Development of an autonomous sailing boat in a realistic maritime environment

ROB23

February 20, 2023

Contents

Abstract

This report sums up what have been achieved during robotic architecture courses. The class worked on the development of an autonomous sailing boat in a realistic maritime environment, by splitting the project in six parts: control, obstacle avoidance, reliefs, UFO(Unidentified Floating Object), AIS(Automatic identification systems) and simulation. This report presents the approach of each team: why they choose to deal with their part as they are done, and what they achieve. It will finally deliver the results of the whole project, using each team's work. And the possibilities of improvement.

Chapter 1

Introduction

1.1 Problematic

A call for expressions of interest has been launched in 2021 by the IMOCA class and the maritime cluster. The objective: to prevent and minimize the risks of collision at sea. More and more collisions are being observed during races, which the safety of sailors, the preservation of boats, and the lives of mammals. In the Vendée Globe 2020/2021 alone, three collisions have been reported.

It is in this context that BSB Marine and Pixel sur Mer have set up a joint consortium to respond to these challenges in the nautical world. these challenges in the nautical world. BSB Marine is developing Oscar, an obstacle detection system based on vison and artificial intelligence. Pixel sur Mer develops advanced automatic piloting solutions for racing sailboats.

A sailboat under autopilot evolves in an environment with obstacles of various types:

- UFOs (Unidentified Floating Objects) : Buoys, containers, cetaceans, small boats...
- Vessels: Vessels equipped with the AIS (Automatic Identification System), allowing them to transmit information such as their position, speed, size to other vessels in the same navigation zone.
- Charted landmarks: Rocks, shoals and coasts referenced on nautical charts.

It is equipped with a camera at the top of the mast to detect UFOs (protruding from the surface of the water), an AIS, a GPS and a marine chart of the navigation area.

We wish to use the data from these information sources to determine if there is a risk of collision for the sailboat and to characterize it (type of obstacle, CPA: Closest point of Approach, TCPA: Time to Closest). for the sailboat and to characterize it (type of obstacle, CPA : Closest point of Approach, TCPA : Time to Closest point of Approach...). If the risk is high, the autopilot will perform an obstacle avoidance maneuver to avoid the collision. The obstacle avoidance algorithm will have to take into account the control constraints of a sailboat where only the rudder angle is operable. It will have to manage situations with multiple moving obstacles. The trajectory will be updated and adjusted continuously to take into account the evolution of the situation provided by the detection equipment.

The objective of this course is the realization of 2 demonstrators of a complete obstacle avoidance system for sailboat. One in simulation to test predefined or random scenarios. The other in real life using the sailboat BRAVE of ENSTA Bretagne and buoys and USVs playing the role of obstacles.

1.2 Organization

This project include twenty members. A good use of such capacities required a strong organization. For that the project have been split into six parts: the boat simulation, is control, the obstacle avoidance, the detection of the reliefs, the UFO(Unidentified Floating Object) and the other boats. For managing the teams a student have been choose as project manager. For learning purpose the class's members choose to work using ROS2. This new version of ROS1 middleware is not yet well documented and we've not be formed yet. But with this new version, the old one will be slowly left behind, for we have to know how to use it. After the research of the state of the art, we defined the main goals of each teams and links between them(figure 1.1). For synchronize the whole development we set goals and share codes using github.

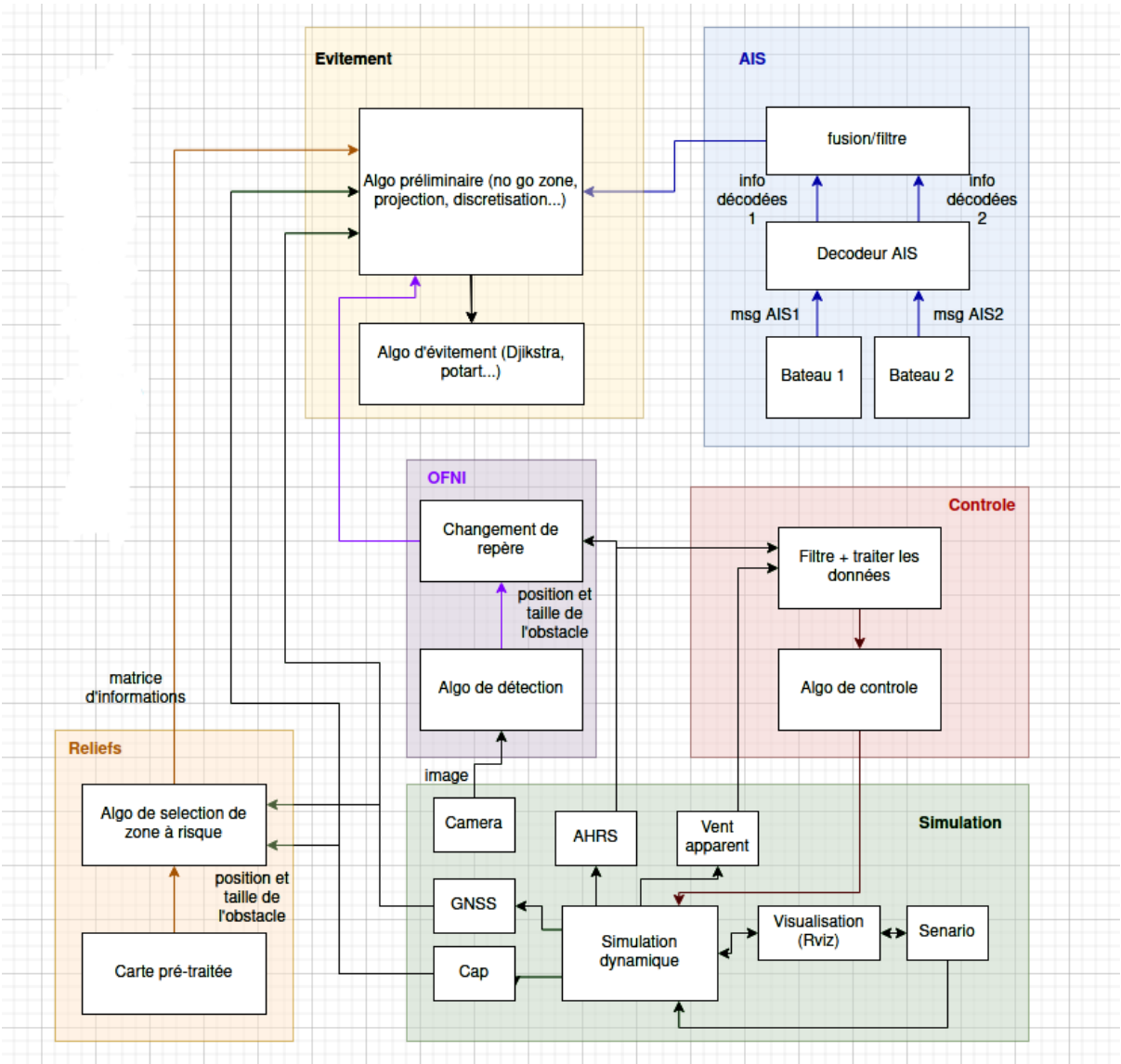


Figure 1.1: Software architecture.

1.2.1 Github tools for teamwork

Git is a great tool for software development. It allows to do version control and distributed code. Github is a website using git, it adds lot of functionalities to simplify the shared software development. One of this functionalities is the organization. It gives a frame for a project, regrouping all the repositories under this framework, allowing team structure with roles and rights, and creating interface to simplify management.

All tasks is show by an issue git. When an issue is resolved it has to be closed with a description of how it have been resolved. Any member of the organization can be assigned to an issue. To differentiate the issues you can add labels. Labels defined the issues, it could be the priority of the issue or the team that have to deal with. Several issues can be group under a milestones(figure 1.2). A milestones is a large task that require little tasks, represented by issues. For the project tracking, in addition to the assignees, every issues can be stated to "To do", "In progress" or "Done", and prioritized with "High", "Medium" or "Low". For visualize progress github provided personalized insights of progression, class by milestones or labels. In figure 1.3 you see the milestones status and the number of issues by milestones.

With this tools each member know what he have to do, in with order, and see the whole advancement. And the manager can easily add or remove tasks, assigned members to issues, and had a view on the advancement for directing the work.

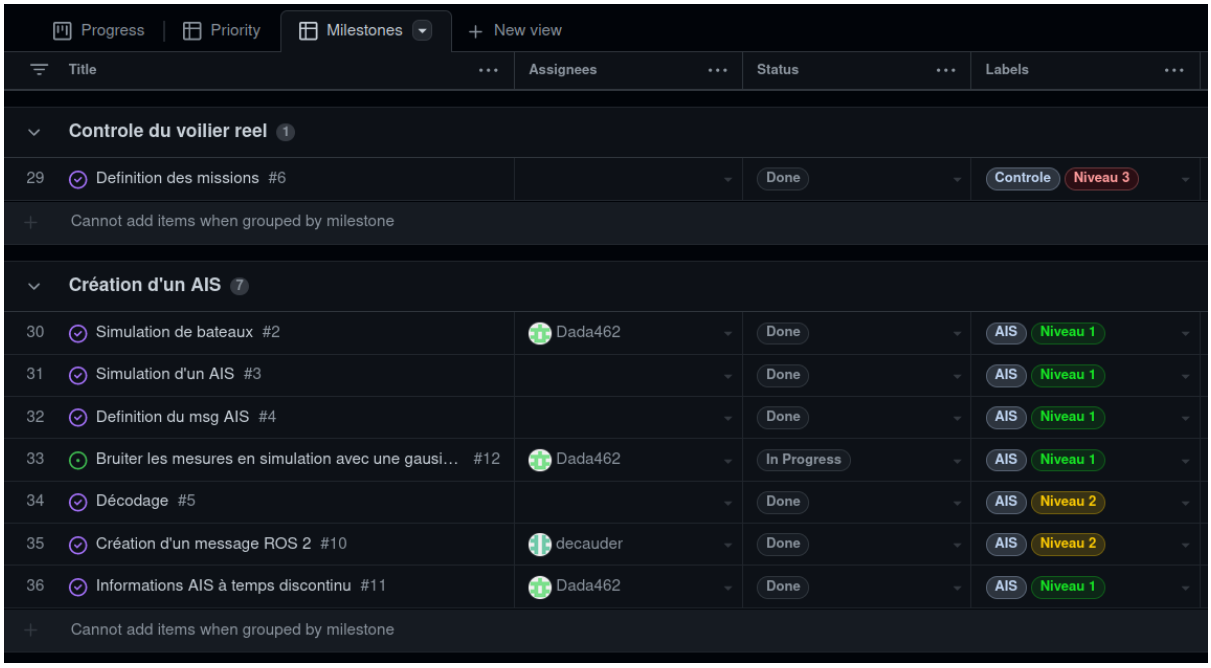


Figure 1.2: View of two milestones.

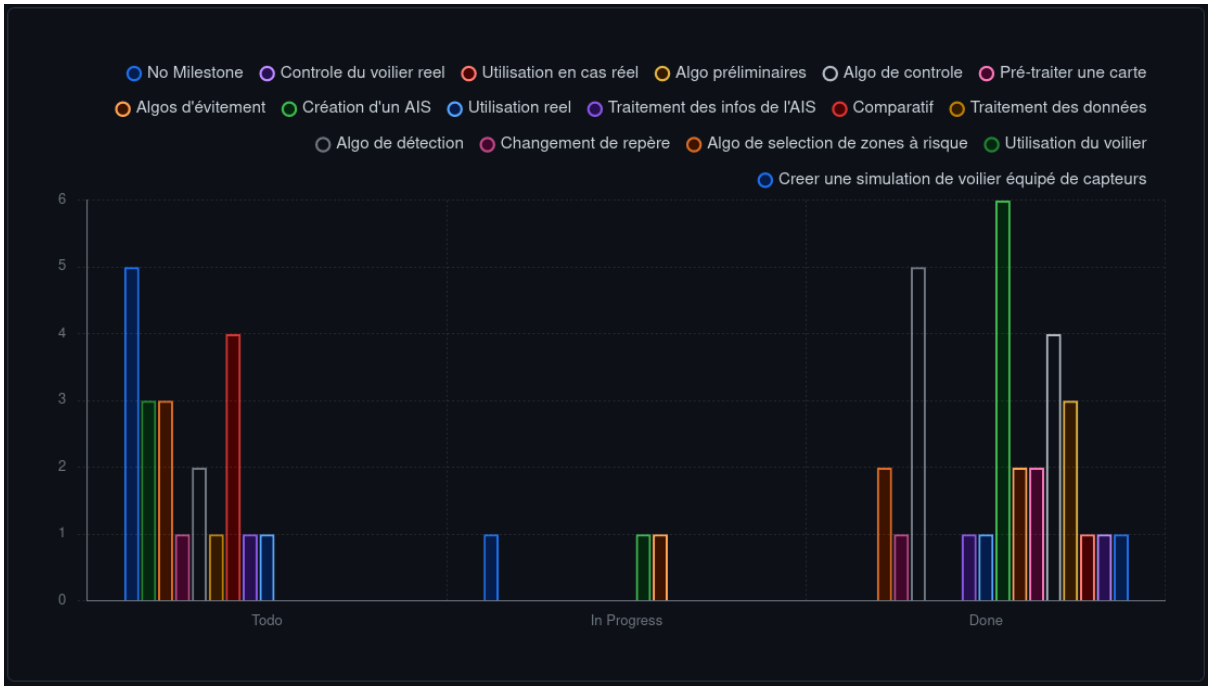


Figure 1.3: Milestones status insights.

