universidade
de aveiro

Mobile Robotics

Robot Probabilistic Localization using Redundant Landmarks

Pratical Work 1

By
Tiago Silva Pereira
98360

Course given by:
Vitor Santos
Nuno Lau

Robotics and Intelligent Systems Master’s

1. INTRODUCTION

The task involves localizing a robot in an environment with multiple landmarks (or beacons) along a predefined path. These landmarks have known positions, and the robot is equipped with sensors to detect and differentiate the beacons, measuring their distance and orientation relative to its current position with a certain degree of uncertainty. However, the system should be adaptable to function even with partial information. We employed Extended Kalman Filter, incorporating appropriated motion and sensor models. The goal is that we can continually update the robot's estimated position by integrating sensor measurements with the predicted state using EKF.

1.1. FUNCTIONS AND METHODS

In this work, i made some support functions to help in the organization the code. Must of the functions wore created by myself from ground, but there implementation was based on the class functions from EKF.

We can start by telling the name of this functions

- linear_path
- pchip_path
- velocities
- EKF_Generate
- plot_ekf_data
- motionmodelDD_TRI
- plot_model_velocities
- ekf

These functions are used in the *rm_98360()* function.

1.2. PATH CALCULATION

1.2.1. linear_path

This function creates the linear path between all the beacons and calculates the number of steps that requires to get from one beacon to another.
The number of steps is calculated using the approximation to the integers number of the distance between the two sequential beacons.That is defined by

$$\frac{\sqrt{(x\_(2)-x\_(1))^2+((y\_(2)-y\_(1)^2}}{(Dt*Vn)} \tag{1.1}$$

Now that we have the number of steps, we are going calculate the x and y values, between each of the pairs of beacons, getting the coordinates of the steps.

1.2.2. pchip_path

This function creates the smother path possible, using the Hermite polynomial interpolation, using the x coordinates calculated at the last point to obtain the y coordinates. The interpolated trajectory is defined has the ideal planned trajectory, and is going to be used as the ground truth, to all the comparisons and analysis.

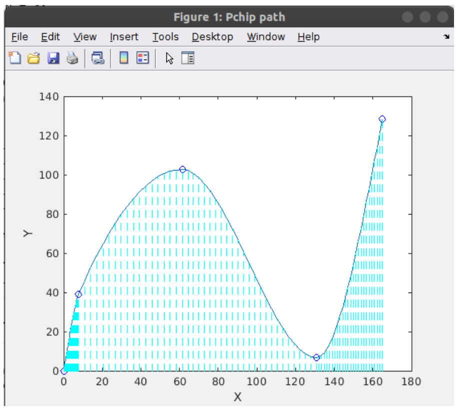

Figure 1.1: Path between each beacon for the hermite plonynomial interpolation

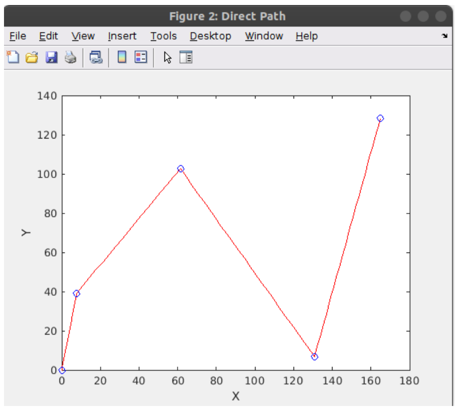

Figure 1.2: Path between each beacon for the linear path

1.2.3. plot_path

This is a function that we use to plot all the information that we found in the last functions. I created a different function just by developing facility. We got a plot from the linear and pchip paths and the comparison between them.

1.3. LINEAR AND ANGULAR VELOCITIES

1.3.1. velocities

This function will calculate the linear and angular velocities between each pair of beacons. To resolve this problem, we started by calculating the distance between the interpolated points, which came from the Hermite function.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1.2}$$

We also need to calculate the θ between each interpolated point.

$$\arcsin \frac{y_{t+1} - y_{t-2}}{d} \tag{1.3}$$

Now we can get $\Delta\theta$. To achieve the values of linear and angular velocities, now, we just need to apply the following equations.

