Simulation of an Autonomous Sailboat with ROS

Alexandre EVAIN – FISE ROB 2021
alexandre.evain@entsa-bretagne.org
alexandre.evain76@gmail.com

Acknowledgement

I would like to thank Jian Wan from the University of Plymouth who made this internship possible and maintained it through weekly videoconferences despite the special circumstances of Covid-19 and the impossibility of coming to the United Kingdom to do the internship on site.

In addition, I would like to thank my senior professor Luc Jaulin who put me in touch with the University of Plymouth and Mr. Wan, and whose algorithms and lessons were extremely useful during this internship.

Abstract

This is a report about the simulation and control of an autonomous sailboat via the Robot Operating System (ROS).

Initially, it was planned that I would come to Plymouth to work directly with the real sailboat and its sensors, in order to develop control algorithms in real life, the simulation part having already been done in previous years; but the health crisis and the impossibility to come to Plymouth forced a change of the internship's subject.

The simulation carried out therefore uses the C++ programming language, and is based on mathematical models developed by the teachers both at the ENSTA Bretagne and in the University of Plymouth. This internship's goal are to simulate the sailboat in order to test its control algorithm, and to use this algorithm in three different missions (line following, station keeping and triangle racing).

Résumé

Il s'agit d'un rapport sur la simulation et le contrôle d'un voilier autonome via le système d'exploitation du robot (ROS).

Au départ, il était prévu que je vienne à Plymouth pour travailler directement avec le voilier réel et ses capteurs, afin de développer des algorithmes de contrôle en situation réelle, la partie simulation ayant déjà été faite les années précédentes ; mais la crise sanitaire et l'impossibilité de venir à Plymouth ont obligé à changer le sujet du stage.

La simulation réalisée utilise donc le langage de programmation C++, et s'appuie sur des modèles mathématiques développés par les enseignants de l'ENSTA Bretagne ainsi que de l'Université de Plymouth. L'objectif de ce stage est de simuler le voilier afin de tester son algorithme de contrôle, et d'utiliser cet algorithme dans trois missions différentes (suivi de ligne, maintien de position et course en triangle).

Contents

Introduction

Unmanned surface vehicles are ships operating at the surface of a water body with no crew. The ship studied in this internship was an autonomous sailboat: its main characteristic is that it possess no engine and therefore will require to be controlled exclusively by acting on its sail and its rudder.

The advantages of USVs are multiple: it allows for much smaller vessels, which consume fewer resources and are much more agile. These ships require either to be remote controlled or to possess their own control algorithms to be operated given the lack of a crew.

It is the latter part that is the most interesting and offers the most perspectives and potential applications, since it allows for the automation of tasks that would require direct human supervision otherwise. Among these tasks, seafloor mapping or routine inspection of maritime equipment following a predefined route comes to mind, and the benefit of automation is very clear in these cases.

Part 1: Modeling

1.1 Description of the system

The system is an autonomous sailboat, with no means of propulsion or locomotion at all. It is composed of three distinct parts, each of which will be represented differently in the simulation: a hull, a sail and a rudder.

-The hull's purpose is to bring buoyancy to the whole system to make it float. It contains the electronical equipment, the captors, and both the sail and the rudder are fixed on it.
The boat position, position, orientation, speed and rotation are all based on this part of the boat and set in its center of mass.

-The sail's purpose is to use the wind's force in order to control and move the boat. It is controlled by command, and can rotate around the z-axis. Its rotation is defined by the value *delta_s*, and its maximum rotation by *delta_s_max*.

-Finally, the rudder acts on the flow of water under the ship, and its angle generates a force on the ship strong enough to influence its trajectory. Like the sail, it can rotate around the z-axis and its rotation values are defined by *delta_r* and *delta_r_max*.

In order for the sailboat to move and control its trajectory, it must use both its sail and its rudder to take advantage of the wind, since it has no mean of self-propulsion.

To establish a law of control, the sailboat also has at its disposal several sensors:

-a heading sensor, providing the sailboat's cap *x[2]*

-a speed sensor providing the speed *x[3]*

-a sensor measuring the force *alpha_tw* and angle *psi_tw* of the wind.

-an accelerometer which provides the rotation speed *x[4]*

-a GPS giving us the boat's location through NMEA frames.

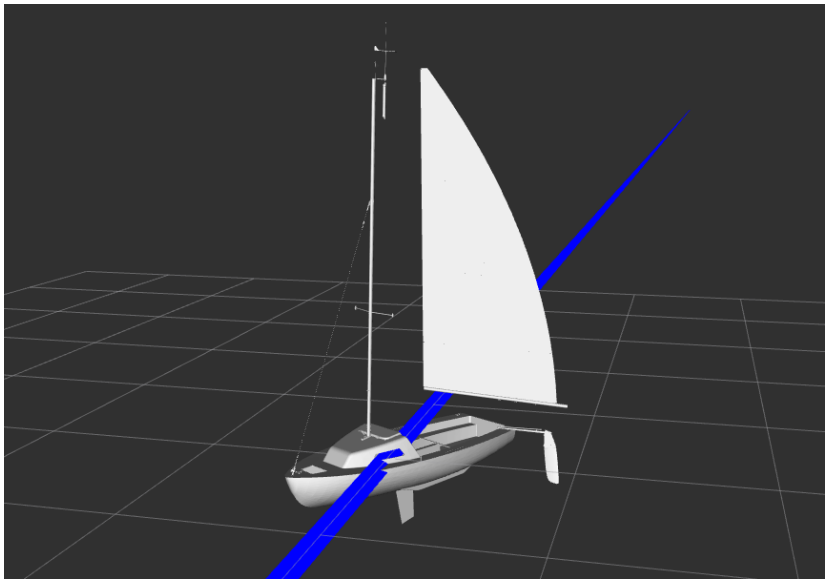

Figure 1: 3D model of the sailboat in RVIZ [1]

1.2 Initial assumptions for the simulation

The purpose of this simulation is to allow us to develop and test algorithms to control the sailboat. For this reason, it is necessary to make several hypotheses in order to simplify the system, given the complexity of all the forces involved. Given the scope and the aim of this simulation, it will assumed that:

- This sailboat will move on a two-dimensional plane on the x and y axes, and will only rotate around the z-axis. The roll or pitch caused by the action of the wind or the wave are deliberately neglected in this scenario.

- With the exception of GPS, all sensors will be assumed to provide us with their value in real time, without delay and without error.