Chapter 2

UFO

2.1 Problematic

The objective of this part is to detect objects with a camera and to estimate the distance between the boat and each object.

The camera is a classic monocular RGB camera placed on the top of the boat's mast. The camera doesn't move and face the front of the boat.

We decide for this project to focus on the buoys detection because this is the most common object in the sea and it's relatively easy to detect.

2.2 State of the art

Because we can't use a depth camera, we have to estimate the position of the object. We first tried to see some algorithms which build a depth map with a single RGB image. We looked at :

- MiDas (https://github.com/isl-org/MiDaS)
- BoostingMonocularDepth(https://github.com/compphoto/BoostingMonocularDepth)

Nevertheless, those solutions were not easy to implement, with heavy installation. Moreover those libraries were quite slow, it would be a problem in the simulation to work in real time.
We decided to separate the work in two main tasks, in order to find simpler algorithms :

- Detection of each buoys in the image
- Estimation of the distance between the boat and each buoys

In order to find a solution to detect buoy in a an image, we decided to use AI, as OSCAR camera does, a camera that detects UFO.

2.3 Strategy

2.3.1 Distance estimation using horizon detection

In order to estimate distance, based on the initial research we found multiple possible approaches.

The horizon detection is a crucial part in distance estimation when using only a monocular camera without any depth map estimation algorithms mentioned above. Given that the camera is fixed on the mast, thus the height above sea level is known. When we know the position of the horizon in the image, we can calculate estimation of the distance to given object. The result is a Python class that takes an input image and the [x, y] coordinates (in pixels, in the image reference frame) of detected object and returns a distance estimation. After a research and multiple tests, we decided to base the algorithm on [1].

Horizon detection algorithm

General structure of the horizon detection algorithm is described on Figure 2.1. The algorithm can be divided into 3 main parts:

1. Image preprocessing
2. Detection of horizontal section containing the horizon
3. Detection of the horizon line itself

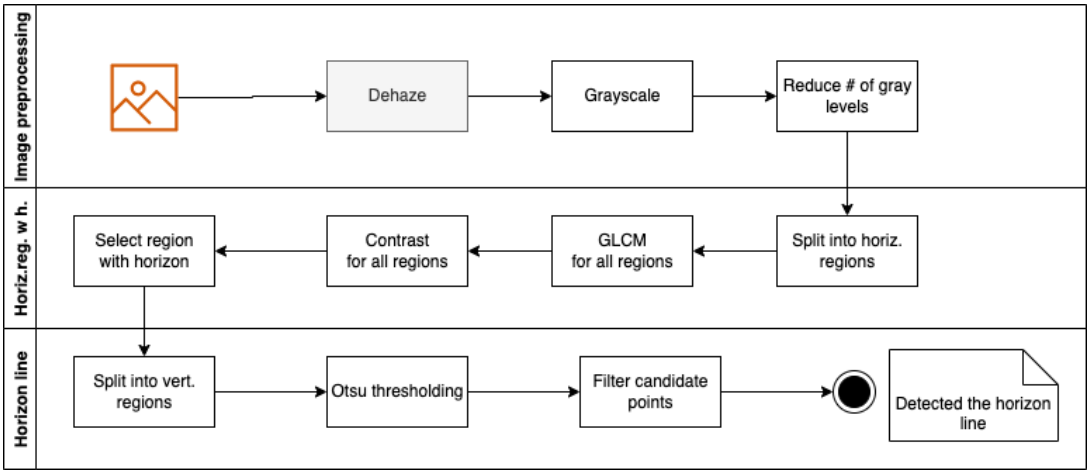


Figure 2.1: Overview of the horizon detection algorithm

We begin with the image preprocessing part. As the real world image can be hazy, it can be useful to dehaze them, this has been done, but as we will be using only Gazebo simulation images as an input after the integration with other teams work, it is not covered in this report. In case you want to implement this, we propose you to read [2], [3] and [4].

The first required step is transforming the image into levels of gray. The paper [1] proposes following solution to obtain lower contrast between the sky and clouds:

$$\mathbf{I_{gray}} = f(x,y)_{MxN} = (\begin{cases} R(x,y) & \text{if R(x,y)} > \text{G(x,y)}, \text{R(x,y)} > \text{B(x,y)} \\ G(x,y) & \text{else if G(x,y)} > \text{R(x,y)}, \text{G(x,y)} > \text{B(x,y)} \\ B(x,y) & \text{otherwise} \end{cases})_{MxN} \tag{2.1}$$

where I_{gray} is a grayscale image, $[x, y]$ are each image pixel's coordinates and R(x,y), G(x,y), B(x,y) are pixel values of red, green and blue channel. Then for performance reasons, it is useful to reduce number of gray levels. We used 16 grayscale levels ($\mathbf{I_{gray}} = \mathbf{I_{gray}}/16$).

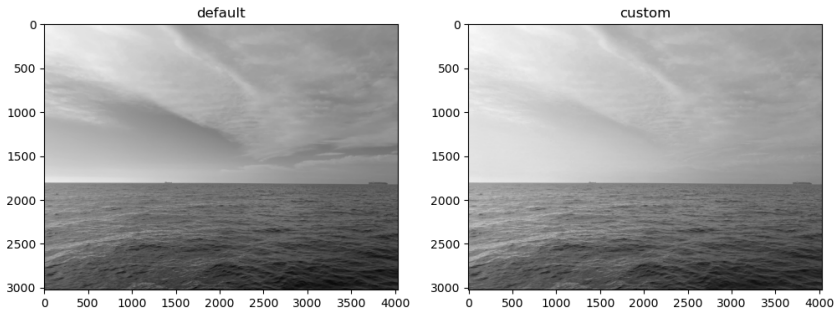


Figure 2.2: Comparison between default /YUV model/ grayscaling (on the left) and the proposed custom method (on the right)

The following step is to detect a smaller horizontal section of the image containing the horizon line. First, we cut the image into $K = 10$ horizontal regions. The number of regions can be adjusted as the input image resolution or performance limitations changes.

For each horizontal region, the Gray-Level Co-ocurrence Matrix (GLCM) is computed. The GLCM tells us information about texture in given region. It is a matrix of shape $N_{grayLevels} \times N_{grayLevels}$ (16×16 for 16 gray levels) containing information about pixel value changes between neighbouring pixels in given direction. Or in other words *how often a pixel with a given intensity value i occurs next to the other intensity value j.* To improve robustness when the image is slightly tilted, we used a $\theta = 45°$ direction and step size $d = 1$ - that means that it takes the closest neighbouring pixels in diagonal direction).

Then the contrast C from Gray Level Co-occurence Matrix $\mathbf{P}$ in that region is calculated by the following formula:

$$C = \sum_i \sum_j (i-j)^2 \cdot \mathbf{P}(i, j, d, \theta) \tag{2.2}$$

where i is a row index, j is a column index in the GLCM, d is a distance between pixels taken into account and $\theta = \{0°, 45°, 90°, 135°\}$ is an angle between the current and the neighbouring pixel taken into account. [1]

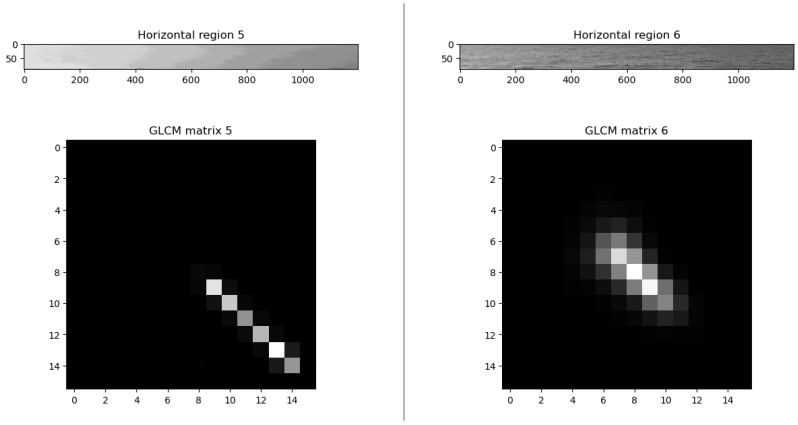


Figure 2.3: GLCM of region containing a sky (on the left), GLCM of region containing a sea (on the right)

The final step is to select the region containing the horizon. The region containing the region should have the highest change in contrast. It can be calculated using following formula:

$$G(k) = |C(k) - C(k-1)|, \text{ for k} \in \{1, 2, ...\} \tag{2.3}$$

where k is index of the current horizontal region, $C(k)$ is a contract of the current horizontal region and $C(k-1)$ is contrast of the previous one. The region with the highest $G(k)$ value should be the region containing the horizon:

$$k_{containingHorizon} = argmax(G(k)) \tag{2.4}$$

When we have the horizontal region with the horizon, we can detect the horizon line itself. To do that, we cut the horizontal region into $e = 20$ vertical sections. Each of this regions is transformed into binary image using Otsu thresholding. (See Figure 2.4)

When we have the binary image for each section, we can find the points on the edge between black and white easily. (See Figure 2.5)

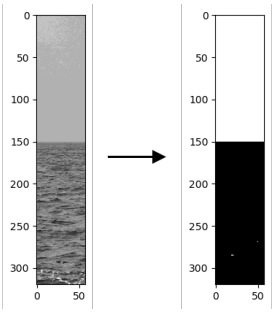


Figure 2.4: Example of thresholding result

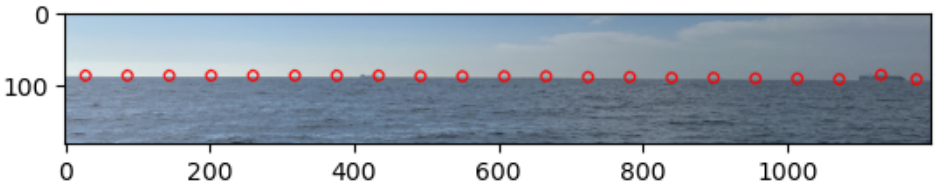


Figure 2.5: Horizontal section containing points on the detected horizon

Then we use these detected points to calculate parameters of the horizon line equation ($y = ax + b$) by using formulas 2.5 and 2.6. [1]

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{j=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \tag{2.5}$$

$$b = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{j=1}^{n} y_i}{(\sum_{i=1}^{n} x_i)^2 - n\sum_{i=1}^{n} x_i^2} \tag{2.6}$$

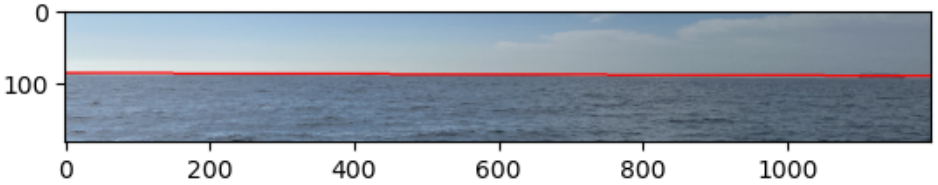


Figure 2.6: Horizontal section with the detected horizon line

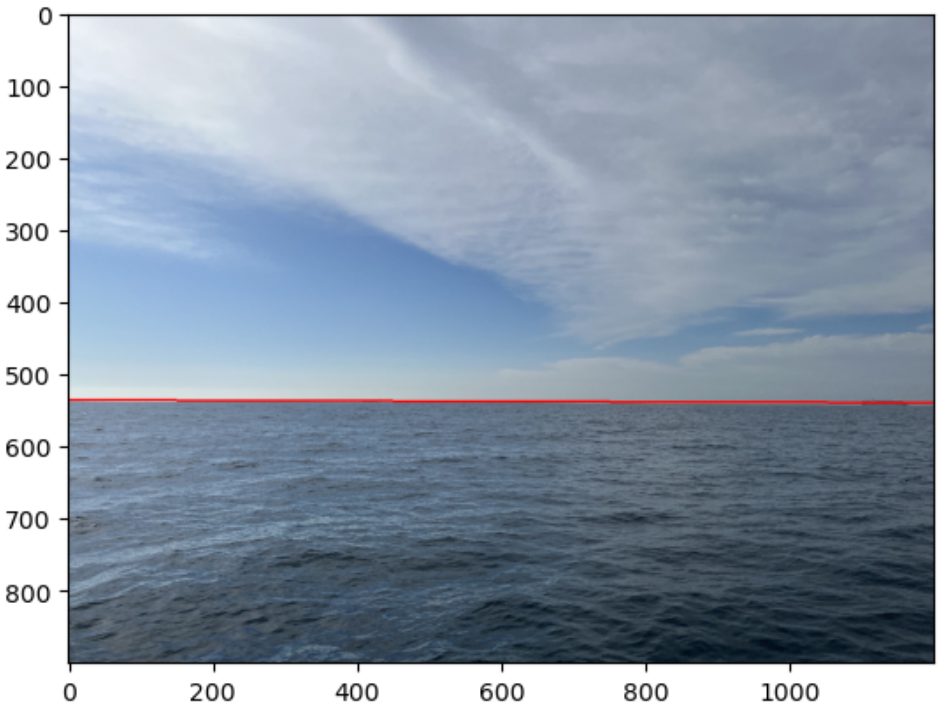


Figure 2.7: Input image with the detected horizon line

Distance estimation

The distance is estimated by using the detected horizon line and calculating the angles between the points on a sphere (we approximate Earth as a sphere). For that, the camera should be calibrated and we need to know before the calculation at least:

