Criterion proving non-existence of a path between two possible states of a robot

Kévin AFFRAIX

***Abstract*—This article aims to find a criterion proving that a robot, given its degrees of freedom and commandability, cannot reach a target state from its initial state. Using interval analysis to differentiate between possible and impossible states, we will then calculate two bounds to estimate the minimum or maximum time needed to reach this state, to finally judge its validity.**

***Index Terms*—Interval analysis, Configuration space, Subpavings, Non-existence criterion, Path-finding**

I. Introduction

THIS article aims to implement the algorithm **FeasiblePath2** conceptualized by J. Luc in his article *Path Planning Using Intervals and Graphs*[1]. Therefore, we will work on the same system, to better compare the results when adding a new stop condition to the algorithm. As a reminder, the problem we tackle is the following:

- A system described by the following coordinates:

$$x : \{0, 0, 14, 14, 10, 10, 12, 12, 2, 2, 18, 18, 20, 20\} \quad (1)$$

$$y : \{0, 14, 14, 6, 6, 8, 8, 12, 12, 2, 2, 18, 18, 0\} \quad (2)$$

- Two obstacles:

$$\vec{a_1} = (8, 10);\ \vec{b_1} = (11, 10) \quad (3)$$

$$\vec{a_2} = (25, 10);\ \vec{b_2} = (28, 10) \quad (4)$$

- A bar on which is fixated the system, represented by the x-axis. The system can either slide on the bar or rotate around its fixation point. A typical situation is represented on Figure 1.

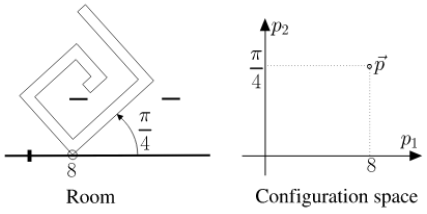


Fig. 1: Example of a possible state of the system, source:[1]

Given an initial initial state $(\vec{p1}_{init}, \vec{p2}_{init})$, the aim is to reach a target state $(\vec{p1}_{goal}, \vec{p2}_{goal})$ considering both movement constrains and obstacles. To better compare the two methods, the same initial and goal states were chosen, as shown in Figure 2.

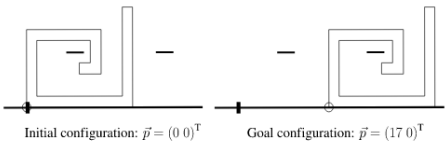


Fig. 2: Initial and goal states, source:[1]

II. State of the art

A. Interval analysis

In order to implement the **FeasiblePath2** algorithm, we must implement first a subdivision algorithm, to obtain subpavings of the configuration space.

Given a box $\vec{p_0} \in \mathbb{IR}^n$ representing the configuration space (i.e: all states considered), we want to obtain a sub-paving out of which we will build two graphs: G_m and G_p. The former would contain all boxes containing only feasible states, while the latter would also include boxes in which both feasible and impossible states mix in together. In order to do that , we will run an *inclusion test* on $\vec{p_0}$[1], that you can see on Figure 8. Three cases are to be studied:

- The result of the *inclusion test* is $\{0\}$: it means that all possible states contained in $\vec{p_0}$ cannot be reached because the system would collide with an obstacle. In this case, $\vec{p_0}$ will be discarded.
- The result of the *inclusion test* is $\{1\}$: it means that all possible states contained in $\vec{p_0}$ are possible states of the system. In this case, $\vec{p_0}$ will be added to both G_m and G_p.
- The result of the *inclusion test* is $\{0, 1\}$: it means that $\vec{p_0}$ is too big and contains both feasible and impossible states. It will later be bisected, but in the meantime, it will be added to G_m.

Based on this result, the **FeasiblePath2** algorithm will then bisect or not $\vec{p_0}$, obtaining a new sub-paving to study. Each iteration will refine the paving of the configuration space, with smaller boxes describing areas of interest.

B. FeasiblePath Algorithm

In order to find a series of boxes both containing only feasible states and linking the initial and desired state, me must decide when to bisect a box of the paving, to enrich the graphs until we find a solution. The only boxes that should be bisected are those whose *inclusion test* result is $\{0, 1\}$, named "uncertain boxes" later in the article. The first algorithm built by J. Luc to do this works as follows:

- Takes all boxes created in the wake of bisections made in the previous step, one by one.
- Identify all boxes using an *inclusion test* and add them to the corresponding graphs.
- Search a path between the initial and desired states in both graphs. Not finding a path in G_p would mean that the desired state is unreachable from the initial state, while finding a path in G_m would mean that we have found a solution.
- If no exit condition has been triggered, bisect all uncertain boxes to obtain a more refined paving, iterate the algorithm once again.

However, as J. Luc showed in his article, bisecting blindly all uncertain boxes is not efficient at all, and would considerably slow down the algorithm. The aim of the **FeasiblePath2** algorithm, which pseudo-code can be found on Figure 9, is to only bisect uncertain boxes that were chosen to build the path in G_p. By doing this, graphs would not grow exponentially at each step, and we would refine only the boxes of the paving considered the best to find the solution.

C. Limitations

Two limitations can be found in the **FeasiblePath2** algorithm. The first limitation concerns the optimality of the solution. In fact, the criterion used to measure the distance between two states is the total number of boxes necessary to link them. This leads the algorithm to prioritize large and secure boxes covering most of the path towards the desired state in a single step. However, we can see experimentally that those boxes often leads to make a detour instead of going through the optimal trajectory in the configuration space. The second limitation concerns the exit conditions: **FeasiblePath2** cannot ensure that not finding a path means that no path can be found for the problem considered.

III. Improvements

A. Compute lower and upper bounds

The new idea developed in this article is to associate to all boxes of the paving a temporal cost, forcing the algorithm to search for more optimal solutions. The second advantage is that we can add exit conditions to the algorithm that would allow it to analyse further a possible solution to decide if it is, or not, viable in our situation. To calculate this temporal cost, we associate for each state variable a unit travel cost δt_i (e.g: time needed to go from $\vec{p_1} = x$ to $\vec{p_1} = x + 1$ meters, and time needed to go from $\vec{p_2} = y$ to $\vec{p_2} = y + 1$ radians). Those values could be deduced experimentally, or calculated thanks to the technical specifications of the actuators used. Then, only a multiplication would be needed to transform a movement δd_i into time cost c_i:

$$c_i = \delta t_i \times \delta d_i \tag{5}$$

The algorithm will now try to approximate two bounds. Note that, to calculate such bound, we only know the box we are currently in, and the box we consider to go next, preventing us to optimise those bounds considering the whole path. This is due to the use of Dijkstra's algorithm utilization to find the best path in the new graph, where the distance metric is no longer equal between all boxes but weighted with the temporal cost to pass though a box. This solution has been chosen as it is one of the fastest algorithms solving shortest paths problems, when considering only positive weights[2].

Considering we have only access to the box we are evaluating, and the neighbour we consider to go next, we make here the assumption that we must be on one of the box "edges", that is to say that each state variable could only be the lower or upper bound of the interval corresponding to the possible values of this state variable in the box. The notion of edges can be seen on Figure 3. Then, excluding the edges belonging to the common boundary between the two boxes, we compute for each edge the cost vector corresponding to the minimal distance to the boundary. Considering an edge on the boundary would mean that the box we were in one step before would lead to this edge, but in this case the algorithm should choose to go directly to the next box without trying to go through the current box.

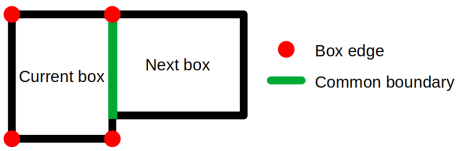


Fig. 3: Concept of box edges and common boundary

To compute the lower and upper cost bounds we then consider, for each edge, the cost vector. Each action corresponding to a state variable can be done independently (i.e: we can slide the system on the bar and rotate it at the same time), so the minimal time to reach the boundary from a given edge would be the maximum value of its cost vector $c_{max,e}$. The lower and upper bounds *m* and *M* corresponding to the whole box would then be :

$$m = \min_{e \in E, e \notin B} c_{max,e} \tag{6}$$

$$M = \max_{e \in E, e \notin B} c_{max,e} \tag{7}$$

where *E* is the set containing all edges of the current box, and *B* is the set containing all states belonging to the boundary.

B. Dijkstra weights

An issue occurred when trying to use the lower bound as a distance metric for Dijkstra's algorithm. Once some areas of the configuration space have been bisected numerous times while neighbouring areas have not, the algorithm would try by all means not to pass through the latter areas. Therefore, those bounds make great criteria for exit conditions, but fail as a distance metric, because its prevents the algorithm from exploring the configuration space.

To fix this issue, a mew metric has been implemented. In attempt to compute a value viable in every possible situation, considering the distance between each middle point (i.e: state

in the configuration space) of the two boxes was chosen. To ensure that we only cross the boxes we consider and not the neighbouring ones, a third intermediate point is introduced, which is the middle point of the common boundary between the two boxes. The Figure 4 illustrate this principle.

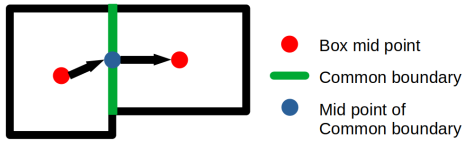


Fig. 4: Distance metric used for Dijkstra's weights

C. Implementation on the algorithm

A few changes have been made to the **FeasiblePath2** algorithm, while maintaining its skeleton:

- First, a new threshold is added to the list of parameters. If the cost lower bound associated with L^+ is above the threshold, the algorithm stops and return "No path". If the cost upper bound associated with L^- is below the threshold, the algorithm stops and return L^-. If the cost upper bound associated with L^- is above the threshold, the algorithm tries to find a faster path between the initial and desired states.
- The function **ShortestPath** is replaced with the Dijkstra's algorithm. The graphs G_m and G_p now contains cost bounds associated with the corresponding neighbours and the weight used in Dijkstra's algorithm.

IV. Comparison

A. Results

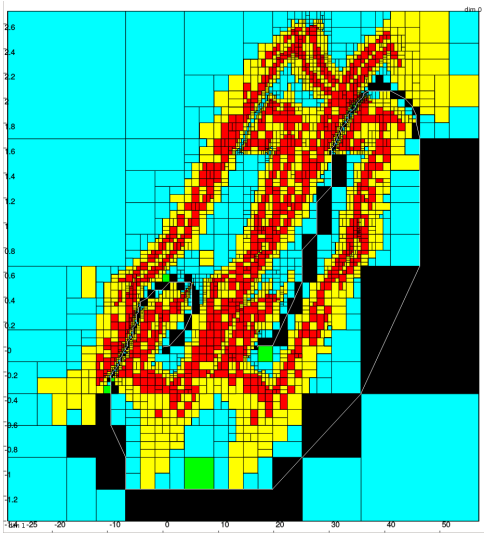


Fig. 5: Result when using **FeasiblePath2**

The new features have greatly improve the quality of the solution, as we can see by comparing the solution with (Figure 6) and without them (Figure 5). We can also see the attempts to optimize the trajectory in some critical areas that where

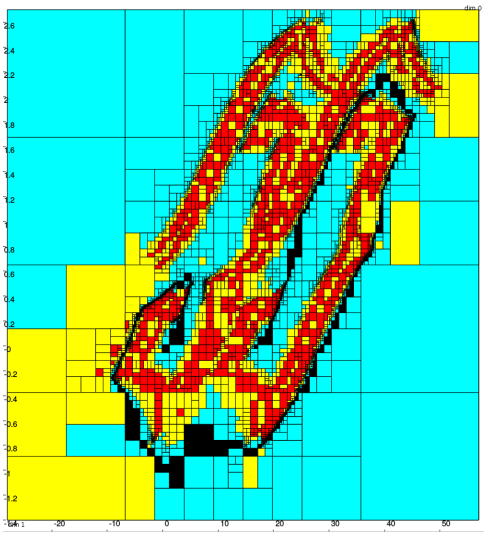


Fig. 6: Result when using new distance metrics and exit conditions

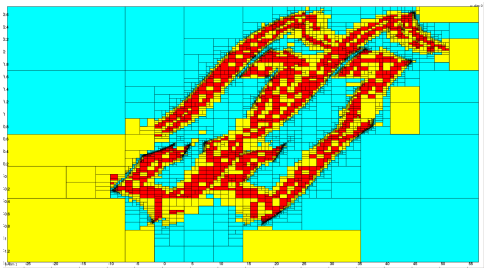


Fig. 7: Critical zones of optimization : "darker areas"

bisected numerous times, making them seem "darker" on Figure 7. Moreover, the algorithm now benefits from more exit conditions that allow it to better identify the validity of a path. Finally it also gives the user an indication on the feasibility of a transformation between two states of a system, thanks to the new configurable threshold. However, it is not yet complete and some future improvement are needed.

B. Issues and possible future improvements

First, the time needed to compute a solution is now far greater. Using an Intel Core i5-9300H, the algorithm **FeasiblePath2** found a solution in 7 minutes (with graphics on) while the new implementation took more than an hour. This is mainly due to the application of Dijkstra's algorithm at each step, which is time-consuming when working with the huge graphs generated by the sub-pavings. Therefore, we have to choose between quality and calculation time.
Then, the criterion found to prove the non-existence of a path is not absolute, and a threshold too ambitious would again lead to an infinite loop in the algorithm.
Finally in some cases the algorithm, trying to optimize the path, would not be able to find a path in L^-, passing too close to obstacles, as shown in Figure 10. To fix this issue, a randomization as been implemented on Dijkstra's algorithm (result in Figure 6), giving a configurable chance to chose a

neighbour box randomly instead of choosing the best solution. This would allow a better exploration of the configuration space, but worsen the quality of the solution found at the end.

V. Conclusion

To conclude, introducing a time as a variable to optimize allowed the algorithm to better perform in the solution it finds. The number of cases when the **FeasiblePath2** algorithm fails to reach a conclusion has also been reduced. However, the time spend to find the solution has been greatly increased, and there are still situations where it fails to reach a solution.

References

[1] J. Luc, "Path planning using intervals and graphs," *Reliable Computing*, vol. 7, pp. 1–15, 02 2001.

[2] A. Manan and S. I. A. Lakyari, "Single source shortest path algorithm dijkstra and bellman-ford algorithms: A comparative study," *International Journal of Computer Science and Emerging Technologies*, vol. 3, no. 2, pp. 25–28, Jun. 2020.

APPENDIX A
INCLUSION TEST ALGORITHM

$[t]\,([\vec{p}])$

result $= 1$;

For $j = 1$ to $j_{\max}$,

$[\tilde{x}_a] = (x_a(j) - p_1)\cos[p_2] + y_a(j)\sin[p_2]$;

$[\tilde{y}_a] = -(x_a(j) - p_1)\sin[p_2] + y_a(j)\cos[p_2]$;

$[\tilde{x}_b] = (x_b(j) - p_1)\cos[p_2] + y_a(j)\sin[p_2]$;

$[\tilde{y}_b] = -(x_a(j) - p_1)\sin[p_2] + y_a(j)\cos[p_2]$;

$[\tilde{a}] = ([\tilde{x}_a], [\tilde{y}_a])^{\mathrm{T}}$; $[\tilde{b}] = ([\tilde{x}_b], [\tilde{y}_b])^{\mathrm{T}}$;

If $\left([\tilde{a}] \text{ or } [\tilde{b}] \text{ are inside the object}\right)$, return 0;

For $i = 1$ to $i_{\max}$,

$[d_1] = \det\left(\vec{s}_i - [\tilde{b}], \vec{s}_i - [\tilde{a}]\right) * \det\left(\vec{s}_{i+1} - [\tilde{b}], \vec{s}_{i+1} - [\tilde{a}]\right)$;

$[d_2] = \det(\vec{s}_{i+1} - \vec{s}_i, \vec{s}_i - [\tilde{a}]) * \det\left(\vec{s}_{i+1} - \vec{s}_i, \vec{s}_i - [\tilde{b}]\right)$;

if $([d_1] < 0$ and $[d_2] < 0)$ return 0;

if $(0 \in [d_1]$ or $0 \in [d_2])$ and $\left[[\tilde{a}] \cup [\tilde{b}]\right] \cap [\vec{s}_{i+1} \cup \vec{s}_i] \neq \emptyset$

result $= [0, 1]$;

EndFor;

EndFor;

Return result;

Fig. 8: Inclusion test algorithm, source:[1]

APPENDIX B
FEASIBLEPATH2 ALGORITHM

FEASIBLEPATH2 $\left([t], \vec{a}, \vec{b}, [\vec{p}_0]\right)$

If $[t](\vec{a}) \neq 1$ or $[t](\vec{b}) \neq 1$, return ("Error: $\vec{a}$ and $\vec{b}$ should be feasible");

If $\vec{a} \notin [\vec{p}_0]$ or $\vec{b} \notin [\vec{p}_0]$, return ("Error: $\vec{a}$ and $\vec{b}$ should belong to $[\vec{p}_0]$");

Denote by $\mathcal{P}$ the paving containing the single box $[\vec{p}_0]$;

Repeat

$\mathcal{P}^+ = \text{Subpaving}(\mathcal{P}, 1 \in [t]([\vec{p}]))$; $\mathcal{G}^+ = \text{Graph}(\mathcal{P}^+)$;

$v_a = \text{vertex}([\vec{p}_a])$, where $[\vec{p}_a] \in \mathcal{P}^+$ and $\vec{a} \in [\vec{p}_a]$;

$v_b = \text{vertex}([\vec{p}_b])$, where $[\vec{p}_b] \in \mathcal{P}^+$ and $\vec{b} \in [\vec{p}_b]$;

$\mathcal{L}^+ = \text{SHORTESTPATH}\ (\mathcal{G}^+, v_a, v_b)$;

If $\mathcal{L}^+ = \emptyset$, return ("No path");

$\mathcal{P}^- = \text{Subpaving}(\mathcal{P}, [t]([\vec{p}]) = 1)$; $\mathcal{G}^- = \text{Graph}(\mathcal{P}^-)$;

If $v_a \in \mathcal{G}^-$ and $v_b \in \mathcal{G}^-$, $\mathcal{L}^- = \text{SHORTESTPATH}\ (\mathcal{G}^-, v_a, v_b)$;

If $\mathcal{L}^- \neq \emptyset$, return $\mathcal{L}^-$;

$\mathcal{C} = \{[\vec{p}] \in \mathcal{P}^+ |\ \text{vertex}([\vec{p}]) \in \mathcal{L}^+ \text{ and } [t]([\vec{p}]) = [0,1]\}$;

Bisect all subboxes of $\mathcal{C}$, thus obtaining a new paving $\mathcal{P}$;

Until False.

Fig. 9: FeasiblePath2 algorithm, source:[1]

APPENDIX C
CONVERGENCE ISSUES

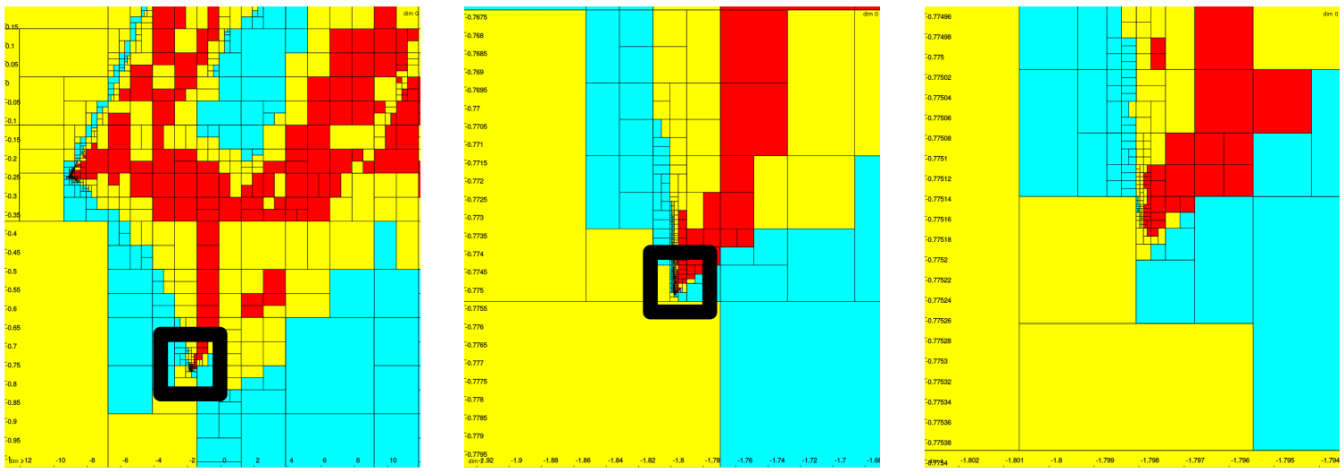


Fig. 10: Convergence issues without randomization

Autonomous path planning for a sailboat based on Deep Reinforcement Learning

Agathe Archet

Abstract—Deep Q-Learning networks have demonstrate encouraging results for path-planning for a motorized boat in dangerous environment with static obtacles. This paper explores different strategies to generalize the problem to a sailboat, whose dynamics and allowed movements are higly depedant of the wind orientation. More precisely, Reccurent Neural Network (Long Short Term Memory) networks, Actor-Critic structure and Aritificial Potential field are implemented. The issue of a non-constant wind is also adressed.

Index Terms—Path planning, Deep Q-Learning, sailboat, Recurrent Neural Networks, LSTM, Actor-Critic

1 Introduction

With their strong autonomy and adaptability, unmanned ships have gradually become the new research direction pursued by the current industry. They are reliable means for marine explorations, material transports, or military missions. Just on their own they are already capable to encounter and plan trips in dangerous environments, and can even gain efficiency when cooperating with other manned ships. Autonomous path planning then is an essential function to help the improvement of autonomous ships in their adaptation capacity. In the recent literature, Deep Reinforcement Learning (DRL) emerged as a privileged tool to deal with the collision threats, constantly raised with uncertain environments study cases. The objectives remain, first the guarantee that the boat will perform a safe trip, and only then that the boat is following an optimized trajectory.

This paper, therefore, aims to continue the previous work achieved by Silva Junior and al. [1], where path planning was based on Reinforcement Learning (Q-Learning) to guide a sailboat in an uncertain environment with a static wind. The new objective would be to introduce deeper methods such as Deep Q-Learning (DQL), in an even more dangerous environment where wind may change during the trip.

The rest of this paper is structured, as follows. Section 2 reviews the current similar works found in the literature. Section 3 presents the main actor modeling and the DQL architecture chosen for the path planning of an autonomous sailboat. Section 4 presents one first approach using LSTM networks. Finally, Section 5 concludes the paper.

2 Literature review

One first simple solution, proposed by Silva Junior and al. [1], uses Q-Learning and limited actions allowance to produce a realizable path respecting the wind constraints which otherwise tend to slow down the sailboat. Although the convergence toward a good solution is fast and cost-friendly, the Q-Table reward matrix apparently does not permit adding other dynamic constraints. However, with sufficient material resources, Deep Reinforcement Learning (DLR) may give more freedom in the definition of constraints.

Guo Siyu and al. [2] justify the choice of the DQL method for the path planning of an unmanned ship to *avoid the huge storage space of the Q-table*. Their work also employs an experience replay memory and a target network to enhance the stability of the training process, but notably relies on an Actor-Critic (AC) algorithm which is capable of handling continuous action problems and is widely used in continuous action spaces. The AC algorithm network structure includes an Actor network and a Critic network. The Actor network is responsible for outputting the probability value of the action, whereas the Critic network evaluates the output action. In the paper, all these structures are also combined with a Deep Deterministic Policy Gradient, which better manages continuous values handling, and so will not be considered here, as discrete values will be used due to available computation resources.

Another strategy studied by Clare Chen [3] in 2016, is to select a special form of neural networks combined with DQL to better acknowledge path planning as a sequence of states instead of just one and take advantage of it. Mostly used in natural language processing, Recurrent Neural Networks (RNN) give the ability to retain information from states further back in time and incorporate them into predicting better Q-values and thus performing better, on games for example, that require long-term planning.

Finally, for the path planning of an autonomous ship, Xinyu Zhang and al [4] experience a mix of RNN and Artificial Potential Field method to predict a path in an evolving environment. Some static and dynamic obstacles are taken into account to embrace even more possible sources of collision. Their work introduces a nonlinear hybrid function, evaluating distances between the ship from the different entities, for the activation function to deal with the ship's transportation which stands as a continuous systematic event.

3 Deep Q-Learning applied to a sailboat

3.1 Q-Learning actors

Reinforcement Learning involves an agent s, a set of states S, and a set A of actions per state. By performing an action $a \in A$, the agent transitions from state to state. Executing an action in a specific state provides the agent with a reward (a numerical score). Q refers to the function that returns the reward used to provide the reinforcement and can be said to stand for the quality of an action taken in a given state. For a finite Markov decision process, Q-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over all successive steps, starting from the current state. The only difference between Q-Learning and DQL lays in the value function providing the score of performing actions in a given state: Q-Learning depends on a Q-Table matrix to which a (state, action) pair is provided to obtain a score, whereas DQL relies on a neural network giving scores/probabilities arising of all actions in one given state.

From this, the agent, the environment, the actions, the reward function, the action selection strategy, and the type of neural network used shall be defined.

3.2 The environment

Although Deep Q-Learning perfectly allows the management of continuous values, a discretized environment is proposed to deal with computational resources. The scene, where the sailboat evolves, is composed of a North-West-orientated grid whose each square stands for one geographical position. Each position can be set to three profiles: an obstacle, a navigable zone, or the target ending point of the trip. The wind is evenly distributed over the scene, its force is fixed to 5 km/h for the sake of simplification, its orientation is allowed to change in a maximum range of $[-\pi, \pi]$ radians at most 2 times per episode. A visual render of the environment with the boat is implemented to allow quick supervision of the progress of the boat.

3.3 The sailboat agent

The Deep Q-Learning agent corresponds to the simplified sailboat model proposed by Jaulin and Le Bars [5], with its dynamics equations and its potential-field strategy controller. This model is defined and evolves accordingly to three main elements: a hull for state modeling, a rudder and a sail for control. Only general dynamics are considered: three degrees of freedom in a 2D plane, with waves, currents, and gravitational forces neglected. A no-go zone reflects the velocity loss due to the wind's force and its orientation with regards to the sailboat.

The sailboat's state at any step of an episode is modeled by 7 elements:

$$[x, y, \theta, \phi_w, \phi_o, d_t, d_o]$$

with (x, y) its discrete position in the environment grid, $\theta \in [0, 2\pi]$ its orientation from North, $\phi_w \in [0, 2\pi]$ the wind's orientation from North, $\phi_o \in [0, 2\pi]$ the angle formed between the boat's orientation with the line linking the boat to its closest obstacle, $d_t \in [0, 100]$ the distance to the target, $d_o \in [0, 100]$ the distance from the boat's closest obstacle. The idea to integrate other entities' distances and angles is suggested by Xinyu Zhang and al's paper [4].

3.4 The actions

Just as the environment modeling, the actions set remains discretized for a faster convergence towards a solution. In this way, actions are not directly associated with the rudder or sail commands but are defined so to permit navigation directly between every adjacent coordinates point. The controller is in charge of calculating the future command to give to the sailboat to pass by these coordinates. The 8 authorized actions are:

$$[right, up_{-45°}, up, up_{+45°}, left, down_{-45°}, down, down_{+45°}]$$

expressed in the global environment frame.

3.5 The reward function

The reward function has to reflect three different constraints to guarantee an appropriate solution to the problem: the sailboat must get closer to the target with time while avoiding collision with obstacles while being well-positioned with regards to the wind to go as fast as possible (in other words, not being orientated in the no-go zone [5]). A non-linear hybrid function can fulfill these criteria [4] as follow:

$$R = \begin{cases} 10 & \text{if the sailboat reaches the target} \\ 2 & \text{if the sailboat gets closer to the target with good wind orientation} \\ 0.5 & \text{if the sailboat gets closer to the target with bad wind orientation} \\ -1 & \text{if the sailboat gets closer with the closest obstacle} \\ -10 & \text{in case of collision} \\ 0 & \text{otherwise} \end{cases}$$

3.6 The action selection strategy

In DRL problems, the action selection strategy must be carefully chosen to allow the boat to sufficiently explore its environment and then efficiently exploit the best strategy found. To achieve it, a commonly used method is the ε-greedy strategy. Within this configuration, the agent is always looking for the action promising the best possible score in a given state, except for a probability of ε of selecting a random action. In so, the agent is encouraged to explore a priori low-reward areas but who providing a better score in the long-term.

Xinyu Zhang and al. [4] decided to fix the value of the ε random parameter with a value based on the inverse of the maximal score found for a given state. This could suggest that, with time, higher scores will emerge for every state and lower the randomness in action selection. However, in this paper, the ε is set to favor a quick convergence. The initial value of ε is fixed at 1 and decreases by a 0.9975 coefficient after every episode. The randomness is forced during the very first episodes (superior to 70% during the 200 first episodes), but descents slowly toward a final 0.1% value in a second time to reinforce the potential optimal strategy.

$$\begin{cases} \varepsilon_0 = 1 \\ \varepsilon_n = max(0.9975^{n\_episode}, 0.001) \end{cases}$$

3.7 The Actor-Critic structure

The Actor-Critic network is a hybrid Reinforcement Learning method to merge advantages of value-based methods (DQL) that map each state-action pair to a value, and policy-based methods (a reinforcement with Policy Gradients) that directly optimize the policy without using a value function. It also corrects deficiencies of each method, so a good exploration strategy with a good performance over a large set of actions is likely to be expected. A separate memory structure is set up to explicitly represent the policy independent of the value function:

- A Critic estimates the value function, in our case the action-value (the Q-value).
- An Actor updates the policy distribution in the direction suggested by the Critic (such as with policy gradients).

In a classic Q-Learning structure, the Q-table value matrix is governed by the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r(s,a) + \gamma \times \underset{a'}{argmax} Q'(s',a') - Q(s,a) \right]$$

with s the current state, a the chosen action, s' the next state triggered by a in state s, γ the discount factor giving more or less weight to the current choice compared to former choices, and α te learning rate.

This equation is kept, but decomposed among the different AC entities:

- The current value network (the actor) estimates $Q(s,a)$ from (s,a).
- The target value network estimates $\underset{a'}{argmax} Q'(s',a')$ from s'.
- A loss function, to be defined, quantifies the error $(r(s,a) + \gamma \times \underset{a'}{argmax} Q'(s',a')) - Q(s,a)$, equivalent to a $target - value$ evaluation.
- The current value network is updated with a gradient descent, parametized by a learning rate α.

The flow of information becomes:

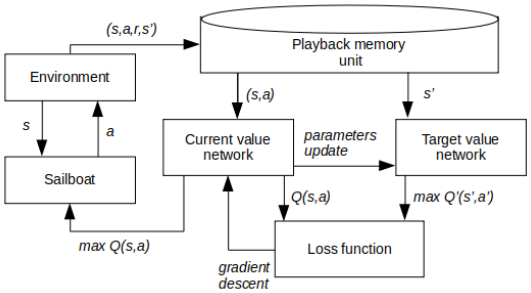


Figure 1: Flowchart of the Actor-Critic network

4 Exploration with a LSTM network

4.1 LSTM networks

The architecture is now set up and the main DQL actors are defined. The nature of the Deep Learning network, which will be the same for the actor and critic networks, will be based on a particular RNN to make the most out of sequence information: Long Short Term Memory (LSTM).

A RNN is a type of artificial neural network which uses sequential data or time-series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation or natural language processing. They are distinguished by their memory as they take information from prior inputs to influence the current input and output. However, two main weaknesses are encountered: the gradient tends to vanish over the training, and they are capable of handling long-term dependencies but can only link current with prior information, and not the prior from current ones.

These problems lead to the use of a LSTM network, which avoids long-term dependency issues thanks to gates that can retain or abandon data, to only keep relevant information.

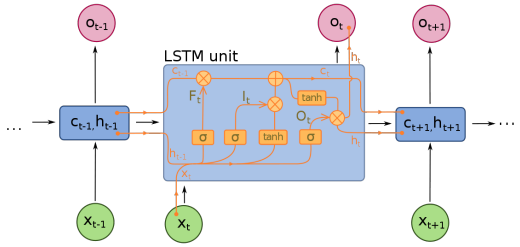


Figure 2: Detail of an LSTM chain

Each LSTM cell has an input, output, and forget gates, combinations of parametrizable activation functions. A LSTM chain is composed of several LSTM units, whose each receives as input one element of the sequence, as well as the state and output of the previous cell. Therefore, the LSTM is fed with a sequence and produces another sequence.

So the AC structure previously described has to take into account the sequence input asked by the LSTM network. The Playback Memory Unit has been designed to automatically provides a temporal sequence in the expected format.

4.2 Implementation

The implementation is realized under Python 3.6, on a Google Colab notebook to have access to powerful GPU resources. The code depends on functions from *Tensorflow* 2.4.1 and *keras* libraries. Numpy, random, and matplotlib.pyplot Python libraries are also used for convenience.

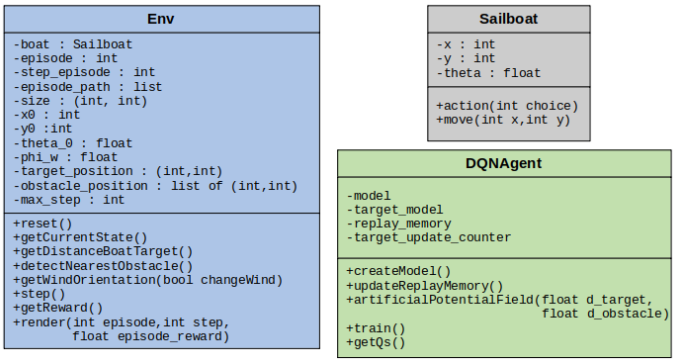


Figure 3: The Python classes

There are three Python classes that serve to distinct the principal entities :

- The Sailboat class, where the boat dynamics and possible movements evaluate the likely result of an asked action.
- The Env class, in which the geographical characteristics of the area are stored (wind orientation, target, and obstacles). Consequently, the calculation of rewards is also attached to this class. A visual rendition is available for debugging and to analyze progress over the training.
- The DQN agent, which handles the Deep Q-Learning objects: the current and target networks, the replay memory unit, an update counter to copy the weights from the current network to the target network, and effects of Artificial Potential Field (described later).

The DQL networks architecture has been adjusted manually, without fine-tuning. They are both composed of two 50 units LSTM layers (tf.keras.layers.LSTM), and a final Dense layer with a linear activation function. A Mean Squared Error loss function combined with an RSMprop optimizer (with an initial learning rate of 0.001) was the best combination found. RMSprop scales the learning rate so the algorithm converges faster. A features normalization is done before using each network for better performance.

The performance of the training is very sensitive to the maximum number of steps allowed per episode if the sailboat does not reach the target. As a first step, the area dimensions will be fixed to 15x15, and the maximum number of steps allowed per episode is set to 40. The wind is considered static in the first examples but will change of a maximum 45° angle after 20 episodes. The main ideas of the training algorithm are described below.

4.3 First results

During the trials, one issue observed is the very slow convergence of the training, even after parameter adjustments for a simple area (15x15 blocks, with two static obstacles, with a static wind). More specifically, the networks' training needs at least 1000 episodes to provide significant changes for a dangerous exploration, which seems to be normal for complex Deep-Learning structures such as LSTM. However,

Algorithm 1 Simplified DQL network training

Input Each observation *obs* describes: the current state s, the selected action a, the reward r, the next state s', the boolean of reaching the target *done*. The input is a sequence S of several observations *obs*, whose initial state is randomly chosen among the possible ones, an update frequency f_{update}.

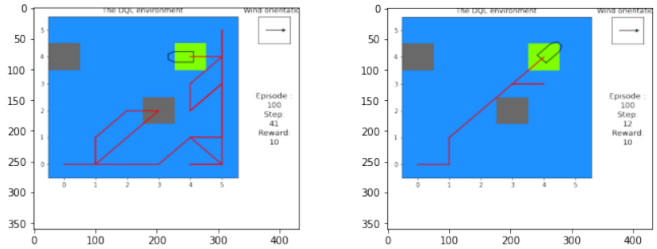
Output Trainned networks *current_network* and *target_network*

Require: A discount factor γ, an initial exploration factor ε (for randomness and as convergence criterion), a minimum sequence length l, a limit for the number of episodes e_max, a limit number of steps s_max.

for episode = 0 **to** e_max **do**
  for step = 0 **to** s_max **do**
    while not(done) or step < step_max **do**
      if replay unit memory size < l **then**
        return
      else
        - create minibatch sequence S from replay memory unit
      end if
      for obs in minibatch **do**
        $Qs \leftarrow current\_network.predict(s)$
        $Qs' \leftarrow target\_network.predict(s')$
      end for
      for obs in minibatch **do**
        $current\_Qs \leftarrow Qs$
        $current\_Qs[a] \leftarrow r + \gamma \times max_a(Qs')$
        $X.append(s)$
        $y.append(current\_Qs)$
      end for
      $current\_network.fit(X, y)$
      if f_{update} is verified **then**
        $target\_netwotk.set(current\_netwotk.weights)$
      end if
    end while
  end for
end for

running 1000 episodes approximately corresponds to 10 hours straight of calculation. Also, there is a time limitation imposed on Google colab notebooks, so that GPU units are lent for a maximum time of 12 consecutive hours. So, unfortunately, further training could not be completed.

Despite this, within the very first hundred episodes, the agent is capable of planning a first logic strategy accordingly to the initial wind orientation.



(a) 100 episodes (b) 1000 episodes

Figure 4: Performance according to the wind orientation

The best path obtained, for 1000 episodes, with a static wind, with an initial position at (0,0), is the following:

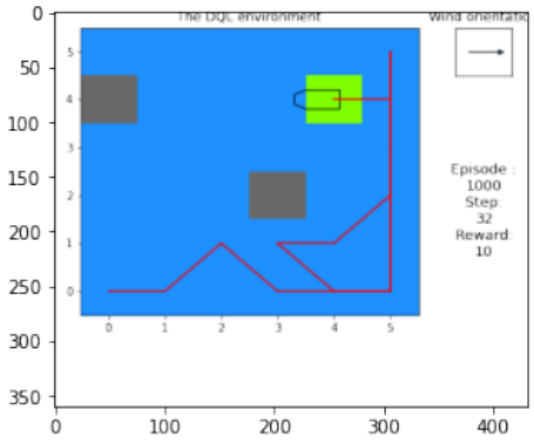


Figure 5: The best observable performance for a difficult wind

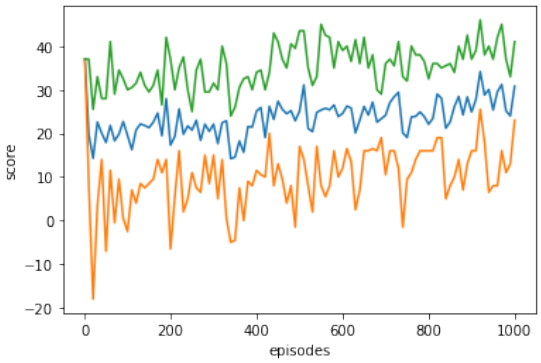


Figure 6: The associated convergence of score (minimum, maximum and average per 10 episodes)

As explained, the convergence is very slow, yet a trend is visible. The low minimum scores seen for the very first episode are due to the high presence of randomness in the action-selection strategy at this part of the training.

When adding a dynamical wind, the 1000 episodes are not enough to allow a good convergence. In order to compensate for that, Artificial Potential Fields have been added to the DQLAgent class to directly interact with the reward obtained from the environment. This solution is usually recommended to accelerate the convergence and more generally in path planning problems. Potential fields tested are attractive, and repulsive at the same time accordingly to this equation:

$$u = 0.5 \times k_{att} \times d_{target}^2 + 0.5 \times k_{rep} \times \frac{1}{max(d_{obstacle}, 0.9))}^2$$

with, d_{target} the euclidean distance between a point and the goal, $d_{obstacle}$ the euclidean distance between a point and its nearest obstacle, k_{att} a positive attraction coefficient (set to 4), k_{rep} a negative repulsion coefficient (set to -30). u is then added to the reward obtained by the boat, in this way strengthening or lessening the initial reward value depending on the place of the boat in its environment.

Even with this modification, the limitated number of episodes encountered with Google colab does not permit to account for a convergence of the path with a dynamical wind.

5 Conclusion

The defined actors, the Deep Q-Learning networks and the Actor-Critic structure implemented together are able to propose a logical path for the sailboat, among static obstacles towards a goal with a static wind. The networks are effectively sensitive to the fixed orientation of the wind and will suggest different paths accordingly.

On another hand, a lack of computation time and power has not permitted to verify if this result could be generalized for a dynamical wind. Artificial Potential Fields were added into the calculation of the reward to accentuate the importance of the sailboat's direct environment, in order to fasten convergence. But it did not overcome the limits imposed by Google colab.

Therefore, to defeat the changing wind issue, it would be possible to try these strategies with a more powerful GPU, or to consider other Deep Q-Learning networks that would better handle such constraints (changing wind in a dangerous environment), or even choose completely different Reinforcement Learning architecture and networks.

Acronyms

AC Actor-Critic. 1, 3

DQL Deep Q-Learning. 1–4

DRL Deep Reinforcement Learning. 1, 2

LSTM Long Short Term Memory. 1, 3, 4

RNN Recurrent Neural Network. 1, 3

References

[1] A. G. d. S. Silva Junior, D. H. d. Santos, A. P. F. d. Negreiros, J. M. V. B. d. S. Silva, and L. M. G. Gonçalves, "High-level path planning for an autonomous sailboat robot using q-learning," *Sensors*, vol. 20, no. 6, March 2020. [Online]. Available: http://dx.doi.org/10.3390/s20061550

[2] G. Siyu, Z. Xiuguo, Z. Yisong, and D. Yiquan, "An autonomous path planning model for unmanned ships based on deep reinforcement learning," *Sensors*, vol. 20, no. 2, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/2/426

[3] C. Chen, "Deep q-learning with recurrent neural networks," *Stanford University*, 2016. [Online]. Available: http://cs229.stanford.edu/proj2016/report/ChenYingLaird-DeepQLearningWithRecurrentNeuralNetwords-report.pdf

[4] Y. L. Xinyu Zhang, Chengbo Wang and X. Chen, "Decision-making for the autonomous navigation of maritime autonomous surface ships based on scene division and deep reinforcement learning," *Sensors*, 2019.

