Alma Mater Studiorum · Università di Bologna

Department of Computer Science and Engineering
Artificial Intelligence
Master Thesis

Sailing Boat Route Finder with Deep Reinforcement Learning

Supervisor:
Prof. Andrea Asperti

Candidate:
Enrico Mannocci

Academic year 2022/2023

Abstract

In this thesis I will implement a system for accomplishing autonomous sailing boat route planning. For the training I will use the public historical wind data from satellites with a 10 km resolution. I will describe a deep reinforcement learning system where states are fake images of wind velocity around the boat, the boat position and orientation and the target location. For evaluating my results I will use the Vendée Globe 2020/21 competition: a sailing boats regatta in which competitors have to accomplish a round of the world trip without any help or breaks. I will use wind data of the race period and I will compare my autonomous agent's track with the GPSes of human competitors'.

Contents

List of Figures

Chapter 1

Introduction

1.1 Sailing Boat

Figure 1.1: A sailing boat underway (1)

Sailing boats (Figure 1.1) are vehicles able to move above water without any engine but wind. A huge amount of designs have been tested and built during human history starting from 3000 a.C. in Taiwan (6) to nowadays. In recent years some incredibly efficient sailing boats where built thanks to computer physical simulation and new hi-tech materials. The most used design in modern days are the yachts: boats with a single hull, two sails, a movable boom and a keel in the lower part underwater to prevent the boat from falling when hit by strong wind (Figure 1.2). Unlike old boats modern yachts are able to move (with different speed) in all directions but a 90° angle centered in the wind coming direction using the appropriate boom and rudder positions. Figure 1.3 shows the speed of boats with different wind directions. As we can see we can reach maximum speed with an angle a little bit bigger than 90° respect to the wind direction, we have a null speed for angle lower than 45° degrees (Up Wind) and low speed with a 180° degrees angle (Down Wind). Finally as expected we have higher boat speed with higher wind speed.

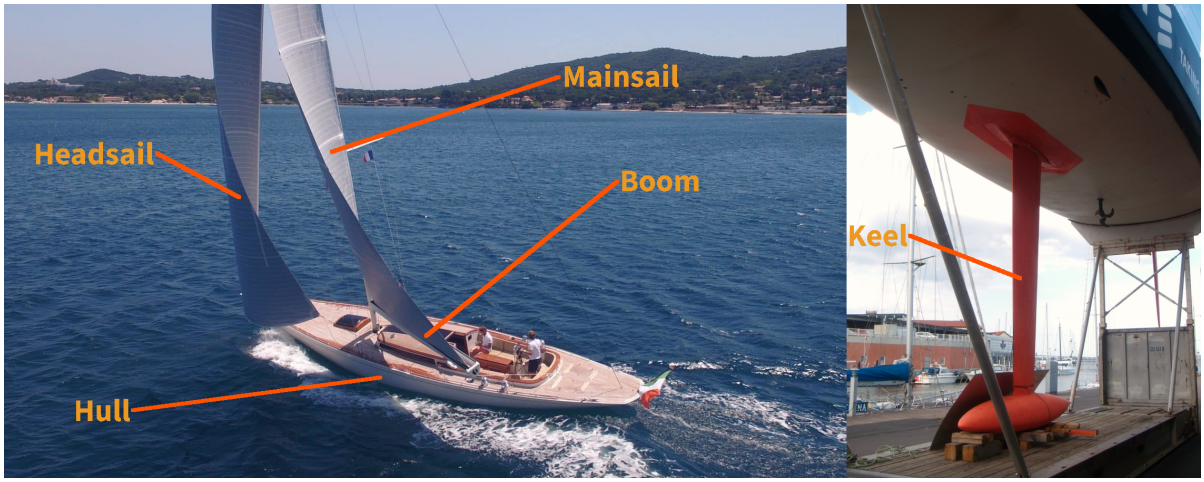


Figure 1.2: The main parts of a sailing boat (2)

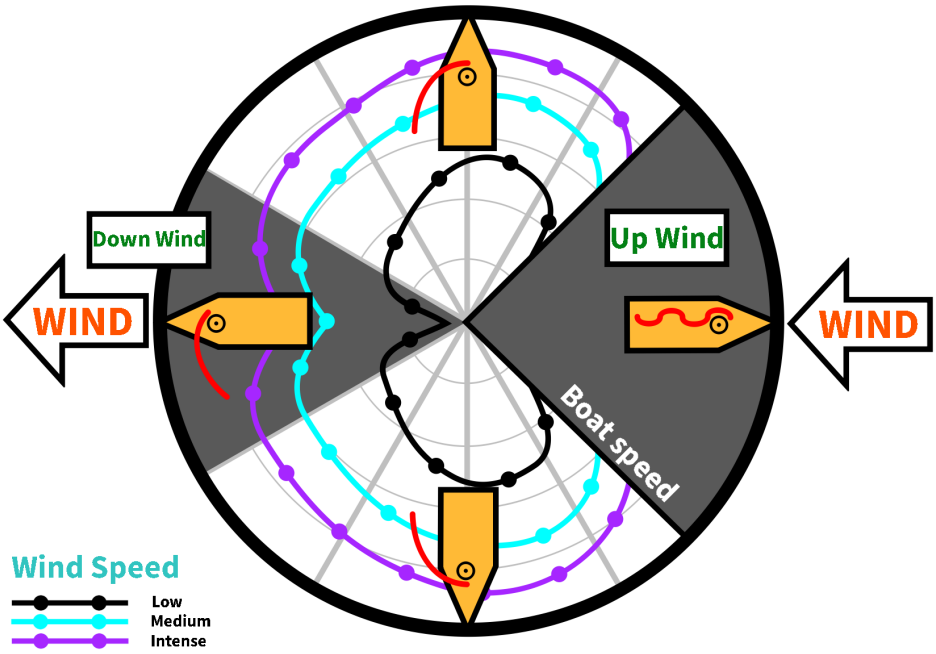


Figure 1.3: The big white arrows show the wind direction, getting further away from the circle center we have bigger boat speeds, and three lines (black, blue and purple) show the boat speed in 3 different conditions of increasing wind intensity. The two areas in gray show direction with null speed (Up Wind) or direction with low speed (Down Wind). The four boats in yellow show how the sail (red) should be taken to go in that direction.

1.2 Reinforcement Learning

Reinforcement Learning (later abbreviated as RL) is one of the three big categories of machine learning algorithms; the other two are supervised and unsupervised learning. RL try to optimize problems where there is a **Decision Maker** that interacts with an **Environment** repeatedly at each time step.

1.2.1 Environment

The **Environment** is a mathematical object called **Markov Decision Process** (later abbreviated as MDP). An MDP is composed by 4 parts (7):

1. A set of states called S
2. A set of actions for each state s called A_s
3. A set of probabilities for each action a of passing from state s to state s' called $P_a(s, s')$
4. A set of Rewards for each action a of transition from state s to s' called $R_a(s, s')$

In Figure 1.4 an example of MDP is showed.

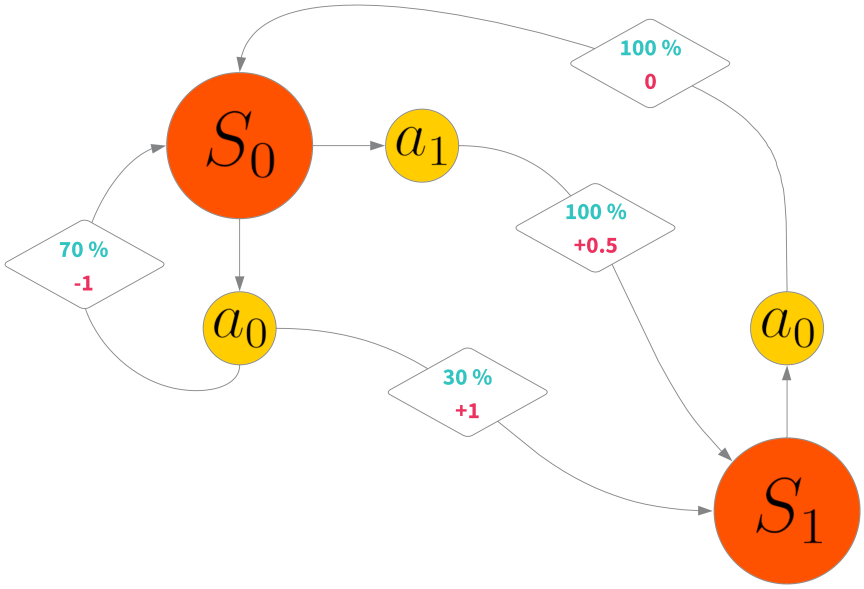


Figure 1.4: A representation of a MDP where we have $S = \{S_0, S_1\}$, $A_{S_0} = \{a_0, a_1\}$, $A_{S_1} = \{a_0\}$, $P_{a_0}(S_0, S_1) = 0.3$, $P_{a_0}(S_0, S_0) = 0.7$, $P_{a_1}(S_0, S_1) = 1$, $P_{a_1}(S_0, S_0) = 0$, $P_{a_0}(S_1, S_0) = 1$, $P_{a_0}(S_1, S_1) = 0$, $R_{a_0}(S_0, S_1) = +1$, $R_{a_0}(S_0, S_0) = -1$, $R_{a_1}(S_0, S_1) = +0.5$ and $R_{a_0}(S_1, S_0) = 0$

1.2.2 Decision Maker

The **Decision Maker** is *located* on a state at each time step, he takes one of the possible actions of that state, he receives a reward from the **Environment** and the possibly new state in which he is located; at the next time step we repeat the same path. The process terminates when the **Decision Maker** enters in a terminal state: a state without any actions.

Let's make an example using the **Environment** (MDP) described in Figure 1.4.

- Assuming that the **Decision Maker** start the process in the state S_0
- Let's suppose that after some calculation the **Decision Maker** decides to take the action a_0
- Now the environment generates a weighted random choice (70% probability to leave him in S_0 and 30% probability to move him to S_1). Let's suppose that the **Decision Maker** is left in S_0 and he receives a reward of -1 as we can see in Figure 1.4.
- Let's suppose that the **Decision Maker** decides to take the action a_1
- The **Environment** brings him to the new state S_1 and gives him a 0.5 reward.
- The process continues indefinitely or till a termination state.

In case the MDP have a termination (a state without any actions), we call episode the subset of time step from the start state to the termination state (episodes can have different time steps length). Instead in case of a MDP without a termination we can either define a maximum number of steps before starting with a new episode or simply have a single infinitely long episode.

1.2.3 Reinforcement Learning Optimization

To optimize a RL system we have to create a **Decision Maker** that analysing the states takes the actions that maximize the **sum** of all the rewards of an episode. As an Example an optimized **Decision Maker** always takes the action a_1 from S_0 and a_0 from S_1 in the MDP described in Figure 1.4.

