Robot Probabilistic Localization using Redundant Landmarks

Professor Vitor Santos - Mobile Robotics

David Ornelas
University of Aveiro - MRSI
david.ornelas@ua.pt
109802

***Abstract*—This article presents a probabilistic localization approach for a robot navigating in an environment with several landmarks. The goal is to estimate the robot's position and orientation along a path defined by the intermediary landmarks until it reaches the last one. The Extended Kalman Filter (EKF) probabilistic model was used for the localization to estimate the robot's position (x,y,theta). With the estimated position and the help of the kinematic models of both the differential drive robot and the tricycle robot, it was performed the calculus of the angular velocities of the wheels and the direction of the tricycle steering wheel.**

***Index Terms*—localization, landmarks, EKF, kinematic models**

I. Introduction

Localization is an important tool to determine the position and orientation of agents or objects within its environment. In robotics, localization is fundamental to allow the robot to navigate and perform his role effectively. In this environment it was considered multiple landmarks, with known positions, along a path that starts at a reference point and after passing through all the middle beacons stops at the last one.

To perform the robot localization it was assumed that it had sensors to identify each beacon and measure their distance and orientation to the robot current position, with an associated uncertainty. As indicated in the assignment there were cases were some beacons would not be detected, therefore we had to adapt our model to avoid redundancy and use the maximum amount of information available to get an accurate localization.

The Extended Kalman Filter (EKF) with the respective motion and sensor models will provide the estimated robot location along the trajectory defined. It could also be done deterministically using other approaches, such as trilateration and/or triangulation. Since it was recommended by the assignment instructions to use the Extended Kalman Filter, it was necessary to measure the readings of the distance and direction from the beacons

To conclude the localization process, after estimating the robot's position using the filter, the next step was to apply the kinematic models of the differential drive (DD) and the tricycle (TRI) robots. This allowed us to calculate the angular velocities of the left and right wheels of the DD robot and the angular velocity of the rear wheels and the direction of the steering wheel of the TRI robot.

A. Guide

To provide guidance on this article, the following topics will be discussed in the subsequent sections:

- Trajectory points
- Sensor readings
- Linear and Angular velocity (Control)
- Observation Data
- Extended Kalman Filter
- Kinematic Models of DD & TRI robots

These will be the overall discussion of this article and the center focus will be the results from the EKF estimated points in comparison to the trajectory defined.

B. Input variables

Therefore to prepare the environment of the robots several variables were defined before the analysis of the localization. They had to be considered to perform the trajectory calculations and provide uncertainty to the probabilistic localization These default variables were:

- num_beacons - number of beacons (4)
- delta_t - sensor sampling time interval (1s)
- radius_wh - radius of the wheels of the robot (0.15m)
- dist_wh - distance of the wheels (1m)
- linear_vel - linear velocity of the robot assumed (5m/s)
- sigma_V - uncertainty of the linear velocity (0.1m/s)
- sigma_W - uncertainty of the angular velocity (0.1m/s)
- obsNoise_dist - uncertainty of the distance of the observation (0.25m)
- obsNoise_ang - uncertainty of the orientation of the observation (0.1rad)

C. Auxiliar functions

To optimize the code and to make it simpler to understand, several auxiliar functions were developed and some were adapted to create a program similar to the guide on the subsection I-A. The most important and primary functions were:

- BeaconDetection (given)
- get_traj (developed)
- control_input (developed)
- observation (developed)

- run_EKF & EKF (adapted)
- DD_model & DD_draw (developed)
- TRI_model & TRI_draw (developed)

There were also some auxiliar functions for the EKF filter named motion,sensor and jacobian, that will be detaily explained on the subsequent sections.

II. Landmarks & Trajectory

To define the trajectory of the robot, several assumptions were made. Firstly the initial position of the robot was assumed to be (0,0,0) while the end goal would be the last beacon on the readings. The trajectory was computed with a linear velocity of 5m/s and a sampling interval of 1s. Additionally to demonstrate the program functionalities, it was considered 4 landmarks on the trajectory.