[5] F. L. B. Luc Jaulin, "Proceedings of the 5th international robotic sailing conference," in *A simple controller for line following of sailboats*, S. Eds, Ed. Cardiff, England: Proceedings of the 5th International Robotic Sailing Conference, 2012.

Application de méthodes de calcul ensembliste pour la conduite d'une cohorte de sous-marins suiveurs

Colin Baumgard
Elève ingénieur
ENSTA Bretagne
Brest, France
colin.baumgard@ensta-bretange.org

***Abstract*—This paper will focus on the localization and navigation of underwater scouts in front of a surface vehicle. Localization will be performed with interval analysis. A singularity avoidance controller will be implemented.**

***Index Terms*—underwater, scouts, interval analysis, codac**

I. Introduction

Cet article s'intéresse à la localisation et la navigation de robots sous-marins utilisés en chiens de garde d'un véhicule de surface. Pour cela nous utiliserons des méthodes ensemblistes fournies par la libraire CODAC développée à l'ENSTA Bretagne.

II. Situation du problème

Un des problèmes majeurs des robots sous-marins est la localisation. En effet, le milieu aquatique est un milieu opaque à la plupart des ondes électromagnétique empêchant les communications haut ou moyen-débit. Seules les ondes mécaniques basses fréquences peuvent être utilisées. Ces ondes ne permettent qu'un transfert très limité de données, et interdit l'utilisation du système de positionnement par GPS. Ainsi il faut trouver d'autres méthodes pour le contrôle des engins. Une idée est de se servir d'un véhicule de surface comme repère, et de placer les robots sous-marins en chien de garde, devant celui-ci. Cela peut avoir un intérêt militaire avec deux robots chasseurs de mines permettant de sécuriser le chemin d'un bateau de la Marine Nationale. Dans la figure 1, le véhicule du milieu à la trajectoire violette qui est un véhicule de surface ayant accès au GPS. Les deux autres véhicules ne communiquent que des distances. Nous nous plaçons donc dans le cas de robots sous-marins capables de n'émettre et de ne recevoir que des "pings" ce qui leur permet de calculer les distances entres chacun. Une première partie sera consacrée aux stratégies mises en places pour l'envoie des ces pings permettant d'acquérir les distances. Dans une deuxième partie nous concentrerons sur la contraction de la localisation du robot par un calcul par intervalle. Enfin nous déterminerons les contrôleurs que nous pouvons associer afin de réduire les incertitudes. Nous faisons le choix pour la simulation de partir

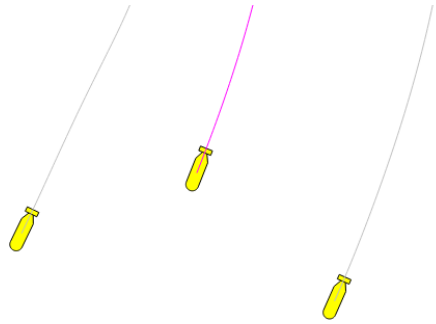


Fig. 1. Principe des chiens de garde

d'un modèle simple. On définit le vecteur d'état et la fonction dévolution des véhicules comme suit :

$$x = \begin{pmatrix} x \\ y \\ \theta \\ v \end{pmatrix} ; \dot{x} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v * cos() \\ v * sin() \\ u1 \\ u2 \end{pmatrix} \quad (1)$$

III. Communication et calcul de distances

On suppose que les missions sont de durée de l'ordre de l'heure, et que les horloges embarquées sont synchronisées durant ce temps. Chaque robot émet sur une fréquence propre permettant de l'identifier. On note 0 le robot de surface, et 1 et 2 les robots sous-marins. La séquence ci-dessous permet que chaque robot puisse calculer les distances entre chacun :

- 0 envoie un ping noté ping0
- 1 envoie un ping1 à la réception du ping0 : 0 a dt 0-1
- 0 envoie un ping0 à la réception du ping1 : 1 a dt 0-1
- 2 a reçu ping0, ping1, et encore ping0 il a donc le dt 0-1 (il y a 2 dt 0-1 entre les deux ping0)
- en parallèle, 2 à envoyé les même pings que 1, donc tout le monde a toutes les distances.

Cette stratégie nous permet d'avoir sur chaque robot les trois distances du triangles. On peut également appliquer cette méthode pour une cohorte plus nombreuse.

IV. Calcul ensembliste

A. Présentation du calcul ensembliste

Le calcul ensembliste permet de palier les problèmes d'incertitudes sur une mesure, ou simplement sur la représentation d'une valeur. Il est par exemple impossible d'avoir la valeur exacte de π mais il est possible de la borner entre $\pi - \epsilon$ et $\pi + \epsilon$ avec ϵ très petit devant π. On place donc π dans un **intervalle**. Cette méthode permet de grandement simplifier les calculs et de fournir des solutions certaines et garanties. On peut ensuite poser des contraintes sur ces intervalles. Cette méthode a été développé par L.Jaulin dans sa thèse [1] puis une librairie a été développé pour faciliter l'utilisation de ces concepts via une API d'une grande qualité [2]. Cette librairie est toujours en développement constant. La librairie intègre des contraintes prédéfinie. Pour prendre l'exemple de la localisation d'un robot, on peut définir sa position dans un vecteur d'intervalle formant une boîte. On peut partir d'une boîte dans laquelle on est sûr de trouver le robot et la contracter avec des contraintes pour réduire la zone d'incertitude. L'article vise à utiliser cette méthode pour localiser de manière relative à un véhicule de surface deux robot chiens de gardes, en ne connaissant que les distances entres ceux-ci.

B. Implémentation

Nous avons donc trois entités : un véhicule de surface, connaissant sa position et maîtrisant sa trajectoire, et deux robot autonomes sous-marins (nommés **scouts** dans la suite de l'article). Ces véhicules n'ont pas de connaissances à priori de la trajectoire à suivre. Leur but est de se maintenir à une certaine position relativement au véhicule de surface. L'article s'appuie sur les travaux présentés dans la vidéo *Distributed localisation of a group of robots with interval analysis* [3] de L.Jaulin.

1) hypothèses: On suppose qu'ils ont accès à tout instant à leur distances. Les distances sont considérés comme exactes. En effet, pour des distances de l'ordre de la dizaine de mètre, on peut considérer l'erreur négligeable. Comme la position absolue du véhicule de surface ne nous intéresse pas, nous la considérons connue de tous. En revanche la position initiale des scouts n'est pas connue. Chaque véhicule connaît avec précision son cap et sa vitesse. Cette hypothèse est simplificatrice dans la mesure où la mesure n'est pas exacte, elle est cependant réaliste dans la mesure où les technologies permettent aujourd'hui de mesurer ces valeurs avec de bonnes précisions.

2) Algorithme: A chaque pas de temps, voici les calculs effectués :

- On calcule la position exacte des robots avec les formules de déplacements
- On estime le cap de chacun des véhicules avec les deux positions estimées précédentes
- On définit les commandes de chaque robot, en accélération et changement de cap
- On déplace et on agrandie la zone de position du robot avec les informations de cap et de vitesse mesurées
- On récupère les distances entre véhicules et on contraint les zones de position

Ces calculs permettent de restreindre la zone de position suffisamment pour pouvoir piloter les scouts de manière sûr.

On voit sur la figure 2 en cyan la zone avant contraction et en rouge la zone après contraction. Les contraction sont faites à partir des contraintes de distance représentées par les cercles. On voit que la contraction permet de diminuer de moitié la

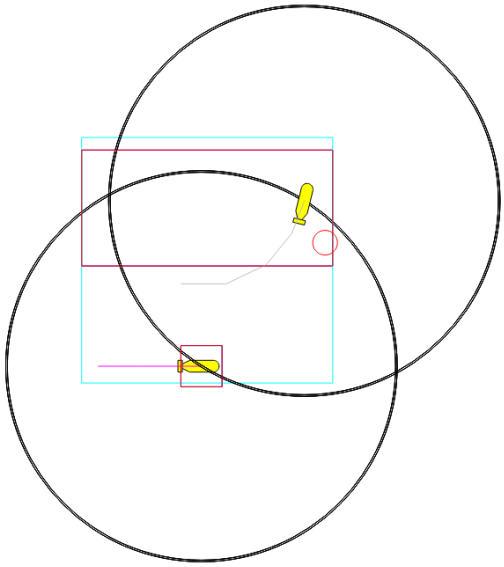


Fig. 2. Contraction de la position avec la distance

zone d'incertitude. L'exemple est ici avec deux robots, avec trois robots les contractions sont plus efficaces.

V. Contrôleur

Le contrôleur se charge de fournir au robot les consignes en cap et en vitesse. Il a en entrée les positions estimées des robots et doit diriger le robot sur son objectif. Il doit aussi pratiquer des manoeuvres particulières lorsque les zones de positions sont trop importantes

On veut que le scout soit placé devant le véhicule de surface, un à gauche et un à droite. Pour cela les paramètres d'entrées sont la distance selon l'axe longitudinale et la distance selon l'axe latéral. On récupère également la position estimée des différents véhicules, en prenant la valeur moyenne des intervalles de positions. On applique ensuite une commande proportionnelle à l'erreur en cap et en distance :

$$u1 = \dot{\theta} = \alpha * (cap_{idéal} - cap_{vrai}) \tag{2}$$

$$u2 = a = \dot{v} = v_{min} + \beta * d^{(*)} \tag{3}$$

(*) d est la distance à l'objectif devant à gauche ou à droite du véhicule de surface. De plus on borne u1 et u2 pour plus de sens physique.

1) Résultats: On obtient le résultat suivant en ligne droite et en suivie de trajectoire :

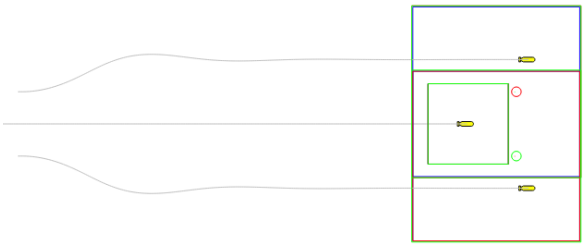


Fig. 3. Estimation de la position en ligne droite

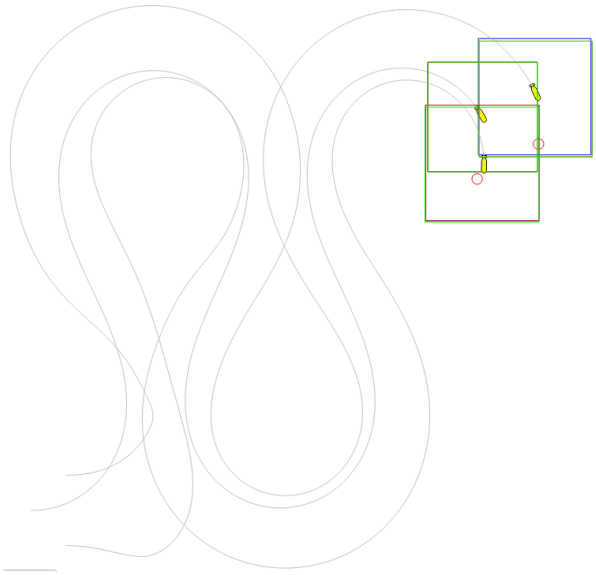


Fig. 4. Estimation de la position en suivie de courbe

On peut remarquer plusieurs choses sur ces figures (3 et 4) : premièrement on s'aperçoit que l'algorithme arrive à contracter les positions, mais qu'en suite celles-ci divergent de manière importantes. On s'aperçoit aussi que le contrôleur a en entrée une donnée erronée, et ne peut donc pas donner les bonnes consignes en cap et vitesse.

A. Limitation des singularités

Pour corriger les trop grandes erreurs de localisation , il faut rompre les singularités. En effet, lorsque les robots se déplacent en ligne droite, la contraction ne dispose pas de données nouvelles, la contraction ne peut donc pas réduire l'incertitude. Il faut donc, dans le contrôleur, ajouter une commande sinusoïdale :

$$u1 = \dot{\theta} = \alpha * (cap_{idéal} - cap_{vrai}) + \gamma * cos(t/\delta) \quad (4)$$

Cela permet de disposer de données nouvelles et donc de contracter. On applique cette nouvelle loi de commande lorsque l'aire de la zone de position est trop importante. On voit sur

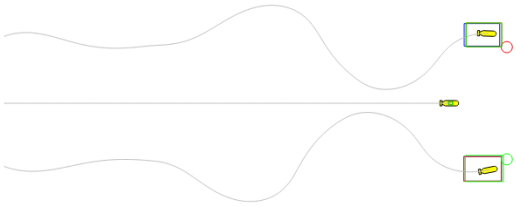


Fig. 5. Estimation de la position en ligne droite avec ruptures des singularités

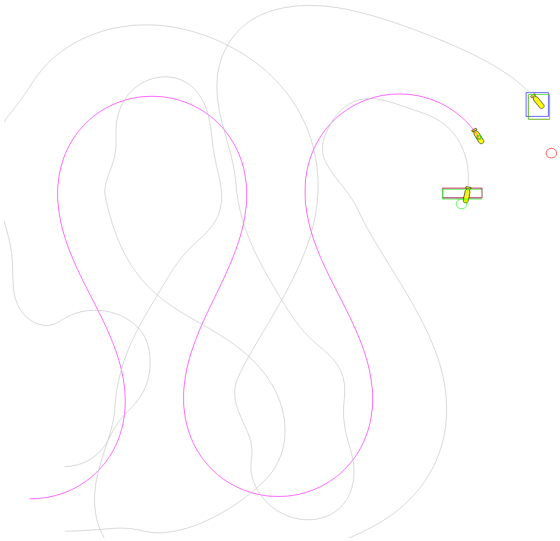


Fig. 6. Estimation de la position en suivie de courbe avec ruptures des singularités

les figures 5 et 6 que les incertitudes sont bien maintenues dans des limites acceptables.

VI. Voies d'améliorations

Le contrôleur doit également s'assurer de la sécurité des robots. Lorsque ceux-ci sont trop proches les uns des autres (i.e quand leurs zones de positions on une intersection non nulle) le contrôleur doit donner une trajectoire qui garantisse qu'il n'y ait pas de collision. Une autres solution envisageable est l'asservissement à deux profondeurs différentes. Ainsi aucune collision n'est possible. De plus il est a noter que le problème a été simplifié à un problème 2D alors qu'il est 3D. Un simple calcul trigonométrique permet de passer de l'un à l'autre, et il serait intéressant de pouvoir implémenter cette particularité. L'utilisation de tubes comme définis par Bethencourt et Jaulin [4] serait ici pertinente et permettrait d'inclure les équations d'état bien plus finement dans les contractions. Il faut noter qu'ici nous nous intéressons au temps réel uniquement et que seul une propagation *forward* est envisageable. Une autres voie d'amélioration est l'utilisation de l'algorithme SIVIA [5].

Cependant l'implémentation sous **codac** n'est pas aisée car peu documentée.

VII. Conclusion

Dans cet article nous avons pu retrouver les résultats obtenus par L.Jaulin [3] avec une implémentation dans la librairie C++ **codac**, et nous avons pu trouver un contrôleur permettant de limiter les incertitudes liées aux singularités. Des pistes d'améliorations ont été proposées. Certaines on commencées à être explorées, notamment l'utilisation des tubes, mais n'ont pas pu aboutir à des résultats tangibles pour le moment. Une discussion avec un des principaux développeurs de la librairie **codac** semble être utile pour mieux comprendre les possibilités offertes par celle-ci.

References

[1] L. Jaulin, Solution globale et garantie de problèmes ensemblistes; Application à l'estimation non linéaire et à la commande robuste, 1994.

[2] S.Rohou, Reliable robot localization:a constraint programming approachover dynamical systems, 2017.

[3] L.Jaulin, Distributed localisation of a group of robots with interval analysis, 2015.

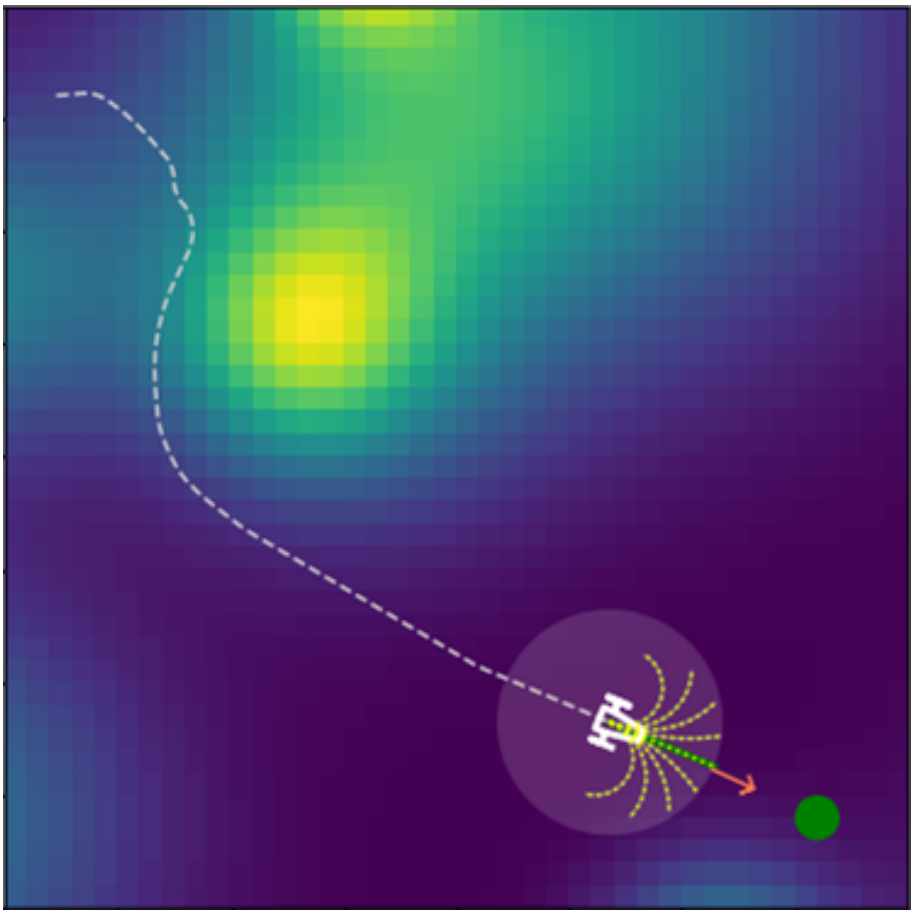
[4] Bethencourt and Jaulin, 2014, SolvingNon-Linear Constraint Satisfaction Problems Involving Time-Dependant Func-tions.Mathematics in Computer Science, 8(3):503–523

[5] L.Jaulin, Set Inversion via Interval Analysis for Nonlinear Bounded-error Estimation, 1993

Direct Model Navigation

A safe and efficient way to navigate in an unknown environment

Jules Berhault



March 2, 2021

Contents

Abstract

Among the different issues raised by autonomous rover navigation in unstructured environments, traversing rough areas is one of the most challenging. Indeed, these robots must operate on surfaces with relief and obstacles.

Knowing that it evolves in an unknown environment, the robot needs first to catch the surface geometry of the area to traverse, then select feasible trajectory according to his geometric and kinematics constraints and finally execute properly the selected path.

1 Introduction

1.1 Research subject

The method consists in determining the command space admissible by the robot (compliance with actuator constraints). This space is then discretised, and the resulting discrete commands are injected into the direct kinematic model of the robot. The generated trajectories are projected into a grid map of the robot's environment. For each trajectory, the movement of the robot is simulated and the variations in elevation of the path are observed. The trajectory that leads the robot in the right direction with a stable and safe path is selected.

My research focused on the selection of the most relevant trajectory, by developing an algorithm that determines safe and efficient movement.

The final objective is to reach a given waypoint by bypassing obstacles and relief that are too large, always heading for the most efficient path, knowing that the robot is non-holonomous and is only aware of the geometry of its surroundings at a range of 4m.

The algorithm consists in evaluating a set of trajectories (circle arcs with fixed steering angle) on a digital elevation map continuously updated as the rover moves, considering geometrical constraints on the robot chassis (maximum steering angle).

The approach has been successfully integrated within a simple autonomous navigation loop: All the possible trajectories are discrete according to the steering angle and evaluated. We then apply a selection algorithm to select the most relevant path in view of the constraints of our problem.

1.2 Context

The objective of the project is to determine the most suitable trajectory to reach a waypoint in view of the constraints of the problem and knowing that the robot has only a limited set of data from its environment. In most cases, the most suitable trajectory is the one that allows the robot to approach the waypoint at a lower cost and without crossing obstacles in a local environment.

Environmental data is acquired in real time by a range of different sensors depending on the topography of the environment:

- Geometry of the environment: LiDAR, 3D Camera, Sonar
- Geographical positioning: GNSS, IMU

Figure 1: Hanback LiDAR SmartCAR [1]

Figure 2: The two LAAS mobile robot Mana and Minnie [2]

These two robots are rovers that can carry out exploration missions in environments with unknown topography. These robots differ in the type of environment in which they are designed to operate.

Hanback's SmartCAR [1] is suitable for smooth terrain, so it must be able to avoid obstacles such as walls.

Mana and Minnie from LAAS/CNRS [2] are designed for rough terrain, but cannot cross any terrain, so it is important to be able to avoid the most difficult ones.

1.3 Hypothesis and assumed resources

To focus only on the part of trajectory planning and autonomous navigation, it was necessary to assume a few points:

- the geographical position of the robot is known at any instant
- the geographical orientation of the robot is known at any instant
- the geographical position of the waypoint is known
- the geography of the surrounding terrain is known at each data refresh step within a radius of 4 m around the robot
- the surrounding relief is discretized as an array with a pitch of 0.5m
- the refresh rate is 1.0s
- the robot has a non-holonomous structure (fixed maximum steering angle and wheelbase)

These different points have been taken into account in the algorithm in order to make the simulation as realistic as possible.

2 Direct model navigation

2.1 Direct model approach

A family of navigation methods that we are considering is the direct model method which has the advantage of being based on an element that is always defined for a mobile robot: its direct kinematic model. All the trajectories resulting from this direct model can be projected in a representation of the robot's evolution environment. From this set, it is possible to choose the optimal trajectory, according to defined criteria, which allows the robot to reach a given position while avoiding obstacles.

2.1.1 Principle of the method

This method consists of predicting the trajectories of the robot according to the commands sent to it. All the possible trajectories are then projected into a map of the local environment in order to determine the most suitable trajectory to reach the target while avoiding obstacles. The idea is therefore to evaluate the different movements that the robot is able to perform and then to select the most relevant in view of the mission constraints.

2.1.2 Kinematic model

The direct kinematic model determines the robot's movements in the Cartesian coordinate system according to the derivatives of the joint coordinates. The matrix for doing this calculation is the Jacobian matrix J:

$$\dot{X} = J \cdot \dot{q} \tag{1}$$

where $\dot{X} = \left(\dot{x}, \dot{y}, \dot{\theta}\right)^T$ is the vector of the robot's movements in the Cartesian space and $\dot{q} = (\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_n)^T$ is the vector of the movements in the joint space.

In wheeled mobile robotics, we rather use a simplified kinematic model, called a posture kinematic model:

$$\dot{X} = C(q) \cdot u \tag{2}$$

In our case, we are studying a car-type robot, (respectively the longitudinal speed of the robot and the average steering angle of the wheels) and the kinematic model is of the form:

$$\dot{X} = (\nu \cdot \cos\theta \quad \nu \cdot \sin\theta \quad \nu \cdot L \cdot \sin\alpha)^T \tag{3}$$

with θ , the orientation of the robot in the Cartesian space and L , the wheelbase of the vehicle.

Thus, the study of this case study makes it possible to study most robots with a similar structure with steerable wheels at the front and a fixed axle at the rear. The only two structural parameters that can differ are the wheelbase and the maximum steering angle.

The principle of this direct model method is to use this type of kinematic model to calculate the trajectories achievable by the robot. The set of final trajectories depends on the commands, the state of the robot (position, orientation, kinematics) and the environment in which it evolves. The integration of all these parameters makes it possible to determine a model capable of predicting the robot's movement.

The achievable trajectories are then projected in a model of the robot's evolutionary environment, and to select among them the one that seems the most appropriate to fulfil the objective assigned to the robot.

2.2 The lines of flight method

We consider that the robot is in the X_0 state at time t_0 and is trying to reach a target position while avoiding obstacles in its path. The different steps of the line of flight method are:

- generating all the trajectories Γ admissible by the robot, i.e., respecting its various constraints (kinematics, dynamics etc...) over a time horizon τ; these trajectories are also called lines of flight,
- eliminating the non-free lines of flight (which intersect or pass too close to obstacles), by projecting them in a local map which is regularly updated during navigation,
- choosing among the free lines of flight the one that will be proposed to the pilot, using a selection criterion which depends on the mission assigned to the robot.

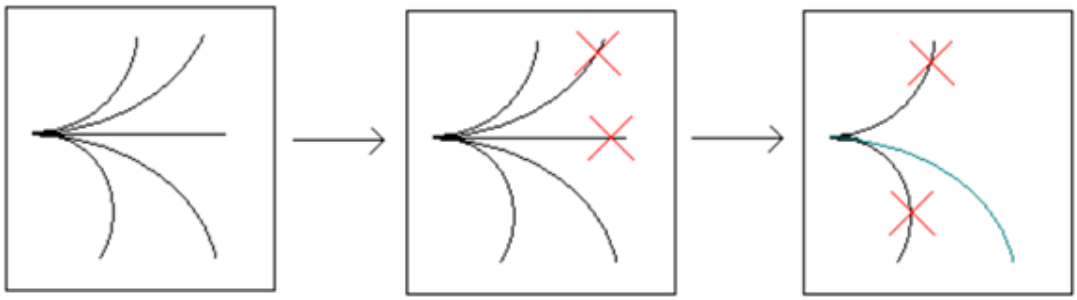


Figure 3: Principle of the lines of flight

Only a part of the selected trajectory is applied by the robot, the whole process is then repeated at the sampling time following $t_0 + T_e$, with the new information collected as the robot progresses.

2.2.1 Generation of the lines of flight

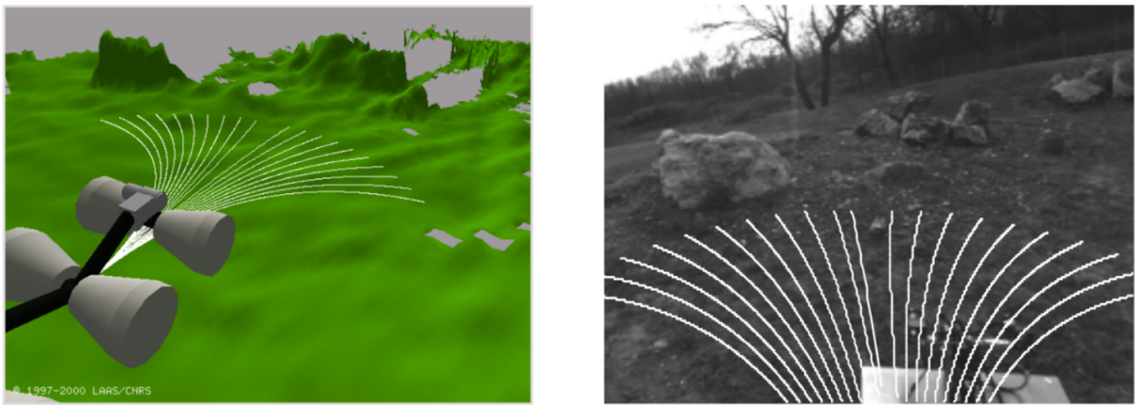


Figure 4: The set of circle arcs evaluated at each step of the loop LAAS/CNRS[3]

In order to choose a set of commands to be executed, we must first study the result of all the commands that can be carried out. In our case, we are studying a car-type robot, $u = (\nu, \alpha)^T$ (respectively the longitudinal speed of the robot and the average steering angle of the wheels), which brings us to look for an optimum with a combination of two variables. This search for an optimum in a continuous two-dimensional space would require too much computing time.

The method proposed by David Bonnafous in "Motion generation for a rover on rough terrains" consists in considering that the vehicule is a Dubins car (moving and turning with constant speed) and discretize the space of the achievable values of the steering angle . With this method, the lines of flight take the form of arcs of circles with radii of curvature giving different directions evenly distributed towards the front of the robot.

Knowing that the discretized space of the achievable steering angle α values is between $-\alpha_{max}$ and α_{max} (with α_{max} the maximum steering angle the robot structure can achieve), we can play on the number of possible choices of angles depending on the complexity of the environment and the robot's computing power. All we need to do is to simulate the robot's movement with the previously determined kinematic model for each possible angle control, assuming the speed remains constant. We then obtain an estimate of the possible trajectories for each associated steering angle.

2.2.2 Elimination of the non-free lines of flight

The choice of this trajectory must not cause the robot to collide with obstacles.

To do this, a simple algorithm runs through these trajectories and determine, for a given angle, at each iteration, whether the robot will collide with an obstacle. In this case, the line of flight is then eliminated. This is called a non-free line of flight and this possibility will be banned from the selection algorithm.

2.2.3 Selection of the best free line of flight

Finally, the lines of flight method must make it possible to determine the most suitable trajectory to meet the mission objectives. In our case, it is a question of reaching a waypoint in the most safe and efficient way (avoiding rough terrains and relief).

The generated trajectories are projected into a grid map of the robot's environment. Each grid square is assigned with à value which quantifies the elevation of the environment on this square. The grid map represents the map of the surrounding environment observed by the robot through the sensors and evolving over time.

The same algorithm that runs through the projected trajectories, records, at each iteration, the values of the boxes crossed. These values will then serve as a set of data associated with the different angle controls. This data is compiled to give a weight on each of the angles quantifying the interest of selecting this command.

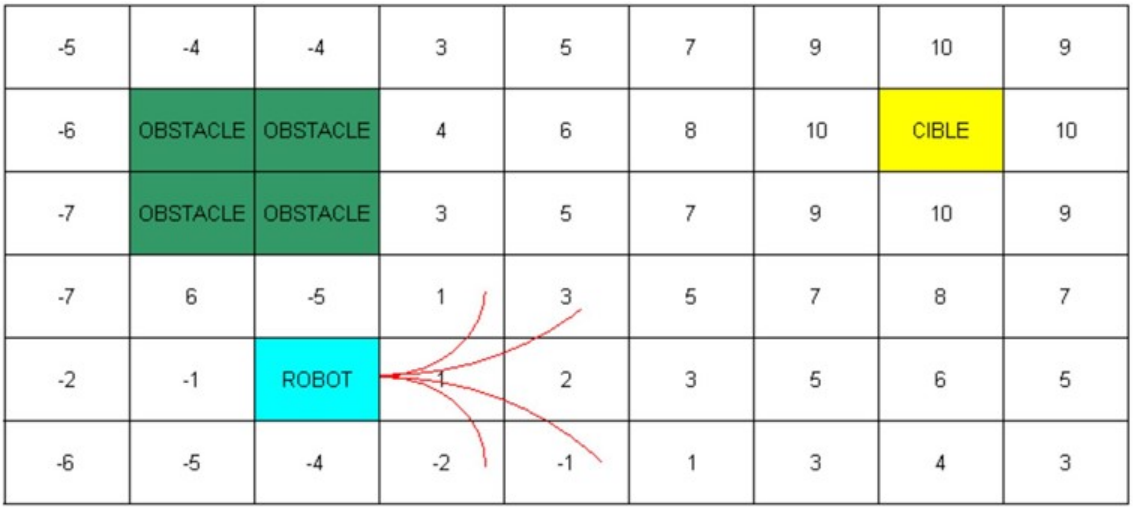


Figure 5: Projection of a set of realizable trajectories in a map of the environment in the form of an occupancy grid [4]

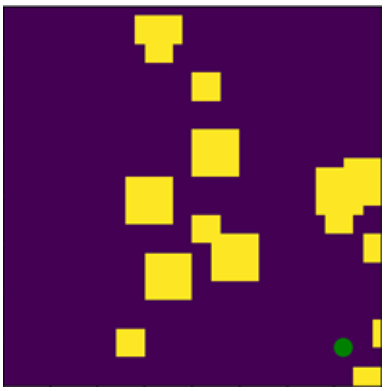
These weights are then associated with those given to the angle commands to maintain the heading towards the waypoint.

Finally, the trajectory that leads the robot to the square with the best associated value is selected, and the corresponding commands are sent as instructions to the robot's actuators.

3 Presentation of the simulation algorithm

3.1 Generation of a random simulation environment

3.1.1 Smooth ground with marked obstacles

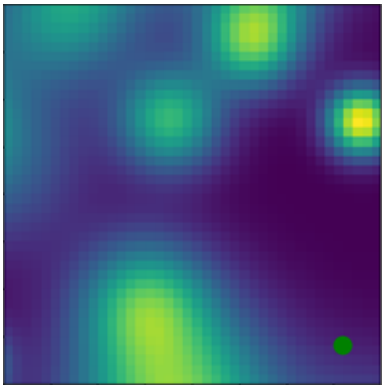


Flat environment containing randomly placed, randomly sized obstacles.

- A box containing the value 0 means that the location is passable (purple)
- A box with a value of 1 means that an obstacle occupies the location (yellow)

```
create_env_obstacles(extent_env, n)
```

3.1.2 Rough terrain with relief



An environment containing randomly placed, randomly sized bumps.

- Each box contains a float that quantifies the elevation of the terrain on that location (blue is deep, yellow is high)

```
create_env_relief(extent_env, n)
```

3.2 Algorithm

3.2.1 Integration of the kinematic model

Dubins path :

In the simulation, we consider the car follows Dubins path because this model is commonly used in the fields of robotics for path planning. Therefore the speed ν become constant and equal to 1 and the turn rate control u is bounded.

In this case the maximum turning rate corresponds to some minimum turning radius (and equivalently maximum curvature). The prescribed initial and terminal tangents correspond to initial and terminal headings. The Dubins path gives the shortest path joining two oriented points that is feasible for the wheeled-robot model.

$$\dot{X} = (\cos\theta \quad \sin\theta \quad L \cdot \sin\alpha)^T \tag{4}$$

where (x, y) is the car's position, θ is the heading and u is the turn rate control.

So we can offer a function to compute the state of the system at each new instant of the simulation.

```
f(x, u, wheelbase)
```

The function calculates the derivative of Cartesian coordinates $\dot{X} = \left(\dot{x}, \dot{y}, \dot{\theta}\right)^T$ in order to model the movement of the robot with an Euler method.

3.2.2 Integration of lines of flight algorithm

```
path_finder(x, dtsim,  all_alpha, extent_env, grid_unit, d=4)
```

This function takes as arguments the state of the robot, the time discrete step, the list of feasible steering angles, prediction distance as well as other environment data.

For each steering angle, the function predicts the trajectory step by step, assuming the controls remain the same. At each iteration, it reads the value contained by the box corresponding to the location of the robot at that moment and determines if it is an obstacle. If it is, the trajectory is rejected by means of a flag and moves to the next angle. If not, the loop continues to predict the same angle command.

Once the trajectory is planned, it calculates the standard deviation of the recorded values and moves to the next angle.

When all the lines of flight have been estimated, the `path_finder` function returns the variable `all_alpha` containing the steering angle values, the standard deviation of the values of the boxes used and the flags indicating the presence of obstacles on the various paths.

```
alpha_choice(x, pc, all_alpha, last_alpha)
```

The aim of this function is to select the best suited steering angle to the situation, considering the mission objectives defined previously: reach the point while avoiding obstacles and relief.

To do so, this function compiles trajectory data according to three selection criteria:

- variations in elevation of the paths
- the course deviation
- stability in the direction of the robot

The first is given by the previous function calculating for each steering angle the standard deviation of the elevations of the environment. A weight of interest is therefore assigned to each steering angle, which will make it possible to assign an order of priority in the selection of the most suitable angle.

```
costs = all_alpha[2]
```

The second criterion is given by a directly visible value, the heading deviation. To implement this criterion in the decision-making process, two methods were compared.

The first one consists in assigning to each angle a weight of interest proportional to this course deviation. `trajectory_relief_1.1.py`

The second consists in distributing the weights of interest corresponding to the steering angles according to a Gaussian function centred on the heading angle. `trajectory_relief_2.0.py`

Experimental comparison of these two methods revealed that the distribution of the weights in proportion to the heading deviation showed more efficient results. The latter method was therefore chosen

```
direction = abs(sawtooth(x[2] + all_alpha[0]-thetac))
```

As the last selection criterion was stability, an integration method was applied. By retaining the previous steering angle values selected, an average value can be deduced which corresponds to a selected angle trend. Approaching this trend would increase stability, which is why a weight of interest is used for each steering angle. In the same way as the above criterion, these weights are distributed proportionally or Gaussian to the average value of the angles previously selected.

```
integ = abs(sawtooth(all_alpha[0]-np.mean(last_alphat)))
```

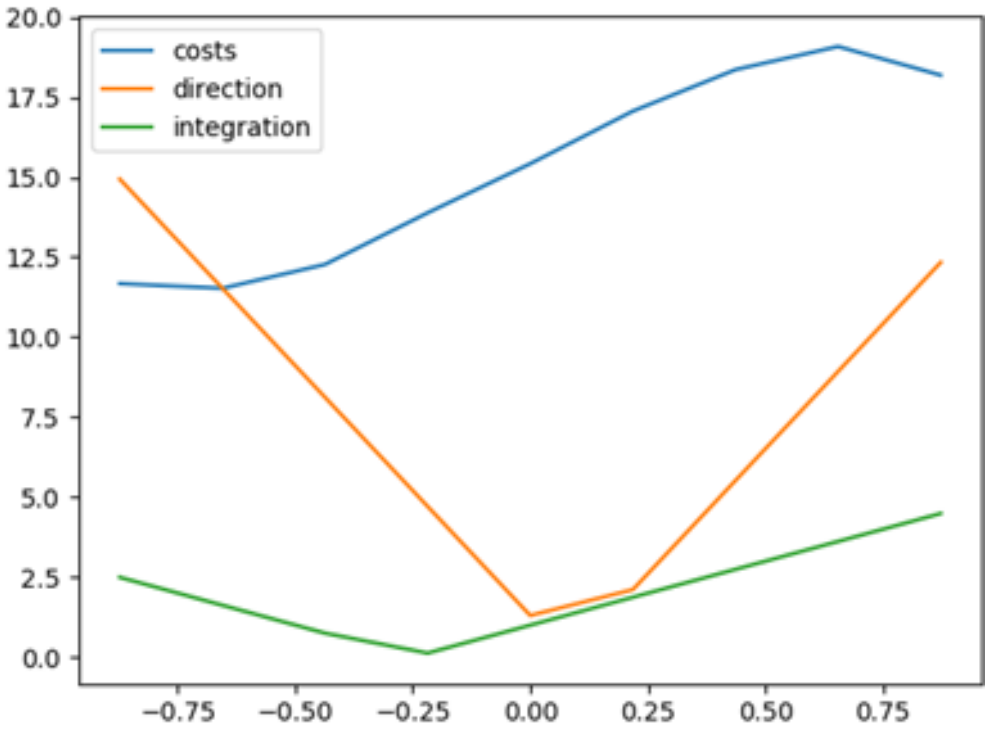
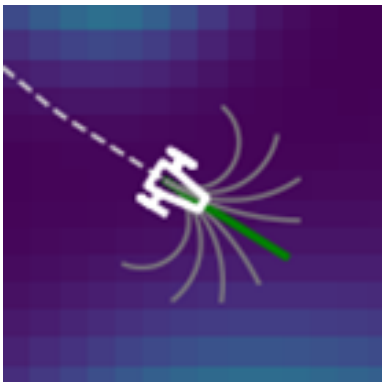


Figure 6: Overview of the distributions of values of the three selection criteria

Finally, the selection function adds these three distributions of values together with their respective coefficients (kc, kp, ki) to generate a total interest weight corresponding to each steering angle. Thus, one simply selects the angle control to be applied by determining the best value, here one chooses the angle associated with the minimum.

```
values = kc * costs + kp * direction + ki * integ
```

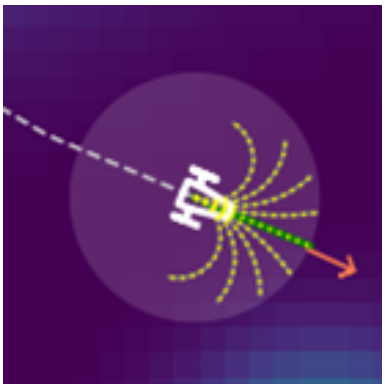
3.3 Robot drawing



Draw the projected lines of flight non-free trajectories are red, selected one is green

```
draw_trajectories(x, all_alpha, d, wheelbase)
```

Draw the circle of confidence representing the current knowledge of the environment given by the LiDAR (4m)



Draw the direction towards the target to be reached

```
draw_cap(x, pc)
```

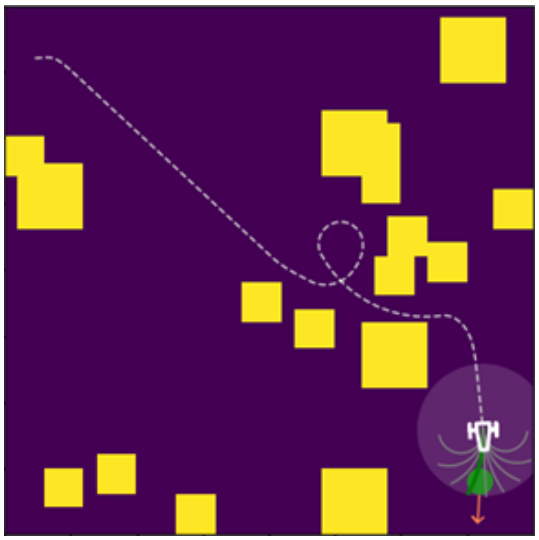


Figure 7: Example of a mission in a smooth terrain

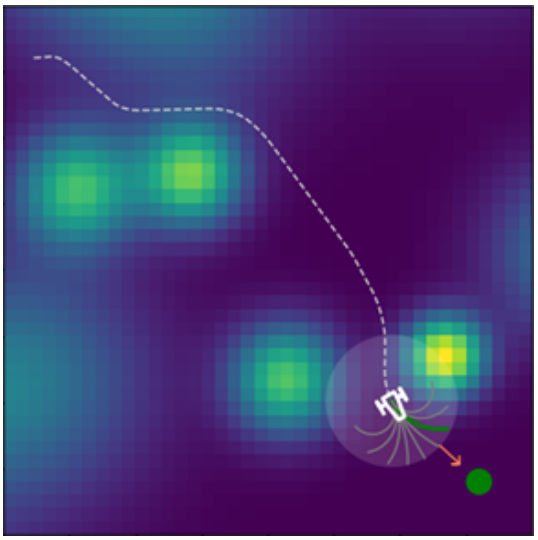


Figure 8: Example of a mission in a rough terrain

As we could expect, the robot seems to avoid the obstacles in the flat space with obstacles and bumps in the space with relief.

