CSE 483: Mobile Robotics

Lecture by: Prof. K. Madhava Krishna — **Lecture** # 04
Scribe: Dhaivat Bhatt, Isha Dua — **Date**: 9th October, 2016 (Sunday)

Extended Kalman filter for SLAM(theory)

This document extensively walks through theoretical aspects of extended Kalman filter for simultaneous localization and mapping**(SLAM)**. Here, reader is assumed to have some familiarity with basics of linear algebra and brief idea of extended Kalman filter for simple localization setting.

1 SLAM problem:

Simultaneous localization and mapping**(SLAM)** is a chicken and egg problem. Our goal is to localize the robot in our world and also update our knowledge about the world. Both, the state of the environment and pose of the robot are ***interdependent***. Initially, it has been assumed that robot is totally unaware about the ***environmental features***. Robot's understanding about the environment will grow throughout its journey.

In SLAM problem, we know ***control commands*** and ***observations***. Control commands are a set of commands which move robot in the physical world. Here, let's denote control at time $\boldsymbol{t}$ by $\mathbf{u_t}$ and observation at time $\boldsymbol{t}$ by $\mathbf{z_t}$. So, in our ***SLAM*** setting, this is what we have,

$$\mathbf{u_{1:T}} = \{\mathbf{u_1}, \mathbf{u_2}, \mathbf{u_3}, \mathbf{u_4}, \ . \ . \ . \ . \ . \ . \ . \ . \ . \ , \mathbf{u_T}\} \tag{1}$$

Equation **1** is a set of commands executed. $\mathbf{u_t}$ represents a control command executed at timestep $\mathbf{t}$. This command can be as simple as telling the robot to move 1 meter ***forward*** and then asking it to ***rotate*** by 30 degrees. The representation of this control command depends on ***motion model*** of the system.

$$\mathbf{z_{1:T}} = \{\mathbf{z_1}, \mathbf{z_2}, \mathbf{z_3}, \mathbf{z_4}, \ . \ . \ . \ . \ . \ . \ . \ . \ . \ , \mathbf{z_T}\} \tag{2}$$

Equation **2** is a set of sensor observations obtained during the journey of the robot. $\mathbf{z_t}$ represents sensor observation obtained at timestep $\mathbf{t}$. Representation of sensor observation depends upon the observation model. In robotics, **Camera** and **LADAR** are widely known sensors to sense the environment. Our observation model will be typically depended upon the output of our sensor readings.

Given sensor observations and robot controls, our task is to estimate ***map*** of the environment and ***trajectory*** of the robot. Here, we are going to use extended Kalman filter to solve ***online SLAM*** problem. In online SLAM, we are interested in estimating the current state of the robot and map. Given previous pose of the robot and control applied to reach current state, our goal is to estimate current state with minimum error. Map of the environment is denoted by $\boldsymbol{m}$ and pose of the robot at time t is denoted by $\mathbf{x_t}$.

$$\mathbf{x_{0:T}} = \{\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \ . \ . \ . \ . \ . \ . \ . \ . \ . \ , \mathbf{x_T}\} \tag{3}$$

2 Extended Kalman filter:

Kalman filter is a concept for state estimation. Kalman filter is a recursive **bayes filter**. The main limitation in the formulation of Kalman filter is that it is designed for linear systems. In most of the practical robotics applications, the models we use to design the system are non-linear in nature. Extended Kalman filter**(EKF)** is one extension of Kalman filter. In ***EKF***, we linearize our system models using Taylor series. **EKF** can be broadly split up into two stages.

- **Prediction step:** In this step, we take commands sent to the robot into consideration and estimate new state of the robot by incorporating control command and pose of the robot in previous step. This new state of the robot denotes where the robot ***thinks*** it is. Which is different from where the robot ***actually*** is, this drift in the position of the robot occurs because of noise in control executed.

$$\overline{\mathbf{bel}(\mathbf{x_t})} = \int \mathbf{p}(\mathbf{x_t}|\mathbf{u_t}, \mathbf{x_{t-1}})\mathbf{bel}(\mathbf{x_{t-1}})\mathbf{dx_{t-1}} \tag{4}$$

