PROJETO FINAL

Implementation and simulation of EKF-SLAM algorithm

Franci Rrapi (UFMG - 2021/1)

INTRODUCTION

This project deals with the CoppeliaSim[1] robotic tools to put in practice most of tools and concepts seen in the "Robotica Movel" course[2] of professor Douglas Macharet, such as path planning, navigation, control and especially the algorithm of the EKF-SLAM[3] with the help of the slides and code of "Probabilistic Robotics" course[4] taken by prof. Giorgio Grisetti of University of Rome "La Sapienza". To do this I created a scene made by me, I used some useful Python functions, I converted the matlab code (of prof. Grisetti) to python and adapt it to this project and CoppeliaSim simulation and I applied some functions writen by me such as the one for the implementation of the "getting observations" from the scene. To run the software is very simple! Just execute each cell one at a time and when above the cell is writed "CoppeliaSim needed first!" it means that before running the cell the CoppeliaSim simulation must be started. All the resources, such as scenes, results and other code, are located in the same directory where this notebook is located. In the "results" folder are also available the images or diagrams produced during the simulation and execution of the algorithm. The presentation videos instead are vailable in the Google Drive link. The notebook of this homework is called "ekf-slam.ipynb" and the scene used, available in the "scenes" folder, is called "slam.ttt". The implementation of this algorithm was very difficult, and also his conversion in python and the adaptation to CoppeliaSim was also very hard and took a lot of time. For this project I executed the simulation several times, and each time the landmarks are well estimated but, unfortunately, the robot position is not. In the original matlab code it works very well, but in this case not. I tried a lot of tests to find the problem and solve it, in such a way to well estimate also the robot position but, did not succeed. For this project I executed the simulation several times, but the main test that I will explain and show below regards the different max range value in the observation step and see how the SLAM works in different laser conditions.

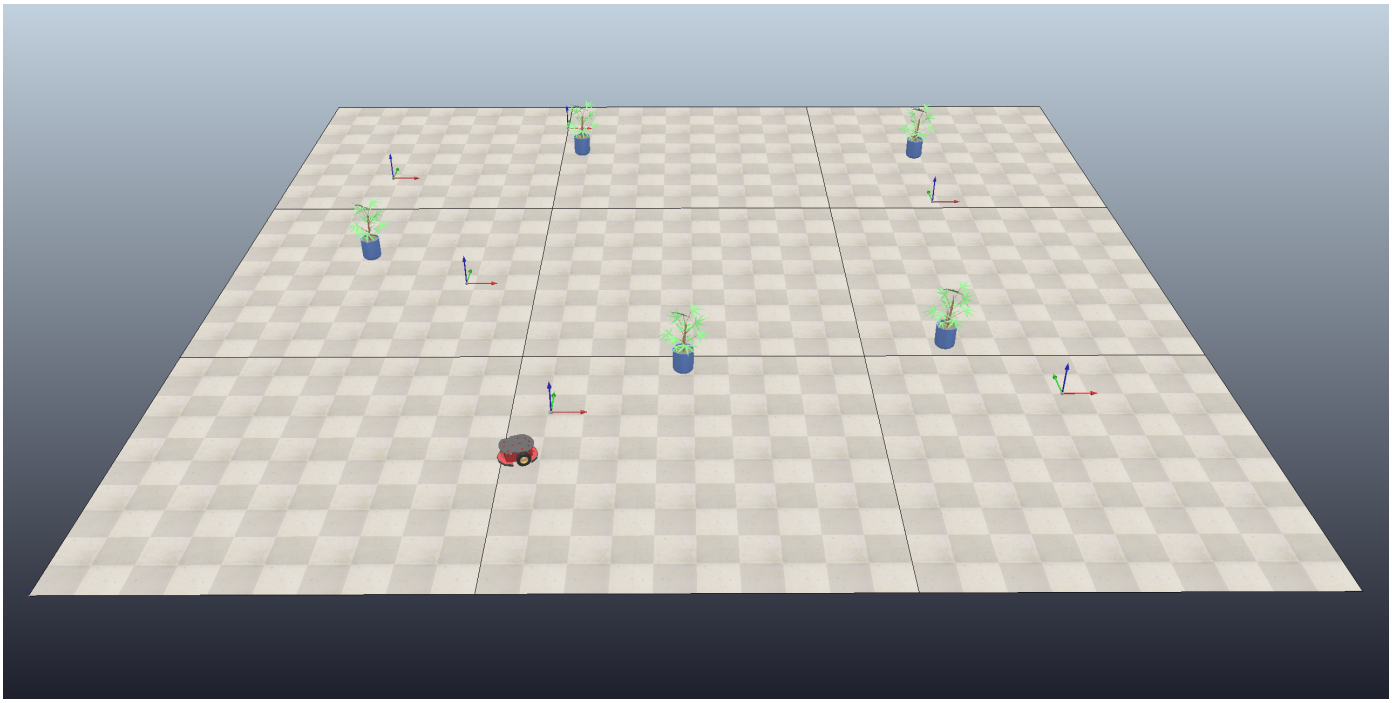
Google Drive link: [https://drive.google.com/drive/folders/1cX3aLIhogskHJSL9UgOVgIy6dGFq4Qj-?usp=sharing (https://drive.google.com/drive/folders/1cX3aLIhogskHJSL9UgOVgIy6dGFq4Qj-?usp=sharing)](https://drive.google.com/drive/folders/1cX3aLIhogskHJSL9UgOVgIy6dGFq4Qj-?usp=sharing)

