Robotics: Mapping EKF

Salvador Carrillo Fuentes

December 2018

1 Introduction

In this exercise we are going to **build a map** consisting of **landmarks** (or features) using an algorithm based on **EKF** and a range-bearing sensor, provided in the exercise's appendix.

For your convenience, it is included here the slide illustrating how the algorithm performs once the sensor takes a measurement to a landmark. Two cases are possible: the landmark is observed for first time, or the landmark was already present in the map. Note: `xEst` is a vector with the coordinates of all landmarks, while `pEst` stores their associated uncertainty.

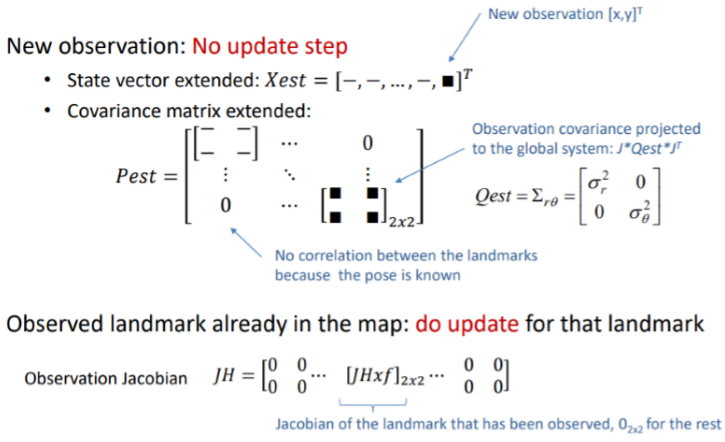

Figure 1: Two possible cases when a landmark is observed

In this case we make the assumption that **the robot's pose is known** (without uncertainty) and we want to estimate the *pdf* of the location of a number of landmarks that are present in the robot's surroundings, utilizing for that a **noisy sensor**.

2 Complete the code

- The algorithm has gaps at some key places, so your first goal is to fill them with the appropriate code. For that, first review the code that is written and understand what is going on. Concretely, your mission is to implement the Jacobians computation, as well as some stuff related to the measurements.

Once the code written has been revised, we can procced to fill the gaps. Suppose we run the program and let's walk through on an execution example. At first, we observe a random feature (a new observation k) so no update is needed because the landmark is not in our map. Before we add it to the map, we need to translate the observation to the global frame, since the measurement was taken from the local frame of the robot. For that task we use

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} x_v \\ y_v \end{bmatrix} + \begin{bmatrix} r\cos\alpha_k \\ r\sin\alpha_k \end{bmatrix}$$

being $\alpha_k = \theta_k + \theta_v$

In the implementation, first we calculate α_k, next we compute the sum of matrices showed right above and then we store the result (cartesian coordinates of the landmark in the global frame) into `xFeature` and we add it to the current state.

```
% Compute global coordinates
alfa = xVehicleTrue(3) + z(2);
xFeature = xVehicleTrue(1:2) +
           z(1) * [cos(alfa); sin(alfa)];

% Add it to the current state
xEst = [xEst;xFeature];

% Compute the jacobian
jGz = GetNewFeatureJacs(xVehicleTrue,z); %Dimension 2x2
```

The function `GetNewFeatureJacs(xVehicleTrue,z)` computes the jacobian by applying

$$J_{xy,r\theta} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \cos\alpha & -r\sin\alpha \\ \sin\alpha & r\cos\alpha \end{bmatrix}$$

```
function [jGz] = GetNewFeatureJacs(Xv, z)
    alfa = Xv(3) + z(2);
    jGz = [cos(alfa) -z(1)*sin(alfa);
           sin(alfa) z(1)*cos(alfa)];
end
```

Finally we extend the covariance matrix `PEst` by adding the observation covariance projected to the global system: `jGz*QEst*jGz'` to the bottom right of the previous version of `PEst`. Notice that covariance matrices of the other landmarks are not changed.

In the other case, when we re-observe a landmark, we do update for that landmark. That landmark is already in our map, that means the robot thinks the landmark is in certain position (x, y). After a movement, the robot re-observe that landmark obtaining a measurement `z` from the new position. `zPred` is the measurement from the new position to where the robot thinks the landmark is (`xFeature`), because that landmark is already represented in robot's map. The robot predicts where the landmark should be based on the current representation of the map.

```
zPred = getRangeAndBearing(xVehicleTrue, xFeature, Q);
```

But, since the sensor it's not perfect, `z` $\neq$ `zPred` and this difference will be used for the update.

