Simultaneous Localization, Mapping and Moving Object Detection

Attila Lengyel, 4220455

Introduction

Simultaneous Localization and Mapping, also known as SLAM, is a popular and well-researched problem in the robotics field. The problem deals with the issue of estimating the robot's location in an unknown environment based, while building a map of this environment consisting of landmarks, based on the robot's own (range-bearing) measurements. This problem can be solved in both a probabilistic way, in which the robot pose and landmark locations are modelled by probability densities and the estimated locations are calculated in a Maximum Likelihood way, and an optimization way, which calculates the robot translation and rotation that gives the smallest least-squares location error with respect to the observed landmarks' positions. In recent years, the probabilistic approach has received more and more attention due to its elegant and computationally feasible solution. Implementations often include a Kalman Filter or Particle Filter to optimally combine a motion model and measurement model.

Various errors can occur while solving the SLAM problem, of which the most common ones are related to false landmark associations (which occurs when two different landmarks are matched together between two subsequent measurements) and observing non-static landmarks. The latter suggests that a more precise estimation can be achieved when moving landmarks can be detected and omitted from the SLAM calculations.

This report describes a SLAM implementation in Python that uses an Extended Kalman Filter (EKF) to deal with the non-linearities in the measurement model and a Bayes classifier to distinguish between static and moving landmarks.

More advanced algorithms are also capable of estimating and tracking multiple possible locations to deal with false data associations when non-distinctive landmarks are observed, or to detect and track moving objects using a multi-modal approach to handle more complicated manoeuvres.

Bayesian Formulation of the SLAM Problem

Based on the background information provided by the *Robot Mapping* [1] course from the University of Freiburg, the basic SLAM problem can be defined as:

Given

- the robot's controls $u_{1:T} = \{u_1, u_2, u_3, \dots, u_T\}$
- the observations $z_{1:T} = \{z_1, z_2, z_3, \dots, z_T\}$

estimate

- a map of the environment $m = \{x_1, y_1, x_2, y_2, \dots, x_N, y_N\}$
- the path the robot traversed $x_{0:T} = \{x_0, x_1, x_2, \dots, x_T\}$, in which each x represents

This can be re-formulated as calculating the probability density function $p(x_t, m|z_{1:t}, u_{1:t})$. According to the Bayesian formulation, this probability density is calculated using the two-step recursive Bayes filter, consisting of a *prediction* and a *correction*.

Prediction step

The motion model corresponding to the prediction step contains an initial rotation θ_1, followed by a translation d and a final rotation θ_2, which allows for 3 DoF in a 2D plane. This results in the following (non-linear) motion predict equations:

$$\begin{matrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{matrix} = \begin{matrix} x_t \\ y_t \\ \theta_t \end{matrix} + \begin{matrix} d * \cos(\theta_t + r_1) \\ d * \sin(\theta_t + r_1) \\ \theta_1 + \theta_2 \end{matrix}$$

The motion errors are modelled as additive zero-mean Gaussian noise, added to each of the input variables θ_1, θ_2 and d. The same way, it would also be possible to plug in a more advanced motion model which takes into account, for example, the velocity, acceleration and steering angle of the robot, but this makes little difference to the filtering problem at hand.

Update step

Many real-world sensors such as Lidars and stereovision setups output range-bearing measurements. The measurement model corresponding to the update step therefore contains the range r and bearing φ for each of the observed landmarks. The measurements of landmark j are then mapped to x/y-coordinates in the following manner:

$$\begin{matrix} x_j \\ y_j \end{matrix} = \begin{matrix} x_t \\ y_t \end{matrix} + \begin{matrix} r * \cos(\varphi_j + \theta_t) \\ r * \sin(\varphi_j + \theta_t) \end{matrix}$$

Again, the measurement errors are modelled as additive zero-mean Gaussian noise, added to each of the measurement variables r and φ.

The non-linearities in both the motion and measurement equations require an Extended Kalman Filter to be implemented to maintain Gaussian probability densities. The EKF uses the linearized motion and measurement equations around the point of operation given by the Jacobians G (motion model) and H (measurement model) to calculate the new covariance matrices.

$$G = \begin{matrix} 0 & 0 & -d * \sin\theta_t + r_1 \\ 0 & 0 & d * \cos\theta_t + r_1 \\ 0 & 0 & 0 \end{matrix}$$

$$H = \frac{1}{q}\begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & \sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{pmatrix}$$

with

- $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix}$ defining the displacement between the estimated robot coordinates after the prediction step and the landmark coordinates, and
- $q = \delta^T \delta$ the squared distance between the estimated robot coordinates after the prediction step and the landmark coordinates.

The expected measurement for each landmark j is then calculated as

$$\hat{z}_j = \begin{pmatrix} \hat{r}_j \\ \hat{\varphi}_j \end{pmatrix} = \begin{pmatrix} \sqrt{q} \\ \tan^{-1}(\delta_y/\delta_x) - \theta_t \end{pmatrix}$$

where the bearing $\hat{\varphi}_j$ is normalized between $-\pi$ and π.

