Aerial localization using the Extended Kalman Filter

Arwed Gadau
Student number 104350

Natan Viana
Student number 91615

Rodrigo Contreiras
Student number 90183

Rui Anjo
Student number 93177

***Abstract*—This paper studied the localization problem of an aerial robot. To get things going we started by studying and analysing all the different components of the Extended Kalman Filter algorithm as well as trying to understand the Robot Operating System and working with Gazebo. To test the algorithm we started by implementing it on a micro-simulator and after we got that working properly we moved on to Gazebo to evaluate it's effectiveness in resolving the problem in hand. After several different simulations, varying both the number of landmarks and their position we reached the conclusion that our approach results in good precision in the localization of the robot for three or more landmarks and that it also works better if the landmarks are distributed in a way that they form a triangle in the case of three landmarks or a square if we have four.**

***Index Terms*—Mobile robots, Localization, Extended Kalman Filter, Landmark, Laser**

I. Introduction

In this project it was decided to tackle aerial mobile robot localization using the Extended Kalman Filter (EKF), our motivation for choosing this project topic came from the fact that the localization problem appeared to us as one of the key components to make an autonomous robot, because so many robotic tasks require knowledge of the location of the robot.

This problem consists of determining the position and orientation of an aerial robot relative to a given map of the environment, with the highest accuracy possible.

One way to solve this problem is to simply start the robot in a known position and update the state of the system based on the robot actions, however this method isn't reliable because of the existence of sensor noise.

So, to do this we use several tools. Relative to the robot sensors, we will use both odometry sensors, to get an estimation of the robot current position and orientation, and laser sensors that measure the distance and angle directions of the robot to an obstacle. Relative to the map of the environment, we will rely on a set of landmarks where their position is known to us. Lastly to test our algorithm, we will implement all this first on a micro-simulator and then through ROS and Gazebo.

With all this data and methods, we will try to predict the position of an aerial robot, when it departs from a known initial position, in a known environment. This will be based not only on its velocity and direction, but according to its distance to the landmarks.

II. Methods and Algorithms

Regarding the method we used to locate the pose of our aerial robot, we chose to work with the EKF method, given that it is inspired by the well-known Kalman Filter but extended to more realistic situations such as non-linear systems. Thus, the Kalman Algorithm is based on the idea that, having the movement of a robot being described by a non-linear system, we can use a sensor capable of monitoring this movement, but with measurement noise, and calculate the mathematical expectation for the value of the sensor and robot motion prediction, as well as the covariance between true and predicted motion error. When arriving at the covariance result, we saw that it depends on the covariance of the noise parameters of the robot's movement system, and therefore, being this positive, the covariance of the error between the two diverges at each iterative instance. To solve this problem, the Kalman Filter comes into play in order to try to find a Kalman gain that brings the value of the model's hope closer to the reality of the system. Therefore, through the linearization of our system and the theoretical representation of the sensors using gradients and Jacobians, we can calculate this Kalman gain capable of correcting the prediction of the robot's location and, therefore, the covariance between the error of the current and predicted pose.

III. Implementation

A. Microsimulator

Our first implementation of the initial problem was through a micro simulator made by a theoretical code in python to simulate a real trajectory of the robot, measurements of the sensors and the calculated hopes for its supposed correction in the application in the EKF. In this type of implementation, we are based on initializing a real robot position, as well as its random movement and control noises to simulate reality and the calculations of the covariances of these matrices. Soon after, we initialized the position where the prediction should start its search for the robot. After we defined the mathematical model of the sensor that we were going to use, as well as its random measurement noise and its noise covariance matrix. Inside a loop to simulate the real system we use Gaussian distributions to multiply a number by our motion noises. After that and with all the parameters in hand, we performed the

calculation of the real pose of the robot over time through the Incremental Robot Model described in "Fig. 1".

$$
\begin{aligned}
x_k^v &= (x_{k-1}^v + \xi_k^x) + (v_k + \xi_k^v)\cos(\theta_{k-1}^v + \xi_k^\theta)\Delta t \\
y_k^v &= (y_{k-1}^v + \xi_k^y) + (v_k + \xi_k^v)\sin(\theta_{k-1}^v + \xi_k^\theta)\Delta t \\
\theta_k^v &= (\theta_{k-1}^v + \xi_k^\theta) + (\omega_k + \xi_k^\omega)\Delta t
\end{aligned}
$$

Fig. 1. Incremental Robot Model.