- **Correction step:** In this step, the algorithm takes sensor observations into consideration. This is how robot is actually perceives the world. In this step, we compare the what robot ***actually*** sees and what robot is ***expected*** to see. Based on the ***mismatch/disparity***, a correction is executed, which results in ***update*** stage of the ***EKF***.

$$\mathbf{bel}(\mathbf{x_t}) = \eta \mathbf{p}(\mathbf{z_t}|\mathbf{x_t})\overline{\mathbf{bel}(\mathbf{x_t})} \tag{5}$$

3 EKF SLAM:

Here, we are considering ***EKF*** for online SLAM. It is online SLAM mainly because we are interested in estimating current pose of the robot. We are ignoring previous poses of the robot because they are not interesting for our online SLAM problem.

In our EKF localization setting, the state space is a 3D vector. In EKF SLAM, we have to incorporate estimation variables for position of the landmarks also. Here we have assumed that we are dealing with point landmarks only. Meaning, each landmark can be uniquely identified with **X** and **Y** co-ordinates. There is no orientation for any landmark. Hence, our state space will be a very large vector describing map of the environment and state of the robot at time step **t**. Here first 3 dimensions describe the pose of the robot. Remaining are X,Y locations of all the landmarks. Here, we have also assumed that the robot moves in a 2D world. Hence 3 variables are enough to define its state at any time instant t. If our robot is moving in a 3D world, we will have a 6 dimensional vector to describe the state of the robot, 3 for position and 3 for angular orientations.

In this setting, we are assuming that our data association problem is solved. Meaning, if we see a landmark, based on its environmental features, we know **ID** of that landmark. We assign a new **ID** if the landmark has not been explored.

Now, let's assume that at time step **t**, our robot has explored ***n*** landmarks. Each landmark is described using **X** and **Y** co-ordinates. So, our state space will be $\mathbf{2n+3}$ dimensional. Hence, the co-variance matrix will be of size $(\mathbf{2n+3})\mathbf{x}(\mathbf{2n+3})$. Our state space is characterized by **Gaussian distribution**.

3.1 Algorithm:

Our algorithm is broadly divided into **5** steps. Step **1** updates the state space($\bar{\mu}_\mathbf{t}$) using motion model($\mathbf{g}(\mathbf{u_t}, \mu_\mathbf{t-1})$). Some of the popular motion models are either **odometry based** or **velocity based**, we use it to estimate the expected pose($\bar{\mu}_\mathbf{t}$) of the robot at next time instant(**t**). This is the pose where robot ***thinks*** it is. Using, Taylor series, we linearize our motion model around expected pose of the robot obtained in previous step . We update co-variance using motion model **Jacobian**($\mathbf{G_t}$) and **control co-variance**($\mathbf{R_t}$). At this step, we are done with the prediction step of our recursive algorithm.

Step **3** involves computation of **Kalman gain**($\mathbf{K_t}$). Kalman gain is computed by using observation model Jacobian($\mathbf{H_t}$) and observation co-variance($\mathbf{Q_t}$). Step **4** involves updation of state space vector. Based on the disparity between expected observation($\mathbf{h}(\bar{\mu}_\mathbf{t})$) and sensor observation($\mathbf{z_t}$), state of the robot as well as the environment is updated. In step **4**, we update our state co-variance. In last step we return updated state vector($\mu_\mathbf{t}$) and co-variance($\mathbf{\Sigma_t}$).

```
1 : EXTENDED_KALMAN_FILTER($\mu_{\mathbf{t-1}}, \mathbf{\Sigma_{t-1}}, \mathbf{u_t}, \mathbf{z_t}$)
2 : $\bar{\mu}_\mathbf{t} = \mathbf{g}(\mathbf{u_t}, \mu_{\mathbf{t-1}})$
3 : $\overline{\mathbf{\Sigma_t}} = \mathbf{G_t}\, \mathbf{\Sigma_{t-1}}\, \mathbf{G_t^T} + \mathbf{R_t}$
4 : $\mathbf{K_t} = \bar{\mathbf{\Sigma}}_\mathbf{t}\, \mathbf{H_t^T}\, (\mathbf{H_t}\, \bar{\mathbf{\Sigma}}_\mathbf{t}\, \mathbf{H_t^T} + \mathbf{Q_t})^{-1}$
5 : $\mu_\mathbf{t} = \bar{\mu}_\mathbf{t} + \mathbf{K_t}\, (\mathbf{z_t} - \mathbf{h}(\bar{\mu}_\mathbf{t}))$
6 : $\mathbf{\Sigma_t} = (\mathbf{I} - \mathbf{K_t}\, \mathbf{H_t})\bar{\mathbf{\Sigma}}_\mathbf{t}$
7 : return $\mu_\mathbf{t}, \mathbf{\Sigma_t}$
```