SLAM Implementation

Let us define the state vector of length 3+2*N as

$$\mu_t = \left(x_t, y_t, \theta_t, m_{1,x}, m_{1,y}, \dots, m_{N,x}, m_{N,y}\right)^T$$

and the covariance matrix Σ_t as a 3+2*N by 3+2*N matrix with the same structure as the state vector, such that the first 3*3 sub-matrix represents the covariances of the robot state.

Furthermore, let us denote the motion equation as the function g, the measurement equation as the function h, the motion and measurement noise covariances as R and Q, respectively, and the Kalman gain as the matrix K. Also, let us assume a fixed and known number of landmarks N and known and error-free landmark associations. The pseudo-code for the EKF SLAM algorithm is then as follows:

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$

3: $\bar{\Sigma}_t = G_t\, \Sigma_{t-1}\, G_t^T + R_t$

4: $K_t = \bar{\Sigma}_t\, H_t^T (H_t\, \bar{\Sigma}_t\, H_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6: $\Sigma_t = (I - K_t\, H_t)\, \bar{\Sigma}_t$

7: $return\ \mu_t, \Sigma_t$

Here, lines 2-3 denote the prediction step whereas lines 4-6 denote the update step.

The state vector and covariance matrix are initialized (under the assumption that no landmarks have been observed yet and the initial robot state is known with certainty) as:

$$\mu_0 = (0, 0, 0.5\pi, 0, \dots, 0)^T$$

$$\Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \infty & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \infty \end{pmatrix}$$

The file “slam.py” is the main file of the Python EKF SLAM implementation in which the robot object is created, and the trajectory is generated and estimated. The predict and update steps are implemented as separate functions in the “ekf.py” file. The robot object is defined in “robot.py”. Functions to generate plots are defined in “plotmap.py”. All code can be found on https://github.com/Attila94/EKF-SLAM.

Results and Performance of the Basic EKF SLAM Algorithm

The algorithm is tested with different motion and measurement noise covariances, generated trajectories and a varying number of landmarks. The general (sometimes obvious) findings are as follows:

- Robot state covariance increases during the prediction step and decreases during the update step.
- The algorithm benefits more from the correction step when the motion covariance is large, as otherwise the prediction step already yields fairly precise estimates.

- A larger number of landmarks result in a more precise position estimate.
- Loop closing, e.g. observing landmarks that have already been observed at an earlier stage greatly decreases the covariance of both the robot state and landmark position. This effect is clearly visible in figure 1, where loop closure occurs twice: first around step 9 and then around step 20. Both times, the squared estimation error decreased.
- The algorithm is unable to recover the true trajectory when the motion noise covariance approximately 2, 0.2 and 0.2 for each of the inputs, respectively.

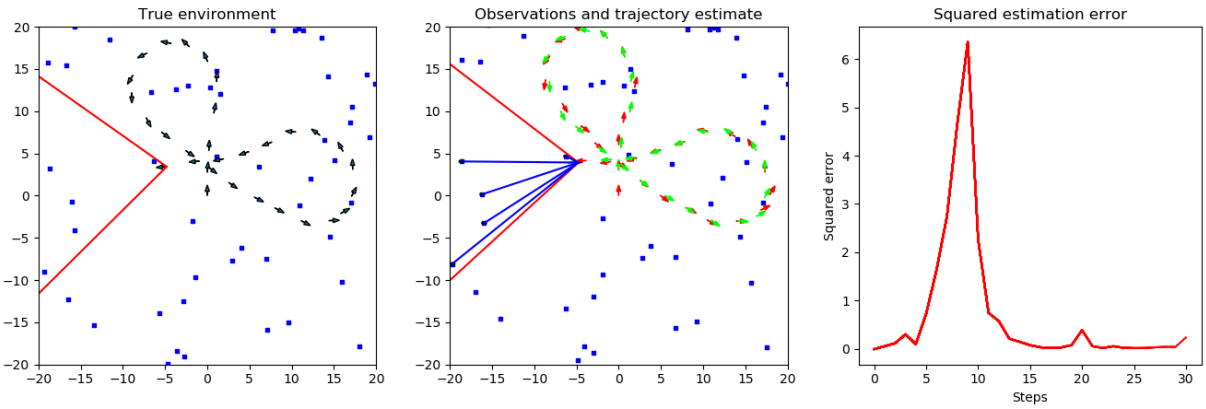

Figure 1: True robot trajectory (left), estimated trajectory (middle) and squared positional error (right) using the EKF SLAM algorithm. Estimates after the prediction step are shown as red arrows, estimates after the update step are shown as green arrows. Blue dots represent (static) landmarks and the red lines indicate the robot's field of view. The following settings were used: $R_t = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.05 \end{bmatrix}$, $Q_t = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$, $x_0 = (0, 0, 0.5\pi)$, $fov = 80°$, $n = 50$.