$$v_t = \frac{d_t}{Dt} * (1 + NOISE) \tag{1.4}$$

$$w_t = \frac{\Delta\theta_t}{Dt} * (1 + NOISE) \tag{1.5}$$

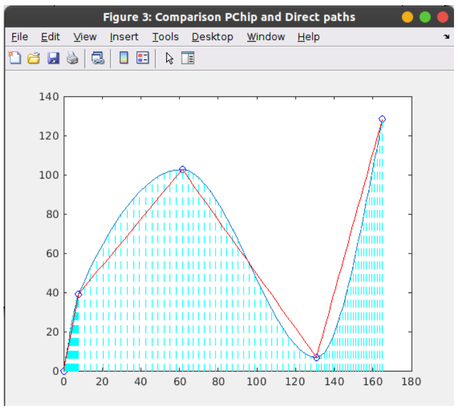

Figure 1.3: Comparison between both paths

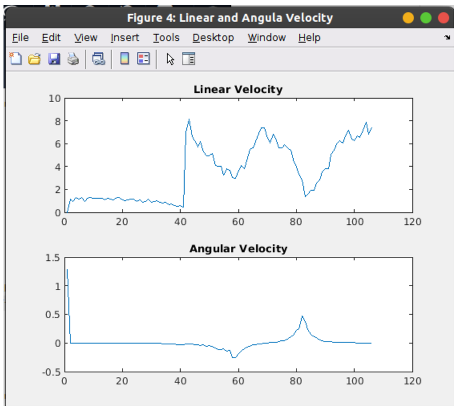

Figure 1.4: Linear and angular velocities in system

We multiplied the formula from the linear and angular velocities to the noise, that is, a random value from 0 to 1, in a way that we can apply some noise to our model and find what is his reaction to noise.

1.3.2. plot_velocites

This function is used to plot the information obtained in the last function.

1.4. EXTENDED KALMAN FILTER

1.4.1. ekf

We use this function to do all the calculations on the estimated position and orientation from the robot while doing the path defined early.

We start by applying the motion model to the robot to get the initial estimation of the robot's position.

$$x_{state} = x_{state} + (V_{in}(1) + D_n(1)) * t * cos(\theta_{state}) \tag{1.6}$$

$$y_{state} = y_{state} + (V_{in}(1) + D_n(1)) * t * sin(\theta_{state}) \tag{1.7}$$

$$\theta_{state} = \theta_{state} + (V_{in}(2) + D_n(2)) * t \tag{1.8}$$

Now we are going to find the uncertainty of the robot measures, using the matrix of predicted covariance, or uncertainty and noise.

To find the uncertainty, we will need to resolve this equation.

$$Jfx * P_t * Jfx' + Jfw * Q * Jfw' \tag{1.9}$$

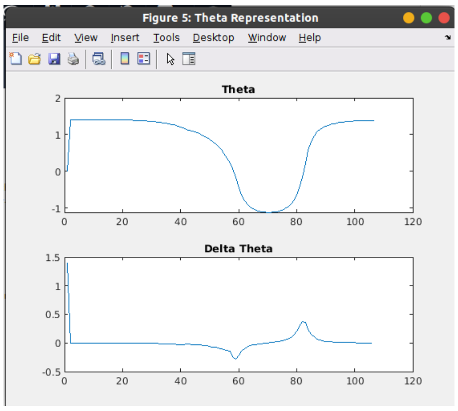

Figure 1.5: θ and θ variation

$$Jfx = \begin{bmatrix} 1 & 0 & -\delta_t * input * sin(est_\theta \\ 0 & 1 & -\delta_t * input * cos(est_\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

$$Jfw = \begin{bmatrix} -\delta_t * input * cos(est_\theta & 0 \\ -\delta_t * input * sin(est_\theta & 0 \\ 0 & \delta_t \end{bmatrix} \tag{1.11}$$

We also need to calculate the Jacobian of the sensor model, applying the following equation.

$$\begin{bmatrix} a & b & 0 \\ c & d & -1 \end{bmatrix}$$

(1.12)

$$a = \frac{x - xym(1)}{\sqrt{(x - xym(1))^2 + (y - xym(2))^2}} \tag{1.13}$$

$$b = \frac{y - xym(2)}{\sqrt{(x - xym(1))^2 + (y - xym(2))^2}} \tag{1.14}$$

$$c = \frac{y - xym(2)}{(\frac{(y-xym(2))^2}{(x-xym(2))^2} + 1)(x - xym(1)^2)} \tag{1.15}$$

$$d = \frac{1}{(\frac{(y-xym(2))^2}{(x-xym(2))^2} + 1)(x - xym(1)^2)} \tag{1.16}$$

Using the Jacobian we can now find the measurement error and erros covariance.

