Internship assistant engineer

subject of the internship

Control of Autonomous sailboats

Author :
Mamadou Dembele
Robotics engineering student,
ENSTA Bretagne

Internship tutor :
Dr Jian Wan
Lecturer in Control Systems Engineering,
University of Plymouth

September 8, 2020

Contents

List of Figures

Acknowledgements

To begin I would like to thank my internship tutor, Dr Jian Wan, Lecturer in Control Systems Engineering at the University of Plymouth for his availability and kindness during this internship. In a context of telework, Mr Wan encouraged me to work and motivate me constantly.
I would also like to thank Mr Jaulin, who offered me this internship.

Abstract

This document is about my telework internship with the University of Plymouth. The subject was the control of autonomous sailboats. It took place in two parts. The first part was devoted to the development of so-called deterministic algorithms. That is to say, algorithms inspired or using control laws from robotics books. The second part was devoted to solving one of the problems treated with deterministic algorithms by artificial intelligence methods. In a context of health crisis, not having been able to make the trip to Plymouth, all the algorithms developed were tested in simulation.

Résumé

Ce document traite du stage que j'ai effectué en télétravail avec l'université de Plymouth. Le sujet portait sur la commande des voiliers. Il s'est déroulé en deux parties. La première partie a été consacré au développement des algorithmes dites déterministes. C'est-à-dire des algorithmes s'inspirant ou utilisant des lois de commande issues des livres de robotique. La deuxième partie a été consacré à résoudre un des problèmes traité avec les algorithmes déterministes par les méthodes d'intelligence artificielle. Dans un contexte de crise sanitaire, N'ayant pas pu faire le déplacement à Plymouth, tous les algorithmes développés ont été testé en simulation.

Introduction

Sailing ships have long been the means of transporting people and goods in the seas and oceans. Starting with the industrial revolution, they were gradually replaced by steamboats. Today, they are only used for entertainment and sport [1]. But sailboats are finding a new place in the field of marine robotics. They consume less energy and pollute less than other unmanned surface vehicles (USM). But automating sailboats is not an easy thing. Indeed, the environment in which sailboats are built is made up of uncontrollable and unpredictable phenomena such as wind, currents, waves... This makes it very complicated to control them.

My internship deals with this problem of controlling sailboat robots. The objective of the internship is to develop robust control algorithms by deterministic and artificial intelligence.

1 sailboat

1.1 Description

Sailboats are sailing boats whose engines are the sails and fuel the wind. More precisely, sailboats are made up of :

- **A hull**: it is the first component of a sailboat and of a boat in general. Its role is to ensure the buoyancy and watertightness of the sailboat. Sailboats can have a single hull: monohulls, two hulls: catamarans or three hulls: trimarans.

- **A sail**: This is the engine of sailing boats. On the water, the wind exerts a force perpendicular to the sail no matter what angle it comes from. It is this force that makes the boat move forward.

- **A centerboard**: which is often associated with a keel (which prevents the boat from heeling), allows the boat not to drift. Indeed when the wind exerts a force on the sails. This tends to make it drift. The presence of a daggerboard, creates an anti-drift force allowing the boat to go straight ahead.

- **A Rudder**: which allows to control the boat. Placed at the rear of the boat, it controls the course to be followed.

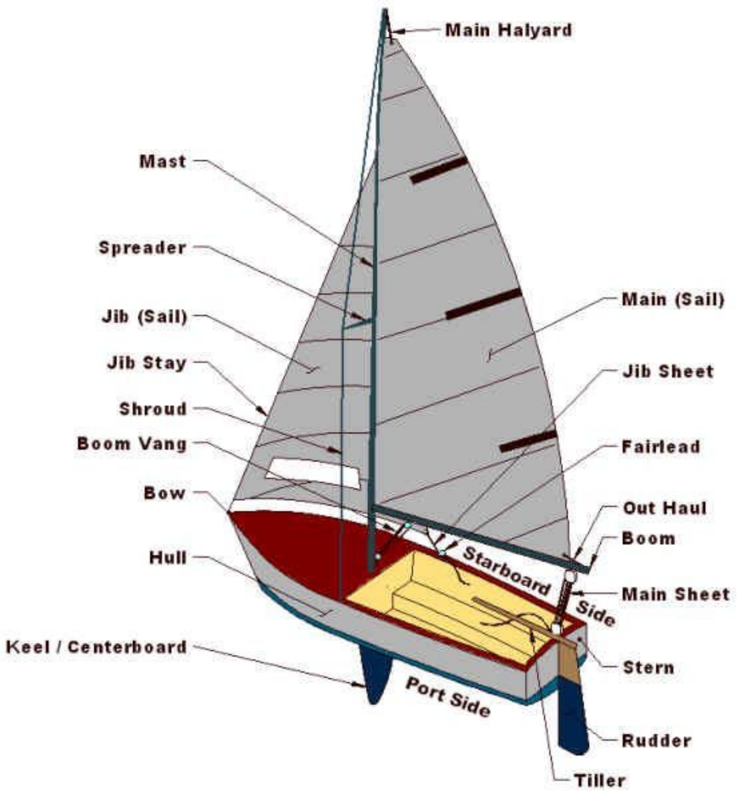


Figure 1: Different part of sail

1.2 Point of sail

On lake, sea, or ocean not all directions are navigable. Sails do not work if the boat is pointing directly into the wind. In reality the sail will only work if it is at an angle of more than 45 degrees to the wind.

