Probabilistic Robotics Lab 4
Extended Kalman Filter

Rodrigo Caye Daudt

I. Introduction

THIS report describes the work done for lab 4 on the Probabilistic Robotics module, for which an Extended Kalman Filter (EKF) was developed in Python to work with ROS simulations. Section II briefly describes EKFs and gives some information about our implementation. Section III shows and analyses the results obtained by using the developed filter. Section IV closes this work with the main observations and conclusions.

II. Extended Kalman Filter

The Extended Kalman Filter is a localization filter used to merge the different types of data available to a robot's controller. The EKF uses a Gaussian vector to describe the robot's state along with the estimated uncertainty at any point in time. EKFs are therefore unimodal filters, unlike the Bayes filter or the particle filter, and need a decent initial estimation of the robot's state.

The first step in applying an EKF is to model the behaviour and measurement of the robot. Our model for the behaviour of the turtlebot is defined in Equations 1 to 7. These equations model the prediction of the position of the turtlebot given a previous position, a control signal and an odometry measurement. The equations also model the measurements performed by the turtlebot at a given state. More information about the modelling can be found in the pre-lab report.

$$\hat{x_k} = \begin{bmatrix} \hat{x}_{r_{k-1}} + \Delta x \cdot cos\theta_{k-1} - \Delta y \cdot sin\theta_{k-1} \\ \hat{y}_{r_{k-1}} + \Delta x \cdot sin\theta_{k-1} + \Delta y \cdot cos\theta_{k-1} \\ \theta_{k-1} + \Delta\theta \end{bmatrix} \quad (1)$$

$$A_k = \begin{bmatrix} 1 & 0 & -\Delta x \cdot sin\theta_{k-1} - \Delta y \cdot cos\theta_{k-1} \\ 0 & 1 & \Delta x \cdot cos\theta_{k-1} - \Delta y \cdot sin\theta_{k-1} \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

We define ρ'_r as

$$\rho'_r = \rho_w - x_r cos\phi_w - y_r sin\phi_w \quad (3)$$

If $\rho'_r >= 0$:

$$h(x_k, v_k) = \begin{bmatrix} \rho_w - x_r cos\phi_w - y_r sin\phi_w + v_\rho \\ \phi_w - \theta \end{bmatrix} \quad (4)$$

$$H_k = \begin{bmatrix} -cos\phi_w & -sin\phi_w & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad (5)$$

If $\rho'_r < 0$:

$$h(x_k, v_k) = \begin{bmatrix} -\rho_w + x_r cos\phi_w + y_r sin\phi_w + v_\rho \\ \phi_w - \theta + \Pi \end{bmatrix} \quad (6)$$

$$H_k = \begin{bmatrix} cos\phi_w & sin\phi_w & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad (7)$$

Once we have data from the internal sensors, we can apply the prediction step of the EKF. This step is defined by the block of equations starting at Eq. 8. This step predicts a new position as best as possible with the available information. In this step the covariance matrix is also modified to incorporate the new uncertainty from the motion estimation.

$$x_{k|k-1} = f(x_{k-1}, u_k, w_k) \quad (8)$$

$$P_{k|k-1} = A_k P_{k-1} A_k^T + W_k Q_k W_k^T \quad (9)$$

Once data from the external sensors is available we can continue with the EKF operations. Since we are comparing our measurements to a line based map, we need to compare the measured lines to the map lines and find the best fit. For this, we use the Mahalanobis distance to calculate in a fair way the distance between the two lines. Mahalanobis distance is used to weigh differently the distance ρ and the angle ϕ of the lines to obtain a more fair comparison. If no line is a good fit for the measured line, the measurement is discarded. This step is accomplished by the operations described in the block of equations starting at Eq. 10.

One extra step was taken here to avoid matching measurements to ghost lines. Since the polar representation of the lines contains no information about the start and the end of the lines, we separately calculated the size of the measured and map lines and compared these values to increase the precision of the data association step. If the measured line was bigger than the best match in the map lines, the measurement would be discarded since that was not possible. An error margin of 10% was used to cope with the measurement errors and the imprecision of the split and merge algorithm.

$$v_{ij} = z_i - h(x_{k|k-1}) \quad (10)$$

$$S_{ij} = H_j P_{k|k-1} H_j^T + R_i \quad (11)$$

$$D_{ij}^2 = v_{ij}^T S_{ij}^{-1} v_{ij} \quad (12)$$

$$Take\ the\ smallest\ D_{ij}^2\ if\ D_{ij}^2 < \chi_{\rho,\phi}^2 \quad (13)$$

Finally, once the data association has been performed and the best fit for each measured line has been found, we can

perform the update step. This step merges the predicted state with all the measurements, using all the available data to calculate the best prediction with the given data. Once again, the state of the robot and the covariance matrix are modified, resulting in a more precise state estimation and to express the uncertainty of the current state estimation.This step is accomplished by the operations described in the block of equations starting at Eq. 14.

$$v_k = z_k - h(x_{k|k-1}) \tag{14}$$
$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{15}$$
$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{16}$$
$$x_k = x_{k|k-1} + K_k v_k \tag{17}$$
$$P_k = (I - K_k H_k) P_{k|k-1} (I - K_k H_k)^T + K_k R_k K_k^T \tag{18}$$

These steps were performed in a loop as fast as the computer and the sensor data allowed in order to obtain a very good state estimation at all times.