Here is the pseudo code of the algorithm, inputs to the algorithm are state space vector($\mu_\mathbf{t-1}$), co-variance of the state space($\mathbf{\Sigma_{t-1}}$) and control($\mathbf{u_t}$) at time **t**. The algorithm returns new state vector($\mu_\mathbf{t}$) and co-variance($\mathbf{\Sigma_t}$).

3.2 EKF SLAM: Filter cycle

This section walks through each step of the algorithm in details. Here is a generic representation of our state space vector. Till time instant **t**, if our robot has identified **n** different landmarks, the state space vector will be of size $\mathbf{2n+3}$.

$$\mu_\mathbf{t} = (\underbrace{\mathbf{x}, \mathbf{y}, \theta}_{Robotpose}, \underbrace{\mathbf{m_{1,x}}, \mathbf{m_{1,y}}}_{Landmark\ 1}, \underbrace{\mathbf{m_{2,x}}, \mathbf{m_{2,y}}}_{Landmark\ 2} \cdots\cdots\cdots \underbrace{\mathbf{m_{n,x}}, \mathbf{m_{n,y}}}_{Landmark\ n})^T \quad (6)$$

As evident from equation **6**, our state space vector will start getting populated as more and more landmarks will be explored. At $\mathbf{t = 0}$, our state space vector is initialized in robot's frame of reference. Hence, $\mathbf{x_0 = (0, 0, 0)}$. In EKF, we keep appending elements to our state space vector as soon as we explore a new landmark.

Initially, robot starts in its own reference frame. At time step **0**, we have zero landmarks. Hence, our initial state space covariance will be a **3x3** matrix initialized with all zeros. As soon as we explore a new landmark, this matrix will start getting populated with variance of the map and co-variance of the map and pose.

$$\Sigma_{t=0} \;=\; \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{\Sigma_t} = \left(\begin{array}{ccc|ccccc}
\sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xm_{1,x}} & \sigma_{xm_{1,y}} & \cdots & \sigma_{xm_{n,x}} & \sigma_{xm_{n,y}} \\
\sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} & \sigma_{ym_{1,x}} & \sigma_{ym_{1,y}} & \cdots & \sigma_{ym_{n,x}} & \sigma_{ym_{n,y}} \\
\sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} & \sigma_{\theta m_{1,x}} & \sigma_{\theta m_{1,y}} & \cdots & \sigma_{\theta m_{n,x}} & \sigma_{\theta m_{n,y}} \\
\hline
\sigma_{m_{1,x}x} & \sigma_{m_{1,x}y} & \sigma_{m_{1,x}\theta} & \sigma_{m_{1,x}m_{1,x}} & \sigma_{m_{1,x}m_{1,y}} & \cdots & \sigma_{m_{1,x}m_{n,x}} & \sigma_{m_{1,x}m_{n,y}} \\
\sigma_{m_{1,y}x} & \sigma_{m_{1,y}y} & \sigma_{m_{1,y}\theta} & \sigma_{m_{1,y}m_{1,x}} & \sigma_{m_{1,y}m_{1,y}} & \cdots & \sigma_{m_{1,y}m_{n,x}} & \sigma_{m_{1,y}m_{n,y}} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\sigma_{m_{n,x}x} & \sigma_{m_{n,x}y} & \sigma_{m_{n,x}\theta} & \sigma_{m_{n,x}m_{1,x}} & \sigma_{m_{n,x}m_{1,y}} & \cdots & \sigma_{m_{n,x}m_{n,x}} & \sigma_{m_{n,x}m_{n,y}} \\
\sigma_{m_{n,y}x} & \sigma_{m_{n,y}y} & \sigma_{m_{n,y}\theta} & \sigma_{m_{n,y}m_{1,x}} & \sigma_{m_{n,y}m_{1,y}} & \cdots & \sigma_{m_{n,y}m_{n,x}} & \sigma_{m_{n,y}m_{n,y}}
\end{array}\right) \tag{7}$$