To optimize and simplify the program, the mathematics for the trajectory, were performed in a auxiliary function named **get_traj**. This function returns the trajectory points (x,y,theta) and takes the following inputs:

```
traj_points = get_traj(landmark_points,
    ↪ linear_vel, delta_t)
```

To evaluate the trajectory, with the inputs mentioned, the following steps were used on the function **get_traj**:

1) Read the positions of each beacon (x,y) based on the sensor measures with the use of the function Beacon-Detection, provided in the assignment. This function requests the number of beacons (N) as input and returns the position of each beacon, as well as, the distance and orientation to each beacon with uncertainty.
2) Calculate distance from each beacon to the next one with the euclidian distance:

$$\sqrt{(Bx_{n+1} - Bx_n)^2 + (By_{n+1} - By_n)^2} \qquad (1)$$

3) To obtain the amount of intermediate points between each set of beacons distance, the distance was divided by the linear velocity and time step assumed.
4) With the use of linspace function, and the amount of intermediate points calculated, all x points of the trajectory were determined.
5) To obtain the corresponding y values for the previously calculated x points, the pchip() function was used, which is the Piecewise Cubic Hermite Interpolating Polynomial.
6) In addition, the orientation of each point was calculated with the following expression that corresponds to the angle between the vector of each point on x-axis:

$$\theta_i = \text{atan2}(y_{i+1} - y_i, x_{i+1} - x_i) \qquad (2)$$

For the assumption (4 beacons), previously mentioned, to analize some results, there was an output of 75 steps without taking into account the initial position, for a linear velocity of 5m/s and 1s for each interval step. The plot of the path of the robot is illustrated in the figure 1.

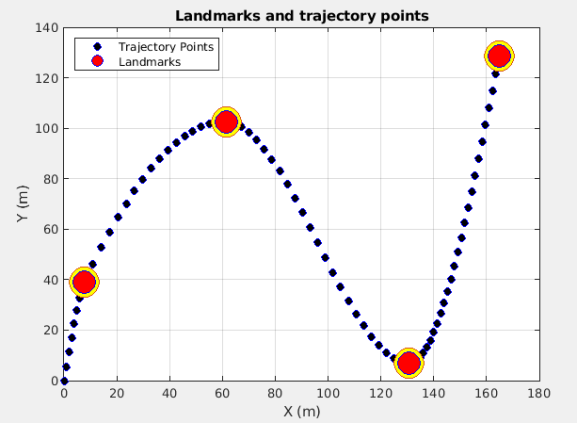

Fig. 1. Trajectory with Piecewise Cubic Hermite Interpolating Polynomial

III. Linear & Angular Velocities

Moving on to the next step, with the trajectory computed, the linear and angular velocities were evaluated on the **control_input** function. This function returns the linear and angular velocity with noise and takes as arguments the following:

```
vel_noise = control_input(num_steps,
    ↪ traj_points,delta_t,sigma_V,sigma_W)
```

The arguments are all already defined, the number of steps was calculated with the previous points from the trajectory, the trajectory points as well. The time interval and the uncertainties are already assumed at the start.

To calculate the linear and angular velocity the following equations were used:

$$\text{V} = \frac{\Delta d}{\Delta t} \qquad \text{W} = \frac{\Delta \theta}{\Delta t}$$

The Δt is the distance between the next point and the current position, while the $\Delta\theta$ is the difference of the next orientation and the current orientation. In the figure 2 we can see the result of the velocities with noise for the previous trajectory.

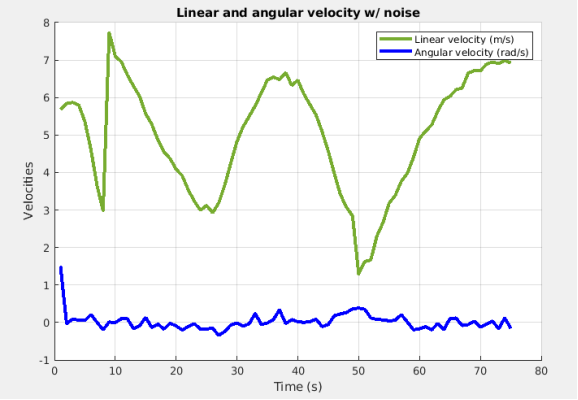

Fig. 2. Linear and Angular velocity with noise

IV. Observation data

Before using the probabilistic localization filter (EKF), we need to gather more pre-process intel. One of the data needed is the observation data, that corresponds to the distance and orientation of the current point to each beacon, and the respective beacon id. To perform a trust worthy localization and get better results for each point, the sensor readings were done to **all available beacons** in the environment

The function **observation** returns the observation data needed, and has the following arguments:

```
obs_data = observation(num_beacons,
    ↪ traj_points,obsNoise_dist,
    ↪ obsNoise_ang)
```

The arguments provided are the number of beacons used in the path, the trajectory points, and the noise of the range and bearing to the beacons. The function stores the readings from the BeaconDetection function needed for the observation, and removes any '**Nan**' readings provided by the beacons, in case a beacon is to far from the current position or the sensor couldn't read his value.