Then we calculate the measurement obtained by the chosen sensor, the position estimated by the robot (Mathematical Model disregarding the noise), we obtain the Linearizations of the Model, Control and Sensor matrices like "Fig. 2", we calculate the expectations of the sensors(GPS and Laser) and we perform the calculation of the Kalman gain in order to correct the expected values of the robot pose and the new covariance of the prediction error.

GPS SENSOR:

$$
y_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \vartheta_k = Cx_k + \vartheta_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \vartheta_k
$$

$$
H = C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}
$$

LASER SENSOR:

$$
S_k = h(L, x_k) = \begin{bmatrix} d_k \\ \alpha_k \end{bmatrix} = \begin{bmatrix} \sqrt{(L_x - x_k) + (L_y - y_k)} \\ \operatorname{atan}(L_y - y_k, L_x - x_k) - \theta_k \end{bmatrix} + \vartheta_k
$$

$$
H = \frac{\partial h(L, x_k)}{\partial x_k} = \begin{bmatrix} -\frac{L_x - x_k}{d_k} & -\frac{L_y - y_k}{d_k} & 0 \\ \frac{L_x - x_k}{d_k^2} & -\frac{L_y - y_k}{d_k^2} & -1 \end{bmatrix}
$$

Fig. 2. Sensor Matrix and Jacobian.

The microsimulation performed robot trajectories in different situations, turning in circles, making a "star", pentagon and random path.

The choice of sensors was made to test each of these situations and when we used the lasers we had to pre-define the location of the landmarks we wanted to use.

It is added that at the microsimulation level, the laser was able to visualize all the existing ones on the map.

B. ROS and Gazebo

When using ROS and Gazebo, we have more data regarding the laser measurements and uncertain variables. Because in the beginning, we were running into a lot of problems setting up a simulation in gazebo it was decided to use the S.T.D.R Simulator (Simple Two Dimensional Robot Simulator) in a first attempt to integrate our Filter into a little more complex environment. The Simulator was featured in the ROS Documentation and allowed us to adept the environment in a wide range of aspects. The Simulator made it very easy to change and create new maps, adept and make your custom robot with different sensor setups and a lot more. We hoped, that in this way it would be easier to debug and test the additional features like landmark recognition. In this case, we define how many laser readings the sensor will do, the maximum range, and the maximum and minimum angle of said readings. In return this means that unlike our micro simulator, the robot won't be able to detect every landmark, it will only be detecting a couple of them or sometimes it won't detect any. This brings twos major issues: which landmark is the robot detecting and is the robot detecting the landmark or is it detecting another obstacle. To overcome these problems, we took a series of predictions shown in the "Fig. 3":

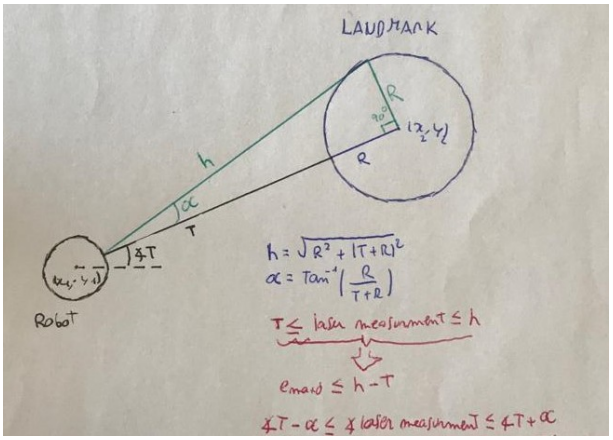

Fig. 3. Landmark angle and distance calculation: T is the theoretic distance, R is the landmark radius and α is the maximum angle difference.