Equation 7 shows $(\mathbf{2n+3})\mathbf{x}(\mathbf{2n+3})$ covariance matrix at timestep $\mathbf{t}$. At this time step, $\mathbf{n}$ landmarks have been explored so far, in next iteration, if our robot encounters a new landmark, the algorithm will append **2 rows** and **2 columns** to the state covariance matrix. These newly added elements are initialized with ∞, which reflects the fact that we are highly **uncertain** about the state of the landmark, and will keep updating it as we revisit the landmark again and again. Also, we will append two entries corresponding to location$(\mathbf{m_x}, \mathbf{m_y})$ in the state space representation. These values will be initialized with values sensed by the sensor.

Equation **7** can be compactly written as following,

$$\mathbf{\Sigma_t} \;=\; \begin{pmatrix} \Sigma_{xx} & \Sigma_{xm} \\ \Sigma_{mx} & \Sigma_{mm} \end{pmatrix}$$

Here, matrix $\mathbf{\Sigma_{xm}}$ is a transpose of $\mathbf{\Sigma_{mx}}$. This block matrix correspond to co-variance of **map** and **pose**.

3.2.1 Prediction step: Update the state space

$$\underbrace{\bar{\mu}_t}_{\mathbf{(2n+3)x1}} \quad = \quad \mathbf{g}\ (\ \mathbf{u_t}\ , \mu_{\mathbf{t-1}}) \tag{8}$$

In this step, we are updating the expected position of the robot by assuming that robot is not modifying the environment with motion command. It's a possibility that robot can bump into a landmark and change it, but for basic setting it has been assumed that nothing like this is happening. So, only first three dimensions of the state vector(**6**) will change. Based on the motion model, $\mathbf{g}(\mu_{\mathbf{t-1}}, \mathbf{u_t})$ will update the first 3 components of the state vector, which corresponds to ***pose of the robot***. Rest of the **2n** entries are untouched.

3.2.2 Prediction step: Update co-variance

$$\underbrace{\overline{\Sigma_t}}_{\mathbf{(2n+3)x(2n+3)}} \quad = \quad \mathbf{G_t}\ \mathbf{\Sigma_{t-1}}\ \mathbf{G_t^T} \quad + \quad \mathbf{R_t} \tag{9}$$

Let there be a **Gaussian distribution** characterized by random variable $\mathbf{x}$ with covariance $\mathbf{\Sigma_{old}}$. Then, after subjecting it to a linear transformation $\mathbf{Ax+b}$, new covariance matrix is $\mathbf{\Sigma_{new}} = \mathbf{A\Sigma_{old}A^T}$. But, in our case, the equivalent transformation function(**motion model**) is non-linear in nature. So we linearize it using **taylor series** around **expected pose** obtained in step **2** of the algorithm. Hence, the motion model **Jacobian** $\mathbf{G_t}$ will be equivalent to the transformation matrix $\mathbf{A}$.