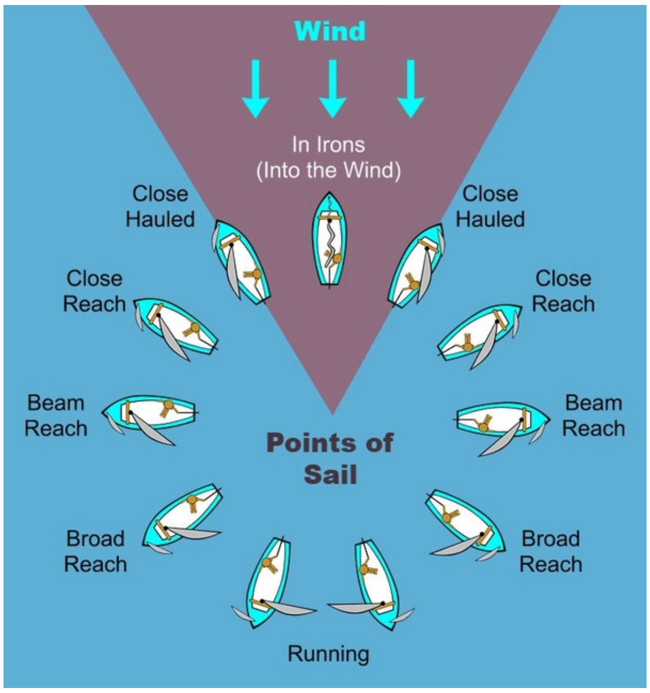


Figure 2: Point of sail

1.3 Modelisation

To simulate our sailboat, we need a model represented the state equations.
The sailboat can be described by the following state equations [3].

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

(x, y) corresponds to the coordinates of the centre of gravity of the boat, θ its orientation, δ_s and δ_r are the angles of the sail and rudder, a the speed of the wind, v is its forward speed, ω is its angular speed, fs is the force of the wind on the sail, fr is the force of the water on the rudder

2 Control of the sailboat by deterministic methods

Numerical simulation is a tool widely used in robotics. It allows the development of control algorithms during the design phase of the robot. Or it allows to limit the case, or to easily debug its code.

2.1 Basic algorithms

2.1.1 Heading following

After modeling the sailboat, the first command that could be implemented is the head following.
The law of control used is the proportional control, the sailboat have two inputs that we have to control. For the input corresponding to the rudder,

I used the sawtooth function that prevents 2-π jumps.This sawthooth function can be defined in two ways: either with trigonometric functions$(2 \times \arctan(\tan(x/2)))$or with the expression $(x+\pi)\%(2\times\pi)-\pi$. So the expression of this command is:

$$u_{rudder} = k_{rudder} \times sawtooth(\theta - \bar{\theta})$$

.

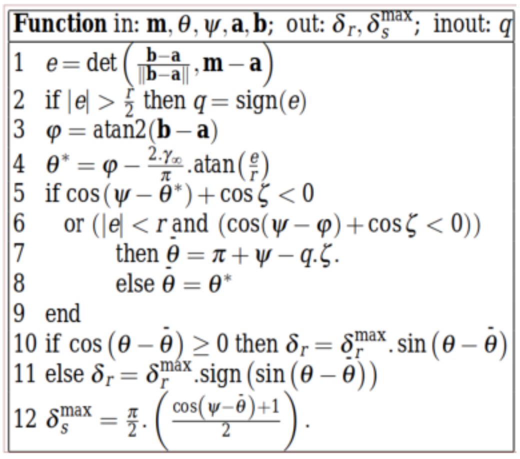
Concerning the control law for the command of the sail angle, I used the formula from the book Mobile robotic of professor Luc Jaulin [3]:

$$u_{sail} = k_{sail} \times \pi/4 \times (\cos(\psi - \bar{\theta}) + 1)$$

ψ being the wind angle, θ the heading of the sailboat and $\bar{\theta}$ the heading we should follow follow. These two expressions imply that $u_{rudder} \in [-\pi/4, \pi/4]$ and $u_{sail} \in [0, \pi/2]$

2.1.2 Line Following

The algorithm used is taken from Luc Jaulin and Fabrice Le Bars paper on sailboat: A simple controller for line following of sailboats [2].



```
Function in: $\mathbf{m}, \theta, \psi, \mathbf{a}, \mathbf{b}$; out: $\delta_r, \delta_s^{\max}$; inout: $q$
1  $e = \det\left(\frac{\mathbf{b}-\mathbf{a}}{\|\mathbf{b}-\mathbf{a}\|}, \mathbf{m}-\mathbf{a}\right)$
2  if $|e| > \frac{r}{2}$ then $q = \text{sign}(e)$
3  $\varphi = \text{atan2}(\mathbf{b}-\mathbf{a})$
4  $\theta^* = \varphi - \frac{2.\gamma_\infty}{\pi}.\text{atan}\left(\frac{e}{r}\right)$
5  if $\cos(\psi - \theta^*) + \cos\zeta < 0$
6     or $(|e| < r$ and $(\cos(\psi - \varphi) + \cos\zeta < 0))$
7           then $\bar{\theta} = \pi + \psi - q.\zeta$.
8           else $\bar{\theta} = \theta^*$
9  end
10 if $\cos(\theta - \bar{\theta}) \geq 0$ then $\delta_r = \delta_r^{\max}.\sin(\theta - \bar{\theta})$
11 else $\delta_r = \delta_r^{\max}.\text{sign}(\sin(\theta - \bar{\theta}))$
12 $\delta_s^{\max} = \frac{\pi}{2}.\left(\frac{\cos(\psi - \bar{\theta})+1}{2}\right)$.
```

Figure 3: Line following controller

This algorithm uses the idea of an attractive line. Indeed, whatever the position of the sailboat in relation to the line, the direction that the sail should follow is a function of the distance of the sail from the line and the angle that the line makes in relation to the abscissa axis. The greater

this distance, the more the sailboat points more towards the line. When the sailboat is able to follow the line its heading will be the angle of the line with respect to the abscissa axis.
But when the angle of the line is in the nogozone, the sailboat will sail on subsequent close hauled angles and tacking between them.
A simulation on ROS and the use of the rviz tool allows us to visualize the figures below.