III. Tests and Results

The implemented EKF was tested using pre recorded data from a turtlebot simulation in a known environment. This simulation is shown in Fig. 1, where four different stages of the simulation are displayed in 1(a) to 1(d). In the images, we can see the following things:

- The map lines in green
- The trajectory given only by the odometry in red arrows
- The trajectory corrected by the EKF in dark blue arrows
- The uncertainty at the current state as a light blue ellipse
- The measurements made by the turtlebot relative to the EKF predicted state as blue dots
- The lines obtained from the split and merge function using the external sensor data as red lines

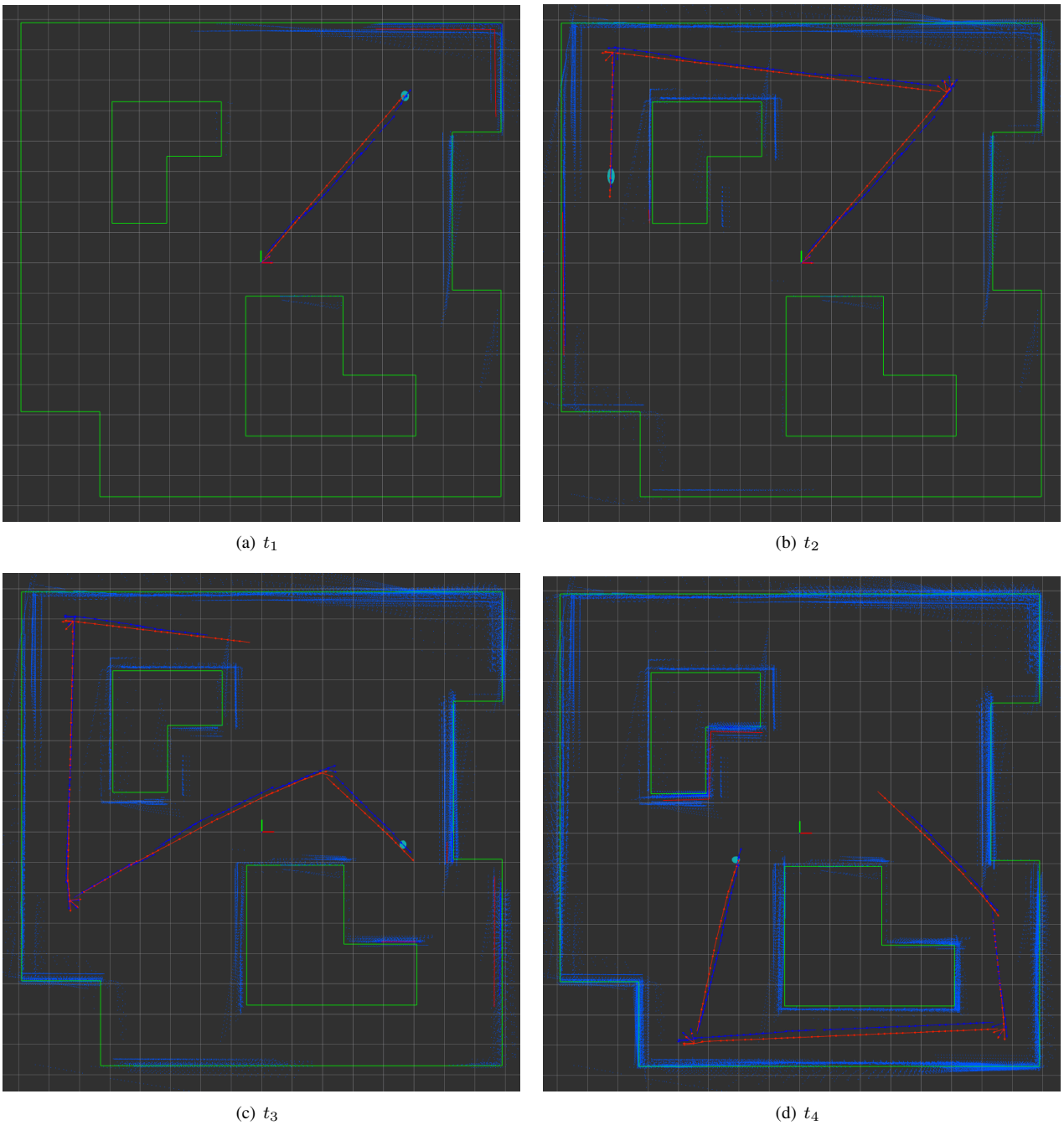
We can see that the EKF is a very good way to merge all the available data into one coherent state prediction, as is demonstrated by how the blue dots are close to the map lines. We can see that the uncertainty at every step is very small, which helps with the robot's navigation.

During the simulation we could see that the uncertainty regarding the robot's state grew when there was not much data from the external sensors, but the state prediction would recover quickly once a wall was observed once again. We could also observe that the data would be very poor when the robot turned, seen that at those moments the observed data would differ dramatically from the map lines. But once again, the state would recover very well once the robot had walls to observe and match to the known map.

IV. Conclusion

In this work we developed an EKF to help us predict the current state (position and direction) of the turtlebot. We observed that although being mathematically complex, the operations performed in an EKF are not too complex and result in a way to associate different types of data with relatively low computational effort compared to other options such as the Bayes filter or the particle filter. It is computationally efficient since it compresses all the uncertainty into the parameters of the Gaussian vectors instead of fully describing an arbitrary probability density function.

We also observed no problems regarding the stability of the EKF, since it performed well in all runs of the program. That is not the case for a particle filter, for example, which being a stochastic filter may have weird behaviours if not enough particles are used. This is mainly a consequence of the EKF being a deterministic filter, i.e. it uses no random sampling of any kind.



(a) t_1 (b) t_2

(c) t_3 (d) t_4

Fig. 1. Simulation of the EKF at four different simulation times