$$\mathbf{G_t} = \underbrace{\left(\begin{array}{ccc|ccccc}
\frac{\partial \bar{x}_{t+1}}{\partial x_t} & \frac{\partial \bar{x}_{t+1}}{\partial y_t} & \frac{\partial \bar{x}_{t+1}}{\partial \theta_t} & \frac{\partial \bar{x}_{t+1}}{\partial m_{1,x}} & \frac{\partial \bar{x}_{t+1}}{\partial m_{1,y}} & \cdots & \frac{\partial \bar{x}_{t+1}}{\partial m_{n,x}} & \frac{\partial \bar{x}_{t+1}}{\partial m_{n,y}} \\
\frac{\partial \bar{y}_{t+1}}{\partial x_t} & \frac{\partial \bar{y}_{t+1}}{\partial y_t} & \frac{\partial \bar{y}_{t+1}}{\partial \theta_t} & \frac{\partial \bar{y}_{t+1}}{\partial m_{1,x}} & \frac{\partial \bar{y}_{t+1}}{\partial m_{1,y}} & \cdots & \frac{\partial \bar{y}_{t+1}}{\partial m_{n,x}} & \frac{\partial \bar{y}_{t+1}}{\partial m_{n,y}} \\
\frac{\partial \bar{\theta}_{t+1}}{\partial x_t} & \frac{\partial \bar{\theta}_{t+1}}{\partial y_t} & \frac{\partial \bar{\theta}_{t+1}}{\partial \theta_t} & \frac{\partial \bar{\theta}_{t+1}}{\partial m_{1,x}} & \frac{\partial \bar{\theta}_{t+1}}{\partial m_{1,y}} & \cdots & \frac{\partial \bar{\theta}_{t+1}}{\partial m_{n,x}} & \frac{\partial \bar{\theta}_{t+1}}{\partial m_{n,y}} \\
\hline
\frac{\partial m_{1,x}}{\partial x_t} & \frac{\partial m_{1,x}}{\partial y_t} & \frac{\partial m_{1,x}}{\partial \theta_t} & \frac{\partial m_{1,x}}{\partial m_{1,x}} & \frac{\partial m_{1,x}}{\partial m_{1,y}} & \cdots & \frac{\partial m_{1,x}}{\partial m_{n,x}} & \frac{\partial m_{1,x}}{\partial m_{n,y}} \\
\frac{\partial m_{1,y}}{\partial x_t} & \frac{\partial m_{1,y}}{\partial y_t} & \frac{\partial m_{1,y}}{\partial \theta_t} & \frac{\partial m_{1,y}}{\partial m_{1,x}} & \frac{\partial m_{1,y}}{\partial m_{1,y}} & \cdots & \frac{\partial m_{1,y}}{\partial m_{n,x}} & \frac{\partial m_{1,y}}{\partial m_{n,y}} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\frac{\partial m_{n,x}}{\partial x_t} & \frac{\partial m_{n,x}}{\partial y_t} & \frac{\partial m_{n,x}}{\partial \theta_t} & \frac{\partial m_{n,x}}{\partial m_{1,x}} & \frac{\partial m_{n,x}}{\partial m_{1,y}} & \cdots & \frac{\partial m_{n,x}}{\partial m_{n,x}} & \frac{\partial m_{n,x}}{\partial m_{n,y}} \\
\frac{\partial m_{n,y}}{\partial x_t} & \frac{\partial m_{n,y}}{\partial y_t} & \frac{\partial m_{n,y}}{\partial \theta_t} & \frac{\partial m_{n,y}}{\partial m_{1,x}} & \frac{\partial m_{n,y}}{\partial m_{1,y}} & \cdots & \frac{\partial m_{n,y}}{\partial m_{n,x}} & \frac{\partial m_{n,y}}{\partial m_{n,y}} \\
 & & & \multicolumn{5}{c}{\underbrace{\hphantom{\frac{\partial m_{n,y}}{\partial m_{1,x}} \quad \frac{\partial m_{n,y}}{\partial m_{1,y}} \quad \cdots \quad \frac{\partial m_{n,y}}{\partial m_{n,x}} \quad \frac{\partial m_{n,y}}{\partial m_{n,y}}}}_{I_{2nx2n}}}
\end{array}\right)}_{\mathbf{(2n+3)\ x\ (2n+3)}}$$

Here, the state space representation is of size $\mathbf{2n+3}$. Hence, the motion model Jacobian $\mathbf{G_t}$ is of size $(\mathbf{2n+3})\mathbf{x}(\mathbf{2n+3})$. It can be computed using above matrix.

3.2.3 Correction step: Final update

In correction step, we are incorporating sensor measurements. Let's assume that we have a range-bearing sensor. Our sensor will provide distance of the landmark($\mathbf{r}$) and its orientation(ψ) in robot's frame of reference. So if we are able to observe landmark $\mathbf{i}$ at time step $\mathbf{t}$, the observation will be $\mathbf{z_t^i} = (\mathbf{r_t^i}, \phi_\mathbf{t}^\mathbf{i})$.

Here, the correction step can be executed using ***two methods***.