To remove this unwanted values it was used an auxiliar function from matlab called 'isnan', with this every wanted sensor reading was given in the observation data.

The following figure 3 represents the amount of beacons used for each step:

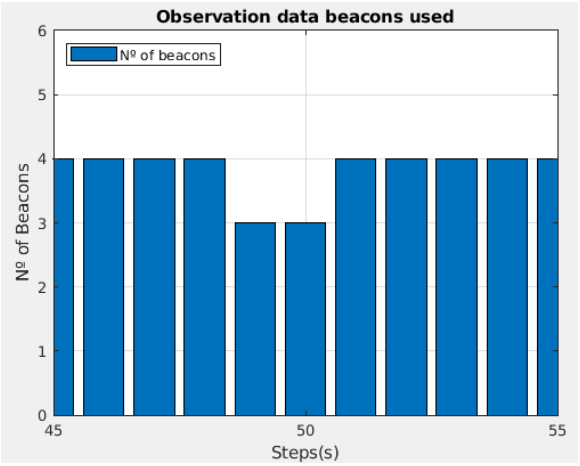

Fig. 3. Linear and Angular velocity with noise

To be more readable, it was taken into account only a few steps of the trajectory as example. As we can see, when the robot reaches one of the beacons on step 50, the sensor readings remove that beacon because it was close to it, and keeps the localization with just 3 beacons for those steps.

V. Extended Kalman Filter

A. Pre-process

Futhermore, with all the pre-process almost done, there was just missing the covariance noise matrix to perform the filter, the input noise matrix:

$$Q = \begin{bmatrix} \sigma_V^2 & 0 \\ 0 & \sigma_W^2 \end{bmatrix}$$

And the observation noise matrix:

$$R = \begin{bmatrix} w_r & 0 \\ 0 & w_\phi \end{bmatrix}$$

It was also necessary to create variables to store the next state estimate, that depends on the current state, and the process state covariance matrix. To run the filter all these variables were loaded into a loop that stores the output from the filter, this loop was stored in the function **run_EKF** that takes as input all the pre-process done previously

```
run_EKF(num_steps,control_vel,obs_data,
    ↪ landmark_points,delta_t,Q,R_i)
```

B. Filter adaptation

To load the data into the filter, it was adapted to our environment the function from the author Shoudong H., named **EKF** that consists on the process of the Extended Kalman Filter.

Before getting into detail of the filter, the function was adapted so that the observation models, the innovation covariance matrix, the jacobian and the Kalman Gain had variable dimensions according to the number of beacons used at each step. For that it was necessary to expand all this matrices to N.

The filter is divided into two main steps: the prediction and the update step. **The prediction step** focus on estimating the next state of the system based on the current state and control input and estimating the covariance matrix based on the jacobian matrices of the state and the noise.

This step as the following equations:

- Motion Model Estimate:

$$\bar{x}k + 1 = f(\hat{x}k, u_k, w_k)$$

- Predicted Covariance Matrix:

$$\bar{P}k + 1 = Jfx(\hat{x}k, u_k, w_k)P_k Jfx^T(\hat{x}k, u_k, w_k) + Jfw(\hat{x}k, u_k, w_k)Q_k Jfw^T(\hat{x}_k, u_k, w_k)$$

Where the Jfx corresponds to the jacobian of the state and the Jfw to the jacobian of the noise.

In the **update step**, with the sensor observations to the environment the state estimate is improved. The prediction is based on the known landmark positions, on the update the innovation is calculated, which is the difference between the predicted and the actual observations (this is done for [-pi,pi]). Then the jacobian matrix is calculated based on the state variables and the Kalman gain is computed using the predicted covariance matrix and the predicted observation noise covariance matrix.

The state estimate is updated using the innovation and the Kalman gain, and the covariance matrix is updated using the

Jacobian matrix and the Kalman gain. As seen in the following expressions: - State estimate

$$\hat{x}k+1 = \bar{x}k+1 + K(z_{k+1} - h(\bar{x}k+1))$$

Where the estimated state corresponds to the predicted state plus the Kalman gain, and inside the brackets its the innovation. While the Covariance matrix is the predicted covariance matrix minus the Kalman gain and the innovation covariance gain.