References

[1] Hanback Electronics, https://hanback.com/en/archives/1886

[2] LAAS CNRS, https://www.laas.fr/public/fr/deux-semaines-de-dur-labeur-au-maroc-pour-les-robots-mana-et-minnie

[3] S. L. T. S. David Bonnafous, *Motion generation for a rover on rough terrains*, LAAS/CNRS, Toulouse, 2019.

[4] Nicolas Morette, *Contribution à la navigation de robots mobiles : approche par modèle direct et commande prédictive* (French), [*Contribution to the navigation of mobile robots: direct model approach and predictive control*], Université d'Orléans, Orléans, 2010.

[5] T. Belker and D. Schulz, *Neural Network Hybrid*, 2003.

List of Figures

Flexible Tether Modeling for Underwater Environment Simulation

Quentin Brateau, ENSTA Bretagne, Brest, France, quentin.brateau@ensta-bretagne.org

***Abstract*—Simulation in the field of robotics is a powerful tool. Indeed, it allows to easily and quickly test the robot in different conditions and to have a reproducibility of the results. Then it let us be able to create situations that would be difficult to find in reality, to make sure of the robot's behavior. It should be noticed that the simulation of robots does not replace tests in real conditions, but it remains practical during the development phase. However, the simulation of robots in the maritime environment is a field that still has shortcomings, especially when we want to simulate the tethers of submarine robots. Indeed, it is not easy to find an analytical way to simulate a tether that does not require large resources, especially when simulation environments become complex. This is why we will try to suggest a finite difference simulation of the tether by proposing a behavioral model of force between each tether element. The results seem satisfactory and with a correct initialization of the position of the tether elements, the behavior of the tether seems quite right.**

***Index Terms*—Tether, Modeling, Simulation, Remotely Operated Underwater Vehicle.**

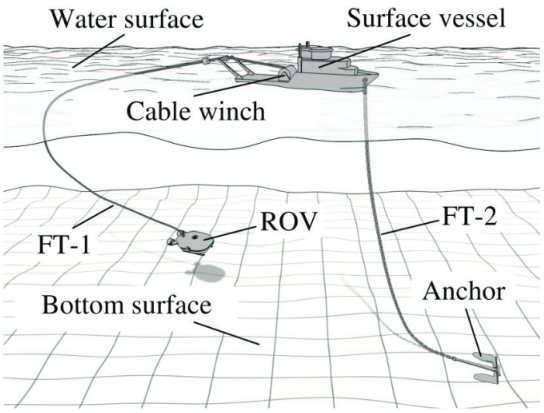


Fig. 1: Example of a complex underwater environment, *O. Blintsov, "Development of the mathematical modeling method for dynamics of the flexible tether as an element of the underwater complex", 2017.* [1]

I. Introduction

In the world of marine and underwater robotics, we can identify two categories of elements: mobile marine objects (MMO) and flexible tethers (FT) [1]. Mobile marine objects include surface vessels, submarines, and remotely operated vehicles [2]. Flexible tethers can represent umbilical cables, traction cables, and anchor chains in the marine environment [2]. They constitute all the necessary elements to achieve a mission in this environment. FIGURE 1 shows an example of a complex underwater environment as we could find in real conditions.

The simulation of moving marine objects is something well known. We are now able to know from state equations the behavior of robots in their environment, and these equations are known for ships, submarines, sailboats, etc... [2], [3]

On the other hand, determining the behavior of a flexible tether becomes more complicated. Indeed, the equation of motion of these objects involves non-linear partial differential equations and the motion between the different objects in the environment are dynamically dependent [1]. It is well illustrated that if the boat moves, it will induce a motion in the flexible tether that will modify the trajectory of the remotely operated vehicle. This is why it is difficult to find a way to model flexible tethers in underwater environments.

Most of the existing solutions in terms of physical simulation are position-based simulations [4]. This is a field that is well found in the various references that propose flexible tether simulation methods. The idea is to make a simulation based on a discretization of the tether by finite elements, whose successive positions over time are calculated by a force-based approach [1], [5]–[9].

The problem induced by this method is that it is essential to know the force that each node applies to its neighbor, which is not necessarily known. Some use the equations of continuum mechanics to determine this force [8], [9], and others propose a method based on a proportional corrector on the erroneous distance between adjacent nodes [1], [5], [7].

The solution proposed in this paper is to propose a behavioral model of this force and to see to what extent the results given by this modeling are physically acceptable. This solution has not been as advanced in the other papers proposing a simulation based on position by discretization [1], [7]. We will therefore be able to use this proposed force model to make a finite element simulation.

II. Formalism

Suppose we want to simulate a tether of length L. We will then divide it into a finite number n of nodes connected by links. These links should be of length $l = \frac{L}{n-1}$ as the two nodes at the ends of the tether will not be connected to any other links.

We will focus here on the case where the first node and the last node are immobile, because otherwise we would have to simulate a mobile marine object that would be attached at the end, which is not the goal of our study.

Next, it is necessary to make a balance of the forces that apply to each tether element. For this simulation, we will

take into account the weight, noted $\mathbf{w}$, the buoyancy, noted $\mathbf{b}$, the force exerted by the previous element on the considered element, noted $\mathbf{f_p}$, as well as that of the next element, noted $\mathbf{f_n}$, and the drag force, noted $\mathbf{d}$

- **Weight $\mathbf{w}$** : Considering that each element has a mass m, and by noting g the standard gravity, we have :

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ -m.g \end{bmatrix}$$

- **Buoyancy $\mathbf{b}$** : If we note the volume of each element V and ρ the density of the fluid in which the tether is immersed, we have :

$$\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ \rho.V.g \end{bmatrix}$$

- **Tether force $\mathbf{f_p}$ and $\mathbf{f_n}$** : It is difficult to find an analytical form to describe these two forces. Therefore, we must find a way to describe them. This is why we will use a behavioral model here. We know that each node will have to be at a distance l from each of its neighbors. We can assume that the system behaves here as a three-dimensional damped mass-spring system and we will then consider that these forces are like elastic spring forces and viscous frictionnal forces.
  By noting then p_p the position of the previous node and p_c the position of the current node, by introducing three coefficients K_p, K_d and K_i allowing to express the stiffness with which a node will correct its position with respect to its neighbors, we are able to express the behavioral model of these two forces:

$$\mathbf{f} = -\left(K_p \cdot e(t) + K_d \cdot \dot{e}(t) + K_i \cdot \int_0^t e(\tau) \cdot d\tau\right) \cdot \mathbf{u}$$

  In these expressions, it is assumed that $\mathbf{u}$ is the unitary vector oriented from the current node to the neighboring node, e is the error of position between two nodes, $\dot{e}$ is the derivative of this error and $\int_0^t e(\tau) \cdot d\tau$ is the integral of this error. Both derivative and integral part will be estimated numerically respectively using the Euler's method and the rectangles method. We have therefore :

$$\mathbf{u} = \frac{\mathbf{p_c} - \mathbf{p_p}}{\|\mathbf{p_c} - \mathbf{p_p}\|} \qquad e(t) = \frac{\|\mathbf{p_c} - \mathbf{p_p}\| - l}{\|\mathbf{p_c} - \mathbf{p_p}\|}$$

  These two forces $\mathbf{f_p}$ and $\mathbf{f_n}$ can therefore be expressed using the expression of $\mathbf{f}$, taking care to take the correct current and previous element for each case.
- **Drag $\mathbf{d}$** : By noting A the cross section area, C_D the drag coefficient, ρ the density of fluid, and $\mathbf{v}$ the velocity of the node we have :

$$\mathbf{d} = -\frac{1}{2} \cdot \rho \cdot A \cdot C_D \cdot \|\mathbf{v}\| \cdot \mathbf{v}$$

3

Thus we have expressed the forces necessary for the simulation of the tether. FIGURE 2 shows us the modelization of the tether. We see the different TetherElements represented in blue. For clarity reasons, the different forces are represented only on one node but must be computed for each node. The speed of the node is noted $\mathbf{v}$. Finally, in order not to fully draw the tether, the representation has been deliberately cut after the first node and before the last node to represent only three central nodes.

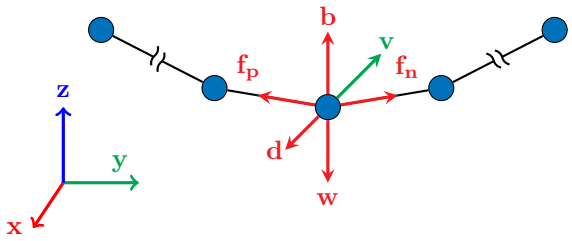


Fig. 2: Modelization of the problem

III. IMPLEMENTATION

The Python 3 implementation of this simulation is available in a GitHub repository[1]. This code is based on Numpy [10], Matplotlib [11] and Scipy [12] packages. The goal of this simulator is to study the viability of such a system, in particular to validate the performance of the tether with this behavioral model.

So we will create a class *TetherElement* which will represent a node. It will have to contain its mass, volume and distance information from its neighbors, but also its position, velocity and acceleration, as well as a pointer to each of its two neighbors. Finally it is necessary to have each coefficient K_p, K_d and K_i to compute $\overrightarrow{F_{t,p}}$ and $\overrightarrow{F_{t,n}}$.

This will allow us later on to implement a *Tether* class to be able to simulate a tether. This object must have a length, a number of elements and a list containing the different *TetherElement* that compose it. It must also have the mass and the volume of each node, but also the length of each link between nodes in order to correctly instantiate each *TetherElement*.

A diagram of these two classes is visible on the FIGURE 3. It respects the UML format and allows to see the different class variables and methods associated to each class.

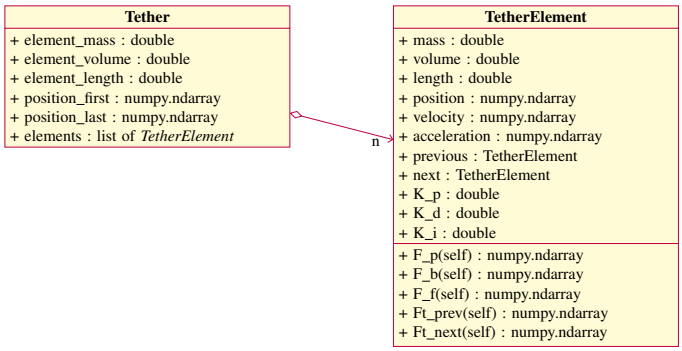


Fig. 3: UML diagram of the *Tether* and *TetherElement* classes

[1] Avalable at : https://github.com/Teusner/Tether_Modeling

IV. Initialization

Initialization is an important step because if the initial position of each TetherElement is random, the Tether will take a long time to converge and the system will be inconsistent. This is mainly due to the fact that the coefficients of the behavioral model are set to keep the nodes at a good distance from each other when a small perturbation is brought to the system.

To initialize the different nodes, we use the catenary equation [13] [14]. The idea is to use the shape taken by a rope attached at the ends to two fixed coordinate points. This rope will want to minimize its energy and so it takes this shape. This chain should check the following second order differential equation.

$$\ddot{z} = \frac{1}{k} \cdot \sqrt{1 + \dot{z}}$$

The solutions are known [13] [14] and of the form:

$$z(x) = k \cdot cosh\left(\frac{x}{k}\right)$$

However, this solution shows a rope centered around the ordinate axis. This is not necessarily a situation that we will find in our simulation. Here we would like to set the two fixed extremities, noted (x_1, y_1, z_1) and (x_n, y_n, z_n). By introducing c_1, c_2 and c_3 three coefficients allowing to correctly place the rope [14], we will then want to find here an equation of the form:

$$z(x) = c_1 \cdot cosh\left(\frac{x + c_2}{c_1}\right) + c_3$$

To find these coefficients, we have at our disposal three conditions: the two conditions related to the end points and the length of the string which must be equal to L. These conditions are expressed by the following equations [14]:

$$\begin{aligned} L =& c_1 \cdot sinh\left(\frac{x_n + c_2}{c_1}\right) - c_1 \cdot sinh\left(\frac{x_1 + c_2}{c_1}\right) \\ z_1 =& c_1 \cdot cosh\left(\frac{x_1 + c_2}{c_1}\right) + c_3 \\ z_n =& c_1 \cdot cosh\left(\frac{x_n + c_2}{c_1}\right) + c_3 \end{aligned}$$

It is possible to solve the system of equations numerically using the function *fsolve* of the package *scipy.optimize* [12]. Finally, from the calculated coefficients, and by knowing the length between the first node and the i^{th} node, it is possible to initialize the nodes by reusing the previous constraints but the unknowns become the position x_i and z_i. Note that the constraint on the position of the first node brings nothing to the system of equations, which leads to a system of two equations with two unknowns. The y_i of each TetherElement are linearly spaced between the two extremities.

V. Results

This section will present the results of the simulation. We then have the necessary tools to simulate the behavior of a tether. The Figure 4 shows the results of a tether simulation with a length $L = 25m$ with 11 TetherELements. Each node has a mass $m = 10\ kg$, and a volume $v = 1.10^{-3}\ m^3$. The coefficients have been set to $K_p = 350$, $K_d = 5.0$ and $K_i = 35.0$. To set these coefficients we need to build some tools to analyze the behavior of the tether to validate the modeling of the system.

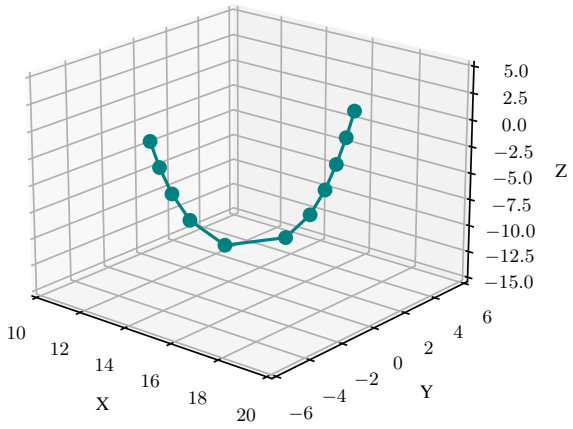


Fig. 4: Tether simulation

The different tools that will be presented in the rest of this section will allow analyzing the simulation launched with the previous parameters.

A. Length of different links

Monitoring the length of each link is important to verify the good performance of the simulation. The plot of these lengths will validate the behavioral force presented in this paper.

The Figure 5 shows in gray the plot of the length of each link. In crimson is plotted the average of these lengths and in yellow the target length. Finally, the blue area shows the 95 % confidence interval.

As we can see, we can have a coherent behavior a few seconds after initialization. On average the link lengths converge towards the set length and the standard deviation is not too high. However, there are still some oscillations and not all links are exactly at the target length due to the weight and buoyancy perturbations.

B. Relative error

Another way to interpret the link length error is to plot the error relative to the target length. This makes it possible to check the behavior of the system with tuned coefficients of the implemented behavioral model.

The Figure 6 shows us the average relative length error of the different links compared to the target length.

The adjustment of the coefficients of the behavioral model is done as follows. First, the K_p is set to obtain an oscillating

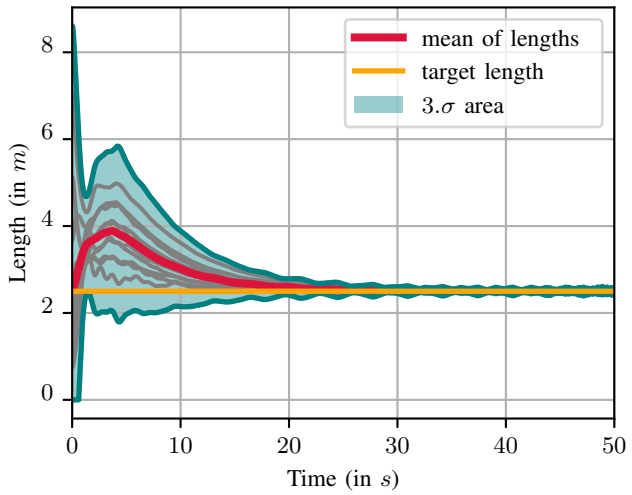


Fig. 5: Length of the links, average length and confidence interval

system with a bounded average link length. Then we add a derivative effect to reduce oscillations on the system by increasing the coefficient K_d. Finally, we add an integrator effect to remove a static error to make the length of the links reach the target length despite the presence of disturbances.

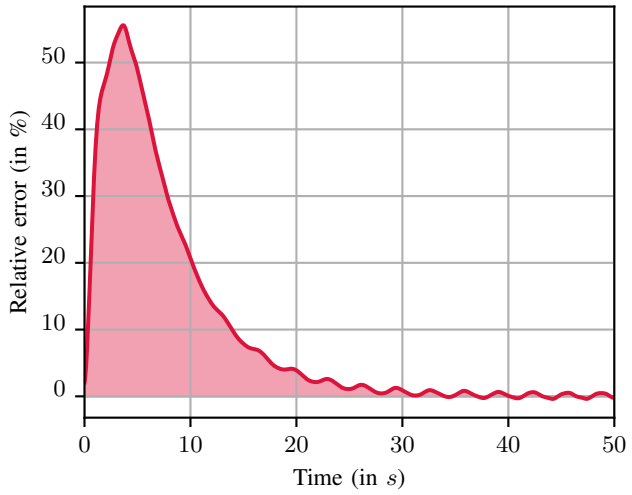


Fig. 6: Relative error between the target length and the mean length of links

C. Energetical approach

Finally, the energy approach is the most important to validate the simulation, because it determines whether the simulation makes physical sense. The system must not have divergent energy, which would be a physical counter-sense, but here, as there is no energy source, the overall energy must decrease over time, as soon as there is some fluid friction due to the drag force.

The mechanical energy of the system decomposes into the sum of two energies: kinetic energy and potential energy [15].

- **Kinetic Energy** : The global kinetic energy of the Tether is calculated simply by summing the kinetic energies of the different TetherElements.

$$E_k = \sum_{i=0}^{N} \left(\frac{1}{2} \cdot m \cdot v_i^2 \right)$$

Where v_i is the velocity of the i^{th} element.

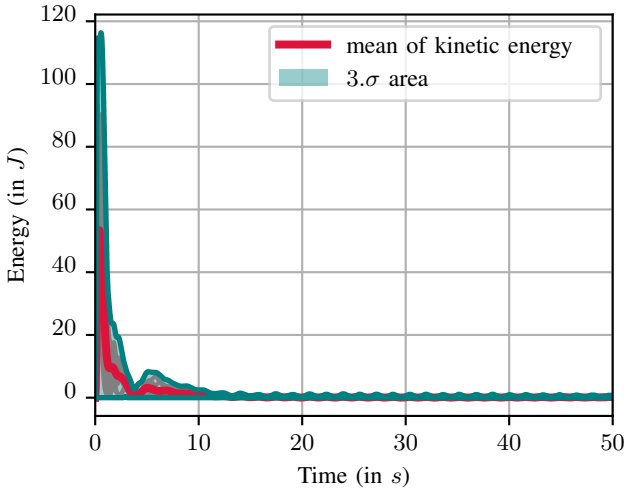


Fig. 7: Kinetic energy of the system

The FIGURE 7 presents the evolution of the kinetic energy of the system, within gray the kinetic energy of each node, in crimson the average of the kinetic energies, and the blue area represents the 95 % confidence interval. We can see that the kinetic energy decreases and cancels rapidly. This is explained by the fact that the Tether after a few seconds is correctly initialized and the nodes come to a standstill.

- **Potential Energy** : To calculate the potential energy related to the application of a force on a solid, it is necessary to use the definition of potential energy [15]. Indeed we know that by noting dt the time step of the simulation :

$$\delta W(\overrightarrow{F}) = \overrightarrow{F} \cdot \overrightarrow{dOM} = \overrightarrow{F} \cdot \overrightarrow{v} \cdot dt$$

$$E_p = \int_0^t \sum_{F_{ext}} \delta W(\overrightarrow{F_{ext}}) + cste$$

Thus we can calculate the potential energy by calculating the sum of the elementary work of each force on the system, i.e. the forces that apply to each node, and then by integrating this quantity over time. We then have an expression of the potential energy of the system over time for the Tether which is defined to within a constant, which is set to 0 in our case.
The FIGURE 8 shows us the evolution of potential energy over time. In gray is plotted the potential energy of each

node, in crimson is the average potential energy and the blue area represents the 95 % confidence interval. We see that the potential energy of the system does not diverge and that the system tends to position itself in a minimum of potential energy.

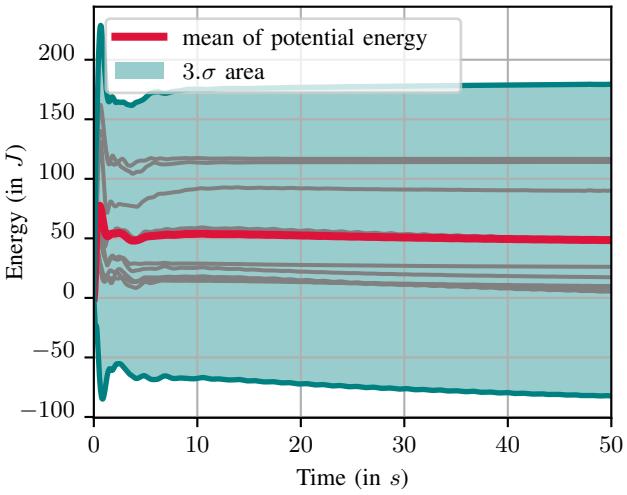


Fig. 8: Potential energy of the system over the time

- **Mechanical Energy** : Finally, by summing the kinetic and potential energies previously calculated, we are able to get the mechanical energy of the system [15]. This will give us information about the non-conservative forces that are included in this system. Indeed, the drag force will cause the system to lose energy since this energy will not be transformed into another form that can be used by the system, but will be dissipated as heat.
  The FIGURE 9 shows us the evolution of the mechanical energy over time. In gray is plotted the mechanical energy of each node, in crimson is the average mechanical energy and the blue area represents the 95 % confidence interval.

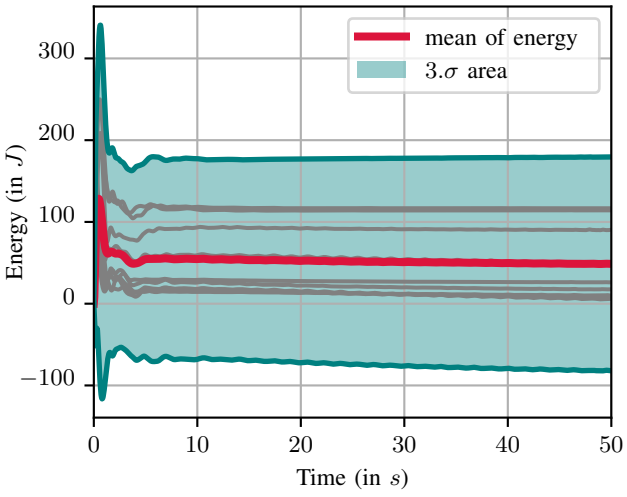


Fig. 9: Energy of the system

We can see that the energy of the system does not diverge, which seems to support the idea that the modeling is correct. Then, we notice that the system tends to minimize its energy, which is exactly the behavior expected for any physical system.

VI. CONCLUSION

In conclusion, the tether modeling as presented in this paper seems to give correct results. The idea of reasoning by finite differences allows us to simulate the tether as a succession of tether elements linked by variable-length links. The problem induced by this method is to provide a model to describe the force of a node on its neighbors to reach the target length. The proposed behavioral model provides good results here, and it leads to a simulator with a physical meaning.

Besides, there are still some issues that have not been addressed. First of all the tether was not tested in an environment where the extremities were subjected to movement. Second, the Tether is simulated with a fixed length, which is not necessarily the case during a submarine mission. Indeed it is common to have to unroll and rewind the Tether during the mission to prevent it from becoming tangled. Then, no force simulates the stiffness of the Tether, but it is not infinitely flexible. A solution to the two previous problems seems to be presented in [7]. Finally, there is no transmissible torque across the tether, which can typically be induced by a tether twist.

REFERENCES

[1] O. Blintsov, "Development of the mathematical modeling method for dynamics of the flexible tether as an element of the underwater complex," 2017.

[2] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011. [Online]. Available: https://books.google.fr/books?id=oR3sBgAAQBAJ

[3] L. Jaulin, *Mobile Robotics*. Wiley, 2019. [Online]. Available: https://books.google.fr/books?id=OMWxDwAAQBAJ

[4] J. Bender, M. Müller, M. A. Otaduy, M. Teschner, and M. Macklin, "A survey on position-based simulation methods in computer graphics," *Comput. Graph. Forum*, vol. 33, no. 6, p. 228–251, Sep. 2014. [Online]. Available: https://doi.org/10.1111/cgf.12346

[5] J. R. Ellis, "Modeling, Dynamics, and Control of Tethered Satellite Systems," Ph.D. dissertation, Virginia Polytechnic Institute and State University, The address of the publisher, 03 2010. [Online]. Available: http://scholar.lib.vt.edu/theses/available/etd-03182010-130812/

[6] R. Marshall, R. Jensen, and G. Wood, "A general newtonian simulation of an n-segment open chain model," *Journal of Biomechanics*, vol. 18, no. 5, pp. 359–367, 1985. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002192908590291X

[7] O. Ganoni, R. Mukundan, and R. Green, "Visually realistic graphical simulation of underwater cable," jan 2018.

[8] J. Koenemann, P. Williams, S. Sieberling, and M. Diehl, "Modeling of an airborne wind energy system with a flexible tether model for the optimization of landing trajectories **Support by the EU via ERC-HIGHWIND (259 166), ITN-TEMPO (607 957), and ITN-AWESCO (642 682) and by DFG in context of the Research Unit FOR 2401." *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11 944–11 950, Jul. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896317315227

[9] S. Prabhakar and B. Buckham, "Dynamics modeling and control of a variable length remotely operated vehicle tether," in *Proceedings of OCEANS 2005 MTS/IEEE*, 2005, pp. 1255–1262 Vol. 2.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[11] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[12] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[13] A. S. of Mechanical Engineers, *Applied Mechanics Reviews*. American Society of Mechanical Engineers, 1968, no. vol. 21. [Online]. Available: https://books.google.fr/books?id=00muEpyMcn0C

[14] W. Ren, M. Huang, and W. Hu, "A parabolic cable element for static analysis of cable structures," *Engineering Computations*, vol. 25, no. 4, pp. 366–384, May 2008. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/02644400810874967/full/html

[15] J. Viegas, *Kinetic and Potential Energy: Understanding Changes Within Physical Systems*, ser. The Library of Physics. Rosen Publishing Group, 2004. [Online]. Available: https://books.google.fr/books?id=q1JHaokEC2QC

Mise en place d'une station de référence de transmission de corrections RTK: Exemple des GPS ublox

Mots-clés : GPS, RTK, Géolocalisation, RTCM, Réseau

Resumé :

En robotique et dans d'autres domaines de l'ingénierie, le GPS est un l'un des capteurs les plus utilisé. Il sert en général à donner la position d'un objet (un robot par exemple) avec une précision de l'ordre du mètre. Dans certaines utilisations du GPS une précision centimétrique peut être recherché. Un moyen d'atteindre cela est d'envoyer des corrections RTK au GPS qui localise notre objet. En fonction de la ville dans laquelle on se trouve, il peut y avoir une station de référence, non loin de là où on utilise le GPS, pour transmettre ces corrections. Mais dans certains endroits, on ne reçoit pas bien ces corrections (car trop éloigné de la station de référence), ou encore elles sont mêmes payantes. Ce article permet de mettre en place notre propre station de référence avec un GPS, puis de les envoyer un autre GPS grâce à un code python.

Introduction:

Comment décrire la position d'un objet sur la Terre ? C'est à cette question que la géolocalisation répond. Donc la géolocalisation est un ensemble de procédés permettant de localiser en 3D un objet, une personne, un véhicule sur la terre à l'aide de coordonnées géographiques. Ce procédé s'appuie principalement sur les systèmes de de positionnement par satellites. Ces applications sont nombreuses : La navigation, le guidage, la géophysique (carte bathymétrique). Pour chaque application les précisions attendues ne sont pas les mêmes. Par exemple pour le guidage, la précision attendue est de l'ordre 5cm ou encore pour la génération des cartes bathymétriques les précisions de la localisation doivent être de l'ordre 5mm. Pour des positionnements standards c'est-a-dire l'utilisation simple de gps sans amélioration de la position, la précision varie entre 5m et 15m. Ce qui est un frein aux applications de guidage et de géophysique. Pour améliorer la précision d'autres méthodes de positionnement existent telles que DGNSS (Differential GNSS), PPP (Precise Point Positionning), RTK (Real Time Kinematic), PPK (Post Processed Kinematic).

Contexte: Pendant le projet magmap dont mon rôle était la prise en main d'un robot terrestre appelé saturne, il nous a été demandé par le groupe localisation de la luge d'enregistrer des données de positions de la luge et du robot afin qu'il valide leur modèle. En plus il nous a été demandé d'envoyer des corrections RTK aux gps se trouvant sur la luge et sur le robot. Afin de répondre à leur problématique, nous avons voulu créer notre propre base qui enverra les corrections RTK aux différents GPS. Même si finalement le groupe Localisation de la luge n'a plus utilisé les corrections RTK pour améliorer la précison des gps sur le robot, Je tenais quant à présenter ce travail pour plusieurs.

-Tout d'abord je pense que le travail réalisé pourra servir aux prochaines promotions de la filière robotique de bénéficier de moyens de mettre en place une base RTK.

-Tout le travail effectué a été testé puis validé

-Enfin le travail que j'ai réaliser sur mon sujet d'initiation à la recherche (réglage des Coefficients du PID par le bias de l'analyse par Intervalle) n'a pas atteint l'objectif escompté. Car mon sujet était étroitement lié au cours de Commande Robuste a due être arrêter, je n'ai pas pu terminer le travail comme je le voulait. Même si j'ai réussi à implémenter la solution de l'article, mais je n'ai pu intégrer les nouvelles fonctionnalités pour améliorer l'algorithme.

Dans les lignes qui suivent, je vais présenter dans un temps comment fonctionne le positionnement par GNSS, ensuite comment configurer un GPS pour qu'il se comporte comme une base de référence pour transmettre les corrections, en outre comment envoyer ces corrections et les appliquer au GPS récepteur et enfin les résultats obtenus.

I- **Positionnement par GNSS : comment ça fonctionne ?**

Le positionnement par GNSS(Global Navigation Satellite System) est basé sur l'émission des signaux de satellite en orbite autour de la terre et fournissant une couverture mondiale. C'est au milieu des années 90 que le GNSS apparaissait. Il s'agit du GPS concu et développé par le département de défense américaine. Depuis 2007 d'autres systèmes de GNSS sont apparus tels que Galileo (UE), Beidou (Chine) ou encore Glonass (Russie). Donc l'objectif de ces systèmes de positionnement est de fournir à un récepteur sa position, sa vitesse de déplacement et l'heure.

Deux choix de positionnement s'offre à nous : Le positionnement absolu et le positionnement relatif. En absolu, le positionnement se fait directement par rapport au satellite. Ce qui permet d'avoir un positionnement autonome. Mais l'inconvénient est que ce positionnement n'est pas précis. En relatif, le positionnement est plus précis mais requiert une station de référence proche pour corriger les erreurs de positions.

Comment obtenir une station référence afin d'avoir un positionnement relatif précis ?

II- **Configuraton de station de reference pour la transmission de corrections**

Comme mentionner précédemment le positionnement en absolu n'est pas précis à cause des erreurs de modélisations. Pour corriger ces erreurs, une station de référence se charge d'envoyer des corrections au format RTCM (Radio Technical Commision for Maritime Services) au récepteur GPS (Ici des corrections RTK). Durant ce projet nous avons à dispositions des GPS ublox. Le constructeur de ces GPS met à notre disposition un logiciel (U-center) pour la configuration d'un GPS en station de référence et des autres GPS en Récepteurs de corrections RTK. Les étapes pour la configuration de la base GPS sont :

1- Brancher le GPS par USB sur la machine sur laquelle se trouve le logiciel U-center.

2- Ouvrir U-center, sélectionner le port correspondant au GPS et sélectionner du baudrate (regarder le champ Receiver).

3- Indiquer su u-center que l'on travaille avec des trames NMEA haute precisions

4- Dans Receiver->configuration View, activer les messages RTCM

5- Enter les coordonnées de la station de référence. Lorsqu'on dispose d'une position précise (par exemple une position pourrait être donnée par les élèves de la filière hydrographie) , entrer ces coordonnées ou selectionner le mode Survey-in (qui permet de calculer automatiquement la position de la station de reference) si cela n'est pas le cas.

Durant le projet j'ai récupéré avec les hydros les coordonnées de quelques points situés près du terrain de foot. Le point pouvant etre pris comme été situé :

-Lattitude=48°25' 05.06174'' N

-Longitude=04°28' 23.60489'' O

-Hauteur=139.622

	
Picture of marker	Picture of site

.

6- On enregistre le travail effectué

Il faut également configurer le GPS recepteur des corrections RTK. La méthode est presque la même que celui fait précédemment. Après l'étape 3, au configure les ports d'entré/sortie puis on enregistre. Suivre ce tuto lorsqu'on voudra configurer nos GPS [1].

Maintenant que les des GPS ont été configurer en station de référence et récepteur de corrections RTK, comment envoyer les trames RTCM de corrections vers le ou les récepteurs GPS.

III- Transmissions des Corrections RTK vers les GPS.

Afin de transmettre les corrections RTK, Nous a besoin de faire communiquer la station de référence et les autres récepteurs GPS. Il faut donc relier tous les GPS au même réseau (celui du robot par exemple) que celui de la station de contrôle en leur donnant une adresse IP. Le problème est que Les GPS à notre disposition ont des ports USB et non des ports Ethernets. Donc impossible de les relier directement au réseau. Pour la station de référence, if faut simplement brancher le GPS (en USB) à un PC qui doit être connecter au réseau du robot. Pour les GPS récepteurs RTK, On peut les brancher directement sur le PC embarqué du robot. Si on n'a pas assez de ports USB sur le PC embarqué, on peut utiliser un adaptateur USB-Ethernet et brancher directement le GPS sur le switch.

Dans le cadre du projet, nous avons donc deux GPS qui doivent recevoir des corrections RTK : un sur la luge l'autre placé sur le Robot. Le code développé pour transmettre les corrections RTCM aux récepteurs est basé sur le protocole TCP/IP:

-La station de référence se comporte comme un serveur qui publie des corrections RTK

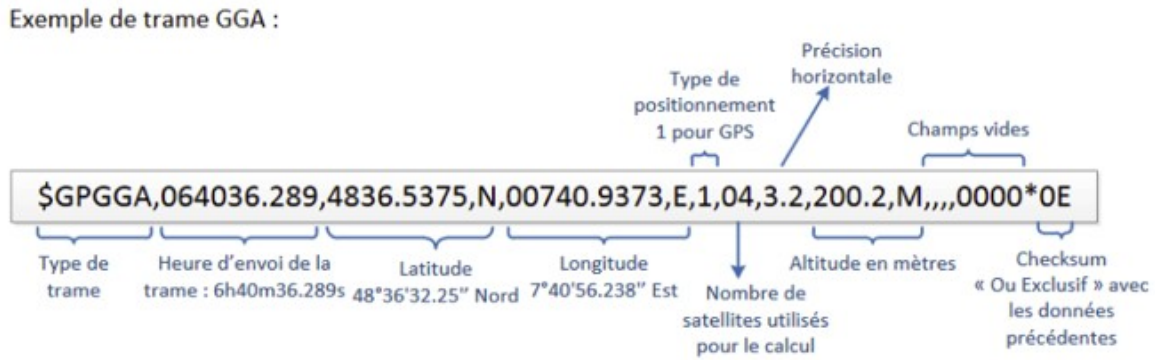
-hôte= 0.0.0.0 on publie en broadcast sur le port 12800. Donc n'importe quel PC sur le réseau pourra recevoir ces corrections et les écrire sur le GPS connecté sur son port USB ou directement au réseau.

-Le code *serveur2-2.py* permet de publier les corrections sur deux GPS uniquement, mais il pourrait être adapter pour 1 et plusieurs GPS.

-Le code *client1.py* permet de pour recevoir les corrections RTK et les appliqués aux GPS récepteurs.

IV- **Résultats**

Pour vérifier si les différents GPS recevaient les corrections RTK, nous avons analyser les trames NMEA de chaque GPS.



On a observé que au niveau du *type de positionnement*, on obtenait une valeur de 4. Ce qui signifie que le GPS reçoit effectivement des corrections RTK.

Conclusion:Beaucoup d'applications utilisent le GPS pour déterminer une position. Une application standard du GPS ne nous donne pas une localisation satisfaisante dans certains cas. Utiliser des corrections RTK s'impose donc. Or Ces corrections ne sont pas toujours facile à obtenir. Cet article a permit de mettre en place une base RTK pour transmettre des corrections.

Référence:

[1] https://www.ensta-bretagne.fr/lebars/Share/uCenter_RTK_guide.pdf

serveur2-2.py

```python
#!/usr/bin/env python
import time
import serial
import socket

hote='0.0.0.0'
port=12800
base_gps=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
base_gps.bind((hote,port))
base_gps.listen(5)
print("le serveur ecoute sur le port {}".format(port))
rover_gps, _ =base_gps.accept()
rover_gps2, _ = base_gps.accept()

ser = serial.Serial(
 port='/dev/ttyACM0',
 baudrate = 9600,
 parity=serial.PARITY_NONE,
 stopbits=serial.STOPBITS_ONE,
 bytesize=serial.EIGHTBITS,
 timeout=1
)

while True:
    msg=ser.read(100)
    v = rover_gps.send(msg)
    v2 = rover_gps2.send(msg)
    print("taille envoye = {} , \t {}, \t {}".format(len(msg), v, v2))

print("fermeture connexion")
rover_gps.close()
rover_gps2.close()
base_gps.close()
```

File Edit View Help client-1.py - ...\Local\Temp

serveur2-2.py client-1.py

```
#!/usr/bin/env python
import time
import serial
import socket

hote="192.168.9.199"
port=12800
gps_rover=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
gps_rover.connect((hote,port))
print("connexion etabli avec le port{}".format(port))



ser = serial.Serial(
 port='/dev/ttyACM0',
 baudrate = 9600,
 parity=serial.PARITY_NONE,
 stopbits=serial.STOPBITS_ONE,
 bytesize=serial.EIGHTBITS,
 timeout=1
)


msg_recv=b""
while True:
    msg_recv=gps_rover.recv(1024)
    v=ser.write(msg_recv)
    print("taille envoye = {} , \t {} ".format(len(msg_recv), v))
    #print(msg_recv.decode())

print("Fermeture de la connexion")
gps_rover.close()
```

Simultaneous Localization and Mapping with ROS in an inside enviremnent

EVAIN Alexandre
FISE 2021 ROB - ENSTA Bretagne

In this work, we studied and compared different existing Simultaneous Localization and Mapping (SLAM) algorithms in the context of interior mapping in a Gazebo simulation using the Robot Operating System (ROS). We used a Turtlebot3 Burger robot, and tested Gmapping, Hector-SLAM and KartoSLAM. This report focus first on a short theoretical explanation of of how each SLAM algorithm works, then in the second part on the simulation itself and its results. After several simulations in the same conditions, we can see the advantages and drawbacks of each solution, in order to see which one is the most adapted to our case.

***Index Terms*—ROS, SLAM, Cartography, Mapping**

I. Introduction

SIMULTANEOUS Localization and Mapping (SLAM) presents the double challenge of doing a map of an unknown environment while at the same time trying to find the position of the robot within this environement. Because of the complexity of this challenge, several SLAM algorithms were created using different methods and algorithms.

ROS (Robot Operating System) is a software development platform for robot which provides tools used to create, run and distribute ROS-based softwares, clients for C++ and python languages (roscpp and rospy) and packages containing programs for ROS using one or more ROS clients.

Therefore, we will use our simulation on the ROS environement, and we will test three different SLAM algorithms: Gmapping, Hector-SLAM and KartoSLAM. All these algorithms and packages used in this report are all based on the ROS environement.

A. Mapping Challenges

The first challenge that we face are the sensor errors which are not randomly distributed, but instead cumulative. While it wouldn't be a problem in the case of short movements, because the sensors have limitated ranges, robotic mapping requires that the robot moves through its environment to map it completely; resulting in important cumulative errors. In addition to sensor errors, we also face motion errors, meaning that we cannot merely use the command sent to the robot to deduce its position.