Moving Object Detection

As the results in the previous section show, the implemented SLAM algorithm is capable of precisely estimating the robot trajectory, even when both motions and observations are heavily affected by noise.

Let us now drop the static landmark assumption and apply the same SLAM algorithm with several dynamic, constant velocity landmarks (figure 2). 10 dynamic landmarks have been initialized at random, the trajectories of which are shown in green in the left subplot. Static landmarks are again shown in blue. All other settings are equal to the previous example (figure 1). As the middle subplot shows, the basic EKF SLAM algorithm was unable to recover the true trajectory from the landmark measurements. When comparing the true and estimated trajectories it becomes clear that the decrease in the squared estimation error towards the end of the trajectory is a mere coincidence.

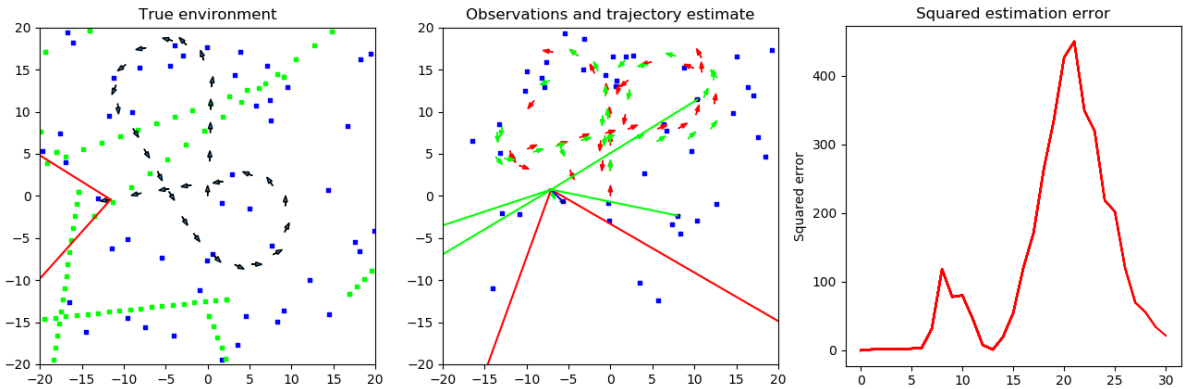

Figure 2: Errors introduced by dynamic landmarks. Dynamic landmark trajectories are marked with green in the left image. The basic EKF SLAM algorithm fails to recover the true trajectory.

The previous example clearly illustrated the need for implementing a way to deal with dynamic landmarks in the SLAM calculations. For this purpose, a class label has been attached to the landmarks which allows them to be classified as either static or dynamic. A class probability of $P \geq 0.5$ denotes a static landmark, whereas $P < 0.5$ indicates that the given landmark is dynamic. It is assumed that there are no state transitions, e.g. a static landmark remains static and vice versa, and thus this problem falls in the category of estimation of dynamic continuous and static discrete variables.

The following method has not been implemented in the Python code yet.

Initially, all landmarks that have not yet been observed are assumed to be static as no velocity information can be obtained from a single measurement and are assigned a class probability of $P = 0.5$. Each subsequent observation of a known landmark is then processed in the following way:

1. Given the previously observed positions of the landmark, the x/y-velocities are calculated by a ML estimator. A constant velocity model is assumed, as well as that the x and y velocities are uncorrelated and therefore can be estimated separately.
    - The landmark state vector is defined as $x = \begin{bmatrix} p \\ v \end{bmatrix}$, where p denotes the initial landmark position and v denotes the landmark velocity.
    - The measurement model is defined as $H = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 0 & t_1 & & t_N \end{bmatrix}^T$, where t denotes the time since the first measurement and N is the total number of measurements of the landmark.
    - The measurement vector contains the landmark locations estimated by the EKF SLAM algorithm, such that $z = [p_0 \quad p_1 \cdots p_N]$
    - The measurement covariances are the landmark location covariances estimated by the EKF SLAM algorithm.
    - The ML estimate is then given by:

$$F = H^T \Sigma_n^{-1} H$$
$$x_{ML} = F^{-1} H \Sigma_n^{-1} z$$
$$\sigma_{v_{ML}}^2 = F^{-1}(2,2)$$

2. Given the previously observed positions of the landmark and the constant velocity model, a new position is estimated for the landmark for both the static and dynamic cases. For both position estimates, an expected measurement is calculated in the same way as in the basic EKF SLAM algorithm.
3. A Bayes classifier calculates the class probabilities given the expected measurements for both the static and dynamic models and real measurement.
4. If the landmark is classified as static, it is then used to update the robot position using the basic EKF SLAM algorithm.

As the algorithm has not been implemented yet, unfortunately no information can be given about its performance.

Bibliography

[1] “Robot Mapping - WS 2013/14,” 21 10 2018. [Online]. Available: http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/.