- The water plane is considered as perfectly stable and devoid of any wave or water current.

-The wind's direction and strength will be considered as constant for the duration of the simulation.

-The velocity is small enough to ignore the Coriolis forces

1.3 State equations

There are in total five states $X = [x\ y\ \theta\ v\ \omega]$, with x and y being the sailboat's position on the x and y axes, θ being the cap, v is the ship's velocity and ω its rotation speed. This state equation is defined below:

$$\dot{x} = v\cos(\theta) + p_1 a_{tw}\cos(\psi_{tw})$$
$$\dot{y} = v\,sin(\theta) + p_1 a_{tw} sin(\psi_{tw})$$
$$\dot{\theta} = \omega$$
$$\dot{v} = g_s \sin(\delta_s) - g_r p_{11}\sin(\delta_r) - p_2 v^2)/p_9$$
$$\dot{\omega} = (g_s(p_6 - p_7\cos(\delta_s)) - g_r p_8\cos(\delta_r) - p_3\omega v)/p_{10}$$

With p_1 the drift coefficient, p_{11} the rudder break coefficient [-00], p_2 the tangential friction [kgs^{-1}], p_9 the mass of the sailboat [kg], p_6 the distance to the sail's center of effort [m], p_7 the distance to the mast [m], p_8 the distance to the rudder [m], p_3 the angular friction [kgm] and p_{10} the moment of inertia [kgm^2].

	parameter	value		parameter
p_1	drift coefficient	0.03	$p_5[kgs^{-1}]$	rudder lift
$p_2[kgs^{-1}]$	tangential friction	40	p_6[m]	distance to sail
p_3[kgm]	angular friction	6000	p_7[m]	distance to mast
$p_4[kgs^{-1}]$	sail lift	200	p_8[m]	distance to rudder

value		parameter	value
1500	p_9[kg]	mass of boat	300
0.5	$p_{10}[kgm^2]$	moment of inertia	400
0.5	$p_{11}-$	rudder break coefficient	0.2
2			

Figure 2: Table of the model parameters and their values [2]

g_s is the sail force: its value is based on the difference between the sail and the apparent wind's angles and on the wind's force. The coefficient p4 is the sail's lift [kgs-1]

$$g_s = -p_4 a_{aw} sin(\delta_s - \psi_{aw})$$

On the other hand, g_r is the rudder force, and is solely based on a resistance against the sailboat's movement (given that there is no water current as per the assumptions).

$$g_r = -p_5 v^2 \sin(\delta_r)$$

With p_5 the rudder's lift [kgs-1]

a_{aw} and ψ_{aw} are here the velocity and the cap of the apparent wind, which is the wind measured from the sailboat in movement.

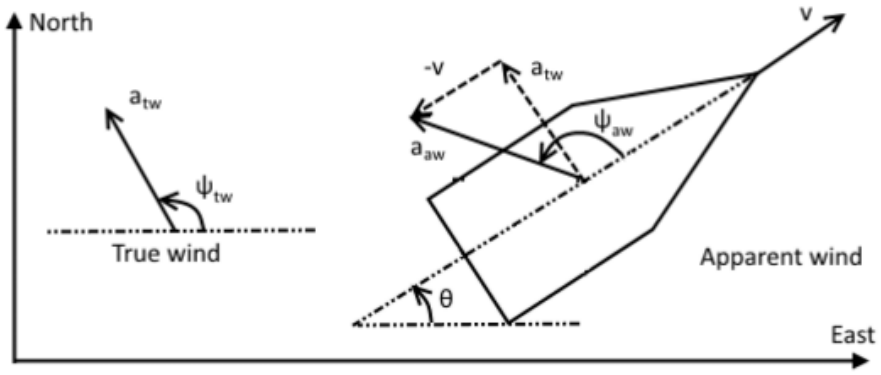

Figure 3: Difference between the true wind and the apparent wind [2]

$$W_{c,aw} = \begin{bmatrix} a_{tw}cos(\psi_{tw} - \theta_i) - v \\ a_{tw}sin(\psi_{tw} - \theta_i) \end{bmatrix}$$

From these cartesian coordinates, the polar coordinates can be obtained and then used to compute both g_s and g_r.

$$W_{c,aw} = \begin{bmatrix} a_{aw} \\ \psi_{aw} \end{bmatrix} = \begin{bmatrix} |W_{c,aw}| \\ atan2(W_{c,aw}) \end{bmatrix}$$

All these equations are enough to simulate the boat on the main node, by simply using the Euler method with a small step to compute the boat's new position and orientation.

Part 2: ROS simulation

2.1 ROS architecture

In order to simulate the sailboat in a ROS environment, I organised the simulation around a main central node **boat_simu_node**. This node will include the five states, which will be updated by the main ROS loop using the state equations and the Euler method.

In order to display the sailboat in RVIZ, the main node will have several publishers who will send to RVIZ the different marker: one marker for each part of the boat (hull, sail, rudder), and some mission specific markers (line marker, wind marker, etc…). Furthermore, to keep track of the previous positions of the sailboat, a set of sphere markers will keep in memory the position of the boat every 0.1 seconds (this time is adjusted according to the loop_rate).