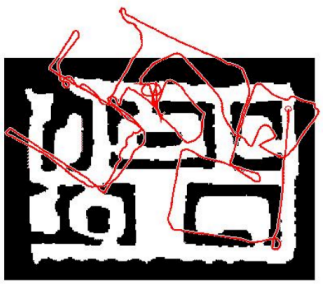


Fig. 1. Cumulative odometric errors, resulting in an unreliable path[1]

B. LIDAR and Occupancy Grid Map

LIDAR (Laser Imaging, Detection, And Ranging) is a method to get a distance between an object and a sensor by emiting a laser beam and measuring the time of reflection of the beam (given that the light velocity is well known).

An Occupancy Grid Map is a map of the environment as an field of number each representing the presence or absence of an obstacle at that location in the environment. When doing measures, the laser provides binary values for each coordinate: $M_{x,y} = 1$ if the laser hit an object, 0 in the other case. Using this data, the different SLAM algorithms that we will use will produce an Occupancy Grid Map. In order to remain able to correct the measurement, the data on the occupancy grid map are not values but probabilities, allowing the algorithm to correct both the laser measurement imprecisions and the data noise.

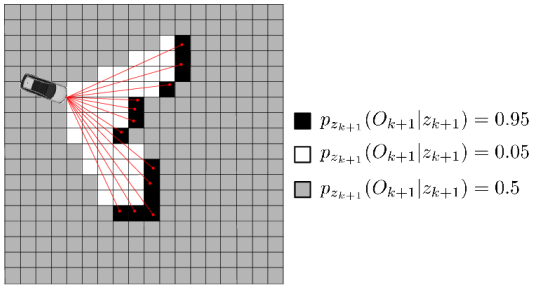


Fig. 2. An example of an Occupancy Grid Map, with the probabilities associated to each point. We can notice that even in the case of a wall, the probability remains under 1, allowing further adjustments if needed (such as following a change of the environment)[2].

II. Previous Work

Comparisons of the different SLAM algorithms already exists, some more complete and more accurate than this one. Notably, [2], [3] and [1] are all existing reports involving the SLAM algorithms used here. Most importantly, the reports focusing only on a single algorithm are usually the most complete (some were even written by the algorithm's creator

themselves) available, and provides much more detailled theorical explanation of the algorithms functions. [4] is focused on Hector-Slam and was written by its designers, [5] is focused on KartoSLAM and involved the company that made it and finally [6] focuses on the Gmapping algorithm.

III. GMAPPING

A. Overview

Gmapping is an algorithm created by Brian Gerkey requiring a robot able to provide odometry data and equipped with a horizontally-mounted, fixed, laser range-finder. It is based on the Rao-Blackwellized particle filter (RBPF). This particle filter was then modified in Gmapping to take into account the previous odometry data and the newest observation in order to get more accurate data estimations.

B. The RBPF approach

Particles filters operate on the same main method: each particle is considered as a landmark. After a movement of the robot, several particle matching hypothesis are made: the ones with the highest weight are the most likely to be exact and are therefore guarded, while the ones with low weights are unlikely to be accurate and are discarded.

The Rao-Blackwellized method allows to compute the probability of each hypothesus with the equation:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}-1) = p(m \mid x_{1:t}, z_{1:t}).p(x_{1:t} \mid z_{1:t}, u_{1:t}-1) \quad (1)$$

with $x_{1:t}$ the list of the position of the robot, $z_{1:t}$ the map data and $u_{1:t}$ the odometric data. [2]

It should be also noticed that Gmapping is able to function without odometric data by relying fully on the scan matching algorithm; however its accuracy is decreased in such a situation.

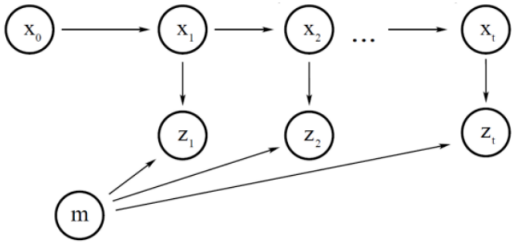


Fig. 3.
Without odometric data, the Gmapping algorithm follows these steps[6]:
1) Initial state guess. The pose is obtained from the previous pose with measurement zt-1.
2) Scan-matching algorithm obtains map mt-1 from initial state guess.
3)Updates of particles.

IV. HECTOR-SLAM

A. Overview

Hector SLAM is an algorithm created by Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer and Uwe Klingauf from the Technische Universitat Darmstadt[4]. This algorithm is different from the other SLAM methods tested because it does not require odometric data, however it does require accurare LIDAR data. The operation of this algorithm is quite simple to understand: at the initial time, the LIDAR scan is saved as the initial Occupancy Grid Map (OGM). At the next instant, the robot moves, the new LIDAR data is matched and aligned with the previous map in order to obtain an estimate of the change in position. From this estimate, the algorithm sends back this estimated position to the robot controller, and updates the map.

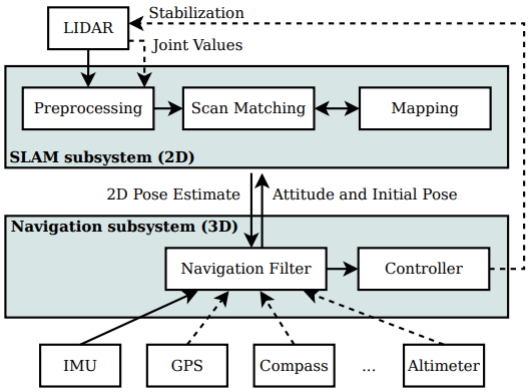


Fig. 4.
Hector mapping and navigation system[4]

B. Scan matching: the Gauss-Newton approach

The process of aligning the new LIDAR scans with the existing map is called Scan Matching. Hector-SLAM uses a Gauss-Newton approach to do this matching, allowing Hector-SLAM to bypass the robot pose search. When the scans are aligned on the map, the robot position is deduced from the scan matching. All the mathematical expressions presented in this section comes directly from [4].

The goal of the scan matching is to minimise the differences between the laser scan and the map, in order to find the best alignment between them. Mathematically, this amounts to finding the transformation $\xi = (px, py, \psi)^T$ that minimises:

$$\xi^* = argmin \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2 \quad (2)$$

WIth $S_i(\xi)$ the world coordinates of scan endpoint and $M(S_i(\xi))$ the resulting map value.

Searching this minimum transformation is equivalent to searching the limit $\sum_{i=1}^{n}[1 - M(S_i(\xi + \Delta\xi))]^2 \rightarrow 0$

After using a first order Taylor expansion to $M(Si(\xi+\Delta\xi))$ and setting to zero the partial derivative with respect to $\Delta\xi$ we get:

$$\Delta\xi = H^{-1} \sum_{i=1}^{n} \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]^T [1 - M(S_i(\xi))] \quad (3)$$

with

$$H = \left[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}\right]^T \left[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}\right] \quad (4)$$

This approach do not use any kind of loop closure in order to keep computational requirement low, the authors considering that their algorithm is sufficiently accurate in small scale scenarios to avoid it[4].

V. KartoSLAM

A. Overview

KartoSLAM is an algorithm developed by SRI International's Karto Robotics which uses a Sparse Pose Adjustment (SPA) algorithm[3]. This algorithm uses the Levenberg-Marquardt (LM) method as a base framework, and the direct sparse Cholesky decomposition to solve the linear system in order to do the scan matching. Unlike the other algorithm previously used, KartoSLAM uses pose graphs, which are sets of robot poses connected by constraints obtained from observations of features (such as the lidar data).The advantages of this method according to its authors are that it possesses a very fast convergence, it takes covariance information into account and it is very robust with low failures rates.[5]

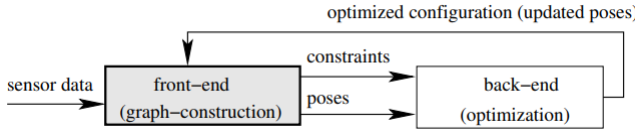


Fig. 5.
A usual graph-based slam system[5]

B. The SPA method

The variables of the system are the set of global poses c of the robot, parameterized by a translation and angle: $c = [t_i, \theta_i] = [x_i, y_i, \theta_i]^T$

The error e is expressed by: $e_{ij} \equiv \bar{z}_{ij} - h(c_i, c_j)$ with $h(c_i, c_j) = \begin{cases} R_i^T(t_j - t_i) \\ \theta_j - \theta_i \end{cases}$ [5]

Algorithm 1 ContinuableLM(c, e, λ), continuable Levenberg-Marquardt algorithm [5]

Input: nodes c and constraints e, and diagonal increment λ
Output: updated c
If λ = 0, set λ to the stored value from the previous run
Set up the sparse H matrix using CreateSparse(e, c–c0), with c0 as the fixed pose.
Solve $(H+\lambda diagH)\Delta x = J^T \Lambda e$, using sparse Cholesky with an approximate minimal degree (AMD) algorithm.
Update the variables c – c0 using equation: $t_i = t_i + \Delta t_i$ and $\theta_i = \theta_i + \Delta\theta_i$
If the error e has decreased, divide λ by two and save, and return the updated poses for c – c0
If the error e has increased, multiply λ by two and save, and return the original poses for c – c0

VI. Methodology

A. The Equipement

During this experiment, we used a TurtleBot3 robot, developped by the Open Source Robotics Foundation (OSRF) and ROBOTIS. It is equiped with two motors, a 1,800mAh battery pack, a LDS-01 360 degree LIDAR, a camera and a single board computer (Raspberry PI 3). In our case, we used the model "Burger", chosen because its small size makes it easier to handle and control.

TABLE I
LIDAR SPECIFICATIONS[7]

Distance Range	120 ~ 3,500mm
Distance Accuracy (120mm ~ 499mm)	±15mm
Distance Accuracy (500mm ~ 3,500mm)	±5.0%
Distance Precision (120mm ~ 499mm)	±10mm
Distance Precision (500mm ~ 3,500mm)	±3.5%
Scan Rate	300±10 rpm
Angular Range	360°
Angular Resolution	1°

B. Simulation

In order to compare the different SLAM algorithms in the absence of real robots or any kind of LIDAR equipement, a simulation was the only available choice. This simulation is done in Gazebo, and present a real scale house, included in the Turtlebot3 package. While other worlds were available, this one was chosen because it was the closest case to the real use of the robot in real conditions.

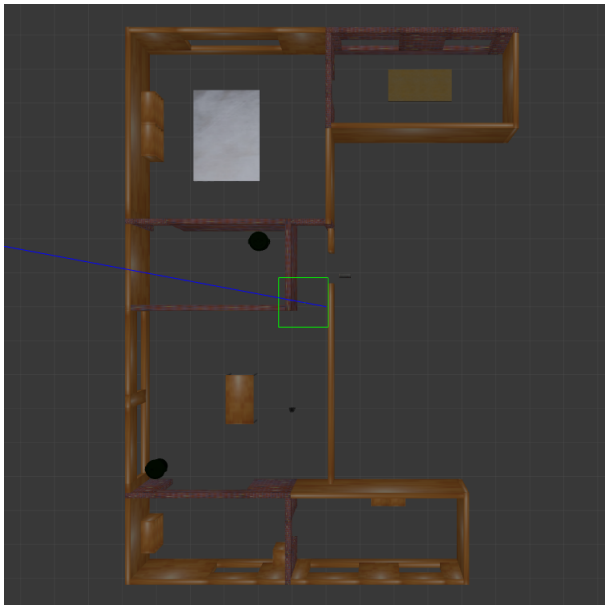


Fig. 6.
The house used during the simulations

The house is continuous and is filled with furniture, many obstacles that we might encounter in real life situations. Because the LIDAR is elevated on the Turtlebot3, it is too high to detect the bottom of the different shelves, however their walls remain detectable. Although the door of the house

is open, the exterior will be left unexplored, the focus of this report beeing interior mapping.

The Turtlebot package comes with an automatic controller for mapping surveys. However, the behavior of this controller was too sensitive to obstacle avoidance, and struggled to move from one room to another. As a result, the robot was manually controlled in all simulations and followed the same path.

VII. Results

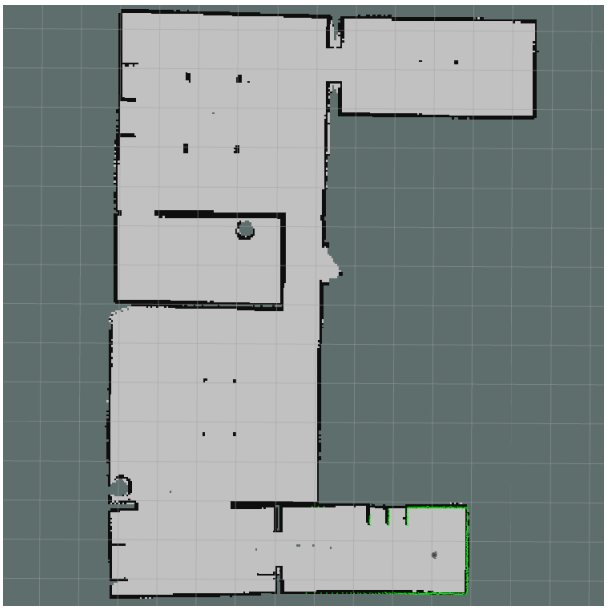


Fig. 7. The map obtained after using Gmapping

The first test was done with Gmapping. As we can see on the Figure above, while Gmapping allowed to make a map and individually conserve the shape of each room, globally the angle errors are clearly visible, and the resulting map is distorded. However, the loop closing was working, and the algorithm was able to know when the robot was going into room previously explorated. Given that it is a small scale building, the algorithm do not seems suited for larger scale buildings, the angle errors distording the building's shape too much

At first the Hector SLAM algorithm seemed much more promising than Gmapping. The room's shapes and angle were conserved with minimal distortion. However, despite such a good start, this algorithm do not handle well sudden change of angle, and everytime the robot made a sharp turn, the scan matching failed completely, resulting in the new map overlaping on the old in an unsatisfactory way. While it might be possible to use Hector SLAM with slower movements, in our simulation it failed several times, and ultimately do not seems suited for the simulation.

Of all the algorithms used, the KartoSLAM algorithm is the one that gave the best results. It handled perfectly the loop closure and the shape convervation, and the angle errors are too small to be seen by the bare eye. It was able to remain precise even with fast robot movements, and so far I have experienced no drawback when using it in simulations.

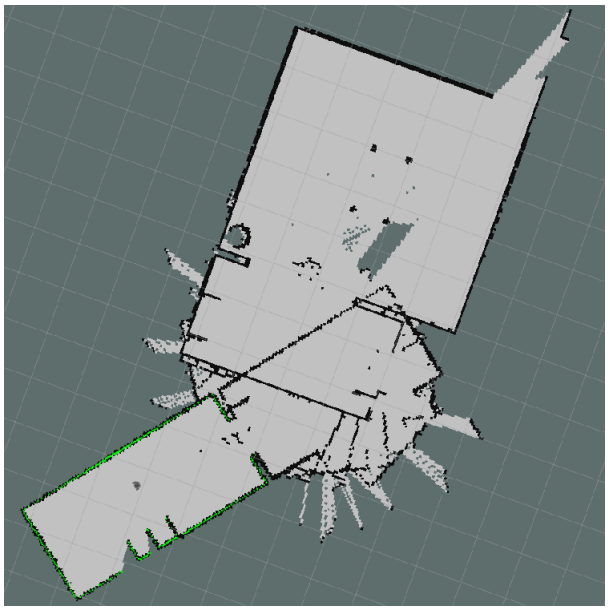


Fig. 8. Results obtained when using Hector SLAM

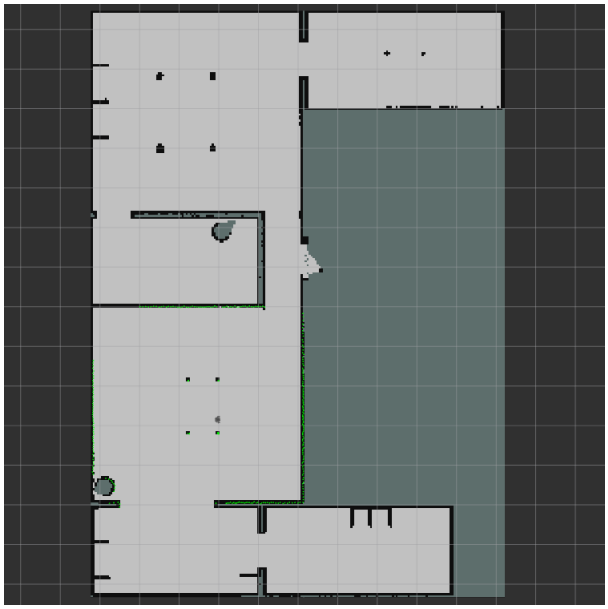


Fig. 9. Results obtained when using Karto

VIII. Conclusions

Of all the algorithm tested in this simulation, KartoSLAM was the one who gave reliably the best results and the map closest to the exterior environement. Hector-SLAM was too prone to failure at each robot rotation while Gmapping cannot compensate cummulative angle errors, resulting in distorded

TABLE II
Summary of the results

SLAM method	Main approach	Full map	Angle accuracy	Selected me
Gmapping	Particle Filters	Yes	No	No
Hector SLAM	Gauss-Newton	No	Yes	No
Karto	Sparse Pose Adjustment	Yes	Yes	Yes

maps of the environement. However; it should be noted that there are other SLAM algorithms that were not tested, and that this test was always done in the same conditions. It is very likely that under different circumstances Hector-SLAM is able to provide much better results; however I remain doubtfull of the ability of gmapping to solve the cumulative odometric errors in its current state.

In addition, all the results were only done through a simulated environement, and from these we cannot give an evaluation of the actual behavior of these mapping algorithm in real situations. From previous experience, it is unlikely that the SLAM algorithms behaves exactly in the same way in reality and in the simulation.

Finally, not all comparison criterias were evaluated in these simulations, and parameters such as GPU usage, the algorithm's speed or numerical accuracy were not compared due to the impossibility of conducting real life experiments.

ACKNOWLEGMENT

All the theorical knowledge was obtained through research and comes from the references bellow.

REFERENCES

[1] S. Thrun, "Robotic mapping: A survey," http://robots.stanford.edu/papers/thrun.mapping-tr.pdf, February 2002.

[2] K. Kamarudin, S. M. Mamduh, A. Y. M. Shakaff, and A. Zakaria, "Performance analysis of the microsoft kinect sensor for 2d simultaneous localization and mapping (slam) techniques," https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4299068/, December 2014.

[3] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2d slam techniques available in robot operating system," https://core.ac.uk/download/pdf/29175747.pdf, June 2013.

[4] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," https://www.researchgate.net/publication/228852006_A_flexible_and_scalable_SLAM_system_with_full_3D_motion_estimation, November 2011.

[5] K. Konolige, G. Grisetti, R. Kummerle, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," http://ais.informatik.uni-freiburg.de/publications/papers/konolige10iros.pdf, 2010.

[6] J. Esenkanova, H. O. Ilhan, and S. Yavuz, "Pre-mapping system with single laser sensor based on gmapping algorithm," https://www.researchgate.net/publication/274466838_Pre-Mapping_System_with_Single_Laser_Sensor_Based_on_Gmapping_Algorithm, January 2013.

[7] "Robotis e-manual - turtlebot 3," https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/.

Word Embedding and Semantic Networks for vocabulary representation in Natural Language Processing (NLP)

Romane Fléchard, *ENSTA Bretagne Student*

***Abstract*—This article presents the work carried out in the framework of a research initiation project on the theme of robotics and artificial intelligence. The subject chosen is focused on the study of the different technical solutions for the representation of the meaning of words for a computer. This paper firstly presents the different existing technical solutions such as semantic networks and Word Embedding algorithms, as well as their concrete applications in the field of natural language processing. Then, the Word2Vec and GloVe models, which are two Word Embedding learning algorithms, have been tested and their performances compared with each other, but also with data from the literature. The methodology followed as well as the results obtained are therefore also presented in this article. The different Word Embedding algorithms learned were visualised in the Tensor Flow Projector and the performance of the algorithms measured using the functions of the Python gensim library. The results show that the GloVe algorithm appears to be more efficient than the Word2Vec algorithm, and that the size of the learning database plays a crucial role in the performance obtained. The work carried out has therefore led to rather satisfactory results, in line with the information from the literature, although time and means were lacking to be able to carry out a truly exhaustive study of the different word representation solutions, as well as of the impact of the different learning parameters on the performance obtained.**

***Index Terms*—Artificial intelligence, Machine Learning algorithms, Word Embedding, Natural Language Processing, Semantic Networks**

I. Introduction

NATURAL Language Processing (NLP) is a field at the crossroads of computer science, artificial intelligence and linguistics, with multiple applications, particularly in the field of robotics. Indeed, it is now possible, for example, to create robots capable of understanding human language in order to carry out various tasks based on the words spoken by the user. Chatbots, automatic translation or correction systems, automatic document classification or conversational agents are all applications that are also representative of the usefulness of natural language processing. However, understanding the meaning of words is a very special capacity of the human brain that cannot be provided by a computer. Indeed, a computer only processes encrypted data, and the words used in computer programs are in fact only a sequence of characters, themselves digitally coded thanks to the ASCII computer standard (American Standard Code for Information Interchange). For any conventional computer program, words are therefore just a series of numbers without any meaning. However, in order to create robots capable of understanding human language, it seems obvious that this is not enough. How can we make a computer understand the meaning of words and their semantics? The aim of this article is therefore to look at the various existing technical solutions that make it possible to represent words and their semantics, so that they can be used in the various applications of natural language processing. In this article, a first part will be dedicated to the study and explanation of existing solutions. Semantic networks and various Word Embedding algorithms will be presented and explained, as well as the results of studies comparing these different methods. In a second part, the work carried out on the subject will be presented, i.e. the implementation of the most efficient Word Embedding algorithms according to the literature review carried out previously. The implementation of comparison tools between these different algorithms will also be detailed. Finally, a third part will be devoted to the results obtained. The performances of the different algorithms will be compared with each other, but also with the values obtained in the articles of the literature presented in the first part, and ideas for improvement will be proposed.

II. Literature Review

In this part, different technical solutions for the representation of words and their meanings will be presented and explained, based on information from the literature.

A semantic network is an oriented and labelled graph to represent knowledge. This method of knowledge representation was first used in cognitive psychology, because this model corresponds to the organisation of the semantic memory of the human brain, i.e. the memory of words, concepts and general knowledge about the world. This model makes it possible to represent both the organisation of knowledge in memory, but also the paths to access the different concepts. Following the first models proposed for the representation of knowledge in cognitive psychology [1], semantic networks were quickly used in the field of artificial intelligence because they structure data in such a way that they can be easily represented by computers. To better understand the representation of knowledge in a semantic network, an example is given Fig. 1:

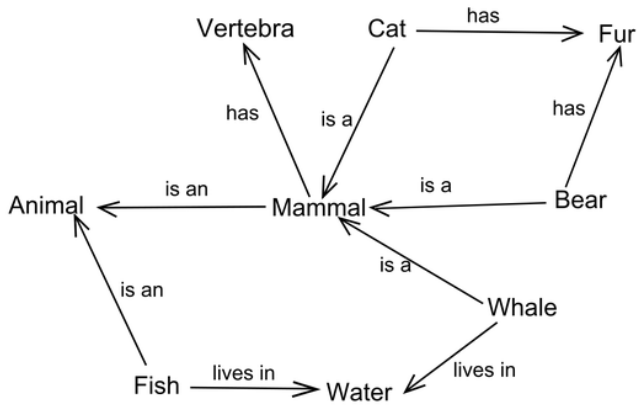


Fig. 1: Example of a reduced semantic network [2]

A semantic network is composed of nodes representing concepts, linked by arrows representing the links between these concepts. These links are labelled in order to define the relationship between two concepts. These relationships are supposed to be universal [3], they are a kind of semantic primitives that allow to link many concepts together.

One of the most elaborate semantic networks so far is WordNet. It was developed by the Princeton cognitive science laboratory. The database is available free of charge and can be used via different programming languages [4]. It is also possible to create one's own application-specific database and create one's own semantic network, for example using the semantic-net Python library. As mentioned above, semantic networks are very efficient for modelling the functioning of memory, the storage of information and the access paths to retrieve this data in memory. The other main advantages [5] of semantic networks are their ease of implementation and use. They are also easy to understand because they represent knowledge in the most natural and logical way possible. However, their definition is not very objective and depends on the creator. For example, there is no standard definition for the labels of the different links [5]. Moreover, some concepts such as verbs are less well represented than nouns, and only binary and unidirectional links between two concepts may, in some cases, be too constraining and constitute the limits of the model.

Another more standardised way of representing words for natural language processing is called Word Embedding. Word Embedding refers to the set of techniques for representing words as vectors of real numbers, so that semantic information can be transformed into numerical data that can be processed by a computer. Word Embedding is based on the linguistic theory founded by Zelling Harris [6], which states that the meaning of a word is directly linked to its context in the sentence or text in which it is found. In other words, two words that are relatively close together in a text are more likely to be semantically close than two words that are very far apart in the text evoking two completely different concepts. The idea of Word Embedding is to represent each word by an N-dimensional vector, so that two semantically close words are represented by two spatially close vectors.

In order to measure the distance between two vectors (which corresponds to the rate of similarity between the two corresponding words), a metric called cosine similarity is defined [7].

$$cosine\,(\vec{v},\ \vec{w}) = \frac{\vec{v} \bullet \vec{w}}{|\vec{v}|.|\vec{w}|}$$

The score thus calculated between two vectors indicates whether the words are very similar (score close to 1) or very different (score close to 0). To better understand the meaning of vectors, here is a simplified example of Word Embedding :

	Small	Gray	Orange
Cat	4	3	1
Dog	2	2	0
Apricot	1	0	5

Fig. 2: Simplified example of a Word Embedding

In this example, N = 3: each word is represented by a vector of 3 values. We notice for example that the words dog and cat have rather similar vectors, and this is what the cosine calculation will show. On the other hand, the calculation of similarity between the words cat and apricot will give a much lower score. In reality, the N dimension of the vectors representing the words is usually of the order of a hundred (N = 100 or 300). As part of automatic language processing, Machine Learning algorithms are now capable of automatically learning the representation of words as vectors. The best known and most widely used algorithm is the Word2Vec algorithm. There are two types of Word2Vec algorithms. The first model is called "Continuous-Bag-Of-Words" (CBOW): it takes as input the context of a word in a text (a window is defined in order to select a certain number of neighbouring words) and predicts as output the most probable word in view of this context. The second Word2Vec algorithm is the Skip-Gram model which performs the exact inverse of the CBOW algorithm: it takes a word as input and predicts its context. The two types of Word2Vec algorithms are three-layer neural networks, the hidden layer of which is composed of about a hundred neurons. Here is their architecture [8] :

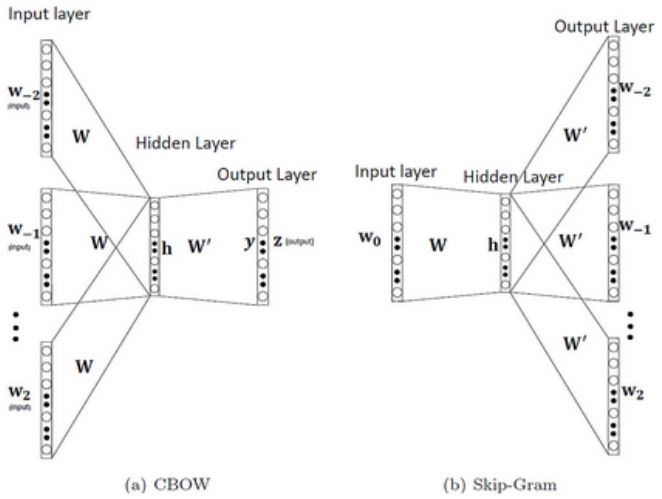


Fig. 3: Architecture of CBOW and Skip-Gram models

Word2Vec algorithms are very powerful and have the advantage of being easy to implement. This is unsupervised learning because the data does not need to be labelled: learning is only done from a database containing many texts, and the analysis of prediction performance is directly carried out from the texts provided as input. Machine learning provides very good prediction performance. However, it is important to note that the validation or not of the predictions of the algorithm remains subjective. There is no right value to predict, since the perception of the language (and especially for complex notions) depends on each individual, and this is precisely why it is a question of unsupervised learning. In order to evaluate the performance of this type of algorithm, it is nevertheless possible to compare the predictions of the algorithm with those that a human being would have made. As far as the Word2Vec algorithm is concerned, the Skip-Gram variant seems to be the most used because it corresponds better to the targeted applications. Moreover, Skip-Gram works well for small databases and represents rare words well, while CBOW is faster and returns a better representation of common words [9]. There are many other algorithms [10] for learning Word Embeddings. First of all, any algorithm that performs dimensionality reduction can be used to create Word Embeddings. This is notably the case of Latent Semantic Analysis (LSA), which uses the classical matrix factorisation technique. Today, it is the Word2Vec algorithm that remains the reference, but there have been attempts to improve it. The GloVe algorithm (Global Vectors for Word Representation) is in particular an extension of the Word2Vec algorithm which uses the matrix factorisation technique used by LSA. Indeed, the Word2Vec algorithm works very well but is dependent on certain parameters, such as the window size used to define the context of the word. It also means that words far away in the text will be more difficult to associate with each other, even if their meaning is close. Matrix factorisation, on the other hand, makes it possible to obtain global statistics on the entire database. For example, a co-occurrence matrix of words over the entire text corpus can be used to link words which would not have been linked in a classic Word2Vec algorithm [11]. A paper from Stanford University has also been published [12] comparing the performance of the Word2Vec and GloVe algorithms. Despite some inhomogeneities due to the fact that the two algorithms do not have the same parameters, and therefore cannot be compared perfectly on the same basis, it emerges from this article that the GloVe algorithm is globally more efficient than the Word2Vec algorithm.

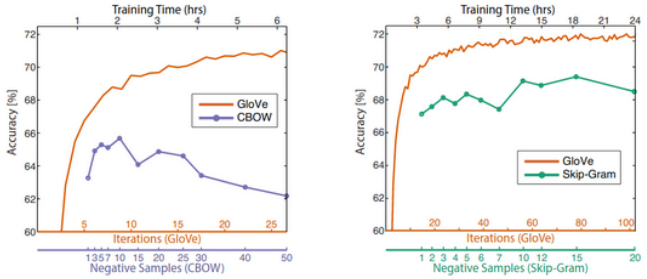


Fig. 4: Performance comparison between Word2Vec and GLoVe algorithms

III. Methodology

In this part, the work carried out to compare two Word Embedding algorithms will be presented: Word2Vec and GloVe. Given the time allocated to this research project, it would have been impossible to test all the Word Embedding methods presented in the literature review. It was therefore necessary to make a choice and the Word2Vec and GloVe algorithms emerged as the most relevant ones to test because, according to the literature review, they are the two most efficient algorithms and, as a result, the most used and most documented. The objective of the work carried out was therefore to test these two algorithms, to compare their performances, and finally to compare these results in particular with those of Stanford University, presented in the literature review. Moreover, the choice was also made not to implement a semantic network because this method is not really comparable with the two other algorithms of Word Embedding, in particular because the performances of a semantic network are difficult to evaluate. Indeed, as previously mentioned, semantic networks are often used for particular applications such as memory modelling, but their implementation here would not have been relevant for the objectives.

To begin with, a first Word2Vec algorithm model was developed step by step, then trained on a small database created for the occasion. This allowed a real understanding of how a Word Embedding algorithm works, the role of the different parameters to be set but also the potential limits of this method. Then, in order to compare the performances of the Word2Vec and GloVe algorithms, two pre-trained models were downloaded and evaluated. Indeed, the computing power required to learn such models on very large databases was too great for the means available. The use of pre-trained models makes it possible to free oneself from this learning time

while still being able to compare the two models according to different criteria.

For the implementation and training of the Word2Vec algorithm on a small database, a Colab Notebook was created in Google Drive. Tensor Flow's high level Keras API was used, as it enables simple and efficient implementation, training and evaluation of Machine Learning and Deep Learning models. It was also necessary to create a database in the form of a text file (Einstein.txt). In order to measure the impact of the size of the text corpus given as input to the algorithm on its performance, the Einstein.txt file initially contained a single biography of Albert Einstein retrieved from the internet. Later, in order to multiply the input data, several biographies (again of Albert Einstein) were added. The corpus of raw text present in the .txt file is first pre-processed, in order to obtain a database usable by the Word2Vec algorithm. Here are the different steps of data pre-processing :

1) Definition of the parameters of the algorithm

2) Splitting the corpus of texts into sequences

3) Vectorization of sequences

4) Removal of words with a very high frequency of occurrence

5) Generation of training data

In a Word2Vec algorithm, it is necessary to adjust certain parameters, depending on the chosen database and the application in question. Here are some of them:

- $vocab\_size$ = vocabulary size: number of (different) words that we consider

- $sequence\_length$ = length of a sequence: number of words for a sequence

- $window\_size$ = size of the window: number of neighbours taken into account for the definition of the context of the selected word

- $embedding\_dim$ = dimension of the embedding: dimension of the vectors representing the words of the vocabulary at the output of the neural network.

A study was carried out to determine the optimal value of the various parameters. The corpus of texts is then divided into sequences, a sequence generally consisting of about ten words. A sequence can be likened to a sentence. Each sequence is then vectorised, i.e. each word in the vocabulary is associated with a unique integer number. Words that appear too frequently in the text corpus are then removed from the vocabulary. This stage normally allows you to get rid of linking words, pronouns, etc. which are not useful for the creation of Word Embeddings, and which could even lead to prediction errors if left as they are. Once the corpus of text has been transformed, the training database can be created, which is made up of targets, concepts and labels. As a reminder, this is unsupervised learning, as the labels are determined directly from the data in the text corpus. The Word2Vec model (three-layer neural network) is then created from the training data using the functions provided by the Keras API and trained. Two .tsv files (vectors.tsv and metadata.tsv) are then created and used to visualise the Word Embedding in the Tensor Flow projector tool, which will allow the data obtained to be analysed and the performance of the model to be evaluated.

Regarding the implementation of the pre-trained Word2Vec and GloVe templates, the principle is the same as in the previous paragraph, however, the steps of creating the database, pre-processing the data, creating the template (Word2Vec or GloVe) and learning the template have already been carried out beforehand. For the Word2Vec algorithm, the model pre-trained by Google has been selected because it is the most complete. The GoogleNews-vectors-negative300.bin file (3.6 GB) contains the Word Embeddings resulting from the training of a Word2Vec model on the Google News database (approximately 100 billion words, or a vocabulary of three million words) [13] and whose output vector size is $embedding\_dim = 300$. This model is very popular and is used in many document classification applications. Similarly, the GloVe model pre-trained by Stanford University has been downloaded. The glove.6B.zip archive contains four GloVe models trained on the Wikipedia database containing a total of 6 Billion words (tokens), or a vocabulary of 400,000 words (tokens are the total number of words in the text corpus, while the vocabulary size is the number of unique words in the input database). The four models have a different dimension, which makes it possible to evaluate the impact of the Word Embedding dimension on the performance of the algorithm. The following four Word Embeddings could therefore be evaluated, which made it possible to estimate the impact of the $embedding\_dim$ parameter on the performance of the model :

- glove.6B.50d.txt ($embedding\_dim$ = 50, 136 MB)

- glove.6B.100d.txt ($embedding\_dim$ = 100, 331 MB)

- glove.6B.200d.txt ($embedding\_dim$ = 200, 661 MB)

- glove.6B.300d.txt ($embedding\_dim$ = 300, 990 MB)

The gensim library was used for the import of pre-trained models and their evaluation. Gensim is a widely used Python library for Word Embedding. For each case (Word2Vec or GloVe), arithmetic tests could be carried out in order to evaluate the performance of each model. Indeed, the interest of transforming words into vectors is that it is then possible to perform arithmetic operations between these words. Examples of arithmetic operations as well as the results obtained will be given in the following section. In addition, the gensim library also provides a reference database called WordSim353 with which models can be compared. This database contains pairs of words that have been associated by human judgement. This database could be likened to labelled data with the value "true" as is usually done in supervised learning. However, this database is not used for learning but simply provides a similarity score between the Word Embeddings created by the model and the associations that a human would have created naturally. The results of the evaluation of the two models will also be given in the next section. Finally, in the same way as for the Word2Vec model trained on the small specific database, the Word Embeddings associated with the two large databases (Google News for Word2Vec and Wikipedia for GloVe) could be visualised in the Tensor Flow Projector. Once again, these results will be provided in the next section and will confirm the performance evaluation of each model.