In first method, we run a loop over a set of landmarks sensed at this timestep. We consider **1 landmark** at a time and update the state space representation($\mu_\mathbf{t}$) and state co-variance($\mathbf{\Sigma_t}$). This method is known as **incremental EKF SLAM**. Incremental mode doesn't keep **interlandmarks** variations into consideration.

In second method, we construct a huge observation matrix and estimate our final state space representation($\mu_\mathbf{t}$) and state co-variance($\mathbf{\Sigma_t}$) in a single go. This method is known as **batch EKF SLAM**. Batch mode preserves **interlandmark** variations. We will see, that batch mode is computationally expensive, as we need to estimate inverse of a huge to compute Kalman gain($\mathbf{K_t}$).

- **Incremental mode:**

$$\underbrace{K_t}_{\mathbf{(2n+3)x2}} \;=\; \bar{\mathbf{\Sigma}}_\mathbf{t} \;\underbrace{H_t^T}_{\mathbf{(2n+3)x2}}\; (\,\mathbf{H_t}\;\bar{\mathbf{\Sigma}}_\mathbf{t}\;\mathbf{H_t^T} + \underbrace{Q_t}_{\mathbf{2x2}})^{\mathbf{-1}}$$

The above equation estimates Kalman gain($\mathbf{K_t}$). Kalman gain balances between **motion model** and **observation model**. If the uncertainly in motion model($\mathbf{Q_t}$) is more, then $\mathbf{K_t}$ is near zero. So, in subsequent updates, the new estimation will be nearly equal to what we estimated using our **motion model**. If $\mathbf{K_t}$ is high, our system will trust the observation model.

The term $\mathbf{Q_t}$ in the equation is observation covariance. For incremental mode, it can be represented as following,

$$Q_t \;=\; \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix}$$

For incremental mode, we are considering **1 landmark** at a time, $\mathbf{Q_t}$ will be of size **2x2**. So computational complexity to compute the inverse will be $\mathbf{O(2^3)} = \mathbf{O(8)}$. In worst case, if we see all the $\mathbf{n}$ landmarks, the net computational complexity of incremental mode will be $\mathbf{O(8n)}$. Observation Jacobian($\mathbf{H_t}$) can be computed using the following equation,

$$H_t \;=\; \underbrace{\begin{pmatrix} \frac{\partial r}{\partial \bar{x}} & \frac{\partial r}{\partial \bar{y}} & \frac{\partial r}{\partial \bar{\theta}} & \frac{\partial r}{\partial m_{1,\bar{x}}} & \frac{\partial r}{\partial m_{1,\bar{y}}} & \cdots & \frac{\partial r}{\partial m_{n,\bar{x}}} & \frac{\partial r}{\partial m_{n,\bar{y}}} \\ \frac{\partial \phi}{\partial \bar{x}} & \frac{\partial \phi}{\partial \bar{y}} & \frac{\partial \phi}{\partial \bar{\theta}} & \frac{\partial \phi}{\partial m_{1,\bar{x}}} & \frac{\partial \phi}{\partial m_{1,\bar{y}}} & \cdots & \frac{\partial \phi}{\partial m_{n,\bar{x}}} & \frac{\partial \phi}{\partial m_{n,\bar{y}}} \end{pmatrix}}_{\mathbf{2\ x\ (2}n\mathbf{+3)}}$$

In incremental mode, our Kalman gain($\mathbf{K_t}$) will be of size $\mathbf{(2n+3)x2}$.

Once we are done with the computation of Kalman gain, we simply update our state space representation by incorporating **disparity/mismatch** between sensor observation($\mathbf{z_t^i}$) and expected observation($\mathbf{h}(\bar{\mu}_\mathbf{t})$). $\mathbf{z_t^i}$ and $\mathbf{h}(\bar{\mu}_\mathbf{t})$ are **2x1** dimensional vectors. The final state space is obtained by following equation,

$$\underbrace{\mu_t}_{\mathbf{(2n+3)x1}} \;=\; \bar{\mu}_\mathbf{t} \;+\; \mathbf{K_t}\;(\,\underbrace{z_t \;-\; h\;(\bar{\mu}_t)}_{\mathbf{2x1}}) \tag{10}$$

Once we estimate $\mu_\mathbf{t}$, we update our expected observations($\mathbf{h}(\bar{\mu}_\mathbf{t})$) before we move on to the landmark of our incremental update. Theoretically, our $\mu_\mathbf{t}$ should be dragged towards the actual state after each iteration.

