

# EKF-SLAM Derivation

Kevin Robb

## Contents

<b>1</b>	<b>Kalman Filter Intro</b>	<b>2</b>
<b>2</b>	<b>Extended Kalman Filter (EKF) Motivation</b>	<b>2</b>
<b>3</b>	<b>EKF for Localization Only</b>	<b>2</b>
3.a	Predict Stage . . . . .	3
3.b	Update Stage . . . . .	4
<b>4</b>	<b>EKF for (Landmark) Mapping Only</b>	<b>5</b>
4.a	Predict Stage . . . . .	6
4.b	Update Stage . . . . .	6
4.b.i	Update - Existing Landmark . . . . .	6
4.b.ii	Update - New Landmark . . . . .	7
<b>5</b>	<b>EKF for Localization and Mapping (SLAM)</b>	<b>8</b>
5.a	Predict Stage . . . . .	9
5.b	Update Stage . . . . .	10
5.b.i	Update - Existing Landmark . . . . .	10
5.b.ii	Update - New Landmark . . . . .	11
5.b.iii	Update - End . . . . .	12
<b>6</b>	<b>Demo</b>	<b>12</b>

# 1 Kalman Filter Intro

A Kalman Filter is an iterative process of estimating some state in the most effective way possible. Generally, the inputs include commands sent to the vehicle ( $\mathbf{u}$ ), and a measurement obtained after the commanded motion is performed ( $\mathbf{z}$ ). The output will be the current estimate of the state, which takes the form of some probability distribution. For most Bayesian filters, we assume the state can be represented by a Gaussian distribution, so we track the state mean vector  $\mathbf{x}$  and covariance matrix  $\mathbf{P}$ . The elements of  $\mathbf{x}$  are all the variables we want to estimate, and  $\mathbf{P}$  gives an idea of the uncertainty of the current state estimates; a more spread-out distribution shows less confidence than a highly-peaked one.

This type of filter has two main phases that it alternates between, the “predict” and “update” stages. The prediction stage involves propagating the state forward using the motion model and the commands sent to the vehicle; this stage always involves injecting noise into the system. The update stage involves taking some measurement of the world, and using it to change our state estimate.

A standard Kalman filter uses Gaussian noise profiles, a Gaussian state distribution, and a linear model for the predictions and measurements.

## 2 Extended Kalman Filter (EKF) Motivation

As evident from my simulator setup, the vehicle is described by not only a position, but an orientation (yaw). This makes our system nonlinear, since the change in position on each timestep relies on the sine and cosine of the current yaw. Fortunately, we can still represent the state as a Gaussian distribution, since each state variable’s distribution is symmetric and unimodal about some mean. One method to estimate the state in such a system is the Extended Kalman Filter.

The Extended Kalman Filter (EKF) keeps the same general flow as the standard KF, but requires the calculation of several Jacobian matrices on each iteration; these are used to linearize the system about the current estimated mean, and carry out the state propagation *as if* it were a linear system. This works great in many situations, but will fail for highly-nonlinear systems whose behavior may change drastically with a slight change in input.

## 3 EKF for Localization Only

In the localization case, we assume the true map is known. Our goal is to use the commands and measurements to create a “live” estimate of the vehicle’s current pose. As such, our state will contain exactly that. The vehicle is initially at the origin.

$$\mathbf{x}_0 = \begin{bmatrix} x_v \\ y_v \\ \theta_v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We initialize the covariance as a diagonal matrix (since initially there is no correlation between state variables) whose entries reflect our perceived variance of each state variable. This need not be extremely precise, as it

will be changed on each EKF iteration.

$$\mathbf{P}_0 = \begin{bmatrix} 0.01^2 & 0 & 0 \\ 0 & 0.01^2 & 0 \\ 0 & 0 & 0.005^2 \end{bmatrix}$$

We also need to define some noise terms that represent some multidimensional Gaussian distributions that affect the EKF. The process noise is denoted by

$$\mathbf{v} \sim \mathcal{N}\left(\begin{bmatrix} v_d \\ v_\theta \end{bmatrix}, \mathbf{V}\right),$$

and the measurement noise by

$$\mathbf{w} \sim \mathcal{N}\left(\begin{bmatrix} w_r \\ w_\beta \end{bmatrix}, \mathbf{W}\right).$$

The former represents error in our prediction, such as the robot not actually moving the exact amount commanded, while the latter represents errors in measurement, such as sensor noise. We assume the nominal case of no systematic bias, so the mean terms are 0. The variance matrices  $\mathbf{V}$  and  $\mathbf{W}$  will be used when updating the state, and are set manually by the designer. In this demo, I use the following values.

$$\mathbf{V} = \begin{bmatrix} 0.02^2 & 0 \\ 0 & \left(\frac{\pi}{360}\right)^2 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 0.1^2 & 0 \\ 0 & \left(\frac{\pi}{180}\right)^2 \end{bmatrix}$$

With all requisites now defined, we can enter the EKF loop.