IMPLEMENTATION

ENVIRONMENT

In this project, I've imported some of the functions used in the past homeworks of this course. The scene created and used is made by some objects such as the "Pioneer" robot, some reference frames that represents the "Global" and "Goal" frame and also some plants that represents the "Landmarks" of the environment. The scene size is set to (15, 15) and the global reference frame is set in the lower left part. For the navigation of the robot during the simulation I used some Goal reference frames that the robot has to reach in way to navigate all the environment. For reaching this goals, I used the motion and cinematic control of "De Luca and Oriolo" seen also in the "aula-09"[5] of this course. To navigate the environment, the robot has to reach in order Goal1, Goal2, Goal3, Goal4, Goal5 and Goal6.

Here below is possble to see the scene environment created.



ALGORITHM

What we know in this problem is that the robot is controlled by translational and rotational velocities, he senses the landmarks through a laser and that the position of the landmarks in the worls in unknown (so we have no prior knowlwdge of the map).

So, we need to implement a KF based algorithm that, simultaneously, track the position of the robot (localization) and also the position of the seen landmarks (mapping). The inputs of the algorithm will be:

- velocity measurements
- landmark measurements

The **algorithm** used is made by 3 main steps, that are:

1. PREDICT the new state using the control functions (incorporate new control)
2. CORRECT the belief in landmark positions based on the estimated state and measurements (incorporate new measurement)
3. Add new landmarks to the state (SLAM)

EKF SLAM models the SLAM problem in a single EKF where the full **state space** is represented by the state of the robot and the states of the landmarks. The robot state is x_r = (x, y, theta) in $\mathbb{R}^3$ and the landmark state is x_l = (x, y) in $\mathbb{R}^2$. So the total state space will be x = (x_r, x_1,.....,x_N) and his domain will be $\mathbb{R}^{(3+2N)}$ where N is the number of landmarks.

The **controls** domain will be $u_t = (u_t^1, u_t^2)$ in $\mathbb{R}^2$ and the **measurements** domain will be $z_t^m = (x_t, y_t)$ in $\mathbb{R}^2$ for $m = 1, \ldots, M$ where M is the number of measurements.

The **transition function**, given the control inputs, updates the position of the robot while the positions of the landmarks remain unchanged and is given by:

$$x_t = f(x_{t-1}, u_{t-1}) = (x_{t-1} + u_{t-1}^1 * cos(\theta_{t-1}), y_{t-1} + u_{t-1}^1 * sin(\theta_{t-1}), \theta_{t-1} + u_{t-1}^2, x_{t-1}^1, \ldots, x_{t-1}^N)$$

Instead, the **measurement function** is given by: $z_t^n = h^n(x_t) = h(x_t^r, x_t^n) = R_t^T(x_t^n - t_t)$ for $n = 1, \ldots, N$ where N is the number of landmarks.

Also we assume that the control inputs are effected by a zero-mean **Gaussian noise**, that is: $n_{u,t} \sim \mathcal{N}(n_{u,t}; 0, \Sigma_u)$ with

$$\Sigma_u = \begin{bmatrix} \sigma_u^2 + \sigma_T^2 & 0 \\ 0 & \sigma_u^2 + \sigma_R^2 \end{bmatrix}$$

where $\sigma_T = u_t^1, \sigma_R = u_t^2$.

