MORBO Localization Lab Report

-Xihao Wang and Ragesh Ramachandran

1. Lab 1 - MATLAB Code

In the MATLAB code, the main file is "*MagnetLoc.m*". Therefore, we start from this part of the code and explain the methodology which we used in this lab to locate the moving robot.

At the first part, we set the robot initial position X = [*0 0 0*pi/180*].' in lab1 and replace it to X = [*0 0 0*pi/180 rwheel+0 rwheel+0*].'. Moreover, rwheel+0 will change in the following part and we set the noise equal to zero temporarily. Then, load the data file about the moving robot. Next, we use a function named *PreprocessData* to skip motionless part of the data which record the robot's posture after it starts and after it stops. When we skip those no meaning data, we extract the robot position at each period and other useful data including left and right wheel position in radians*(ql, qr),* Raw sensor data, the time instants of each period and the number of the loop.

In the second step, we need to calculate the equation about robot state vector:

$$X_{k+1} = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta D_k \cos\theta_k \\ y_k + \Delta D_k \sin\theta_k \\ \theta_k + \Delta\theta_k \end{bmatrix} = f\left(X_k, U_k\right)$$

So, we need the parameters: ΔD_k and $\Delta\theta_k$. And we build the matrix U and matrix Δq (delta q) and matrix JointToCartesian

$$U_k = \begin{bmatrix} \Delta D_k \\ \Delta\theta_k \end{bmatrix} = \begin{bmatrix} (r_r \Delta q_{r,k} + r_l \Delta q_{l,k}) / 2 \\ (r_r \Delta q_{r,k} - r_l \Delta q_{l,k}) / e \end{bmatrix}$$

$$\Delta q = \begin{bmatrix} q_{r,k} - q_{r,k-1} \\ q_{l,k} - q_{l,k-1} \end{bmatrix} \quad deltaq = [\text{qR(mainLoopIndex) - qR(mainLoopIndex-1) ;} \\ \text{qL(mainLoopIndex) - qL(mainLoopIndex-1)] ;}$$

$$JointToCartesian = \begin{bmatrix} rwheel/2 & rwheel/2 \\ rwheel/trackGauge & -rwheel/trackGauge \end{bmatrix}$$

All those parameters have been defined in the file" *RobotAndSensorDefinition.m*". So, the matrix U could be expressed as *U = jointToCartesian * deltaq*

When we defined that matrix, we are able to calculate the equation of the extended Kalman filter's prediction phase.

$$P_{k+1/k} = A_k * P_{k/k} / A_k^T + B_k * Q_\beta / B_k^T + Q_\alpha$$

According to the equation, the $A_k = \frac{\partial f}{\partial X}(\hat{X}_{k/k}, U_k)$ and $B_k = \frac{\partial f}{\partial U}(\hat{X}_{k/k}, U_k)$.

So, we are able to use the Jacobian matrix to express the A_k and B_k

$$A_k = \frac{\partial f}{\partial X}(\hat{X}_{k/k}, U_k) = \begin{bmatrix} 1 & 0 & \text{-U(1)*sin(X(3))} \\ 0 & 1 & \text{U(1)*cos(X(3))} \\ 0 & 0 & 1 \end{bmatrix}$$

$$B_k = \frac{\partial f}{\partial U}(\hat{X}_{k/k}, U_k) = \begin{bmatrix} \text{cos(X(3))} & 0 \\ \text{sin(X(3))} & 0 \\ 0 & 1 \end{bmatrix}$$

Moreover, we have to create a file to write the function named "*EvolutionModel*" to implement the odometry measurement of the robot. And this function is the same as the equation which expresses the state vector

$$\hat{X}_{k+1/k} = f(\hat{X}_{k/k}, U_k) = \begin{bmatrix} \text{Xold(1)+U(1)*cos(Xold(3));} \\ \text{Xold(2)+U(1)*sin(Xold(3));} \\ \text{Xold(3)+U(2)} \end{bmatrix}$$

When we finished the equation about predicted phase, we can enter the estimation part or measurement part. In this part, we have been got the measurement data from the odometry data what we have done above. In other words, we have been got Y which is the measurement point in robot frame.

