Visual Perception: Corner detection

Guillaume Lemaître
Heriot-Watt University, Universitat de Girona, Université de Bourgogne
g.lemaitre58@gmail.com

I. Introduction

In this paper, we will present an implementation of a extended Kalman filter. The aim of this implementation is to localize a robot in a 2D world knowing correspondences. We will present the different steps that we implemented to localize the robot. First, we will present the prediction step. Then, we will introduce the updating step. Finally, we will answer to the several questions of the guideline.

II. Prediction

The first step consists on estimate the position of the robot knowing the previous position of the robot and the odometry measurements. In order to estimate the position of the robot, we have to compute the prediction and the covariance matrix. Assume that:

$$\hat{X}^B_{k-1} = [x_r\, y_r\, \theta_r]^T$$
$$u^{k-1}_k = [x_u\, y_u\, \theta_u]^T$$

A. Theory

1) Prediction: The prediction represents the estimation of the position of the robot. The prediction can be expressed as:

$$\hat{X}^B_{k-1|k} = f(X^B_{K-1}, u^{k-1}_k, w_k)$$

The previous equation is an non-linear equation. However, it can be linearized Taylor series and the first derivative approximation. The decomposition with Taylor is:

$$f(b) = f(a) + f'(a)(b-a)$$

In our case,

$$\hat{X}^B_{k-1|k} = f(\hat{X}^B_{k-1}, u^{k-1}_k, 0) + J_{1\oplus}[X^B_{k-1} - \hat{X}^B_{k-1}] + J[u^{k-1}_k - u^{k-1}_k] + J_{2\oplus}[w^{k-1}_k]$$

where $J_{1\oplus}$, J, $J_{2\oplus}$ are the Jacobean functions for each variables. When the variable is stochastic, it can be replaced by the mean. Hence:

$$\hat{X}^B_{k-1|k} = f(\hat{X}^B_{k-1}, u^{k-1}_k, 0)$$
$$\hat{X}^B_{k-1|k} = \hat{X}^B_{k-1} \oplus u^{k-1}_k$$

where $\oplus$ is the compounding operator.

Expending the compounding function, the equation can be formalized as:

$$\hat{X}^B_{k-1|k} = \begin{pmatrix} \cos(\theta_r)x_u - \sin(\theta_r)y_u + x_r \\ \sin(\theta_r)x_u + \cos(\theta_r)y_u + y_r \\ \theta_r + \theta_u \end{pmatrix}$$

2) Covariance: The covariance matrix allows to integrate the incertitude concept of the prediction. The covariance follows a normal distribution in x and y. Hence, the covariance can be formalize as:

$$P^B_{k-1|k} = J_{1\oplus} P^B_{k-1} J^T_{1\oplus} + J_{2\oplus} Q_k J^T_{2\oplus}$$

where P^B_{k-1} is the previous covariance matrix, Q_k is the noise of the odometry sensors. $J_{1\oplus}$ and $J_{2\oplus}$ are defined as:

$$J_{1\oplus} = \frac{\partial f(X^B_{K-1}, u^{k-1}_k, w_k)}{\partial (X^B_{K-1})}$$

$$J_{1\oplus} = \begin{pmatrix} 1 & 0 & -\sin(\theta_r)x_u - \cos(\theta_r)y_u \\ 0 & 1 & \cos(\theta_r)x_u - \sin(\theta_r)y_u \\ 0 & 0 & 1 \end{pmatrix}$$