### 3.a Predict Stage

In the predict stage, we receive the command for the current timestep,

$$\mathbf{u}_t = \begin{bmatrix} u_d \\ u_\theta \end{bmatrix}$$

and can use this to predict the state mean at the next timestep,

$$\mathbf{x}_{pred} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{v}) = \begin{bmatrix} x_v + (u_d + v_d) \cos(\theta_v) \\ y_v + (u_d + v_d) \sin(\theta_v) \\ \theta_v + u_\theta + v_\theta \end{bmatrix}.$$

To propagate the state covariance, we need to compute some Jacobian matrices to handle the nonlinearity caused by including  $\theta$  in the state.

$$\mathbf{F}_x = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} 1 & 0 & -u_d \sin(\theta_v) \\ 0 & 1 & u_d \cos(\theta_v) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{F}_v = \frac{\partial f}{\partial \mathbf{v}} = \begin{bmatrix} \cos(\theta_v) & 0 \\ \sin(\theta_v) & 0 \\ 0 & 1 \end{bmatrix}$$

The propagated state covariance is then calculated as

$$\mathbf{P}_{pred} = \mathbf{F}_x \cdot \mathbf{P}_t \cdot \mathbf{F}_x^T + \mathbf{F}_v \cdot \mathbf{V} \cdot \mathbf{F}_v^T$$

### 3.b Update Stage

In this section,  $x_v$ ,  $y_v$ , and  $\theta_v$  refer to the elements of the predicted state mean,  $\mathbf{x}_{pred}$ .

Recall that our measurements will come in the form of an ID, range, and bearing for each landmark detected this timestep. If nothing is detected, the update stage is skipped, and we simply use our predictions to propagate the state for this timestep:

$$\mathbf{x}_{t+1} = \mathbf{x}_{pred}$$

$$\mathbf{P}_{t+1} = \mathbf{P}_{pred}$$

If there is at least one landmark detection, we perform the update step for each in series. For one such measurement, then, we will have

$$\mathbf{z}_{t+1} = \begin{bmatrix} z_{id} \\ z_r \\ z_\beta \end{bmatrix}.$$

Since we're in the localization-only case, we assume that we know the true map. Let

$$\mathbf{p}_l = \begin{bmatrix} x_l \\ y_l \end{bmatrix}$$

be the known true position of the landmark denoted by  $z_{id}$  in the world frame.

We can compute the distance from our predicted vehicle position and the known landmark position,

$$d = \sqrt{(x_l - x_v)^2 + (y_l - y_v)^2},$$

and use this to estimate the measurement we would have gotten if our state estimate is completely correct,

$$\mathbf{z}_{est} = h(\mathbf{x}_{pred}, \mathbf{w}) = \begin{bmatrix} r_{est} \\ \beta_{est} \end{bmatrix} = \begin{bmatrix} d \\ \arctan\left(\frac{y_l - y_v}{x_l - x_v}\right) - \theta_v \end{bmatrix} + \begin{bmatrix} w_r \\ w_\beta \end{bmatrix},$$

where  $\beta_{est}$  should remain in the range  $(-\pi, \pi)$ . This can be achieved in both C++ and Python3 using the

remainder function with divisor  $\tau = 2\pi$ :

```
beta_est = remainder(atan2(y_l - y_v, x_l - x_v) - theta_v, tau)
```

Using  $z_{est}$ , we can compute the “innovation”, which gives a sense of how the actual measurement differs from our expectation given our predicted state.

$$\nu = \begin{bmatrix} z_r \\ z_\beta \end{bmatrix} - \begin{bmatrix} r_{est} \\ \beta_{est} \end{bmatrix}$$

To proceed, we must compute some more Jacobian matrices to perform linearization.