Therefore, our work is to detect the closest magnet and using the magnet's location to locate the robot's position. Because of that noise, for example, the input noise and state noise, the location of the robot is a Gaussian Cloud rather than a precise point. Moreover, the magnet position is also a Gaussian Cloud, because every magnet has its own magnetic field. So we need to use a series of the equations to find the closest magnet and use this magnet's position to calculate the expected position of the robot $\hat{Y}$.

At first step. We need to transfer this point into world frame by matrix ${}^{0}T_{m}$ and the robot position in world frame ${}^{o}P_{est}$.

$$ {}^{o}T_m = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(X(3)) & -\sin(X(3)) & X(1) \\ \sin(X(3)) & \cos(X(3)) & X(2) \\ 0 & 0 & 1 \end{bmatrix} $$

$$ oPest = oTm * Y $$

Then, because the magnets are arranged at a specific distance from each other. we are able to get the position of the magnet ${}^{o}P_{est}$. In the file" *RobotAndSensorDefinition*", we have sat the $X_{spacing}$ and $Y_{spacing}$ are both equal to 55mm.

$$ oPmagnet = round(\frac{x_{robot}}{55}, \frac{y_{robot}}{55}) * 55 $$

$$ {}^{o}P_{mag} = \begin{bmatrix} {}^{o}X_{mag} \\ {}^{o}Y_{mag} \\ 1 \end{bmatrix} = round\left(oPest / \begin{bmatrix} X_{spacing} \\ Y_{spacing} \\ 1 \end{bmatrix} \right) * \begin{bmatrix} X_{spacing} \\ Y_{spacing} \\ 1 \end{bmatrix} $$

Now, we can build the equation which use the predict state X and magnet position to calculate the expected position of the robot, Y hat.

$$ \hat{Y} = {}^{m}T_{o} * {}^{o}P_{mag} = \begin{bmatrix} \cos\theta({}^{o}X_{mag} - X) + \sin\theta({}^{o}Y_{mag} - Y) \\ \cos\theta({}^{o}Y_{mag} - Y) - \sin\theta({}^{o}X_{mag} - X) \\ 1 \end{bmatrix} = \begin{bmatrix} g_{mag}(X) \\ 1 \end{bmatrix} $$

Moreover, we also have to build the matrix C which will express how the output varies when the robot moves around the posture *X* at which the function is calculated. And we have to use the matrix C to calculate *Mahalanobis* distance.

$$ C_k = \frac{\partial g_{mag}}{\partial X} = \begin{bmatrix} \text{-cos(X(3)) -sin(X(3)) -sin(X(3))*(oPmagnet(1)-X(1))+cos(X(3))*(oPmagnet(2)-X(2));} \\ \text{sin(X(3)) -cos(X(3)) -sin(X(3))*(oPmagnet(2)-X(2))-cos(X(3))*(oPmagnet(1)-X(1))} \end{bmatrix} $$

Mahalanobis distance is the equation coherent between the measurement vector Y and its predict value $\hat{Y}$ (Y hat).

$$d^2 = (Y_k - \hat{Y}_{k+1/k})^T (C_k \cdot P_{k+1/k} \cdot C_k^{\,T} + Q_\gamma)^{-1} (Y_k - \hat{Y}_{k+1/k})$$

$$dMaha = \text{sqrt(innov.' * inv(C*P*C.' + Qgamma) * innov)}$$

Finally, we select the point which smaller than the threshold and utilize extended Kalman filter to calculate a satisfied and credible position to locate the robot location.

2. Tuning the extend Kalman filter

After building the matrix of the filter, we have to define the noise parameters, they are Pinit (error of initial position) Q_{alpha} (state noise), Q_{beta} (input noise) Q_{gamma} (measurement noise). We have to define those noise parameters and tune them. Otherwise, the filter will not work correctly. All those parameters are in the file named" *DefineVariances.m*".