More precisely speaking we have to find out a policy π that maximizes the return R.

Assuming to have the same actions set in each state, we define (7):

$$\pi : S \times A \to [0;\ 1]$$

$$\pi(s, a) = \text{probability of taking the action } a \text{ being in state } s$$

$$\forall \hat{s} \in S \text{ we have } \sum_{a \in A} \pi(\hat{s}, a) = 1$$

and:

$$R = \sum_{t=0}^{\infty} \gamma^t r_t$$

Where $\gamma \in [0;\ 1[$ (usually 0.99 or 0.999) is called discount factor and it's used to give more importance to temporally closer rewards and r_t is the reward obtained at the time step t. The last sum of course becomes finite if episodes are finite.

1.2.4 Model Base and Model Free

We say that our RL system is Model Base when the **Decision Maker** knows everything about the **Environment** such as the example in Figure 1.4. We instead say that we have a Model Free RL system when the **Decision Maker** just knows the states and the possible actions. The Model Free version of the MDP represented in Figure 1.4 is displayed in Figure 1.5. Of course after some trials (in a simple scenario) it's possible to estimate the unknown information of the **Environment** and to solve the problem as a Model Base one.

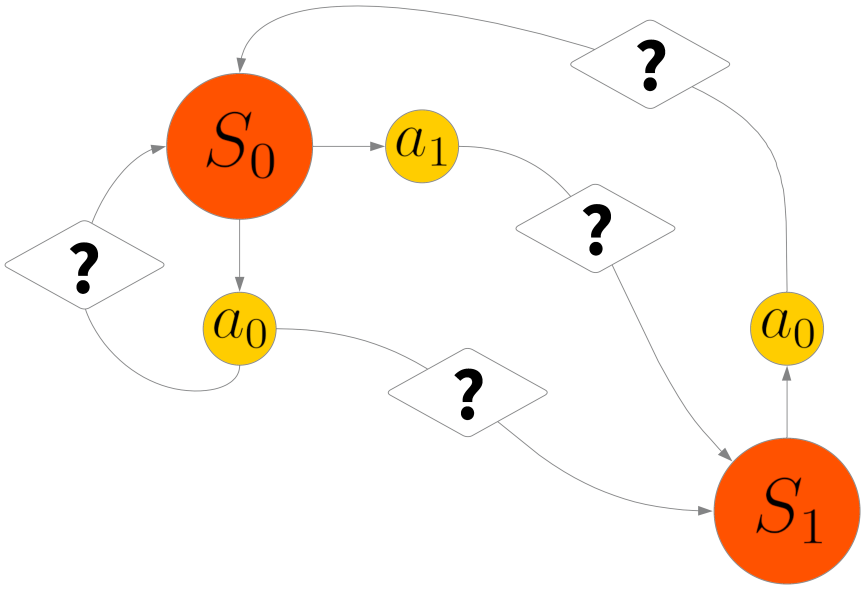


Figure 1.5: A representation of the Model Free version of the MDP represented in Figure 1.4

1.2.5 Q Learning as an Example to solve RL problems

Q Learning is a method that could be used to solve Model Free RL problems. It creates a Q-table (supposing to have the same actions set in each state):

$$Q : S \times A \to \mathbb{R}$$

$$Q(s, a) = \text{The return obtained selecting action } a \text{ from state } s$$

Of course the return of each action in each state is not known and could not be calculated in model free problems so this method approximates its values interacting with the **Environment**. In practice we start assigning to each element of the Q-table the same value and we start simulating episodes. Every time we choose action a in the state s we update $Q(s, a)$ in the following way:

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_{\hat{a}} Q(\hat{s}, \hat{a}) - Q(s, a) \right]$$

where $\hat{s}$ is the state at which the **Decision Maker** is brought by the **Environment** after having chosen action a, the maximum is calculated between all the possible action $\hat{a}$ from the state $\hat{s}$ and α is the learning rate. It should be noticed that the Q-value of all the final states (states without any actions) are never updated and remain at the value assigned at the start of the learning process to all elements of the Q-table.

1.2.6 Deep Reinforcement Learning: Deep Q Network

Q learning is a really simple and power full method but it has a big limitation: it can only be applied when we have a reasonably small amount of states. This can be explained thinking that we have to try all the state-action combinations in order to try to estimate their values. To understand the maximum problem size that this method can handle let's make an example.

Assuming to have 300 iterations per second, 1 hour of computation time, that we need to update each element of the Q-table 10 time for it to be reliable and that each state has 10 actions. We can handle only 10800 states.

$$\frac{\frac{300 \cdot 60 \cdot 60}{10}}{10} = 10800$$

For comparison a 4×4 image where pixels can be either 0 or 1 has 65586 states

$$2^{4 \cdot 4} = 65586$$

and a gray scale 4×4 image where pixels have values between 0 and 255 has $3.4 \cdot 10^{38}$ states.

$$256^{4 \cdot 4} = 3.4 \cdot 10^{38}$$

As we can see this method can only handle small problems, for solving bigger ones we can use neural networks as function approximator: Deep Q Learning.

In practice we substitute the Q-table with a neural network that we train while simulating episodes.

Chapter 2

Simpler Problems

Before trying to solve the whole autonomous sailing boat problem, I have decided to test my RL system on simpler ones. I have started with a really easy environment where the agent can freely move in a small wind-less world, and I have finished with the whole Earth as agent's world and a temporal and spatial varying wind.

I have used this approach for two main reasons:

- to find out errors in the RL implementation
- to get out an appropriate value for each hyperparameter

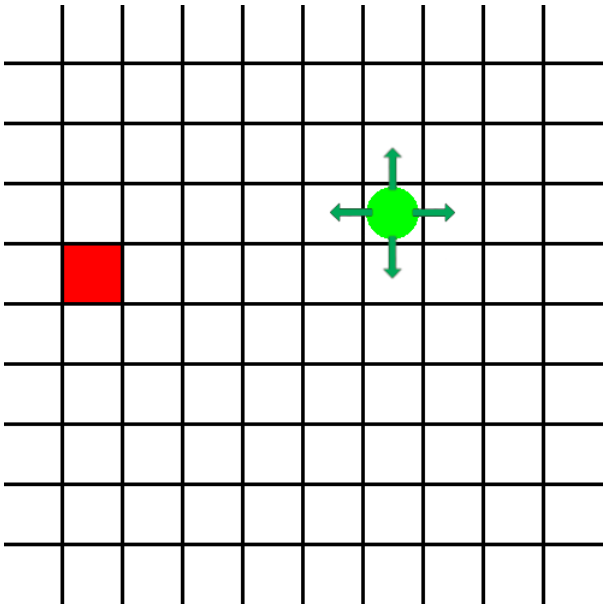
Of course both the above algorithm-checks are easier when the whole training of the network lasts 3 minutes instead of 3 days, but why not just make a really simple environment to check everything and directly pass to the real and highly complex one?

The answer is that I assumed the hyperparameters of similar problems to be similar, and I slightly adjusted them passing from a problem to a sightly more complex one, but this could probably not be done in case of a big complexity gap between them.

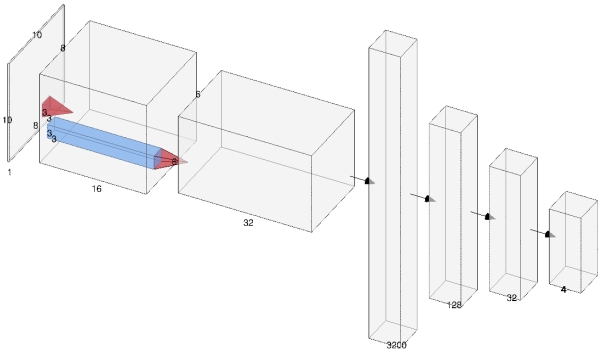
Following this approach slowed me during the implementation (I have implemented a lot of different problems) but saved me a lot of time when I had to find out logic errors. As an example, after finding out a non learning agent, you do not have any information about where the problem in your logic is and as a consequence you have to check again the entire program. With my approach instead, you know all the unchanged parts that were working in the easier problem and focus just on the modified ones.

2.1 10x10 Manhattan Problem

The *world* of this problem is a 10×10 board; the agent position at each time-step and the target position are cells. The agent at each time step can move in 4 directions: North, Est, South, West. The agent's objective is to arrive at the target position.



(a) The green circle represents the agent location, the red square the target location and the green arrows all the possible actions.



(b) Here we have a representation of the network with as input a 1 channel 10×10 image and as output a 4 elements vector (one for each action).

Figure 2.1: Representation of the *10x10 Manhattan Problem* [a] and its Neural Network [b]

2.1.1 States

The states are represented by 10×10 images where we have a 1 in the agent location and a -1 in the target location, all the other cells are 0. As an example, the state represented in Figure 2.1a is:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2.1.2 Actions

The actions are 4 in each state, and they move the agent in the four adjacent cells (North, Est, South, West). If the action moves the agent out of the grid, it instead leaves him in his position.

2.1.3 Rewards

The agent obtains a reward equal to 1 when moving to the target cell and 0 in all other cases.

2.1.4 Training

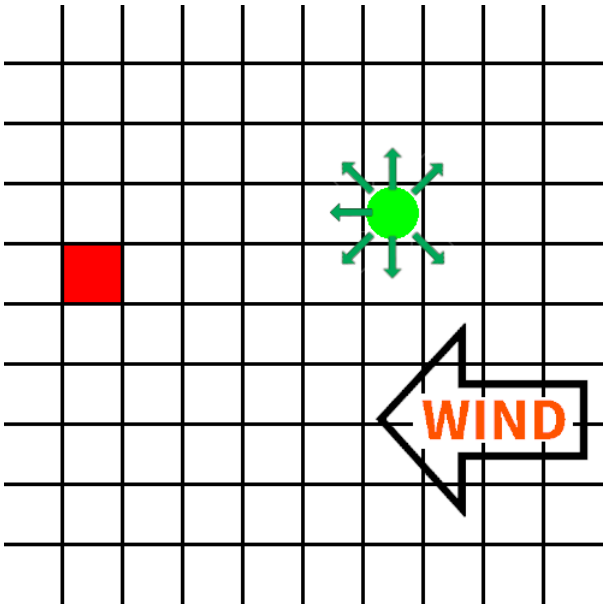
The environment chooses a random agent and target position at the start of episodes. The episodes terminate only when the agent arrives to the target location (they could be really long). During the training process, the probability of random actions is fixed at 0.8.