Before using the data from the laser measurements, our code computes the theoretic distances and angles from the robot current position, taken from the odometry (in the case of S.T.D.R. simulator the predicted positon because the odometry always gave the real robot pose), to each center of the landmarks. Then based on the theoretic distances and the radius of the landmark, the algorithm computes the maximum distance that the laser reading could be, so that it would be considered the landmark. This way any distance between the theoretic distance and the maximum distance could be from that landmark.

IV. Experimental Results

A. Results from Micro-simulator

For the Micro-simulator we decided to separate our results considering the sensor used, the defined trajectories and, in the case of the laser, the number of landmarks and at last the kidnapping effect.

1) Using GPS sensor with a circular trajectory: We saw that by initializing the robot to an initial pose (0,0,0) and our hope at the same point on the map. The convergence of the algorithm was almost instantaneous using the GPS in the EKF. It is possible to see through "Fig. 4" this meeting between the routes. Also, by analysing "Fig. 5", it is possible to observe that even if the hope for the initial pose it's different than the actual initial pose, the robot route estimation will approximate

with the actual robot route in a short amount of time, taking account the robot parameters implemented, proving how well the EKF algorithm works.

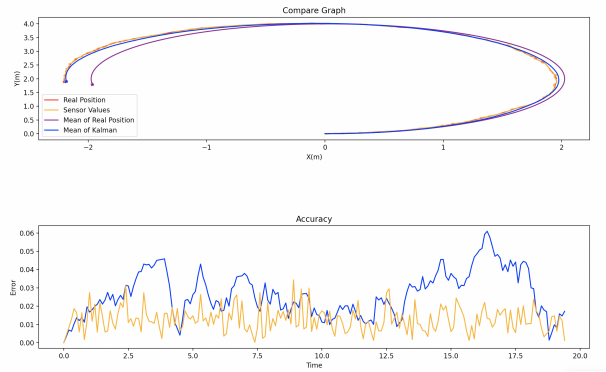

Fig. 4. GPS sensor with same initial pose.

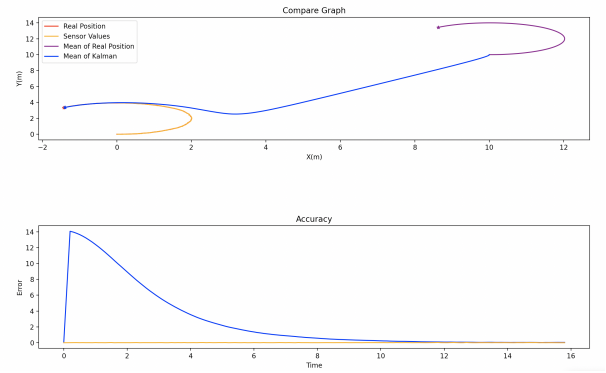

Fig. 5. GPS sensor with Different initial pose.

2) Using Laser sensor with a circular trajectory using different configurations of landmarks and with the initial hope pose different than the real: At this situation our problem is to know what is the influence of the number of landmarks and how far the robot is from them. We have 5 Situations:

- With only one landmark:
  - In this case, the influence of the landmark is still too small for the robot's movement prediction to be coherent with the real pose.
- With Two landmarks
  - In this case, the influence of two landmarks is better than the last one but still too small for the robot's movement prediction to be coherent with the real pose.
- With Three landmarks (Triangle)
  - In this case, the influence of three and mainly with a triangle disposition of these landmarks works with a good accuracy and the robot's movement prediction matches with the real pose after some time.
- With Four landmarks near (Square near)
  - In this case, the influence of four and mainly with a square disposition of these landmarks works with a better accuracy than the last one, and the robot's movement prediction matches with the real pose after a little time.
- With more than Four landmarks far (Square far)
  - The EKF also works with landmarks that are far and some time we saw the robot's movement prediction matching with the real one.