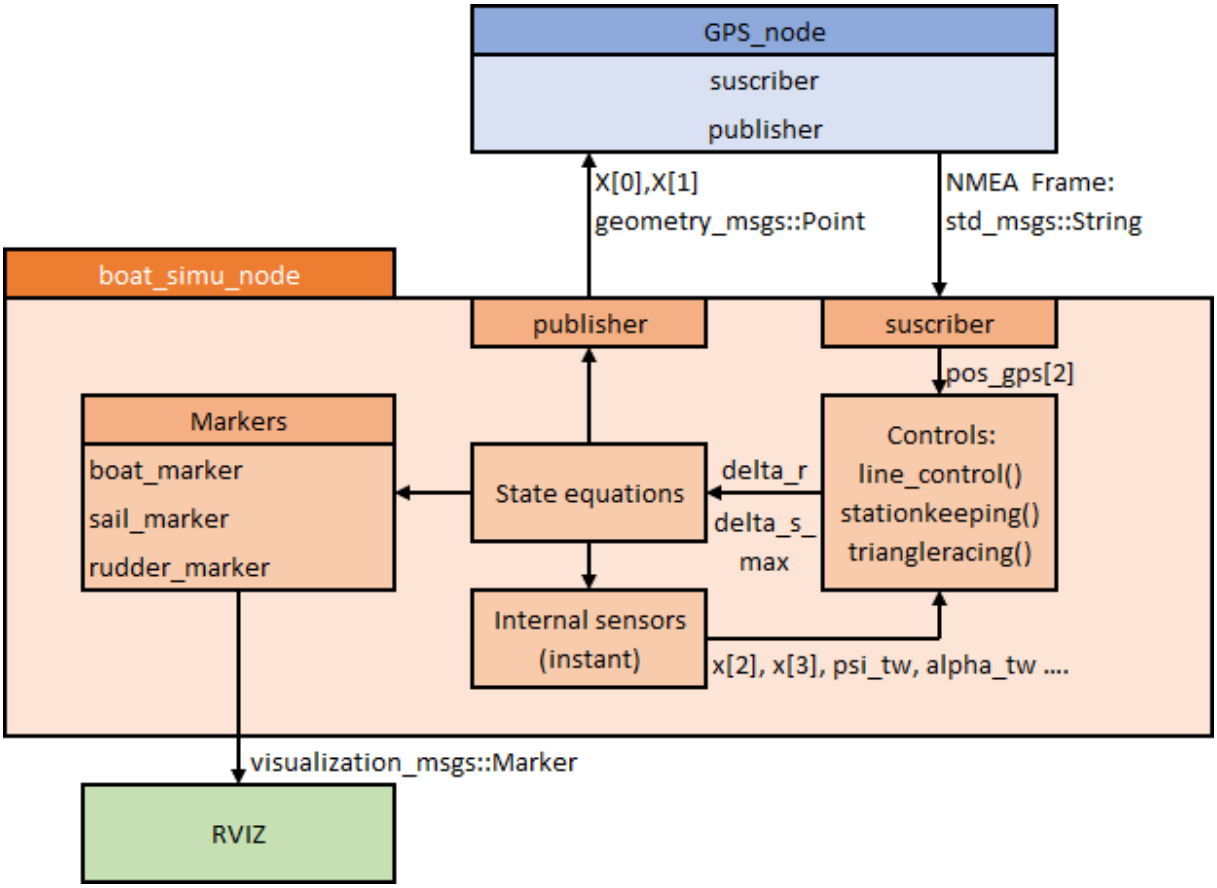

Figure 4: A simplified graph of the ROS nodes

Because of the assumption that all sensors provide us with their value in real time with no delay or error, these values can be directly used in the main_node, and creating a "echo" node that would merely subscribe to their value and then republish them would have little to no interest. For the same reason, the different controllers will be also part of the main node, rather than in a separated node.

These sensors could have been easily part of their own node, and would have in all cases simply required a double subscriber/publisher: one publisher on the main node sending the data, one receiver in the sensor receiving it, then one publisher to send the data after modifications / interferences, and finally one receiver on the main node. This was not done both because of the above assumption, and for memory saving purposes.

In the reality, even with the sensors in their own node they would have been connected to the main node in a one way direction: the main node would have several subscriber nodes, to get all the sensors data, some functions to compute a command following the instructions sent to it and a single publisher node transmitting these commands to the sail and the rudder.

2.2 The GPS node

In the simulation, the GPS node serves three different purposes:

-Simulating the delay between the movement of the sailboat and the data acquisition from the GPS

-Simulating the errors linked to the GPS accuracy, which are in a range of one meter around the real location for a cheap working GPS. This is done by generating two random numbers each time the GPS is called and adding them to the x and y values before conversion into latitude/longitude.

-Finally, representing the errors due to the transmission of the information through NMEA frames. In these frames, only the GPGGA sentences are relevant for this simulation.

```
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Where:
     GGA          Global Positioning System Fix Data
     123519       Fix taken at 12:35:19 UTC
     4807.038,N   Latitude 48 deg 07.038' N
     01131.000,E  Longitude 11 deg 31.000' E
     1            Fix quality: 0 = invalid
                               1 = GPS fix (SPS)
                               2 = DGPS fix
                               3 = PPS fix
                               4 = Real Time Kinematic
                               5 = Float RTK
                               6 = estimated (dead reckoning) (2.3 feature)
                               7 = Manual input mode
                               8 = Simulation mode
     08           Number of satellites being tracked
     0.9          Horizontal dilution of position
     545.4,M      Altitude, Meters, above mean sea level
     46.9,M       Height of geoid (mean sea level) above WGS84
                      ellipsoid
     (empty field) time in seconds since last DGPS update
     (empty field) DGPS station ID number
     *47          the checksum data, always begins with *
```

Figure 5: an example with explanations of a GPGGA sentence [3]

This sentence give is transmitted through a string message in ROS, and convey the latitude and longitude of the boat, which is calculated from both the boat real location (transmitted through a Point message from the main node) and a reference GPS coordinate.

$$x = p * cos(ly) * (lx - lxm);$$
$$y = p * (ly - lym);$$

With x and y the sailboat positions, lx and lx_m the measured and reference latitudes, ly and ly_m the measured and reference longitudes, and p the Earth radius. These equations are done in the other way to compute the latitude and longitude of the boat to transmit in the NMEA frame.

$$ly = \left(\frac{y}{p} + ly_m\right);$$

$$lx = \left(\frac{x}{p * \cos(ly)} + lx_m\right);$$

```
[ INFO] [1602432819.134950230]: $GPGGA,161339,413.950246,N,5037.620354,E
[ INFO] [1602432819.134983632]: $GPGGA,161339,413.950260,N,5037.620407,E
[ INFO] [1602432819.135038080]: $GPGGA,161339,413.950282,N,5037.620390,E
[ INFO] [1602432819.135078188]: $GPGGA,161339,413.950244,N,5037.620362,E
[ INFO] [1602432819.135118247]: $GPGGA,161339,413.950285,N,5037.620408,E
[ INFO] [1602432819.135161541]: $GPGGA,161339,413.950228,N,5037.620390,E
[ INFO] [1602432819.135197889]: $GPGGA,161339,413.950252,N,5037.620381,E
```

Figure 6: NMEA frames being transmitted as string messages from the GPS node to the main node in ROS

In the simulation, the sentence NMEA is stopped in its middle and transmitted directly after reaching the longitude, given that the information following it is both unobtainable by the simulation and of little interests for the locating purposes.

Here, as a reference the GPS coordinates of the Plymouth University were used. Given that this is a simulation with no real use of any GPS, any reference coordinates can be taken. However, in the case of a real use, using real GPS coordinates as a reference for the x and y coordinates will be necessary for proper computations.