Firstly, the error of initial position is caused by the reason that we can't place the robot at a precise point. So we define a very small value, less than 5, of x and y-direction and 2*pi/180 of Theta.

```
Pinit = diag( [sigmaX^2 sigmaY^2 sigmaTheta^2] )
sigmaX     = 4
sigmaY     = 4
sigmaTheta = 2*pi/180
```

If we increase the value of the *sigmaX*, *sigmaY* and *sigmaTheta*, the first series data will be influenced. However, the value which we increased will not influence the following data, because of other constrain function correct the data. For example, we define the *sigmaX*=40.

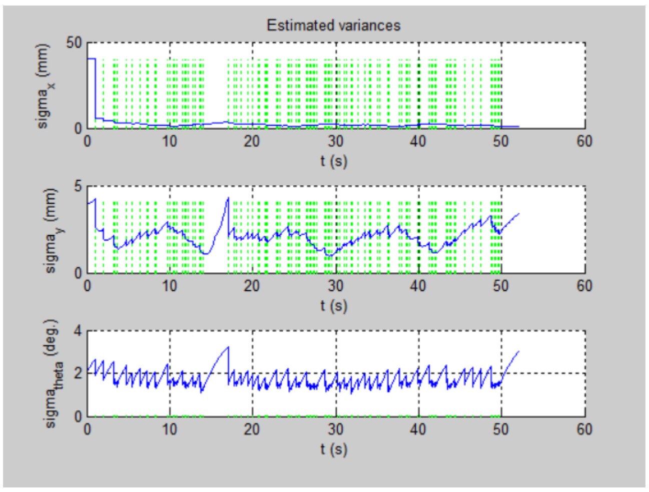

Figure 1: variances.

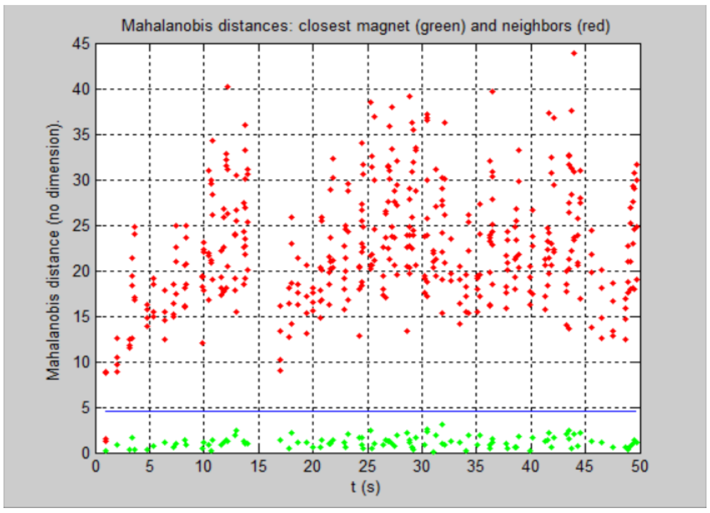

Figure 2: Mahalanobis distances.

Secondly, the measurement noise is caused by the magnet field of each magnet. So we have to analyse the diameter of the magnetic field. And we are able to find the answer in the following figure.

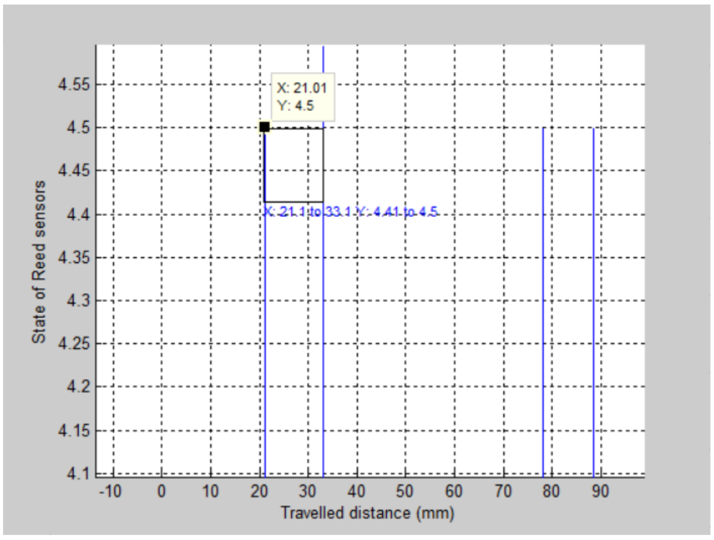

Figure 3: raw sensor data.