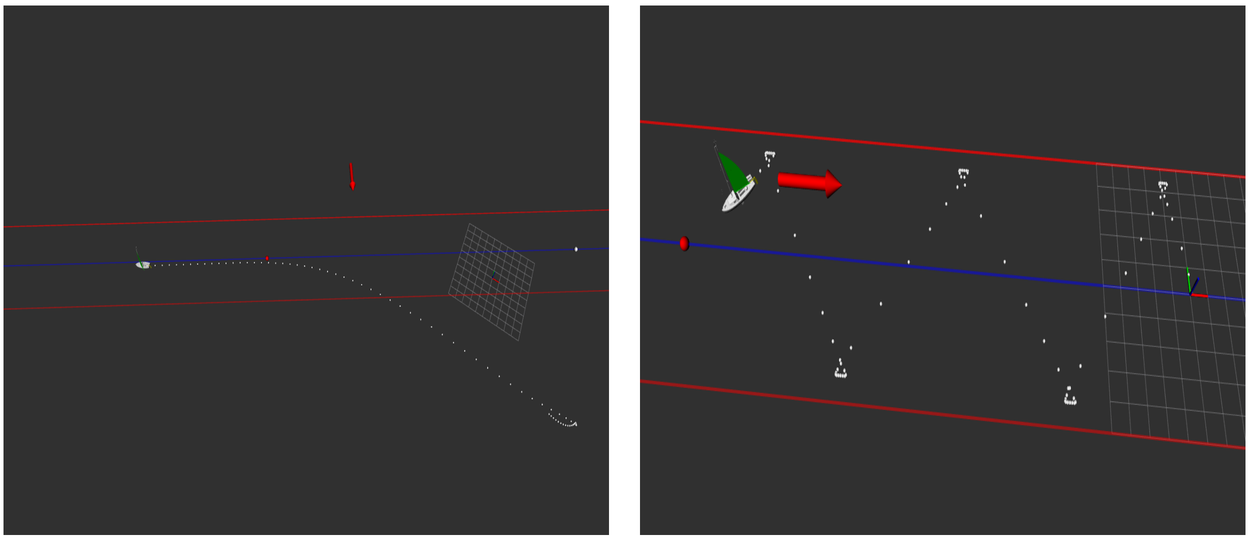


Figure 4: (a) The boat follows the line AB shown in blue. The red lines represent the strip to never leave. (b) In this figure, we can see the boat going upwind. It performs a series of tacks called beating

2.1.3 Station Keeping

Station keeping consists of sailing the boat around a certain area. An area represented by a circle. Many algorithms have been implemented to solve this problem. The idea of the first algorithm is to make the boat do a cycloid. This cycloid will be composed of two circles connected by two straight lines. To make the boat follow a circle, a vector field has been used. The next 3 figures show the boat performing the holding mission with different wind speed values. The purpose of these figures is to justify the robustness of the control law.

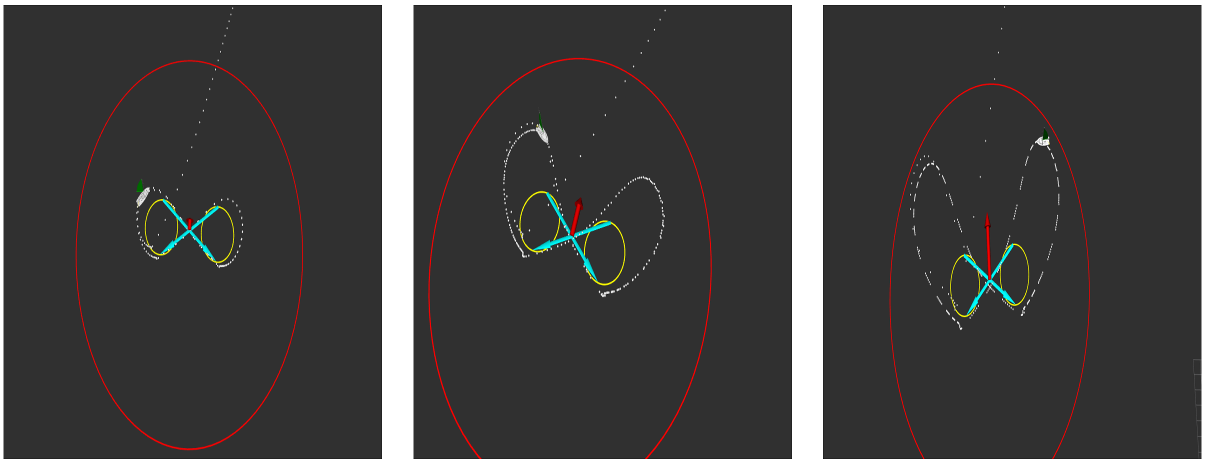


Figure 5: From left to right, we observe the station keeping challenge for the different speed values v=1, v=3, v=5

2.1.4 Station keeping and avoidance

For this problem, the objective is to make the station keeping while avoiding buoy in the center of the circle.
For this situation two algorithms have been developed. The first one consists in making a triangle around a physical buoy while remaining a circle of well defined radius. As in the previous case, I varied the wind speed to test the efficiency of the algorithm.
The second algorithm is that the boat can make a square. For this same problem the results are rather satisfactory than the first one. The boat manages to stay in the circle with the different speed values.