2.1.5 Take Home Message

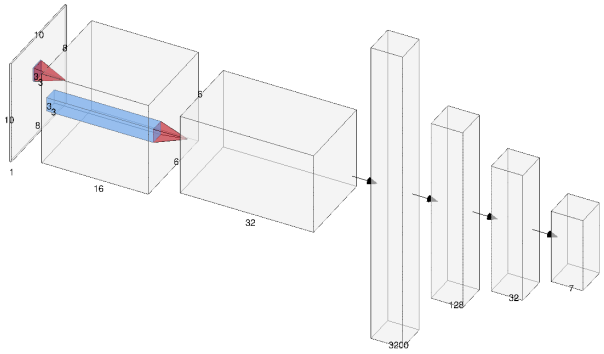
The agent was able to learn and completely solve the problem (it always chooses the best path). This means that the RL system is working and the hyperparameters have an appropriate value.

2.2 Always the Same Wind Problem

This problem is really similar to the previous one, but it adds wind to the world as well as diagonal moves; the wind direction is always the same in each episodes. The agent can move in all directions but the ones from which the wind comes from.



(a) The green circle represents the agent location, the red square the target location and the green arrows all the possible actions.



(b) Here we have a representation of the network with as input a 1 channel 10×10 image and as output a 7 elements vector (one for each action but the direction of wind).

Figure 2.2: Representation of the *Always the Same Wind Problem* [a] and its Neural Network [b]

2.2.1 States

The states are the same as the previous problem.

2.2.2 Actions

The actions are 7 in each state, and they move the agent to adjacent cells except the direction where the wind come from. If the action moves the agent out of the grid, it instead leaves him in his position.

2.2.3 Rewards

The agent obtains a reward equal to 1000 when moving to the target cell and 0 in all other cases.

2.2.4 Training

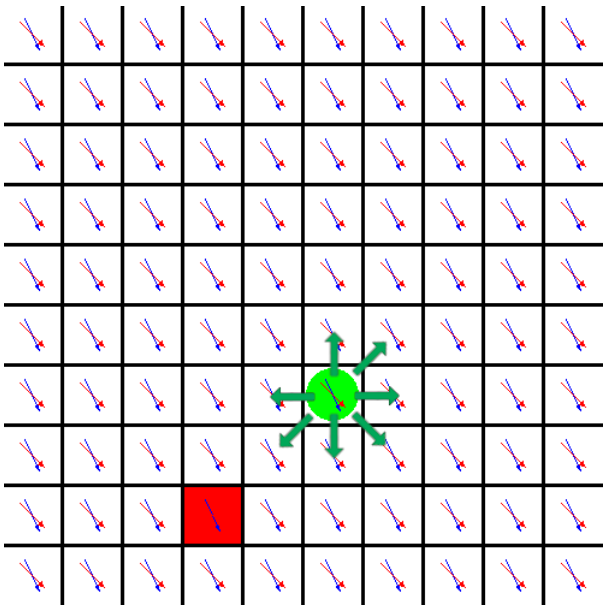
The environment chooses random agent and target positions at the start of episodes; the wind direction is the same in all episodes. The episodes terminate when the agent arrives to the target location (they could be really long). During the training process, the probability of random actions is linearly decreased from 1 to 0.

2.2.5 Take Home Message

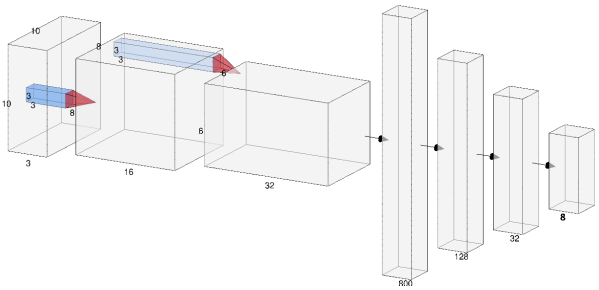
The agent understands which direction is useless (where the wind comes from) and rapidly learns the new paths shape. I noticed how keeping the reward between -1 and 1 increases the network learning speed. This is probably because it has to spend time just learning that it has to increase its output with a specific factor. It seems that linearly decreasing the random actions during training increases the training speed. At the start just random actions are accomplished (general information about the goodness of the choices are taken out) and at the end the agent behavior is fine-tuned (more specific information about the goodness of the agent behavior are taken out).

2.3 Temporally Varying and Spatially Constant Wind Problem

This problem is similar to the previous one but the wind direction changes at each episode: spatially speaking it's constant.



(a) The green circle represents the agent location, the red square the target location, the green arrows all the possible actions and the arrows in each square represent the real wind direction (blue) and the discretized one (red).



(b) Here we have a representation of the network with as input a 3 channel 10×10 image and as output a 8 elements vector (one for each action).

Figure 2.3: Representation of the *Temporally Varying and Spatially Constant Wind Problem*[a] and its Neural Network [b]

2.3.1 States

The states are 3 channels 10×10 images. The first channel has in each cell the distance from the agent location, the second one has in each position the distance from the target location and the third one has the discretized wind position in each cell (all cells always have the same value). An example of the three channels in a smaller 5×5 grid is reported below:

$$\text{Distance from the Agent} = \begin{bmatrix} 0.1414 & 0.2000 & 0.3162 & 0.4472 & 0.5831 \\ 0.0000 & 0.1414 & 0.2828 & 0.4243 & 0.5657 \\ 0.1414 & 0.2000 & 0.3162 & 0.4472 & 0.5831 \\ 0.2828 & 0.3162 & 0.4000 & 0.5099 & 0.6325 \\ 0.4243 & 0.4472 & 0.5099 & 0.6000 & 0.7071 \end{bmatrix}$$

$$\text{Distance from the Target} = \begin{bmatrix} 0.8000 & 0.7071 & 0.6325 & 0.5831 & 0.5657 \\ 0.7071 & 0.6000 & 0.5099 & 0.4472 & 0.4243 \\ 0.6325 & 0.5099 & 0.4000 & 0.3162 & 0.2828 \\ 0.5831 & 0.4472 & 0.3162 & 0.2000 & 0.1414 \\ 0.5657 & 0.4243 & 0.2828 & 0.1414 & 0.0000 \end{bmatrix}$$

$$\text{Discretized Wind Direction} = \begin{bmatrix} \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} \\ \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} \\ \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} \\ \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} \\ \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} \end{bmatrix}$$

2.3.2 Actions

The actions are 8 in each state, and they move the agent in the adjacent cells (North North-Est, Est, South-Est, South, South-West, West, North-West). If the action moves the agent outside from the grid or against the wind, it instead leaves him in his position.

2.3.3 Rewards

The agent obtains a reward equal to 100 when moving to the target cell and minus the distance from the target otherwise; 5 is subtracted if it's trying to go against the wind.

2.3.4 Training

The environment chooses random agent and target positions and wind direction at the start of episodes; the wind direction is the same in all cells. The episode terminates when the agent arrives to the target location or after a maximum number of time steps. During the training process the probability of random actions is linearly decreased from 1 to 0.

2.3.5 Take Home Message

Giving a reward not just at the end of the episode increases the learning speed: the agent starts learning also while trying to reach the first target and not just when it really reaches it. At the start all actions are random and it's hard for the agent to reach the target point; setting up a maximum number of time steps it's able to see different setups without waiting to reach the first target.

Chapter 3

The System

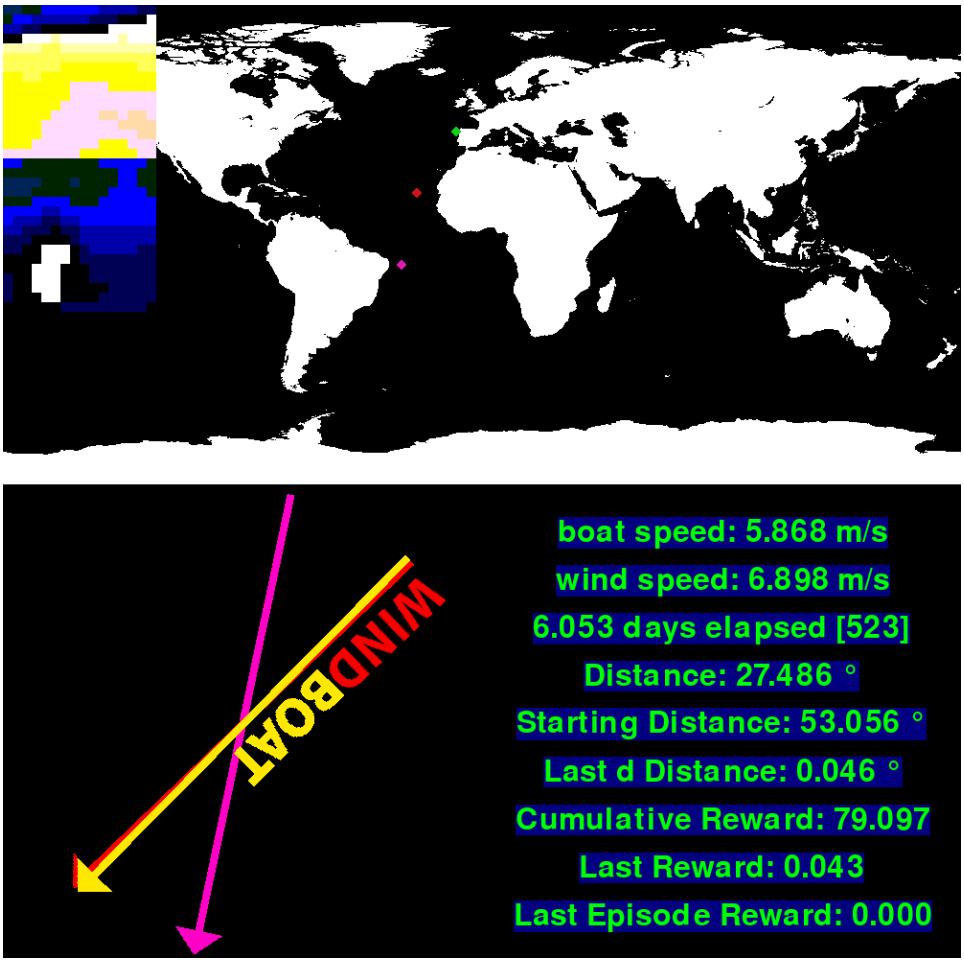


Figure 3.1: The green diamond represents the start location, the red one the agent location, the purple one the target location, the red arrow the wind direction, the yellow arrow the boat direction and the purple arrow the target direction based on the current agent position. On the bottom right we have some parameters such as the boat speed, the wind speed, the time elapsed from the start (and the number of time steps in the square brackets), the current, the starting and the last delta angular distance of the agent from the target location and the cumulative, last step and last episode rewards. On the top left it's possible to see the wind speed along with the latitudinal and longitudinal directions.

3.1 Data

As common in artificial intelligence I have used a lot of real data. More precisely I have used three different sets of data:

- World Sea Surface Wind data 0.25° spatial and 6h temporal resolutions (3)
- Polar Speed data for sailing boats (8) (9)
- GPSes of real competitors in Vendée Globe 2020/21 (5)

3.1.1 Wind Data

Wind data at each time step are composed by two 720×1440 matrices, the first one having for each pixel the wind latitudinal speed (in knots) and the other having the longitudinal one. I have modified the data in two ways: first of all I have transformed the speeds from *knots* to $\frac{m}{s}$ and secondly I have set both the speeds to 0 on lands to prevent the boat from becoming an amphibious. I have downloaded and used, both for training and validation, data from November 2020 to February 2021.