Part 3: Control algorithms

Part 3.1 Line following

The goal of the line following method is to get the sailboat to move along a line defined by two points A and B, using the available data: the estimated position given by the GPS, the cap given by the heading sensor, the rotation speed given by the accelerometer and finally the force and direction of the wind.

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 $\cos\left(\theta - \bar{\theta}\right) \geq 0$ then $\delta_r = \delta_r^{\max}.\sin\left(\theta - \bar{\theta}\right)$
11 else $\delta_r = \delta_r^{\max}.\text{sign}\left(\sin\left(\theta - \bar{\theta}\right)\right)$
12 $\delta_s^{\max} = \frac{\pi}{2}.\left(\frac{\cos\left(\psi-\bar{\theta}\right)+1}{2}\right).$

Figure 7: the line following algorithm [4]

The first step of the algorithm is to compute the distance e between the sailboat and the line. This is an algebraic distance, so its sign is depending on the position of the boat compared to the line.

The tacking variable q is defined according to the absolute distance between the boat and the line, this value will set the trajectory that the sailboat will follow (step2).

At the same time, the algorithm computes φ_{target} (φ), the angle of the line between the sailboat and its target; and $\varphi_{nominal}$ (θ^*), the nominal angle from a line attraction vector field (step 3&4).

Then, it compares the nominal angle to the direction of the wind $\varphi_{tw}(\psi)$ to obtain $\varphi_{actual}(\bar{\theta})$, the actual angle to follow, in order to avoid going directly against the wind (step 5). If the sailboat is going against the wind, it adopt the close hauled mode (step 6).

In this mode, the sailboat keeps going forward in a diagonal, using the tack variable to maintain its cap even after crossing the line. It leaves the close hauled mode only after reaching the cut-off distance from the line. This close hauled mode is what cause the sailboat to "zigzag" to press forward despite a wind in an opposite direction (step 7).

In the case the boat is not going against the wind, the nominal angle is satisfactory and is kept (step 8).

Finally, the rudder angle (step 10&11) and the maximum sail angle (step 12) are both set up so that the sailboat align itself with φ_{actual}.

Part 3.2: Line following results

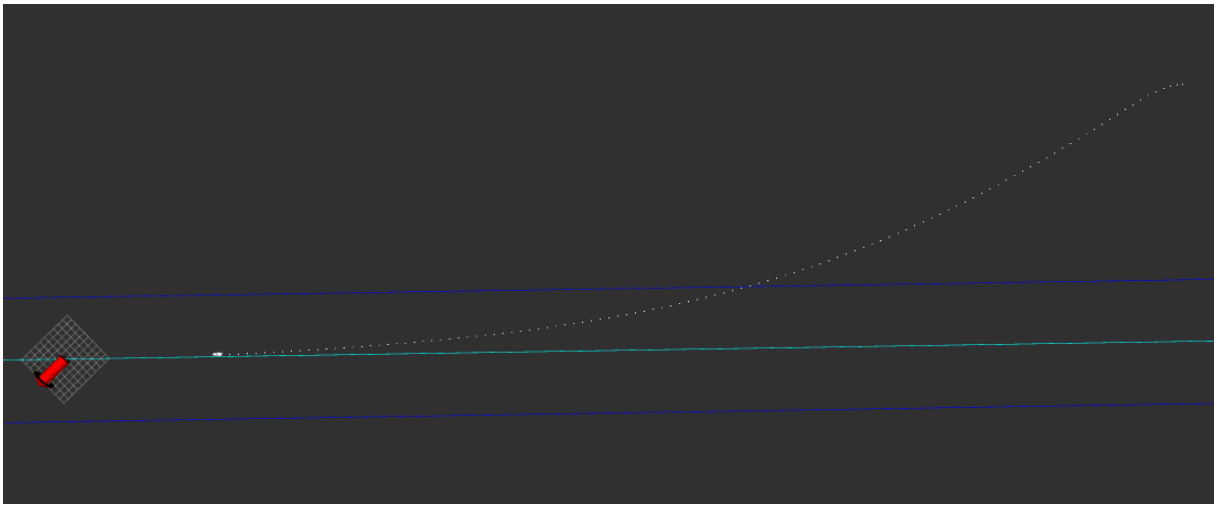

Figure 8: Results of line_follow( ) with case_n = 1

When the line's direction is not opposed to the wind's, there is no need for the boat to use the close hauled mode. As a result, the trajectory is curved, and there is no zigzag. The results obtained with and without the GPS (in the latter case, directly using the theorical values with no computations) were very close, and the difference between the two trajectories did not exceeded 1 meter.

However, since this case do not involve the close hauled mode, it is of little interests, apart from showing that the algorithm works in the absence of an opposing wind.