3) Using Laser sensor with different trajectories: Here we test some new paths to know the efficiency of the EKF with three different configurations:

- Circular Movement
- "Star" Movement
- Random Movement

You can see de "star" configuration at "Fig. 6":

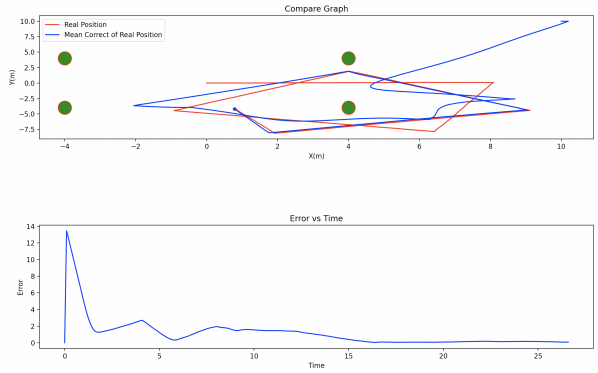

Fig. 6. "Star" Path

After these implementations we saw that, the EKF works good independent of the trajectory. But, in some cases the time that it takes to match the hope position and the real one changes.

4) Kidnapping: Here we test a particular effect that is named by kidnapping and we saw the EKF algorithm doesn't work well after the kidnapping. You can see this effect visualized in "Fig. 7".

B. Results from S.T.D.R Simulation

When implementing our algorithm we used in the micro simulator into a more complex simulation, we needed to overcome some obstacles. Firstly like mentioned in one of the previous chapters, because we only used a certain amount of laser measurements, it is likely that our robot is not seeing all the landmarks in our map. Because of that it was necessary

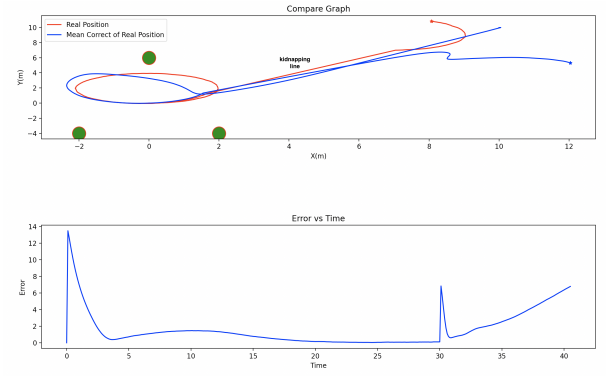

Fig. 7. Kidnapping situation

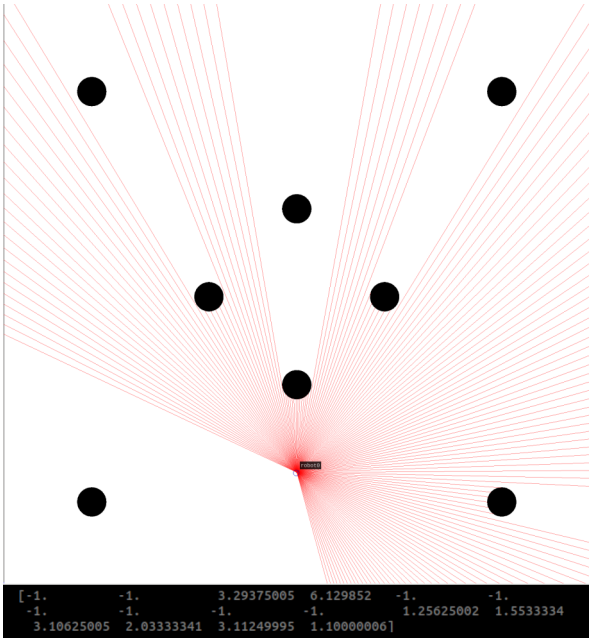

Fig. 8. Landmark recognition tests examples

to find a way of determining if the robot is sensing a landmark and which of the landmarks he is looking at. The theory of our solution for this problem was already explained previously and now needed to be tested in the environment of the S.T.D.R. Simulator. Secondly we needed to adapt our micro simulator code, so that it works with the new requirements like changing matrix sizes.