$$J_{2\oplus} = \frac{\partial f(X^B_{K-1}, u^{k-1}_k, w_k)}{\partial (w_k)}$$

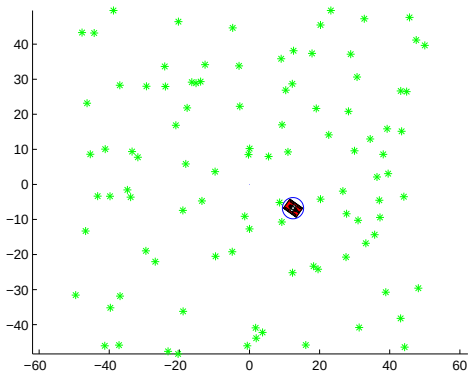

Figure 1. Initialisation of the robot

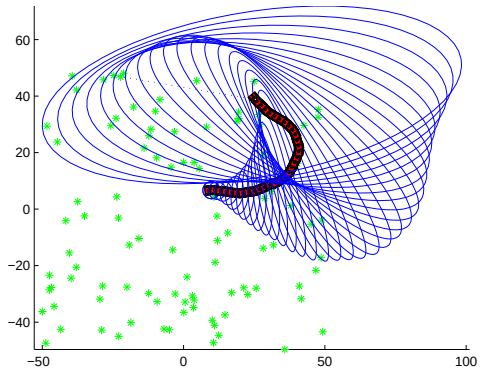

Figure 2. Robot during the prediction step

$$J_{2\oplus} = \begin{pmatrix} \cos(\theta_r) & -\sin\theta_r & 0 \\ \sin(\theta_r) & \cos(\theta_r) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

B. Results

The function is shown in the appendix A-A. In this section, results of the prediction step will be presented. Figure 1 show the state of the robot at the initialization. Figure 2 illustrates the evolution of the prediction and the incertitude (covariance matrix) of the robot. To notice that iteration after iteration the due to the noise, the incertitude increases. This phenomena is translate by the size of the ellipse in blue which increases. Figure 3 presents the all predictions did during the simulation. Graphics on figure 4 show the variations of the incertitude during the time. The increases of the ellipse saw in the figure 3 are translated by increases of errors on graphics relating the covariance x and y.

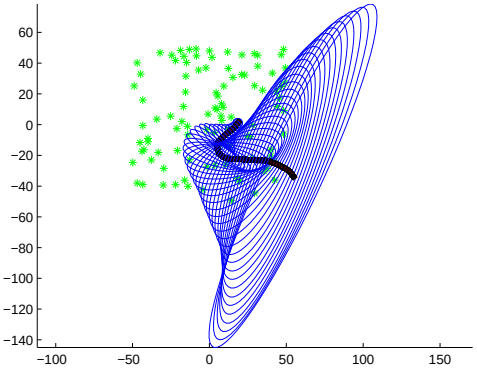

Figure 3. Robot at the end of the prediction step

III. Updating

A. Theory

The second step of the extended Kalman filter is the updating stage. This step corrects the position estimate in the prediction part using measurements. This step is realized each time that a measurement is taken. In this example, positions of features and the features observed is known. The aim is to compute an estimation of the position of the feature observed from the robot. This function is called $h(\hat{X}^B_{k-1|k}, F_{obs})$. The measurement taken by the robot is an absolute measurement and will allow to correct the position. This measurement

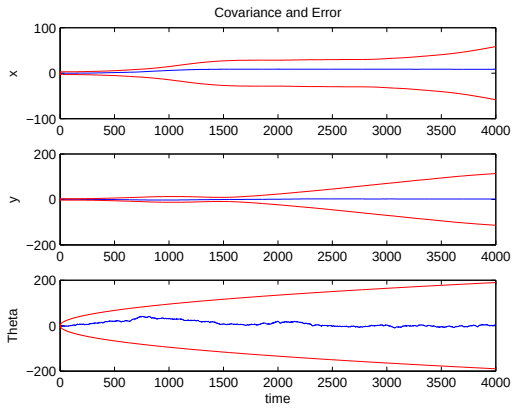

Figure 4. Graphics representing the covariance matrix (incertitude) of the robot

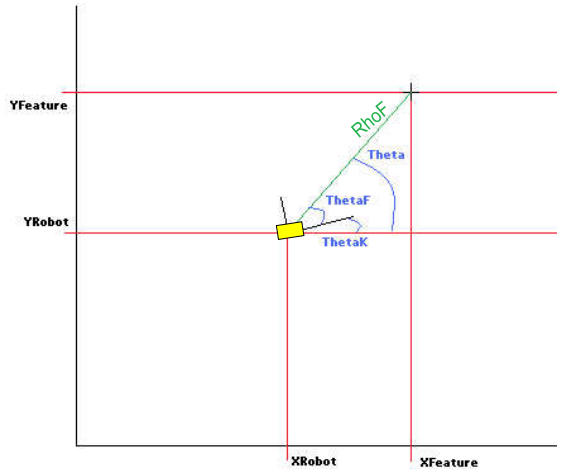

Figure 5. Representation of the problem to obtain the range and the angle

is called z_k. The measurement function can formalize as:

$$z_k = h(\hat{X}^B_{k-1|k}, F_{obs}) + V_k$$

where V_k is the noise.