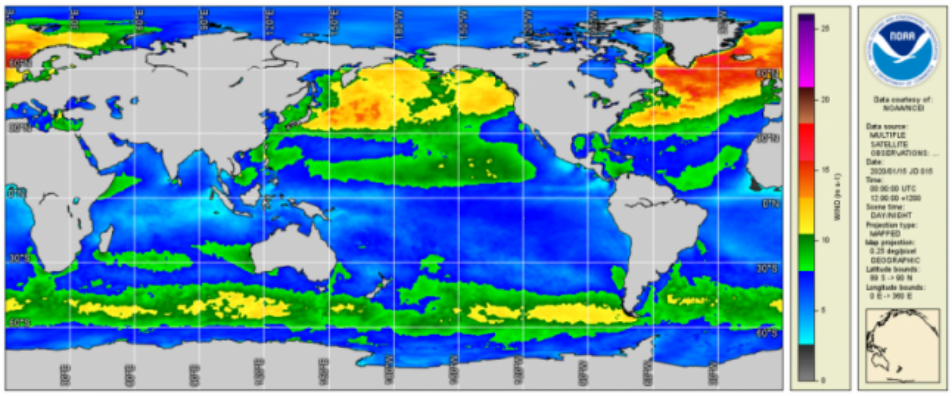


Figure 3.2: Example representation of the world wind data.(3)

3.1.2 Boat Speed Data

In my physical model I have used what is called the Polar Diagram of a boat for having an idea of the boat speed in each wind direction and speed conditions. What a polar is and how it's built was shown in Figure 1.3. I have found out a project (9) that takes out polar diagrams from the ORC (Offshore Racing Congress) data set of a wide range of boats. More precisely I have chosen a boat really similar to the ones that are competing in length, weight and performances: Vesper shown in Figure 3.3.

Figure 3.3: The boat from which I have taken the polar data for simulations during a race in May 2023 near Naples.(4)

3.1.3 Vendée Globe 2022 GPS Data

To validate my results I have used the GPS data from the Vendée Globe competition of 3 competitors. These data were not public, so I have asked for them directly to the race organization that kindly answered sending them to me. The three boats that I have chosen are respectively: the winner Yannick Bestaven (Maître CoQ IV), the 8^{TH} Giancarlo Pedote (Prysmian Group) and the 25^{TH} Ari Huusela (Sark). The competition had started in November 2020 in Lorient (France) and had finished in February 2021 in the same place.

Figure 3.4: The three Vandée Globe participants and their boats that I have chosen for validating my experiments. From left to right Yannick Bestaven, Giancarlo Pedote and Ari Huusela.(5)

3.2 Physical Model

The whole physical model is composed of two parts: the wind and the boat. The first one is simply composed of a couple of matrices that associate to each place on earth a wind velocity vector. Properly updating the time we are able to take care of the temporal changes of wind (I have new data every 6 hours). Just with that the best we can do for estimating the wind vector in a given spatial and temporal location is by using a 3D linear interpolation (latitude, longitude and time); more information about this in A.3.

Let's talk about the boat. Assuming a generic time step in which we know the boat position and orientation; we give the possibility to the driver to change the boat's orientation clockwise, anticlockwise or not to change it at all. After that we calculate the relative angle between the boat and the wind directions, we calculate the wind speed in the boat's position and we get out the boat speed from the polar wind data with a 2D linear interpolation (boat-wind relative angle and wind speed); more information about this in A.2.

In conclusion we simply integrate over a predefined amount of time the boat position with the speed that we have found and we start again the process with a possibly new position; it doesn't change just in case of null wind speed or relative angle lower than 45°. The whole process is repeated at each iteration.

3.2.1 Parameters

The only parameter that we have to choose in the physical model is the integration time interval. It turns out to be a really important and sensitive parameter. With a big integration time the RL agent needs to take much fewer decisions and it can better understand how good or bad its actions are (we have big changes in the positions); in simpler words the training is faster and simpler. But if we look at the other side of it, we will have bigger and more unpredictable changes in wind direction on adjacent time steps and the RL agent will limit itself by analyzing only the current one.

Choosing a small integration time we will end up with a huge number of actions that the agent needs to take. As a consequence each action will be less important for the bigger result and harder to be correctly evaluated; we will have a slower and harder training. Of course if we look at the other side of it, we will have a much smoother change of wind direction and the information about the surrounding conditions will become more important.

3.3 Reinforcement Learning

Let's now explore the Reinforcement Learning part of the system. Like the simpler ones the problem was treated with a deep Q-network solution.

3.3.1 States

States are composed of two parts. The first one is a 2 channels 128×128 fake image with the latitudinal and longitudinal wind speed from a square area of 1000 km around the boat. The second one is instead a vector of 7 elements with values between 0 and 1. More precisely the vector components are: the normalized boat direction angle respect to north, the normalized direction angle of the vector that points to the target point respect to north, the normalized wind direction, the normalized wind speed ($40\frac{m}{s}$ was arbitrarily set as the maximum accepted speed), the normalized boat speed ($30\frac{m}{s}$ was arbitrarily set as the maximum accepted speed), the normalized angular distance to the target (the

starting angular distance was set as the maximum) and the normalized maximum of episode steps (the maximum number of episode steps was previously set).

Final State

There are 3 ways to conclude an episode or equivalently arrive in a final state:

- Having already done the maximum number of steps
- Being at less than a predefined distance from the target location; more precisely the angular distance is used, for more information see Appendix B
- Being at more than double the starting episode distance from the target location; the angular distance is used

3.3.2 Actions

The possible actions are 3: rotate the boat clockwise, anticlockwise and not rotating the boat at all.

Parameters

The only parameter of the actions part is the rotation step. In order to give importance to each action it's good that it's as big as possible but also not too big, in such a way that the boat is able to go near to 45° facing the wind; otherwise it's not able to properly move against it. The chosen rotation step was $\frac{\pi}{8}$.

3.3.3 Rewards

I have tried an high number of reward functions and as I expected they changed a lot the training speed and effectiveness. I will report the last and more effective one.

The sum of all the rewards in an episode is constrained between 0 and 100; more precisely it's composed of two parts that are constrained between 0 and 50 and that are summed to obtain the final reward. The first part is about space; more precisely relative to how close the agent arrived to the target point respect to the starting distance from it. The second one is about time; more precisely relative to how fast the agent obtained the result. Mathematically speaking at each non final step t we have a reward of:

$$r_t = \underbrace{25 \cdot \frac{\text{distance}_{t-1} - \text{distance}_t}{\text{distance}_{t_0}}}_{\text{Spatial Part}} - \underbrace{\frac{50}{\text{maximum number of steps}}}_{\text{Temporal Part}}$$

where distance_t is the distance from the position of the boat at time t and the target point; more precisely I have calculated the angular distance along the circumference passing through both the target point and the boat position on the approximated earth sphere, more information in Appendix B.

Each final state t takes instead a reward of

$$r_t = 50 \quad \text{where } t \text{ is a final state}$$

Some Examples

Let's make some examples to clarify this equation.

Let's start by analyzing an episode in which the boat is not smart at all and it always moves away from the target. There are two possible conclusions to such an episode:

- The maximum number of steps is reached
- The starting distance from the target is doubled

In the first scenario we will have a *Spatial Part* $<<$ 25 and a *Temporal Part* $= 0$, so a final recall:

$$R = \sum_{t=0}^{\text{Maximum Number of Steps}} r_t << 25$$

In the second scenario we will have a *Spatial Part* $= 0$ and a *Temporal Part* $<<$ 50, so a final recall:

$$R = \sum_{t=0}^{\text{Maximum Number of Steps}} r_t << 50$$

Generally speaking we will have in both cases really low recalls.

Let's analyze the scenario in which we have a boat that simply stays at the starting point. The only possibility of conclusion is to reach the maximum number of steps. We would have a *Spatial Part* $= 25$ and a *Temporal Part* $= 0$, so:

$$R = \sum_{t=0}^{\text{Maximum Number of Steps}} r_t = 25$$

We turn out to have a better reward than in the previous example but of course still a poor one.

In conclusion let's analyze the scenarios in which we have a really smart boat that arrives to the target location firstly using $\frac{1}{2}$ of the time steps and secondly $\frac{1}{5}$ of them. In the first case we would have a *Spatial Part* $= 50$ and a *Temporal Part* $= 25$, so:

$$R = \sum_{t=0}^{\text{Maximum Number of Steps}} r_t = 75$$

and in the second scenario we would have a *Spatial Part* $= 50$ and a *Temporal Part* $= 40$, so:

$$R = \sum_{t=0}^{\text{Maximum Number of Steps}} r_t = 90$$

3.3.4 Neural Networks

The Neural Network is composed of two parts: a convolutional one that takes care of the fake images and a fully connected one that merges the output of the first one and the normalized numbers shown in the State description 3.3.1. A representation of both the parts is shown in Figure 3.5 and 3.6.

More precisely the input of the fully connected network is a concatenation between the 36 output elements of the convolutional one and the 7 numbers explained in 3.3.1. Its output is a 3 entries vector that should contain the expected recall of each action: turn clockwise, anticlockwise and do not turn at all.

This network is used as a function approximator, so the **ReLU** activation function was chosen.

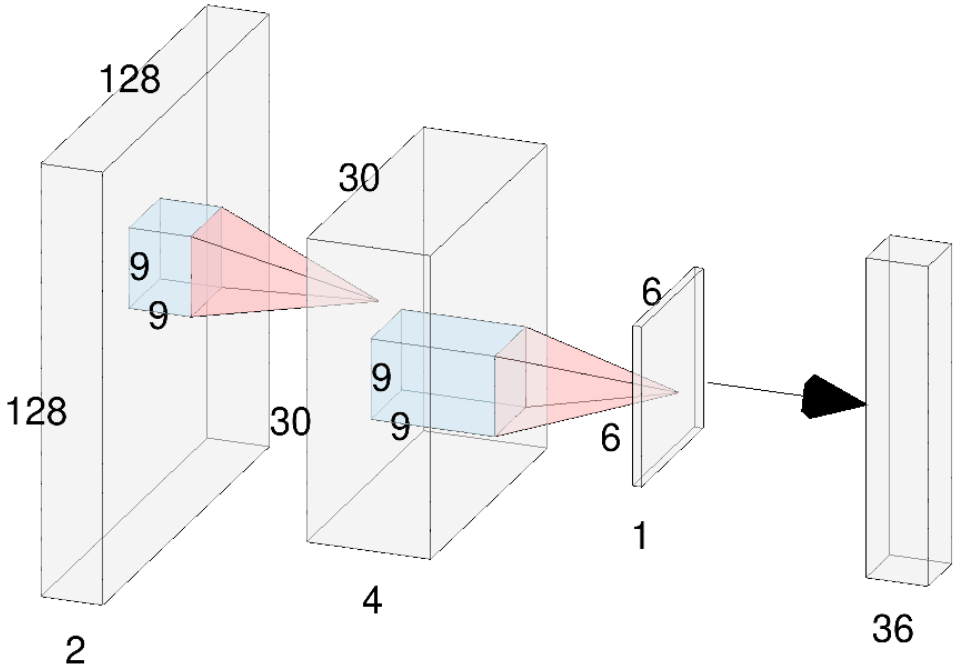


Figure 3.5: Here we have a representation of the small convolutional part of the network that takes as input the 2 channels fake image with the wind speed around the boat and returns a vector of 36 numbers. In each convolution a stride of 4 is used.