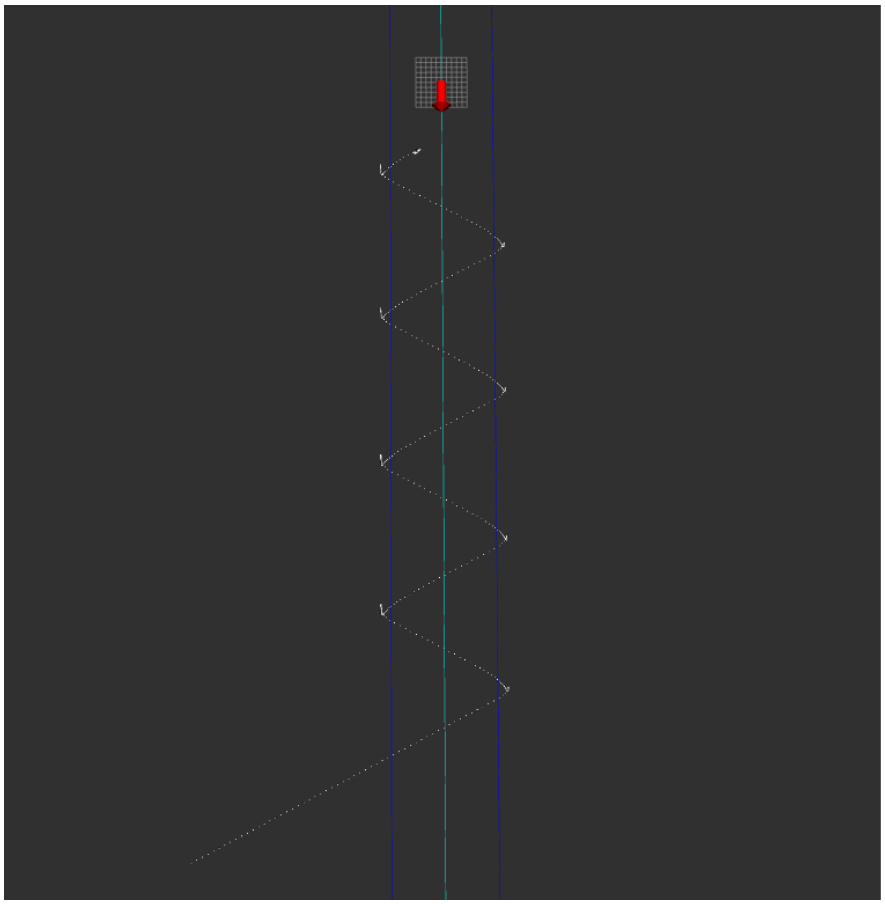

Figure 9: Results of line_follow( ) with case_n = 2

In the case_n = 2, the line following algorithm can be seen in action, and most importantly, it enters the close hauled mode. When the boat reaches the cut-off distance (here shown by the dark blue lines), it tries to keep going in the direction of the line and to follow the nominal cap $\varphi_{nominal}$.

However, because of the opposing wind, such a movement is impossible, it therefore enters the close hauled mode, using instead φ_{actual} as a cap to follow instead. It takes some time for the sailboat to change its cap when it reaches the cut-off distance, and during this time it is stuck on the spot, moving back slightly due to the action of the wind on the sail.

In this special case, because the wind is directly opposed to the line's direction, once the sailboat enters the close hauled mode, it never leaves it. As a result, the sailboat is always moving in a zigzag pattern, avoiding facing the wind.

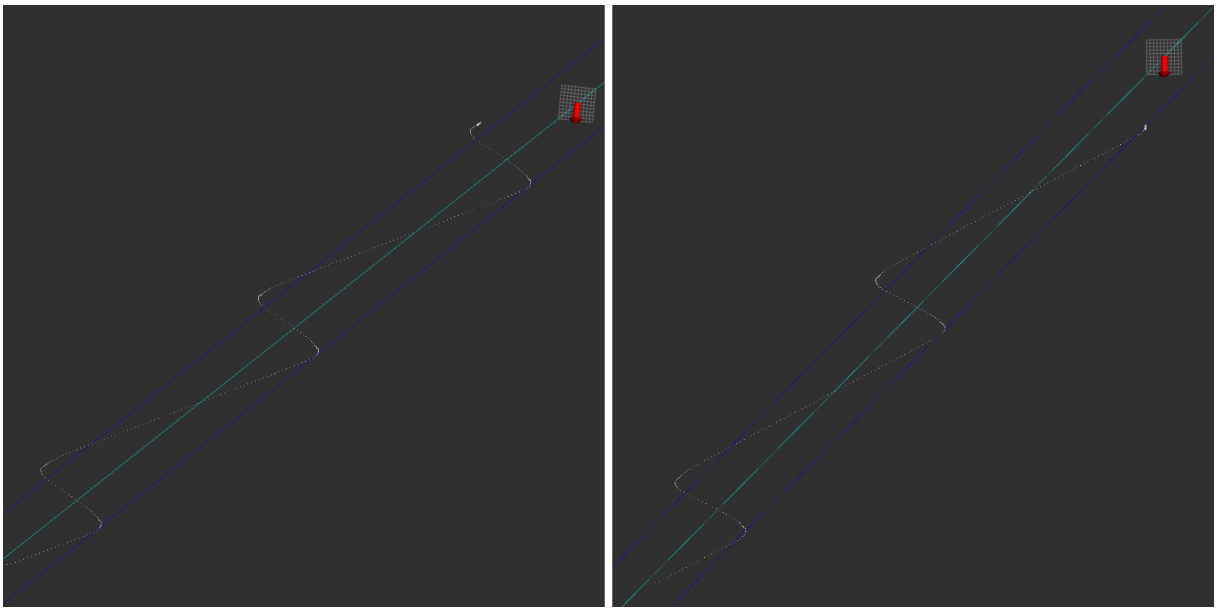

Figure 10: Results of line_follow( ) with case_n = 3, without GPS (left) using theorical data, and with GPS (right)

Finally, the case_n = 3 was tested in order to check if the algorithm behaves in the same way in the ROS environment as it did in Matlab in Mr. Wan's screen recording. The trajectory of the sailboat is identical in both cases; however the ROS simulation is much slower by default and the loop rate needs to be adjusted.

Additionally, since the algorithm behaves in the same way as it is supposed to, it can be used to test the impact of using the data received from the GPS on the trajectory.

While the trajectory remained very similar, when using the GPS the sailboat took a bit more time to detect it reached the distance r to the line, resulting in a slightly longer zigzag pattern. Although these errors were hardly noticeable in the beginning of the simulation, the distance between the two pattern started to slowly builds on, and after 30 repetitions of the zigzag pattern there was more than 20 meters of difference between the theorical and the GPS control.

Despite these differences, the main line following pattern remained fundamentally the same and the algorithm behaved in the same way whether it used the theorical or the GPS data.

Part 3.3: Triangle racing

The triangle racing algorithm is quite easy to understand. The sailboat goes from point to point, directly using the previous line following algorithm, and storing a global variable to keep track of the movement phase.

When one point is reached (when the distance between the sailboat and the point goes under a target value, here the waypoint size), the global variable is increased, signifying the boat must now go towards the next point, still using the line algorithm between the old and the new point. This global variable is cyclic, so that the sailboat is always going from one point to another in order.