The jacobian of the landmark that has been observed is computed applying

$$\nabla h = \frac{\partial h}{\partial\{x,y,\theta\}} = -\begin{bmatrix} -\frac{x_l - x_v}{r} & -\frac{y_l - y_v}{r} & 0 \\ \frac{y_l - y_v}{r^2} & -\frac{x_l - x_v}{r^2} & -1 \end{bmatrix}_{2\times 3}$$

being (x_l, y_l) the coordinates of the landmark, (x_v, y_v) the coordinates of the robot and r the Euclidean distance between that two points.

```
jHxf = GetObsJacs(xVehicleTrue,xFeature);
```

```
function [jHxf] = GetObsJacs(xPred, xFeature)
    delta = xFeature-xPred(1:2);
    r = norm(delta);
    jHxf = -[-delta(1) / r, -delta(2) / r;
             delta(2) / (r^2), -delta(1) / (r^2)];
end
```

Notice we only need the leftmost 2×2 matrix.

Finally, we insert `jHxf` into the state jacobian `jH` and set $0_{2\times 2}$ for the rest of landmarks.

```
jH = zeros(2,length(xEst));
jH(:,FeatureIndex:FeatureIndex+1) = jHxf;
```

Next plot shows the result of the program:

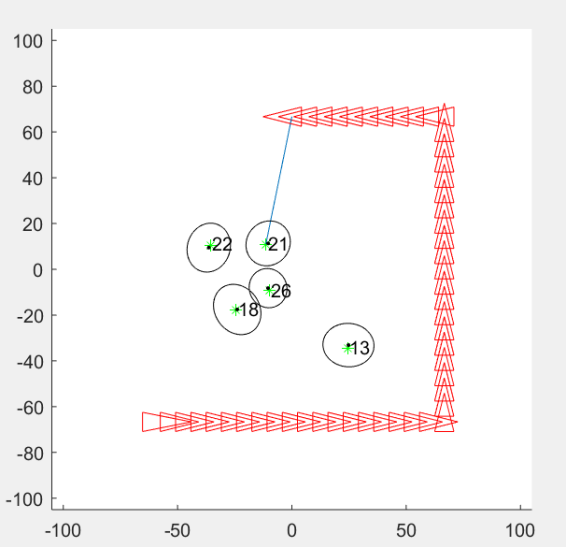

Figure 2: Plot resulting after execution with five landmarks

3 Consider that only a landmark exists

- Set the variable `nFeatures` to 1. Execute the program and show the content of the vector of states `xEst` and the covariance matrix `PEst` each 5 iterations. What dimensions do they have?

$$\texttt{xEst} = \begin{bmatrix} \blacksquare \\ \blacksquare \end{bmatrix}_{2\times 1}$$

$$\texttt{PEst} = \begin{bmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{bmatrix}_{2\times 2}$$

In the first iteration, the only landmark is observed by first time, so the state vector is extended with the (x, y) coordinates. Next, the jacobian `jGz` it's computed and the covariance matrix is extended by adding `jGz*QEst*jGz'` to the right bottom.

From now on, the dimensions will remain the same over the iterations. In the following iterations, we re-observe the only landmark and both, the state state vector and the covariance matrix are modified by an innovation and gain after computing the observation jacobian.

The next plot shows that for less uncertainty in the final estimation of a landmark we have two non-exclusive possibilities: $\uparrow$ `nSteps`, $\downarrow$ `nFeatures`.

In general, for mapping purposes, the optimal solution will be the one that finds a balance between `nSteps` and `nFeatures`.

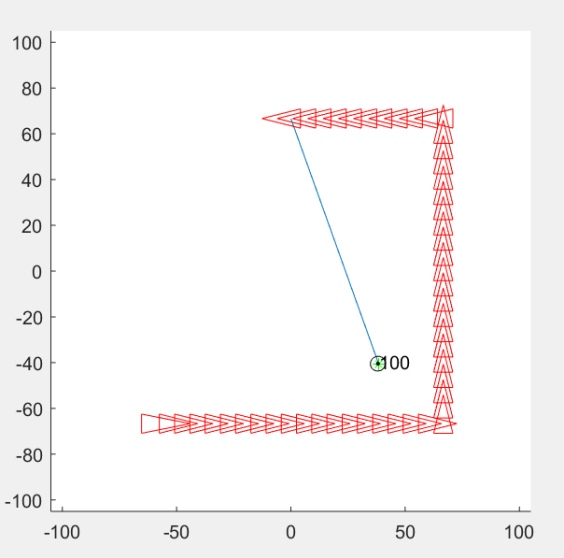

Figure 3: Plot resulting after execution with one landmark

As we can see in the previous plot, the estimation of the landmark position is more precise than with five landmarks but we don't now much about the rest of the environment.