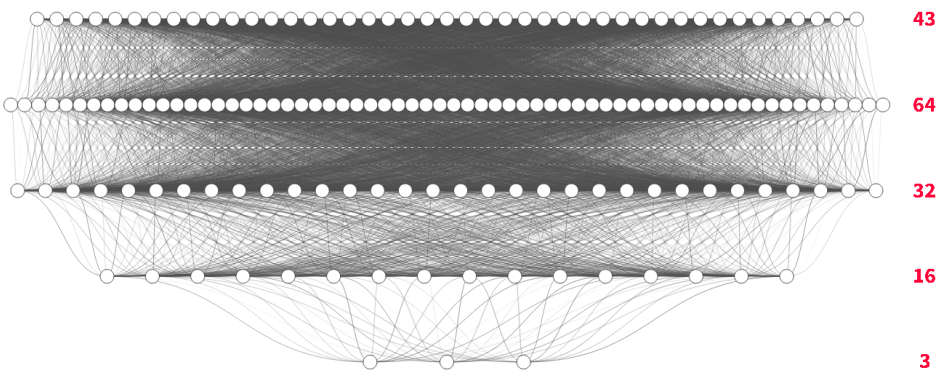


Figure 3.6: Here we have a representation of the Fully Connected part of the network. It takes as input vectors of 43 elements: 36 from the output of the convolutional part and 7 that are the normalized elements shown in 3.3.1. It gives as output 3 numbers that are the expected episode rewards taking respectively the turn clockwise, anticlockwise and do not turn actions.

3.4 Training

The training process of the final model was of course more complex and long than the already reported simpler problems ones; in the following I will show and comment the three more important training sessions.

3.4.1 Training Setup

The training episodes were built with different wind conditions randomly choosing a day amongst the data, and choosing random start and target location amongst the Vandée

Globe checkpoints that are shown in Figure 3.7 (If the i checkpoint has been chosen as starting point the $i+1$ checkpoint is always chosen as target point).

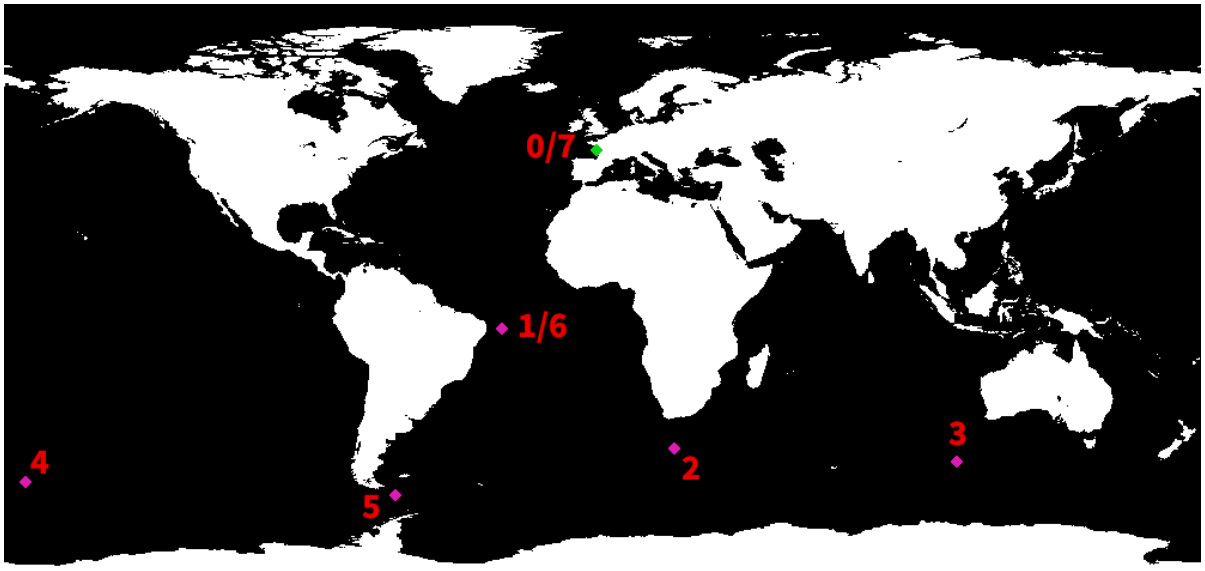
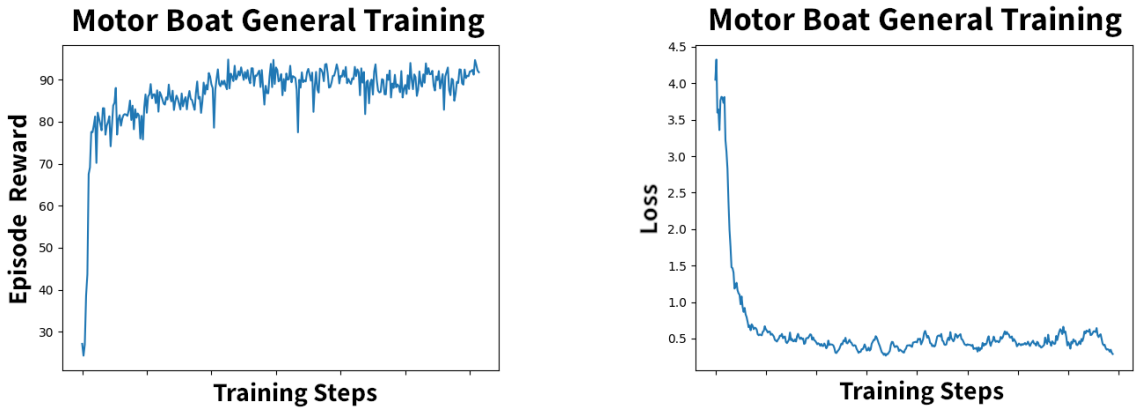


Figure 3.7: All the Vandée Globe Checkpoints numerated. The rules state that you need to pass near to checkpoint 1/6 after having reached both checkpoint 0 and 5.

3.4.2 Motor Boat General Training

After having tried to train the entire model without obtaining any results I have tried to simplify the problem by giving the boat the possibility to go in any direction with the same speed; in practice I have given a motor to my sailing boat. I have linearly decreased the random actions probability from 1 to 0.2 along the training obtaining in a really fast way good results. The mean episode reward and the loss along training is shown in Figure 3.8a and Figure 3.8b.



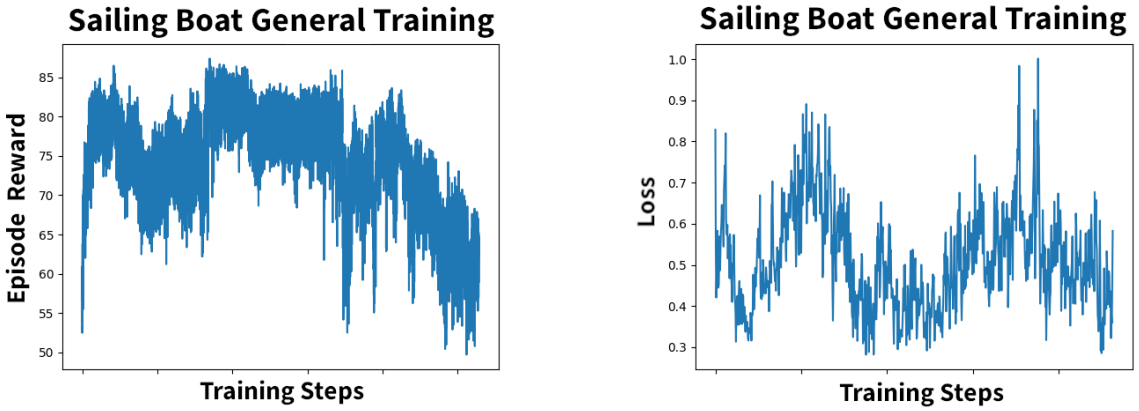
(a) (b)

Figure 3.8: Mean Episode Reward [a] and Loss [b] along the training of the simplified version of the problem

3.4.3 Sailing Boat General Training

Starting with the weight trained as shown in the last section I have come back to the full problem of sailing boats. Also during this training session I have linearly decreased the probability of random actions but between 0.8 and 0.2 this time. The mean episode reward and the loss along training is shown in Figure 3.9a and Figure 3.9b.

In the last part of the training, when there was a low number of random actions, the episode reward clearly decreased highlighting a not completely successful training. Trying the weights with different checkpoints I figured out the problem: the agent was really good when near Antarctica with strong and spatially constant wind but got stuck in Pacific due to really close to the coast passage and highly spatially variable wind. With an high probability of random moves it was able to avoid getting stuck but it was not able to learn a general behaviour to avoid that.

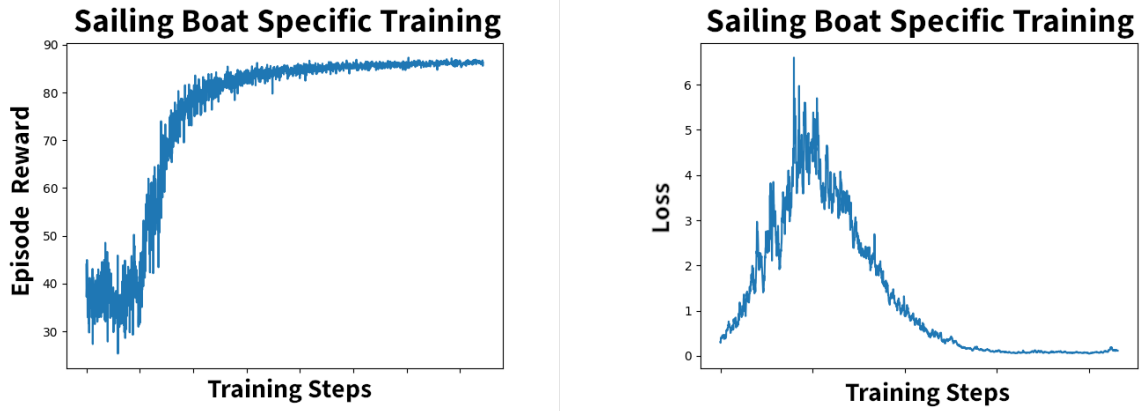


(a) (b)

Figure 3.9: Mean Episode Reward [a] and Loss [b] along the training of the full version of the problem. In the last part of the training, when random actions were more rare the episode reward clearly decreased.