- height of the camera above sea level (h)
- camera's diagonal Field Of View (FOV_{diag}) and output image dimensions

First, we calculate the diagonal image dimension in pixels

$$D = \sqrt{img_{height}^2 + img_{width}^2} \tag{2.7}$$

where img_{height} and img_{width} are dimensions of the image in pixels. Then we calculate a focal length in pixels

$$f_{px} = \frac{D}{2 \cdot tan(\frac{FOV_{diag}}{2})} \tag{2.8}$$

and we continue with computation of the needed angles (see Figure 2.8)

$$\Theta = 2 \cdot tan^{-1}(\frac{D}{2 \cdot f_{px}}) \tag{2.9}$$

$$\Omega = cos^{-1}(\frac{R}{R+h}) \tag{2.10}$$

where $R = 6371000\ m$, and h is a height of the camera above sea level in meters.

$$\gamma = \frac{\Delta y}{D} \cdot \Theta \tag{2.11}$$

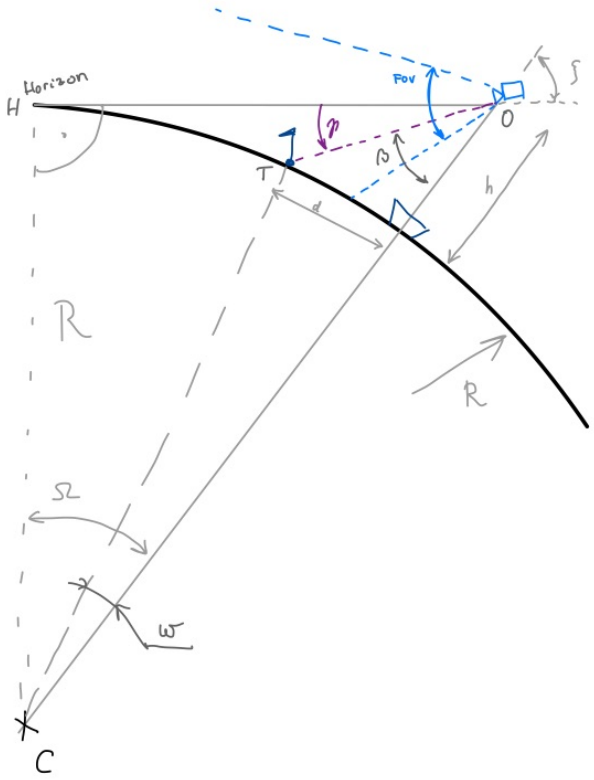


Figure 2.8: Schema of the horizon, camera and Earth for calculating the distance

where Δy is a vertical pixel distance between horizon line and point which distance we are aiming to estimate. The image should be already leveled. The angle for leveling the image is determined from the previously detected horizon line.

$$\beta = 90° - \Omega - \gamma \tag{2.12}$$

$$\omega = 180° - (\beta + 180° - sin^{-1}(R + h) \cdot \frac{sin(\beta)}{R}) \tag{2.13}$$

and finally the estimated distance to the object is an arc length:

$$distance = R \cdot \omega \tag{2.14}$$

2.3.2 Obstacle detection

Isolate and detect an object in an image could be simple when the object has some specific geometrical features. It could be its color, its shape or others. This method avoid us to use AI that could use a lot of time and power of calculation. Unfortunately, buoys didn't have the same colors or same shape so we had to use AI.

Because buoys are easily distinguishable from their environment, we decided to use Unsupervised Classification. This is a method to sort elements in different classes using AI, in images, the elements are each pixels of the image and the classes are a RGB value.

We tried and compared two methods of sorting the pixels of the image : *K-means* and *Gaussian-Mixture-Model*

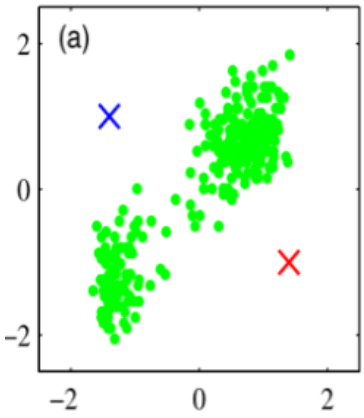


Figure 2.9: Initialization of the K-means method

K-means

The concept is quite simple, we want to sort all our element in K classes (two for this example), so we place randomly two points, called *clusters* (figure 2.9).

Then we calculate the distance between each elements and the two clusters. The elements will belong to the class which correspond to the closest cluster. The position of each cluster is re-evaluated by averaging the position of the elements in its class. We repeat those steps until the elements didn't change of classes. (figure 2.10)

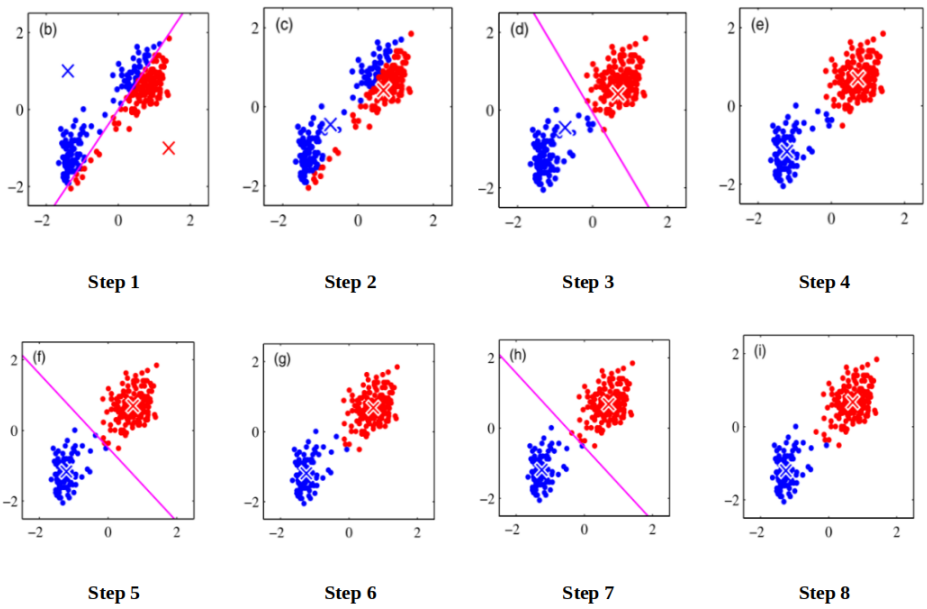


Figure 2.10: Steps of the K-means method

The K-means method can work well on simple images, when we specify the right number of classes. However, it is a rather simple method for classifying groups of pixels, and sometimes it is better to use the Gaussian-Mixture Model.

Gaussian-Mixture-Model

The Gaussian Mixture Model is an improved version of the K-means method, but where we try to fit K Gaussian curves over the set of data, as shown in Figure 2.11, and the Expectation–maximization algorithm : We start with a random mean and covariance, compute for each point the likelihood for each point to belong to each distribution, and update the mean and covariance of each distribution until convergence, as shown in Figure 2.12.

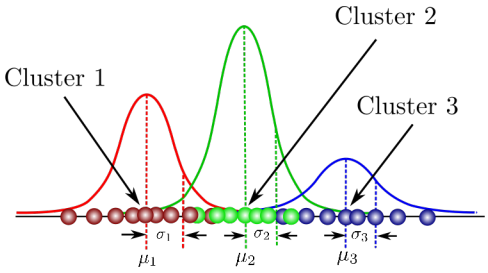


Figure 2.11: Gaussian Mixture method

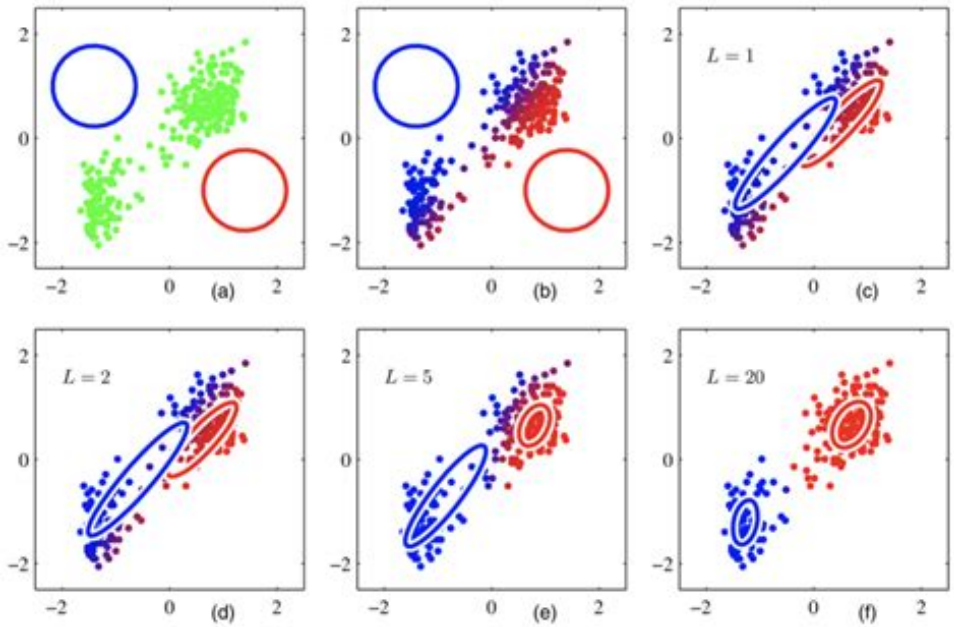


Figure 2.12: Steps of the Gaussian Mixture method

Just as the K-means method, the Gaussian-Mixture expect the known number of classes before training, and classify each point inside a class, which is useful for repetitive image processing, but for real world application such as buoy detection, this algorithms are still a bit lacking as depending on weather condition, luminosity, etc. For a better classification, the model can be improved with an Neural Network based AI for object detection and recognition, that we did not have time to train here. Additionally, these two methods can be fused with basic image processing function, such as blur, HSV masks, and morphology operations.

2.4 Merge of the work

Merging the different part of the work is mostly straightforward, and was implemented as a Class, containing three different process : The image prepossessing, filtering and thresh-holding before classification, an HSV filter and a Gaussian Mixture Classification and finally the Horizon Detector and distance estimation of an object. We added different kind of filtering methods to keep only the relevant points inside each frame before calculating the distance. The result can be seen in Figures 2.13 and 2.14, and a youtube video can be seen here

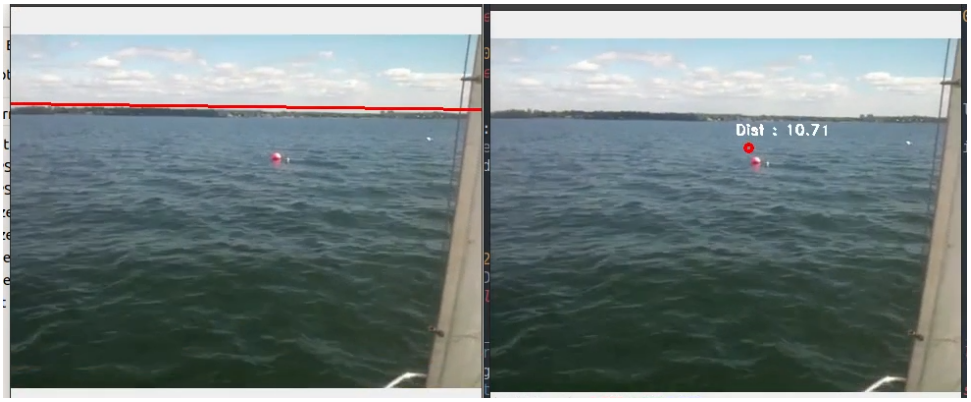


Figure 2.13: Example 1 on a video

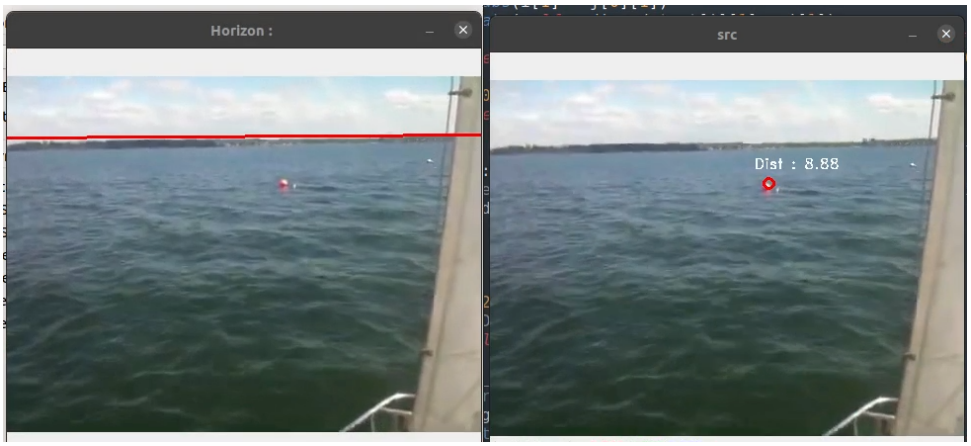


Figure 2.14: Example 2 on a video

From the estimated distance, we can see that the algorithm is doing well, with a few meters of uncertainty and a little flickering effect.

2.5 Integration in the global project

The implementation in the Gazebo environment was done using the same Object Detector class, applied on the published camera topic image, and we publish a *Float64MultiArray* message containing the coordinates of each detected objects in a successive form of [x1, y1, z1, x2, y2, z2, ...]. The ObjectDetector have been simplified using only an HSV mask to detect yellow objects, as it is easier for now and faster in computation time. The results can be seen in Figure 2.15 and 2.16.