1) Landmark recognition: To test the effectiveness of our landmark recognition algorithm we placed the robot into different positions, around different maps with sets of landmarks, calculated the distance and angle to all sensed landmarks. ”Fig. 8” and ”Fig. 9” show two of those examples in two different maps with our codes output.

As it can be seen, our code evaluates every laser reading and decides if he sees a landmark or not. When a certain landmark is not sensed by any laser ray we output a -1 at this landmarks position. This can later be used to resize all matrices that are dependent of the number of seen landmarks. To us it made

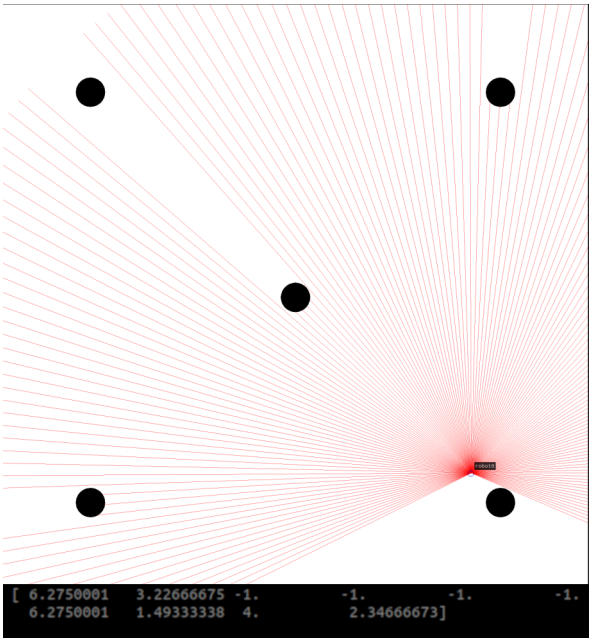

Fig. 9. Landmark recognition tests examples

the impression that this part of the code was working fine and we could move on to the next part of implementing the rest of our EKF code.

2) Adapting, Testing and Running the Filter: The second biggest change we had to make to our code was to adept to the varying size of matrices duo to the changing number of sensed landmarks. After that was done we could start testing the code inside our simulation. First we created two ROS packages with python scipts. One for moving the robot around and creating different path he could follow around the map and one for running the filter. But later we changed it to have both movement control and the filter update in one package and script to make it easier to adept and test. Our setup was the following: we created a robot with 151 laser rays, a maximum angle of 2rad and a minimum angle of -2rad for those laser rays and maps with 5 and 8 landmarks. The sensor data we needed was acquired through subscribing to the cmd_vel, laser_0 and odom topics of the robot. The robot movement was achieved by publishing to the cmd_vel topic of the robot. Important to mention here is, that the odometry readings of the simulation were without noise. Therefore we used this data to set our true position of the robot and added noise manually during our filter update. The laser sensors are implemented as hokuyo lasers which already had a Gaussian noise integrated. The movement inputs of the simulation followed ideal kinematic which made it necessary to add noise manually as well. With these prerequisites set we could start testing the algorithm. Because our algorithm had a lot of trouble predicting the position of our robot we tested a lot of different setups to figure out were our mistake was. One of those setups with the simulation and the corresponding plot of the real and estimated position can be seen in ”Fig. 10”.