$$\mathbf{H}_x = \frac{\partial h}{\partial \mathbf{x}} = \begin{bmatrix} -\frac{x_l - x_v}{d} & -\frac{y_l - y_v}{d} & 0 \\ \frac{y_l - y_v}{d^2} & -\frac{x_l - x_v}{d^2} & -1 \end{bmatrix}$$

$$\mathbf{H}_w = \frac{\partial h}{\partial \mathbf{w}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

We can now compute the Kalman Gain, which captures the ideal relative weights of our prediction vs the new measurement.

$$\mathbf{S} = \mathbf{H}_x \cdot \mathbf{P}_{pred} \cdot \mathbf{H}_x^T + \mathbf{H}_w \cdot \mathbf{W} \cdot \mathbf{H}_w^T$$

$$\mathbf{K} = \mathbf{P}_{pred} \cdot \mathbf{H}_x^T \cdot \mathbf{S}^{-1}$$

We can then update our prediction of the state mean and covariance:

$$\mathbf{x}_{pred} = \mathbf{x}_{pred} + \mathbf{K} \cdot \nu$$

$$\mathbf{P}_{pred} = \mathbf{P}_{pred} - \mathbf{K} \cdot \mathbf{H}_x \cdot \mathbf{P}_{pred}$$

We should also ensure that the vehicle’s heading estimate  $\theta_v$  stays within  $(-\pi, \pi)$  using the same method described above for  $\beta_{est}$ .

After running this update process for each landmark detected this timestep, we update the state estimate.

$$\mathbf{x}_{t+1} = \mathbf{x}_{pred}$$

$$\mathbf{P}_{t+1} = \mathbf{P}_{pred}$$

At this point, we move to the next timestep and the loop goes back to the predict stage.

## 4 EKF for (Landmark) Mapping Only

In the mapping case, we assume the vehicle’s true position is known for all timesteps. This is achieved in our architecture by assuming the odometry commands are exactly correct and correspond to the robot’s true movement during that timestep. Our goal is to use this known position along with any measurements to produce a list of all landmarks and their positions, i.e., a map. Note that a landmark which is never detected by the robot will not appear in the map; this is fine, as in a real application we will either design our trajectory to intentionally observe all parts of the environment, or only need the map for local navigation and thus do not care about parts of the map that aren’t near the robot.

The state in the mapping case is a vector of landmark positions,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ y_1 \\ \cdot \\ \cdot \\ x_M \\ y_M \end{bmatrix}$$

We also keep a list of landmark IDs,  $L = [id_1, \dots, id_M]$ . The position of an ID in  $L$  corresponds to its position in the state. When we begin, there are no landmarks in the state.

$$\mathbf{x}_0 = [ \quad ]$$

$$\mathbf{P}_0 = [ \quad ]$$

With each detection, we will either insert that landmark into the state, or update its entry if it already appears.

#### 4.a Predict Stage

For the mapping-only case, there is no process noise. We know the robot's true position at all times, and we know that all landmarks are static. Thus, the prediction stage contains only a single equation which keeps track of the vehicle position given the known odometry for that timestep.

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{v}) = \begin{bmatrix} x_v + u_d \cos(\theta_v) \\ y_v + u_d \sin(\theta_v) \\ \theta_v + u_\theta \end{bmatrix}$$

The state “prediction” simply initializes  $\mathbf{x}_{t+1} = \mathbf{x}_t$ , since we predict landmarks will remain where they are.

#### 4.b Update Stage

If no landmarks are detected, we skip straight back to the prediction stage for the next timestep. If one or more landmarks are detected, we first need to check if each landmark has been seen before, or needs to be added to the state. A detection for one landmark takes the same form as previously discussed,

$$\mathbf{z}_{t+1} = \begin{bmatrix} z_{id} \\ z_r \\ z_\beta \end{bmatrix}.$$

##### 4.b.i Update - Existing Landmark

If  $z_{id}$  appears in  $L$ , we have seen this landmark before and added it to the state. Let the index of  $z_{id}$  in  $L$  be denoted  $i$ . We know then that our current estimate of this landmark's position is at the  $2i$  and  $2i + 1$  entries in the state.

$$\mathbf{p}_l = \begin{bmatrix} \mathbf{x}_t[2i] \\ \mathbf{x}_t[2i + 1] \end{bmatrix} = \begin{bmatrix} x_l \\ y_l \end{bmatrix}$$