In order to use the coordinates of the detected object in the global we had to transform our values. With our program, we have a distance between the object and the camera but we must have the coordinates of the object in the simulation referential. We implement to first make a transformation between the 2D coordinates of the object in the image to a 3D coordinates in the camera referential :

Figure 2.15: Example 1 on the simulation

Figure 2.16: Example 2 on the simulation

$$X = (u - c_x) * Z/f_x \tag{2.15}$$

$$Y = (v - c_y) * Z/f_y \tag{2.16}$$

where X and Y are the 3D coordinates of the object, u and v are the 2D coordinates of the object, c_x and c_y

2.6 Results and discussion

We saw, with the results, that the horizon detection and the ball detection works well (Figure 2.17) but the distance estimation is conditioned to calculate precise distance. It requires to know the radius of the buoy (estimated as a ball in the simulation). While it's possible to have this information in the Gazebo simulation, we won't be able to acquire it in a real life experimentation.

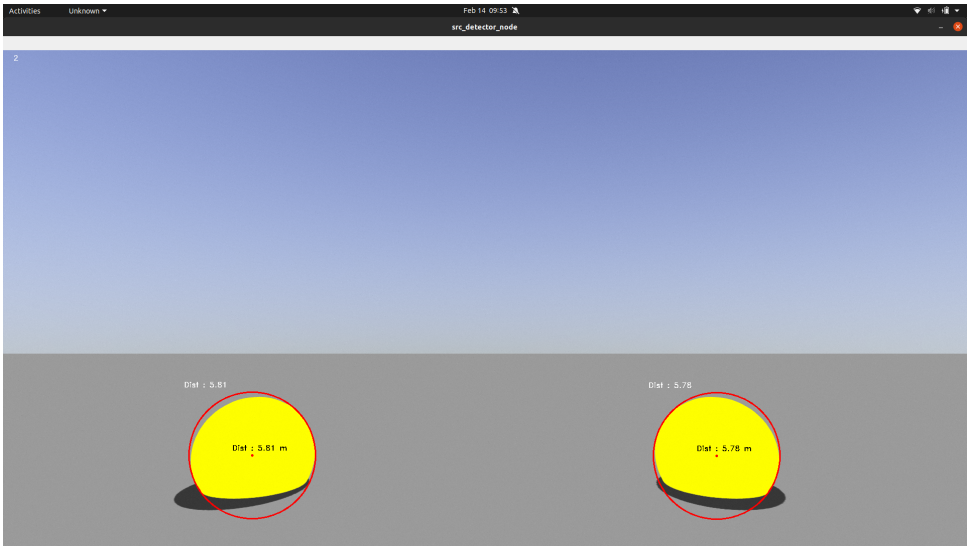


Figure 2.17: Distance estimation of two buoys

Moreover, the modelisation of the buoy in the simulation with a basic shape and a uniform color simplify the detection. We could have more mistakes or a precision loss with classic buoy. It's the same case for the horizon, but the simulation adds another problem. Because the horizon distance estimation is based on curvature of the Earth (approximated as a sphere), it brings some inaccuracies as the world in the simulation is a perfectly flat plane. Nonetheless, those inaccuracies doesn't impact the final results, as we can see in this video.

Chapter 3

AIS

3.1 Architecture

Here is a diagram that summarizes our AIS architecture:

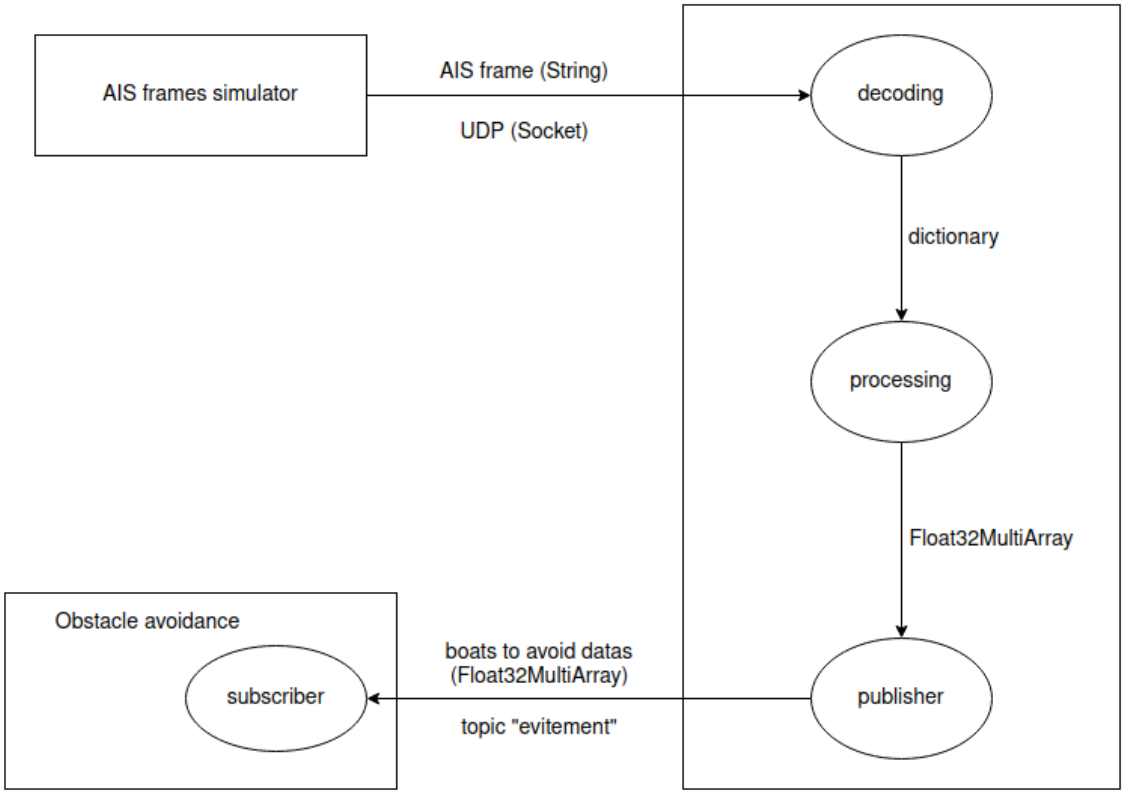


Figure 3.1: Architecture

3.2 Creating the frames using a custom simulator in python

3.2.1 Matplotlib Simulation

We made a python simulator that simulates a given number of boats. While running, the simulator uses the 'pyais' python module to encode the boats position, their mmsi (maritime mobile service identity), speed, heading etc. The frame used here is an AIDVM message of type 19 from the NMEA norm. Once the message is encoded, it is sent using a socket to the AIS of the main boat. Here is what an illustration of it:

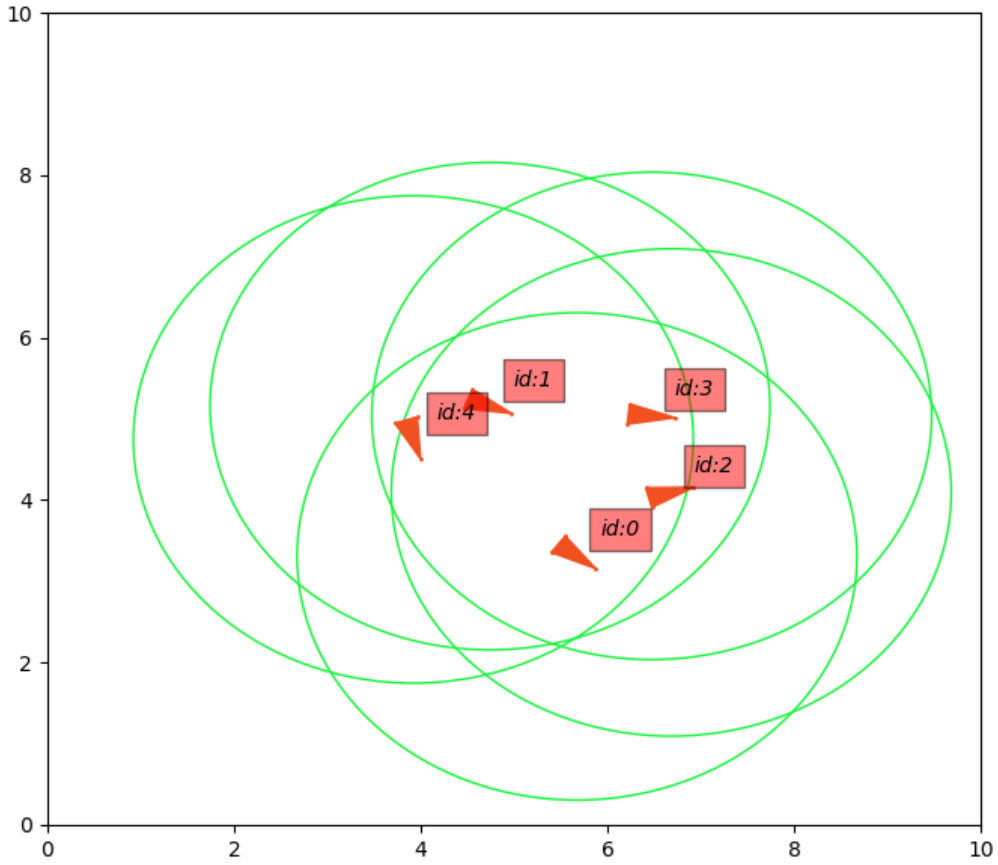


Figure 3.2: Python Simulation using matplotlib

This simulation sends the simulated boats via an UDP socket. They can be recovered via an UDP socket and used.

3.2.2 Rviz Simulation

We also created a simulator in rviz. It simulates the boats as rectangle and the range of the boat as red transparent spheres. Here's an example:

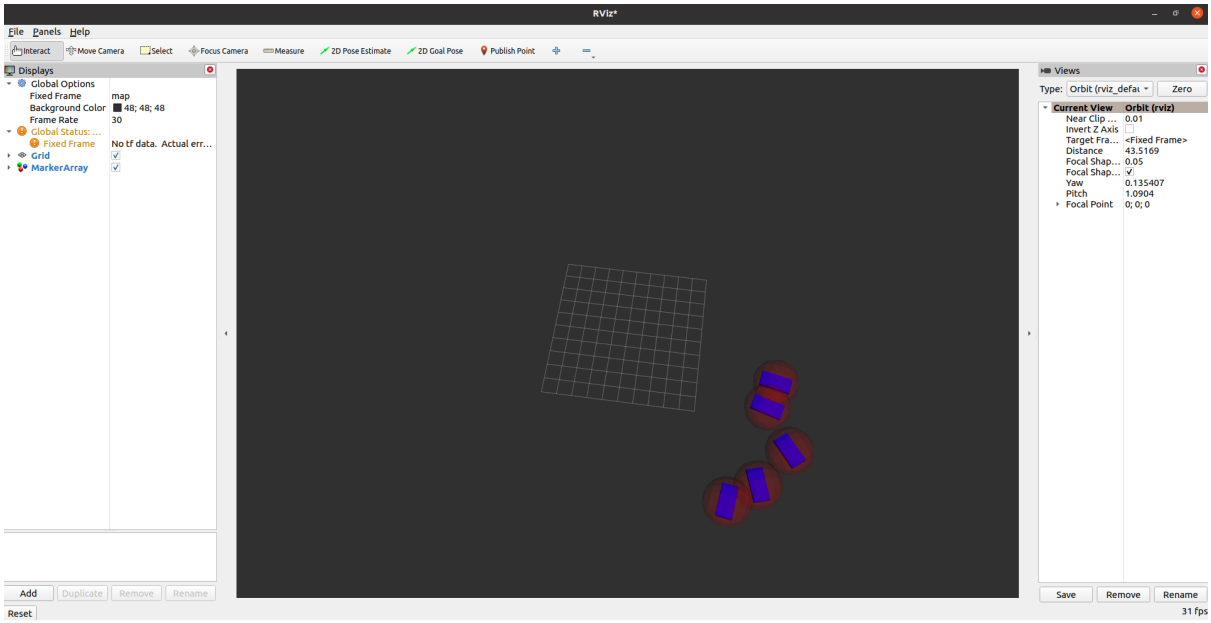


Figure 3.3: Python Simulation using rviz

This simulation is different. It sends its boats using a Float32MultiArray message directly on a ros2 topic ('/boat_pub_id').

3.3 Receiving the frames

We receive AIS frames from the simulated ships via an UDP socket. Those frames are received as string messages, and they can be in one piece or two pieces, in which case we store the first one until we receive the second and then unite messages. The python library that we use is very well done because it can concatenate messages received in multiple parts.