About the deviation in x coordinate, we found that there are two blue lines which means the sensor detected the magnet, but the distance is approximate 10mm rather than 55mm which is the distance between two magnets. So, we assume that this is because the sensor detected the same magnet twice and the diameter of the magnet field is equal to 20mm. About the deviation in y coordinate, the main reason that causes the noise is the distance between each sensor and this distance is equal to 10mm. Moreover, these noises are distributed in Gaussian distribution.

$$Gaussian = \frac{b-a}{\sqrt{12}}$$

sigmaXmeasurement = sqrt(20^2/12)
sigmaYmeasurement = sqrt(10^2/12)
Qgamma = diag([sigmaXmeasurement^2 sigmaYmeasurement^2]) ;

If modified the value of the *Qgamma*, it will influence the Mahalanobis equation. For example, if we decrease the value of *sigmaX* measurement to *sqrt(*5^2*/12)*, the green dot in figure 4 will above the threshold line. This is because the too small defined value of magnet field diameter let the robot assume the closest magnet is two different magnets. According to the equation of the Mahalanobis, this modification will increase the Mahalanobis distance to let it longer than the threshold.

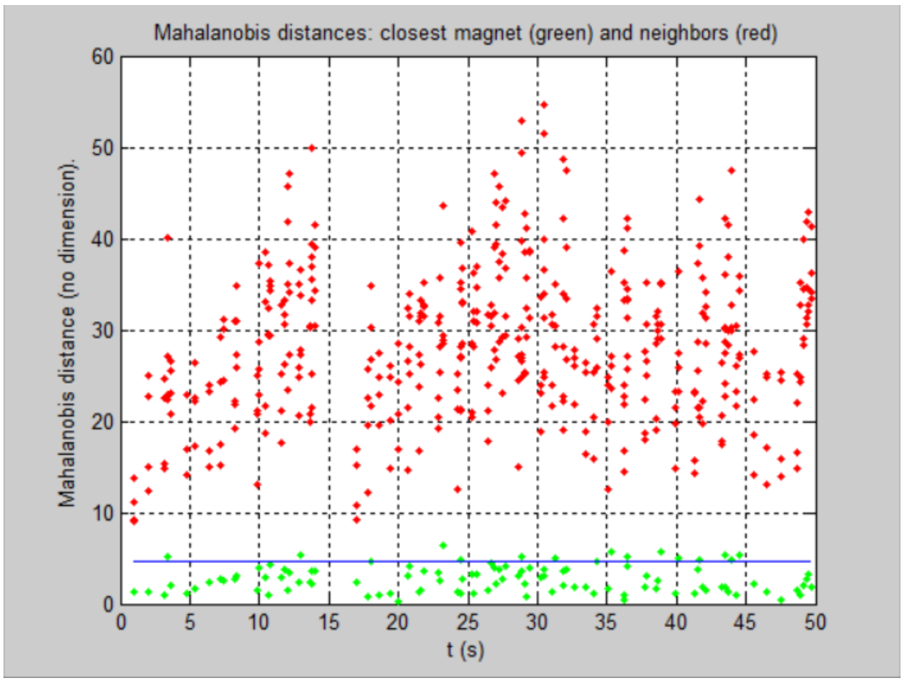

Figure 4: Mahalanobis distances.

And if we increase the value of *sigmaYmasurement*, it will make the dot in figure 4 much more sparse. For example, we define the *sigmaXmeasurememt* to sqrt(50^2/12). Obviously, the *Qgamma* didn't play its role in constraining the error propagation.