4 Repeat the last point employing 5 landmarks

- Explain why and how the content of the variables `xEst` and `PEst` has change. Show also, each 5 iterations, the content of the jacobian of the observation (`jH`). What structure does the matrix of covariances have? Is there any kind of correlation among the observations of different landmarks?

In the first iteration, `pEst` has dimension 2×1 and `PEst` has 2×2. The process is similar to the previous section but in this case not only change the values but also change the dimensions until all landmarks are included in the map.

Each new observation of a landmark adds two rows (coordinates x, y) to `xEst` so, for five landmarks the final dimension of `xEst` will be 10×1. In the case of `PEst`, when the robot observes a landmark for first time, `jGz*QEst*jGz'` it's added to the right bottom of the previous `PEst` and the rest it's filled with zeros since there is no correlation between landmarks because the robot's pose

is perfectly known. For five landmarks, the final dimension of `PEst` will be 10×10.

For example, suppose we observed a landmark in the first iteration and in the second we observe a different one. Our variables will have the next form:

$$\texttt{xEst} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \end{bmatrix}_{4\times 1}$$

$$\texttt{PEst} = \begin{bmatrix} \begin{bmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{bmatrix} \end{bmatrix}_{4\times 4}$$

This aggregation is done every time the robot sees a landmark for first time.

When a landmark is re-observed, the observation jacobian `jH` is the jacobian of the re-observed landmark $\texttt{[jHxf]}_{2\times 2}$ and $0_{2\times 2}$ for the other landmarks. For example, suppose we have in our map three landmarks and one of them is re-observed. `jH` will be something similar to this:

$$\texttt{jH} = \begin{bmatrix} 0 & 0 & 0 & 0 & \blacksquare & \blacksquare \\ 0 & 0 & 0 & 0 & \blacksquare & \blacksquare \end{bmatrix}_{2\times 6}$$

Notice the $\blacksquare_{2\times 2}$ submatrix could be placed in columns 1-2 or 3-4 as well, it depends on the position the landmark occupies in the state vector.

5 Results of the mapping process

- Modify the code to store the determinant of the covariance matrix of each landmark and the error in the fitting along the algorithm iterations. Plot it for the case of 5 landmarks.

For doing this task, we check every element in `MappedFeatures`, that relates the index relates the index of a feature from the true map and it's place within the state (which depends on when it was observed).

If that landmark has been mapped, we procceed to compute the determinant of the covariance matrix of that landmark (that is a submatrix of `PEst`), using the relation index and store it. The error is computing as the magnitude of the estimated value and the real one.

```
for(i = 1:nFeatures)
    if(~isnan(MappedFeatures(i,1)))
        ind = MappedFeatures(i,1);
        Q_Detstore(k,i) = det(PEst(ind:ind+1,ind:ind+1));
        Err_store(k,i) = norm(xEst(ind:ind+1)-Map(:,i));
    end;
end;
```

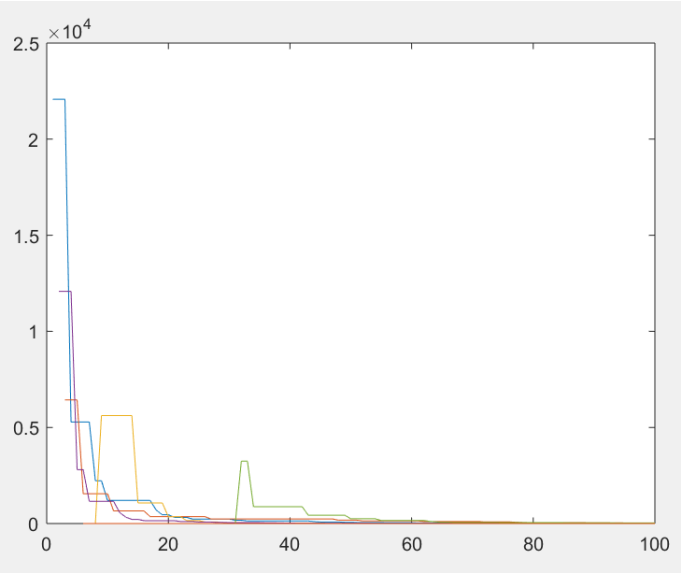

Figure 4: Determinant value through iterations for each landmark (represented by different colors)