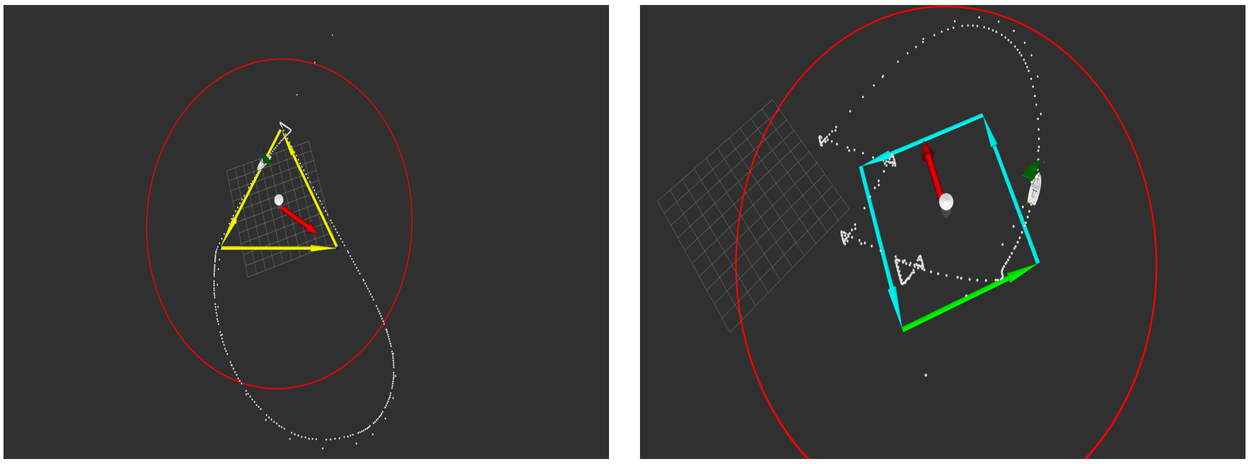


Figure 6: On the figures, the red arrow indicates the direction of the wind, the small white dots indicate the trajectory of the boat after several turns, The big dot in the middle is the physical buoy.

2.2 Challenge of WRSC

The other main objective of the internship is to be able to use previously developed algorithms in order to be able to carry out certain missions. These missions are challenges of the WRSC competition (World Robotic Sailing Championships) [4]. See for more information https://www.roboticsailing.org/2019/rules/

2.2.1 Fleet race

For this mission the boat starts from a start line, passes three virtual buoy and then crosses the finish line. To validate a buoy, the boat must be within a radius of 5m around the buoy.The figure below illustrates the principle of the mission.

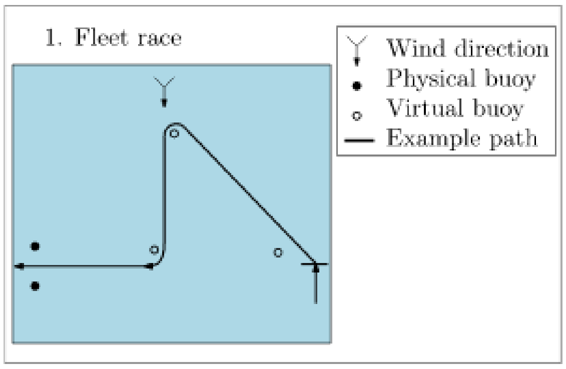


Figure 7: Fleet race

To carry out this mission, I proceeded to a sequence of line tracking between two buoy. But for this challenge, what counts is to cross the finish line in the shortest possible time. So it is necessary to find the shortest way between the start line and the finish line while passing through all the buoy. We could model this small problem in a graph and apply the Dijkstra algorithm. But I proceeded in a different way. At each buoy where the sailboat is, I look for the next closest buoy. And I do this step by step until I cross the finish line

Here is below a capture showing the sailboat carrying out the challenge.

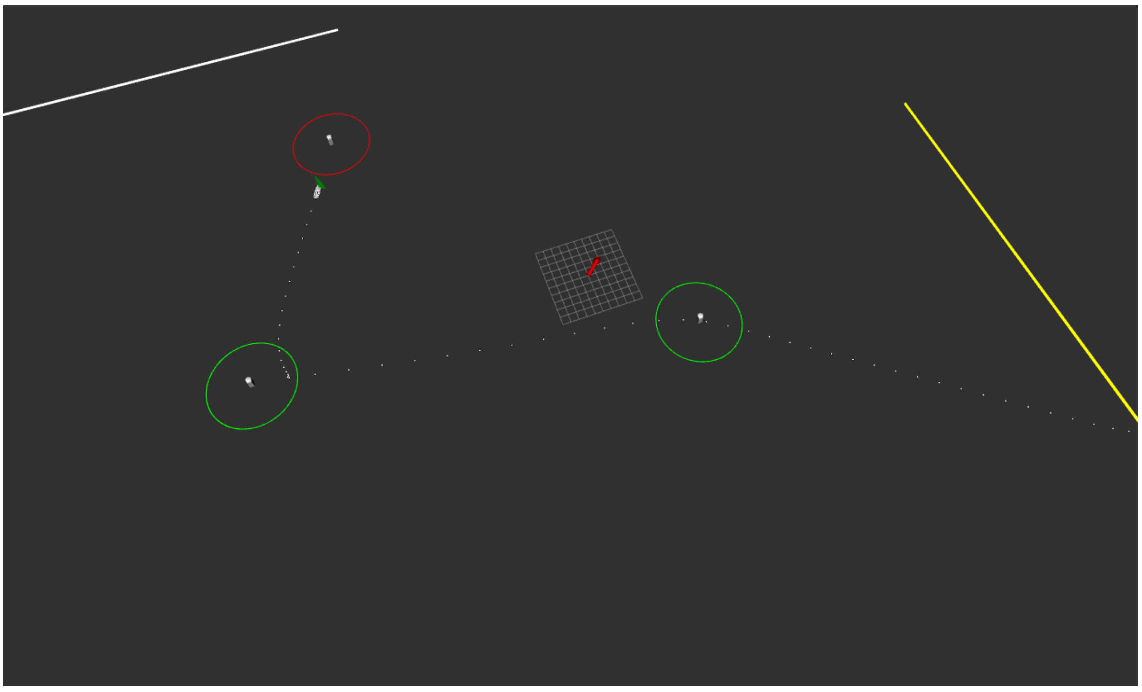


Figure 8: Boat perfoming the fleet race challenge

2.2.2 Station keeping and avoidance

This mission is a combination of two of the algorithms described above. The first one is to stay in the zone for a certain time without constraint, then to leave this zone and go to another zone but this time to stay there while avoiding collisions with the physical buoy.