1) Prediction position of the feature: $h(\hat{X}^B_{k-1|k}, F_{obs})$ returns the range and the angle of the feature observed from the robot. Figure 5 give a graphic representation of the problem. The function have to return ρ_F and θ_F using $\hat{X}^B_{k-1|k}$ and F_{obs} where:

$$\hat{X}^B_{k-1} = [x_r\ y_r\ \theta_r]^T$$
$$F_{obs} = [x_f\ y_f]^T$$

Hence, the function $h(\hat{X}^B_{k-1|k}, F_{obs})$ can be formalized as:

$$\rho_f = \sqrt{(x_f - x_r)^2 + (y_f - y_r)^2}$$
$$\theta_f = \arctan 2(\tfrac{y_f - y_r}{x_f - xr}) - \theta_r$$

Hence:

$$h(\hat{X}^B_{k-1|k}, F_{obs}) = [\rho_f\ \theta_f]^T$$

Then, the innovation and the uncertainty have to be computed. The innovation represents the difference between the measurement given by the sensor and the prediction using the function $h(\hat{X}^B_{k-1|k}, F_{obs})$. The uncertainty represents the addition of the incertitude of the measurement (sensors noise) and the incertitude of the prediction of the position of the robot ($\hat{P}^B_{k-1|k}$). Innovation and uncertainty can be formalized as follow:

$$\nu = z_k - h(\hat{X}^B_{k-1|k}, F_{obs})$$
$$S = H_k \hat{P}^B_{k-1|k} H_k^T + R_k$$

where H_k is the Jacobean matrix of the function $h(\hat{X}^B_{k-1|k}, F_{obs})$ and R_k is the noise of the sensors taken measurements.

H_k is defined as follow:

$$H_k = \begin{pmatrix} -\frac{2x_f - 2x_r}{2\sqrt{(x_f - x_r)^2 + (y_f - y_r)^2}} & -\frac{2y_f - 2y_r}{2\sqrt{(x_f - x_r)^2 + (y_f - y_r)^2}} & 0 \\ \frac{y_f - yr}{(x_f - xr)^2(\frac{(y_f - y_r)^2}{(x_f - x_r)^2} + 1)} & -\frac{1}{(x_f - xr)(\frac{(y_f - y_r)^2}{(x_f - x_r)^2} + 1)} & -1 \end{pmatrix}$$

Before to update the position of the robot, the gain K has to be computed:

$$K_k = \hat{P}^B_{k-1|k} H_k^T S^{-1}$$

Then, the position can be update as follow:

$$X^B_k = \hat{x}^B_{k-1|k} + K_k \nu$$
$$P^B_k = (I - K_k H_k) \hat{P}^B_{k-1|k} (I - K_k H_k)^T + K_k R_k K_k^T$$

B. Results

The function is shown in the appendix A-B.

1) Example 1: In this first example, during each iteration, the position of the robot is estimate and corrected with measurement. However, on the graphics figure 7, no measurement are taken between the time 2000 and 2500. The result of these missing measurements is that the graphics representing the covariance and errors show an increase of the value during this period. On the figure 6, this phenomenon is translated by an increase of the covariance ellipse.

2) Example 2: The second example shows with more accuracy the problem of non-measurement. In fact, on the graphics 7, no measurement is taken after the time 2000. Hence, the prediction realized during each iteration is not correct. So, the incertitude of the position of the robot increase that it is translate by errors increase on graphics of figure 7 and the size of the ellipse increase on the figure 6.