IV. Results

In this first section will be given the observations resulting from the implementation and training of the Word2Vec algorithm on a small database, as explained in the previous part of the article. The Word2Vec algorithm takes as input a corpus of texts. The idea was therefore to create a corpus on a particular theme in order to be able to evaluate the performance of the algorithm for different corpus sizes. In order to stay within the scientific theme, it was decided that the corpus would contain biographical texts of Albert Einstein. Moreover, in order to evaluate the performance of the model, it is relevant to choose texts whose meaning is familiar to the user, so that he can compare the Word Embeddings created with the associations he would have made himself. Initially, a single biography was copied into the Einstein.txt file but this gave very poor results. Indeed, the Word2Vec algorithm requires a sufficiently large input database to be able to create relevant associations between words. With a single text (a single biography), two similar words are likely to appear only once in the context of each other, if at all. The vectors associated with them will therefore not be in close proximity in the spatial representation of Word Embeddings. Conversely, two words that are not similar may be close to each other in a text. If the text is large enough and this approximation occurs only once, the Word2Vec algorithm will not normally take this into account. On the other hand, if the size of the text corpus is too limited, these associations will be taken into account and will therefore distort the Word Embeddings created. In order to improve the input data, several other biographies have been added to the Einstein.txt file for a total of 14,000 words. Here is the visualisation of the Word Embeddings in the Tensor Flow Projector :

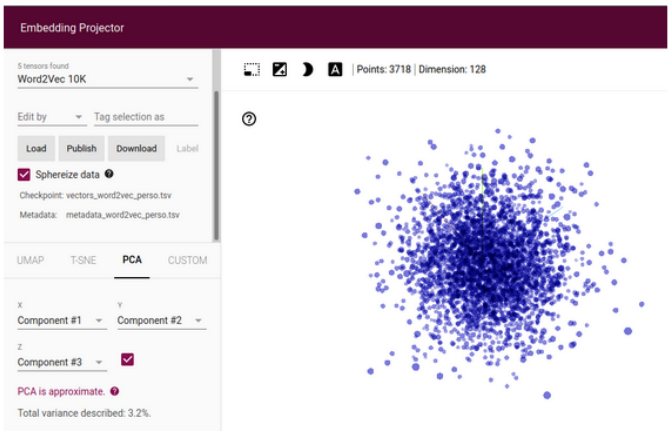


Fig. 5: Visualization of the Word Embedding learned by the Word2Vec algorithm on the text corpus Einstein.txt

A point corresponds to a vector. The size of the circle associated with this point is proportional to the number of occurrences of this word in the input text corpus. Two close points in this three-dimensional space correspond (normally) to two words with a close meaning. The TensorFlow Projector also allows you to view the nearest n neighbours of a selected word. Here are the results for the words 'théorie' and 'science':

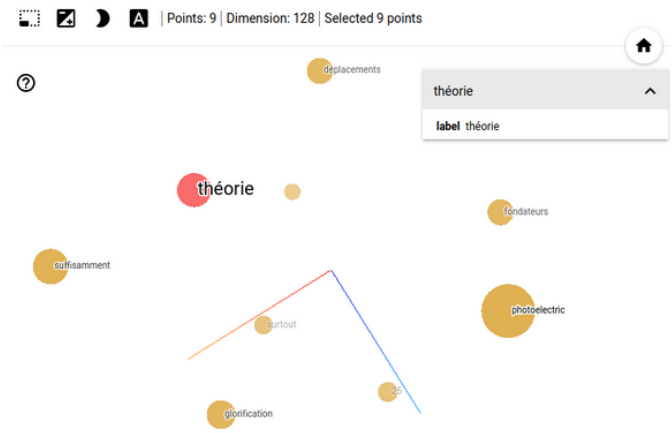


Fig. 6: Visualization of the neighbours of the word "theorie" in the Word Embedding learned by the Word2Vec algorithm on the small dataset Einseint.txt

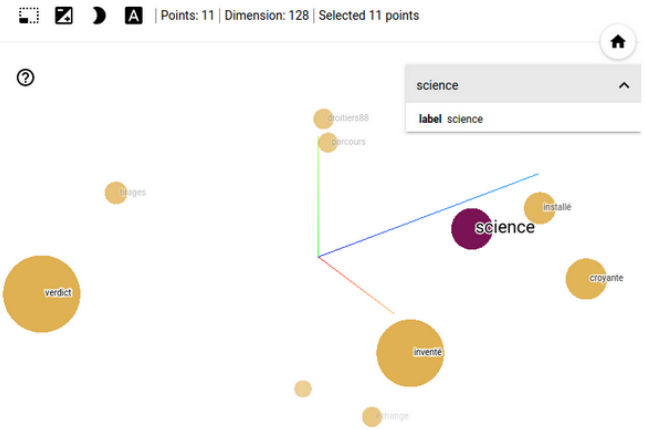


Fig. 7: Visualization of the neighbours of the word "science" in the Word Embedding learned by the Word2Vec algorithm on the small dataset Einseint.txt

For the word "théorie", the words "fondateurs" and "photoélectrique" are parts of the 10 closest words, which appears to be consistent with the input data. However, other words such as "déplacements" or "glorification" should not be included in the list. It can be observed that these are not words whose meaning is totally removed from the word "théorie", but other words such as "relativité" or "scientifique" would have been considered as a priority. The same limitations are observed with the word "science". This is due to a much too small body of text. Indeed, with 14,000, the corpus created is a far cry from the billions of words used by Google or Stanford University. Moreover, some pre-processing of the data could be improved to ignore dates such as "1912" which appears linked to the word science. Another bug related to the small amount of input data is that the word "Einstein" was removed from the vocabulary in the data pre-processing phase because it appeared too recurring compared to other words in the text corpus. This should not have happened because it is actually one of the most important words in the corpus! Nevertheless,

it can be noted that the results are already much better than when a single biography was used. The Word Embeddings created are not optimal, but the results are still consistent and they validate the functioning of the Word2Vec algorithm. Thus, the size of the input text corpus is really decisive in evaluating the performance of a Word Embedding algorithm. Moreover, as mentioned above, different parameters have been tested to try to evaluate the value of the optimal parameters of the algorithm. Here are those that were retained:

- $vocab\_size$ = 4096
- $sequence\_length$ = 10
- $window\_size$ = 2
- $embedding\_dim$ = 128

In the second phase of the work, the impact of the Word Embedding dimension on the performance of the algorithm was evaluated, then the GloVe and Word2Vec models pre-trained on large amounts of data were compared. In order to evaluate the impact of the Word Embedding dimension, the GloVe algorithm was trained for four different embedding dimensions, and in all four cases, the 10 words most similar to the expression ”woman” + ”king” - ”man” were displayed. Here are the results:

“woman” + “king” - “man” = ?				
	D = 50	D = 100	D = 200	D = 300
1	queen (0.852)	queen (0.770)	queen (0.698)	queen (0.671)
2	throne (0.766)	monarch (0.684)	princess (0.608)	princess (0.543)
3	prince (0.759)	throne (0.676)	monarch (0.589)	throne (0.539)
4	daughter (0.757)	daughter (0.659)	throne (0.578)	monarch (0.535)
5	elizabeth (0.746)	princess (0.652)	prince (0.575)	daughter (0.498)
6	princess (0.742)	prince (0.652)	elizabeth (0.546)	mother (0.496)
7	kingdom (0.734)	elizabeth (0.646)	daughter (0.540)	elizabeth (0.483)
8	monarch (0.721)	mother (0.631)	kingdom (0.532)	kingdom (0.477)
9	eldest (0.718)	emperor (0.611)	mother (0.517)	prince (0.467)
10	widow (0.710)	wife (0.610)	crown (0.517)	wife (0.465)

Fig. 8: 10 words most similar to the expression ”woman” + ”king” - ”man” depending on the embedding dimension for the GloVe algorithm

It is noticeable that the 10 words most similar to the expression ”woman” + ”king” - ”man” are almost identical from one dimension to the other, only the order of the words varies somewhat. The word ”queen” comes first in all four cases and all the words are perfectly coherent. The difference from one dimension to another is due to the calculation of the Word Embeddings but this does not affect the overall result. Moreover, one solution is no more accurate than the other, as there is no absolute true value defining a ranking of the 10 words that are semantically closest to the expression ”woman” + ”king” - ”man”. From one person to another, human judgement can vary and the solutions given by the GloVe algorithm are all satisfactory, regardless of the dimension chosen. On the other hand, we can still note that the dimensions tested are not insignificant. Indeed, according to the literature review, the optimal dimension for a Word Embedding algorithm is of the order of a dozen or a hundred, generally between 100 and 300; beyond that, performance may begin to decline. For further work, the dimension chosen will be D = 300 because this is also the dimension used by Google when learning the Word2Vec model. In addition, in order to visualise the impact of the input database size on the calculated Word Embeddings, those of the GloVe algorithm have also been displayed on the Tensor Flow Projector :

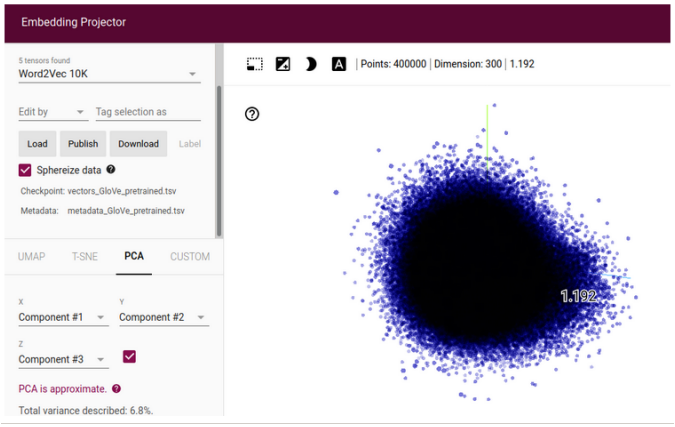


Fig. 9: Visualization of the Word Embedding learned by the GloVe algorithm on the Wikipedia dataset (glove.6B.300d.txt)

A much denser and less legible cloud of dots is observed. In fact, the cloud has 400,000 points (compared to barely 3,000 previously), with a dimension of 300 for each vector. As the Tensor Flow Projector has reached its limits here, it was impossible to load the Word Embedding resulting from the Word2Vec algorithm because the number of points (3 million) exceeds the capabilities of the software. It was nevertheless possible to visualise a few particular words and their closest neighbours. Here are the results obtained for the words ”science” and ”queen”:

Fig. 10: Visualization of the neighbours of the word "science" in the Word Embedding learned by the GloVe algorithm on the Wikipedia dataset (glove.6B.300d.txt)

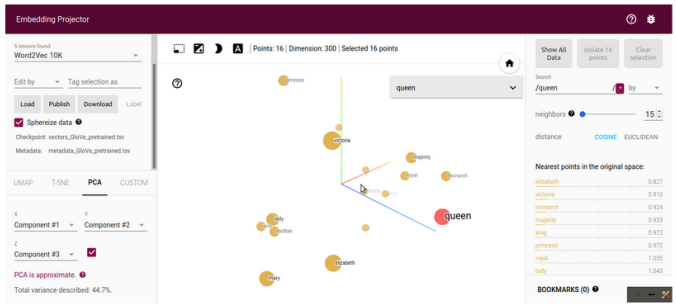


Fig. 11: Visualization of the neighbours of the word "queen" in the Word Embedding learned by the GloVe algorithm on the Wikipedia dataset (glove.6B.300d.txt)

To the right of each image is a list of the closest neighbours associated with the selected word, while in the centre of the image are the vectors in three-dimensional space. For the word "science", the words determined as the closest neighbours are "fiction", "science", "scientific", "physics", "astronomy", "biology", "technology" and "biomedical". The results are therefore much better than those obtained by learning from the small database. It is nevertheless important to note that the Einstein.txt corpus created is far too small, which has led to poor performance on Word Embeddings. On the other hand, one should not necessarily eliminate the idea of creating one's own text corpus and carrying out the model training. Indeed, each method has its advantages and disadvantages. The first option offers the possibility to train your own template and to choose or create your own database adapted to your needs. The advantage: the model created is completely applied to the particular task it will have to perform afterwards. For example, if the purpose of Word Embedding is to classify scientific documents on Albert Einstein, it is interesting to choose this method. If a model pre-trained on a large dataset was used, the results would surely be less good because there would not be enough useful data on Albert Einstein and especially too much useless data. Moreover, if the choice was made to learn one's own model on a large dataset, the learning process would be very long and would not be appropriate for the task at hand. On the other hand, if the targeted application is fairly generalist (common and varied vocabulary), it is interesting to use a large database because it is complete and can be adapted to other applications. Moreover, the use of a pre-trained model allows a considerable time saving because there is no need for additional learning. Thus, the two methods are complementary and the choice of one or the other should depend on the application in question:

→ If the application requires a specific vocabulary: prefer to train your own model on a database specific to the target application.

→ If the application uses a varied and non-specific vocabulary: prefer the use of a pre-trained model on a large database.

Moreover, the pre-trained GloVe and Word2Vec models could be compared thanks to the functions of the gensim library. For example, the function model.similarity() can be used to display the similarity score between two words.

Model	`model.similarity('woman', 'man')`
Word2Vec pre trained model	**0.76640123**
GloVe pre trained model	**0.6998663**

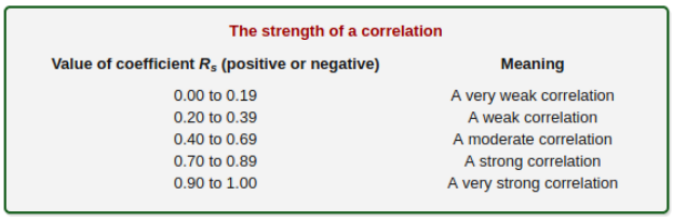
Fig. 12: Similarity score between the words "man" and "woman" in the Word Embedding learned by both the GloVe and the Word2Vec algorithms

However, without comparison with other words, this score is difficult to interpret. This is why, in order to better evaluate Word Embedding templates, the gensim library has created a function to evaluate the performance of the templates in a more general way. The function $evaluate\_word\_pairs()$ calculates the correlation between the Word Embeddings created by the model in question and associations coming from human judgement (wordsim353.tsv). This function returns a Spearman's correlation coefficient and the associated p value. Here are the results obtained :

	`model.evaluate_word_pairs(datapath('wordsim353.tsv'))`	
	Spearman's Correlation Coefficient	***p value***
Word2Vec pre trained model	**0.6589215888009288**	**2.5346056459149263e-45**
GloVe pre trained model	**0.6085349998820805**	**3.879629536780527e-37**

Fig. 13: Spearman's correlation coefficient and the associated p value between the Word Embedding learned by both the GloVe and the Word2Vec algorithms, and associations made from human judgement (wordsim353.tsv)

Spearman's correlation coefficient measures the correlation between two databases. The coefficient is between -1 and 1, where 1 is a perfect correlation; -1 is a perfect negative correlation; and 0 is no correlation at all. The following is a guide to interpreting the score [14] :



The strength of a correlation	
Value of coefficient R_s (positive or negative)	**Meaning**
0.00 to 0.19	A very weak correlation
0.20 to 0.39	A weak correlation
0.40 to 0.69	A moderate correlation
0.70 to 0.89	A strong correlation
0.90 to 1.00	A very strong correlation

Fig. 14: Guide for the interpretation of the Spearman's Correlation Coefficient [14]

For both models, the p-value is very low, which means that the calculated correlation coefficient can be trusted. Indeed, the p value corresponds to the probability that the correlation found is due to chance. We can therefore conclude that for both models, there is a good correlation between the Word Embeddings learned by Machine Learning algorithms and the associations made by human judgement, and that this correlation is assured, although it is not very strong. Finally, the Stanford University paper comparing the GloVe and Word2Vec algorithms attributed the best performance to the GloVe algorithm (see literature review), which seems logical given that the GloVe algorithm is supposed to be an improvement of the Word2Vec algorithm. However, in the work carried out and presented in this article, the correlation coefficient associated with the Word2Vec algorithm is slightly higher than that of the GloVe algorithm. This can be explained by the fact that the database used to learn the Word2Vec model was much larger than the one used for the GloVe model (100 Billion vs. 6 Billion words). Nevertheless, the correlation rates are relatively close in comparison, which confirms the efficiency of the GloVe algorithm despite a smaller training database.

Furthermore, the Stanford University article also pointed out the difficulties encountered in comparing the two models on an equal footing, due to the many parameters to be varied which are not always accessible. To improve the work carried out here, it would be interesting to be able to compare the two models on exactly the same basis, i.e. by initially setting all the parameters identically (quantity of learning data (tokens), vocabulary size, embedding size, size of the context window, etc.) and then varying them one by one. It would also be interesting to be able to train these models on larger databases in order to compare their learning times. Another idea which could prove useful in the case of learning Word Embeddings on a small database and for a particular application would be to create your own database for the assessment (an equivalent of the wordsim353.tsv file) so that you can adapt the calculated score to the specific needs of the application and interpret it more easily.

V. Conclusion

As planned, the work carried out made it possible to test the Word2Vec and GloVe algorithms and to compare their performances. The results obtained are rather satisfactory as they corroborate the information presented in the literature review. Nevertheless, more time and additional means would have been needed to carry out a truly exhaustive study of the various existing solutions. Moreover, the comparison between the Word2Vec and GloVe algorithms remains difficult if pre-trained models are used, as the databases and learning parameters used are not necessarily identical. Ideally, it would therefore be interesting to continue this work by training one's own models on large databases, varying the different learning parameters, in order to be able to more reliably evaluate their impact on the performance of each algorithm.

REFERENCES

[1] F. Rastier, *Sémantique et recherches cognitives*. Presses Universitaires de France, 2010.

[2] "Semantic network," Available at https://en.wikipedia.org/wiki/Semantic_network (2021/02/02).

[3] J.-P. Desclés, "Réseaux sémantiques : la nature logique et linguistique des relateurs," *Langages*, vol. Sémantique et intelligence artificielle, pp. 55–78, 1987.

[4] "Wordnet," Available at https://fr.wikipedia.org/wiki/WordNet (2020/06/04).

[5] M. F.R., "How important are semantic networks in artificial intelligence," Available at https://analyticsindiamag.com/semantic-networks-ai/ (2019/01/23).

[6] D. A. Post, "Word embedding," Available at https://dataanalyticspost.com/Lexique/word-embedding/.

[7] B. Pierrejean, "Comprendre et utiliser les word embeddings," Available at http://w3.erss.univ-tlse2.fr/UETAL/2018-2019/Tuto-embeddings.pdf.

[8] "Word2vec architecture," Available at https://figshare.com/articles/figure/_i_Word2Vec_i_architecture_The_figure_shows_two_variants_of_word2vec_architecture_CBOW_and_Skip_gram_26_for_a_sample_/11982951(2020/03/13).

[9] D. Karani, "Introduction to word embedding and word2vec," Available at https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa(2018/09/01).

[10] A. Bilogur, "Notes on word embedding algorithms," Available at https://www.kaggle.com/residentmario/notes-on-word-embedding-algorithms(2020).

[11] J. Browlee, "What are word embeddings for text?" Available at https://machinelearningmastery.com/what-are-word-embeddings/(2017/10/11).

[12] C. D. M. Jeffrey Pennington, Richard Socher, "Glove: Global vectors for word representation," Ph.D. dissertation.

[13] A. Pai, "An essential guide to pretrained word embeddings for nlp practitioners," Available at https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/(2020/03/16).

[14] B. F. S. Center, "Spearman's rank correlation coefficient rs and probability (p) value calculator," Available at https://geographyfieldwork.com/SpearmansRankCalculator.html(2020/09/01).

An overview of Reinforcement Learning

Hamid HACENE[1,✉]

[1]ENSTA Bretagne third year student in Autonomous Robotics field, Brest, France

Reinforcement learning techniques are particularly useful in the field of artificial intelligence when it is difficult to predict the right decisions to be made. Reinforcement learning allows an agent to learn autonomously a behavior that has never been previously defined by humans. The agent discovers the environment and the different consequences of his actions through interactions with it: he learns from his own experience to build the optimal policy by trial-and-error and continuously interacting with the environment.

This paper reviews the different aspects of reinforcement learning and discusses its basic framework : the optimal policies used in RL (model free policies and model based policies). At last but not the least, this paper briefly describes the Neuro Evolution of Augmenting Topology (NEAT) algorithm, one of the most successful algorithms for solving traditional reinforcement learning tasks.

Reinforcement Learning | IA | Q-learning | NEAT algorithm

Correspondence: ***hamid.hacene@ensta-bretagne.org***

Contents of the paper

Introduction

Reinforcement learning (RL) finds its origins in some subjects: statistics, control theory and game theory. It has a very long history, until the late 80s and early 90s that reinforcement learning technology obtains the wide research and application in some fields such as artificial intelligence and machine learning (1). RL is one of the techniques in artificial intelligence which is usually considered as a goal-directed method for solving problems in uncertain and dynamic environments.

Machine learning problems can be categorized in three ways:

- Supervised learning;
- Unsupervised learning;
- Reinforcement learning.

In RL algorithms, we seek to maximize a reward function. It differs from the supervised learning (SL) methods in the following aspect: in SL, an agent learns from examples (labelled data) which are provided by an external supervisor.

In RL, the agent learns by trying many actions and determines which of those actions produce the best reward (2). This is achieved by directly interacting with the system and its environment.

Basic concepts of RL

A reinforcement learning model consists of:

- A discrete set of environment states "S";
- A discrete set of agent action "A";
- A set of scalar rewards 0, 1 or reals;

A reinforcement learning agent is autonomous which means that its behavior is determined by its own experience. Learning is the mechanism through which an agent can increase its intelligence while performing operations. What is outside the agent is considered as the environment. The states are parameters that describe the environment. A RL agent senses the environment and learns the optimal policy by taking actions in each state of the environment. It tries to maximize the reward or minimize the punishment (3). The Continuous learning and adapting through interaction with environment helps the agent to learn online in terms of performing the required task and improving its behavior in real time.

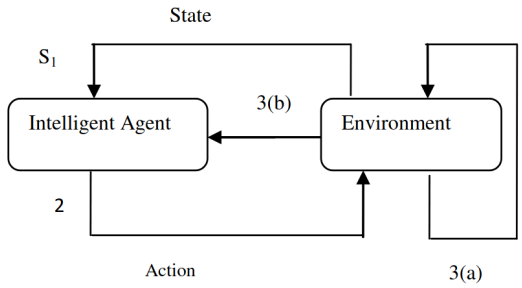


Fig. 1. Basic model of RL

Figure 1 shows the basic model of the reinforcement signal :

1. The agent receives as input, some indication of current state of the environment;
2. The agent then chooses an action from the set of actions (A) to generate as output;
3. (a) The action changes the state of environment;

(b) The value of this state transition is communicated to the agent through a scalar reward (r).

Knowledge gained by an RL agent is specific to the environment in which it is trained and cannot be easily transferred to another agent, even if the environments are very similar. This is one of the significant issues in the domain.

Learning of the optimal policy

Optimal Policies of RL. Before making RL algorithms to behave optimally, we have to choose the models of optimality. There are four optimality criterions used for generation of scalar reward/punishment signal (4):

- *Finite-horizon model*: in this model, at a given moment the agent should optimize its expected reward for the next n steps : $E(\sum_{i=0}^{n} r_i)$. In this expression, r_i represents the scalar reward after n steps into the future. It is the easiest model.
- *Receding-horizon control*: it always takes n-step optimal action. The agent acts always according to the same policy, the value of n limits how far an agent looks in choosing its actions.
- *Infinite-horizon discounted model*: The rewards received in the future are geometrically discounted according to the discounted factor γ whose value should lie between 0 and 1 : $E(\sum_{i=0}^{n} \gamma^i r_i)$. This model takes into account a long-run reward of the agent.
- *Average-reward model*: the agent takes actions which optimize its long-run average rewards : $\lim_{n\to\infty} E(\frac{1}{n}\sum_{i=0}^{n} r_i)$

The choice of optimality model and the choice of parameters matters much and so should be chosen carefully.

Learning the policy. In a Markov Decision Problems (MDP), a model is already present to obtain an optimal policy. An MDP model is usually consists of :

1. A discrete set of environment states S;
2. A discrete set of agent action A;
3. A reward function R;
4. A state transition probability function P, which maps different states to its probabilities.

$P(s_1, a, s_2)$ denotes the probability of state transition from state s_1 to state s_2 using action a. $R(s_1, a)$ denotes the immediate reward value obtained when the environment state s_1 changes to state s_1 using action a.
The model is markov, when the state transitions are independent of any previous states or the agent actions.
Reinforcement learning is mainly concerned with how an optimal policy can be obtained when a model is not known in advance. The agent directly interacts with the environment to obtain information, which is processed through appropriate algorithms to produce an optimal policy. The two ways in which it can be proceed: *model-free method* and *model-based methods*. In model-free methods, a controller is learnt without learning a model whereas in model-based methods , first the algorithm is learnt then it is used to derive a controller. Some model-free methods are : temporal difference methods, Q-learning, average rewards. Some model-based methods are : Certainty equivalent, Dyna, queue Dyna, priority sweeping...
In this paper, we will focus on model–free methods, especially the Q-learning method.

Model-Free Methods. A model-free algorithm is an algorithm which does not use the transition probability distribution associated with the Markov decision process (MDP). It can be thought of as a trial-and-error algorithm.
Q-learning. Q-learning algorithm was proposed by Watkins (5).
Let $Q^*(s_1, a)$ be the expected discounted reward of taking action a in the state s_1, then continues by choosing action optimally.
Let $V^*(s_1) = max_a Q^*(s_1, a)$ where $V^*(s_1)$ is the value of s_1 with the assumption that the best action is taken initially.
The recursion formula of $Q^*(s_1, a)$ is as follows:
$Q^*(s_1, a) = R(s_1, a) + \gamma \sum_{s_2 \in S} T(s_1, a, s_2) max_a Q^*(s_2, a')$
The optimal policy would be given by :
$\Pi^*(s) = argmax_a Q^*(s_1, a)$
The action can be chosen by taking the one which has the maximum Q-value for the current state. The Q-learning rule is given by:
$Q^*(s_1, a) = Q(s_1, a) + \alpha(r + \gamma max_a Q^*(s_2, a') - Q(s_1, a)$
Which can be rewritten as :
$Q(s_1, a) = (1 - \alpha)Q(s_1, a) + \alpha(r + \gamma max_a Q^*(s_2, a'))$
If α decays appropriately and each action is executed in each state an infinite number of times in an infinite run then Q-values will converge with a probability of 1 to Q^*. When Q values are nearly converged to their optimal values, the agent can act greedily by selecting the actions with the highest Q value.
Q-learning is exploration insensitive i.e. the Q values will converge to optimal values, independent ofhow an agent behaves when the data is being collected. The details of the exploration strategy do not affect the convergence of the learning algorithm (6). Q-learning is the most popular and most effective model-free algorithm for learning from delayed environment. Q-learning can be applied to discounted infinite-horizon MDPs. Q-learning can also be applied to undiscounted problems if the optimal policy is guaranteed to reach a reward-free state and the state is periodically reset.

Implementation of Q-learning

To implement the Q-learning algorithm, we are going to be working with *OpenAI's gym*, specifically with the **"MountainCar-v0"** environment.
For the various environments, we can query them for how many actions are possible. In this case, there are three actions

we can pass. This means, when we step the environment, we can pass a 0, 1, or 2 as our "action" for each step. Each time we do this, the environment will return to us the new state, a reward, whether or not the environment is done/complete, and then any extra information.

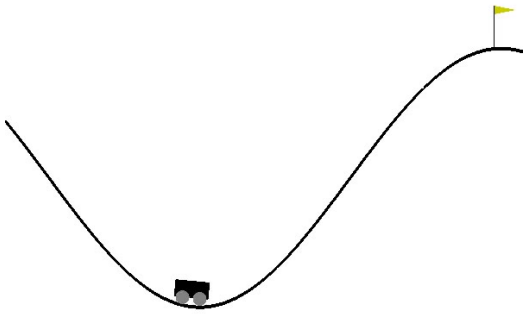


Fig. 2. MountainCar-v0 environment

It doesn't matter to our model, but, a 0 means push left, 1 is stay still, and 2 means push right. We won't tell our model any of this, and that's the power of Q-learning. This information is basically irrelevant to it. All the model needs to know is what the options for actions are, and what the reward of performing a chain of those actions would be given a state. As we can see in figure 2, despite asking this car to go right constantly, we can see that it just doesn't quite have the power to make it. Instead, we need to actually build momentum to reach that flag. To do that, we wouldd want to move back and forth to build up momentum. We will use Q-learning to solve this problem.

In the case of this gym environment, the observations space is position (along an horizontal axis) and velocity.

At each step, we get the new state, the reward and whether or not the environment is done (number of steps for example).

First step, we need to build our Q value table which contains all of our possible discrete states.

Then, we will use the algorithm described before to update those values in the Q table and build an intelligent agent to reach the goal.

Results. The main codes can be found on my *github*. I will expose some results on the evolution of the Q-learning algorithms.

We tracked some very basic metrics from within our program (average rewards, maximum rewards and minimum rewards). This metrics will give us a correct view of how well the agent is dowing. Besides, there are some parameters that we need to tune to find the best way to train the agent by building the optimal Q table.

Figure 3 shows that the agent can still going by training to improve its score. The minimum reward is at its least value. By increasing the number of episodes to 10.000, we significantly improve the metrics : in figure 4 the minimum rewards ar no longer at the least values. The average rewards ar quiet good also.

We can go further by training our agent for 25k episodes. The metrics are shown in figure 5. By analyzing this, it looks like

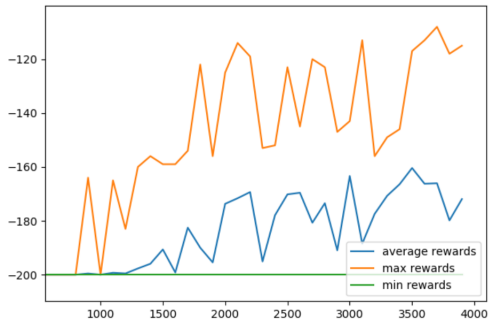


Fig. 3. 4000 episodes

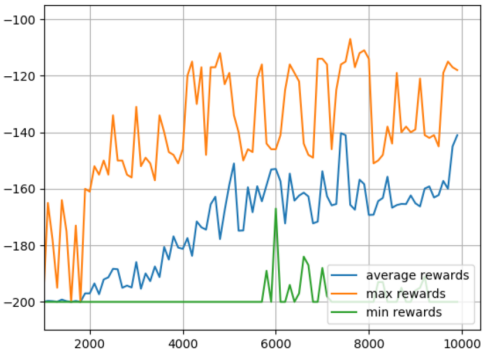


Fig. 4. 10000 episodes

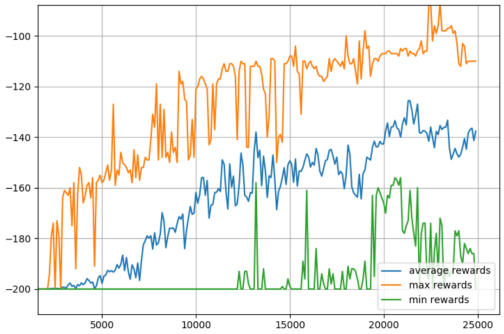


Fig. 5. 25k episodes

we may have the model around 20K episodes since it had high overall rewards, but also the minimum is still high too.

After that, we can save the final trained table to replay our agent later.

We finally found a way to get the Q table for each action. The result is shown in figure 6

NEAT algorithm

Neuroevolution algorithms. Neuroevolution is a field of study dedicated to the generation and improvement of neural networks using evolutionary algorithms. Traditionally associated with the generation of neuron weights through evolution, current approaches associated focus on multiple aspects of the construction of a network, such as learning their activation functions, hyperparameters (learning rates), architectures (number of neurons per layer, number of layers, and which layers connect to which) and even the rules for learn-

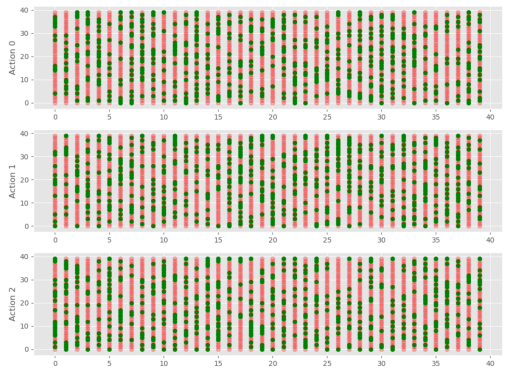


Fig. 6. Q table

ing themselves. One famous neuroevolution approach called Neuroevolution of Augmenting Topologies.

NEAT algorithm. Neuroevolution of Augmenting Topologies, also called NEAT, is an algorithm designed for neural network topology construction (7). NEAT uses a genetic algorithm structure to generate small initial networks that evolve and grow over generations by adding neurons and connections and adjusting their weights to generate structures capable of performing well while keeping them minimal in size. This minimalist aspect of NEAT is one of its core differences to otherneuroevolution algorithms, as it focuses on only adding neurons or connections when they have an active impact in the network's performance (8).
Starting with an initial population of small networks based on a common topology, NEAT evaluates changes tothese networks iteratively by adding and removing neurons and connections across generations. The algorithm defines the genome that describes the nodes and connections by a mechanism called genetic encoding, used in the operations that modify the network structures through classic genetic algorithm operators such as crossover and mutations (9).

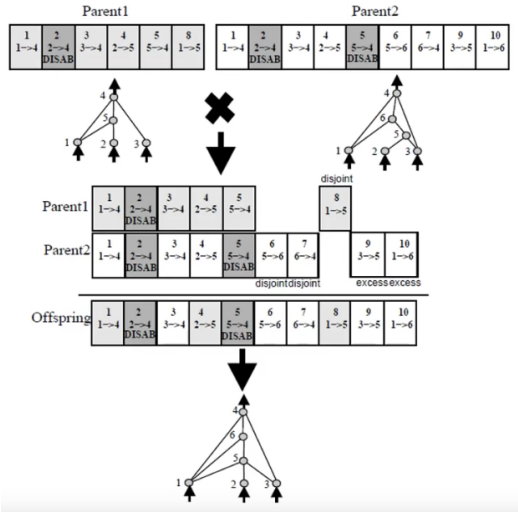


Fig. 7. NEAT mutations

Implementation. For implementation purposes, we will use the *NEAT-Python* library : https://neat-python.readthedocs.io/en/latest/.

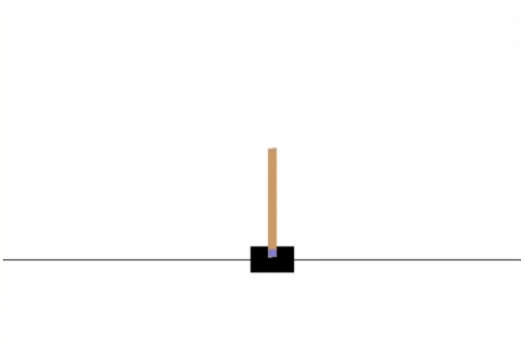


Fig. 8. NEAT configuration file

We are going to be working with *OpenAI's gym*, specifically with the **"CartPole-v1"** environment (figure 8).

```
# NEAT configuration for the bit-sequence memory experiment.

# The `NEAT` section specifies parameters particular to the NEAT algorithm
# or the experiment itself.  This is the only required section.
[NEAT]
fitness_criterion     = max
fitness_threshold     = 50.0
pop_size              = 250
reset_on_extinction   = 0

[DefaultGenome]
num_inputs              = 24
num_hidden              = 2
num_outputs             = 4
initial_connection      = partial_direct 0.5
feed_forward            = True
compatibility_disjoint_coefficient    = 1.0
compatibility_weight_coefficient      = 0.6
conn_add_prob           = 0.2
conn_delete_prob        = 0.2
node_add_prob           = 0.2
node_delete_prob        = 0.2
activation_default      = clamped
activation_options      = clamped
activation_mutate_rate  = 0.00
aggregation_default     = sum
aggregation_options     = sum
aggregation_mutate_rate = 0.0
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_replace_rate       = 0.1
bias_mutate_rate        = 0.7
bias_mutate_power       = 0.5
bias_max_value          = 30.0
bias_min_value          = -30.0
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_replace_rate   = 0.0
response_mutate_rate    = 0.0
response_mutate_power   = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
```

Fig. 9. NEAT configuration file

```
Population of 500 members in 4 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
    14   43    65    500.0    0.141     9
    15   43   161    500.0    0.397    15
    19   43   120    500.0    0.324    17
    23   43   154    500.0    0.392    10
Total extinctions: 0
Generation time: 0.337 sec (0.342 average)

 ****** Running generation 44 ******

Population's average fitness: 200.46800 stdev: 186.39670
Best fitness: 500.00000 - size: (6, 9) - species 15 - id 20495

Best individual in generation 44 meets fitness threshold - complexity: (6, 9)
Key: 324
Fitness: 500.0
Nodes:
        0 DefaultNodeGene(key=0, bias=0.17922921425757102, response=1.0, activation=
clamped, aggregation=sum)
        1 DefaultNodeGene(key=1, bias=0.5211318770179293, response=1.0, activation=c
lamped, aggregation=sum)
        648 DefaultNodeGene(key=648, bias=-0.8839543746599205, response=1.0, activat
```

Fig. 10. Result of running NEAT algorithm

In the NEAT library, we have to set a configuration file as shown in figure 9. In this file, we set various hyperparameters, but at this early stage (when this paper was written),

we don't have all the expertises to tune the parameters. The only results that we can show at this moment are presented in figure 10.

Conclusion

There are a large variety of RL techniques that work effectively on a variety of small problems. But there exist very few techniques that can be scaled to larger problems. The problem is it is very difficult to solve arbitrary problems in the general case. NEAT algorithm produced very good results in simple environments, but we have to test the implementation on several environments to have a precise idea of how good the model could be.

Through this work, we have conducted extensive research on a constantly and rapidly evolving scientific subject. This work allowed us to experience research and constitutes a first approach to the world of PhD studies, doors that could be opened to us after our engineering degree.

Bibliography

1. Rashmi Sharma, Manish Prateek, and Ashok K. Sinha. Article: Use of reinforcement learning as a challenge: A review. *International Journal of Computer Applications*, 69(22):28–34, May 2013. Full text available.
2. M. Iosifescu and R. Theodorescu. On bush-mosteller stochastic models for learning. *Journal of Mathematical Psychology*, 2(1):196–203, 1965. ISSN 0022-2496. doi: https://doi.org/10.1016/0022-2496(65)90025-8.
3. *Artif. Intell. Rev.*, 17(3), 2002. ISSN 0269-2821.
4. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.
5. Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.
6. Gary G. Yen and Travis W. Hickey. Reinforcement learning algorithms for robotic navigation in dynamic environments. *ISA Transactions*, 43(2):217–230, 2004. ISSN 0019-0578. doi: https://doi.org/10.1016/S0019-0578(07)60032-9.
7. Kenneth O. Stanley. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2004.
8. Kenneth O. Stanley and Risto Miikkulainen. Efficient evolution of neural network topologies. In William B. Langdon, Erick Cantu-Paz, Keith E. Mathias, Rajkumar Roy, David Davis, Riccardo Poli, Karthik Balakrishnan, Vasant Honavar, G"unter Rudolph, Joachim Wegener, Larry Bull, Mitchell A. Potter, and Alan C. Schultz, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1757–1762, Piscataway, NJ, 2002. San Francisco, CA: Morgan Kaufmann.
9. Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Evolving adaptive neural networks with and without adaptive synapses. In *Proceedings of the 2003 Congress on Evolutionary Computation*, Piscataway, NJ, 2003. IEEE.

Initiation à la recherche

Deep learning pour la detection des objets : Comparaison entre le Yolo et le Faster R-CNN avec une application sous Gazebo du YOLO à l'aide d'un drone.

Réalisé par : Maha HALIMI

Encadré par : Luc JAULIN

Mot clés :

Deep Learning,
Détection d'objet,
Drone,
Algorithme,
YOLO,
Faster R-CNN,
CNN,
Classifications.

Résumé :

De nos jours, il existe un besoin croissant de drones volants dotés de capacités diverses pour applications civils et militaires. En effet les tendances croissantes du marché des drones et l'intérêt pour des applications potentielles telles que la surveillance, la navigation visuelle, la détection d'objets et la planification d'évitement d'obstacles basée sur des capteurs ont été porteurs de bonnes promesses dans le domaine du Deep Learning. Les algorithmes de détection d'objets mis en œuvre dans le cadre du Deep Learning, sont rapidement devenus des méthodes de traitement d'images en mouvement capturées à partir de drones.
En outre, la détection d'objet a évolué parallèlement à l'avancement, sans précédent, de Convolutional Neural Network "CNN" et de ses variantes. [1],[2],[3].

Au sein de cet article de synthèse, on présente, à l'aide d'une illustration par simulation sous GAZEBO d'un véhicule aérien sans pilote (UAV), une comparaison entre "Faster R-CNN", et l'un des meilleurs systèmes de détections d'objets de traitement en temps réel, You Only Look Once, ou autrement "YOLO". Ce dernier innove une nouvelle façon de la détection d'objets avec la manière la plus simple, la plus efficace et la plus complète. Vu sa vitesse la plus rapide, il a atteint un résultat incomparable sans précédent, qui a dépassé considérablement le résultat des performances du Faster R-CNN. [4],[5]

En outre, comparé à la dernière solution la plus avancée, YOLO réalise un excellent compromis du point de vue vitesse et précision ainsi qu'une détection d'objets d'une forte capacité de généralisation pour représenter l'image entière. [6]
Lien vers GitHub : https://github.com/MahaHalimi/object-detection

1. Introduction :

Ces dernières années, les véhicules aériens sans pilote (UAV) autonomes, en particulier les drones équipés de caméras, sont devenus très populaires en raison de la diversité de leurs applications telles que la surveillance, la cartographie aérienne, la recherche et le sauvetage, etc. La compréhension des données visuelles collectées de manière autonome à partir de ces drones a été un domaine d'intérêt croissant. En effet, la détection visuelle d'objets est l'un des aspects les plus importants des applications des drones qui demeure essentielle dans les systèmes entièrement autonomes.[7]

Cependant, la détection d'objets avec des drones est très difficile et complexe car elle est impactée par diverses conditions d'imagerie telles que le bruit, le flou, la faible résolution, la petite taille des cibles, etc. La tâche est encore plus difficile en raison de la limite des ressources informatiques disponibles sur les drones et la nécessité de performances en temps réel dans de nombreuses applications telles que la navigation. Les principaux défis dans le déploiement des capacités de vision et d'intelligence sur une plate-forme d'UAV sont: atteindre une consommation d'énergie minimale afin de minimiser l'effet sur la consommation d'énergie de la batterie et globalement temps de vol du drone, nécessiter moins d'empreinte mémoire et de puissance de calcul car les drones typiques ont des limitations de ressources et traiter les données d'entrée de sa caméra avec faible latence et les exécuter plus rapidement afin d'effectuer des tâches critiques telles que la détection d'objets, l'évitement et la navigation de chemin en temps réel. [8] ,[9].

Avec la percée du Deep Learning utilisant les réseaux de neurones convolutifs, il y a eu une augmentation frappante des performances dans le traitement des tâches de vision par ordinateur. L'idée clé est d'apprendre les caractéristiques et le modèle des objets à partir des données brutes de pixels de l'image. La formation de ces modèles d'apprentissage en profondeur nécessite généralement de grands ensembles de données, mais ce problème a également été surmonté par la publication de nombreux ensembles de données étiquetées à grande échelle tels que ImageNet, COCO, Pascal VOC, etc. Cependant, ces méthodes nécessitent une énorme quantité de puissance de calcul et de mémoire intégrée. Il y a toujours eu un compromis entre les performances de détection et la vitesse. [10]

Dans cet article, nous allons explorer et comprendre l'architecture et le fonctionnement de Faster R-CNN et YOLO [URL 7].

A l'aide d'une comparaison entre le Faster R-CNN et le YOLO au niveau de leurs précisions, leurs vitesses, leurs coûts et leurs complexités leurs avantages et leurs lacunes seront déterminés.

Enfin, au niveau de la conclusion il sera question d'aborder les perspectives de chaque algorithme dans l'avenir. [11]

2. Faster R-CNN

Basé sur Fast R-CNN, Faster R-CNN [12] résout le problème de la RPN (Region Proposal Networks), qui est sa contribution clé. Faster R-CNN obtient la RPN non pas sur l'image originale mais sur l'image finale qui sera l'entrée. Comme la résolution de l'image principale est inférieure à celle de l'image originale, le calcul de Faster R-CNN est certainement beaucoup moins que celui de tous les anciens modèles de CNN. [13]

La caractéristique principale de Region Proposal Network (RPN) est que chaque proposition de glissement générera 9 ancres candidates avec différentes échelles: largeurs et hauteurs. Le concept du Faster R-CNN pour extraire les caractéristiques des ancres est similaire à celui du Fast R-CNN, tandis que sa classification d'objet consiste uniquement à reconnaître que les caractéristiques sont au premier plan ou en arrière-plan. Et la régression de la proposition est uniquement pour trouver un emplacement plus précis de l'objet cible. Pour chaque emplacement de la proposition, RPN utilise deux couches entièrement connectées (classification des objets et régression de proposition) pour juger et abandonner les ancres. Il ne fait jamais de proposition régionale explicite. Les principales règles de sélection des ancres sont : rejeter les ancres sur la frontière, les ancres dont la zone de chevauchement avec l'échantillon qui supérieurs à 0,7 seraient classés comme premier plan, et celles dont les zones de

chevauchement inférieures à 0,3 seraient classées comme arrière-plan. [14],[15] [16]

De cette façon, RPN choisit environ 300 ancres pour chaque proposition glissante. Faster R-CNN utilise le mode d'entraînement en alternance pour entraîner les fonctionnalités partagées. Il utilise le récent réseau pour initier le poids du RPN, extrait les bonnes propositions de l'ensemble de données de formation et entraîne le modèle Faster R-CNN avec les propositions à plusieurs reprises jusqu'à ce que le résultat converge bien.

Faster R-CNN a déjà offert un résultat avec une précision de reconnaissance parfaite. Seulement nécessite une amélioration supplémentaire au niveau de sa vitesse, qui est la principale raison pour laquelle sont basés les algorithmes conçus ultérieurs. [19]

3. YOLO

Une nouvelle approche de la détection d'objets nommée You Only Look Once (YOLO), c'est à dire que l' image prédite nous informe sur les objets et de l'endroit où ils se trouvent en un seul coup d'œil. En tant que première méthode qui élimine entièrement le pipeline, elle encadre la détection d'objet comme un problème de régression vers des boîtes de délimitation spatialement séparées et des probabilités de classe associées, qui sont prédites avec un seul réseau de neurones à partir d'images complètes dans une évaluation.

En tant que première méthode qui élimine entièrement le pipeline, elle encadre la détection d'objet comme un problème de régression vers des boîtes de délimitation spatialement séparées et des probabilités de classe associées, qui sont prédites avec un seul réseau de neurones à partir d'images complètes dans une évaluation.

Le réseau de détection de YOLO comprend 24 couches convolutives suivies de 2 couches entièrement connectées. YOLO utilise des couches de réduction 1 × 1 suivies de couches convolutives 3 × 3. L'alternance des couches convolutives 1 × 1 réduit l'espace des caractéristiques des couches précédentes.

Les couches convolutives sont pré-entraînées sur la tâche de classification ImageNet à la moitié de la résolution (image d'entrée 224 × 224), puis doublent la résolution pour la détection.