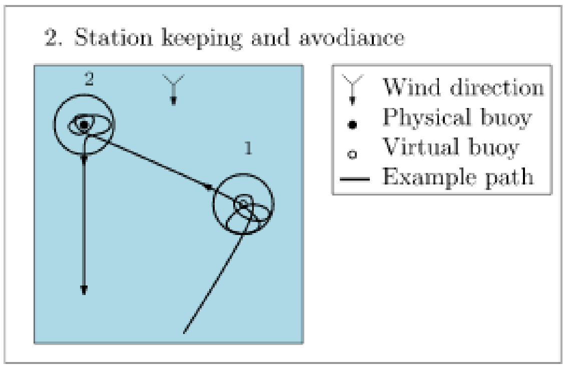


Figure 9: Station keeping and Avoidance
A simulation of this challenge on rviz is below.

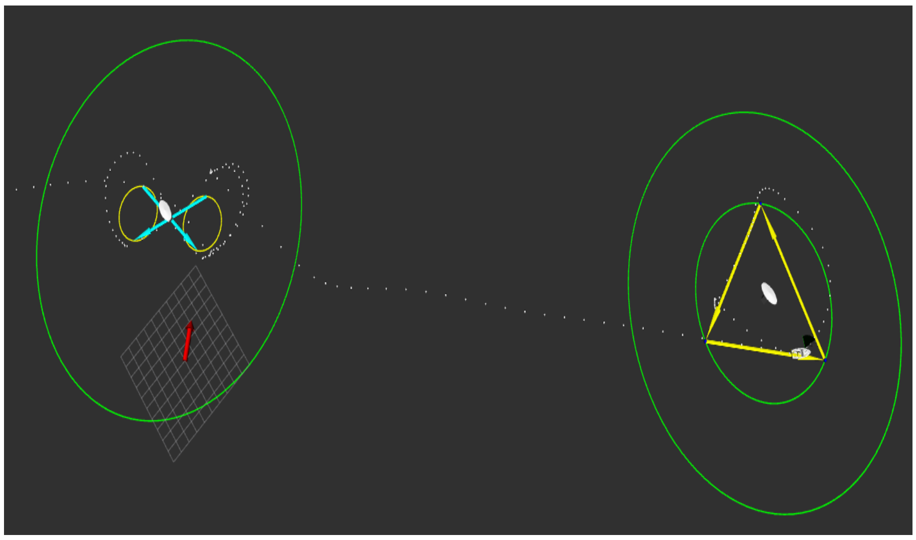


Figure 10: Boat perfoming Station keeping and avoidance challenge

2.2.3 Area scanning

This challenge consists in effectively analyzing a terrain. Four boats are deployed for this task. Each boat must scan a maximum of new area in order to receive a maximum of points. During the challenge the boats receive the areas scanned by the other boats. When a boat scans an area scanned by another boat, it will receive half of the points it could receive if it had scanned a new area. And so the more area has been scanned the more it loses value.

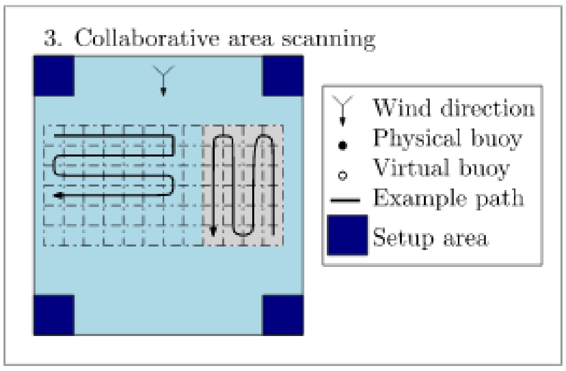


Figure 11: Area scanning

The algorithm I implemented consists in searching the non-scanning area closest to the boat's position. Then when this area is identified, then we use the line following algorithm between the position where the boat was and the center of this area. When I apply this algorithm to the problem, in general, we don't find a single zone as the nearest area but two areas. To choose between these two areas, we use these different criteria:

- if the line to follow is in the nogozone, then we don't choose it. As the two lines that the boat can follow are orthogonal, if one line is inside the nogozone, then the other one is necessarily outside.
- When both lines can be followed, we choose the line that prevents the boat to change its heading.

On the following figure, we simulate this challenge.