Now we can perform the same steps as in the localization case to update our belief about this landmark. First we compute the distance from the known robot position to where we currently believe that landmark to be,

$$d = \sqrt{(x_l - x_v)^2 + (y_l - y_v)^2},$$

and use this to estimate the measurement we would have gotten if our state estimate is correct,

$$\mathbf{z}_{est} = h(\mathbf{x}_t, \mathbf{w}) = \begin{bmatrix} r_{est} \\ \beta_{est} \end{bmatrix} = \begin{bmatrix} d \\ \arctan\left(\frac{y_l - y_v}{x_l - x_v}\right) - \theta_v \end{bmatrix} + \begin{bmatrix} w_r \\ w_\beta \end{bmatrix},$$

where  $\beta_{est}$  should remain in the range  $(-\pi, \pi)$  as discussed in the previous section. This allows us to compute the innovation,

$$\boldsymbol{\nu} = \begin{bmatrix} z_r \\ z_\beta \end{bmatrix} - \begin{bmatrix} r_{est} \\ \beta_{est} \end{bmatrix}.$$

Now we compute some Jacobian matrices:

$$\mathbf{H}_p = \frac{\partial h}{\partial \mathbf{x}} = \begin{bmatrix} \frac{x_l - x_v}{d} & \frac{y_l - y_v}{d} \\ -\frac{y_l - y_v}{d^2} & \frac{x_l - x_v}{d^2} \end{bmatrix}$$

$$\mathbf{H}_w = \frac{\partial h}{\partial \mathbf{w}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Since this will be multiplied with the state covariance, we need to make sure its dimensions match while still only affecting the current landmark in question. We do this by padding it with zeros. The length of the state is  $2M$ , where  $M$  is the number of landmarks stored. Recall that  $i$  is the index of this landmark in  $L$ .

$$\mathbf{H}_x = \begin{bmatrix} \mathbf{0}_{2 \times (2i-2)} & \mathbf{H}_p & \mathbf{0}_{2 \times (2M-2i)} \end{bmatrix}$$

We can then use this to compute the Kalman Gain and update the state.

$$\mathbf{S} = \mathbf{H}_x \cdot \mathbf{P}_t \cdot \mathbf{H}_x^T + \mathbf{H}_w \cdot \mathbf{W} \cdot \mathbf{H}_w^T$$

$$\mathbf{K} = \mathbf{P}_t \cdot \mathbf{H}_x^T \cdot \mathbf{S}^{-1}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_{t+1} + \mathbf{K} \cdot \boldsymbol{\nu}$$

$$\mathbf{P}_{t+1} = \mathbf{P}_{t+1} - \mathbf{K} \cdot \mathbf{H}_x \cdot \mathbf{P}_{t+1}$$

#### 4.b.ii Update - New Landmark

To add a new landmark, we first append  $z_{id}$  to the end of  $L$ .

$$L = \begin{bmatrix} L & z_{id} \end{bmatrix}$$

We then use our detection to compute the global position of the landmark, and append it to the end of the state mean.

$$g(\mathbf{x}, \mathbf{z}) = \begin{bmatrix} x_v + z_r \cos(\theta_v + z_\beta) \\ y_v + z_r \sin(\theta_v + z_\beta) \end{bmatrix}$$

$$\mathbf{x}_{t+1} = \begin{bmatrix} \mathbf{x}_t \\ g(\mathbf{x}, \mathbf{z}_{t+1}) \end{bmatrix}$$

To update the covariance, we need to compute an “insertion Jacobian”,  $\mathbf{Y}_z$ .

$$\mathbf{G}_z = \begin{bmatrix} \cos(\theta_v + z_\beta) & -z_r \sin(\theta_v + z_\beta) \\ \sin(\theta_v + z_\beta) & z_r \cos(\theta_v + z_\beta) \end{bmatrix}$$

$$\mathbf{Y}_z = \begin{bmatrix} \mathbf{I}_{2M \times 2M} & \mathbf{0}_{2M \times 2} \\ \mathbf{0}_{2 \times 2M} & \mathbf{G}_z \end{bmatrix}$$

We can now update our covariance.