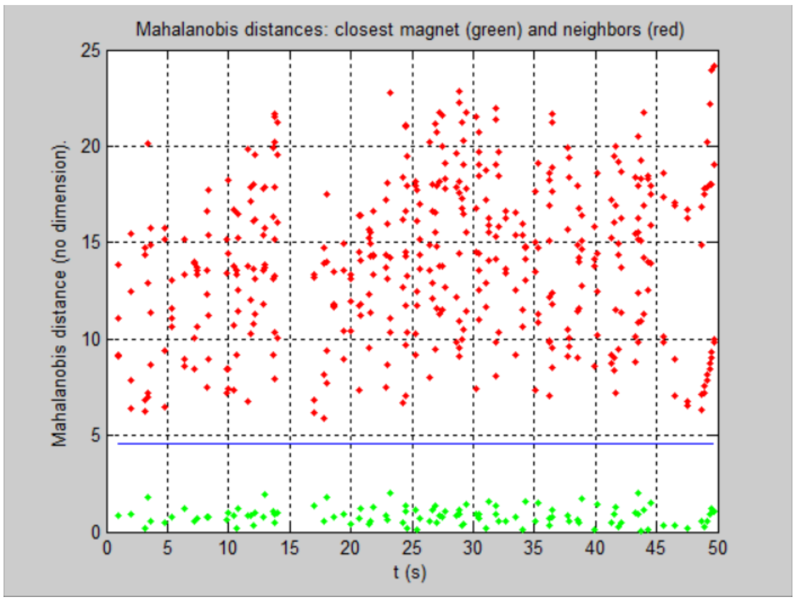

Figure 5: Mahalanobis distances.

Next, we define the *Qalpha* and *Qbeta*. If we define the Qbeta to zero and repeated odometry phases are performed, and provided $P0=0 = diag(\sigma 02; \sigma 02; \sigma\theta 20)$, the uncertainty ellipse in the x - y plane will be a circle of growing radius. This does not fit with how uncertainty grows when performing odometry: uncertainty tends to grow faster in the direction perpendicular to the motion, and so the orientation of the uncertainty ellipse should change when the robot moves

So, we chose a more satisfactory way that to define *Qalpha* to zero and tune *Qbeta*. This is because $Q_\beta = M * Q_{wheel} * M^T$.

$$\begin{bmatrix} \Delta D_k \\ \Delta\theta_k \end{bmatrix} = \begin{bmatrix} \frac{r_r}{2} & \frac{r_l}{2} \\ \frac{r_r}{2e} & \frac{r_l}{2e} \end{bmatrix} * \begin{bmatrix} \Delta q_{r,k} \\ \Delta q_{l,k} \end{bmatrix} = M * \begin{bmatrix} \Delta q_{r,k} \\ \Delta q_{l,k} \end{bmatrix}$$

In practice, there is no reason to consider the noise as being different for the right and the left wheel, so

$$\begin{bmatrix} \Delta q_{r,k} \\ \Delta q_{l,k} \end{bmatrix} = \begin{bmatrix} \sigma_{wheel} & 1 \\ 1 & \sigma_{wheel} \end{bmatrix}.$$

Then, we define sigmaWheel equal to 0.05. If we increase this value to 0.125, the value of sigmaX and sigmaY will both increase up to 5mm in figure 3 and the red dot will below the threshold line in figure 4. This is because that if we define the larger sigamWheel value, the uncertain ellipse will increase too fast and estimated path will unable to constrain it and let it back to the real path trajectory again.