YOLO utilise une fonction d'activation linéaire pour la couche finale et un ReLU pour toutes les autres couches. Ainsi qu'il prédit les coordonnées des boîtes englobantes directement à l'aide de couches entièrement connectées au-dessus de l'extracteur d'entités convolutifs. YOLO ne prédit que 98 boîtes par image. [20] [21]

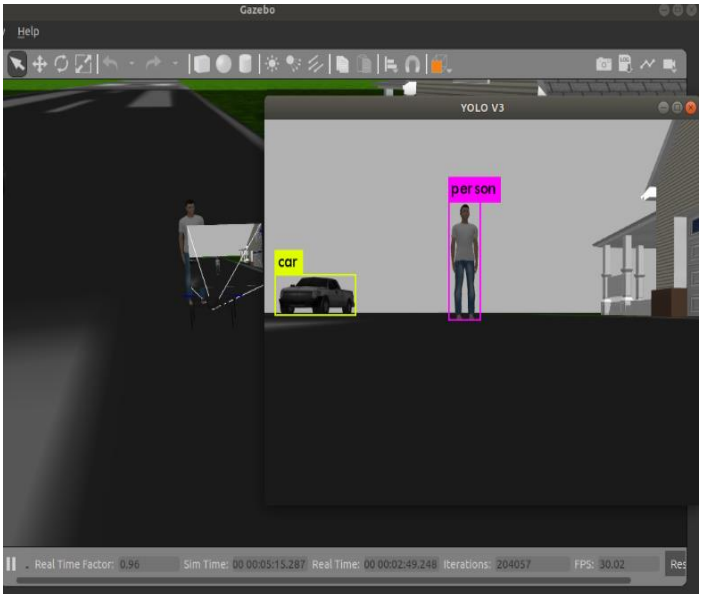


Figure 1 : Détection d'objet par YOLO sous Gazebo

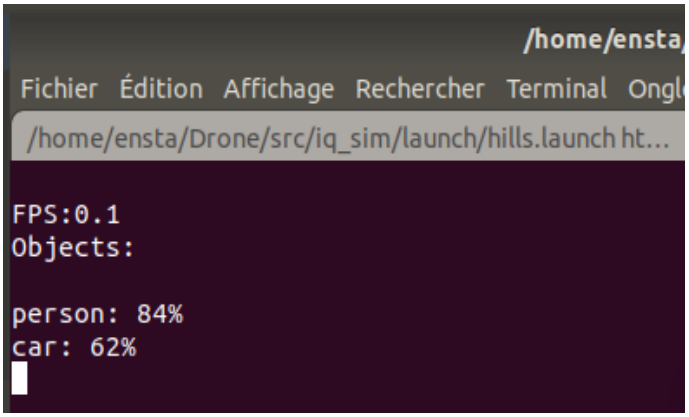


Figure 2 : Pourcentage des objets détectés

4. Comparaison :

YOLO suggère toute une nouvelle façon de traiter l'image qui varie trop, non seulement de Faster R-CNN mais aussi de R-CNN et de

toutes ses variantes. Dans ce qui vient, seules les principales différences entre YOLO et Faster R-CNN seront discutées comme suit :

- ✓ *Le Framework :*

Bien que Faster R-CNN et YOLO utilisent CNN comme noyau, et que leurs objectifs clés soit de trouver une meilleure manière de diviser les propositions basées sur CNN, leurs cadres diffèrent grandement les uns des autres.

Faster R-CNN conserve le cadre général traditionnel de R-CNN : l'utilisation de CNN traite l'ensemble de l'image d'entrée au début et divise les propositions plus tard, en maintenant la proposition régionale et la mise en commun des ROL.

La contribution adopte le Réseau de proposition régional pour accélérer spécialement le calcul des traitements des propositions. YOLO divise l'image entière au tout début et utilise plus tard CNN pour le traitement. En outre, YOLO abandonne complètement les fenêtres coulissantes et la RPN et divise l'image d'entrée en S X S grilles. Pendant ce temps, il innove le mécanisme d'IOU de confiance pour chaque réseau et la probabilité de classe pour prendre une décision. Contrairement à YOLOv2 qui réutilise les boîtes d'ancrage, YOLOv1 l'a abandonné à la création de YOLO. Le YOLOv2 utilise 5 boîtes d'ancrage tandis que Faster R-CNN a 9 boîtes d'ancrage.

De plus, YOLOv2 crée de nombreuses nouvelles techniques pour améliorer la précision, telles que les clusters de dimensions, alors que Faster R-CNN ne prend pas en charge.

- ✓ *Performance extérieure :*

Faster R-CNN se concentre sur l'accélération du Framework R-CNN en partageant le calcul et en utilisant les réseaux de neurones pour proposer des régions au lieu de la recherche sélective. Bien qu'il offre la rapidité, la précision et les améliorations par rapport à R-CNN, les deux sont toujours en deçà des performances en temps réel. Au lieu d'essayer d'optimiser les composants individuels d'un grand pipeline de détection, YOLO jette complètement le pipeline de plus il est rapide intentionnellement. Ainsi, YOLOv2 peut atteindre une haute précision et convient en temps réel pour des images à haute résolution. Alors que pour les images à faible résolution, il montre une vitesse élevée exceptionnelle avec une excellente valeur mAP.

En outre, aucun des systèmes de détection n'a surpassé les performances inégalées de Fast YOLO avec 155 FPS et valeur mAP 52,7 jusqu'à présent.

5. Conclusion :

Cet article aborde brièvement des algorithmes actuels de détection d'objets, spécifiquement Faster R-CNN, et YOLO. Par rapport aux Faster R-CNN, YOLO a une application plus avancée dans la pratique. C'est un modèle unifié de détection d'objets. Il est simple à construire et peut être formé directement sur des images complètes. Contrairement aux approches basées sur les classificateurs, YOLO est formé sur une fonction de perte (loss function) qui correspond directement aux performances de détection. L'ensemble du modèle est entrainé conjointement.

Fast YOLO est le détecteur d'objet polyvalent le plus rapide, et YOLOv2 fournit sur une variété de jeux de données de détection, le meilleur compromis entre la réelle vitesse temporelle et l'excellente précision pour la détection d'objets que d'autres systèmes de détection.

En outre, YOLO généralise mieux la représentation des objets que les autres modèles, ce qui le rend idéal pour les applications qui reposent sur une détection d'objets rapide et robuste. Ces avantages excellents et précieux le rendent digne d'être fortement recommandé et popularisé.

A l'exception de la structure de chaque algorithme, le défi le plus urgent dans l'avenir pour la machine Learning est la portée de l'ensemble de données. La disponibilité de données de formation appropriées pourrait être la partie vitale du processus d'apprentissage, pour obtenir des résultats idéals.

6. Bibliographie :

[1] D. Krijnen, C. Dekker, AR Drone 2.0 with Subsumption Architecture, In Artificial intelligence research seminar, 2014.

[2] A. Cavoukian, Privacy and Drones: Unmanned Aerial Vehicles, Information and Privacy Commissioner of Ontario, Canada, 2012.

[3] S.G. Gupta, M.M. Ghonge, P.M. Jawandhiya, Review of unmanned aircraft system (UAS), Technology 2 (4) (2013).

[4] U.K. MoD, Joint Doctrine Note 2/11 the UK Approach to Unmanned Aircraft Systems, UK MoD The Development, Concepts and Doctrine Centre, SWINDON, Wiltshire, 2011.

[5] R.J. Bachmann, F.J. Boria, R. Vaidyanathan, P.G. Ifju, R.D. Quinn, A biologically inspired micro-vehicle capable of aerial and terrestrial locomotion, Mech. Mach. Theory 44 (2009) 513–526.

[6] Kabrisky, M. (1964). a Proposed Model for Visual Information Processing in the Human Brain.

[7] https://towardsdatascience.com/computer-vision-a-journey-from-cnn-to-mask-r-cnn-and-yolo-part-2-b0b9e67762b1

[8] Springer, Berlin, Heidelberg. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L.D. (1989).Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.

[9] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L.D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.

[10] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[11] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[12] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015).

[13] Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern (p :1-9)

[14]He,K.,Zhang,X.,Ren,S.,&Sun,J.(2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision & pattern recognition (pp. 770-778).

[15] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object detection with discriminatively trained part-based models. IEEE transactions on pattern analysis and machine intelligence, 32(9), 1627-1645.

[16] Redmon, J., & Farhadi, A. (2016). YOLO9000: better, faster, stronger. arXiv preprint arXiv:1612.08242.

[17] Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 806-813).

[18] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[19] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[20] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

[21] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 779-788).

[22] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

Online VS Offline algorithms for the Traveling Salesman Problem

Mourtaza KASSAMALY, student at ENSTA Bretagne (Brest, France) in the field of Autonomous Robotics
mourtaza.kassamaly@ensta-bretagne.org

***Abstract*—Solving the Traveling Salesman Problem is NP-hard and no deterministic solution can be found. Near-optimal solutions can be used by using heuristics or through the use of swarm intelligence and AI algorithms such as Reinforcement or Deep Learning. This study is included in an Initiation to Research course proposed at École Nationale Supérieure des Techniques Avancées de Bretagne (ENSTA Bretagne) in the field of Autonomous Robotics. Its goal is to assess the efficiency of online path-planning algorithms for the solving of the Traveling Salesman Problem, in comparison to offline path-planning algorithms.**

***Index Terms*—Monte-Carlo Tree Search, Ants Colony Optimization, Optimal path, Near-optimal path, Waypoints, Shortest Path, Traveling Salesman Problem, Path planning, Online planning, Offline planning**

I. Introduction

In various fields such as infrastructure monitoring or autonomous deliveries and logistics, Autonomous Vehicles can be used to travel a given number of locations. Trajectories must be optimized to avoid a waste of time and energy and thus, looking for the shortest path traveling key locations becomes crucial. This is known as the Traveling Salesman Problem (TSP). It consists in finding the optimal, i.e. shortest path traveling a set of spatial points only once. It is derived in many variants as described in [1] where each spatial point has to be visited more than once or where obstructions can occur as static or dynamic obstacles. In the Dynamic Traveling Salesman Problem (DTSP) spatial points can be mobile. In the Multiple Traveling Salesman Problem (MTSP) targets are visited by a swarm of vehicles to split the workload. Other variants simulate better the problem of Autonomous Deliveries such as the Black and White Traveling Salesman Problem [2] in which spatial points are colored in black or white and the vehicle has to travel all the points but without exceeding a given number consecutive white points without having visited a black one. This could address the problem of visiting locations by stopping at charging stations.

II. Algorithms

A. Formulation of the problem

Let us consider $G = (V, E)$ a weighted directed graph with V and E its vertices and edges. There are M vertices (waypoints) and the graph is fully connected which means there is an edge between every pair of waypoints. The weight of edges will be determined by the Euclidian distance to travel from one waypoint to another. Let us note C a MxM matrix containing the costs between waypoints: C_{ij} is the cost for moving from waypoint i to waypoint j. The total cost accumulated is noted C_o. For each pair of vertices (i, j), x_{ij} is equal to 1 if the edge from waypoint i to j belongs to the path, 0 otherwise. Let us note that x_{ij} is not necessarily the same as x_{ji}. The "path-planning" issue is presented as follows:

minimize

$$C_o = \sum_{(i,j)} C_{ij} \text{ for (i,j) in path}$$

subject to

$$\sum_{i=1}^{M} x_{ij} = 1 \quad \forall j \in \{1, ..., M\} \quad (1)$$

$$\sum_{j=1}^{M} x_{ij} = 1 \quad \forall i \in \{1, ..., M\} \quad (2)$$

The problem is to be tackled using two different types of algorithms. On the one hand, the Monte Carlo Tree Search Algorithm (MCTS) explained in the next section is used as an online (i.e. on the fly) planning algorithm. On the other hand, an offline state-of-the-art algorithm will be implemented as a comparison and the choice was made to use an Ants Colony Optimization Algorithm (ACO) [4].

B. Online path planning: Monte Carlo Tree Search Algorithm

The Monte Carlo Tree Search, as described in [3], is an algorithm that seeks to find the most rewarding move regarding a given goal at a given time and a given state in a mission or game. It is based on a search tree with different arborescence that will grow and evolve as it is used. The MCTS algorithm is similar to the minimax algorithm but is less time consuming.

The minimax algorithm considers, at each state, all the different states that can be reached from the current state. As a result, absolutely all scenarios are considered and the algorithm is sure to find a suitable action. However, projecting dozens of steps into the future while considering all possible scenarios at each step can quickly increase the memory used and the time needed to converge. The MCTS algorithm solves this problem by not considering all the possible scenarios at each step but only a few: first by chance, then by preferentially selecting the scenarios that have given a high reward, and sometimes by preferring scenarios still unexplored. It balances exploitation and exploration and covers the most promising scenarios, thus saving computational resources. The tree is composed of leaves and branches (nodes and edges). Each leaf or node represents a game state and each branch or edge represents a transition between two game states by an action of

one of the players. The algorithm is used in two parts: training and exploiting.

The first part, training, is divided into 4 steps. The completion of all the 4 steps is often called a rollout. Each rollout will update the search tree and one can do as much rollout as desired. The number of rollouts will surely depend on how much available time one have before making a decision in the game.

1) The selection consists in selecting an unvisited node. The selection is done according to rules or a policy to be defined, random policy is also one of the possibilities.
2) The expansion consists in choosing a child node from the newly selected node, that is to say that the execution of an action in accordance with the rules of the game allows to obtain this child node from the node selected in step 1.
3) The simulation consists in executing a random game from the extended node found in step 2, (this could be done randomly or according to a policy to be defined) until reaching a final node (until the game reaches a final state).
4) Back propagation consists in back propagating the reward obtained at the end of the simulation through the nodes encountered. Each node then contains two information that are kept up to date: the average reward obtained starting from this node, and the number of times this node has been visited by the algorithm.

The second part is the exploiting part, it consists in using the updated tree from the training to make a decision in the game. Considering the state of the game we are in (the node), we have to make a decision (make a move in the game according to the rules), which is to say select which child node to choose according to the updated tree. The node is chosen by maximizing

$$a = r + c * \sqrt{(\frac{ln(N)}{n})}$$

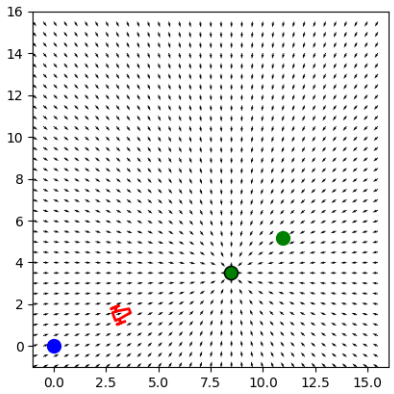
where r is the average reward among all simulations obtained starting from this node, N is the number of times the parent node has been visited, n the number of times the considered node has been visited and c is an exploration parameter which gives weight to unvisited nodes to encourage exploration. This last coefficient is meant to be chosen empirically. In brief, the Monte Carlo Tree Search is a method that uses a search tree that allows to forecast the most likely scenarios from a given state and use it to find the most promising moves towards a given objective.

C. Simulations and results

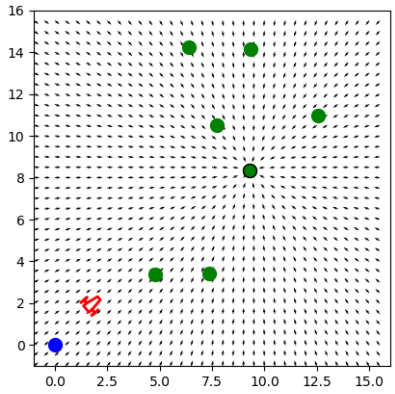
Simulations were made using ROS and Python and consisted in a Dubin's vehicle moving to specific targets one by one using artificial potential fields. The vehicle comes back to its original position after all targets were visited. Targets were chosen by a planner that would either use an ACO algorithm at the beginning of the run or a MCTS algorithm used on the fly whenever a target has been reached by the vehicle. The simulations are made available on a Github [1] repository.

[1]https://github.com/MourtazaKASSAMALY/initiation_to_research

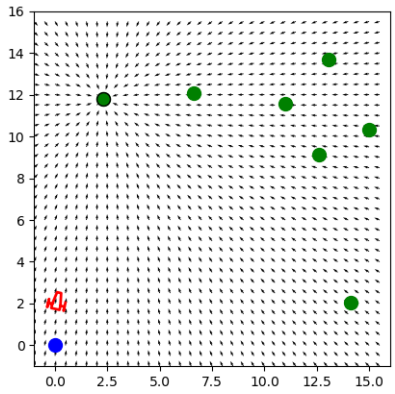
The field is $15x15$ meters and hosts multiple scenarios. A scenario consists in a number of targets that varies from 2 to 10 and each scenario has been repeated in 30 different configurations or layouts, as shown by a few examples in Fig 1. To compare performance, the metric used is the travelled distance by the vehicle for collecting all targets and coming back to the original position. Results of the simulations are shown in Fig 2.



(a) A scenario with 2 targets



(b) A scenario with 7 targets



(c) A different configuration for 7 targets

Fig. 1: Overview of ROS Simulations

Performance are similar with a few number of targets to reach but ACO algorithm rapidly outperforms MCTS algorithm for a higher number of targets. With 10 targets to collect, the vehicle travelled 48 meters using ACO planning while MCTS planning made it travel 53 meters. The gap could just grow higher as the number of targets increases. As explained

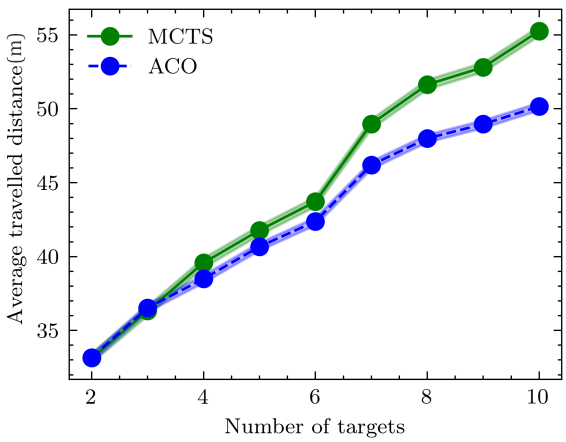


Fig. 2: Average travelled distance vs. number of targets

in the previous sections, the MCTS algorithms grows a search tree to look for promising moves using rollouts. The number of rollouts is surely a critical factor for assessing its efficiency and mostly depends on the computational ressources and time available for making a decision. To simulate online (i.e. almost real-time) path-planning, the choice was made to run only 10 rollouts after each target reached to compute the next one. The outcome would be different if this number was higher but again, it is related to one's requirements.

III. Conclusion

This short comparison shows that state-of-the-art offline algorithms performs better than online algorithms for path-planning to deal with the Traveling Salesman Problem. Digging into online algorithms is still worth when dealing with situations derived from Dynamic Traveling Salesman Problem where targets are non-static and obstructions can appear. Indeed, online algorithms certainly outputs a less optimal answer but they are rapidly executed and can easier adapt to changing configurations. A large amount of work has been made on adapting state-of-the-art offline algorithms to cope with dynamic constraints such as in [6] where a strategy is explained to reuse previous ACO results to recompute a path on the fly without starting again from scratch. As for the MCTS algorithm, improvements can be made in the choice of selection and simulation policies that are just greedy in this study. Significant progress can be achieved by building policies using Deep Reinforcement Learning and by splitting the workload between multiple agents in a decentralized version of the MCTS Algorithm [7] [8].

Acknowledgment

I would like to thank M. Luc JAULIN, professor in mobile robotics at ENSTA Bretagne and a researcher at Lab STICC in Brest (France) in the fields of assembly computing and marine robots.

References

[1] K. Ilavarasi and K. S. Joseph, "Variants of travelling salesman problem: A survey," International Conference on Information Communication and Embedded Systems (ICICES2014), Chennai, India, 2014, pp. 1-7, doi: 10.1109/ICICES.2014.7033850.

[2] G. Ghiani, G. Laporte, and F. Semet, "The Black and White Traveling Salesman Problem," Operations Research, vol. 54, no. 2, pp. 366–378, Apr. 2006, doi: 10.1287/opre.1050.0218.

[3] M. C. Fu, "MONTE CARLO TREE SEARCH: A TUTORIAL," 2018 Winter Simulation Conference (WSC), Gothenburg, Sweden, 2018, pp. 222-236, doi: 10.1109/WSC.2018.8632344.

[4] Chen, Mingzhi and Daqi Zhu. "Data collection from underwater acoustic sensor networks based on optimization algorithms." Computing 102 (2019): 83-104.

[5] KASSAMALY Mourtaza, Initiation to Research, (2021), GitHub repository

[6] Y. Cao, X. Hu and J. Zhang, "Dynamic swarm intelligence algorithms with reuse strategy for dynamic traveling salesman problem," 2017 Seventh International Conference on Information Science and Technology (ICIST), Da Nang, 2017, pp. 169-176, doi: 10.1109/ICIST.2017.7926751.

[7] M. Daneshvaramoli et al., "Decentralized Communication-less Multi-Agent Task Assignment with Cooperative Monte-Carlo Tree Search," 2020 6th International Conference on Control, Automation and Robotics (ICCAR), Singapore, 2020, pp. 612-616, doi: 10.1109/ICCAR49639.2020.9108073.

[8] Czechowski, Aleksander Oliehoek, Frans. (2020). Decentralized MCTS via Learned Teammate Models.

Modeling and State Estimation by Neural Networks

Corentin Lemoine

***Abstract*—Neural networks can be very efficient to approximate nonlinear functions with a large number of parameters. We may need to approximate such functions to model and estimate the state of a robot in its environment. Machine learning of neural networks allows to take into account all the factors influencing the robot's behavior, without making assumptions on the shape of these factors.**

***Index Terms*—Neural networks, modeling, state estimation.**

I. Introduction

MACHINE learning techniques based on neural networks have been greatly developed in recent years. Many fields benefit from these advances, and we can easily imagine how to adapt some of these techniques in the context of mobile robotics. Among the domains with which we can draw parallels, we can mention language learning or time series prediction. In this paper, we will propose an adaptation of these technologies to predict the state of a robot.

II. State of the art

A. Réseaux feedforward

A *feedforward* neural network is a network without cycles, in which the information propagates only in one direction. This type of network has no memory of previous inputs.

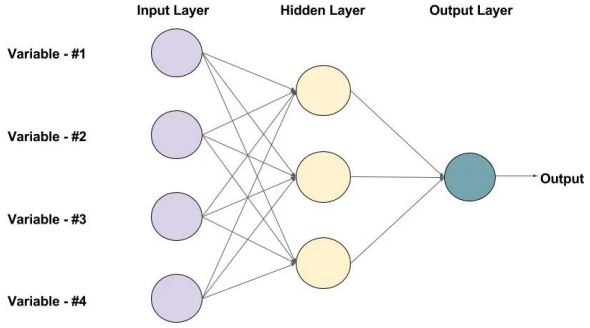


Fig. 1. Feedforward Neural Network

One method of using this type of network is the *NARX*[1] architecture. We suppose in this case that the function we try to estimate is of the form :

$y(t) = f(y(t-1), y(t-2), \ldots, y(t-n_y), u(t-1), u(t-2), \ldots, u(t-n_u))$

Where n_f and n_y are constants. We then construct the input vector of the network :

$(y(t-1), y(t-2), \ldots, y(t-n_y), u(t-1), u(t-2), \ldots, u(t-n_u))$

and we train our *feedforward* network how to estimate the f function.

We can then use the *NARX* network :

- in open loop for prediction on a single time step
- in closed loop for prediction on several time steps

To obtain the closed-loop network, it is sufficient for each time step to reconstruct the input vector with the output of our model for the previous time step.

B. Recurrent Neural Networks

In recurrent neural networks, the network has a memory of the data it has received; it therefore forms an output consistent with a sequence of inputs. This type of network is therefore widely used in the context of predicting chronologically ordered data.

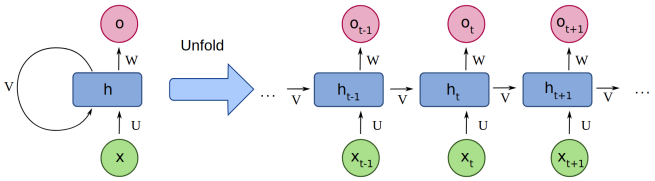


Fig. 2. Réseau de neurones récurrent [2]

Among the classical recurrent neural networks, we can quote the networks based on cells *LSTM*[3], or cells *GRU*[4]. These models allow to get rid of the problem of gradient disappearance [5]. This classical problem of recurrent networks occurs during the computation of the gradient of the cost function, which loses very quickly the memory of the past data. It then becomes very difficult, if not impossible, to perform the learning. The network cannot then extract any link between the data.

The cells *LSTM* for *Long Short-Term Memory* have been conceptualized since 1995. They are used in the context of language learning, or in the prediction of stock market values. It is this type of network that will be implemented here for the prediction of a robot's state.

C. Attention mechanisms

Very often, we have a large amount of input data which makes the learning phase very slow. A solution often proposed to reduce the size of the input data is the use of an attention-based mechanism[6]: a second neural network is trained to extract better descriptors of the state of a system.

This second network can be trained in a fully supervised way, which implies to manually choose the adapted descriptors, or in an automatic way by using an *autoencoder*[7][8]. The number of descriptors is then reduced to an arbitrarily chosen number.

III. Implementation

A. Choice of data

The choice of data is the first step in the implementation of such a solution. Indeed, it is necessary to choose what will be the input data of our neural network and what will be the output data. In order to have a model that can be run in a closed loop, it is necessary that the output $y(t)$ can enter the inputs at time $t+1$. We therefore keep the architecture proposed by the network *NARX* :

$y(t) = f(y(t-1), y(t-2), \ldots, y(t-n_y), u(t-1), u(t-2), \ldots, u(t-n_u))$

$y(t)$ is therefore our state vector at time t and $u(t)$ our command. By nature, the neural network will not be able to extrapolate its model outside the domain of the data on which it has been trained. For example, it will be more difficult to get the (x, y) position of a system from data allowing to calculate its speed, than to predict its speed and then integrate it to get its position. The "black box" aspect of the neural network must be limited as much as possible to obtain the best possible results.

For this implementation, we will use data from a quadricopter drone flight. Our state vector will include the filtered x, y and z axis output of the gyroscope.

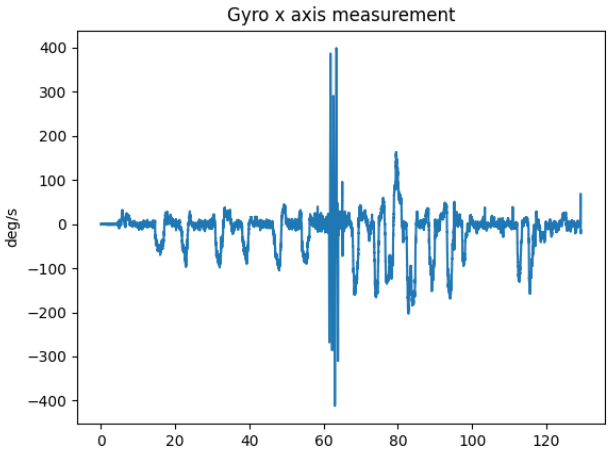


Fig. 3. y_1

The command will be the command applied to the motors.

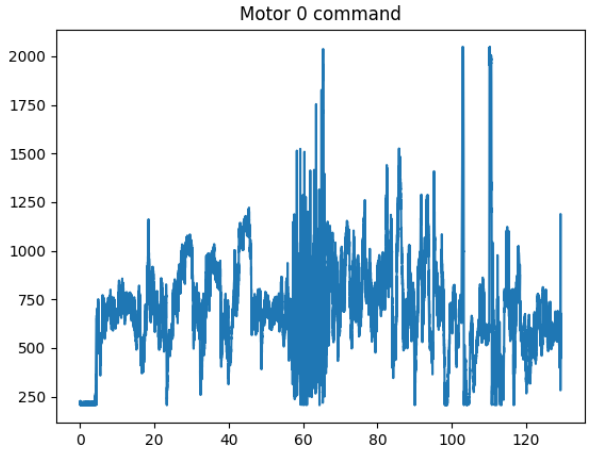


Fig. 4. u_1

B. Data preprocessing

Using a neural network involves preprocessing the data so that the training data has a mean of zero and a standard deviation of 1. This transformation must therefore be kept in mind to apply to the new data, and reversed to obtain consistent output data.

As with the choice of data above, it is always best to keep only the most significant data. Too much data will only noisy the input of the network and decrease the performance of the neural network. The raw dataset used comes from a 2000 Hz sampling of the data of a flight controller; not only is this measurement oversampled in view of the dynamics of the system, but this amount of data implies that it is not possible to keep a large memory of the past for a simple limitation of memory and computing power. Here we choose to subsample to 200 Hz. So we have about 2 minutes of data, or 15453 training samples after separation into training, test and validation data.

Finally, even if it doesn't apply here, it is classical in data sets to find sequences that are more adapted to the network in a certain form: for example, the heading of a vehicle will be much better apprehended in the form of a two-component direction vector than in its raw form in order to avoid a jump between $-\pi$ and π.

C. Building the Neural Network

We choose n_y and n_u arbitrarily to correspond to the dynamics of our system. Here we will use 200 samples which will represent 1 second.

We will experiment with 2 architectures: a network *NARX* and a cell-based network *LSTM*.

D. Training and prediction

1) NARX: The training of the network *NARX* is quite fast. We use the mean square error to measure the performance of the network, which would only represent the average error of our network after the inverse data preprocessing transformation.

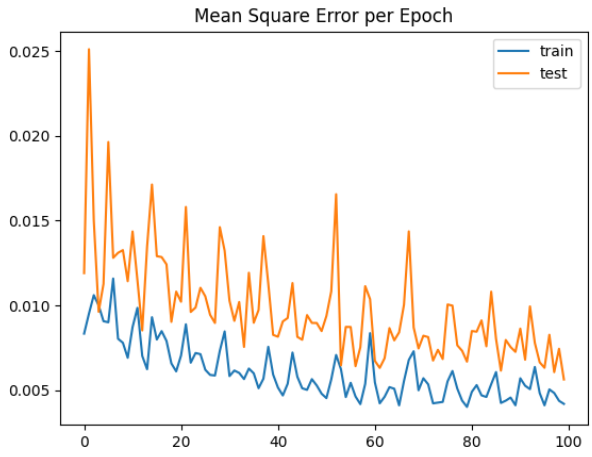


Fig. 5. MSE per epoch, NARX

This network allows us to quickly obtain a prediction at $t+1$.

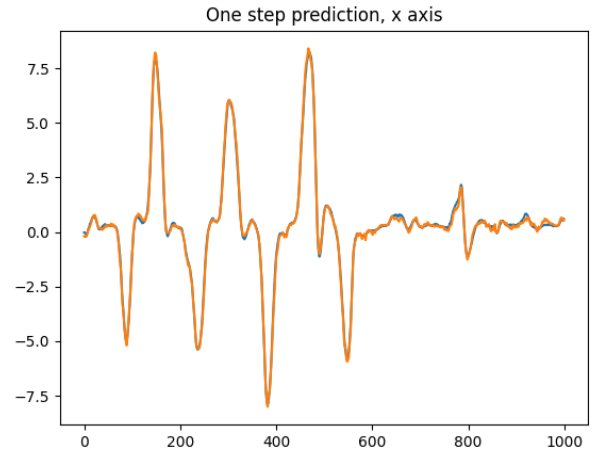


Fig. 6. Single time steps prediction, NARX

In the next figure, we fed our neural network with 200 consecutive time steps of our dataset. We then iteratively predict the next gyroscope measurement, stack this prediction with the actual command sent to the motors, and looped this new vector to the input of the network. By predicting on 200 time steps, i.e. on one second, we obtain the following figure:

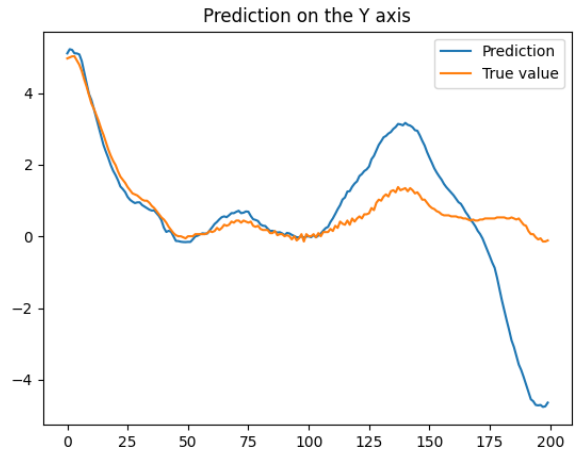


Fig. 7. Mutliple time steps prediction, NARX

We observe that the network is able to predict on only 100 samples before the prediction becomes unstable and the generated noise becomes too large.

2) LSTM: The LSTM network is harder to train; and appears to necessitate more data in order to not overfit. We can see that on the following figure :

Fig. 8. MSE per epoch, LSTM

The network is then able to predict very accurately on a single time step, but the model drifts very quickly when predicting on several time steps. The size of the data set is probably too small here.

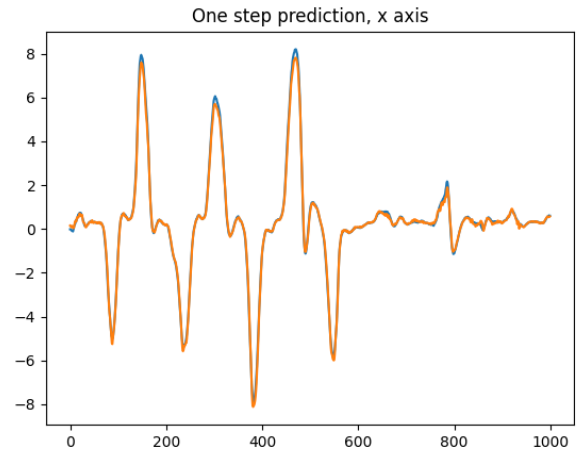


Fig. 9. Single time steps prediction, LSTM

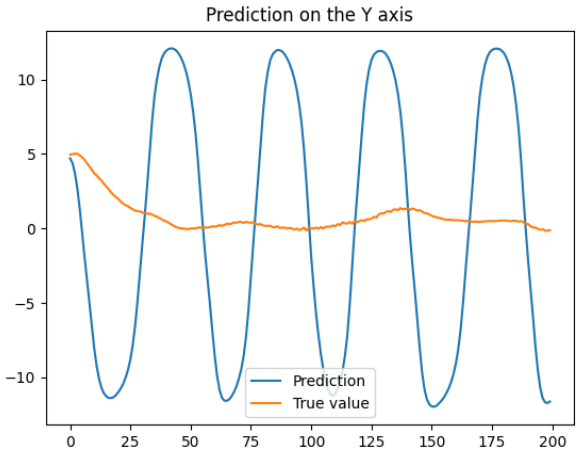


Fig. 10. Mutliple time steps prediction, LSTM

IV. Benefits and limitations

A. Dataset size

The main limitation we encounter is the dataset size : it is quite hard to find a dataset, and by essence we need the robot to be operational to collect data. This solution may be used to get better performances, but necessitate either a working platform or a good simulator.

The neural network should in theory get better with the data it acquires, and could be setup to learn continuously. The dataset size is a limitation because the initialization can be tricky.

B. Reliability

The neural network is a black box, and it is not possible to guarantee a range for its error. By measuring its mean squared error, we should be able to get accurate statistics though. A way to mitigate this limiting factor is to associate this system with a Kalman filter to be able to reject prediction that seems far off.

C. Adaptability

A strength of the neural network is its ability to adapt, and be trained continuously. A robot can then adapt to its wear over time, or to a change in its environment affecting it.

D. Predictive command

A direct application to the state predicting neural network is the predictive command: we can very quickly predict how a series of commands will affect our robot, and choose the best series to reach our goal.

V. Conclusion

To conclude, this technique can effectively predict the state of a system on several time steps. If here the number of data is quite low both in number of series and in length, we have shown that the NARX architecture allows to quickly reach results. Previous studies have shown the superiority of recurrent neural networks in the processing of large databases, which is promising to go further.

Appendix

Code is available on github as a jupyter notebook at https://github.com/Pafnouti/NeuralNetStatePrediction

References

[1] P. P. C. Yip and Yoh-Han Pao. A recurrent neural net approach to one-step ahead control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):678–683, 1994.
[2] fdeloche. Structure of rnn, 2017. [Online; accessed March 9, 2021].
[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
[4] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.
[5] Fakultit Informatik, Y. Bengio, Paolo Frasconi, and Jfirgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, 03 2003.
[6] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *CoRR*, abs/1704.02971, 2017.
[7] Hareesh Bahuleyan, Lili Mou, Olga Vechtomova, and Pascal Poupart. Variational attention for sequence-to-sequence models. *CoRR*, abs/1712.08207, 2017.
[8] J. Pereira and M. Silveira. Unsupervised anomaly detection in energy time series data using variational recurrent autoencoders with attention. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1275–1282, 2018.

Point-to-Point Navigation with Deep Reinforcement Learning

Paul-Antoine Le Tolguenec[1,✉]

[1]ENSTA Bretagne, Brest

Nowadays the problem of the autonomous vehicle is one of the most developed subjects in the world of research. Although there are different methods, methods based on deep learning are more and more used in this field. This article illustrates a solution based on Deep Reinforcement Learning.

Deep learning | Reinforcement | Control | Autonomous car

Correspondence: ***paul-antoine.le_tolguenec@ensta-bretagne.org***

Introduction

In recent years, mobile robotics has experienced unprecedented growth. Most of the methods concerning trajectory estimation are based on automatic laws. A very robust method is the linearizing loop control method. But in some cases this method has to be reworked, because it has to be linearised around a working point or other. Another very robust method is that of potential fields, since it also allows to avoid certain obstacles that could appear spontaneously on the way to the robot. But for this type of method it is often necessary to implement other layers that allow to find the path to reach a point in the defined space. Most of the time it is also necessary to develop a finite state machine to manage the high level of the robot. Usual methods are therefore very time-consuming to implement and can lead to mistakes. Moreover, once the control of the robot is set up, optimisation is difficult. With the progress of deep learning, a new control approach has emerged: deep reinforcement learning. The advantage of this method is that once the method is set up it can be generalised to many problems and it is only necessary to change certain parameters. Also this method is permanently optimized. In this article, I expose my work which has allowed me to train a model to make decisions to orientate a car so that it is able to go from point A to point B as quickly as possible by avoiding objects (2) that appear as they go along. To train the model I use the A2C (Advantage Actor Critic) algorithm which uses the principle of reinforcement learning. Part II illustrates the modelling I have carried out and within which I have trained the network. Part III presents the implementation of the method (network architecture, theory ...). Finally, part IV illustrates the experimentation phase.

Modelisation

A. Formalisme. The objective of our work is to automate a car so that it is able to go from point A to point B while avoiding obstacles. The kinematics of the car is modelled by the following equations:

$$\begin{cases} (i) & \dot{x} = v\cos\phi \\ (ii) & \dot{x} = v\sin\phi \\ (iii) & \phi = u_{\pi_\theta} \\ (iv) & v = k \end{cases} \quad \textbf{(1)}$$

At first, I tried to control only the steering angle of the car. Thus u_{π_θ} represents the ϕ output taken via a policy π_θ. The robot is also equipped with three sonars, one frontal and two lateral, which allow to visualize its environment.

B. Simulation. Once the kinematics were defined, I had to simulate the robot and obtain a visual rendering. I used the Kivy library of python. Kivy is a free and open source Python framework for developing mobile apps and other multitouch application software with a natural user interface. The objective was to have a visual rendering of the robot's evolution, but also to be able to interact with the environment via a man-machine interface. Once the simulation is launched, a window opens and the robot evolves in its environment. The operator can interact with the robot's environment by creating walls that he can draw with the mouse. There is also a button to erase all the walls on the map to rebuild others and see if the robot is able to generalise its behaviour in any type of environment.

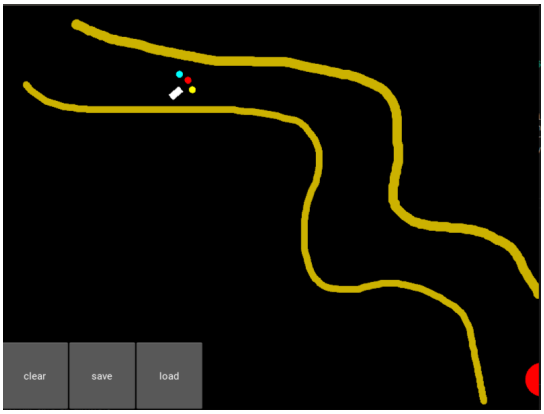


Fig. 1. Simulation of the system

As can be seen in the figure. The car is represented by a white square. The sonars of the car are represented by the three circles at the front of the car.

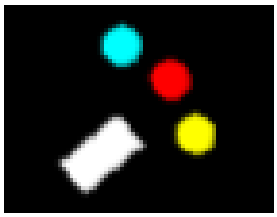


Fig. 2. Robot sonars

The walls are represented by the yellow lines. And the objective of the robot is represented by the red dot. At first, the robot can cross walls but when it does so its speed decreases and it gets a negative reward because it has crossed the wall and because it will arrive less quickly at its target.

Proposed method

The objective of our work is to train an agent to be able to choose the orientation angle of the car according to his perception of the environment.

C. network structure. The structure of the network is quite simple. The network takes as input: the distances output by the sonars, and the angle formed by the direction of the robot and the line passing through the centre of the robot and target. To simplify the teaching process, we have discretised the robot's exit space. The robot can make three decisions: do nothing, increment its angle by +20 degrees or -20 degrees. Of course, making the exit area discreet means poorer results in the long term. But it allows for a quicker convergence towards a more stable policy (1).