3.4.4 Sailing Boat Specific Training

The solution to the problems shown in the last section was simply to train the model more specifically on the Atlantic ocean starting from the weighs obtained in the previous training. During this training I have kept the probability of random moves lower (linearly decreased from 0.5 to 0) and after some starting steps really good results were obtained. The mean episode reward and the loss along training in the Atlantic Ocean is shown in Figure 3.10a and Figure 3.10b.



(a) (b)

Figure 3.10: Mean Episode Reward [a] and Loss [b] along the training of the full version of the problem specifically for the Atlatic Ocean. After a first part of not learning a good increasing of the episode reward can be seen.

Chapter 4

Results

4.1 Vandée Globe Validation

For validating my results I have compared the real sailor GPSes and my Agent's track in the Vandée Globe competition of 2020/2021.

4.1.1 Competition Rules

The race begins and finishes in the French city of Lorient. The competitors have to pass near 6 checkpoints in a streak order and come back to the start point: Lorient. The competition started on the 11^{th} of November 2020. Of course I used and kept updated the wind data of that period to validate my RL agent decisions. All the checkpoints are shown in Figure 3.7.

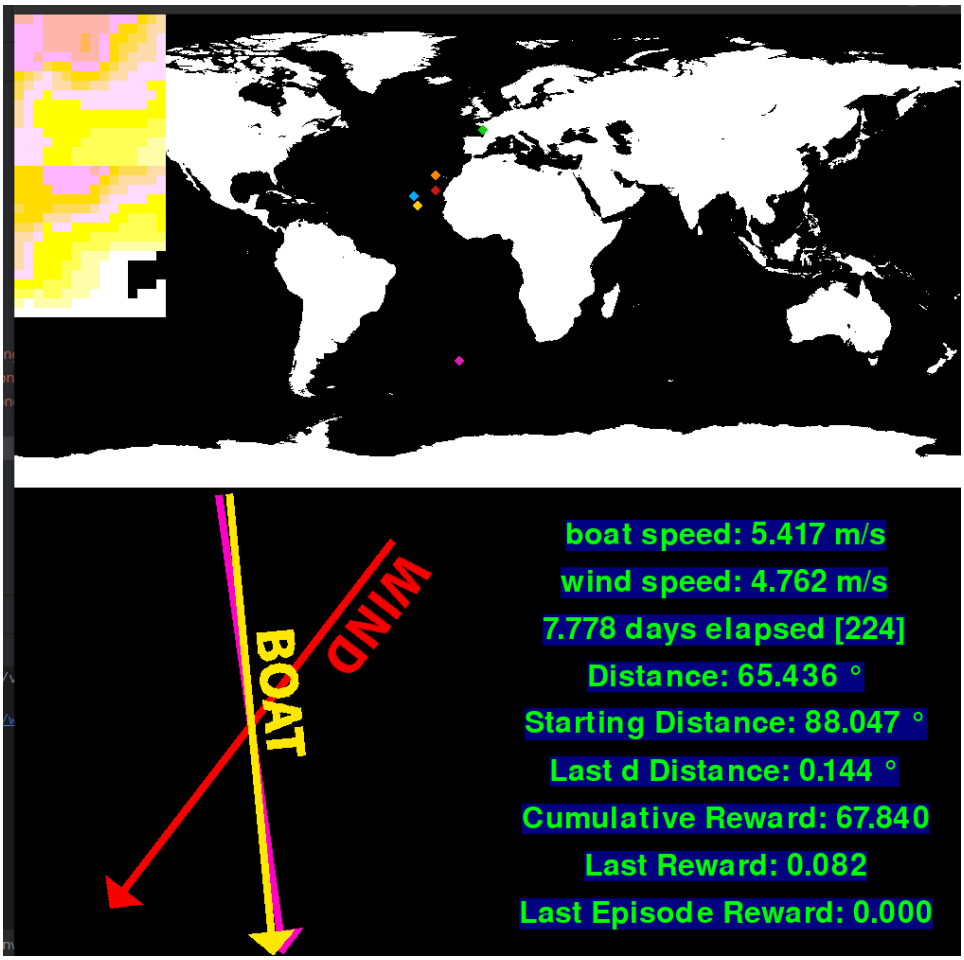


Figure 4.1: Screenshot of the GPSes almost 8 days after the virtual start of the competition. In yellow Yannick Bestaven, in light blue Giancarlo Pedote, in orange Ari Huusela and in red my agent.

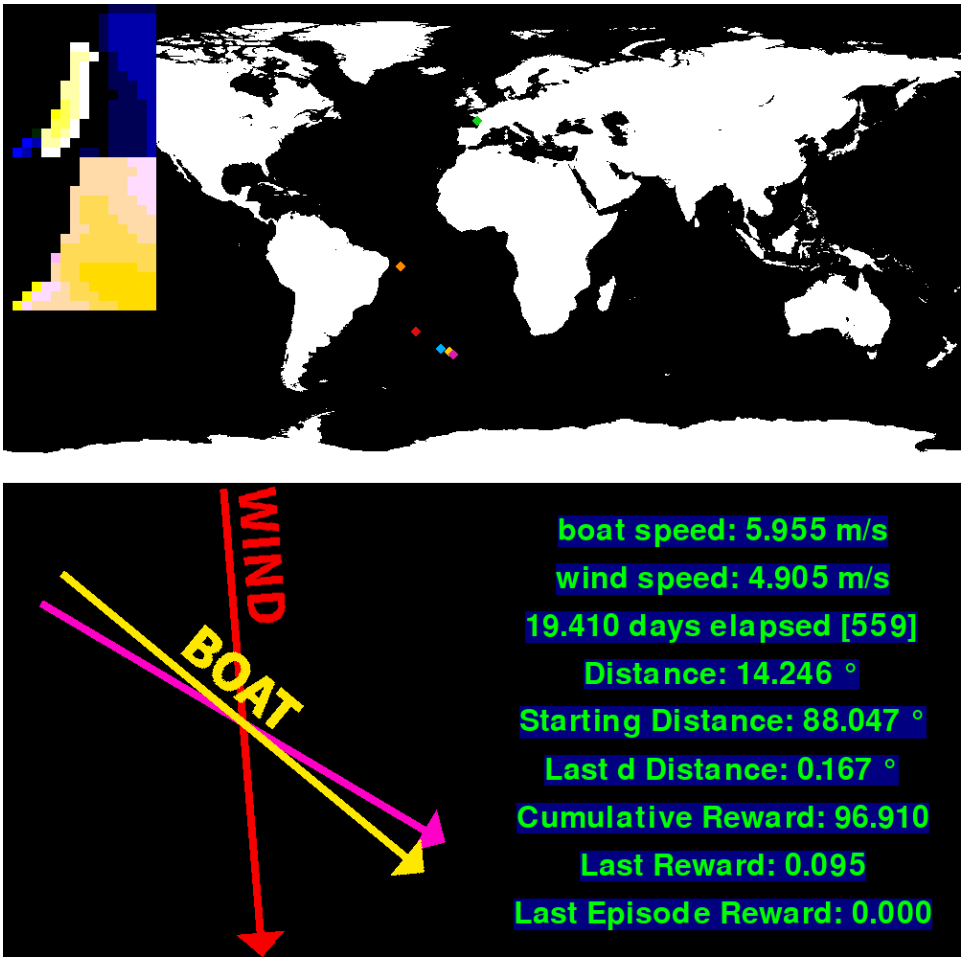
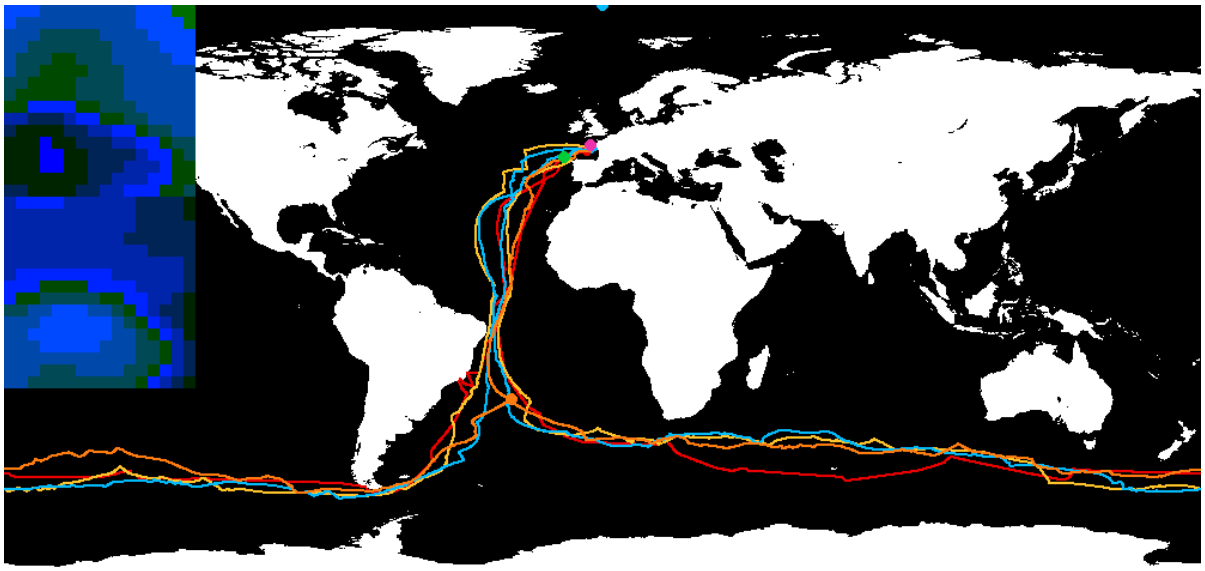


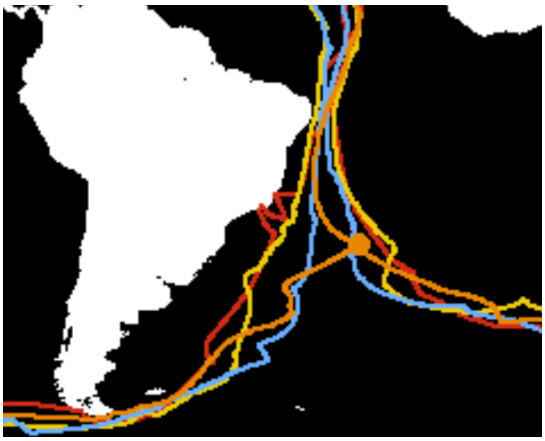
Figure 4.2: Screenshot of the GPSes almost 20 days after the virtual start, approaching the second checkpoint. In yellow Yannick Bestaven, in light blue Giancarlo Pedote, in orange Ari Huusela and in red my agent.