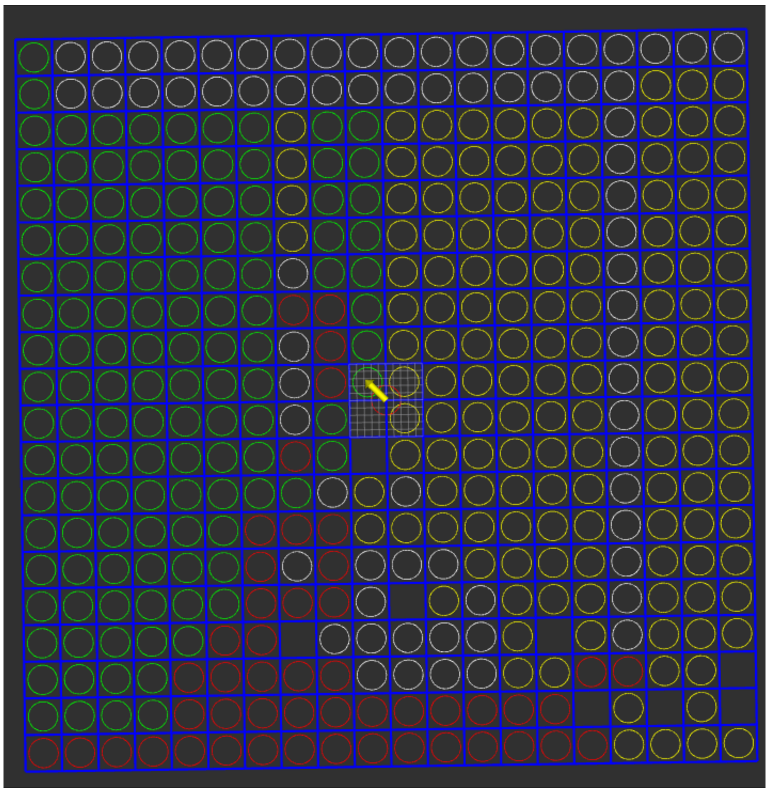


Figure 12: Area scanning challenge

3 Control by artificial intelligence methods

The results obtained previously are very satisfactory. Nevertheless, there is still room for improvement. For example, when the wind speed is very high, the boat does not manage to stay at the guard post. Also the area scanning algorithm does not completely solve the problem. Indeed, when we implement our area scanning algorithm on each of the 4 boats, 90 to 95% of the areas are scanned.

During the second part of my internship, I had to develop artificial intelligence algorithms in order to obtain a better result than deterministic algorithms.

To do this, I first had to have machine learning skills. Because being a second year student, the courses on machine learning are planned for the third year. So I started to study these courses during this period.

After having read enough articles and books, I decided to develop an artificial intelligence algorithm for the area scanning problem.

3.1 Machine Learning

The learning machine is a sub-domain of intelligence. It is about learning a statistical model to make predictions from training data. So when you do machine learning, there are two phases. A learning phase which consists of training the model on training data. Then a prediction phase which consists of using our model to make decisions. There are three paradigms in machine learning [5].

- ***Supervised learning:*** the machine learns a prediction function from labeled examples. More precisely, we want to be able to predict an output y from an input x. So we give the model a large amount of training data corresponding to a set of x and y values. So the model will learn from these examples to make a correct prediction.

  $$f(x) = y$$

  f is le function(model) that translate x to y

- ***Unsupervised learning:*** the machine learns a model without the learning examples being labeled. The machine will have to find the similarities and distinctions within this data alone, and to group together those that share common characteristics.

- ***Reinforcement learning:*** this paradigm differs from the two previously mentioned by the fact that the learning data are generated as the learning takes place. This technique is based on a cycle of experience/reward and improves performance with each iteration. This paradigm will be further explained in the next section.

Of these three paradigms, only the third has been applied to the area scanning problem.

3.2 Reinforcement Learning

Reinforcement learning is one of the three paradigms of machine learning, along with supervised and unsupervised learning. an agent (robot, etc.) is placed in an environment, takes action according to its current state and receives from the environment the next state and a reward (positive or negative).
The objective of the agent in the reinforcement learning phase is then to

seek a behaviour capable of accumulating the maximum of these rewards[6].

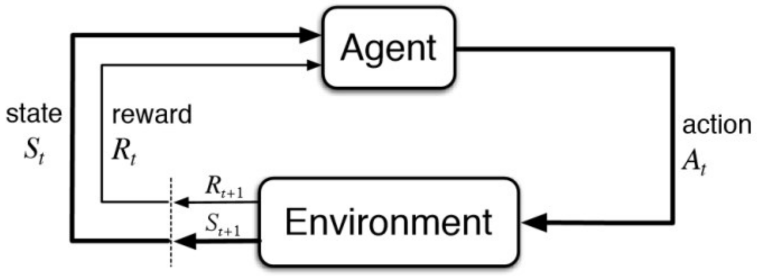


Figure 13: Interaction loop between the agent and its environnment

Markov Decision Problems (MDP) is one of the stochastic models capable of formally describing the interactions between an environment and an agent. A MDP is composed of four components: state, action, transition function and reward function.

- ***State:*** the totality of information necessary and sufficient to predict the future evolution of a system.

- ***Action:*** it is the set of possible movements that the agent can have in order to interact with his environment.
  The set of states and actions can be finite or countless states can be finite or continuous.

- ***Transition function:*** it defines the effect of the agent's actions on the environment. More precisely, it represents the probability of going from state s to state s' by taking an action a. When all states and actions are finite, this function is represented by a matrix.

- ***reward function:*** prescribes, for each transition from one state $S_t = s$ to another $S_{t+1} = s'$, following the execution of an action $A_t = a$, a reward perceived by the agent, called immediate reward and noted $R(s, a, s')$. But this function may not depend on the state s' $R(s, a)$) or on the state s' and the action a ($R(s)$).

There is a wide variety of algorithms to solve reinforcement learning problems. We can classify these algorithms in two groups: those based on a model and those based without a model.
A model-based algorithm is that uses the transition function and the reward function in order to estimate the optimal policy. So the algorithm without

a model does not use the transition and reward functions to estimate the optimal policy. During this internship, I decided to become interested in algorithms without models and more precisely the Q-learning algorithm. Because I find it easier to understand and implement.