3.4 Decoding the frames

To decode those frames, we used the python library 'pyais' that does the encoding and decoding of AIS messages, with the following lines of code:

encoded_msg = pyais.encode_dict(data, radio_channel="B", talker_id="AIVDM")[0]
decoded_msg= decoded.asdict(pyais.decode(encoded_msg))

We manage the decoded frames as dictionaries. For example:

Coded AIS frame:
"!AIVDM,1,1,,B,C002rs011mkw7rVk‘T<01JP000000000000000000000T5R2P0,0*2E"

Decoded frame as dictionnary:
{'msg_type': 19, 'repeat': 0, 'mmsi': 47852, 'reserved_1': 0, 'speed': 26.3, 'accuracy': False, 'lon': -122.345832, 'lat': 47.582832, 'course': 0.0, 'heading': 181, 'second': 0, 'reserved_2': 0, 'shipname': '', 'ship_type': 0, 'to_bow': 9, 'to_stern': 11, 'to_port': 4, 'to_starboard': 5, 'epfd': 0, 'raim': False, 'dte': False, 'assigned': False, 'spare_1': bx00}

3.5 Publishing the decoded messages

After having decoded the messages, we put them in a Float32MultiArray ROS2 message which is just an array containing respectively: the boat identifier(mmsi),the latitude of the boat (lat), the longitude (lon), the heading, the speed, the distance to the bow (to_bow), to the port (to_port), to the stern (to_stern), and to the startboard (to_starboard). This is published on 'ais_id' topic to which the avoidance group subscribes.

To further ease the sharing of information between our group and the obstacle avoidance group, we created a function for them that manages a dictionary of boats. The key is the identifier of the boat (mmsi), and the container is the last known information of the boat. If this boat has already been detected, we update its information. if it is detected for the first time we add it to the dictionary.

3.6 Results

We created a simulation in RVIZ which is not completely finished. The information concerning the boats is not published for the obstacle avoidance group, only the arrays for rviz are published.

We encountered difficulties at the beginning because we needed AIS frames from boats in order to verify our results. For that we needed a simulation but because of the challenge of creating a simulation in gazebo we could not have it rapidly from the simulation group. Therefore, we made the choice of making a very simple simulation and display it using matplotlib so that we can test our AIS system. Moreover, we noticed that is was a good choice because other groups such as the obstacle avoidance group used our simulation to test their algorithms since it was only a file, thus being easy and fast to launch.

The work done is almost plug and play when going on the real system. Indeed, since we used sockets, it is sufficient for the real AIS to just sends its information on the right port and therefore easy to deploy.

Chapter 4

Reliefs

4.1 Problematic

All isles or seabed above a certain altitude are dangerous for the boat. They can stop it or even damage it. Therefore, we must know the topography of the field we are evolving in and the positions of the reliefs that could be of danger. The objective of this part is to preprocess a DTM to determine prohibited areas and inform the avoidance group of these dangerous areas. The specificity of these obstacle is that they do not move, meaning that they can be known in advance.

4.2 State of the art

Even though several articles about DEM and contour lines exist, they are quite unrelevant in our situation, because we focused on extracting contour lines. The thing is, this is a rather basic operation which would not suit for a whole scientific article. This is why most of the subjects we found was about optimizing or improving those extractions or even the DEM itself. However, some articles still are of interest, like like [5], even though it focuses on first optimizing the DTM. Moreover, some maps are not turned to a point cloud format yet, this is why it can be interesting to have a method to convert them. This can be done with [6].

4.3 Strategy and results

4.3.1 Choice of map

The choice of the map we will have to preprocess is the first step to determine dangerous areas. Each map has its advantages and drawbacks. We decided to focus on three types of charts: DTM, raster marine and vector marine.

The raster marine chart are nautical paper chart digitised. They contain a lot of information such as sea level, shipping routes or fishing areas. For our use, these charts contain too much information. We only need the sea level, and on these charts this information is drown under too much symbols and other lines. Furthermore, in order to extract the contour line of the sea level, we should have used image processing and contour detection, then determined the contours that are sea level and those which are not to keep only the sea level. So we decided to leave this type of chart aside.

The vector marine chart is an intelligent version of the raster marine chart. It provides the same information, but some features such as lights or buoys are clickable and contain more information. This kind of charts could have been interesting, if we could have access to one. It was impossible to find one for free, without material to work with, we decided not to choose this type of chart either.

The DTM is a map realised with a point cloud. Each point supply three information: longitude, latitude and depth. The format is easily treatable, and DTM are freely obtainable on the Shom webside: https://data.shom.fr. The data is stored in a txt file as a table of depths, abscissa and ordinate represents the position in latitude and longitude of the points. One corner point have its GPS coordinates in the header, and each point is equally distant from the previous with a constant step provided in the header. We decided to use this kind of map, even if it is voluminous.

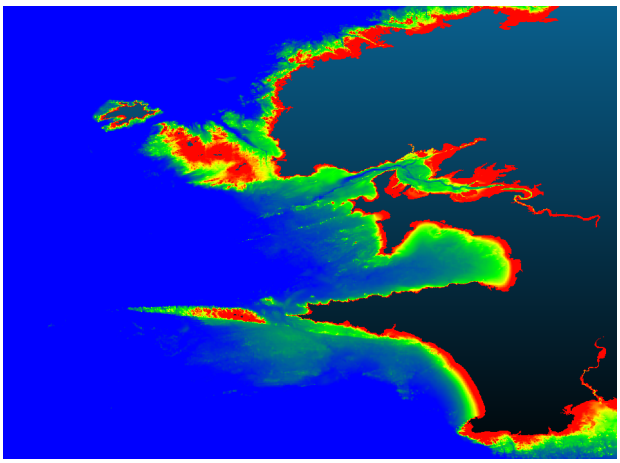


Figure 4.1: Example of used DTM

4.3.2 Preprocessing

Because the position of the reliefs is known and will not be modified with time, some of the work can be done before the start of the mission. Given a DTM, the goal is to extract the contour lines we are interested in and to save them in a file in order to call them later.

The process is done in a python file. The taken input is a DTM of the explored region in a .asc file. This can be generated with CloudCompare. In this DTM, we find a header with a description of the file and the number of points, and at least three columns representing the position and the altitude of the points. We can also signify the path of the map and the desired levels to be extracted with variables within the program. By default, their values are "/data/mnt_pyth_3_full.asc" and [-30, -25, -20, -15, -10, 0]. To extract the lines, we use the function contour of matplotlib.pyplot. Each contour depth is then saved in a csv file with the level in its name.

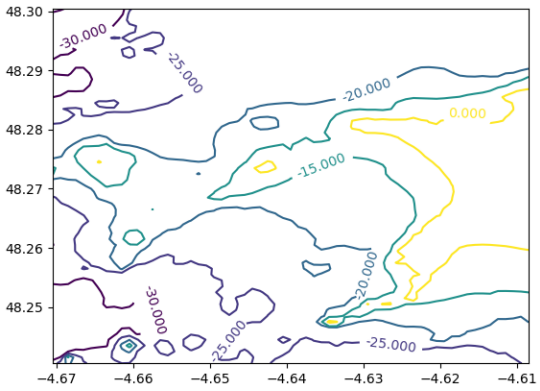


Figure 4.2: Contour lines obtained from extraction

4.3.3 Publisher

Our contour lines are useful to avoidance and simulation, but with different formats.

The avoidance need the contour lines as arrays. We publish the contour lines of a choosen depth as a "std_msgs/msg/Float32MultyArray" point per point, each point have two dimensions, longitude and latitude. Each feature in the map is store one after the other, it means that all contour points of an isle are stored in the array, then an other isle, etc.

On the other hand, simulation need a certain contour line as a point cloud, so we also publish the contour line as a "sensor_msg/msgs/PointCloud2". This time, we have to transfer the long/lat dimensions in the local coordinate system. To do so we are using the python package pyproj.

Chapter 5

Obstacle avoidance in a complex marine environment

5.1 Problematic

The detection of an obstacle is one thing, but the avoidance is another. The objective of this part is to exploit the obstacle detection (Charted landmarks, boats, UFOs) obtained in the previous sections to implement an optimal and guaranteed avoidance algorithm. This algorithm will then provide navigation instructions that must be applied to reach the arrival point.

It is important to remember that the boat will be navigating in complex environments, especially with boats piloted by humans, and therefore potentially unpredictable. Therefore, the avoidance, in addition to being dynamic, must act logically so as not to disturb other navigators. The algorithm must guarantee a path or indicate the impossibility of the request so as not to place the boat and its crew in a dangerous situation. For obvious reasons of navigation time, resources, etc., it must also be as optimal as possible.

In our case, it is not possible to use off-the-shelf avoidance algorithms. Indeed, the size of the navigation map, the complexity of the landform and the number of boats in a high traffic area, may generate important computational complexities and therefore saturate the onboard computer. It is therefore necessary to research the various existing algorithms, to define their strengths and weaknesses and to establish an avoidance strategy that allows a good compromise between calculation time and path optimality. However, the path must always be guaranteed to ensure the integrity of the carrier during navigation. These different topics will be discussed later in this report.

5.2 Strategy

The overall objective of our group is to : **Give the COG (Course Over Ground, heading to the bottom to the next waypoint) instruction to bring the boat from the starting point to the arrival point**. This objective can be divided into three parts:

- Search for the global path
- Local avoidance
- Verification of the path

The search for the global path will allow you to plot the boat's route on a map of fixed obstacles, the landforms.

Local avoidance will deviate the boat to avoid dynamic or unpredictable obstacles, UFOs and other AIS equipped boats.

Path verification will finally ensure the feasibility of the calculated routes.

We are thus the link between the perception groups (landform, UFO and AIS) and the control group. Our work differs from the control group firstly by the fact that we make the guidance, that is to say that we calculate the instruction and not the command of the actuators. Also our work is purely kinematic, we do not take into account the external disturbances in the guidance but only the capacities of movement of the boat (kinematics).

Finally, to explain our software interfaces, it should be noted that the global path search is done offline and the local avoidance online. Indeed, in discussion with our customers, *Pixel sur Mer*, we came to the conclusion that the routing could be done on the port or upstream. This is compatible with the calculation time of the route which can be quite long. Then the obstacle avoidance is inevitably done in real time because the principle is to avoid obstacles that are not yet known because they are dynamic or not predictable.

However, the route will be taken into account during local guidance by giving a local objective point (different from the global objective) which is the intersection of the point furthest away in time from the route with the perception boundary of the boat.

Offline pathfinding:

- Input:
  - Map of landform with a size that allows at least the visibility of the starting point and the objective point.

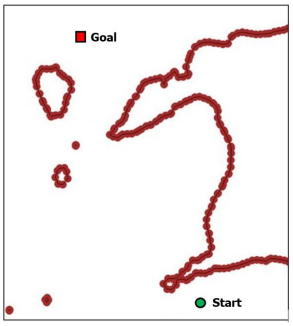


Figure 5.1: Example of a landform map with an initial position and an arrival point

- Output:
  - Route to take to reach the end point from the starting position.

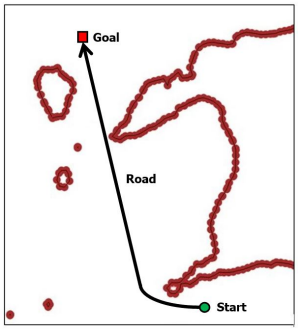


Figure 5.2: Route calculated from the previous example

On-boad local avoidance:

- Input:
  - Intersection of road and perception boundary = Local objective
  - landform, UFOs, AIS boats in the form of a point table with a radius approximating the size of the obstacle.

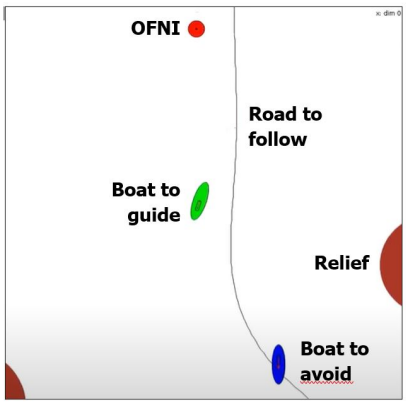


Figure 5.3: An example of obstacle's local configuration

- Output:
  - Setpoint in COG (Course Over Ground, heading to the bottom to the next waypoint).

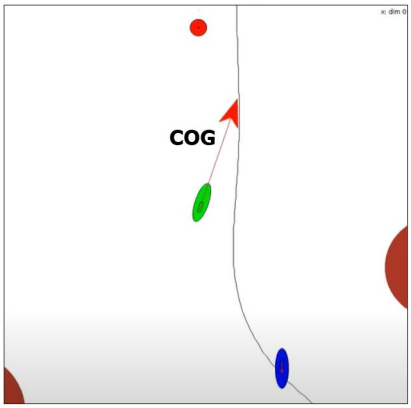


Figure 5.4: COG calculated from the previous example

5.3 Local obstacle avoidance through the use of potential fields

One of the steps in our strategy is the implementation of local avoidance to avoid unforeseen obstacles. To do this we used the artificial potentials algorithm and we will now detail this choice.

Why artificial potential fields. Among the local avoidance algorithms, we can note two techniques: the artificial potential fields and the Bug algorithm. The Bug algorithm is similar to the technique generally used intuitively to solve a maze. We choose a left or right direction and the idea is to always follow the wall that is to the left of us or to the right depending on the choice. This way, we make sure to follow a set of walls belonging to the same circuit. Then you just have to repeat the operation on a different circuit until you find the goal. The Bug algorithm adds in addition to this intuitive method the knowledge of the direction of the objective in relation to our current position. We try to fly as much as possible from our position to the goal and if we can't we follow the walls. This writing of Bug is not complete, that is to say that it does not ensure to find a path, a variant of Bug, Bug2, is completed.

The technique of artificial potential fields will make the robot move on a surface of energy. We make sure that the point of space where the objective is located is a point of minimal energy, globally minimal. We then move on this surface so as to minimize the energy by taking for direction the opposite of the gradient of the surface in our position. Thus we hope to fall in the global minimum which is our objective. The opposite of the gradient of the energy surface at a given position is in fact a virtual conservative force. The problem with this method is that we can fall into a local minimum without being able to get out. It will then be necessary to detect the blockage and avoid it by computing a route with a full motion planning algorithm such as A star on the local perception map.

