

Exercise 2: Simultaneous Localization and Mapping using the Extended Kalman Filter

August 11, 2010

1 Introduction

One important aspect of robotic navigation is the ability to fuse multiple sources of data. In the case of a mobile robot, we might have a number of sensors telling us our current position. Each of these sensors is subject to noise and errors of various kinds. Some sources of position data, such as triangulated range and bearing observations to known targets, are often quite noisy and subject to short term errors. Other position information, such as odometry, is subject to an accumulation of errors that result from inaccuracies in our model and noise on the control lines. The fusion of these two sources of data can, however, give us much better results since one is good over the long term while the other is fairly reliable for predicting our position over a short distance.

In this exercise, you will implement the Extended Kalman Filter (EKF) for Simultaneous Localization and Mapping in Matlab. Large parts of the code are provided, but you need to implement parts of the transition model (also termed prediction step) and the sensor model.

The position of the robot and the landmarks in the map are contained in the state vector:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_{v_k} \\ m_1 \\ \dots \\ m_n \end{bmatrix} \quad (1)$$

where \mathbf{x}_{v_k} is the current vehicle state we are estimating in the prediction stage. m_i is the position of landmark i in the map. For clarity \mathbf{x}_{v_k} can be decomposed as:

$$\mathbf{x}_{v_k} = \{x_{v_k}, y_{v_k}, \psi_{v_k}\}^T \quad (2)$$

where x_{v_k} and y_{v_k} are the x- and y-position of the robot and ψ_{v_k} is the orientation of the robot.

1.1 Prediction Stage

In the prediction stage we will use a simple bicycle vehicle model which appears in Figure 1. Vehicle control signals consist of a velocity and a steering angle that can be retrieved from the robot. The vehicle model and update equations are included here for your implementation in the code.

The state vector in this step looks like:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_{v_k} \\ m_1 \\ \dots \\ m_n \end{bmatrix} \quad (3)$$

In the prediction step we want to update the position of the robot in the state vector and the uncertainties in the covariance matrix. The position of the robot is updated using the wheel direction and the velocity of the robot:

$$\mathbf{x}_{v_k} = \begin{bmatrix} x_{v_k} \\ y_{v_k} \\ \psi_{v_k} \end{bmatrix} \mathbf{f}_v(\mathbf{x}_{v_{k-1}}, \mathbf{u}_k) = \begin{bmatrix} x_{v_{k-1}} + V_k \Delta T \cos(\psi_{v_{k-1}} + \gamma_k) \\ y_{v_{k-1}} + V_k \Delta T \sin(\psi_{v_{k-1}} + \gamma_k) \\ \psi_{v_{k-1}} + \frac{V_k \Delta T}{B} \sin(\gamma_k) \end{bmatrix} \quad (4)$$

where $x_{v_{k-1}}$, $y_{v_{k-1}}$ are the xy position of the robot at time $k - 1$ and where V_k is the velocity control input and γ_k is the steering angle input and B is the wheel base of the vehicle and $\psi_{v_{k-1}}$ is orientation from the current head of the SLAM state vector.

In the matlab code you will work with, the state vector is contained in the vector `XX`. The robot's xy and angle position are in `XX(1:3)`. The other elements in the array contain the landmark positions.

Besides updating the position of the robot in the state vector. We also want to update the uncertainty of the estimated position of the robot and the estimated positions of the landmarks. These uncertainties are stored in a covariance matrix. In matlab code, the covariance matrix `PX` is updated as follows:

```
PX(1:3,1:3)= Gv*PX(1:3,1:3)*Gv' + Gu*Q*Gu';
if size(PX,1)>3
    PX(1:3,4:end)= Gv*PX(1:3,4:end);
    PX(4:end,1:3)= PX(1:3,4:end)';
end
```

In the first line, the uncertainty of the robot's xy-position and orientation is updated. In the next four lines, the uncertainty of the relation between the robot and all the landmarks in the map are updated.

To calculate these updates, we need the Jacobians G_u and G_v for our robot model. The Jacobians are as follows:

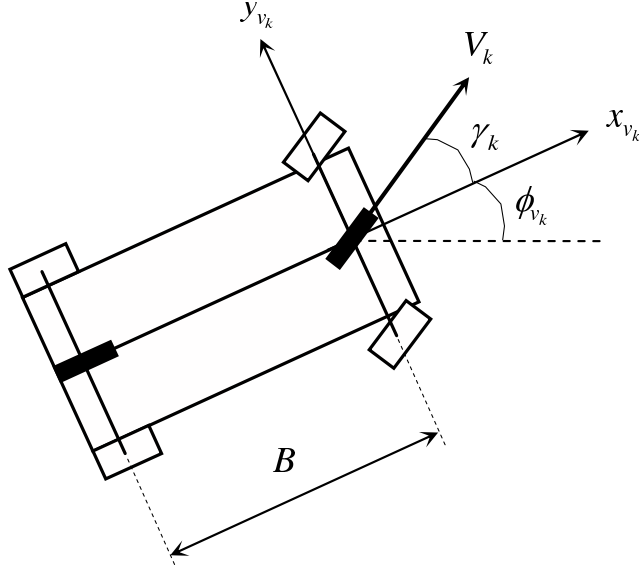


Figure 1: Vehicle Motion Model

$$\mathbf{G}_v = \begin{bmatrix} 1 & 0 & -V_k \Delta T \sin(\psi_{v_{k-1}} + \gamma_k) \\ 0 & 1 & V_k \Delta T \cos(\psi_{v_{k-1}} + \gamma_k) \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