3.3 Q-learning

As said before, the Q-learning algorithm belongs to the class of algorithms without models. It seeks to learn a policy that maximizes the total reward. For this it uses a state-action value function Q which measures the quality of an action in a state. When the number of states is not very large then this function can be replaced by a matrix called q-table. The rows of this q-table correspond to the states and the columns to the actions. Initially the values of this table are zero. The objective of the training phase is to find the right values of this table. So at the end of the training phase, The q-table will guide us to the best action at each state.
During the training phase, in each state, the agent takes an action in order to interact with its environment. Now comes the dilramma of the choice of actions. Indeed, in each state of the agent, several actions are possible but which one to choose. After the training phase, can we be sure to have passed through all the states? A very simple algorithm called epsilon-greedy allows us to answer this problem. This algorithm uses the exploration-exploitation trade-off:

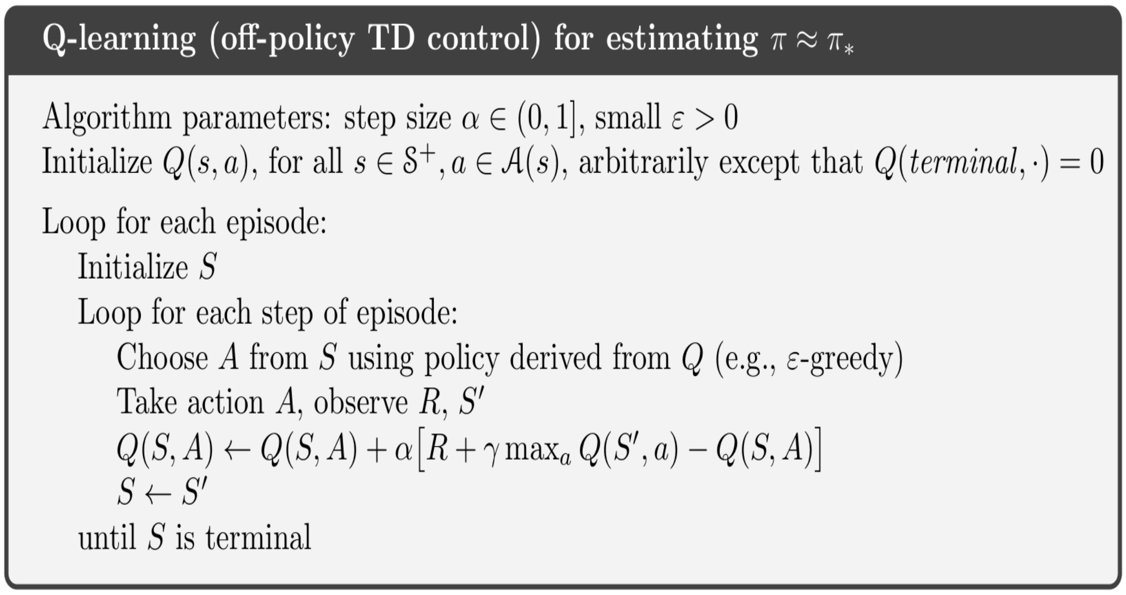
- ***Exploration:*** consists in randomly choosing (with a probability) of epsilon an action among all possible actions. This allows the agent to improve its current knowledge on each action but especially to discover new states.

- ***Exploitation:*** consists in choosing for a given state the action that has the highest value.

At each episode of the training, the Q-learning algorithm uses the bellman equation to update the values in the table.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot max_a Q(s_{t+1}, a_t) - Q(s_t, a_t))$$

$Q(s_t, a_t)$ value of the function in state s_t for the action a_t, r_t is the reward received when passing from the state s_t to the state s_{t+1}, $\alpha \in [0, 1]$ is the learning rate and can be defined as how much you accept the new value compared to the old one, $\gamma \in [0, 1]$ is the discount factor and has the effect of valuing rewards received earlier than future rewards, $max_a Q(s_{t+1}, a_t)$ take the maximum of the future reward and apply it to the reward for the current state.
The following figure shows the Q-learning algorithm.



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Figure 14: Q-learning algorithm

3.4 Solving the area scanning problem with the Q-learning algorithm

Among the classical problems solved by the deterministic algorithms developed during the first part of the course, it was necessary to choose a problem and try to solve it with the Q-learning algorithm. Whatever the problem to be solved, three elements must be well defined: The state of our agent, the possible actions he can take, the rewards obtained by the execution of each action. The definition of the rewards is the most important part when we want to use a reinforcement learning algorithm. For example for the station keeping problem, we could easily define the rewards by: As long as the sailboat is the zone, it receives a positive reward and this reward is greater if the sailboat is close to the center of the zone. But if the sailboat is out of

the zone then it receives a negative reward.
I chose instead to solve the area scanning problem. It is true that it is more complicated to solve compared to the station keeping problem. Above all, defining its reward function is not easy.

3.4.1 table Generation

The area to be scanned is composed of 400 grids of dimension 50×50 and each grid is a square of 2.5×2.5. In order to have the dimension of my reasonable Q-table, I thus considered the position of each grid as being the state of our agent. The actions that can take the agent are eight in number. Depending on the wind direction some actions are not possible. The following figures illustrate this.

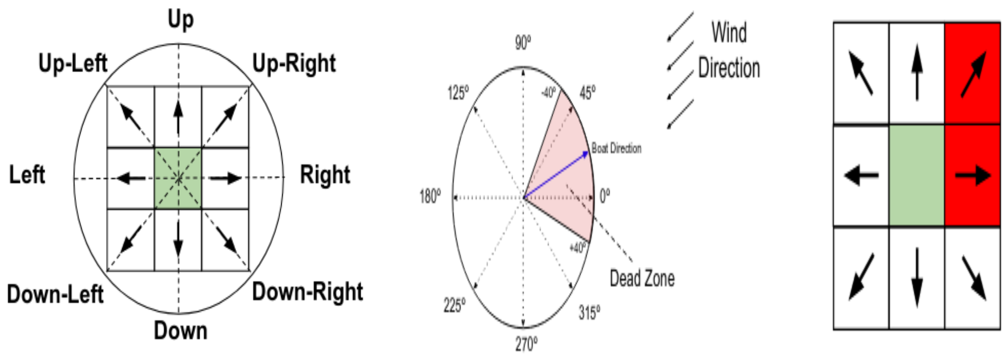


Figure 15: The different actions the agent can take

During the implementation of the Q-table in python, I represented it with a dictionary. The keys being the states (thus the positions of each grid) and each value of a key is represented by an array of eight real numbers (like the eight actions).