Also, for each landmark that we observe, we consider a Gaussian noise, that is: $n_z \sim \mathcal{N}(n_z; 0, \Sigma_z)$ with

$$\Sigma_z = \begin{bmatrix} \sigma_z^2 & 0 \\ 0 & \sigma_z^2 \end{bmatrix}$$

The **Jacobians** will be:

- **State**:

$$\mathbf{A}_t = \left.\frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mu_{t-1|t-1}} = \begin{pmatrix} \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}^{[r]}} & \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}^{[1]}} & \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}^{[2]}} & \cdots & \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}^{[N]}} \end{pmatrix}$$

$$\frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}^{[r]}} = \begin{pmatrix} 1 & 0 & -u_{t-1}^{[1]} \cdot \sin(\theta_{t-1}) \\ 0 & 1 & +u_{t-1}^{[1]} \cdot \cos(\theta_{t-1}) \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{pmatrix} \qquad \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}^{[2]}} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix}$$

- **Controls**:

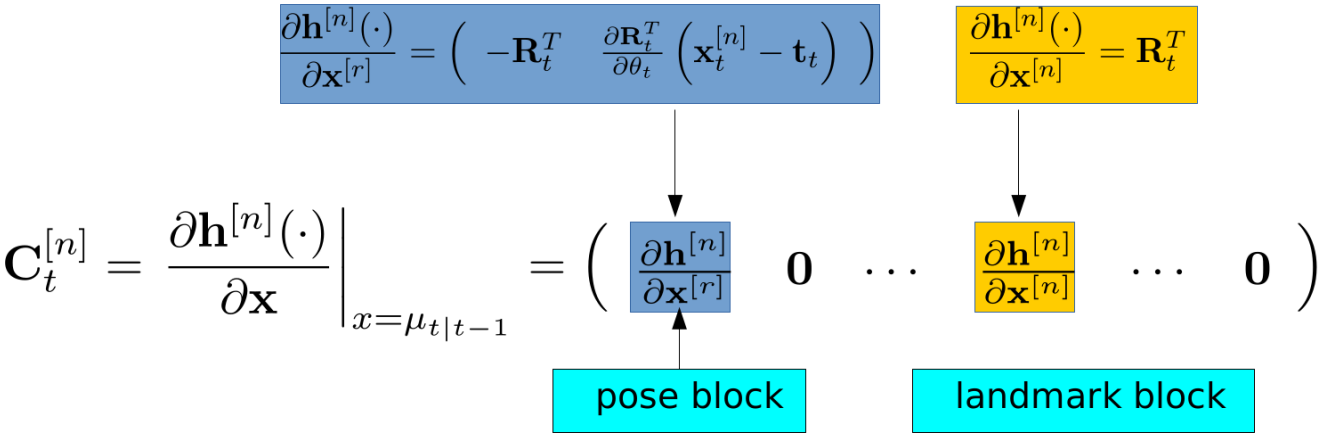
$$\mathbf{B}_t = \left.\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_{t-1}} = \begin{pmatrix} \cos(\theta_{t-1}) & 0 \\ \sin(\theta_{t-1}) & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix}$$

pose block

landmark block 1

landmark block N

- The n^{th} **Measurement** Jacobian will have a block structure:



As the robot can navigates the environment in multiple ways, the order in which the robot observes the landmarks can not be the same as the order of the landmarks in the state. So, for this reason, we need **data association**! This is very useful to map each landmark from state to ID and each ID to the state and helps us to know exactly which landmark we observed. After this, what we do is to populate the state with the positions estimated of the observed landmarks.

```
1 ## TESTS
```

The tests that I mainly executed for this algorithm regards different values of the max range parameter, that is important in the observation step. I executed the tests in order with max range value set to 5, 3 and 7.