$$\mathbf{P}_{t+1} = \mathbf{Y}_z \cdot \begin{bmatrix} \mathbf{P}_{t+1} & \mathbf{0}_{2M \times 2} \\ \mathbf{0}_{2 \times 2M} & \mathbf{W} \end{bmatrix} \cdot \mathbf{Y}_z^T$$

## 5 EKF for Localization and Mapping (SLAM)

Now that we have derived the EKF for both localization and mapping separately, we can combine them without too much effort. Note that this implementation of EKF-SLAM will estimate the current vehicle pose as well as the full map, but will **not** retroactively estimate all previous vehicle poses. That wider problem is referred to as “Full SLAM” or “Pose-Graph SLAM”. We are doing an easier problem called “Online SLAM”, since we run it online to track the best possible estimate of the *current* state only.

We use the same process noise  $\mathbf{v}$  and measurement noise  $\mathbf{w}$  as before. Our initial state mean and covariance will be a combination of those for both localization and mapping, which is trivial as they were initially empty in the latter.

$$\mathbf{x}_0 = \begin{bmatrix} x_v \\ y_v \\ \theta_v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{P}_0 = \begin{bmatrix} 0.01^2 & 0 & 0 \\ 0 & 0.01^2 & 0 \\ 0 & 0 & 0.005^2 \end{bmatrix}$$



The state will grow to contain both the vehicle pose and all landmark estimates, and will have length  $3+2M$ .

$$\mathbf{x}_t = \begin{bmatrix} x_v \\ y_v \\ \theta_v \\ x_1 \\ y_1 \\ \vdots \\ x_M \\ y_M \end{bmatrix}$$

We will track landmark IDs in the same way as in EKF mapping, using a list  $L = [id_1, \dots, id_M]$ .

## 5.a Predict Stage

Similarly to EKF localization, we receive the command for the current timestep,

$$\mathbf{u}_t = \begin{bmatrix} u_d \\ u_\theta \end{bmatrix}$$

and use it to predict the state mean at the next timestep. However, we need to account for the fact that the state now potentially includes more than just the  $3 \times 1$  vehicle pose. Recall that our prediction for all landmark estimates is that they will remain constant.

$$\mathbf{x}_{pred} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{v}) = \begin{bmatrix} x_v + (u_d + v_d) \cos(\theta_v) \\ y_v + (u_d + v_d) \sin(\theta_v) \\ \theta_v + u_\theta + v_\theta \\ x_1 \\ y_1 \\ \vdots \\ x_M \\ y_M \end{bmatrix}.$$

To propagate the state covariance, we need to compute some Jacobians.

$$\begin{aligned}\mathbf{F}_{xv} &= \begin{bmatrix} 1 & 0 & -u_d \sin(\theta_v) \\ 0 & 1 & u_d \cos(\theta_v) \\ 0 & 0 & 1 \end{bmatrix} \\ \mathbf{F}_x = \frac{\partial f}{\partial \mathbf{x}} &= \begin{bmatrix} \mathbf{F}_{xv} & \mathbf{0}_{3 \times 2M} \\ \mathbf{0}_{2M \times 3} & \mathbf{I}_{2M \times 2M} \end{bmatrix} \\ \mathbf{F}_{vv} &= \begin{bmatrix} \cos(\theta_v) & 0 \\ \sin(\theta_v) & 0 \\ 0 & 1 \end{bmatrix} \\ \mathbf{F}_v = \frac{\partial f}{\partial \mathbf{v}} &= \begin{bmatrix} \mathbf{F}_{vv} \\ \mathbf{0}_{2M \times 2} \end{bmatrix}\end{aligned}$$

The sub-entries  $\mathbf{F}_{xv}$  and  $\mathbf{F}_{vv}$  exactly match the Jacobians in EKF localization; we have simply padded them to the correct dimensionality for EKF SLAM. The propagated state covariance is then calculated as

$$\mathbf{P}_{pred} = \mathbf{F}_x \cdot \mathbf{P}_t \cdot \mathbf{F}_x^T + \mathbf{F}_v \cdot \mathbf{V} \cdot \mathbf{F}_v^T$$

## 5.b Update Stage

In this section,  $x_v$ ,  $y_v$ , and  $\theta_v$  refer to the vehicle pose in the predicted state mean,  $\mathbf{x}_{pred}$ .