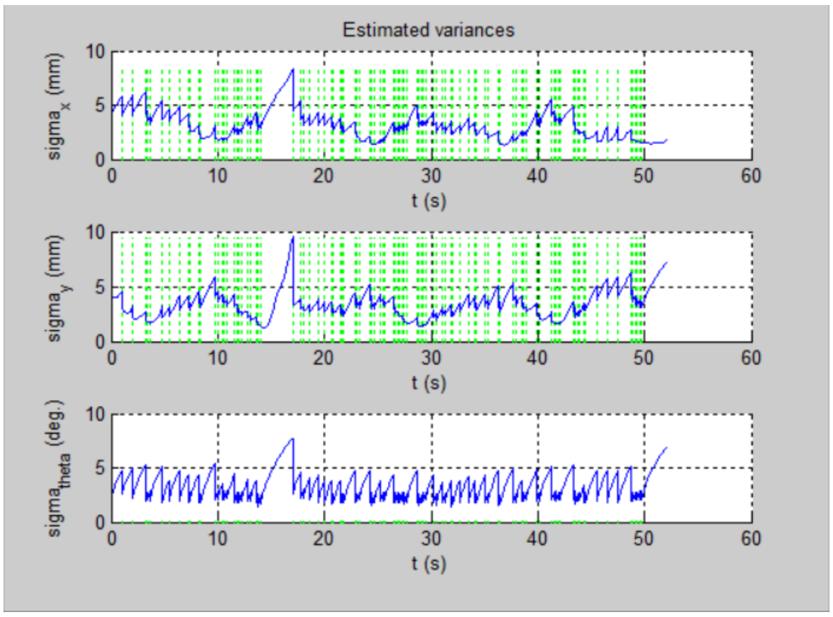

Figure 6: variances.

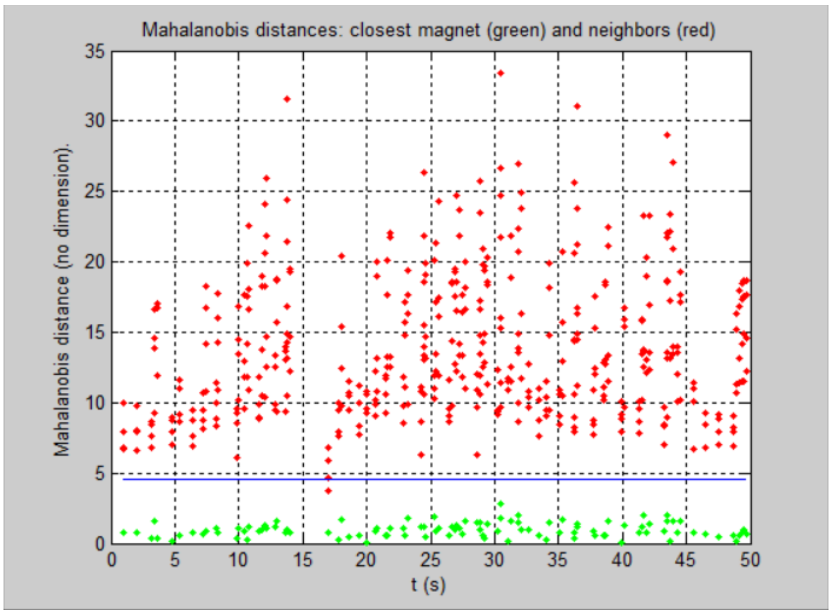

Figure 7: Mahalanobis distances.

Finally, we have to define the threshold value. We used Chi-square test function to judge whether measurement value Y and the estimated value Y hat are coherent reliably.

mahaThreshold = chi2inv(0.9,2) ;

In this sentence, we define that if the coherent degree of those two values is more than 90%, we can assume the estimated value Y hat is calculated by the closest magnet with the robot position, measurement value Y. Undoubtedly, only the coherent degree is more than at least 90% could we assume that they are coherent. So, we don’t need to modify this value and the modified result is increasing or decreasing the height of the blue line which means the threshold value.

When we finished all those noises tuning, we are able to get the correct graphs. For example, the two loops graphs.