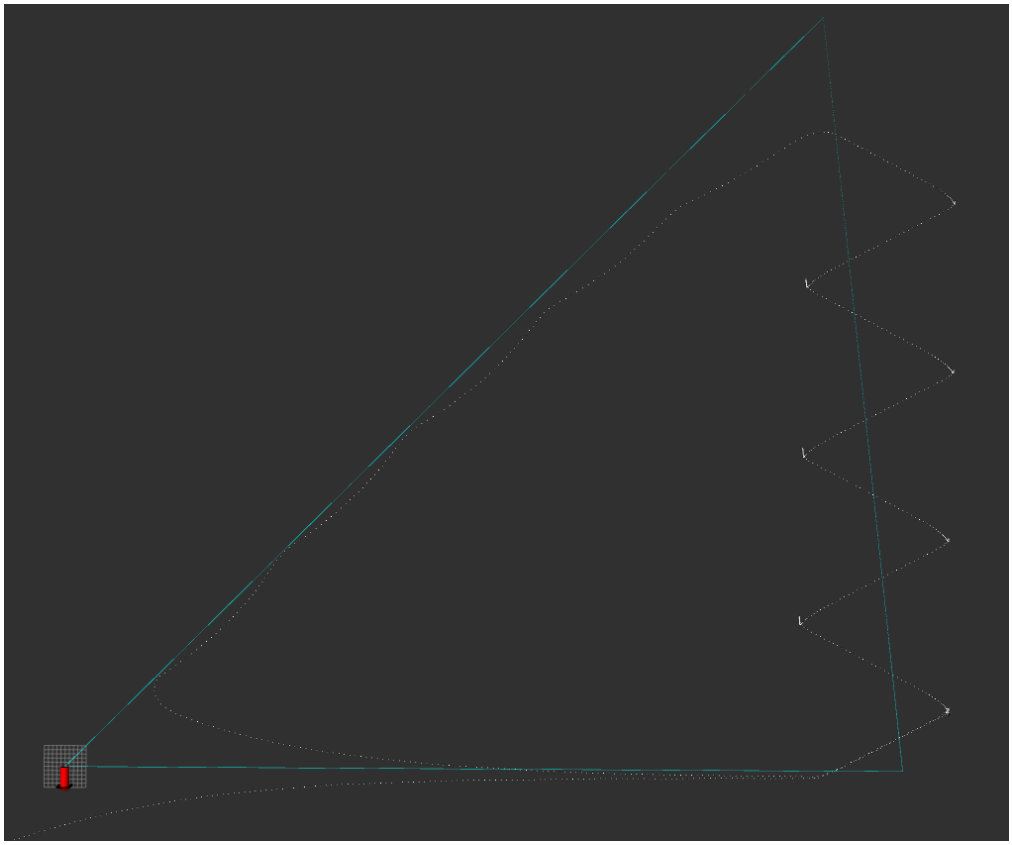

Figure 11: Results of triangleracing ( )

As seen above, the boat reaches its targets without problems, the fidelity of the trajectory when compared to the triangle depends here on two values:

-The waypoint size, which determines how near the sailboat needs to be for the target to be considered as reached

-The cut-off distance, which determines how far the sailboat can move away from the line.

Reducing these two values will result in a trajectory closer to the triangle, at the expense of its speed.

Part 3.4: Station keeping 1: modified line following algorithm

The main goal of the station keeping is to maintain the boat at a given position. Given that the sailboat is above water and has no anchor, the only way to achieve this goal is to keep controlling the boat so that even if its inertia or external action keeps pushing it away it goes back to its position.

The first algorithm tested is a modified version of the line algorithm, and depends on two values, *diaouter* and *diainner*.

-As long as the boat distance to the targeted point is greater than *diaouter,* then it will simply use the line following algorithm to go toward this point.

-Once the *diaouter* distance has been reached, the algorithm compares the targeted angle φ_{target} between the sailboat and the target to the wind's direction. If these two angles are too close, then it will use a perpendicular angle to φ_{target} instead.

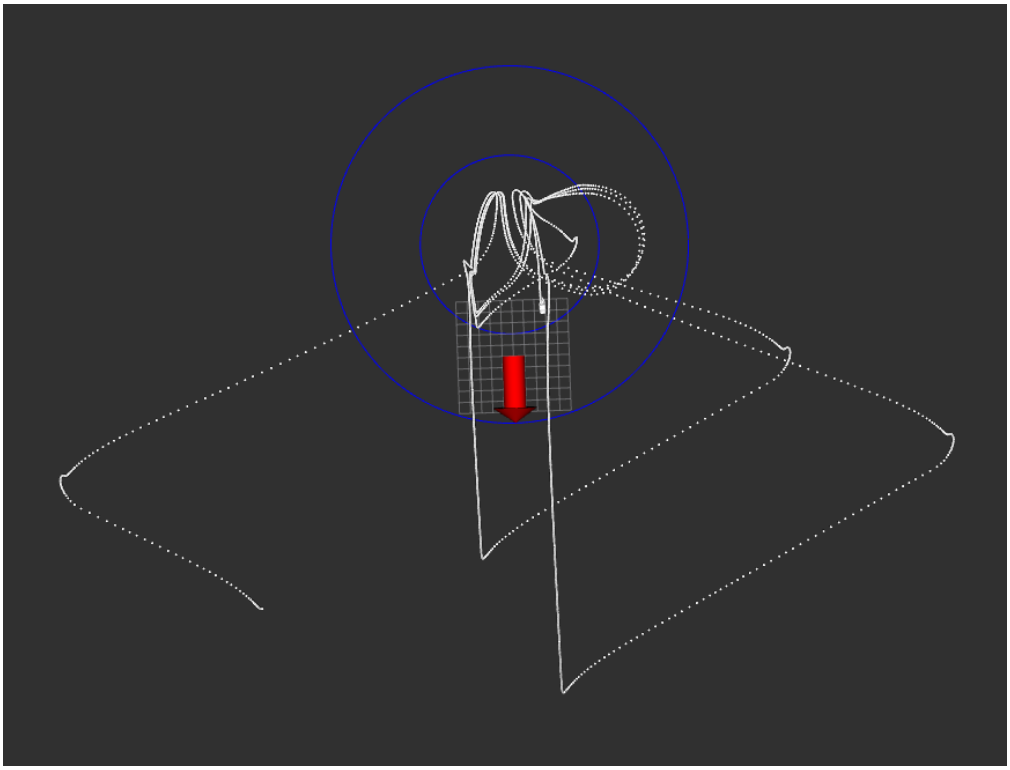

Figure 12: Results of stationkeeping ( )

Regardless of the targeted angle, the algorithm now enter the close hauled mode, and do the same computations as the line following algorithm to obtain φ_actual, the only difference is that the nominal angle here is completely bypassed. From this actual angle, it then computes the rudder and the maximum sail angles.

-Finally, if the sailboat is within the distance *diainner*, the algorithm computes φ_{actual} based on the wind's speed, and from it the maximum sail angle. The rudder's angle is then computed from the comparison between the sailboat's cap and φ_{actual}.