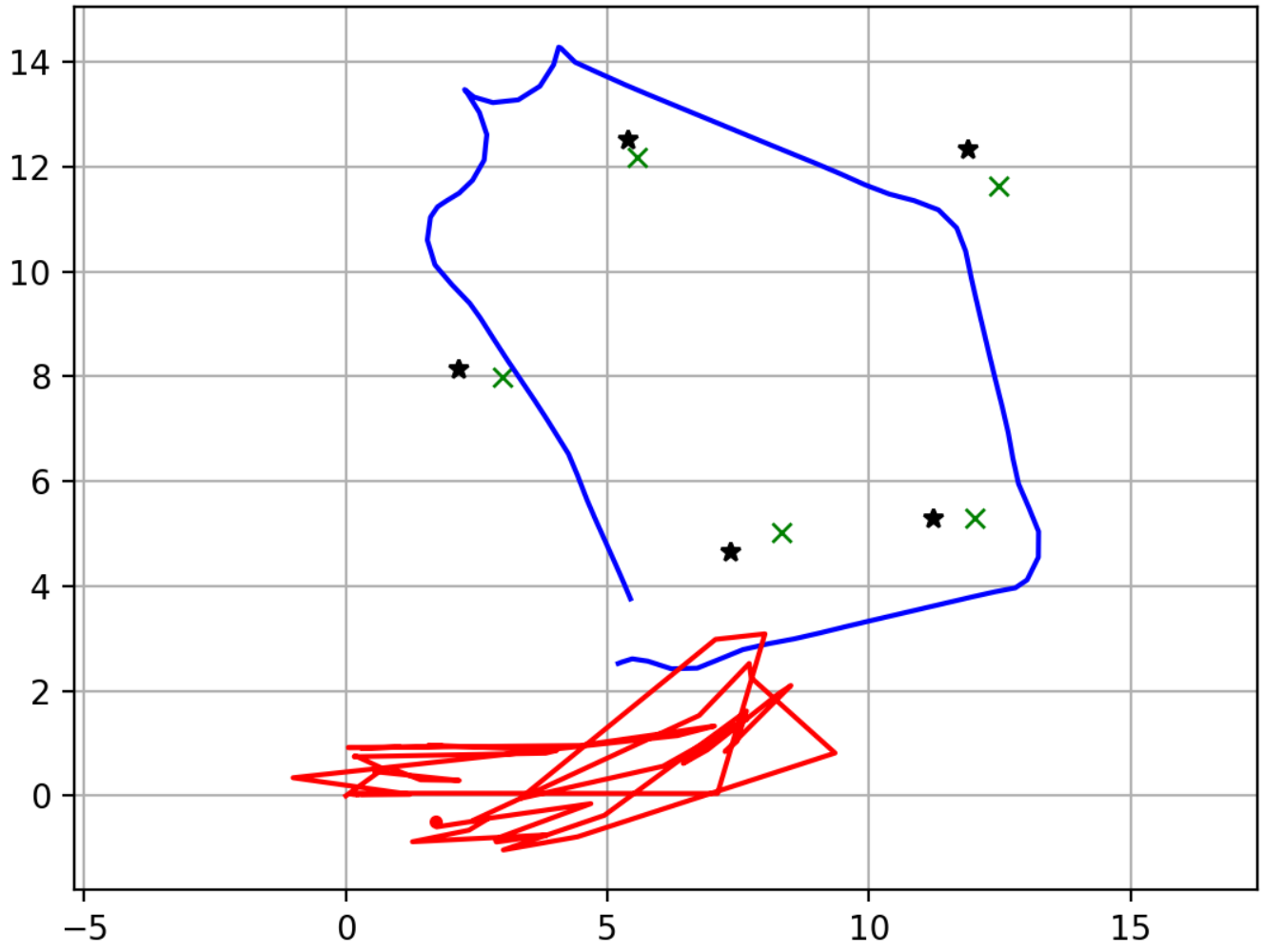
In this tests of EKF SLAM, the meaning of the items shown in the results is:

- Black stars: landmarks
- Green crosses: estimates of landmark positions
- Blue line: ground truth position
- Red line: EKF SLAM position estimation

TEST 1

In the first test I start the simulation of the environment and execute the algorithm with a max range value set to 5. This value is very important for the observation step as from this value depends the observability of the landmark and so depends the accuracy of the estimation.

Here below we can see the result of the first test, such as the landmark and the robot pose estimation. The image is also avaliable in the resource folder with name "test1.png". Also in the Google Drive link, it is possible to see the video presentation of the simulation called "test1.mp4".