If no landmarks are detected, the update stage is skipped, and we simply use our predictions to propagate the state for this timestep:

$$\begin{aligned}\mathbf{x}_{t+1} &= \mathbf{x}_{pred} \\ \mathbf{P}_{t+1} &= \mathbf{P}_{pred}\end{aligned}$$

If one or more landmarks are detected, we first need to check if each landmark has been seen before, or needs to be added to the state. A detection for one landmark takes the same form as previously discussed,

$$\mathbf{z}_{t+1} = \begin{bmatrix} z_{id} \\ z_r \\ z_\beta \end{bmatrix}.$$

### 5.b.i Update - Existing Landmark

If  $z_{id}$  appears in  $L$ , we have seen this landmark before and added it to the state. Let the index of  $z_{id}$  in  $L$  be denoted  $i$ . We know then that our current estimate of this landmark's position is at the  $3 + 2i$  and  $3 + 2i + 1$  entries in the state.

$$\mathbf{p}_l = \begin{bmatrix} \mathbf{x}_t[3 + 2i] \\ \mathbf{x}_t[3 + 2i + 1] \end{bmatrix} = \begin{bmatrix} x_l \\ y_l \end{bmatrix}$$

Now we follow the same procedure as before to update our belief about this landmark. First we compute the distance from the predicted robot position to the estimated landmark position,

$$d = \sqrt{(x_l - x_v)^2 + (y_l - y_v)^2},$$

and use this to estimate the measurement we would have gotten if our state estimate is correct,

$$\mathbf{z}_{est} = h(\mathbf{x}_t, \mathbf{w}) = \begin{bmatrix} r_{est} \\ \beta_{est} \end{bmatrix} = \begin{bmatrix} d \\ \arctan\left(\frac{y_l - y_v}{x_l - x_v}\right) - \theta_v \end{bmatrix} + \begin{bmatrix} w_r \\ w_\beta \end{bmatrix},$$

where  $\beta_{est}$  should remain in the range  $(-\pi, \pi)$  as discussed previously. This allows us to compute the innovation,

$$\boldsymbol{\nu} = \begin{bmatrix} z_r \\ z_\beta \end{bmatrix} - \begin{bmatrix} r_{est} \\ \beta_{est} \end{bmatrix}.$$

Now we compute some Jacobian matrices. Again the submatrices will look familiar, and we arrange their placement in larger matrices to fit all dimensionality correctly.

$$\begin{aligned} \mathbf{H}_{xv} &= \begin{bmatrix} -\frac{x_l - x_v}{d} & -\frac{y_l - y_v}{d} & 0 \\ \frac{y_l - y_v}{d^2} & -\frac{x_l - x_v}{d^2} & -1 \end{bmatrix} \\ \mathbf{H}_{xp} &= \begin{bmatrix} \frac{x_l - x_v}{d} & \frac{y_l - y_v}{d} \\ -\frac{y_l - y_v}{d^2} & \frac{x_l - x_v}{d^2} \end{bmatrix} \\ \mathbf{H}_x = \frac{\partial h}{\partial \mathbf{x}} &= \begin{bmatrix} \mathbf{H}_{xv} & \mathbf{0}_{2 \times (2i-2)} & \mathbf{H}_{xp} & \mathbf{0}_{2 \times (2M-2i)} \end{bmatrix} \\ \mathbf{H}_w = \frac{\partial h}{\partial \mathbf{w}} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

We can then use these to compute the Kalman Gain and update the state.

$$\begin{aligned} \mathbf{S} &= \mathbf{H}_x \cdot \mathbf{P}_t \cdot \mathbf{H}_x^T + \mathbf{H}_w \cdot \mathbf{W} \cdot \mathbf{H}_w^T \\ \mathbf{K} &= \mathbf{P}_t \cdot \mathbf{H}_x^T \cdot \mathbf{S}^{-1} \\ \mathbf{x}_{pred} &= \mathbf{x}_{pred} + \mathbf{K} \cdot \boldsymbol{\nu} \\ \mathbf{P}_{pred} &= \mathbf{P}_{pred} - \mathbf{K} \cdot \mathbf{H}_x \cdot \mathbf{P}_{pred} \end{aligned}$$

### 5.b.ii Update - New Landmark

To add a new landmark, we first append  $z_{id}$  to the end of  $L$ .

$$L = \begin{bmatrix} L & z_{id} \end{bmatrix}$$

We then use our detection to compute the position of the landmark in the world frame and append it to the end of the state mean.