$$S = J * P_t * J' + R \tag{1.17}$$

$$K = P_t * J' * S^{-1} \tag{1.18}$$

To finish we need to calculate the new estimated position and orientation o the robot

$$x_{t+1} = x_t + K * (error) \tag{1.19}$$

$$P_{t+1} = (I - K * J) * P_t \tag{1.20}$$

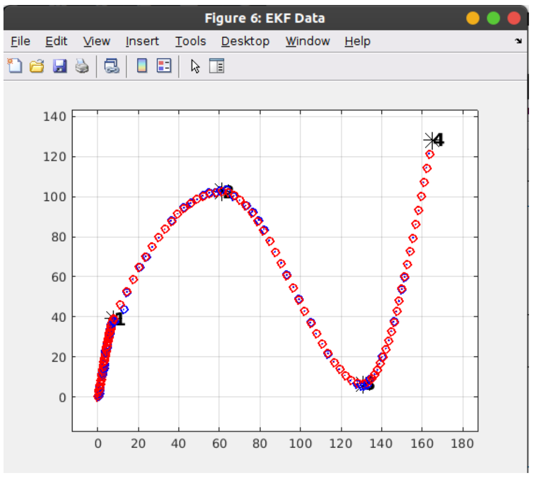

Figure 1.6: Real and predicted positions

1.4.2. EKF_generate

This is one of the most important function in our project. This function is used to generate and prepare the values used in the Extended Kalman Filter.

In the first part of the program, we use two types of variables. Two of these variables are used to assist the prediction, and the others are used to verify the accuracy of the Kalman Filter. This creates all the necessary data to be used afterward. We start by generating the collection of true robot poses, that is, the true state used in the EKF. After that we are going to create the observation data.

Now, the function enters the second part. Here, the function will obtain the true and estimated position, the range and bearing of the beacons at all times, and the control input. We can see the results of the use of this function, on the figure 2.6

In this plot, we can see the estimated values and the true position of the robot. We can observe that we achieved good estimation of the position, but when we are closer to the tops and down's, that are defined by the beacons, we see that there is differences between the real orientation and the estimated orientation.

1.5. DIRECT DRIVE AND TRICICLE

1.5.1. motionmodel_DD_TRI

In this function we are going to calculate the values to the angular and linear velocities of the virtual Direct Drive and Tricycle robots. To obtain these values, we had to take the following into account:

$$DD_{rightwheel} = \frac{v + w * \frac{L}{2}}{r} \tag{1.21}$$

$$DD_{leftwheel} = \frac{v - w * \frac{L}{2}}{r} \tag{1.22}$$

$$TRI_{velocity} = v - w * L \tag{1.23}$$

$$TRI_{\alpha} = arcsin(\frac{w * L}{TRI_{velocity}}) \tag{1.24}$$

1.5.2. plot_model_velocities

This function serves to plot all the data obtained before

1.6. ERRORS

By calculating the norm of the difference between the values of the predicted states and the real positions, we can find the errors between them.

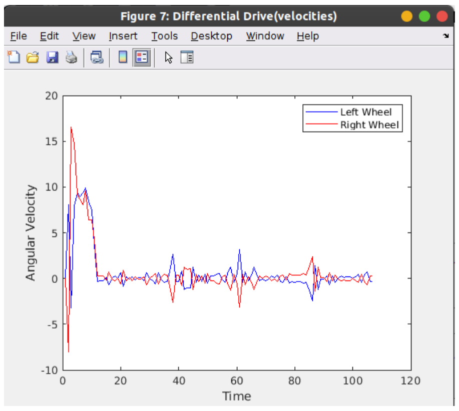

Figure 1.7: Direct Drive wheels velocities

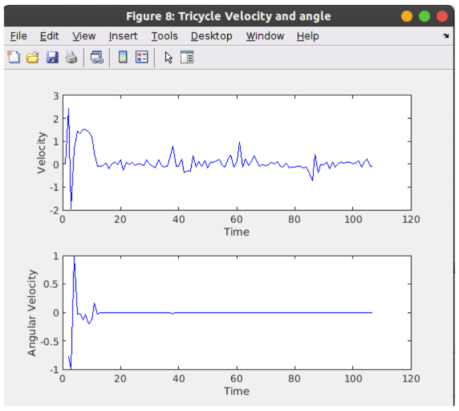

Figure 1.8: Tricicle wheels angle and velocities

.	Average Error	Smallest Error	Maximum Error	Deviation
x	0.214343	0	1.503143	0.210749
y	0.294070	0	1.726984	0.337704
θ	0.072853	0	1.430293	0.198748

1.7. CONCLUSION

In summary, this project provided valuable insights into the Extended Kalman Filter (EKF) and its practical application in predicting the position of a robot. Although the EKF is a robust system capable of accurate predictions, it is not infallible. As evidenced by our results, discrepancies between the estimated and real paths exist due to sensor noise and the inherent uncertainty in the robot's movement, factors that the EKF does not fully account for.

Despite its imperfections, the EKF remains a powerful tool for estimating robot positions. However, it's essential to acknowledge its limitations. The similarities between the results for the Direct Drive and Tricycle robots stem from the comparable equations used to calculate velocities. The primary discrepancy lies in the longer length of the Tricycle and the presence of a third wheel.

In conclusion, while the EKF offers valuable insights into robot localization, it's imperative to recognize its limitations and continuously refine its application to achieve more accurate results.