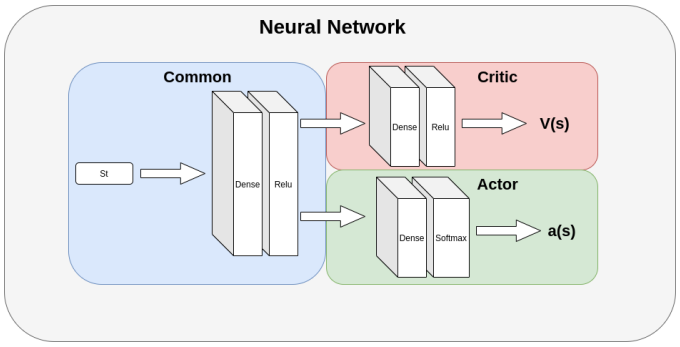


Fig. 3. Neural Network Structure

As you can see the network is divided into two parts(6):

- Critic

  The Critic part is used to determine the value of the state in which the agent is in. This part corresponds to the value-based part of the learning. It allows to stabilize the learning and to converge towards a global maximum of the policy.

- Actor

  The actor part corresponds to the part of the agent that takes action. It is the policy-based part of learning. The actor part pulls out a probability distribution $\pi_\theta(s)$ which makes it possible to determine what probability is for the agent to take this or that action given the state.

The advantage of such a method is that a continuous space of action can be used.(8) Unlike conventional Deep Q-Learning methods.(9) The problem is that algorithms such as the DDPG (Deep deterministic policy gradient) which provide a continuous policy space do not guarantee to find the optimal policy for a given network and MDP, which is the case for a stochastic critical actor. Moreover, they introduce many problems that make them difficult to converge.(4) That's why we discretize the space in this problem. Here he purpose of learning is to change the probability distribution in the direction that gives the maximum reward.

The value part allows to compare the quality of an action carried out in relation to the value of a state already estimated to know if this action is more optimised than what has been done before.

D. Advantage Actor Critic algorithm. To update the network parameters we used reinforcement learning. We used one of the most powerful reinforcement learning algorithms before the A3C. A2C is an algorithm halfway between value based learning and policy based learning.

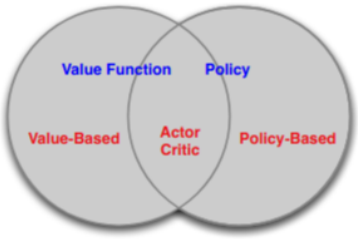


Fig. 4. Actor Critic

The Advantage Actor Critic algorithm takes the following form :

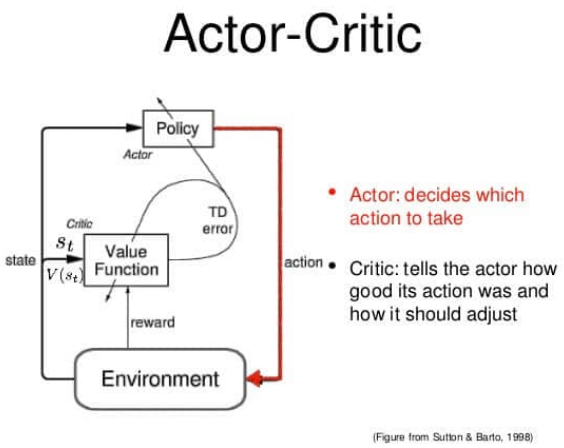


Fig. 5. Actor Critic

As can be seen in figure 4, the principle of the critical actor is

Algorithm 1 N-step Advantage Actor-Critic

procedure N-STEP ADVANTAGE ACTOR-CRITIC
  Start with policy model π_θ *and value model* V_ω
  repeat:
    Generate an episode $S_0, A_0, r_0, \ldots, S_{T-1}, A_{T-1}, r_{T-1}$
    for t *from* $T-1$ *to* 0:
      $V_{end} = 0$ if $(t+N \geq T)$ *else* $V_\omega(s_{t+N})$
      $R_t = \gamma^N V_{end} + \sum_{k=0}^{N-1} \gamma^k (r_{t+k}$ *if* $(t+k<T)$ *else* $0)$
    $L(\theta) = \frac{1}{T}\sum_{i=0}^{T-1}(R_t - V_\omega(S_t)) \log \pi_\theta(A_t|S_t)$
    $L(\omega) = \frac{1}{T}\sum_{i=0}^{T-1}(R_t - V_\omega(S_t))^2$
    Optimize π_θ *using* $\nabla L(\theta)$
    Optimize V_ω *using* $\nabla L(\omega)$
end procedure

to take an action thanks to the actor and to criticise the action achieved thanks to the critic. But to criticise this action you need something to compare this action with the critic. This is the Advantage part. There are several ways of expressing Advantage. But the most common method is to take as the advantage : $Rt - V_{\pi_\theta(s,a)}$

with $Rt = \gamma^N V_{end} + \sum_{k=0}^{N-1} \gamma^k (r_{t+k})$.

We have used the Monte-Carlo evaluation. The Monte-Carlo method involves letting an agent learn from the environment by interacting with it and collecting samples. This is equivalent to sampling from the probability distribution P(s, a, s') and R(s, a).

However, Monte-Carlo (MC) estimation is only for trial-based learning. In other words, an MDP without the P tuple can learn by trial-and-error, through many repetitions.

In this learning process, each "try" is called an episode, and all episodes must terminate. That is, the final state of the MDP should be reached. Values for each state are updated only based on final reward Rt, not on estimations of neighbor states — as occurs in the Bellman Optimality Equation.

MC learns from complete episodes and is therefore only suitable for what we call episodic MDP.

Here it's not the case, but we are going to use the same method to take into account more information.

As can be seen in figure 5, the advantage part of the algorithm can also be called the TD error.

In the literature one initially finds $Rt = \gamma V_{t+1} + r_t$ but recent work on replay experience has shown that taking into account more rewards (like the monte-carlo evaluation) makes it easier to converge towards a global minimum of the cost function estimating the MDP (Markovian decision process). On the other hand, using this method implies a certain volatility of the estimator since a stock affects the value of a state further in time, and it often takes longer to converge. It is therefore a question of finding the best hyper-parameters, which are often found empirically.

E. Reward process. In reinforcement learning methods the reward process is very important as it allows the robot to perceive its environment. More importantly, it corresponds to the MRP (Markovian reward process) that we try to estimate thanks to the neural network (the critical part). So if the reward process does not correspond exactly to the environment, the agent will not correctly estimate the behaviour it has to adopt.

$$\begin{cases} R_t = -2, \text{ if } None(\text{life penalty}) \\ R_t = +1, \text{ if } distance_t < distance_{t-1} \\ R_t = -5, \text{ if } x, y \in \Omega_{sand} \end{cases} \quad \textbf{(2)}$$

The life penalty, allows the agent to get out of certain situations such as when he comes into contact with a wall and gets stuck. If a negative life penalty is not applied, the robot may get stuck in its situation.

Experimental discussion and results

The neural network used is quite simple so the learning process is not long but the final results can be improved. To evaluate the quality of a reinforcement learning project, it is not enough to look at the final quality of the model. It is necessary to evaluate the training time, the response time of the system, and to determine on parameters such as the learning curve if the model can still learn or if it is not the case.

Learning. It is difficult to visualize neural network learning for reinforcement learning problems. Especially when using a critical actor. In fact, if the error on the critic decreases it does not necessarily mean that the actor learns correctly. But it is really the actor who interests us. The error on the actor can be interpreted but it does not allow us to visualize the quality of the policy used by the agent at a given moment t. To visualize the evolution of an agent in his environment, we must take into account the number of rewards he acquires over time.

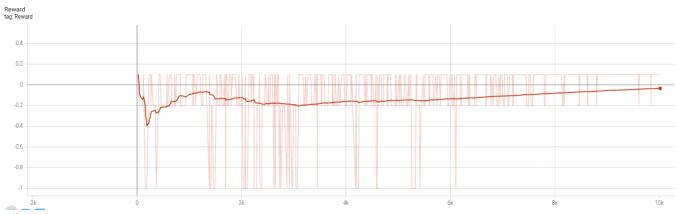


Fig. 6. Rewards curve

As can be seen on the reward curve the model converges to an acceptable policy after 10K iterations, which is about 5 minutes. We can consider that the model learned very quickly. The environment is quite simple since at first we don't use walls.

Obstacle avoidance. Our initial goal was to create an agent capable of avoiding all the obstacles he could find. With the chosen network and the established configuration our agent is able to reach his target avoiding most of the obstacles.

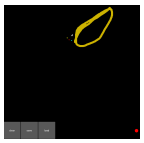


Fig. 7. Avoiding one Obstacle

Even following a path.

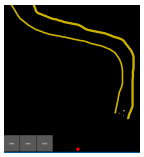


Fig. 8. Avoiding one Obstacle

However in some configurations the robot is sometimes forced to cross the wall to reach the target.

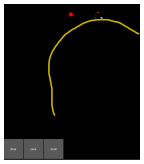


Fig. 9. The U problem.

So if the model presents very good results, it is still perfectible. To check the results go to : Actor-Critic.

Prospects for improvement. As the results obtained are the result of a detailed search for the best hyper-parameters of the model (learning rate, gamma, etc.), there are three ways of improving the agent's performance:

- **Building a deeper network** One of the possible reasons is that the network that has been built is not sufficient to find the most optimal policy in this Markovian decision process problem. One of the solutions is therefore to create a deeper network and thus to add layers of neurons.
- **Add an lstm layer** LSTM (7) are recurrent neural networks. They make it possible to store information in memory. So in case the robot is stuck in a corner, it would be able to know where it comes from. For the moment it only goes with what it sees. Keeping information in memory could probably optimise the robot's performance. However, using this type of network leads to problems that can be difficult to solve, such as the vanishing gradient.(3)
- **Implementing another algorithm** As we have seen, there are several ways to train a neural network to find the CDM. But one of the most optimal methods is the A3C (5) algorithm. (asynchronous advantage actor-critic) This method maximizes the exploration of several agents communicating with each other to find the best policy.

Conclusion

Our method makes it possible to approximate the CDM and find an acceptable policy. However, the model can still learn. For the moment a more conventional method such as a vector field would work without any doubt. But the advantage of reinforcement learning methods is that it can be applied to a totally different problem without changing the agent structure and therefore the method. This is not the case with conventional methods where each problem introduces a new method.

1. Bhatnagar, S., R. S. Sutton, M. Ghavamzadeh, and M. Lee (2009) "Natural actor–critic algorithms," *Automatica* 45(11), 2471–2482, ISSN 0005-1098
2. Evans, B., H. W. Jordaan, and H. A. Engelbrecht (2021) "Autonomous obstacle avoidance by learning policies for reference modification,"
3. Hu, Y., A. E. G. Huber, J. Anumula, and S. Liu (2018) "Overcoming the vanishing gradient problem in plain recurrent networks," *CoRR* abs/1801.06105
4. Matheron, G., N. Perrin, and O. Sigaud (2019) "The problem with DDPG: understanding failures in deterministic environments with sparse rewards," *CoRR* abs/1911.11679
5. Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016) "Asynchronous methods for deep reinforcement learning,"
6. SC., W. (2003) *Artificial Neural Network*, The Springer International Series in Engineering and Computer Science,
7. Staudemeyer, R. C., and E. R. Morris (2019) "Understanding LSTM - a tutorial into long short-term memory recurrent neural networks," *CoRR* abs/1909.09586
8. van Hasselt, H., and M. A. Wiering (2007) "Reinforcement learning in continuous action spaces," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 272–279
9. Yang, Z., Y. Xie, and Z. Wang (2019) "A theoretical analysis of deep q-learning," *CoRR* abs/1901.00137

Autonomous navigation of an underwater robot (AUV) by genetic algorithm

Paul Pineau
Brest, France
Email: paul.pineau@ensta-bretagne.org

***Abstract*—There are multiple methods to navigate an AUV independently. Creating an autonomous mission controller and planner with FSM is of several interest for underwater missions. First of all the architecture is algorithmically simple. The positioning of an AUV is also problematic and this method of navigation offers a lot of room for maneuver if the robot is lost. This paper discusses how to find the optimal headings for the state machine in order to quickly validate the waypoints. For this purpose a genetic algorithm is used in order to optimize by a few iterations the optimal headings.**

Keywords: FSM, AUV, Genetic algorithm, autonomous

I. Introduction

The ocean remains an extremely unknown environment for man because it is dangerous and difficult to explore. If before one could study the surface with boats relatively easily, studying the seabed was (and remains) extremely complex. Several technological innovations have gradually allowed to replace divers by robots. The first were the ROV (Remotely Operated Vehicles) which made it possible to remove the human risk but being linked by a wire to an operator the missions are thus limited, in time as well as in space. This is why AUVs (Autonomous Underwater Vehicles) offer many advantages for underwater exploration. Their range is infinite (using the battery) and they can take readings on their own. However, the difficulty lies in the positioning of the robot that does not have access to GPS underwater. In a lake a robot often has to make a path between different waypoints. Using a finite state machine has many advantages, especially on the algorithmic complexity of the controller that results from it. However finding this controller can be complicated. In this topic, the goal is to find the optimal FSM for the path of the AUV through a genetic algorithm.

The purpose of the state machine is to "bounce" the AUV on isobaths. On a nautical chart, an isobath is a line joining points of equal depth. It is thus a curve of level, indicating the depth of a surface below the water level. At each isobath detection the AUV takes a new course in a predefined list. The goal is to find this list of heading in order to validate in an optimal way the waypoints given by the user at the beginning of the mission.

II. Genetic Programming for headings optimisation

The genetic programming method consist of iterate a list of data in order to find the optimal value. In this cas the data are a list of headings. For the algorithm[1] we need an initial population called M_0, composed of N members denoted $m_k, k \in [1, N]$. We will iterate this population until we find at least one member that satisfies our needs. Then we define $f(m_k)$ the fitness function that will define the quality of the member, in our case it will be the travel time. The principle of the algorithm is to keep the best members and combine them in order to improve the result in the image of the theory of evolution. We therefore define two operations, a unary one, the mutation that we will note $g_m(m_k)$ and another binary one, the crossover that we will note $g_c(m_k)$. We can then apply the genetic algorithm[1].

Algorithm 1 Mutation(m = [$h_1, h_2, .., h_n$])

Require: $k \leftarrow randint(1, N)$
Require: $p \leftarrow random(0, 1)$
if p > 0.5 **then**
    $m[k] \leftarrow m[k] + 20$
else
    $m[k] \leftarrow m[k] - 20$
end if

Algorithm 2 Calculate m = $g_c(m1 = [h_{11}, h_{12}, .., h_{1n}], m2 = [h_{21}, h_{22}, .., h_{2n}])$

Ensure: m1 and m2 are sorted
for $iteration = 1, 2, k \ldots$ **do**
    $m[k] \leftarrow 1\frac{}{2}.(m1[k] + m2[k])$
end for

A. Fitness Function

So we have $f(m_k)$ the fitness function. $f(m_k)$ calculates the time taken by the submarine to complete the route delimited by the waypoints. Thanks to an Euler method, the time taken by the submarine for each m_k of the population M is calculated. We need to check if all waypoints are validated. Let define $v(w_k)$ for $w_k \in W$, with W the list of waypoints. $v(w_k)$ return True if the waypoint is validated, false otherwise. A matrix M is then defined with $M(i, j) = 1$ if waypoint j is validated starting from i. We then have that all the waypoints are validated if $\forall i, \sum_j M(i, j) > 2$ and $\forall j, \sum_i M(i, j) > 2$ which means that at least the AUV came to the waypoint and

leave. For example, if W = ([0,0],[10,0],[0,10]), we have M that can look like this :

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

Here, the path is w1 to w2, w2 to w3 and finally w3 to w1.

B. The algorithm

Let define M the population,$PopSize$ the size, τ the probability to choose the crossover operation instead of the mutation, N_{max} the maximum iteration of the algorithm.

Algorithm 3 Compute optimal headings h

Require: $globalbest \leftarrow \inf$
Require: $bestheading \leftarrow []$
for $iteration = 1, \ldots, N_{max}$ **do**
  $globalbest \leftarrow$ best score in the population
  $bestheading \leftarrow$ the associate population member
Require: M is sorted with the fitness
  for $1, \ldots, k, \ldots, PopSize$ **do**
    if $random(0,1) > \tau$ **then**
      $M[k] \leftarrow mutation(bestMembers)$
    else
      $M[k] \leftarrow crossover(bestMembers)$

Where $bestMembers$ are the best members of the population at that time according to the fitness function.

III. Python simulation

I chose to implement the algorithm in python. To do so, I started by modeling the physics of a riptide submarine [2][3]. Then I had to apply a controller via FSM[4] (Finite State Machine). The principle is to measure a distance to the bottom that will have been modeled beforehand is to change state each time the $k-meter$ isobath is reached. I choose to use only a list of three headings.

A. FSM

The FSM look like this in my case.

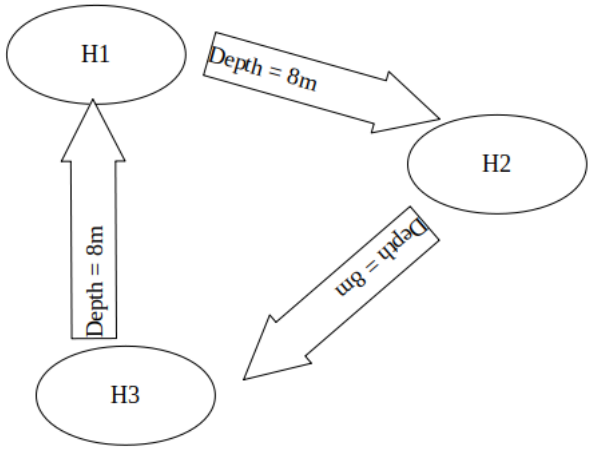


Fig. 1: FSM

There is one problem we need to take into account. When the AUV detect the isobath, he change heading and a few moment later detect almost the same isobath so we need to create special state. For example for the state $H1$ corresponding to heading 1. We need to have $H1in$ and $H1out$. the out state is useful for the transition and when we detect an isobath, we change state to the out one and when it's detected another time we change to the in state. With this tip there is no problem when the depth is rising or falling.

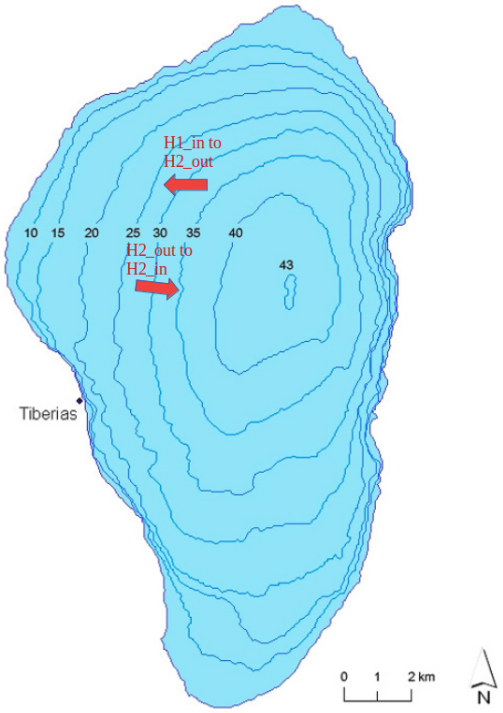


Fig. 2: In and Out States, source:"https://educalingo.com"

IV. Resulats

The algorithm makes it possible to find by having as waypoints $[[10, 20, -2], [20, 20, -2], [20, 10, -2]]$, with as background a giant bowl, optimal headings worth: $[1.7856, 2.9496, 5.0813]$ for an optimal time of 5.3.

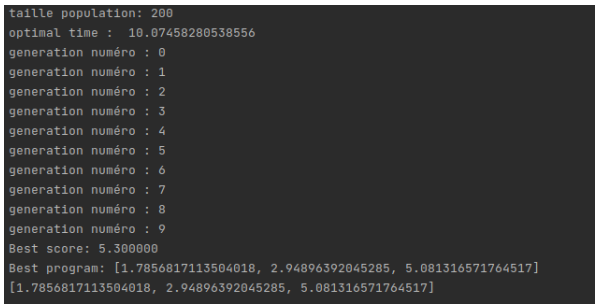


Fig. 3: Results

References

[1] Koza, J., 1998. Genetic programming 1998. San Francisco: M. Kaufmann Publishers

[2] Thor I Fossen.Handbook of marine craft hydrodynamics and motion control. John Wiley Sons, 2011

[3] Luc Jaulin.Mobile robotics. John Wiley Sons, 2019

[4] Timothy Kam, Tiziano Villa, Robert K. Brayton et Alberto Sangiovanni-Vincentelli, Synthesis of Finite State Machines : Functional Optimization, Springer, 1996, 282 p.

Swarm Robotics

Julien Piranda
Brest, France
Email: julien.piranda@ensta-bretagne.org

***Abstract*—As an emerging field of research in which swarm intelligence is applied to multi-robot systems, swarm robotics (a very specific sub-domain of collective robotics) studies how to coordinate large groups of relatively simple robots through the use of local rules. It focuses on the study of the design of a large number of relatively simple robots, their physical bodies and their control behaviours. We will focus on the phenomenon by which robots move by themselves, using only limited environmental information and simple rules.**

Keywords: Robotic, AUV, Swarm robotics, autonomous

I. Introduction

Inspired by the complex behaviours observed in natural swarm systems (e.g. social insects and live animals in order), swarm intelligence is a new field that aims to build fully distributed decentralised systems in which the overall functionality of the system emerges from the interaction of individual agents with each other and with their environment. In order to apply the knowledge gained in this research area to multi-robotics, a new research field called "swarm robotics" is being created [1]. Swarm robotics (SR) is the study of how to coordinate large groups of relatively simple robots using local rules. It focuses on the study of the design of a large number of relatively simple robots, their physical bodies and their control behaviour. Swarm robotics is closely related to the idea of swarm intelligence and shares its interest in self-organised decentralised systems.

II. Swarm Intelligence

Based on simple but collectively complex individual behaviour natural grouping systems including social insects such as colonies of ants, termites, bees, wasps, etc. [2], and high-level live animals such as flocks of birds, schools of fish and packs of wolves, etc. are examples of collective intelligence. Inspired by the robustness, extensible and principles of distributed self-organisation observed in these astonishingly complex natural collective behaviours resulting from simple and individual local interaction rules, an attempt to apply the knowledge gained from this research to artificial systems (e.g. massively distributed computer and robotic systems) has given rise to a new research topic called swarm intelligence [3].

A. Overview

This growing field of research, which was first introduced in the context of cellular robotic systems by Beni and Wang, is considered a sub-field of artificial intelligence based on the study of collective behaviour in decentralised and self-organised systems [4]. Although there is no specific definition of swarm intelligence, we adopt the heir to that of Dorigo Birattari: "The discipline that deals with natural and artificial systems composed of many individuals who coordinate using decentralised control and self-organisation. In particular, the discipline focuses on collective behaviours that result from the local interactions of individuals with each other and with their environment". Thus, a swarm intelligence system typically consists of a population of relatively simple agents that interact only locally with themselves and their environment, without having a global knowledge of their own state and the state of the world. Moreover, the observed global behaviour is in response to the local environment and local interactions between agents that follow often very simple rules.

B. Existing Algorithms

Theories based on natural swarms have been applied to solve similar engineering problems in several fields: engineering, from combinatorial optimisation to the rooting of communication networks, via robotic applications, etc [5]. The best-known swarm-based algorithms are: ant colony optimisation algorithms (ACO), particle swarm optimisation algorithms (PSO), artificial fish swarm algorithm (AFSA) and bee-based algorithms. The ACO algorithm is based on the foraging behaviour of ant colonies to find the shortest routes between their nests and food sources. The AFSO algorithm is based on the collective movement observed in different fish behaviours such as foraging, tracking other fish, protecting the group from hazards and stochastic searching.

III. Swarm simulation

There are therefore several complex algorithms developed for swarm robotics. In this project, we will try to implement swarm robotics inspired on the collective behavior. We will draw inspiration from the behaviour of animals, especially birds. Indeed, there are algorithms which simulate the flocking behaviour of birds. We will use them in order to simulate the basic movements of our swarm robots.

A. Boids

Boids is an artificial life program, developed by Craig Reynolds in 1986, which simulates the behaviour of birds in flight. His paper on the subject was published in 1987 in the proceedings of the ACM SIGGRAPH conference. 1] The name "boid" corresponds to an abbreviated version of "bird-oid object", which refers to a bird-like object. By the way, "boid" is also a pronunciation of the New York Metropolitan dialect for "bird".

B. Principe

Boids is an example of emergent behavior, the complexity of Boids arises from the interaction of individual agents adhering to a set of simple rules [6]. In our case, it will allow to define the basic movements of our robots.
The main rules are :

1. Separation
Each boid also tries to avoid running into the other boids. If it gets too close to another boid it will steer away from it. You can control how quickly it steers with the "separation" slider.

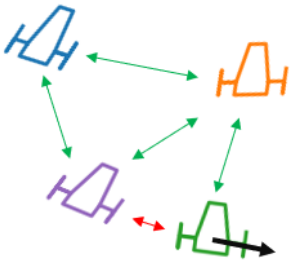


Fig. 1: Separation rule

2. Alignment
Finally, each boid tries to match the vector (speed and direction) of the other boids around it. Again, you can control how quickly they try to match vectors using the "coherence" slider.

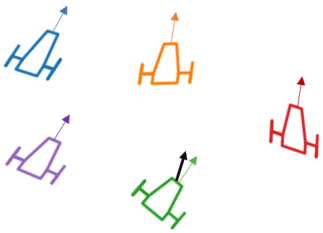


Fig. 2: Alignment rule

3. Cohesion
Each boid flies towards the the other boids. But they don't just immediately fly directly at each other. They gradually steer towards each other at a rate that you can adjust with the "coherence" slider.

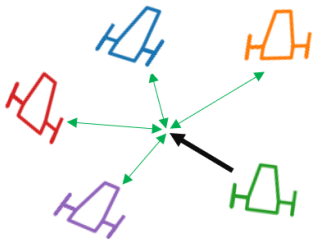


Fig. 3: Cohesion rule

IV. Our Model

Thus, we can represent how our robots will move during the mission. We will now establish the intelligence of the swarm. For that, we will try to implement swarm robotics inspired on the collective behavior of fish schools. Indeed, in this project our robots are underwater and their collective behaviour could be represent by a fish swarm.

A. Fish School Search

The Fish School Search (FSS), created by Bastos Filho and Lima Neto in 2007, is, in its basic version, a unimodal optimisation algorithm inspired by the collective behaviour of fish schools [7]. The FSS is directly inspired by the feeding and coordinated movement mechanism of schools of fish to create coordination between the robots. The basic idea is to make the fish "swim" towards the positive gradient in order to "eat" and "gain weight". Collectively, the heavier fish have more influence on the overall search process, causing the centre of gravity of fish school to move to better locations in the search space over iterations [8].

B. Overview of FSS Algorithms

1. Individual component of the movement
Each fish in the school carries out a local search for promising regions in the research area. This research is carried out as follows

Let $x_i(t) \in R^3$, the position of the i^{th} fish at time t, where $t \in R$ and $i \in N$

$$x_i(t+1) = x_i(t) + rand(-1,1)step_{ind} \tag{1}$$

where $x_i(t)$, $x_i(t+1)$ represent the position of the fish i before and after the individual movement operator, respectively. $rand(1,1)$ is a uniformly distributed random number varying from -1 up to 1 and $step_{ind}$ is a parameter that defines the maximum displacement for this movement.
The new position $x_i(t+1)$ is only accepted if the fitness of the fish improves with the position change. If it is not the case, the fish remains in the same position and $x_i(t+1) = x_i(t)$.

2. Collective-instinctive component of the movement
The model takes into account the trend of the group by calculating the average of the individual movements.
Let consider I a vector which represents the weighted average of the displacements of each fish.

$$I = \frac{\sum_{i=1}^{N} \Delta x_i \Delta f_i}{\sum_{i=1}^{N} \Delta f_i} \tag{2}$$

This will attract the fish that have improved more will attract the fish in its position. Thus, every fish will be encouraged to move according to : $x_i(t+1) = x_i(t) + I$.

3. Collective-volitive component of the movement
Let consider $B \in R^3$ the barycenter which are calculated

based on the position x_i and weight W_i of each fish with the followed relation :

$$B(t) = \frac{\sum_{i=1}^{N} x_i(t) W_i(t)}{\sum_{i=1}^{N} W_i(t)} \tag{3}$$

This operator is used in order to regulate the exploration/exploitation ability of the school during the search process.
Then, there are two possibilities :

Case 1 : if the total school weight $\sum_{i=1}^{N} W_i$ *has increased*

The fishes are attracted to the barycenter according to the equation :

$$x_i(t+1) = x_i(t) - step_{vol} rand(0,1) \frac{x_i(t) - B(t)}{distance(x_i(t), B(t))}$$

Case 2 : if the total school weight $\sum_{i=1}^{N} W_i$ *has decreased*

$$x_i(t+1) = x_i(t) + step_{vol} rand(0,1) \frac{x_i(t) - B(t)}{distance(x_i(t), B(t))}$$

where $step_{vol}$ defines the size of the maximum displacement performed with the use of this operator. $distance(x_i(t), B(t))$ is the euclidean distance between the fish i position and the school barycenter. $rand(0,1)$ is a uniformly distributed random number varying from 0 up to 1.
Besides the movement operators, it was also defined a feeding operator used in order to update the weights of every fish according to

$$W_i(t+1) = W_i(t) + \frac{\Delta f_i}{max(|\Delta f_i|)} \tag{4}$$

where $W_i(t)$ is the weight parameter for fish i, Δf_i is the fitness variation between the last and the new position, and $max(|\Delta f_i|)$ represents the maximum absolute value of the fitness variation among all the fishes in the school. W is only allowed to vary from 1 up to $W_{scale}/2$, which is a user defined attribute. The weights of all fishes are initialized with the value $W_{scale}/2$.

V. Conclusion

Because of the lack of time, we could not couple its 2 algorithms, combining the boids to control the basic movements of our swarm to the FSS allowing the search of waypoints. The boids algorithm is a very good algorithm to represent and order a swarm of robots. The FSS algorithm seems useful in the search for waypoints which it interprets as food. However, we are therefore not able to draw any conclusion on the veracity of using these 2 types of algorithms together.

References

[1] Banks, A., Vincent, Æ. J., Anyakoha, Æ. C. Banks, A. A review of particle swarm optimization. Part I: Background and development. Natural Computing 467–484 (2007)
[2] E. Bonabeau, M. Dorigo, and G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York, NY, USA, 1999
[3] G. Beni, “From swarm intelligence to swarm robotics,” in Swarm Robotics Workshop: State-of-the-Art Survey, E. Şahin and W. Spears, Eds., no. 3342, pp. 1–9, Springer, Berlin, Germany, 2005. View at: Google Scholar
[4] S. Garnier, J. Gautrais, and G. Theraulaz, “The biological principles of swarm intelligence,” Swarm Intelligence, vol. 1, no. 1, pp. 3–31, 2007. View at: Google Scholar
[5] Miner - Swarm Robotics Algorithms A Survey.pdf.
[6] DJ-boids — Proceedings of the 8th international conference on Intelligent user interfaces. https://dl.acm.org/doi/10.1145/604045.604089.
[7] Lima Neto, F. Lacerda, M. Multimodal Fish School Search Algorithms Based on Local Information for School Splitting. in (2013). doi:10.1109/BRICS-CCI-CBIC.2013.35.
[8] Weight based fish school search - IEEE Conference Publication. https://ieeexplore.ieee.org/abstract/document/6973919/.

UV 5.1 Initiation à la recherche

Navigation Autonome en utilisant des méthodes de renforcement.

Introduction

En tant qu'êtres humains, quand nous voulons aller dans un endroit étrange, nous essayons d'abord de comprendre l'environnement dès le départ et de trouver la destination, puis de planifier inconsciemment la voie la plus efficace dans le cerveau. Ce principe est au cœur de la navigation des appareils mobiles. De manière abstraite, le cœur comprend deux parties, l'environnement perceptuel et le cheminement basé sur l'environnement connu. Nous avons d'abord eu, l'émergence du SLAM (simultaneous localization and mapping) qui a eu a des répercussions importance pour la navigation des robots mobiles. [3] Le SLAM est un processus dans lequel les robots sont équipés de capteurs tels que la camera, le laser ou l'odomètre pour construire une carte tout en comprenant l'environnement inconnu. Une fois la carte obtenue nous pouvons définir un chemin le plus court entre le robot et l'objectif, par exemple en utilisant l'algorithme A*. Il s'agit du planificateur global.[1] Pour des applications dans le monde réel l'environnement est changeant, un obstacle peut entraver le chemin et le planificateur global ne suffit plus. Il est d'usage de définir alors un planificateur local permettant d'adapter la trajectoire au besoin pour par exemple éviter un obstacle et rejoindre le chemin du planificateur global. De manière classique, la méthode des champs de potentiels permet d'éviter des obstacles. Les capteurs de perception (LIDAR, camera RGB etc.) repèrent un obstacle, le définissent comme obstacle, font une mise à jour de la carte puis evite l'obstacle. Toutefois ce processus peut être couteux en temps et en énergie et le robot n'a aucune connaissance de la nature de l'obstacle. Les méthodes de renforcement utilisant des réseaux de neurones proposent des pistes intéressantes pour limiter ces problèmes : identifier un obstacle (convolution) et adptation à de nombreuses situations. Dans cet article nous étudierons d'abord les planificateurs globaux pour définir une trajectoire, puis nous nous attarderons sur les planificateurs locaux pouvant éviter des obstacles entravant la trajectoire. Nous aborderons enfin la simulation et son environnement et les pistes d'amélioration.

Planificateur global

Le problème de la planification de chemin peut être décrit comme faire naviguer un robot mobile dans un espace dans lequel un nombre d'obstacles est à éviter. La tâche repose généralement sur certains critères d'optimisation, tels que le moindre coût de fonctionnement en énergie, la plus courte distance de marche, temps de marche minimal, etc. Il s'agit d'un sujet important et stimulant en robotique.

En ayant effectué un SLAM, le robot a une connaissance de la carte. Les obstacles

statique et globaux son reconnus. Il faut maintenant trouver un chemin le plus court possible le menant à la cible. Différents algorithmes peuvent être utilisé comme par exemple l'algorithme A*. Il est analogue à l'algorithme de Dijkstra mais converge plus rapidement dans la plus part des cas, car il ne parcourt pas seulement le graphe en largeur comme peut le faire Dijkstra. Nous obtenons alors une trajectoire de waypoint qui peut être envoyé au planificateur local.

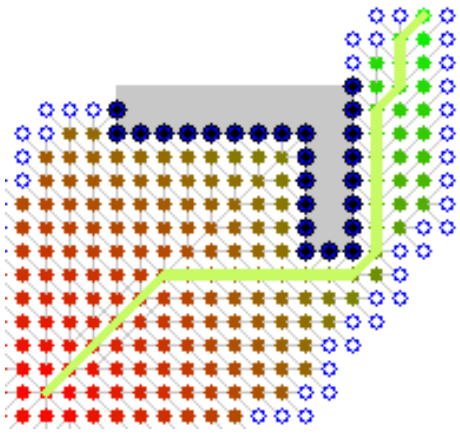


Figure 1: Trajectoire de waypoints trouvée par A*

Planificateur local

Nous souhaitons ici construire un planificateur basé sur des méthodes de d'apprentissage par renforcement.

En intelligence artificielle, plus précisément en apprentissage automatique, l'apprentissage par renforcement consiste, pour un agent autonome (robot, etc.), à apprendre les actions à effectuer à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps. L'agent est plongé au sein d'un environnement, et prend ses décisions en fonction de son état courant. En retour, l'environnement procure à l'agent une récompense, qui peut être positive ou négative. L'agent cherche au travers d'expériences itérées un comportement décisionnel optimal, appelé stratégie ou politique, et qui est une fonction associant à l'état courant l'action à exécuter. Son but étant de trouver le meilleur comportement décisionnel maximisant la somme des récompenses au cours du temps.

Nous allons plus précisément parler ici de Deep Reinforcement Learning, puisqu'il s'agit de la fusion de deux mondes qui sont l'apprentissage par renforcement qui émerge de la programmation dynamique, et du Deep Learning.

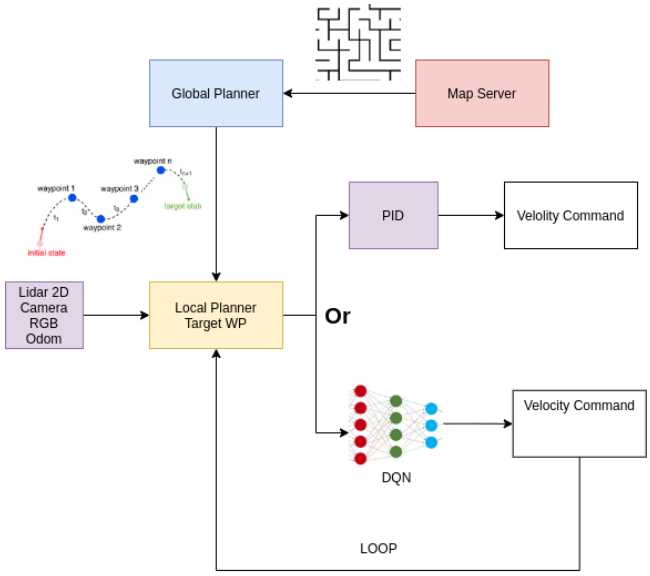


Figure 2: Couche de navigation : lien entre le planificateur global et le planificateur local. Le planificateur global définit une série de WP tandis que le planificateur local tente de se rapprocher de la trajectoire sans percuter d'obstacles

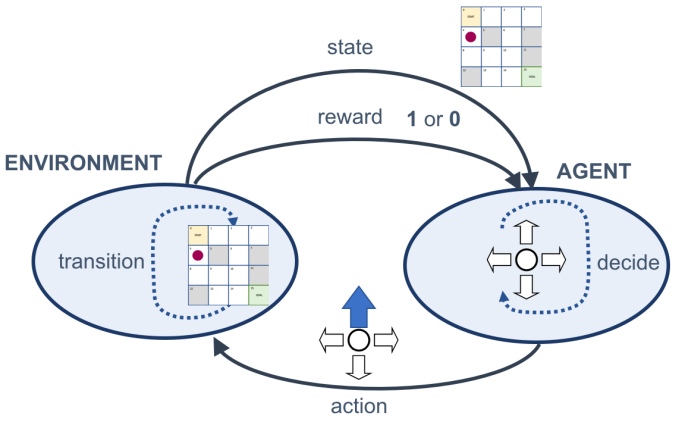


Figure 3: On voit ici, la récompense attribuée à l'agent suivant l'état dans lequel il se trouve, definir la fonction de récompense est une partie critique dont va dépendre la convergence plus ou moins rapide vers le bon contrôleur

Apprentissage par renforcement (RL)

Avant même de comprendre l'intérêt du Deep Learning dans les méthodes de renforcement, il faut comprendre le fonctionnement de la méthode d'apprentissage par Q-Learning. En effet, le Q-Learning ou reinforcement est plus ancien et donc utilisé depuis plus longtemps que le Deep Learning en programmation dynamique. Le principe du Q-Learning est d'estimer la qualité de l'action ou de l'état d'un agent dans un environnement donné. Pour estimer cette qualité, on fait appelle aux processus de décision markoviens (MDP), la valeur associée à chaque états est estimée récursivement par l'équation de Bellman. [2]

$$Q(s,a) = \sum_{s'} P_{ss'}[R_{ss'} + \gamma \sum_{a'} \pi(s',a')Q(s',a')]$$

Figure 4: $P_{ss'}$ correspond à la probabilités de transition d'un état à un autre, R est la récompense obtenue [2]

Quand l'espace d'état est discret ainsi que l'espace d'action est discret de faible dimension, on peut facilement stocker et les valeurs associées à chaques états et de manière et itérer récursivement de manière à trouver la valeur de chaque état. Pour ensuite se déplacer d'états en états ayant les plus grandes valeurs, on dit qu'on agit **greedly**.

Dans notre cas, notre capteur pour repérer les obstacles est un Lidar-2D devant couvrir au moins 120 degrès, nous avons l'odométrie à ajouter à l'espace d'état. Celle-ci variant dans un monde qui doit être suffisamment découpé pour représenter la réalité de la façon la plus exacte possible. L'espace d'état est donc gigantesque et ne peut plus être représenté par un seul graphe et être ré-estimé à chaque itération. Le Deep Q-Learning (DQN) entre alors en jeu.

Deep-Q Learning

En programmation dynamique, il faut absolument stocker les valeurs des différentes qualités des états dans une matrice ou autre, ce qui n'est pas le cas dans les méthodes de deep reinforcement learning. En effet, les réseaux de neurones sont des estimateurs universels. On les utilise aussi bien en classification d'image que pour des problèmes d'estimations de prix d'une maison. Ainsi, plutôt que de stocker la valeur d'un état ou d'une action, nous pouvons la ré-estimer à chaque instant. [6]

En reinforcement, l'objectif est d'entraîner le réseau de neurones pour qu'il trouve une politique (un comportement) optimal par rapport à l'environnement. Pour ce faire, il faut définir au préalable une fonction de reward. En effet, à chaque instant le robot va évoluer avec son environnement et recevoir une récompense. Les méthodes développées visent à optimiser la policy (politique de l'agent) de façon à maximiser ces récompenses.

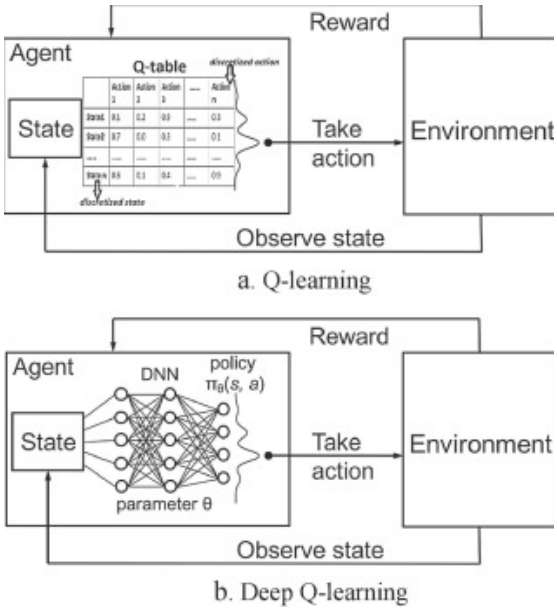


Figure 5: Difference entre le Q-Learning et le Deep-Q-Learning

Environnement de simulation et résultats

L'environnement de simulation utilise le moteur physique Gazebo, on a un robot mobile avec deux roues motrice et une roue folle. Les equations d'état sont les suivantes :

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ v = u_1 \\ \dot{\theta} = u_2 \end{cases}$$

Pour utiliser un algorithme de Deep-Q-Learning, nous sommes obligés de discrétiser l'espace d'actions ici en 9 actions qui correspondent chacune à un couple (u_1, u_2). La fonction de récompense définie est la suivante :

$$r = \begin{cases} 1 \quad if \quad dis_{cible} < 0.5 \\ -1 \quad if \quad dis_{mur} < 0.1 \\ \frac{dis_{cible}}{10} \quad else \end{cases}$$

Nous obtenons bien le fait que le robot navigue en évitant les murs, toutefois le robotne se déplace pas de manière directe sur la cible. Cela est dû au fait que l'espace d'action est trop faible et donc le pilotage n'est pas assez fins.