- Covariance Matrix estimate

$$Pk+1 = \bar{P}_{k+1} - KSK$$

The resulting updated state estimate and covariance matrix are then used as the initial values for the next prediction step. This process is repeated for each step as new observations become available, resulting in an improved estimate of the system state over the path.

VI. Estimated Results

The obtained results of the Kalman Filter can be compared with the trajectory defined on the figure 4

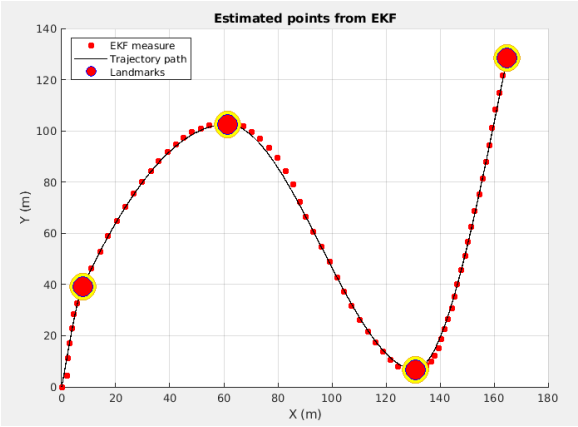

Fig. 4. EKF estimate

As visualized in the figure, the results from the filter were very closed to the trajectory, we had some small shitfs when the orientation was bigger, for example in the initial position (0,0) where the robot has orientation 0º and has to rotate to the first point. When the robot reaches the 2nd landmark, it has a major rotation to the opposite side, so the robot also starts to shift from the path. The same happens in the 3rd landmark. But regarding the small shift the performance results from the EKF model were good.

VII. Kinematic Analysis of Wheel Angles

With the estimated position it was calculated the angular velocity of the wheels from the differential drive robot (DD) and the tricycle robot (TRI).

For this it was created 2 functions called **DD_model** and **TRI_model** to calculate the required. The results from the expressions used on the models were:

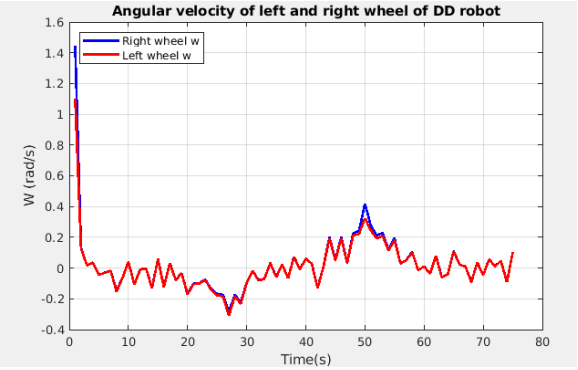

Fig. 5. Angular velocities of the right and left wheel

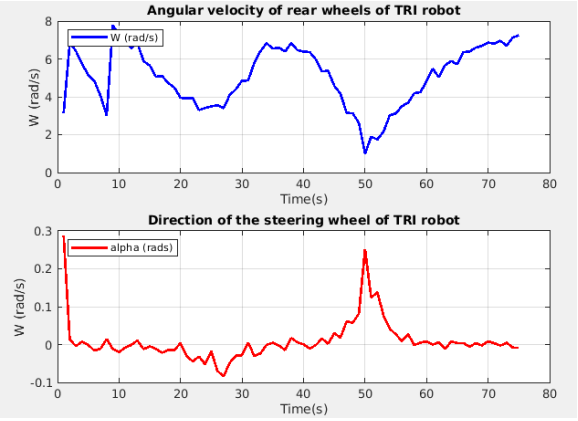

Fig. 6. Angular velocities of the rear wheels and direction of steering wheel

VIII. Additional Features

In order to run the program it was added an interactive mode, to get a more enriched experience, and visualize all the process on the display, and the plots used.

To use this mode the user must use this command line:

```
run: rm1_109802('i_mode', 1)
```

IX. Conclusions

Overall, the article describes in detail what was required from the assignment, but it has room for improvements, due to the time schedule it was not possible.

One of future additions to the work could be the animation of the robot along the path. For more comparison it could also be implemented deterministic localization methods and compare the results with the filter used.

To conclude, the article provides a good performance of localization using the Extended Kalman Filter on this environment and allowed to understand more about mobile robotics and each robot model works.