$$\begin{aligned} g(\mathbf{x}, \mathbf{z}) &= \begin{bmatrix} x_v + z_r \cos(\theta_v + z_\beta) \\ y_v + z_r \sin(\theta_v + z_\beta) \end{bmatrix} \\ \mathbf{x}_{pred} &= \begin{bmatrix} \mathbf{x}_{pred} \\ g(\mathbf{x}, \mathbf{z}_{t+1}) \end{bmatrix} \end{aligned}$$

Now we compute the insertion Jacobian,  $\mathbf{Y}_z$ .

$$\mathbf{G}_z = \begin{bmatrix} \cos(\theta_v + z_\beta) & -z_r \sin(\theta_v + z_\beta) \\ \sin(\theta_v + z_\beta) & z_r \cos(\theta_v + z_\beta) \end{bmatrix}$$

$$\mathbf{G}_x = \begin{bmatrix} 1 & 0 & -z_r \sin(\theta_v + z_\beta) \\ 0 & 1 & z_r \cos(\theta_v + z_\beta) \end{bmatrix}$$

$$\mathbf{Y}_z = \begin{bmatrix} \mathbf{I}_{(3+2M) \times (3+2M)} & \mathbf{0}_{(3+2M) \times 2} \\ \mathbf{G}_x & \mathbf{0}_{2 \times (2M)} & \mathbf{G}_z \end{bmatrix}$$

We can then update the covariance.

$$\mathbf{P}_{pred} = \mathbf{Y}_z \cdot \begin{bmatrix} \mathbf{P}_{pred} & \mathbf{0}_{(3+2M) \times 2} \\ \mathbf{0}_{2 \times (3+2M)} & \mathbf{W} \end{bmatrix} \cdot \mathbf{Y}_z^T$$

### 5.b.iii Update - End

After running this update process for each landmark detected this timestep, we update the state estimate.

$$\mathbf{x}_{t+1} = \mathbf{x}_{pred}$$

$$\mathbf{P}_{t+1} = \mathbf{P}_{pred}$$

At this point, we move to the next timestep and the loop goes back to the predict stage.

## 6 Demo

In all plots, the empty circles are the true landmark positions, and the red circles are the EKF estimates. Orange ellipses, when shown, are the EKF covariances for each landmark estimate. The blue path is the true trajectory the vehicle follows for the total duration; if a blue path is not shown, the blue arrow shows the current true position. The green arrow shows the current EKF estimate for the vehicle pose, and the grey ellipse shows the current EKF covariance for the vehicle's position.

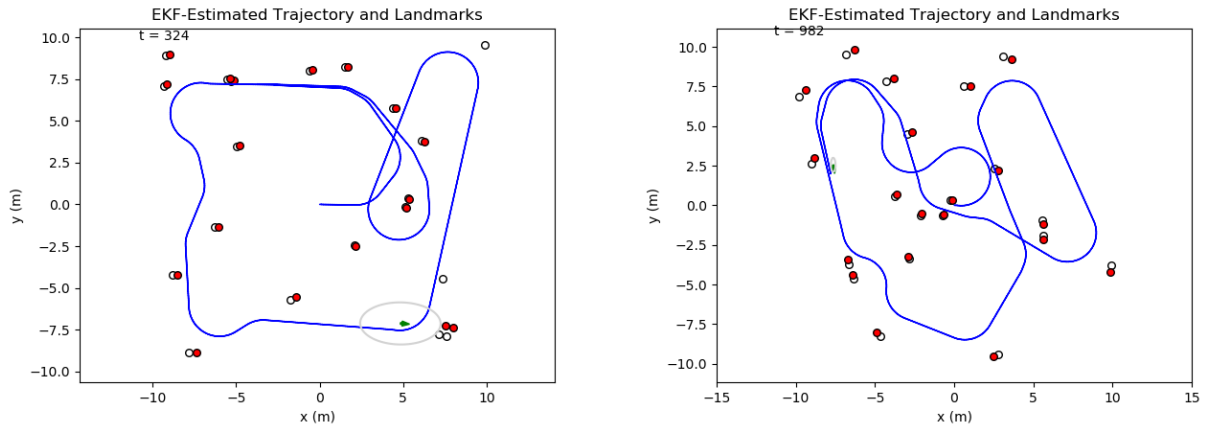


Figure 1: Performance with random maps.