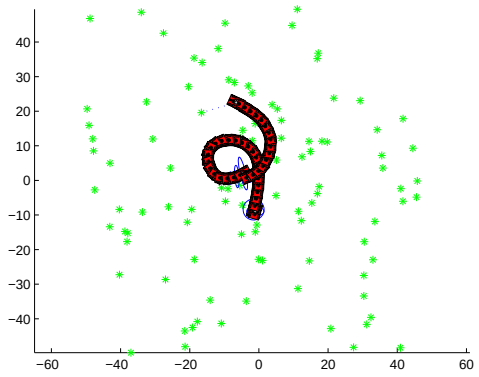

Figure 6. Example 1

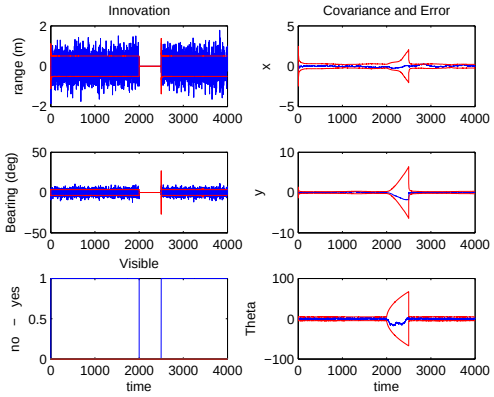

Figure 7. Example 1

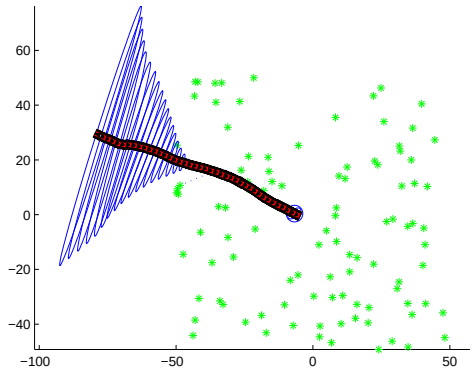

Figure 8. Example 2

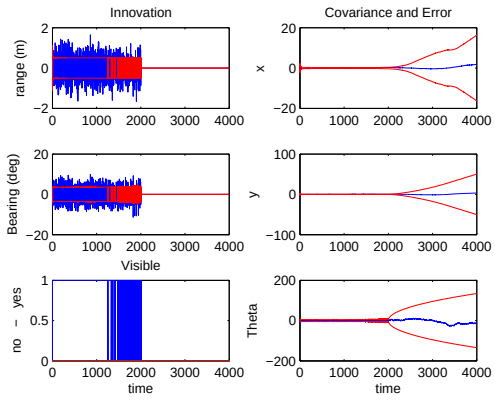

Figure 9. Example 2

IV. Other questions

In this section, we will answer to the questions written in the guideline.

A. Sensor default measurement

In the code, Faultingk0 and FaultGap allows to simulate a sensors default which stop measurement. During this non-acquisition, the prediction is not corrected and the uncertitude increase. These results were commented in the previous part.

B. Motion simulation

The motion simulation is computed inside the function *SimulateWorld*. This function is organised in two steps: computation of an absolute motion and noise generation. The absolute motion is computed using an homogenous matrix which permit to draw the new position using the position robot and the velocity. The second step is to generate a noise which will be add to the absolute value. In the case of an angular velocity of zero degrees per seconds, we can note that the robot turn a little due to the noise added. Hence, after the first iteration, the different velocities $x\,\theta$ are not equal to zero. The figure 10 shows an example of simulation without adding noise. We can see that the robot move in straight line because all velocities of the robot stay constant and the angular velocity was 0. The trajectory of the robot is not a perfect straight line due of the incertitude of the prediction and correction of the measurement.

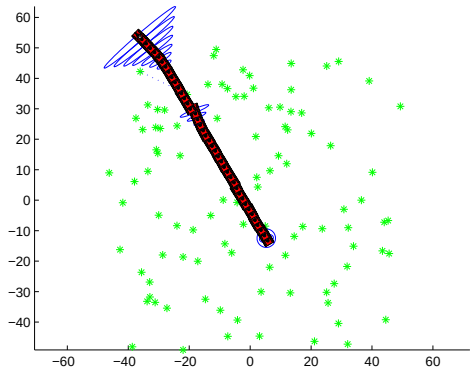

Figure 10. Motion simulation without noise added

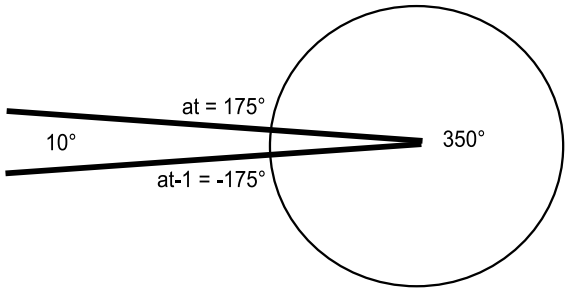

Figure 11. Explaination of the AngleWrap function

C. *AngleWrap justification*

AngleWrap function allow to convert an angle between $[-\pi, \pi]$. We can present an example. For instance, if the angle robot at the time $t-1$ was $-175°$ and at time t the measurement give an angle at $175°$. The motion between t and $t-1$ give a rotation of $175° + 175° = 350°$. Figure 11 shows the representation. In reality, the motion is not an angle of $350°$ but an angle of $10°$ as shown on 11.