The bug algorithm if you use version 2 always reaches the goal. However it follows the edge of the land coast. This is already dangerous because the probability of running aground on the land is higher. Moreover, it slows us down because we do not take into account the effect of obstacles at a distance, we bypass them only if they block the flight of birds towards the objective when we are close to them. This is why we choose to use the method of artificial potential fields instead.

The different levels of implementation of the potential field

- **Level 1:** The calculated potential field ensures that the objective is a global minimum. Moreover, we manage the multiplicity of repulsive obstacles that the perception groups send us. Here the repulsive potentials will be centered on the centers of the obstacles. The radii modeling the obstacles' clutter will be used to modify the impact of the repulsive potential at a distance.
- **Level 2** We manage the blocking of the boat in a local minimum by giving it a mini route in the local perception map when it is blocked.
- **Level 3** Knowing the wind direction, we translate by potentials the kinematic impossibilities of the boat.

Result. Since there is in addition to the level advancement of the potential field algorithm, the interfacing work :

- Interfacing within the group : Recovery of the local objective = Intersection of the global route with the perception boundaries of the boat.
- Interfacing with other groups: Retrieving obstacles with perception groups and sending the instruction to command groups.

We made sure to validate level 1 with the interface with the other groups.

5.4 Space discretisation and obstacle avoidance using the A* algorithm

The global avoidance allows to trace the route to follow to reach the objective point by avoiding fixed obstacles such as landforms. Local avoidance will allow to avoid other mobile or unpredictable obstacles.

Choice of the algorithm

The algorithms allowing to establish the trajectory to follow to reach the arrival are mostly algorithms requiring to discretize the space and are based on the Dijkstra path search algorithm.

Of all the possibilities, we have chosen the A* algorithm. The algorithm is a variant of the Dijkstra algorithm to which a heuristic function has been added that directs the path search in the direction of the goal to be reached. Thus, it does not explore all possibilities and directs the path to the goal. It does not guarantee to find the shortest path unlike Dijkstra. But it allows to find one of the best paths more quickly and with less memory requirements since it does not process all the nodes. In addition, this algorithm can take into account obstacles such as landforms and avoids them.

Therefore, it seems to be one of the best algorithms to solve our problem.

Discretization of the space

Path finding algorithms are mostly based on a discretized map of the space in which they have to find a path. This is notably the case for the A* algorithm. The choice of the discretization step is an important factor on the speed of the path determination. A larger step size can speed up the convergence of the algorithm, but it can also lead to a less accurate solution. Conversely, a smaller step size can increase the accuracy of the solution, but it can slow down the convergence. It is therefore necessary to determine the step size that is adapted to the search map as well as to the PC available to calculate it (RAM memory space) and the maximum time to determine the path. Note that the path will be calculated before the navigation and will just be adapted if the boat moves too far away from it during the navigation.

The chosen step is 50m in our simulation. The points are therefore too far apart to be able to find the objective point of the boat. To overcome this problem, an interpolation was applied to calculate intermediate points between the points generated by the algorithm. This can result in a smoother and more accurate trajectory by using a larger step size for the A* algorithm.

It is therefore concluded that the space must be discretized to use the A* algorithm. The discretization step chosen is large to reduce the time and memory required for A* to converge. An interpolation was then performed to improve the route found.

Communication of the global path during the mission

The calculation of the global path is done before the mission. During the mission, a local objective is given to the local avoidance algorithm. It is then necessary to extract the objective for the local map so that the boat follows the global path.

The position of the boat is retrieved from the simulation. Once the boat is placed on the map, the farthest temporal path point in the boat's vision (local map of $250m^2$) is sent as the local objective to the local avoidance. It is sent on a ROS topic named 'localGoal' of type 'PointStamped'.

If the route to follow disappears from the vision of the boat, the A* algorithm must be called again. A new path search is then performed by taking as a starting point the current position of the boat. It is therefore important that the algorithm works quickly enough because it will be impossible to give a correct instruction to the local avoidance during the time of calculation of the new path. An improvement would be to search for a new route to the goal by stopping as soon as a node belonging to the old path is processed because we know the path to follow from this node to reach the goal. This would reduce the computation time of the algorithm.

5.5 Obstacle avoidance and reachability through the use of Intervals

The different algorithms and simulations described above consider our robot as punctual. They also don't take into account the uncertainty that we can have on the position of the robot after it is given a command . In a context of autonomous navigation where we need to ensure that the robot will navigate without any risk of collision, these approximations can be dangerous to the integrity of the system. The use of a modeling by Intervals can thus be used to take this parameters into account.

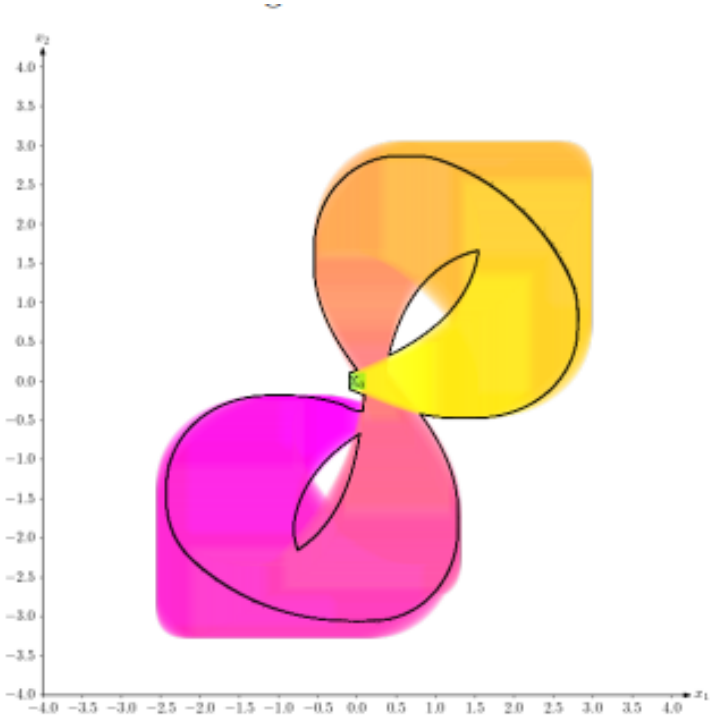


Figure 5.5: Tube of the possible trajectories considering the initial state of the robot

The problem of being able to tell for sure that an obstacle will never be reached is called *Unreachability*. The main difficulty in a continuous model like this one is to ensure that at every time and considering the uncertainties, no possible position of the robot will endangered it.

Lie groups and especially Lie symmetries can be applied to Interval integration to solve both the Unreachability and the Reachability problem(ensuring an area will be reached)[7].

Once applied on the tube of the possible trajectories, the algorithm using Lie symmetries should return an empty tube, meaning that no trajectory is crossing the obstacle area. However if such a trajectory exists, the algorithm is able to return the part of the obstacle that may be encountered by the robot. This information can be given back to the pathfinding and obstacle avoidance to correct the planed trajectory in order ta make it safer. We can then loop until no more risk of collision is detected. We can then confirm our path to follow and start the mission.

5.6 Development of a simplified simulator for the comparison and validation of avoidance algorithms

In order to estimate the performance and the limits of our strategy, it is necessary to be able to test our programs on different scenarios. For this, the use of a simulator is more than necessary. In this section, we will present the simplified simulator that we have developed, as well as the simulations that have been carried out.

Why design our own simulator and not use a realistic one? As will be detailed later in the report, one of the working groups was in charge of developing a realistic simulator to simulate a boat with realistic physics, an onboard camera and a positioning system. However, we chose to develop our own simulator for several reasons:

- **Developing time:** In order to verify our algorithms, we would have had to wait until the end of the development of the realistic simulator and therefore postpone the testing phases. Moreover, if the tests showed that the strategy was not correct, the implementation of a new strategy would have caused significant delays in the project demonstration.
- **Dependence on the work of other groups:** The data provided by a realistic simulator are usually raw measurements that require processing and analysis to extract relevant information. However, this extraction work is precisely the task of the other groups. We would then have two choices: redo the work of the other groups or wait until the other groups have finished to do our tests. In any case, neither of these configurations is desirable.
- **The heaviness and rigidity of a realistic simulator:** The resources needed to run a realistic simulator are often significant. Furthermore, changing the scenario or the behaviour of an obstacle is often time-consuming and requires preparation. These constraints are not consistent with the testing phases of avoidance algorithms, which often boil down to a question: Did I achieve the objective without endangering the carrier?

For all these reasons, we considered it more appropriate to develop a simplified simulator. This simulator is qualified as simplified for several reasons:

- **Using a 2D display:** For our use case, it is not necessary to have a 3D display, a view from above is more sufficient.
- **Representation of obstacles in their processed forms:** In the case of the UFO for example, we do not need the camera image but only the position of the UFO in relation to the boat. Size information is also relevant to know the avoidance radius of the object.
- **Use of text files to import obstacles:** The use of configuration files to declare and load obstacles makes it easy to change the simulation scenario

In addition to these simplifications, we have also used the library which allows us to have a dedicated display and which does not consume the resources linked to the updating of the simulation. All these simplifications have had the impact of significantly reducing the weight of the simulator and making it very modular.

What obstacles can be simulated by this simulator? In order to remain relevant to the project, our simulator must model a range of obstacles representative of those that the yacht may encounter. We have grouped these obstacles into 3 categories:

- **Landform:**

  The landform class corresponds to obstacles that are immobile in the terrestrial frame of reference and that are located or identified by external data such as maps. In our case, we will use landform type obstacles to model the coastline but this class can also be used to model offshore platforms, restricted areas or other types of well known obstacles to be avoided.

This type of object is often continuous and has a safety zone that must not be crossed to avoid any danger. In the case of the coast, this zone corresponds to an isobath that must not be crossed to avoid the boat's hull touching the bottom. However, a continuous representation would be too heavy in terms of memory and display. We have therefore chosen to discretize these obstacles into several points and to assign to each point a safety radius that must not be penetrated. This safety radius can be defined with the help of the depth map or also in relation to the discretization step of the landform.

For example, for a coastline with a 50m step between each point and with a 20m isobath limit, a safety radius of 50m will be used to avoid crossing the isobath and to avoid the risk of passing between two points on the coastline. This simplification makes it possible to easily represent a landform with a very low memory cost. The next figure shows the case of an island that has been broken down into 4 points with a safety radius of 50m. This island had a complex contour but it could be simplified into 4 points, this resolution is certainly rough but it allows to avoid the island for sure. By decreasing the discretization step and thus increasing the number of points, it is possible to obtain a detailed resolution of the island if one wishes to pass it as close as possible.

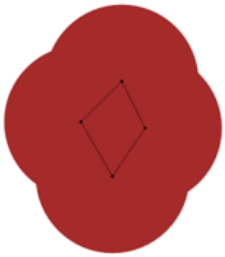


Figure 5.6: Simplified representation of an island

- **Boat:**

  The boat class corresponds to mobile obstacles whose location is obtained by sensors present on the carrier and whose behaviour can therefore be random. For example, in the case of boats, they can be located thanks to the AIS and can have a random behaviour which will depend on the person piloting it. This class could also contain marine mammals such as whales that we wish to avoid but which may have random behaviour.

  To represent this class we have based ourselves on the information available from the AIS such as position, speed, heading and size of the boat. The boat is represented by a rectangle corresponding to its size, its centre corresponds to the AIS position on the vessel and its speed and heading are represented by a speed vector. In addition to this, we want to avoid getting too close to the vessel and therefore we need to define a safety zone that should not be crossed. This zone is represented by an ellipse whose centre is at the front of the boat. This configuration ensures that the boat is always included in the safety ellipse but also that the area in front of the boat is more important to avoid than the area behind the boat. In short, it is better to pass behind a boat than to cut it off.

  The next figure is the representation of a boat in the simulation. The rectangle corresponds to the boat, the red arrow to its speed vector, the points are respectively the position of the AIS and the centre of the safety ellipse shown in green.

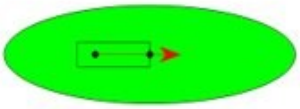


Figure 5.7: Simplified representation of a boat

- **UFO:**

  The UFO class corresponds to mobile obstacles whose location is obtained by sensors present on the carrier and whose behaviour is therefore generally simple (immobile or oscillating). This class can be used to represent any type of drifting obstacle. As this class can represent any type of obstacle, its representation must be as general as possible. For that, we based ourselves on the way in which a UFO could be represented following a detection by an optical means.

  The information recovered will be the position of the centre of the UFO and its apparent width. A position and a distance make it possible to characterise a circle and the objects of this class will thus be represented by a circle whose centre corresponds to the position of the obstacle and whose radius is equal to the apparent width of the obstacle with a safety coefficient. As the obstacle may not be very mobile, it also has a velocity vector allowing it to be given a speed and a heading. The behaviour of the obstacle can be defined by the user according to the scenario. The next figure shows an UFO type obstacle.