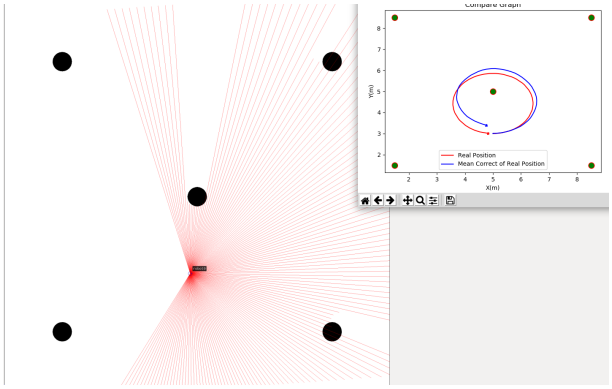

Fig. 10. Example Test Setup of the Plot and Simulation

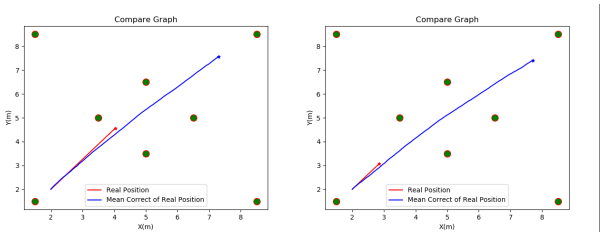

Fig. 11. Comparison of two test with different robot speeds

To better understand what was going on we made multiple simulations with varying initial position, moving and standing robot, different maps and robot paths and landmarks as well as changing speeds.

As it can be seen in "Fig. 11" in this case the variation of the speed with which the robot was moving in the map determined how fast our estimated position was moving away from the real position of our robot. This behaviour could be observed in all test. When varying the speed from low to very high values. The prediction first came closer to the real position and after surpassing a certain value went further away again. After this we continued changing the path of the robot.

It was already stated in one of the previous chapters that the number of sensed landmarks is absolutely crucial for a good performance of our position prediction. This statement can be verified by our observation in the simulation as well. In "Fig. 12" even though the performance of our filter is very poor in general it can be seen, that as soon as the robot only sees one or two landmark the predictions moves away from the real position even faster.

As for the variation of the robots route across the map (some examples in "Fig. 13") it can be said that besides the already discussed behaviour when sensing different numbers of landmarks no big influence on the performance could be observed. In "Fig. 14" another of our tests can be seen where we decided to simulate and run our filter with a standing robot. It becomes obvious that one big problem in our code has to be inside the landmark sensing part or in the update of some of the variables that depend on that sensing since the estimated position of the filter proceeds to move away further and further

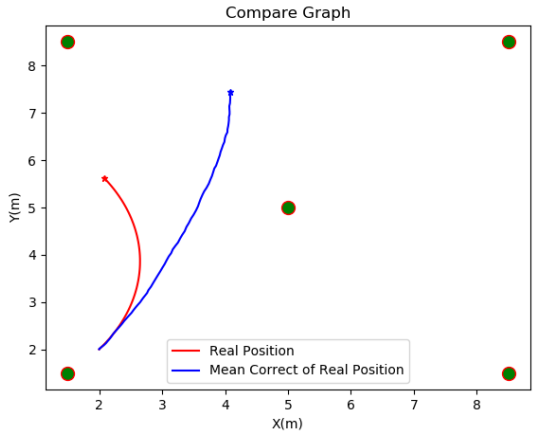

Fig. 12. Example low number of sensed landmarks

from the real position which is not moving at all. With this in mind we tried to find what in our code leads to the algorithm giving these bad results. But even though we are pretty sure on where in our code the mistake is probably happening we were not able to find the problem. Another explanation for the poor performance of our algorithm could be, since our micro simulator is working fine, a possibly very enormous error in the laser readings. Reason for that could be the rather big landmarks with a big radius we used which led depending on how the laser hit the landmark to a big error. We thought about a possibly too big manually implemented noise too. But this is very unlikely since when the robot is not moving the influence of the manual noise can be neglected. As a final suspicion we had in mind, that because of our only superficial understanding of the ROS software we made a mistake by subscribing or publishing data in a wrong way that led to the errors in our simulation. Lastly we can conclude this part of the report by saying that even though sadly we did not manage to find the reason for the poor performance of our algorithm, by searching for it we learned a lot about the behaviour of robot localisation algorithms as well as the ROS environment. Two additional way of further improving the performance could be to significantly increase the number of landmarks and at the same time reducing their size or introducing the walls of the map as additional landmarks. With that we could be able to make sure we see enough landmarks at all times and additionally reducing the error effect of the landmarks radius. But because of the limited time we were not able to follow through with these additional approaches.

Because we were not able to make our algorithm work with this simplified and easy controllable simulation in a satisfying way we stepped down from trying to implement it into a gazebo simulation and focused on improving our code instead. But it can be said that we were able to set up a gazebo simulation and link it to mavros to gather sensor data and after our code would work the next step would have been to integrate it into a gazebo aerial drone simulation.