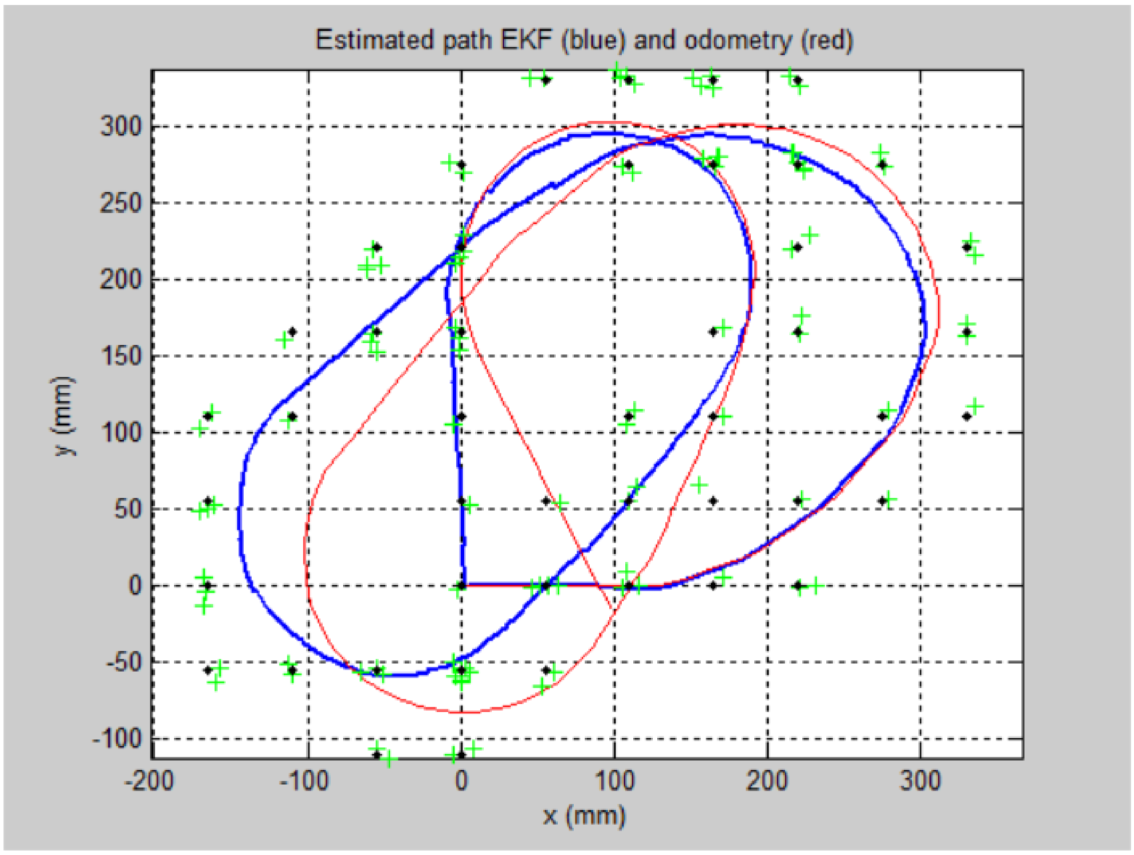

Figure 8: paths.