Similarly, the co-variance of the state can be updated using following equation,

$$\underbrace{\Sigma_t}_{\mathbf{(2n+3)x(2n+3)}} \quad = \quad (\,\mathbf{I} \,-\, \mathbf{K_t}\,\mathbf{H_t}\,)\bar{\mathbf{\Sigma}}_\mathbf{t} \tag{11}$$

In the above equation, a square block of $\mathbf{\Sigma_t}$ corresponding to the landmark of consideration will be updated. Hence, It is evidently clear that, **incremental EKF** doesn't update interlandmark variations, because it is only modifying block diagonal entries of $\mathbf{\Sigma_{mm}}$.

- **Batch mode:**

$$\mathbf{K_t} \quad = \quad \bar{\mathbf{\Sigma}}_\mathbf{t}\,\mathbf{H_t^T}\,(\,\mathbf{H_t}\,\bar{\mathbf{\Sigma}}_\mathbf{t}\,\mathbf{H_t^T} + \mathbf{Q_t})^{-\mathbf{1}}$$

Equation to compute Kalman gain is still same. But, the computational complexity changes. Our observation co-variance will be a diagonal matrix of size **2nx2n**.

$$Q_t \;=\; \underbrace{\begin{pmatrix} \sigma_{r_1}^2 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & \sigma_{\phi_1}^2 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \sigma_{r_2}^2 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \sigma_{\phi_2}^2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \sigma_{r_n}^2 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & \sigma_{\phi_n}^2 \end{pmatrix}}_{2n \textbf{ x } 2n}$$

As evident from the matrix $\mathbf{Q_t}$, we need to compute inverse of a large matrix, as after a few iterations, robot would have seen large number of landmarks and $\mathbf{n}$ will be larger. The computational complexity to invert this would be of $\mathbf{O(8n^3)}$. This is computationally quite expensive in comparison to incremental mode. But, batch mode preserves **interlandmark variations** and keep updating it at each time we run algorithm. Inter landmark variations correspond to off-diagonal elements in the block matrix $\mathbf{\Sigma_{mm}}$.

Observation jacobian($\mathbf{H_t}$) for batch mode is a very large matrix of size $\mathbf{2n}$ **x** $(\mathbf{2n+3})$. In this matrix, we only compute values for the landmarks we have seen. For the observation model described above, we will have **two rows** for one landmark. As we are yielding two outputs from the sensor, one correspond to distance of the landmark($\mathbf{r}$) and other correspond to relative angular orientation ϕ.

$$H_t = \underbrace{\begin{pmatrix} \frac{\partial r_1}{\partial \bar{x}} & \frac{\partial r_1}{\partial \bar{y}} & \frac{\partial r_1}{\partial \bar{\theta}} & \frac{\partial r_1}{\partial m_{1,x}} & \frac{\partial r_1}{\partial m_{1,y}} & \cdots & \frac{\partial r_1}{\partial m_{n,x}} & \frac{\partial r_1}{\partial m_{n,y}} \\ \frac{\partial \phi_1}{\partial \bar{x}} & \frac{\partial \phi_1}{\partial \bar{y}} & \frac{\partial \phi_1}{\partial \bar{\theta}} & \frac{\partial \phi_1}{\partial m_{1,x}} & \frac{\partial \phi_1}{\partial m_{1,y}} & \cdots & \frac{\partial \phi_1}{\partial m_{n,x}} & \frac{\partial \phi_1}{\partial m_{n,y}} \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \frac{\partial r_4}{\partial \bar{x}} & \frac{\partial r_4}{\partial \bar{y}} & \frac{\partial r_4}{\partial \bar{\theta}} & \frac{\partial r_4}{\partial m_{1,x}} & \frac{\partial r_4}{\partial m_{1,y}} & \cdots & \frac{\partial r_4}{\partial m_{n,x}} & \frac{\partial r_4}{\partial m_{n,y}} \\ \frac{\partial \phi_4}{\partial \bar{x}} & \frac{\partial \phi_4}{\partial \bar{y}} & \frac{\partial \phi_4}{\partial \bar{\theta}} & \frac{\partial \phi_4}{\partial m_{1,x}} & \frac{\partial \phi_4}{\partial m_{1,y}} & \cdots & \frac{\partial \phi_4}{\partial m_{n,x}} & \frac{\partial \phi_4}{\partial m_{n,y}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial r_n}{\partial \bar{x}} & \frac{\partial r_n}{\partial \bar{y}} & \frac{\partial r_n}{\partial \bar{\theta}} & \frac{\partial r_n}{\partial m_{1,x}} & \frac{\partial r_n}{\partial m_{1,y}} & \cdots & \frac{\partial r_n}{\partial m_{n,x}} & \frac{\partial r_n}{\partial m_{n,y}} \\ \frac{\partial \phi_n}{\partial \bar{x}} & \frac{\partial \phi_n}{\partial \bar{y}} & \frac{\partial \phi_n}{\partial \bar{\theta}} & \frac{\partial \phi_n}{\partial m_{1,x}} & \frac{\partial \phi_n}{\partial m_{1,y}} & \cdots & \frac{\partial \phi_n}{\partial m_{n,x}} & \frac{\partial \phi_n}{\partial m_{n,y}} \end{pmatrix}}_{2n\ \mathbf{x}\ (2n+3)}$$

