges and finished 7th out of 22, first non-Chinese.

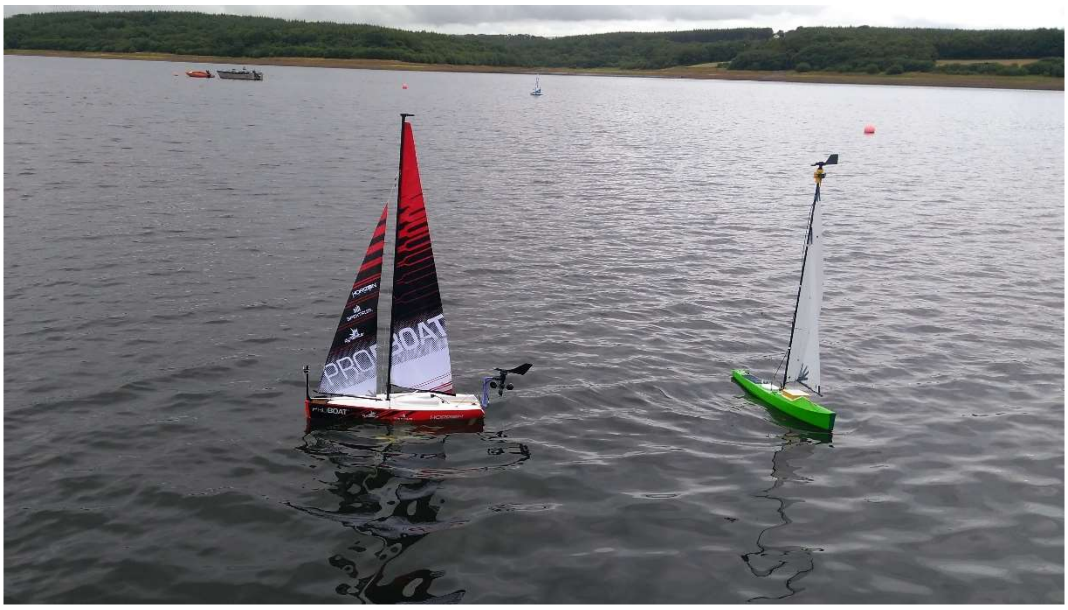


Figure 16: Two sailboats during the first outdoor test

7. Next steps

Even though the current system is fully operable, many things could still be improved or added. Indeed the last of the three months was mostly dedicated to make sure everything would work as expected for the *World Robotic Sailing Championship* so only little time was given for fleet-specific development.

On the one hand, improving the monitoring ability of the operator would probably be an interesting thing. This could be done by displaying some more information from the boats on his screen and adding an error feedback to the command launcher. Indeed, the command launcher is a really powerful tool as it has much more operating range than a standard Wi-Fi router and is directly included inside the system, but currently only the behaviour of the sailboats allow to see whether an error occurred or not. Improving the communication speed could be a good idea too in order to manage a bigger fleet.

On the other hand, one drawback of the current network architecture is that the missions that require fleet coordination can be done only a radius of ≈300m around the *coordinator*, meaning that there are still restrictions to its autonomy. Thus, a possible improvement would be to adapt or change completely the architecture of the network so each boat can communicate directly with others and the operator can connect or disconnect for monitoring purpose whenever he wants.

Concerning the artificial vision software, the current version uses a thread to grab pictures from the raspberry camera. It could be interesting to compare the performances of this architecture with dedicated nodes that already exist with ROS.

Eventually, one interesting objective could be to make the software able to detect more objects in the water, specifically pollution signs.

Conclusion

Although the share of work allocated to the championship was higher than initially planned, most of the original objectives were met. In view of the performances achieved in the championship, it can be said that each sailboat equipped with the system developed during the course has the individual qualities expected. Although the majority of the tests were conducted on monohulls, the first outdoor test showed that the system could also be installed without problems on a catamaran.

Artificial vision software is part of the sailboat's individual skills and seems reliable and robust for maritime environments. It might have been good to conduct more tests in such environments in real conditions, but for safety reasons our yachts were mainly sailing in relatively closed areas.

However, fleet performance has not really been tested, but some functionalities have been implemented. In particular, the communications software is fully operational despite the few limitations detailed above. It allows each boat in the fleet to have access to all the interesting information from the others, so it remains to develop controllers using this data to give coordination skills to the sailboats.

This internship was an excellent opportunity to rediscover sailing under a robotic approach. Moreover, the time devoted to artificial vision made it possible to complete the courses taught in the second year at ENSTA Bretagne. Added to the work done on inter-system communications, this internship proved to be very instructive and gave me new areas of expertise.

List of figures

Annexe

Assessment report

ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – *Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE*
☎ 00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* University of PLymouth

Adresse / *Address* Drake Circus, plymouth, Devon, PL48AA, UK

Tél / *Phone (including country and area code)* +44 01752586157

Nom du superviseur / *Name of internship supervisor* Jian Wan

Fonction / *Function* Lecturer in control systems Engineering

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

Nom du stagiaire accueilli / *Name of intern* Bouveron Matthieu

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
Please attribute a mark from ***A (excellent)*** *to* ***F (very weak).***

MISSION / *TASK*

❖ La mission de départ a-t-elle été remplie ?
Was the initial contract carried out to your satisfaction? [✓ A] B C D E F

❖ Manquait-il au stagiaire des connaissances ?
Was the intern lacking skills? ☐ oui/*yes* [✓ non/*no*]

Si oui, lesquelles ? / *If so, which skills?*

ESPRIT D'EQUIPE / *TEAM SPIRIT*

❖ Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*

[✓ A] B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here*

Version du 05/04/2019

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

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

[✓ A] B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* ______________________

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? A B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? [✓ A] B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* ______________________

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? [✓ A] B C D E F
Was the intern open to listening and expressing himself/herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* ______________________

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : [✓ A] B C D E F
Please evaluate the technical skills of the intern:

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?
Would you be willing to host another intern next year? [✓ oui/*yes*] ☐ non/*no*

Fait à / *In* Plymouth, le / *on* 30/08/2019

Signature Entreprise / *Company stamp* [illegible]

Signature stagiaire / *Intern's signature* [illegible]

School of Engineering
University of Plymouth
Drake Circus
Plymouth
Devon
PL4 8AA

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