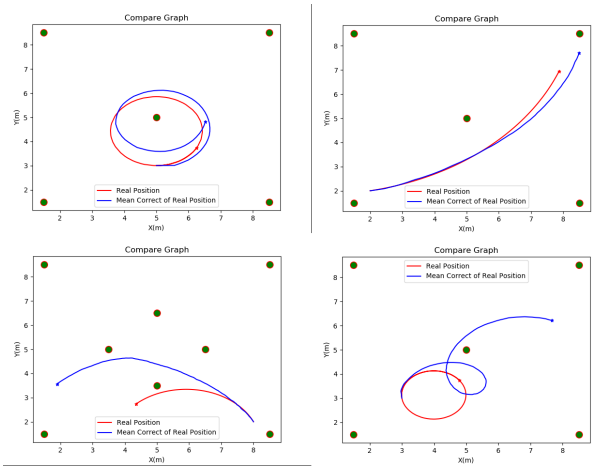

Fig. 13. variation of the path

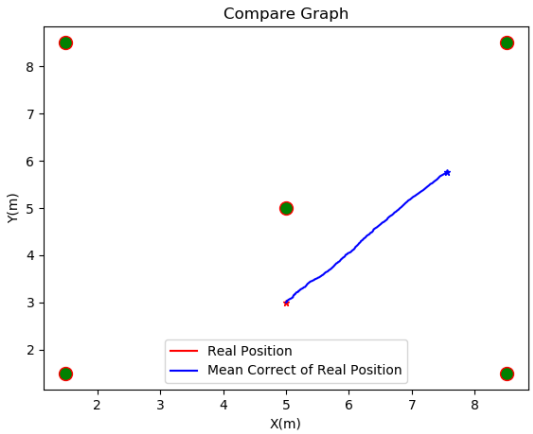

Fig. 14. Simulation with standing robot

V. Conclusions

With the realization of this report we were able to reach several conclusions in regards to the localization of a robot using the EKF. The difference in effectiveness in using different combinations of sensors such as the odometry with a GPS and odometry with a laser. Comparing both combinations of odometry with GPS and laser, firstly we noticed that by using GPS we get a better estimate because it gives us the position of the robot instead of a distance to an object like the laser does. However in the case of localization with odometry and laser sensors we noticed that the number of landmarks plays a big role in the efficiency of the algorithm, for example we noticed that with only one landmark, for most of the time, the robot can't make a good estimation of it's position, but as we increase the number of landmarks we noticed that it gets better as the number goes up and we can achieve good results at all time when it reaches three landmarks, especially if the landmarks are positioned in a triangle formation, because in that case we get triangulation. But the number of landmarks stops making a big difference performance wise after four landmarks are placed in a square formation, which means that if we go from four landmarks to five and above, the difference in terms of performance is really small. We also tried the algorithm with multiple path forms, such as circular, triangular, pentagon, star form and also a random path, and the conclusion we reached is that it works well with all paths, it's much more relevant the number and position of landmarks than the robot path. Finally, concerning the kidnapping problem we reached the conclusion, as we expected from an EKF based algorithm, that our robot would not be robust to kidnapping, which means that if we take the robot from it's pose when its moving and we put it somewhere else he cannot successfully localize himself again, because he strongly believes he is somewhere else at the time of kidnapping, and since the EKF probability density function is a Gaussian which is unimodal, it is not suited for situations where the robot pose is equivocal.

References

[1] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, MIT Press, 2005.
[2] P. Lima, and R. Ventura, Probabilistic Robotics basics Kalman Filters Particle Filters, Instituto Superior Técnico, 2020.
[3] M. Ribeiro, P. Lima, and R. Ventura, Localization Fundamentals, Instituto Superior Técnico, 2019.
[4] M. Ribeiro, P. Lima, and R. Ventura, EKF-Based Localization, Instituto Superior Técnico, 2019.
[5] Documentation - ROS Wiki, ROS, June 2022. Available: http://wiki.ros.org/