$$\mathbf{G}_u = \begin{bmatrix} \Delta T \cos(\psi_{v_{k-1}} + \gamma_k) & -V_k \Delta T \sin(\psi_{v_{k-1}} + \gamma_k) \\ \Delta T \sin(\psi_{v_{k-1}} + \gamma_k) & V_k \Delta T \cos(\psi_{v_{k-1}} + \gamma_k) \\ \frac{\Delta T \sin(\gamma_k)}{B} & \frac{V_k \Delta T \cos(\gamma_k)}{B} \end{bmatrix} \quad (6)$$

where V_k is the velocity control input and γ_k is the steering angle input and B is the wheel base of the vehicle and $\psi_{v_{k-1}}$ is orientation from the current head of the SLAM state vector.

You will need to look through the code. Open **predict.m** and find how these variables match up with update equations and fill in the empty code. Note **function angle = pi_to_pi(angle)** should be used when adding headings and steering angles to perform correct modulus wrap around.

1.2 Observation: Sensor model

The observation stage consists of a series of observations that arrive from sensors. In this case, the vehicle is equipped with a laser range finder. Observations of beacons

in the environment can be used to provide a position estimate when the position of the beacons is known. This model is depicted in Figure 2.

To compute the vehicles position from a beacon observation, we need to calculate the distance and angle towards the beacon:

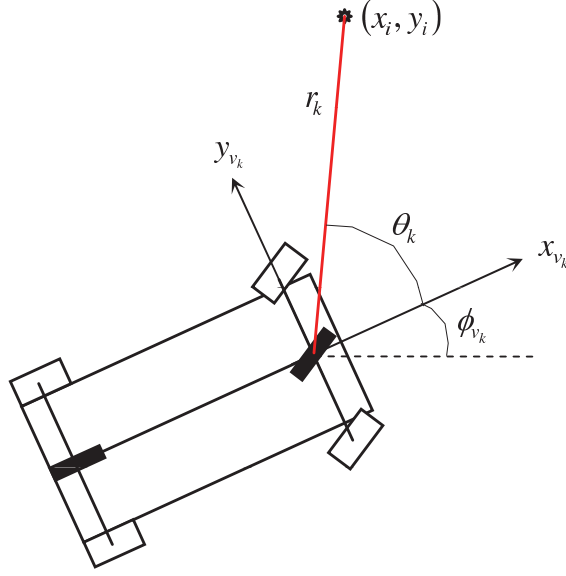


Figure 2: Observation Model

$$\mathbf{z}_{i_k} = \mathbf{h}_i(\mathbf{x}_k) = \begin{bmatrix} \sqrt{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2} \\ \arctan \frac{y_i - y_{v_k}}{x_i - x_{v_k}} - \phi_{v_k} \end{bmatrix} \quad (7)$$

where x_i and y_i are the x- and y-position of the beacon.

The Jacobian of the observation model is :

$$\frac{\partial h_r}{\partial x} = \begin{pmatrix} -\frac{x_i - x_{v_k}}{\sqrt{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2}} & -\frac{y_i - y_{v_k}}{\sqrt{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2}} & 0 \\ \frac{y_i - y_{v_k}}{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2} & -\frac{x_i - x_{v_k}}{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2} & -1 \end{pmatrix} \quad (8)$$

$$\frac{\partial h_l}{\partial x} = \begin{pmatrix} \frac{x_i - x_{v_k}}{\sqrt{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2}} & \frac{y_i - y_{v_k}}{\sqrt{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2}} \\ -\frac{y_i - y_{v_k}}{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2} & \frac{x_i - x_{v_k}}{(x_i - x_{v_k})^2 + (y_i - y_{v_k})^2} \end{pmatrix} \quad (9)$$

where $\frac{\partial h_r}{\partial x}$ is the jacobian for the robot and $\frac{\partial h_l}{\partial x}$ is the jacobian for the landmark.

You will need to look through the code. Open **observe_model.m** and find how these variables match up with observation equations and fill in the empty code. Hint: H is a matrix of size $2 \times$ (length of state vector) and $\frac{\partial h_r}{\partial x}$ is the top of the H matrix and $\frac{\partial h_l}{\partial x}$ is stored in $H(:, \text{fpos}:\text{fpos}+1)$ positions calculated in the code. And the index **fpos** in the state vector \mathbf{x} is the position of the landmark (x_i, y_i)

2 Running

Once the functions have been filled in you can run the EKF slam simulator as follows:

```
load example_webmap;
ekfslam_sim(lm, wp);
```

After you have successfully run the example run **frontend.m** and create a map where the landmarks are spaced quite far apart along the trajectory and watch how the true path and estimated path diverge. Note the effect of landmark placement on the successful localization of the vehicle.