As seen in the Figure 3.6, the algorithm seems to works most of the time, apart when in some cases the boat gets "stuck" in front of the wind. The tack value calculated while the boat is inside the circle leads to the sailboat trying to rotate against the wind, and it needs some time to be able to rotate properly and then go back to the station.

Part 3.5: Station keeping 2: repurposed triangle racing

Rather than making a brand-new algorithm from scratch, one possible way to try to implement a station keeping algorithm is to repurpose the triangle racing algorithm around the targeted point.

Four different configurations were tested:

1st configuration: triangle between diainner and diaouter

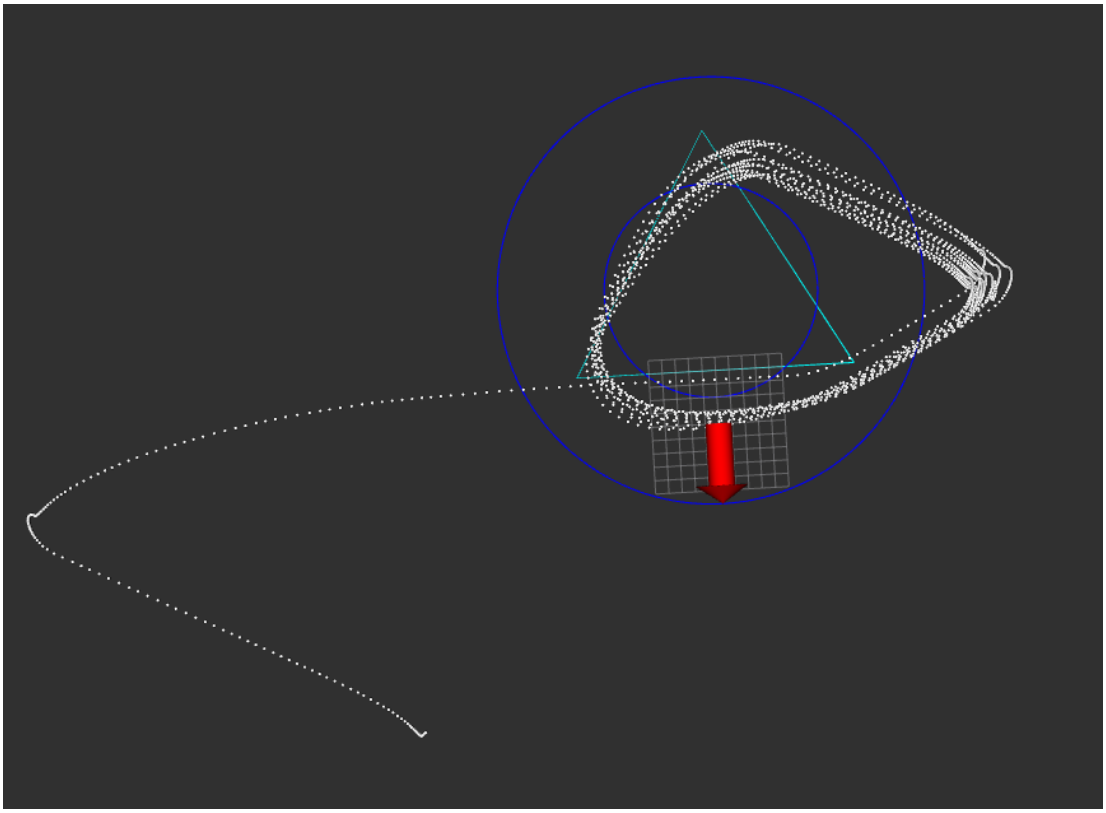

Figure 13: Results of stationkeeping ( ) with case_n = 3

With a large waypoint, the sailboat triangleracing gives a smooth trajectory, and stay mostly between the two circles. However, because it needs to go against the wind, the close hauled mode leads it to get outside the *diaouter* circle.

Nevertheless, by adjusting the cut-off distance and the waypoint size it is possible to get the sailboat to stay within the circle. This can be said for all the algorithms completely reliant on the triangleracing algorithm.

2nd configuration: inverted triangle

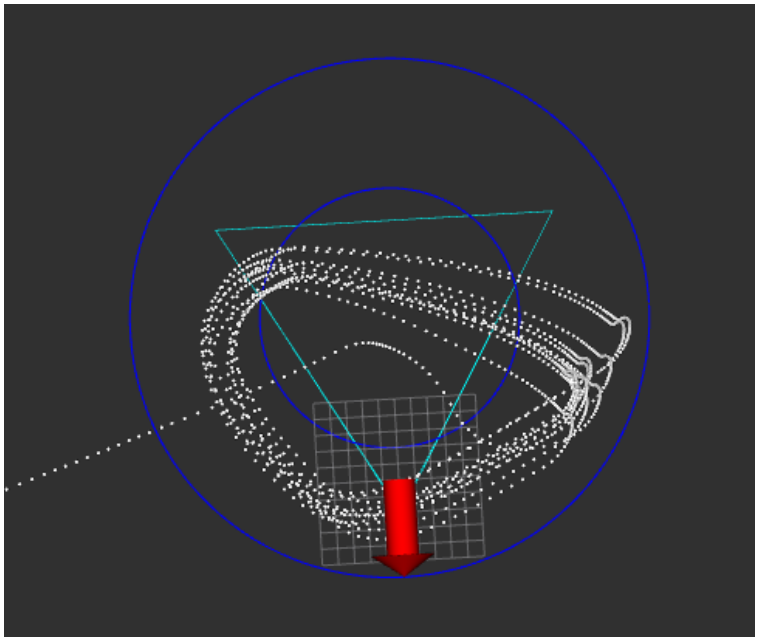

Figure 14: Results of stationkeeping ( ) with case_n = 2

In this configuration the sailboat behaves like in the previous configuration, however the triangle is set in such a way that the close hauled mode do not requires the sailboat to turn before reaching the next waypoint, leading to the ship always staying within the circle.