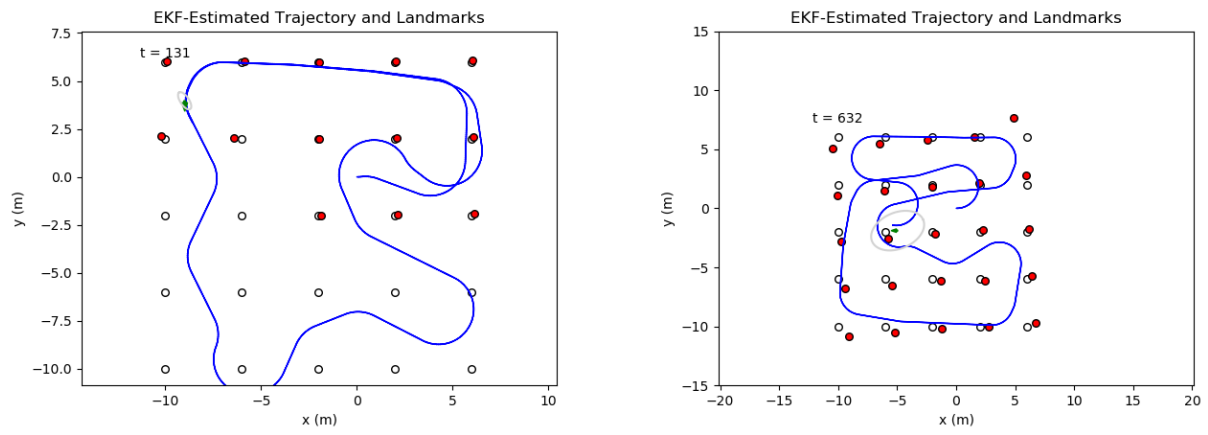


Figure 2: Performance with grid map.

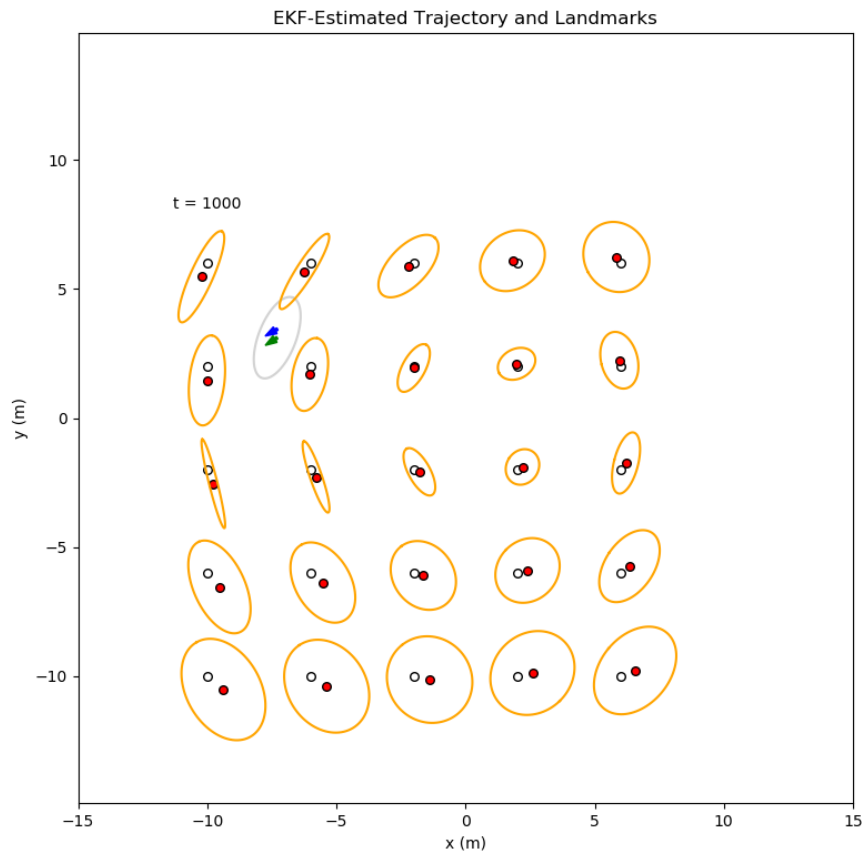


Figure 3: Performance with all covariances shown.