4.1.2 Validation Result

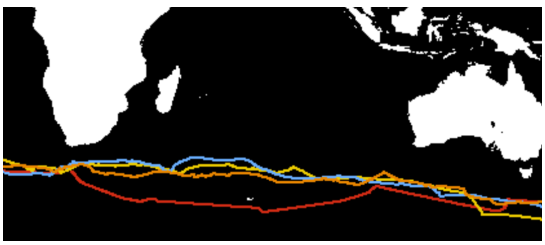
In conclusion I show the whole GPSes of the virtual regatta in Image 4.3 and the arrival data and time for each of the analyzed sailors in Table 4.1. I want to highlight the two main parts in which the agent has taken different decisions respect to humans shown in Figure 4.3b and 4.3c. In 4.3b it has decided to stay closer to the coast and to go against the wind while humans have chosen to stay more further away from land with a more manageable wind. In 4.3c it has decided to pass closer to Antarctica than all the humans.



(a)



(b)



(c)

Figure 4.3: The whole GPSes track of the competitors in my virtual Vandée Globe race [a] and two parts in which the agent has taken different decisions respect to humans: [b] and [c]. In yellow Yannick Bestaven, in light blue Giancarlo Pedote, in orange Ari Huusela and in red my agent.

Boat	Arrival Date	Arrival Time
Yannick Bestaven	28-01-2021	03:00
Giancarlo Pedote	28-01-2021	12:00
My RL Agent	04-02-2021	16:00
Ari Huusela	05-03-2021	07:00

Table 4.1: Arrivals date and time for the three selected sailors and my RL agent. They have started the virtual regatta on the 11^{th} of November 2020.

Appendix A

1D/2D/3D Linear Interpolation

A.1 1D

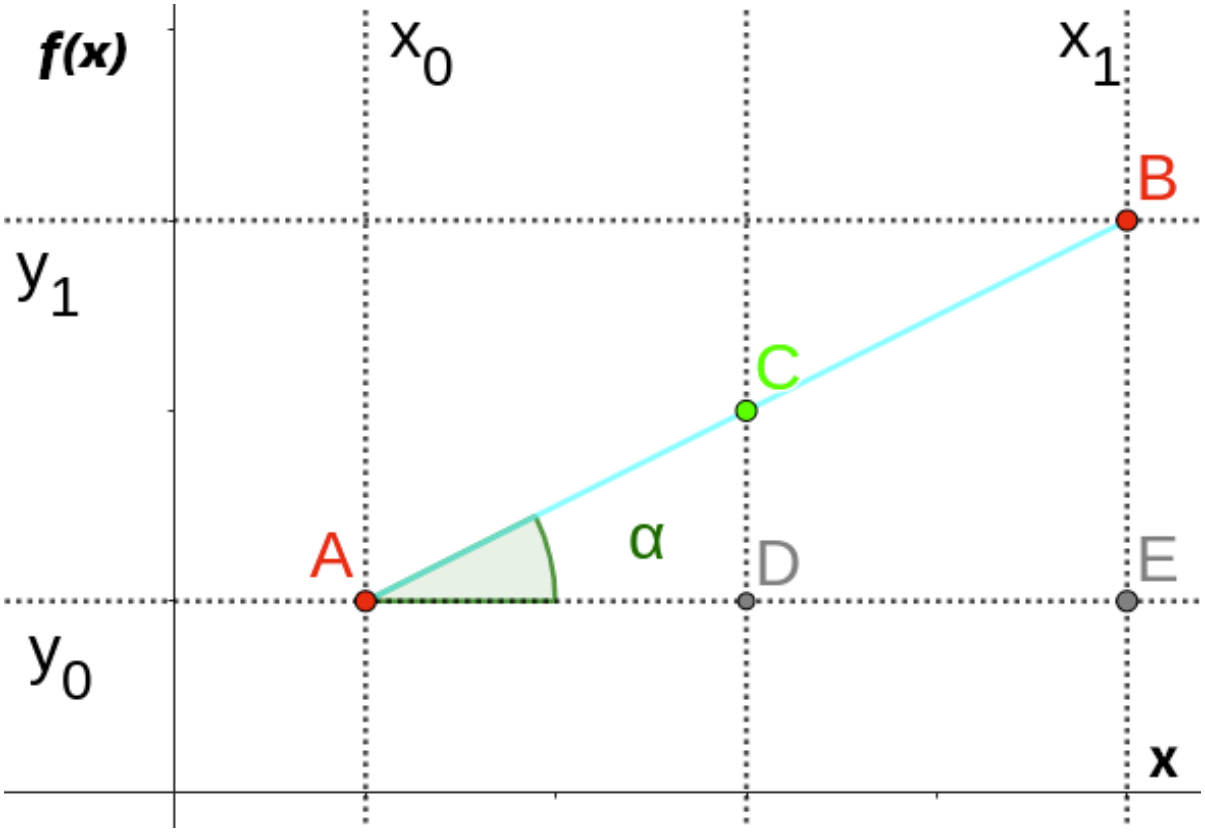


Figure A.1: Representation of a 1D-linear interpolation of $f : \mathbb{R} \to \mathbb{R}$ on a point C located between $\{A, B\} \in \mathbb{R}$ where the function values $\{f(A), f(B)\} \in \mathbb{R}$ are known.

Assuming $A = x_0, B = x_1, C = x \in \mathbb{R}$, $x_0 < x < x_1$ and to have a function $f : \mathbb{R} \to \mathbb{R}$ and $f(C) = f(x)$ is unknown but $f(A)$ and $f(B)$ are known. We can find y in the following way.

Considering the $\overline{ACD}$ triangle we have:

$$\tan(\alpha) = \frac{\overline{CD}}{\overline{AD}}$$

Considering the $\overline{ABE}$ triangle we have:

$$\tan(\alpha) = \frac{\overline{BE}}{\overline{AE}}$$

So we have:

$$\frac{\overline{CD}}{\overline{AD}} = \frac{\overline{BE}}{\overline{AE}}$$

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

After some calculation we find out:

$$y = y_0 \left(\frac{x_1 - x}{x_1 - x_0} \right) + y_1 \left(\frac{x - x_0}{x_1 - x_0} \right)$$

For later use we can define:

$$\Delta_1(x_0, x, x_1, y_0, y_1) = y_0 \left(\frac{x_1 - x}{x_1 - x_0} \right) + y_1 \left(\frac{x - x_0}{x_1 - x_0} \right) \tag{A.1}$$

A.2 2D

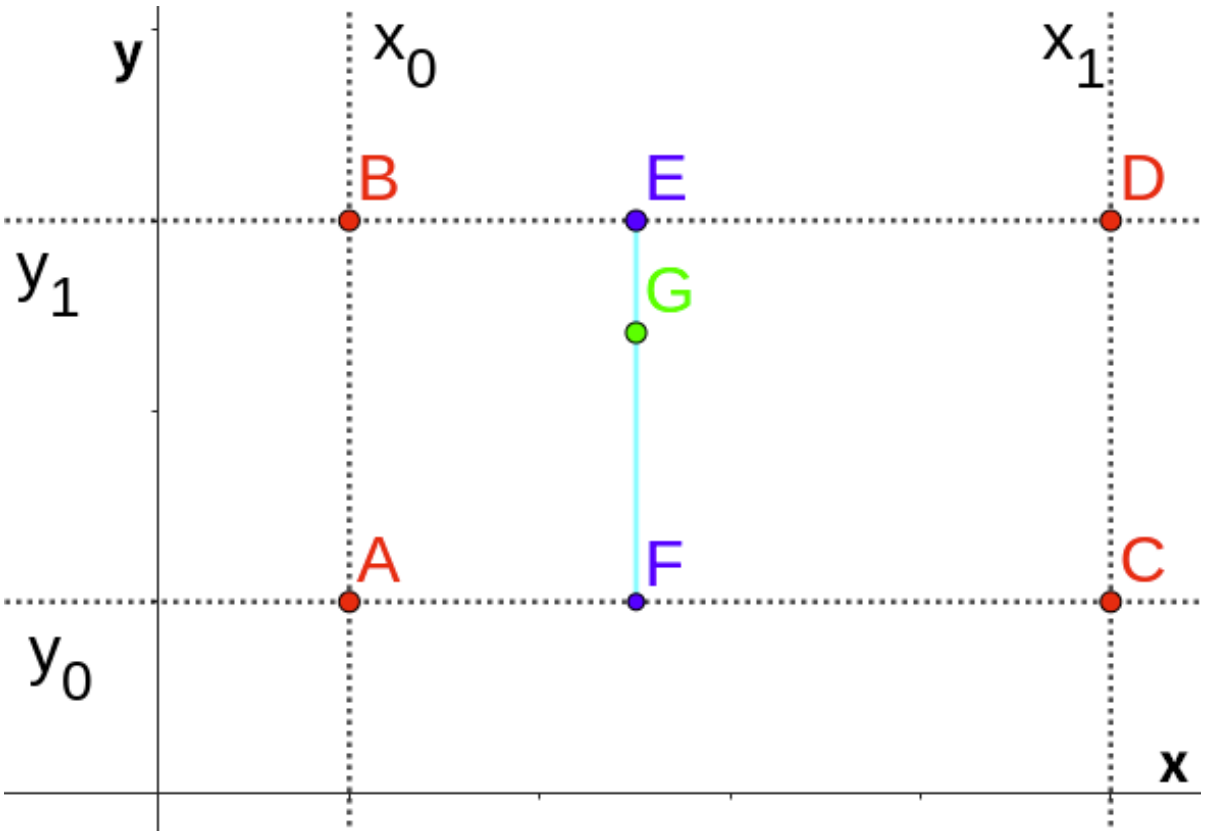


Figure A.2: Representation of a 2D-linear interpolation of $f : \mathbb{R}^2 \to \mathbb{R}$ on a point G located between $\{A, B, C, D\} \in \mathbb{R}^2$ where the function values $\{f(A), f(B), f(C), f(D)\} \in \mathbb{R}$ are known.