3rd configuration: line parallel to the wind (with a point in the center)

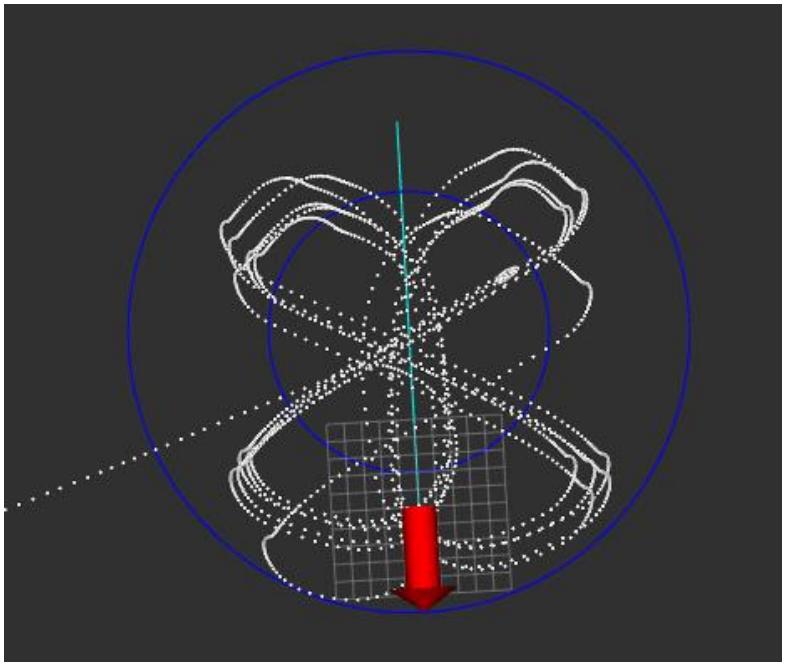

Figure 15: Results of stationkeeping ( ) with case_n = 1

This configuration leads the sailboat to be constantly in close hauled mode. It stays mostly within the outer circle, and in this configuration the sailboat goes the most often above the central point. However, unlike the two previous methods, it requires the target to be in a position accessible to the sailboat, since it do not circle around the target but instead adopt a sort of 88-figure-like trajectory.

4th configuration: line perpendicular to the wind (with a point in the center)

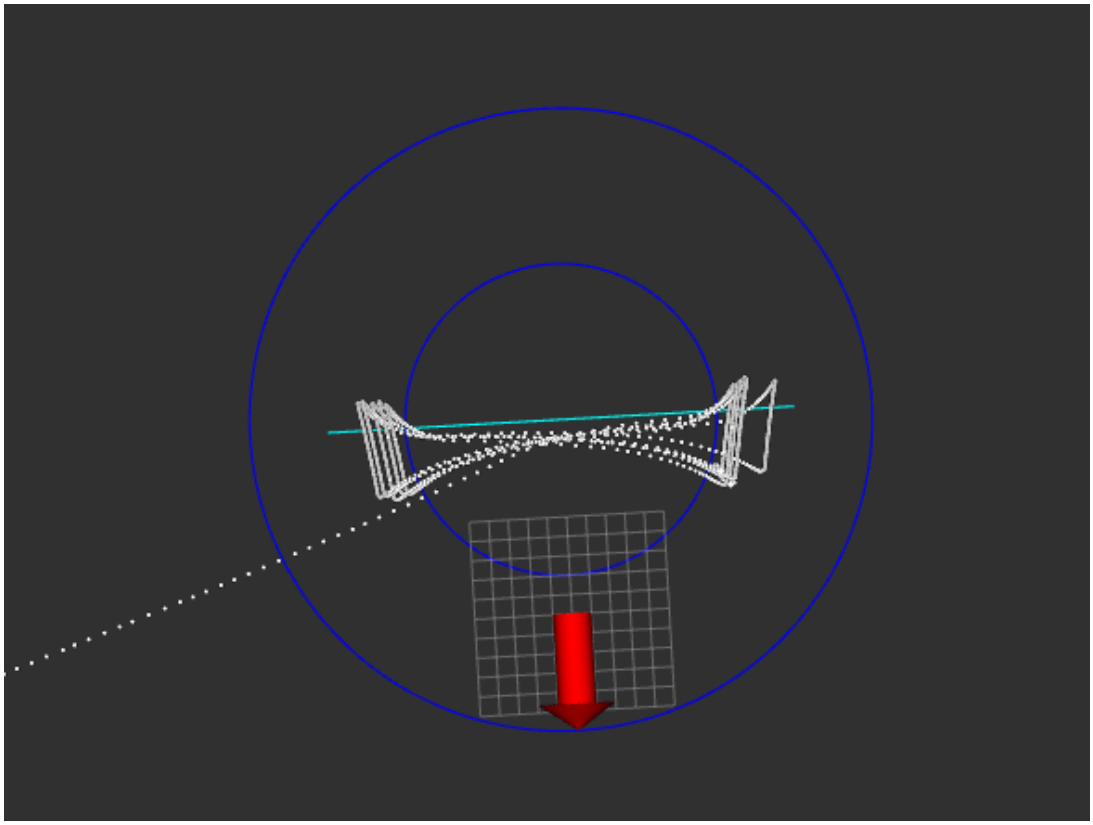

Figure 16: Results of stationkeeping ( ) with case_n = 0

With this configuration, the sailboat has a very tight 8-shaped trajectory which remains very close to the line: this is due to the latter being perpendicular to the wind, allowing the boat to avoid facing the wind most of the time. This method guarantee the sailboat to stay within the outer circle, however the whole line need once again to be accessible by the boat.

Conclusion

The line following algorithm is able to handle different situations without encountering any problem.

The close hauled mode handles the cases where the wind goes against the sailboat well, and so far, no fail states were found in any of the simulations using directly the algorithm with no modification. The precision is flexible, and we can adjust it as needed depending on the simulation's requirements.

This algorithm is a strong base for all the other algorithms such as the station keeping and the triangle racing, and these fared the best when they called directly the line following with no modifications.

The end goal of simulating the three missions in the ROS environment was a success, and while experimenting with the real sensors and the real algorithms would have been more interesting from both a personal and a educational point of view, this internship nevertheless remained instructive and made the best of a difficult situation.

List of figures

References:

[1] A.Courjaud: *Autonomous Sailboat*
https://github.com/AlexandreCourjaud/Stage2APlymouth

[2] C.Viel, U.Vautier, J.Wan, and L.Jaulin: *Platooning Control for Sailboats Using a Tack Strategy*
https://www.ensta-bretagne.fr/jaulin/paper_ijcas_veil.pdf

[3] D. DePriest: *NMEA data* https://www.gpsinformation.org/dale/nmea.htm

[4] L. Jaulin: *A simple controller for line following of sailboats*
https://www.ensta-bretagne.fr/jaulin/paper_jaulin_irsc12.pdf