Figure 5.8: Simplified representation of an UFO

Conducting a simulation on a given scenario. Shortly before this project was due, we were able to interface the simulator with the avoidance algorithms and thus run a simulation on a defined scenario. The scenario takes place on the Brittany coast in an area of 6km by 6km with various landforms such as a coastline or islands. There are also the presence of several boats and several UFOs scattered over the area. The following figure shows the simulator display at a given time. The display is divided into two windows, one to display the map in its entirety and another to display the local detection zone of the boat in which the potential field algorithm is applied. The black line on the global window corresponds to the path determined by the A* algorithm. We observe that in the local window the boat is not exactly on the line because it has just avoided another boat and therefore preferred to move away from its main trajectory to avoid the collision. This simulation went smoothly and our boat never collided. Further simulations will have to be organised to test the strategy in much more complex cases, especially on larger maps or on examples where there is sometimes no existing path to the finish. Lorem ipsum

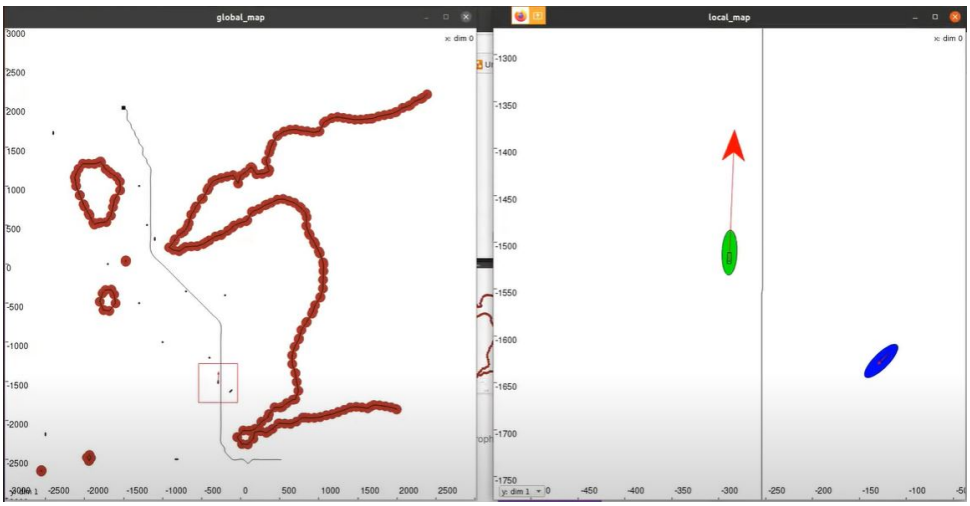


Figure 5.9: Example of a simulation

5.7 Result

The problem of obstacle avoidance being quite complex, it has been chosen to use two different types of algorithms to solve it.

First a discrete pathfinding algorithm such as the A* allows us to find a way from our starting point to our objective while avoiding landforms. This kind of algorithm is efficient but is still pretty heavy to run and can't be runned multiple times in a row to consider moving obstacles. So another layer of obstacle avoidance is necessary to address specifically the issue of moving obstacles.

Potential fields are suited to this task. They allow us to create a repulsion between our vehicle and any obstacle that we want to avoid. This method is way lighter to run and has the only drawback that we can't state for sure that it will be able to find a path if one exists.

By combining these two algorithms we are able to ensure that we are able to find a path if one exists and we are able to avoid obstacles on our way. We can even define different strategies by selecting the form of the repulsive field depending on the nature of the obstacle (UFO, other boats ...)

These algorithm still have the drawback of considering our vehicle as punctual. An implementation of Minkowski algebra or Interval modeling could help ensuring that the shape of the vehicle is taken into account when finding a path, by making obstacles larger if needed. A loop between the pathfinding and the safety checking could then be done until a safe path is found to then be able to follow it without any risk.

Chapter 6

Contrôle

6.1 Structure

6.1.1 Subscribers

- AHRS
  Type : sensor_msgs::msg::Imu
  Contient : Acceleration, gyroscope, orientation.
  Pour ce capteur seule l'orientation (quaternion) nous intéresse pour en déduire le cap. Donnée par le groupe simulation.

- Cap à suivre ($\theta_{consigne}$)
  Type : std_msgs::msg::Float64
  Contient : Le cap à suivre donné par le groupe évitement.

6.1.2 Publishers

- Angle of the rudder δ_r Type : std_msgs::msg::Float64
  Contient : L'angle en radian entre $-\pi/4$ et $\pi/4$ du rudder.

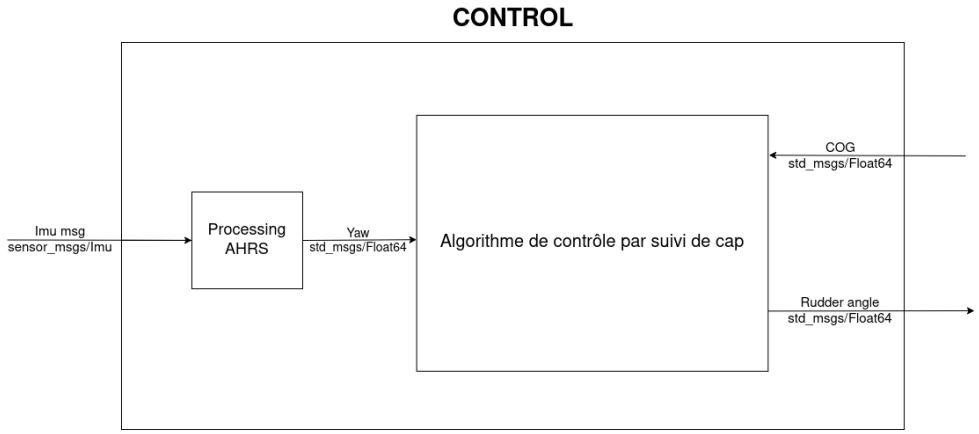


Figure 6.1: Architecture logicielle

6.2 Contrôleur

6.2.1 Suivi de cap

Notre objectif est de suivre un cap:

Un skipper est à bord d'un bateau. Il veut imposer un cap en ayant connaissance des informations par rapport au vent. On considère ainsi impossible que le skipper décide d'emmener son bateau face au vent.

On a donc accès qu'à un seul capteur qui nous intéresse : l'angle du bateau θ.
De plus, le skipper impose un cap $\theta_{consigne}$.

Le contrôleur s'écrit donc de la manière suivante [Fig.6.1] :

inputs : $\theta, \theta_{consigne}$
$\hat{\theta} = 2 * atan(tan((\theta - \theta_{consigne})/2))$ $\delta_r = \frac{\delta_{rmax}}{\pi} * \hat{\theta}$
outputs : δ_r

Table 6.1: Algorithme de contrôle pour un suivi de direction.

Nous avons le theta à suivre $\hat{\theta}$ qui correspond à un sawtooth entre notre cap et la consigne, puis il convient de ramener la consigne du rudder entre -1 et 1 en divisant par π, puis de le borner avec ses angles max.

Ainsi, un groupe s'occupant de la simulation du projet nous envoie le cap du bateau θ alors que le cap à suivre nous est donné par défaut par le skipper. Mais ce dernier peut aussi nous être fourni par le groupe évitement. En effet, si un obstable est détecté le cap à suivre doit être changé.

Ainsi, nous pouvons compléter le bloc d'algorithme dans la figure 6.1.

6.3 Verification du contrôleur

Pour vérifier le contrôleur, nous avons développé deux programmes python utlisant la bibliothèque *roblib* Fig.6.2.

- *display.py*: programme python qui se lance avec la commande *ros2 run control display.py*. Ce programme s'abonne aux différents capteurs pour vérifier que l'angle du rudder renvoyé est cohérent.
- *simulation.py*: fonctionne indépendemment de ROS. Permet de tester l'algorithme de contrôle avec une simulation dynamique du voilier.

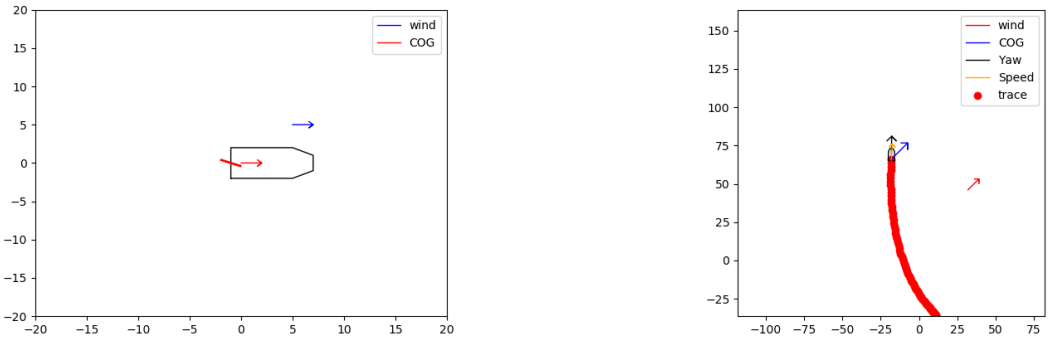


Figure 6.2: Simulations pour vérifier notre controlleur

6.4 Conclusion

Notre controlleur fait bien du suivi de cap. Il a été testé avec le groupe simulation et la mission donnée initialement a été remplie.

Il y a cependant des limites à ce contolleur :

- Si le skippeur donne un cap face au vent, le bateau s'arrête. Il faudrait ainsi aussi controller la voile.
- Si il y a du courant, le cap du bateau est bien là où l'on donne le cap désiré mais la trajectoire sera déviée. Il faudrait donc faire un suivi de droite à un instant t pour un cap donné. (Cela implique de prendre aussi en compte le GNSS.)

Chapter 7

Simulation

L'objectif de la simulation est de fournir aux autres groupes d'une part un visuel qui leur permet de voir si leurs programmes fonctionnent et d'autre part des données des capteurs dont ils ont besoin. La simulation est réalisée à la fois sur RViz et sur Gazebo. La plupart des capteurs sont simulés directement via ROS excepté la caméra qui a besoin d'un visuel et qui fonctionne avec Gazebo.

7.1 Gazebo

Nous avons choisi Gazebo pour nous former au logiciel pour la simulation physique du voilier, mais également dans l'optique de réaliser un code utilisant des outils standards. L'interêt d'utiliser les standards est de permettre la maintenance et l'amélioration du code et ainsi de faciliter l'interfaçage avec les autres groupes du projet.

Gazebo est un un logiciel de simulation et de visualisation qui a pour principal intérêt de s'interfaçer simplement avec ROS. Cependant, Gazebo a représenté de grosses difficultés pour notre groupe. Il présente de nombreux problèmes de stabilité, rendant la répétabilité des codes assez aléatoire et demandant beaucoup d'attention et de méfiance pour éviter les nombreux pièges et erreurs. La quasi-absence de documentation, cours ou tutoriels compatible avec ROS2 nous a rendu la tâche difficile.

7.2 Capteurs

7.2.1 IMU

L'IMU est simulée sous ROS via un topic '/simulation/boat/imu' qui publie un message de type 'IMU'. Pour la construction de ce message nous nous récupérons le topic de la position et l'orientation du robot. Nous publions ensuite la vitesse linéaire et angulaire correspondant à la vitesse entre deux positions successives. Un bruit peut également être ajouté aux données qu'envoie ce message.

7.2.2 GPS

Le GPS correspond à un message ROS 'GeoPoseStamped'. De la même façon qu'avec l'IMU, nous nous abonnons à la position du robot. Une projection sur l'UTM 30 ('Transverse Universelle de Mercator') est ensuite réalisée. Il s'agit d'une projection sur le fuseau 30 (entre 6 degrés Ouest et 0 degré Greenwich) dans le système Géodésique WGS84 où la Terre est modélisée comme étant une ellipsoïde. La latitude et longitude du point d'origine ont été fixés respectivement à 48.344986 et -4.480286.

7.2.3 Camera

La caméra est le seul capteur pour lequel Gazebo est nécessaire. Il est en effet impossible de simuler une caméra sous RViz. Ainsi, pour la mettre en place, nous sommes passés par un plugin Gazebo qui permet de définir tous les paramètres intrinsèques de la caméra Ceux-ci ont permis au groupe OFNI de calibrer leurs modèles de façon adéquate. Le positionnement de la caméra a été réalisé dans le fichier en .xacro contenant la description du bateau, ici considéré comme un robot.

7.2.4 Girouette

La girouette correspond à la simulation du vent. Pour ce faire, nous publions un 'Vector3Stamped' dans le topic 'simulation/wind'. Dans un premier temps, nous avions des équations du vent variant sinusoïdalement selon x, y et z. Cependant, dans le but d'avoir une simulation plus simplifiée et garder un meilleur contrôle et une meilleure compréhension, nous avons choisi de conserver un vent fixe. Il est cependant possible d'avoir un vent variant en fonction du temps.

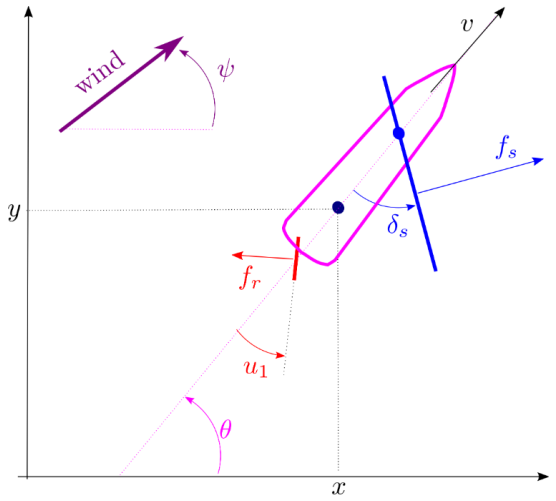
7.3 Interfaçage avec les autres groupes

7.3.1 Contrôle

Pour la simulation du voilier nous avons choisi de reprendre un modèle que nous avions déjà vu précédemment en guidage (Figure 7.1).