TEST 2

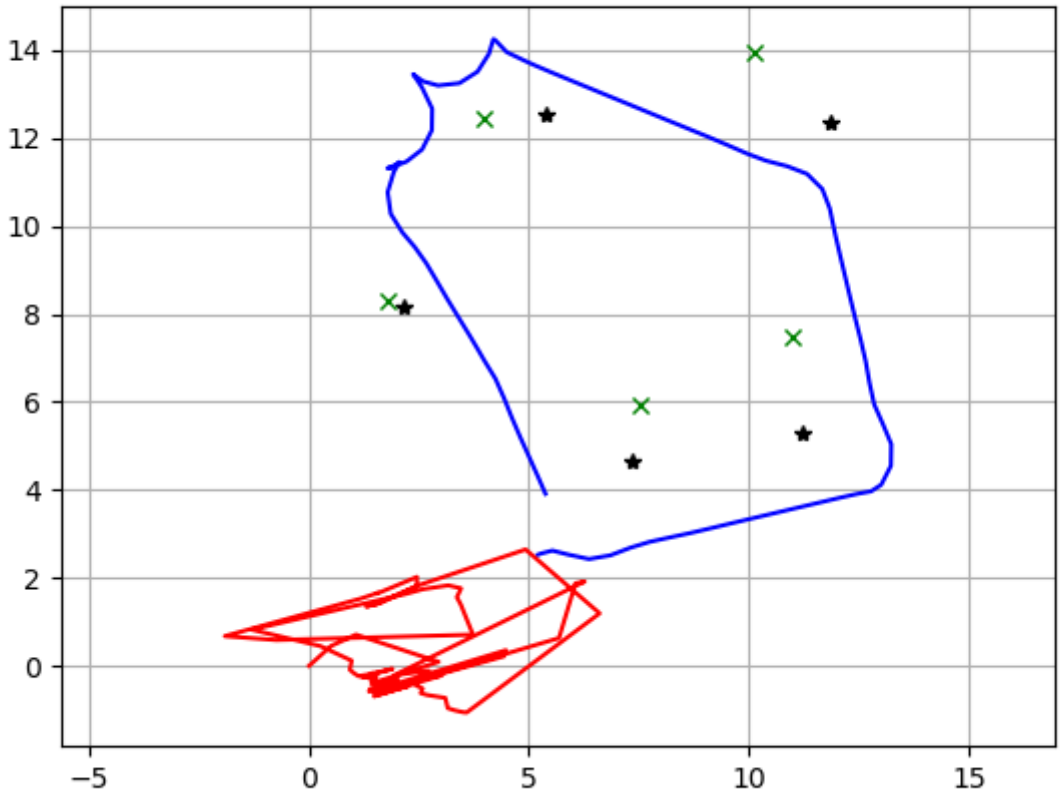
In the second test I start the simulation of the environment and execute the algorithm with a max range value set to 3.

Here below we can see the result of the second test, such as the landmark and the robot pose estimation. The image is also avaliable in the resource folder with name "test2.png". Also in the Google Drive link, it is possible to see the video presentation of the simulation called "test2.mp4".

TEST 3

In the third test I start the simulation of the environment and execute the algorithm with a max range value set to 7.

Here below we can see the result of the third test, such as the landmark and the robot pose estimation. The image is also avaliable in the resource folder with name "test3.png". Also in the Google Drive link, it is possible to see the video presentation of the simulation called "test3.mp4".



CONCLUSION

I liked very much this project because of dealing with the CoppeliaSim robotic tool and with a very important and interesting algorithm such as EKF-SLAM. The main difficulties I encountered regards the well estimation of the poses. As we can see from the tests, the first test with max range value set to 5 has the best accuracy in the landmark pose estimation. Instead in the other two tests, the estimation is worse as in the case of the lower max range value the robot does not observe very well the world and in the case of higher max range value the robot observes more the world but this makes to him confusion. Maybe one solution for a better estimation could be to let the robot navigate more in the environment and so in the future it would be very interesting to see how the algorithm behaves with more time available for the navigation. Unfortunately I did not understand why the robot pose is never well estimated,as in the original matlab code it performs very well but in the CoppeliaSim simulations does not. I think that the imlementation of the EKF-SLAM algorithm made by prof. Grisetti is well done and I am very happy to have dealt with it, to have studied it and also contribued and partecipated to this work through the conversion of all the code to python and adaptation to CoppeliaSim as it can be very useful.

BIBLIOGRAPHY

The main resources used for this homework comes from the prof. Douglas Macharet classes of Robotica Movel and from the CoppeliaSim APIs (https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm (https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm)). For the implementation of the algorithm I used the slides and code from "Probabilistic Robotics" course of prof. Giorgio Grisetti.

REFERENCES

1. CoppeliaSim: https://www.coppeliarobotics.com/helpFiles/en/apiFunctions.htm (https://www.coppeliarobotics.com/helpFiles/en/apiFunctions.htm)
2. https://github.com/verlab/dcc042-robotica-movel (https://github.com/verlab/dcc042-robotica-movel)
3. https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping (https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping)
4. https://sites.google.com/diag.uniroma1.it/probabilistic-robotics-2019-20 (https://sites.google.com/diag.uniroma1.it/probabilistic-robotics-2019-20)
5. Jupiter_File (dcc042-robotica-movel): "aula09-controle-cinematico.ipynb"