V. Conclusion

In this paper, we presented an implementation of an extended Kalman filter using as example a robot moving in a 2D world knowing correspondences. First we explained how we predicted the position of the robot. Then how we corrected the predicted position using the measurements.

APPENDIX A
CODE

A. *Prediction function*

```
function [xHat_B_kIk_1, PHat_B_kIk_1] = move_vehicle(x_B_kIk_1, P_B_kIk_1, uk, Qk)

%%% Formalize parameters
%%% XBK_1
xbk_1 = x_B_kIk_1(1);
ybk_1 = x_B_kIk_1(2);
thbk_1 = x_B_kIk_1(3);
%%% XK_1K
xk_1k = uk(1);
yk_1k = uk(2);
thk_1k = uk(3);

%%% Compute function to find xHat_b_kIk_1

xHat_B_kIk_1 = [ cos(thbk_1)*xk_1k - sin(thbk_1)*yk_1k + xbk_1 ; ...
                 sin(thbk_1)* xk_1k + cos(thbk_1)*yk_1k + ybk_1 ;...
                 AngleWrap(thbk_1 + thk_1k) ];

%%% Compute the covariance prediction
%%% Computation of the Ak matrix
Ak = [ 1, 0, (-sin(thbk_1)*xk_1k -cos(thbk_1)*yk_1k) ;...
        0 ,1 , (cos(thbk_1)*xk_1k -sin(thbk_1)*yk_1k) ; ...
        0, 0, 1];
%%% Computation of the Wk matrix
Wk = [ (cos(x_B_kIk_1(3))), -sin(x_B_kIk_1(3)), 0 ; ...
    sin(x_B_kIk_1(3)), cos(x_B_kIk_1(3)), 0; ...
    0 ,0, 1 ];
%%% Finally compute the prediction of the covariance matrix
PHat_B_kIk_1 = Ak*P_B_kIk_1*Ak' + Wk*Qk*Wk';
```

B. *Update function*

```
%%% Update
function [x_B_k, P_B_k, Innovk, S] = update_position(xHat_B_kIk_1, PHat_B_kIk_1, zk, Rk, cFeature)

xRobot = xHat_B_kIk_1(1);
yRobot = xHat_B_kIk_1(2);
thRobot = xHat_B_kIk_1(3);

xFeature = cFeature(1);
yFeature = cFeature(2);

%%% Compute the prediction range and angle
rangepred = sqrt((xFeature - xRobot)^2+(yFeature - yRobot)^2);
anglepred = AngleWrap(atan2((yFeature - yRobot),(xFeature - xRobot)) - thRobot);

%%% Compute the Jacobian matrix
% syms xRobot yRobot thRobot xFeature yFeature
% h = [ sqrt((xFeature - xRobot)^2+(yFeature - yRobot)^2); ...
% atan((yFeature - yRobot)/(xFeature - xRobot)) - thRobot];
% X = [xRobot,yRobot,thRobot];
% R = jacobian(h,X);

%%% Jacobian matrix
Hk = [-(2*xFeature - 2*xRobot)/(2*((xFeature - xRobot)^2 + (yFeature - yRobot)^2)^(1/2)), ...
      -(2*yFeature - 2*yRobot)/(2*((xFeature - xRobot)^2 + (yFeature - yRobot)^2)^(1/2)),  0;
      (yFeature - yRobot)/((xFeature - xRobot)^2*((yFeature - yRobot)^2/(xFeature - xRobot)^2 + 1)), ...
      -1/((xFeature - xRobot)*((yFeature - yRobot)^2/(xFeature - xRobot)^2 + 1)), -1];
```

```
%%% Computation of the innovation
Innovk = zk - [rangepred; anglepred];
S = Rk + Hk*PHat_B_kIk_1*Hk';
Khk = PHat_B_kIk_1*Hk'*(S)^-1;

%%% Update the position and the covariance matrix
x_B_k = xHat_B_kIk_1 + Khk*Innovk;
P_B_k = ([1 0 0 ; 0 1 0 ; 0 0 1] - Khk*Hk)*PHat_B_kIk_1*([1 0 0 ; 0 1 0 ; 0 0 1] - Khk*Hk)' + Khk*Rk*Khk';
```