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

(a) Equations d'état du modèle



(b) Schéma du modèle

Figure 7.1: Le modèle choisi

Les différents éléments du modèle:

- x,y et θ correspondent à l'état du bateau (sa position en x et en y ainsi que son orientation dans l'espace).
- v est la vitesse linéaire du bateau.
- ω est la vitesse angulaire du bateau.
- fs est la force du vent sur la voile.
- fr est la force de l'eau sur le rudder.
- δs est l'angle de la voile.

- ψ est l'angle du vent vrai.
- wap est le vent apparent.

Par ailleurs, les coefficients p1 (coefficient de dérive), p2 (coefficient de trainée), p3 (frottement angulaire de la coque sur l'eau), p4 (portance de la voile), p5 (portance du gouvernail), p6 (position du centre de poussée du vent sur la voile), p7 (position du mât), p8 (position du gouvernail), p9 (masse du voilier) et p10 (moment d'inertie du voilier) sont fixés et repris du modèle Vaimos de l'Ifremer.

Nous récupèrons la consigne en position du ruddder envoyée par le groupe contrôle sous la forme d'un Float64. En parallèle nous calculons, l'angle maximal entre la voile et le bateau en fonction de l'orientation du vent. Grâce à l'état actuel du voilier et des consignes nous calculons la dérivée du vecteur d'état du voilier ce qui nous donne l'état du voilier à l'instant t+dt.

7.3.2 Relief

Pour visualiser les reliefs dans notre simulation, il suffit de sélectionner sur RViz le topic publié par le groupe Relief sous la forme d'un nuage de point. Ainsi, des cubes colorés apparaissent pour former le relief désiré (Figure 7.2 et Figure 7.3). Il faut néanmmoins régler manuellement la taille des cubes car ces derniers peuvent être trop importants dans un premier temps.

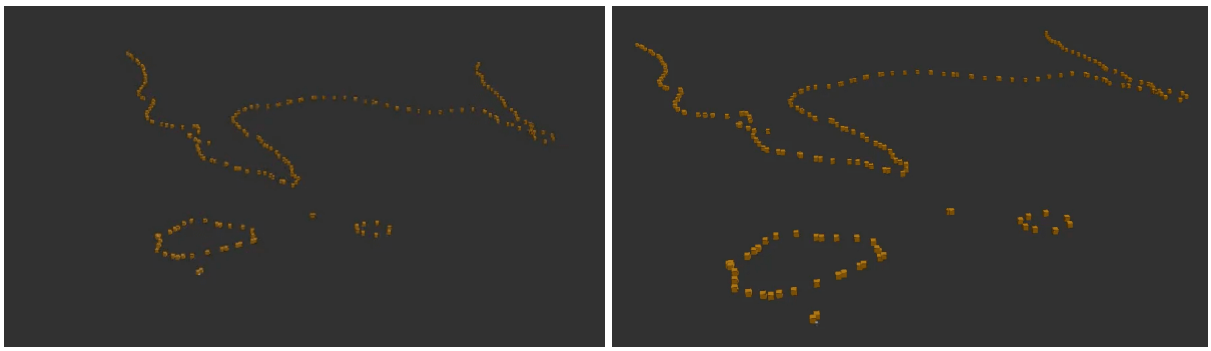


Figure 7.2: Visualisation des reliefs sur RViz

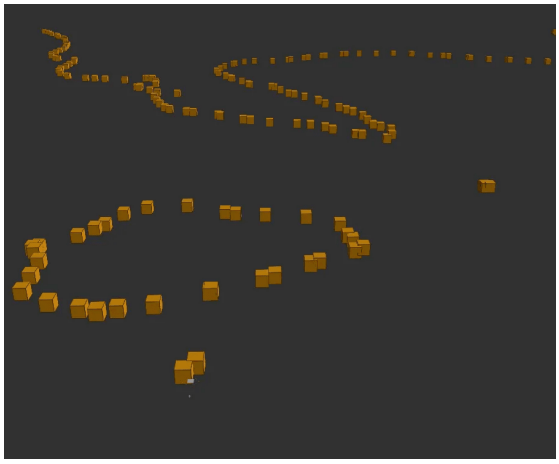


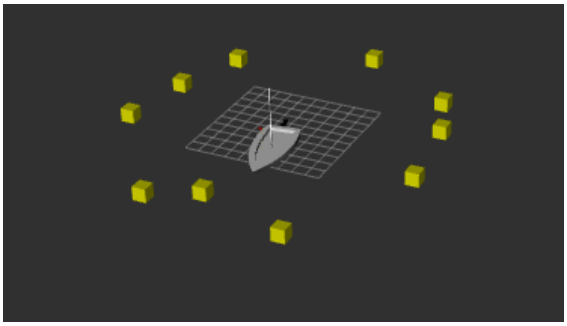
Figure 7.3: Visualisation des reliefs sur Rviz

7.3.3 Détection

Afin de simuler l'environnement nécessaire au groupe Détection, on utilise en majorité Gazebo, vu que ce groupe se base uniquement sur la caméra pour déterminer la position des obstacles dans le repère du monde. Lors de l'intégration du groupe à la simulation, on avait déjà la plupart des éléments présents sur RViz. Cependant, rien n'était encore fonctionnel sur Gazebo. Ainsi, pour qu'ils puissent mener leurs essais sans attendre, on leur a fourni un fichier en .world simulant la caméra, le bateau statique et un obstacle.

La simulation complète avec une intégration à Gazebo fut un peu plus compliquée à mettre en place. Les tutoriels et informations à notre disposition étaient parcellaires, et concernaient majoritairement ROS1. Cepedant, après de nombreux essais infructueux, nous avons trouvé la solution sous la forme des services ROS. Lors de la création d'un objet, Gazebo met en place un certain nombre de services qu'il est possible d'appeler afin d'avoir des informations par exemple. Ce qui est utile ici, c'est de pouvoir donner une position et une orientation de l'objet dans Gazebo. Ainsi, à chaque tour de boucle, la position du bateau et de chaque obstacle est actualisé en temps réel, en prenant appui sur les topics ROS qui permettent la simulation sous RViz.

(a) Vue de la caméra du bateau dans Gazebo



(b) Vue du bateau dans la simulation Rviz

Figure 7.4: Visualisation de la caméra et de l'environnement du bateau à un même instant

7.4 Affichage

Lorsque la simulation se lance nous avons un affichage sur RViz comprenant le modèle en trois dimensions du voilier, publié selon la position et l'orientation du bateau. Des obstacles placés aléatoirement sont également affichés. Pour une meilleur compréhension des données que nous recevons et publions, nous avons également affiché la barre du voilier pour voir si le contrôle envoyé est cohérent. (Si aucune commande n'est fournie, la barre aura une position neutre correspondant à un angle nul). Nous affichons également l'orientation du vent qui correspond à un vent constant sur la simulation actuelle (Figure 7.5).

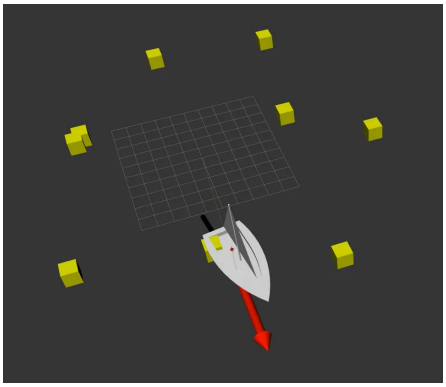


Figure 7.5: Visualisation de la simulation sur Rviz

7.5 Architecture de la simulation

Notre code se divise en deux packages, selon les standards ROS2. Le premier package, nommé sailboatdescription, regroupe l'ensemble des fichiers nécessaires pour générer l'arbre de tf (fichier .urdf et publisher). Le second package est simulation, son but est de regrouper l'ensemble des nodes nécessaire pour simuler la physique du voilier mais aussi l'ensemble de ses capteurs.

7.6 Perspectives

Pour compléter notre simulation, il faudrait ajouter l'affichage des bateaux détectés par le groupe AIS. Par la suite, il serait intéressant de créer plusieurs scénarios qui seraient séparés en fichier launch. On pourrait ainsi tester séparément un évitement d'obstacle, de bateau ou de relief puis avoir un scénario global qui fusionnerait toutes les contraintes afin de pouvoir tester l'ensemble des fonctionnalités en conditions "réelles". Il est possible de changer tous ces paramètres manuellement, mais il faudrait l'automatiser pour le rendre plus aisé à modifier.

Par ailleurs, certains tests avec d'autres groupes, notamment le groupe détection nous ont permis de voir que des calculs assez lourds ralentissent la simulation ce qui pose des problèmes au niveau de Gazebo. Il faudrait donc probablement changer certains programmes afin d'alléger les calculs liés à la simulation. En effet, dans l'état actuel, nous avons fait le choix de ne mettre aucune collision sur Gazebo, afin que la simulation que nous avons mise en place ne rentre pas en conflit avec celle du logiciel. Cepedant, il n'est pas possible d'arrêter complètement la simulation Gazebo car cela coupe le topic de la caméra, qui ne renvoie alors plus d'images. L'inconvénient de laisser la simulation tourner dans ces conditions est la présence de gravité, qui va faire tomber les objets entre chaque calcul de notre simulation. Cela se traduit par un déplacement légèrement saccadé du bateau et des obstacles, qui s'accentue lorsqu'on rajoute des calculs dans la boucle.

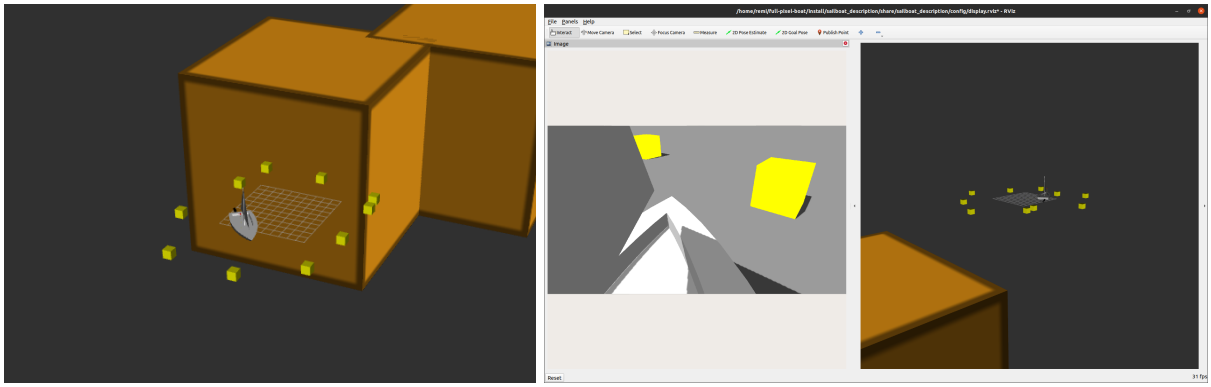
Chapter 8

Conclusion

At the end of the project, each repository was launched by using the git submodules on a new repository. Except the AIS' code that is not launchable with a lunch file, every piece of code runs well. The boat simulation displayed on RViz work perfectly, it been affect by the wind and response to control command, that is show by the saffron. Also reliefs and buoys are display on RViz, creating a whole environment, and allow the view of the obstacles avoidance(figure 8.1a). The gazebo simulation provide camera view for UFO avoidance(figure 8.1b). So the boat go on the desired point by dodging obstacles.

In split of this there still things to fix. Not all the groups have achieve all there goals. UFO team is not detecting moving obstacles, relief team is not taking in charge the tide, obstacle avoidance team have still work to do for implemented the global path, control team in not dealing with drifting, AIS team have to improve there code implementation, and simulation team have not simulate the physics of water. Finally, by lack of time, the BRAVE boat have not be use for testing our codes.

About displaying, the reliefs are not yet in the same plan that the boat, so the relief placement is not the same in RViz that in fact. We could also add other boats and reaching point on RViz. As well for simplify scenario modification, it could be good the add arguments in the launch file for the boat starting position, reaching position and wind power.



(a) View of the boat with reliefs and buoys (b) View of the camera topic and the environment

Figure 8.1: Views of the project under RViz

Bibliography

[1] Dong Liang et al. "Robust sea-sky-line detection for complex sea background." In: *2015 IEEE International Conference on Progress in Informatics and Computing (PIC)*. IEEE. 2015, pp. 317–321.

[2] Kaiming He, Jian Sun, and Xiaoou Tang. "Single image haze removal using dark channel prior." In: *IEEE transactions on pattern analysis and machine intelligence* 33.12 (2010), pp. 2341–2353.

[3] Ming-Zhu Zhu, Bing-Wei He, and Li-Wei Zhang. "Atmospheric light estimation in hazy images based on color-plane model." In: *Computer Vision and Image Understanding* 165 (2017), pp. 33–42.

[4] Vidya Nitin More and Vibha Vyas. "Removal of fog from hazy images and their restoration." In: *Journal of King Saud University-Engineering Sciences* (2022).

[5] François Lecordix Kusay Jaara. "Extraction des courbes de niveau cartographiques à partir d'un modèle numérique de terrain (MNT)." In: (2011).

[6] Pascal Guitton Joachim Pouderoux Jean-Christophe Gonzato. "Création Semi-Automatique d'un Modèle Numérique de Terrain." In: *Actes des 16èmes Journées de l'AFIG* (2003).

[7] Julien DAMERS, Luc JAULIN, and Simon ROHOU. "Lie symmetries applied to interval integration." In: (2022).