Let's say our sensor has observed landmark with ID **1**. Hence, first two rows of the above Jacobian in populated with non-zero values. Similarly, our sensor hasn't seen landmarks corresponding to ID **2** and **3**. So, rows **3,4** and **5,6** corresponding to landmarks **2,3** will be zero. Reason for this is to not update the entries corresponding to the landmarks which are not sensed.

Once, we are done with estimation of Kalman gain, we update state space representation in the same way we do for incremental mode. A minor change is, our sensor observation($\mathbf{z_t}$) will be of size **2n**. Similarly, our expected observation($\mathbf{h}(\bar{\mu_\mathbf{t}})$) will also be of size **2n**. In reality, our robot perceives the local environment. So number of landmarks seen by the robot will be low compared to the large scaled map. Hence, entries in $\mathbf{z_t}$ and $\mathbf{h}(\bar{\mu_\mathbf{t}})$ will be zero for the landmarks which haven't been observed.

$$\mathbf{z_t} = (\underbrace{r_1, \phi_1}_{\mathbf{lm\ 1}}, \underbrace{0, 0}_{\mathbf{lm\,2}}, \underbrace{0, 0}_{\mathbf{lm\,3}}, \ldots\ldots.. \underbrace{r_n, \phi_n}_{\mathbf{lm\ n}})^T_{\mathbf{2nx1}}, \quad \mathbf{h}(\bar{\mu_\mathbf{t}}) = (\bar{\mathbf{r}}_\mathbf{1}, \bar{\phi}_\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \ldots\ldots.. \bar{\mathbf{r}}_\mathbf{n}, \bar{\phi}_\mathbf{n})^T_{\mathbf{2nx1}}$$

Above equation depict the fact that our sensor has sensed landmark **1**(**lm 1**) but not **2**(**lm 2**) and **3**(**lm 3**). Hence, distance($\mathbf{r}$) and relative orientation(ψ) values for the unseen landmarks are **0**.

$$\underbrace{\mu_t}_{\mathbf{(2n+3)x1}} = \underbrace{\bar{\mu_t}}_{\mathbf{(2n+3)x1}} + \underbrace{K_t}_{\mathbf{(2n+3)x2n}} \underbrace{(\ z_t \ - \ h\ (\bar{\mu_t}))}_{\mathbf{2nx1}} \tag{12}$$

Similarly, co-variance of the state space representation is computed using the following equation. Since we are considering all the landmarks at once, off diagonal entries of $\mathbf{\Sigma_{mm}}$ will also get populated. Over the time, landmarks will get correlated with each other.

$$\underbrace{\Sigma_t}_{\mathbf{(2n+3)x(2n+3)}} = (\underbrace{\ I \ - \ K_t\ H_t\ }_{\mathbf{(2n+3)x(2n+3)}})\bar{\mathbf{\Sigma}}_\mathbf{t} \tag{13}$$