Assuming $A = (x_0, y_0), B = (x_0, y_1), C = (x_1, y_0), D = (x_1, y_1), G = (x, y) \in \mathbb{R}^2$, $x_0 < x < x_1$, $y_0 < y < y_1$, and to have a function $f : \mathbb{R}^2 \to \mathbb{R}$ and $f(G) = (x, y)$ is unknown

but $f(A)$, $f(B)$, $f(C)$ and $f(D)$ are known. We can find $f(G)$ in the following way. First of all we find out $f(E) = f(x, y_1)$ and $f(F) = f(x, y_0)$ using 1D interpolation:

$$f(E) = \Delta_1(x_0, x, x_1, f(B), f(D))$$

$$f(F) = \Delta_1(x_0, x, x_1, f(A), f(C))$$

and after that we find out $F(G)$ with another 1D interpolation:

$$F(G) = \Delta_1(y_0, y, y_1, f(F), f(E))$$

For later use we can define:

$$\Delta_2(x_0, x, x_1, y_0, y, y_1, f(A), f(B), f(C), f(D)) = \\ \Delta_1(y_0, y, y_1, \Delta_1(x_0, x, x_1, f(A), f(C)), \Delta_1(x_0, x, x_1, f(B), f(D)))$$

A.3 3D

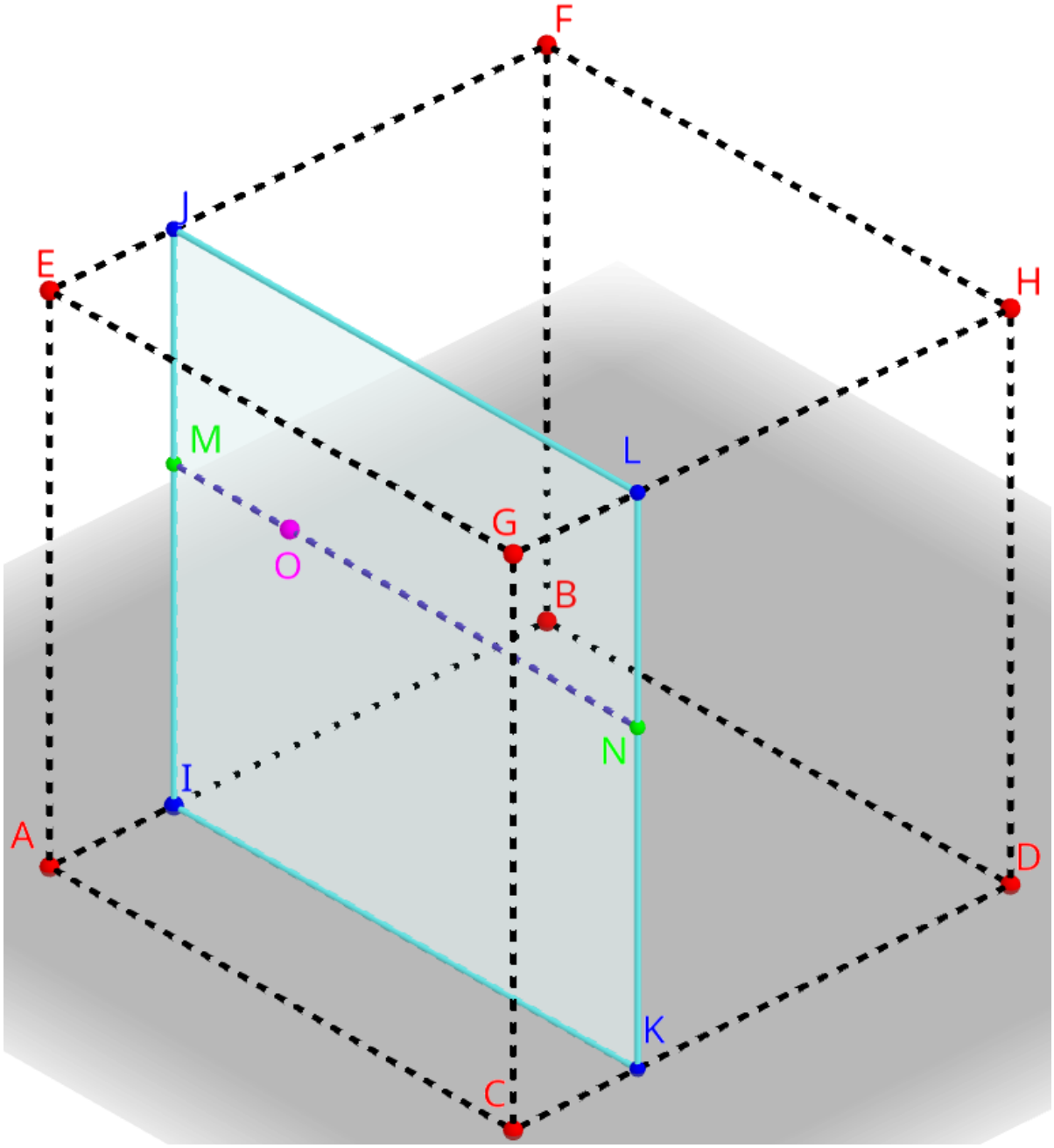


Figure A.3: Representation of a 3D-linear interpolation of $f : \mathbb{R}^3 \to \mathbb{R}$ on a point O located between $\{A, B, C, D, E, F, G, H\} \in \mathbb{R}^3$ where the function values $\{f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H)\} \in \mathbb{R}$ are known.

Assuming $A = (x_0, y_0, z_0), B = (x_0, y_1, z_0), C = (x_1, y_0, z_0), D = (x_1, y_1, z_0), E = (x_0, y_0, z_1), F = (x_0, y_1, z_1), G = (x_1, y_0, z_1), H = (x_1, y_1, z_1), O = (x, y, z) \in \mathbb{R}^3$, $x_0 < x < x_1$, $y_0 < y < y_1$, $z_0 < z < z_1$ and to have a function $f : \mathbb{R}^3 \to \mathbb{R}$ and $f(O) = (x, y)$ is unknown but $f(A)$, $f(B)$, $f(C)$, $f(D)$, $f(E)$, $f(F)$, $f(G)$, $f(H)$ are known. We can find $f(O)$ in the following way.

$$f(M) = \Delta_2(y_0, y, y_1, z_0, z, z_1, f(A), f(B), f(F), f(E))$$

$$f(N) = \Delta_2(y_0, y, y_1, z_0, z, z_1, f(C), f(D), f(H), f(G))$$
$$f(O) = \Delta_1(x_0, x, x_1, f(M), f(N))$$

For symmetry we can define:

$$\begin{aligned}
&\Delta_3(x_0, x, x_1, y_0, y, y_1, z_0, z, z_1,\\
&f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H)) =\\
&\Delta_1(x_0, x, x_1,\\
&\Delta_2(y_0, y, y_1, z_0, z, z_1, f(A), f(B), f(F), f(E)),\\
&\Delta_2(y_0, y, y_1, z_0, z, z_1, f(C), f(D), f(H), f(G)))
\end{aligned}$$

Appendix B

Angular distance between two Points on a Sphere

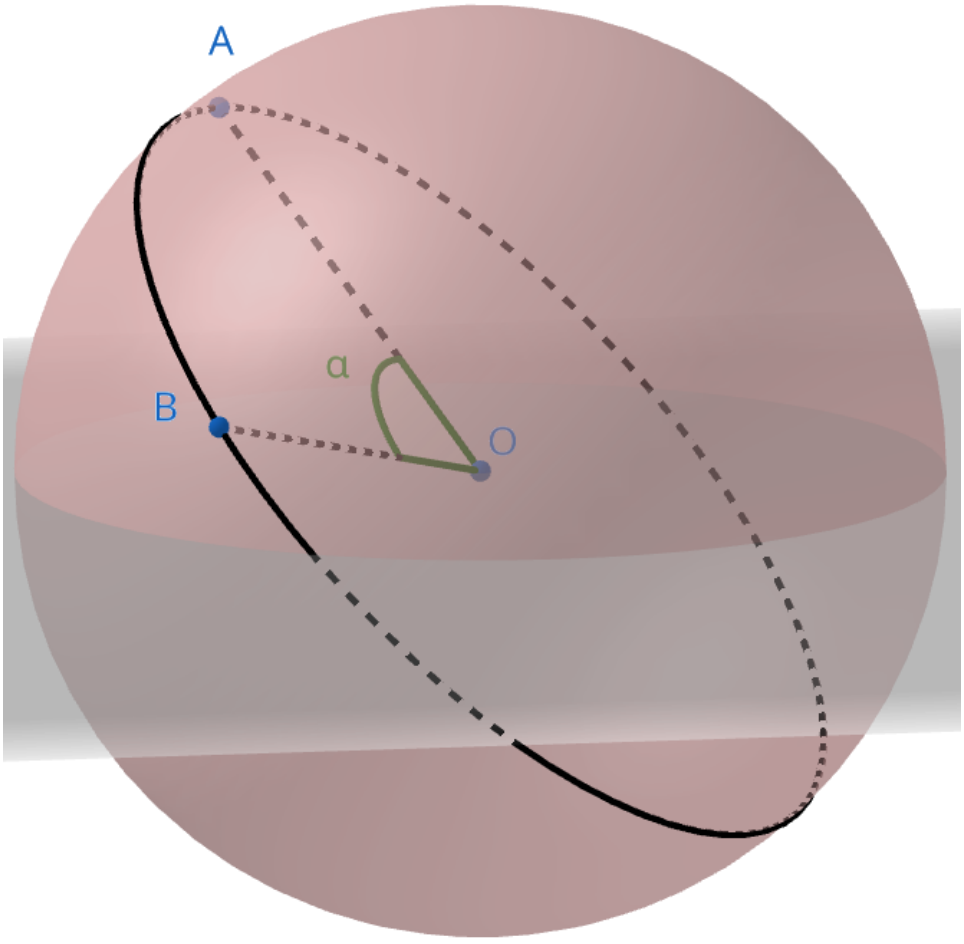


Figure B.1: A red spherical surface centered in (O) with two points on it (A and B) and the construction of the angular distance α between them.

To calculate the distance between points on earth I have used the angular distance approximating the earth to a sphere. More precisely I called *angular distance* the size of the angle α shown in Image B.1 where we want to calculate the distance between A and B with O in the center of the sphere.

Bibliography

[1] Wikipedia contributors. Sailboat — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Sailboat&oldid=1168176515, 2023.

[2] Wikipedia contributors. Canting keel — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Canting_keel&oldid=1134720463, 2023.

[3] Korak Saha and Huai-Min Zhang. Hurricane and typhoon storm wind resolving noaa ncei blended sea surface wind (nbs) product. *Frontiers in Marine Science*, 9, 2022. https://doi.org/10.3389/fmars.2022.935549.

[4] James Boyd. Double bullet day for vesper at ima maxi europeans. https://www.mysailing.com.au/double-bullet-day-for-vesper-at-ima-maxi-europeans/, 2023.

[5] Gps data of vandée globe 2020. https://www.vendeeglobe.org/. 2023.

[6] Wikipedia contributors. Sailing ship — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Sailing_ship&oldid=1171108466, 2023.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[8] Polar speed data for sailing boats. https://orc.org/sailors/active-certificates-database. 2023.

[9] Jan Pieter Waagmeester. Orc club certificate scoring. https://github.com/jieter/orc-data.git, 2023.