3.4.2 Reward function

As I said before, defining the rewards well will lead to a better result. The overall objective of the area scanning mission is to achieve (for the four boats) to scan all the grids (400 in total) in a minimum time. So it is necessary to avoid that the boats do not pass again on zones already scanned, otherwise it will make us lose time. So each time the agent scans a new grid, he gets a positive reward (+5), if he scans an area already scanned, then he gets a negative reward (−5). But this definition is not enough. Indeed when we consider for example that the boat has an upward heading and that the following grids are not yet scanned, it would be better to choose the **Up** action. Because the other actions would force the boat to change course, which would waste the agent's time. But if the grid in the **Up** direction is already scanned, choose the **Up-right** action. The following figure explains the choice of the different grids according to the boat's heading.

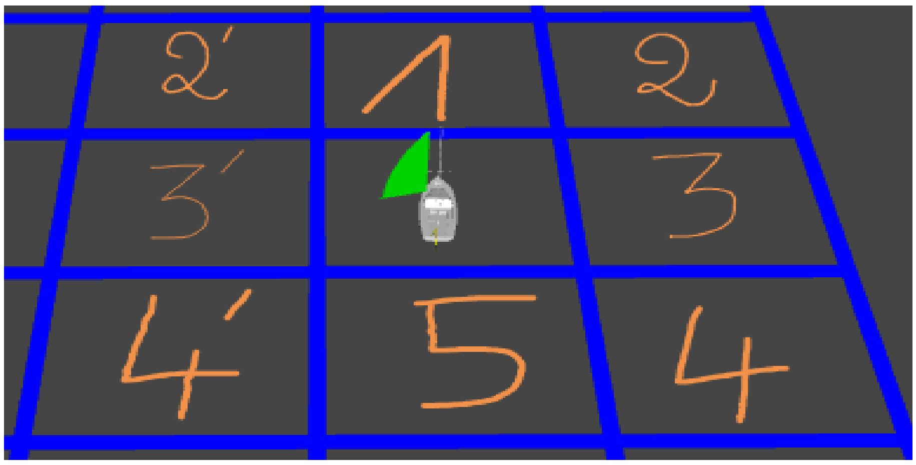


Figure 16: The 1 corresponds to the preferred direction. The 2 and 2' correspond to the following actions to be taken if the grid on action 1 has been scanned. The 3 and 3' correspond to the following actions to be taken if the three previous actions are not possible. Etc....

3.4.3 Results

Unlike other previously implemented algorithms, this algorithm has been implemented in python. The training phase consisted in creating 4 agents (i.e. four instances of a class) and making these agents evolve in the same environment.

The reward function described above was the best reward function I ever implemented. Indeed other reward functions were defined but they did not have satisfactory results.
On the following figures are the results of simulations with two different reward functions.

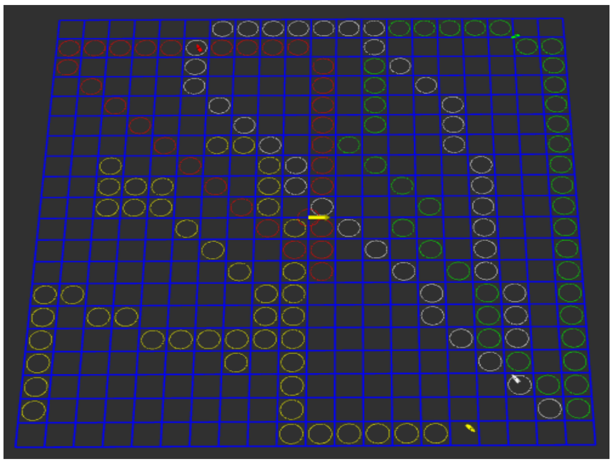
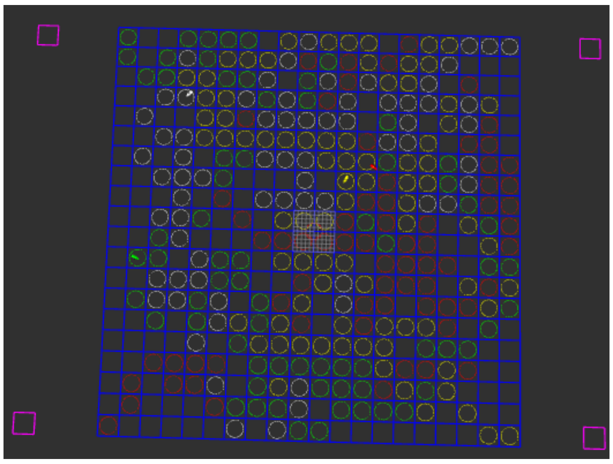



Figure 17: On the left the result of the first reward function implemented. On the right the result of the simulation with the reward function explained in the previous paragraph.

Conclusion

At the end of these four months of internship, I was able to consolidate some knowledge seen during the school year such as the courses on robot control, ROS, C++. Moreover I learned new skills in artificial intelligence even if this knowledge will be learned in 3rd year.
The whole internship was done in simulation. Indeed because of the health crisis, I did not go to Plymouth to do experiments on sailing boats. I think that this is the downside of this course. Because one of my main objectives during this training course was to do enough practice.

References

[1] https://fr.wikipedia.org/wiki/Voilier
[2] Luc Jaulin and Fabrice Le Bars. *A simple controller for line following of sailboats*
[3] Luc JAULIN. *Mobile robotic*, **RobMOOC**. 2019.
[4] WRSC Rules on https://www.roboticsailing.org/2019/rules/.
[5] Wouter De Winter. *Build an Autonomous Sailboat Using Machine Learning*
[6] Richard S. Sutton and Andrew G. Baro, *Reinforcement Learning: an introduction*