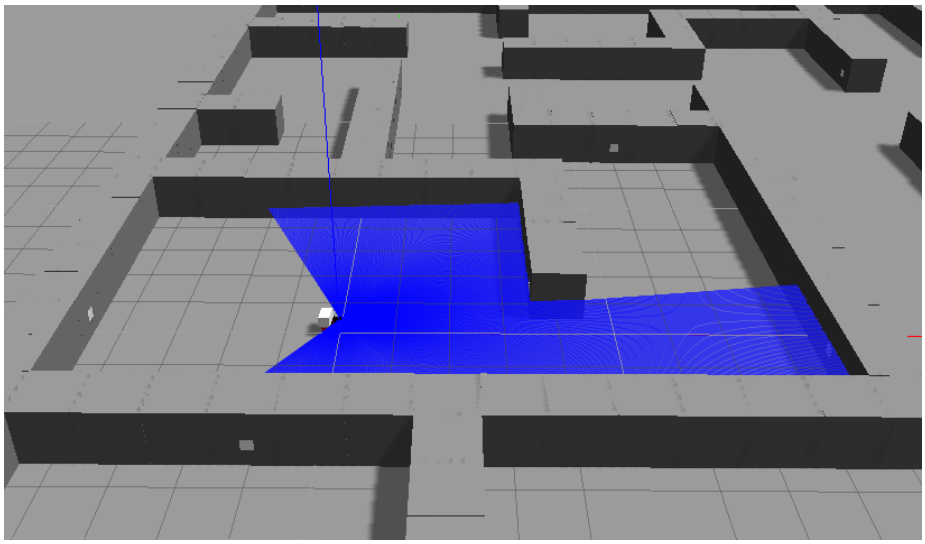


Figure 6: Robot dans Gazebo

Conclusion et Améliorations

Dans cet article nous avons proposé un contrôleur basé sur des méthode d'apprentissage par renforcement. Le contrôleur est perfectible. L'idéal serait d'utiliser un espace d'action continu afin d'avoir des trajectoires moins anguleuses, mais pour cela l'algorithme de Deep-Q Learning ne suffit plus il faut utiliser un algorithme de type acteur-critique [5], qui n'est plus seulement basé sur l'estimation des Q-Valeurs. On a deux réseaux de neurones un pour l'acteur, un pour le critique :

- Acteur : L'objectif de l'acteur est d'indiquer la politique à adopter (on parle en effet d'un acteur). Ainsi, c'est l'acteur qui choisit quelle action adopter en fonction de l'état dans lequel il se trouve. Il prend donc en entrée l'état de l'agent et il ressort en sortie l'action à adopter.
- Critique : L'objectif du critique est de critiquer l'action ou l'état du robot. En effet, cela dépend de la méthode employé.Il prend en entrée l'état de l'agent et ressort en sortie la valeur de l'état dans lequel se trouve l'agent.

Après avoir fait ses preuves dans le domaine du jeu vidéo. Le défi du RL est de le déployer sur des plateforme réelles, cela constitue un sujet de recherche important.

Bibliographie

[1] Noriyuki Kojima and Jia Deng. To Learn or Not to Learn: Analyzing the Role of Learning for Navigation in Virtual Environments. University of Michigan,

Ann Arbor, Princeton University

[2] Richard S Sutton and Andrew G Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.

[3] Hao Quan , Yansheng Li and Yi Zhang . A novel mobile robot navigation method based on deep reinforcement learning. International Journal of Advanced Robotic Systems

[4] Jing Xin,Huan Zhao,Ding Liu. Application of Deep Reinforcement Learning in Mobile Robot Path Planning. Shaanxi Key Laboratory of Complex System Control and Intelligent Information Processing Xi'an University of Technology

[5] Ivo Grondman; Lucian Busoniu; Gabriel A. D. Lopes. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. IEEE

[6] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, Anil Anthony Bharath. A Brief Survey of Deep Reinforcement Learning. IEEE SIGNAL

Calibrating a magnetometer sensor using multiple methods

Robin Sanchez

***Index Terms*—Magnetometer, calibration, compass, least squares, ellipsoid**

I. Introduction

MAGNETOMETERS are often used in attitude and heading reference systems (AHRS), mainly in order to compute the sensor heading without using an expensive dual GNSS antennas compass or an even more expensive fiber-optic gyroscope. However, magnetometers are very sensitive to magnetic interference in their environment and thus need to be calibrated after every change of environment.
According to Talat Ozyagcilar [2], there is two kind of magnetic interference: hard-iron and soft-iron. Hard-iron interference are generated by permanently magnetized ferromagnetic elements and are additive to the earth magnetic field. Soft-iron interference are distortions to the magnetic field by unmagnetized elements.
We will compare three calibration methods. Two of them are least squares based methods. One will permit to calibrate the sensor in an hard-iron only perturbed environment while the other will do the same in a both hard and soft-iron perturbed environment.The third method is a pseudo calibration, often used for inexpensive sensors, using only minimum and maximum computations on each axis of the magnetometer.

II. Soft and hard-iron calibration

A. Magnetic measurements model

According to [1], in an environment with both hard and soft iron perturbations, the measurement of the magnetic field $\mathbf{B_p}$ is:

$$\mathbf{B_p} = \mathbf{W}\mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\mathbf{R_z}(\psi)B\begin{pmatrix} cos(\delta) \\ 0 \\ sin(\delta) \end{pmatrix} + \mathbf{V} \tag{1}$$

with $\mathbf{W}$ the soft-iron matrix, $\mathbf{V}$ the hard-iron offset, δ the magnetic inclination, $\mathbf{B}$ the magnetic magnitude and $\mathbf{R_x}(\phi)$ $\mathbf{R_y}(\theta)$ $\mathbf{R_z}(\psi)$ rotation matrix around the yaw (ψ), pitch (θ) and roll (ϕ) angles of the sensor.
For any rotation matrix $\mathbf{R}(\alpha)$:

$$\mathbf{R}^{\mathrm{T}}(\alpha) = \mathbf{R}^{-1}(\alpha) \implies \mathbf{R}^{\mathrm{T}}(\alpha)\mathbf{R}(\alpha) = \mathbf{I}$$

Using (1):

$$\begin{cases} \mathbf{W}^{-1}(\mathbf{B_p} - \mathbf{V}) = \mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\mathbf{R_z}(\psi)B\begin{pmatrix} cos(\delta) \\ 0 \\ sin(\delta) \end{pmatrix} \\ (\mathbf{W}^{-1}(\mathbf{B_p} - \mathbf{V}))^{\mathrm{T}} = \left(\mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\mathbf{R_z}(\psi)B\begin{pmatrix} cos(\delta) \\ 0 \\ sin(\delta) \end{pmatrix}\right)^{\mathrm{T}} \end{cases}$$

$$\begin{aligned} &\implies (\mathbf{W}^{-1}(\mathbf{B_p} - \mathbf{V}))^{\mathrm{T}}W^{-1}(\mathbf{B_p} - \mathbf{V}) = B^2 \\ &\implies (\mathbf{B_p} - \mathbf{V})^{\mathrm{T}}\mathbf{A}(\mathbf{B_p} - \mathbf{V}) = B^2 \\ &\text{with } \mathbf{A} = (\mathbf{W}^{-1})^{\mathrm{T}}(\mathbf{W}^{-1}) \end{aligned} \tag{2}$$

According to the introduction of [3], equation (2) describe an ellipsoid of parameters $(\mathbf{A}, \mathbf{V})$.

B. Least squares fitting

The quadratic equation of an ellipsoid is [3]:

$$x^{\mathrm{T}}\mathbf{M}x + b^{\mathrm{T}}x + d = 0 \tag{3}$$

with $\mathbf{M} = \begin{pmatrix} m_1 & m_2 & m_3 \\ m_2 & m_4 & m_5 \\ m_3 & m_5 & m_6 \end{pmatrix}$ $b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$ where $\mathbf{A}$ is a positive-definite matrix.

$$\begin{aligned} (3) \implies & m_1x_1^2 + m_4x_2^2 + m_6x_3^2 + m_2x_1x_2 + m_3x_1x_3 + \\ & m_5x_2x_3 + b_1x_1 + b_2x_2 + b_3x_3 + d = 0 \end{aligned} \tag{4}$$

The equation (4) can easily be fitted using linear least squares.
With $\hat{\mathbf{M}}$, $\hat{b}$ and $\hat{d}$ the least squares estimate of $\mathbf{M}$, b and d, we can then calculate $\hat{\mathbf{A}}$ and $\hat{\mathbf{V}}$.
According to [3] (equation (5)),

$$\begin{cases} \hat{\mathbf{V}} = -\frac{1}{2}\hat{\mathbf{M}}^{-1}\hat{b} \\ \hat{\mathbf{A}} = \frac{1}{\hat{V}^{\mathrm{T}}\hat{\mathbf{M}}\hat{V} - \hat{d}} \end{cases}$$

In order to calibrate the sensor, we need to compute both $\hat{\mathbf{V}}$ and $\hat{\mathbf{W}}^{-1}$. Using equation (20) of [1], we know that $\hat{\mathbf{W}}^{-1} = A^{\frac{1}{2}}$. We could also use a Cholesky decomposition.
Finally, we can obtain calibrated magnetic measurements using the following equation:

$$\mathbf{B}_{cal} = \hat{\mathbf{W}}^{-1}(\mathbf{B}_{mes} - \hat{\mathbf{V}}) \tag{5}$$

III. Hard-iron only calibration

A. Magnetic measurements model

In this case, we assume that soft-iron interference are negligible compared to hard-iron. Thus, $\mathbf{W} = \mathbf{I}_3 \implies \mathbf{A} = \mathbf{I}_3$. Equation (2) became:

$$(\mathbf{B_p} - \mathbf{V})^{\mathrm{T}}(\mathbf{B_p} - \mathbf{V}) = B^2 \tag{6}$$

B. Least squares fitting

Then, using equation (6) and equation (26) of [3], we can obtain the vector $\hat{\mathbf{V}}$ and $\hat{B}$ using linear least squares with the following equation:

$$\begin{pmatrix} \mathbf{B_p}_x & \mathbf{B_p}_y & \mathbf{B_p}_z & 1 \end{pmatrix} \begin{pmatrix} \mathbf{V}_x \\ \mathbf{V}_y \\ \mathbf{V}_z \\ B^2 - (\mathbf{V}_x^2 + \mathbf{V}_y^2 + \mathbf{V}_z^2) \end{pmatrix} = \mathbf{B_p}_x^2 + \mathbf{B_p}_y^2 + \mathbf{B_p}_z^2 \quad (7)$$

Finally, we can obtain calibrated magnetic measurements using the following equation:

$$\mathbf{B}_{cal} = \mathbf{B}_{mes} - \hat{\mathbf{V}} \quad (8)$$

IV. Pseudo-calibration

To pseudo calibrate magnetic measurements with a dataset of n samples of the form $B_m[i] = \begin{pmatrix} B_x[i] & B_y[i] & B_z[i] \end{pmatrix}^\mathsf{T}$:

- Compute x_{min}, x_{max}, y_{min}, y_{max}, z_{min} and z_{max} on the dataset,
- Compute x_{mean}, y_{mean} and z_{mean} on the dataset,
- For each measurement $B_m[i]$, $B_{cal}[i] = \begin{pmatrix} \frac{2(B_x[i]-x_{mean})}{x_{max}-x_{min}} \\ \frac{2(B_y[i]-y_{mean})}{y_{max}-y_{min}} \\ \frac{2(B_z[i]-z_{mean})}{z_{max}-z_{min}} \end{pmatrix}$

V. Results

To test these algorithms, we will use two sensors: a MPU-9250 (SparkFun "9DoF Razor IMU M0") and the magnetometers of an SBG Ellipse 2A. These tests use raw data from the sensors, and thus don't permit to compare or judge there performance as we did not use the manufacturer recommended way of calibrating the sensor.

All the calibrations have been done the same day in the same place. For each sensor, three calibrations have been performed: the first without an apparent perturbation, the second with a 0.8 kg piece of steel fixed to the sensor and the last with a smartphone attached to the sensor.

According to equation (1), after calibration, magnetic measurements should be in the following form:

$$\mathbf{B_{cal}} = \mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\mathbf{R_z}(\psi)B \begin{pmatrix} cos(\delta) \\ 0 \\ sin(\delta) \end{pmatrix} \quad (9)$$

Thus each calibrated measurement must be on a sphere. After each calibration, we will measure the distance of each calibrated measurement to a sphere of radius $r = 1$. We will then compute the mean and standard deviation of these distances to compare the performance of the different calibration methods. For each calibrations, the sensor has been rotated in all the directions at least two times.

No perturbation	MPU-9250		SBG Ellipse 2A	
	Mean	Std	Mean	Std
Soft and hard-iron calibration	0.0028	0.0236	0.0003	0.0108
Hard-iron only calibration	0.0004	0.0296	0.0002	0.0183
Pseudo-calibration	0.0266	0.029	0.0049	0.042

0.8kg piece of steel	MPU-9250		SBG Ellipse 2A	
	Mean	Std	Mean	Std
Soft and hard-iron calibration	0.0088	0.0462	0.0012	0.0219
Hard-iron only calibration	0.0079	0.1254	0.0251	0.2228
Pseudo-calibration	0.0469	0.155	0.0381	0.1226

Smartphone perturbation	MPU-9250		SBG Ellipse 2A	
	Mean	Std	Mean	Std
Soft and hard-iron calibration	0.0189	0.0717	0.0029	0.0288
Hard-iron only calibration	0.0016	0.0562	0.0006	0.0344
Pseudo-calibration	0.0431	0.0575	0.0124	0.0416

MPU-9250	Ellipsoid semi-axis			Hard-iron offset		
No perturbation	262	283	272	−91.5	104	−274
Piece of steel	193	714	1362	−130	−177	−175
Smartphone	275	299	312	−85.0	172	−432

SBG Ellipse 2A	Ellipsoid semi-axis			Hard-iron offset		
No perturbation	1.09	1.02	1.05	0.008	−0.012	0.048
Piece of steel	1.24	0.55	0.50	0.11	0.0037	0.055
Smartphone	0.93	0.85	0.86	0.30	0.24	0.47

Ellipsoid semi-axis and hard-iron offset are obtained with the soft and hard-iron calibration.

The results in the "no perturbation" case shows that the both soft and the hard-iron calibration and hard-iron only one are almost equivalent. However, the pseudo-calibration gives worst results than the two others methods but the standard deviation stay in the same order of magnitude than the others methods. The "smartphone perturbation" gives the sames results. The hard-iron offset is strong in these cases compared to the differences between the ellipsoid semi-axis which represent the soft-iron interference.

In the "0.8kg piece of steel" case, both the pseudo-calibration and the hard-iron only one are really off compared to the soft and hard-iron one. This could be explained by the fact that the piece of steel seems to create an important soft-iron interference which could be observed in ellipsoid semi-axis.

VI. Conclusion

The soft and hard iron seems to be an efficient calibration method. The hard iron only method gives good performances in an magnetically undistorted by soft-iron interference environment. Compared to the soft and hard iron The pseudo-calibation is less efficient than the others methods and thus shall be only used in case we are not able to compute least squares algorithms.

Acknowledgment

The authors would like to thank Fabrice Le Bars for providing the sensors and Luc Jaulin for his help.

References

[1] Talat Ozyagcilar, *Application Note: Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference*, Freescale Semiconductor Inc, 2013.

[2] Talat Ozyagcilar, *Application Note: Layout Recommendations for PCBs Using a Magnetometer Sensor*, Freescale Semiconductor Inc, 2013.

[3] Markovsky, I. and Kukush, A. and Huffel, S. Van, *Consistent least squares fitting of ellipsoids*, Numerische Mathematik, pages 177-194, 2004.

Monocular odometry using ORB-SLAM 2

Bertrand Turck
ENSTA Bretagne
Brest, France
Email: bertrand.turck@ensta-bretagne.org

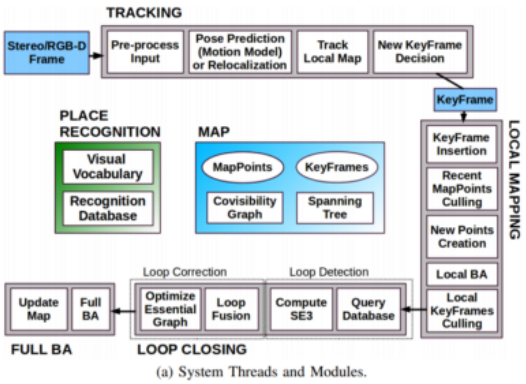
***Abstract*—This paper aim to use ORB-SLAM to estimate the trajectory of a camera using only a monocular camera. ORB-SLAM is able to realize simultaneous localization and mappingin real time. However, this algorithm face a problem with scale when using a single camera. We present a solution to fix this problem.**

I. Introduction

Simultaneous localization and mapping (SLAM) is a famous problem encountered in robotics. The best ways to solve this problem are to use precise sensors (LiDAR, IMU, ...) or to place landmarks on the ground. But for cheaper robots, these solutions are not viable. A solution could be to use a simple camera, as this sensor can be found at low price.
The ORB-SLAM 2 algorithm [1] aim to construct a 3D environment using only a calibrated monocular camera and using bundle adjustments. This algorithm resolve common issues like losing the tracking when the system moves too quickly. But another problem this algorithm is facing is the problem of depth error: with only a monocular camera, you can't directly estimate a distance, thus the constructed environment doesn't have the correct scale. We will try to resolve this problem.

II. ORB-SLAM 2

The ORB-SLAM 2 algorithm is divided into 3 parts which works one after another. The first one is the tracking part which aim to create a local map and estimate the small movement of the camera. The second part is local mapping and tries to optimize the successive maps and estimate a global movement of the camera. Last but not least, the loop closing part aims to notice if the camera as already seen a place and optimize the global map if possible.



(a) System Threads and Modules.

Fig. 1. ORB-SLAM functioning overview.

A. Bundle adjustment

Simultaneous Localization and Mapping (SLAM) can be achieved by solving bundle adjustment, which consist in creating a map of a 3D environment while estimating the consecutive states of our system [2].
Bundle adjustment can be translated as a mathematical problem: minimizing the reprojection error, which is the distance between the projection of a cloud point in the camera frame and points acquired by the camera.

B. Tracking

To begin, the algorithm looks for features which are visible between to successive images by using the ORB features detector algorithm [3] and match them. We can note that only few points are used and every other pieces of information are discarded, which makes this method less effective compare to one which may use artificial intelligence to get features.
A first motion only bundle adjustment is computed knowing only a small movement occurred between two images. Points are added to the local map in a way which minimize the reprojection error. Bundle adjustment estimate rotation and translation between the 2 images as odometry, like the iterative closest point (ICP) algorithm would do for LiDAR data.

C. Local Mapping

As the tracking part add more and more points to the local map, local mapping tries to eliminate the points which are redundant by merging close points together. Also, outlier points are discarded if they are not seen by at least 25 % of the images.
Also, a local bundle adjustment allows to have an estimation of rotation and translation in the global environment. The origin is defined as the first keyframe after the system was initialized.

D. Loop closing

Loop closure is the part which really define a SLAM algorithm. Loop closure is performed by checking if a loop is detected and if one is detected a pose-graph optimization is performed. These operations are quite costly and thus are performed in a parallel thread.
The paper explains that when using a monocular camera, the drift is too large for a loop closing to be performed correctly. In fact, with monocular SLAM, the map can drift with 7 degrees of freedom: 3 rotations, 3 translations and a scale factor.
When a loop closure is achieved, a full bundle adjustment is computed as the map was updated.

III. Solution to the scale problem

A. ROS implementation

With ORB SLAM 2 is given a simple example to use the algorithm using images from ROS topic. To work with, we added a topic which publishes the successive estimated positions of the camera as a path topic. This enable us to visualize the trajectory in real time.
But the algorithm doesn't give us the position directly. The rotation is not given in the base ROS uses, so we have to apply a rotation of minus $\frac{\pi}{2}$ around x to get x forward and y left. This is done using quaternion and Hamilton products.

B. Solution implemented

As ORB-SLAM is unable to get a scale value with monocular camera, the scale of the cloud point and also the scale of the movement depend of the initialization. To get a proper initialization, the system tries to match two local maps. The movement between these two maps define the scale, so scale changes every time, you can't just find a coefficient which works every time.
A solution [4] is to use external sensors to estimate a scale coefficient. By moving the system of a known distance and by checking the value given by ORB-SLAM you can find a coefficient.
As we don't want to use external sensors, we move the system by hand when ORB-SLAM is done with initialization. After moving the system of a known distance, we notify the algorithm of the distance travelled by typing the value in a dedicated ROS topic. Then, poses are modified to take the scale into account. The math is quite simple, we just have to multiply the pose given by ORB-SLAM with the scale and we get the corrected pose.

IV. Results

Experimentations were conducted on the ground floor of the building N of ENSTA Bretagne as a map has already been realized with a good LiDAR and a IMU, so we had a precise reference. To get reproductible data, we used ROS bags to get data and to play them more easily. We can notice by playing the same data over again that the trajectory is never the same and may fluctuate of about 50 cm at the end.

We can observe that scaling is working, but it is not perfect. The position at the end of the experimentation is 1 meter away from where the experimentation really ended. Also, as the initialization of the camera occur spontaneously, it is difficult to have a reference for the measurement of the distance.

V. Conclusion

ORB-SLAM 2 is a powerful tool which can be use in robotics. This algorithm is a cheap way to have access to an estimate of a state. But as the result as shown, it is not to be used on is own. By merging the data with another sensor, we could get much better results.
Also, this algorithm itself can be tricky to install and launch, and his quite expensive in computationnal power. Then, it is not suitable for every robot.

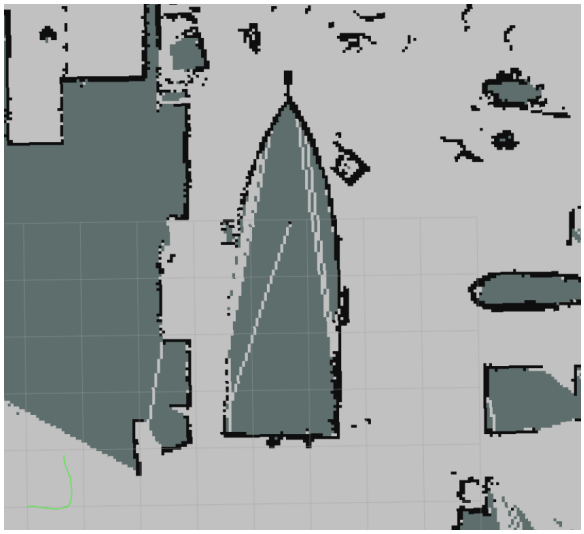


Fig. 2. Trajectory of the camera before applying scale (in green).

Fig. 3. Trajectory of the camera after applying scale (in green).

References

[1] Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163, 2015.

[2] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment a modern synthesis," in Vision algorithms: theory and practice, 2000, pp. 298–372.

[3] Rublee, Ethan; Rabaud, Vincent; Konolige, Kurt; Bradski, Gary, "ORB: an efficient alternative to SIFT or SURF", IEEE International Conference on Computer Vision (ICCV), 2011.

[4] D. Gehrig, M. Göttgens, B. Paden, E. Frazzoli, "Scale-Corrected Monocular-SLAM for the AR.Drone 2.0", 2017.

Searching for Optimal Path for an AUV in the Current: Application to Cycle Search

Quentin Vintras
ENSTA Bretagne
Email: quentin.vintras@ensta-bretagne.org

***Abstract*—Abstract— This paper proposes a method based on Dijkstra algorithm to find an optimal path for an AUV[1] in a ocean environment characterized by strong currents and enhanced space–time variability. The goal is to find a safe path that takes the vehicle from its starting location to a mission-specified destination, minimizing the energy cost. Dijkstra algorithm is based on a graph creation with a current map. The performance of Dijkstra algorithm can be discussed as it presumes that the current field is known before the mission. The final goal of the paper is to use the path planning to search cycle minimizing the energy cost.**

***Index Terms*—optimal, mission planning, paths, cycles, current, dijkstra**

I. Introduction

PATH PLANNING for autonomous underwater vehicles is one of the most important thing. Since the vehicle evolve in harsh conditions without the possibility to communicated with an operator in surface. Path planning works very well when the user wants to avoid dangerous areas or known obstacles. But some applications require to minimize the size of AUVs or their operating range. This require to economize power since they can't have unlimited resources. Path planning must therefore take into account other parameters such as the current that can be a very powerful obstacle. Some techniques of path planning were developed [1]. These methods can be divided into two broad categories, pre-mission planning and real-time in-mission planning. In the first category, the operator will define a plan before the mission and the robot will stick to this plan whatever the hazards encountered. The second category of algorithm assigns to the robot the choice of the plan that is created during the mission.In this article we are going to make an inventory of existing path planning algorithms, in particular for underwater vehicles, then we will set up one of them, to minimise the energy spent, thanks to the force of the current. Finally, we will use these developments to apply this algorithm to the search for optimal cycles from an energy point of view and we will discuss its efficiency.

II. Previous Work

Numerous researches and algorithms have been developed to solve this path planning problem.

A. Potential fields

The techniques on the fields of potential are very effective and very quick to implement. They are based on the creation of an artificially created potential field from goals and obstacles. Obstacles will thus be repulsive while goals will be attractive. The disadvantage of this technique is the susceptibility to local minima that might not make us find the optimal path or prevent us from reaching the goal.

B. Graph Techniques

Graph searching techniques are also widely used [2]. They are based on the discretization of space to form a graph in which it is fairly simple to find a path that avoids obstacles. This way, all unfeasible paths and optimal paths can be found. These techniques are very reliable for local minima but pose big problems when the dimensions increase. These techniques include the A* method [3], the Jump Point Search algorithm, the Dijkstra algorithm...

C. Case Based Path Planning

Case reasoning[4] is based on experience. Indeed, this technique uses the experience of past missions to plan a new mission. When a new road has to be created, we use the old ones for which we know the environment. In this way we can reconstruct a route that we are sure will be reliable. Finally, this new road will be added to the database to increase the reliability of future roads. The advantage of this technique is the speed of calculation, however it requires a large database and a fairly stable environment.

D. Genetic Algorithms

Some algorithms based on Darwinian theories of evolution have been developed and applied to path planning [5]. These algorithms called GAs[2] create a list of possible paths which are then modified by genetic operators such as mutations.Their advantage is the much lower calculation time compared to other techniques, however we are not sure that we can achieve an optimal solution in finished time. Some GAs have also been adapted to calculate paths in real time [6].

[1]Autonomous Underwater Vehicle

[2]Genetic Algorithms

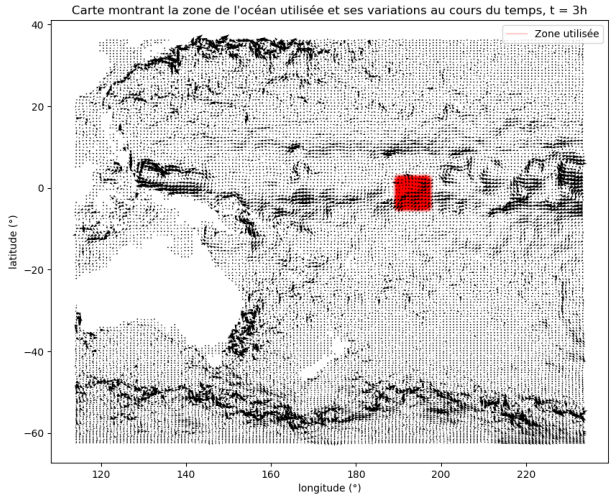


Fig. 1: Current map used to test algorithms

DIJKSTRA(G: graph, s: vertex)

```
1  for each vertex v ∈ V_G
2      dist[v] ← ∞
3      parent[v] ← NIL
4  dist[s] ← 0
5
6  Q ← V_G
7  while Q ≠ ∅
8      u ← EXTRACT-MIN(Q)
9      for each edge e = (u, v)
10         if dist[v] > dist[u] + weight[e]
11             dist[v] ← dist[u] + weight[e]
12             parent[v] ← u
13
14 H ← (V_G, ∅)
15 for each vertex v ∈ V_G, v ≠ s
16     E_H ← E_H ∪ {(parent[v], v)}
17
18 return H, dist
```

Fig. 2: Dijkstra algorithm pseudo-code

III. Current Fields

Currents are continuous water movements. They are the result of several factors such as wind and tides at the surface and salinity and temperature deeper down. Currents are fairly predictable except in coastal areas where the effect of tides and topography can have unpredictable effects. In oceanic areas, currents are fairly redundant and predictable thanks to meteorological technologies. The research around the currents is pretty well done and we have a lot of information. A lot of data is available on ocean currents. For some regions of the world, we have very precise data on the ocean environment thanks to a combination of techniques such as HF radar surface current measurements, satellite observations... The reliable and daily updated open source predictive tools that can be used are : Global Real-Time Ocean Forecast System (RTOFSGlobal), Regional Ocean Model System and NavyCoastal Ocean Model. In this paper, I consider RTOFSGlobal model. It provides current velocities all around the world with a time resolution of 3 hours and a spatial 2 dimension resolution of 7 kilometers. Figure 1 display the part of the world I used to test path planning algorithms.

IV. Dijkstra Algorithm for path Planning

The Dijkstra algorithm with the pseudo-code presented in Figure 2 is an algorithm to find the shortest path between two nodes of a weighted graph. The challenge in this paper is to create the graph from the current field map. The graph is created by discretizing the current field to form a mesh. Each intersection of this mesh will be a node. Each node corresponds to a position where the velocity of the current is known in two dimensions. Each node will be connected to the 8 adjacent nodes. The weights are calculated by a simple scalar product between the velocity of the current at the considered node and the vector that links the two nodes. Since the Dijkstra algorithm does not take into account negative weights, if the current is contrary to the direction of the edge, the weight is set to infinity, this also guarantees that the AUV will never go against the current. Moreover, in the Dijkstra algorithm we try to find the path with the most, which is the same as finding the path with the lowest weights, so I took the inverse of the weights so that when the current is strong in one direction, the weights in that direction are very weak and vice versa. This is equivalent to taking into account the travel time rather than the speed. Finally just need to choose 2 nodes and to launch the Dijkstra algorithm on the previously created graph.

A. Simple Path Planning

First I used the Dijkstra algorithm to plan a mission from point A to point B without taking into account the temporal variation of the current. To do that I just compute the predecessors with the Dijkstra algorithm and after that I start from the point of arrival and goes back up the dictionary of predecessors until arriving at the point of departure. If there is no path between the point of arrival and the point of departure, it means that there is no path in the area described without a counter-current passage. An example of the shortest path in the current with the Dijkstra algorithm is given in figure 3, where the current field is shown and the points represent the nodes through which one has to pass.

B. Path planning With Time Simulation

An improvement in this planning would be to take into account the variation of the current field over time. To add a time component, the predecessors' dictionary must be recalculated with each change in the current field. However, one of the problems of the Dijkstra algorithm is that you have to go back to the starting point of the predecessors' dictionary. This forces us to know the variation of the current field in advance. This can be a problem in areas where currents are not studied or are very unpredictable. However, in most cases the currents are predictable and known. In this article I used a database

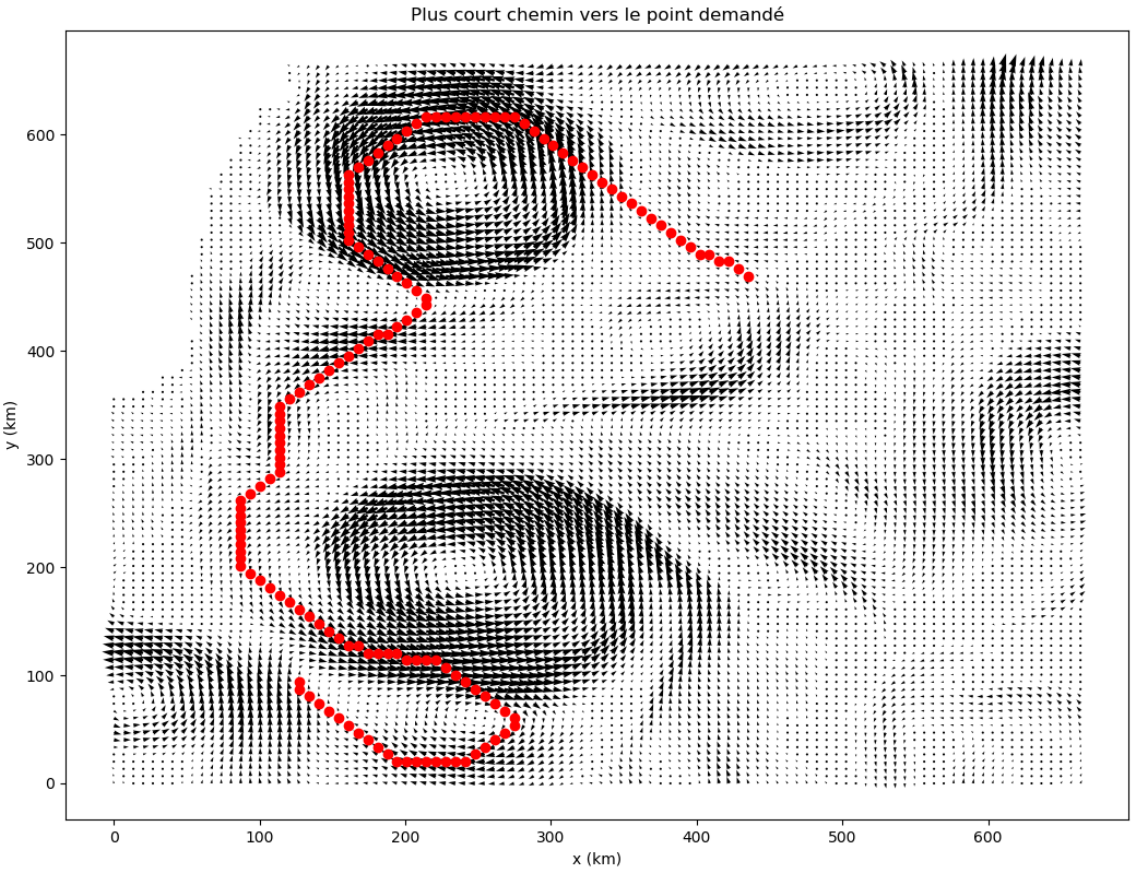


Fig. 3: An example of shortest path in current with Dijkstra algorithm

that recorded the current fields over a given area with a time step of 3 hours. In addition I know the current field at all time steps in advance. To take into account the time step I have therefore first calculated the predecessors' dictionary for the current field at the arrival time which will be noted h.We will note the time step dt.We start by calculating the dictionary of predecessors at time h.The dictionary is then put back together, paying attention to the weights. Between each node we will look at the speed at which the robot will go, we can deduce a travel time between two nodes because we know the distance between each node. We're going to use this travel time. Let's note the travel time between the predecessor nodes i and $i-1$, t_i. As we go back in the dictionary of predecessors we add the t_i. If the sum S of t_i is greater than dt, it means that the current field has changed. The dictionary of predecessors must therefore be recalculated from a new graph and S is reset to 0. This operation is repeated until the starting point is reached. We can thus deduce the date on which we have to leave, if we go through n nodes, the departure date will be $d = h - \sum_{ni=1}^{n} t_i$. Finally in this case Dijkstra algorithm give the shortest path and the departure date to reach the end point at the desired date. In a practical field this time component would be very complicated to put it in place, or it must be done in very short period of time and on very short distance. The uncertainty on current prevision makes the problem very tricky. Moreover it requires a lot of computing resources and this can be time consuming as the dictionary of predecessors has to be recalculated at every time step.

V. Application of Dijkstra Algorithm to cycles

More and more scientific applications require data collection. The oceans are a vast field of study, where data collection is becoming increasingly important with environmental changes. In this hostile environment, where human deployment is complicated and expensive, the use of autonomous vehicles capable of travelling long distances using the least amount of energy is fully justified. From this point of view, the search for the execution of cycles by an AUV propelled solely by currents is a field with a great deal at stake. In this paper I propose the application of the Dijkstra algorithm for path planning in the search for the most economical cycles for an AUV in an ocean.

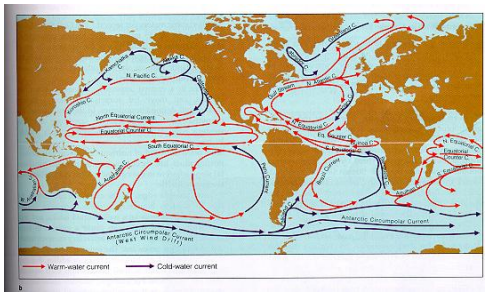


Fig. 4: Surface circulation on the main oceans of the world

A. Methodology

As we have seen previously, with the Dijkstra algorithm we are able to plan the shortest path between two points in the ocean taking into account the current field. However, this requires some assumptions such as knowing the upstream current field. For this application I will not take into account the time component. Thus we assume that the AUV can move in a current field invariant in time and depth. To constitute a cycle we then need at least 2 points through which the AUV must pass. Let's note n the number of points by which we would like to pass and p_i $i \in [\![1, n]\!]$ the i^{th} point. To compute the shortest cycle that pass to all we compute the graph of the zone we choose the points. The dictionary of predecessors from p_1 to p_n is then calculated and for i from n to 2 the dictionary of predecessors from p_i to p_{i-1} is calculated, noting each time the shortest path between the two points. Thus we have a cycle passing through all the points.

B. Results

I tested this methodology with cycles of two points. The python code given in the repository https://github.com/QuentinVintras/AUV_Dijkstra_path.git compute the most economic cycle for two points. It returns a figure with the cycle if there is one and an error message if we can't find a path that satisfies the conditions listed above in the section III.A. The larger the area chosen, the better the chances of finding a suitable path. The currents being circular at the level of an ocean as shown in figure 4 []. If we consider the whole globe or a whole ocean, the probability of finding a cycle is maximized. The only limitation is the computing resources because the more nodes there are in the graph, the longer the calculation takes.

I have computed a cycle between two points in an area defined by a graph of 100 by 100 nodes with a spatial resolution of 7km. It gives a area of 700x700 km in the Pacific Ocean. That is not sufficient to be sure to find a path. After some trials I managed to find some cycles. One of them is displayed in the figure 5.

C. Possible improvements

Finally this technique is powerful to plan cycles in large areas with a very low resolution. In this paper we don't take into account the time variability of ocean currents and we assume that they are predictable. This first point can be taken into account with the technique presented in section IV.B. We can also test our algorithm with more that two passage points but this increases the risk that no path is possible, since the current is seen here as an obstacle, the AUV cannot go against the current.

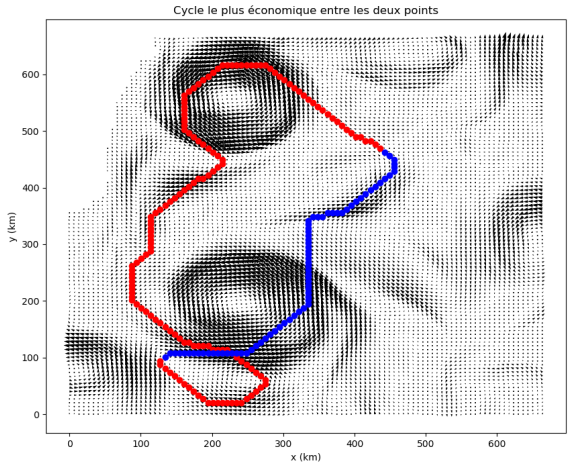


Fig. 5: An example of most economic cycle in current with Dijkstra algorithm, blue points are displaying the path between point 1 to 2 and red between 2 to 1

VI. Conclusion

Dijkstra algorithm which is a graph method works correctly if the resolution and therefore the number of nodes is not too large. Carrying out large cycle planning when the current fields are predictable works well in theory. In practice we could couple this obtained path with a faster algorithm for real time use on a finer resolution. The arrival of underwater gliders and the development of autonomy requirements for AUVs will allow the use and development of new path planning algorithms using current force as the only propulsion.

References

[1] Konuralp Yiğit. *Path planning methods for Autonomous Underwater Vehicles*. PhD thesis, Massachusetts Institute of Technology, 2011. Accepted: 2011-12-19T19:00:13Z journalAbbreviation: Path planning methods for AUVs.

[2] Jan Faigl. Grid and graph based path planning methods. page 87.

[3] K. P. Carroll, S. R. McClaran, E. L. Nelson, D. M. Barnett, D. K. Friesen, and G. N. William. Auv path planning: an a* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones. In *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology*, page 79–84, Jun 1992.

[4] C. Vasudevan and K. Ganesan. Case-based path planning for autonomous underwater vehicles. *Autonomous Robots*, 3(2):79–89, Jun 1996.

[5] C. Hocaoglu and A. C. Sanderson. Planning multi-paths using speciation in genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation*, page 378–383, May 1996.

[6] A. Alvarez, A. Caiti, and R. Onken. Evolutionary path planning for autonomous underwater vehicles in a variable ocean. *IEEE Journal of Oceanic Engineering*, 29(2):418–429, Apr 2004.