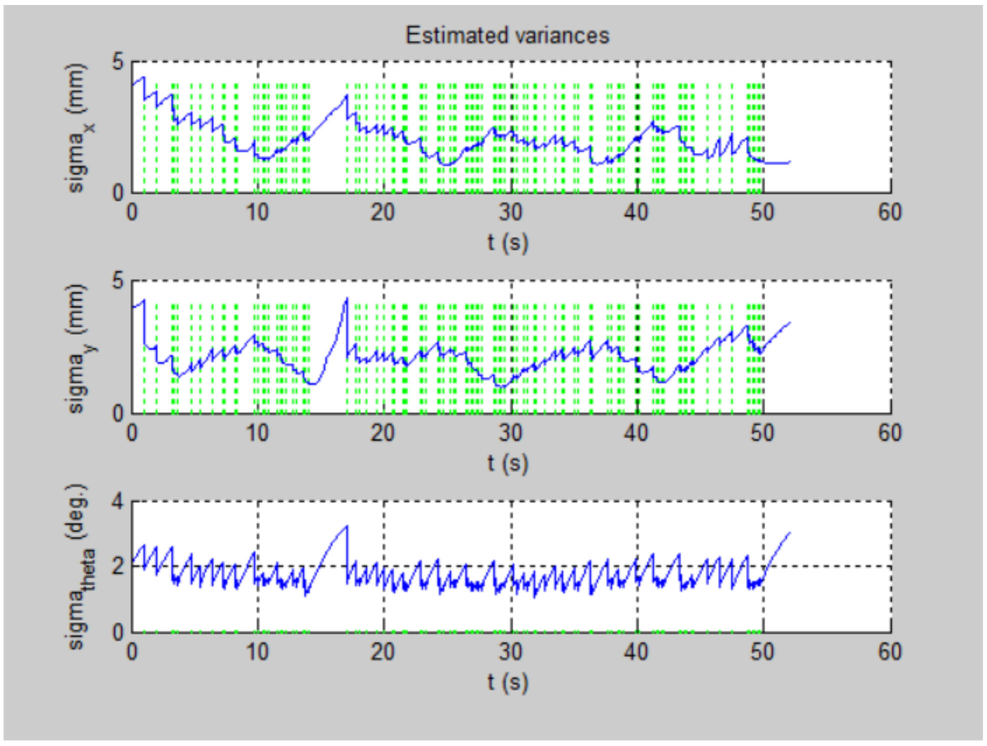

Figure 9: Variances.

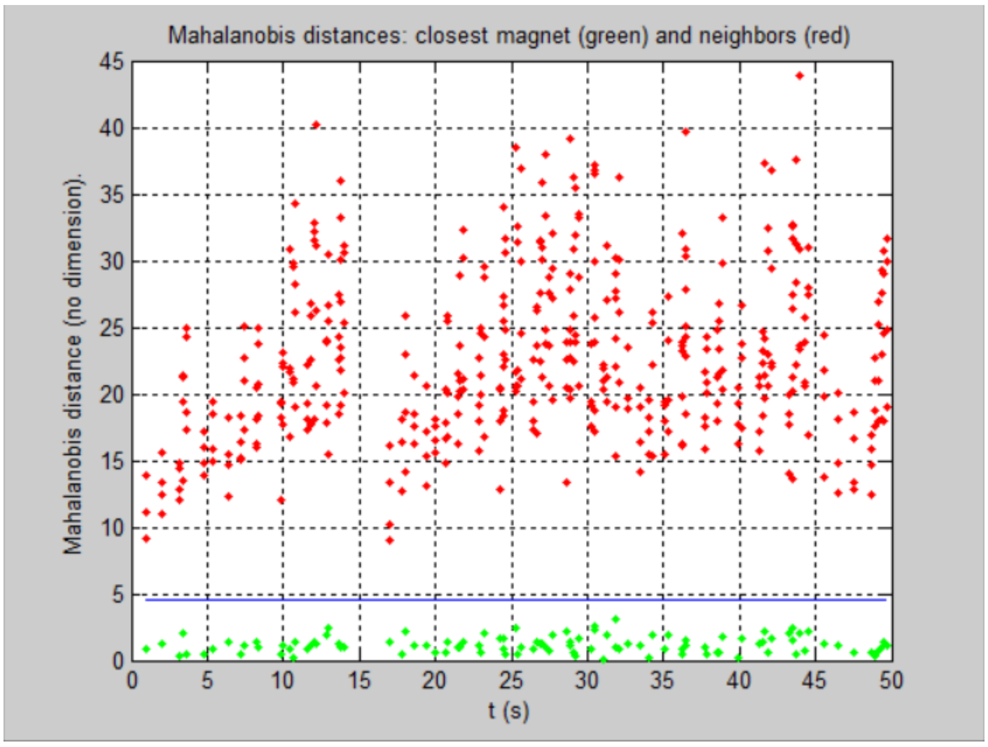

Figure 10: Mahalanobis distance.

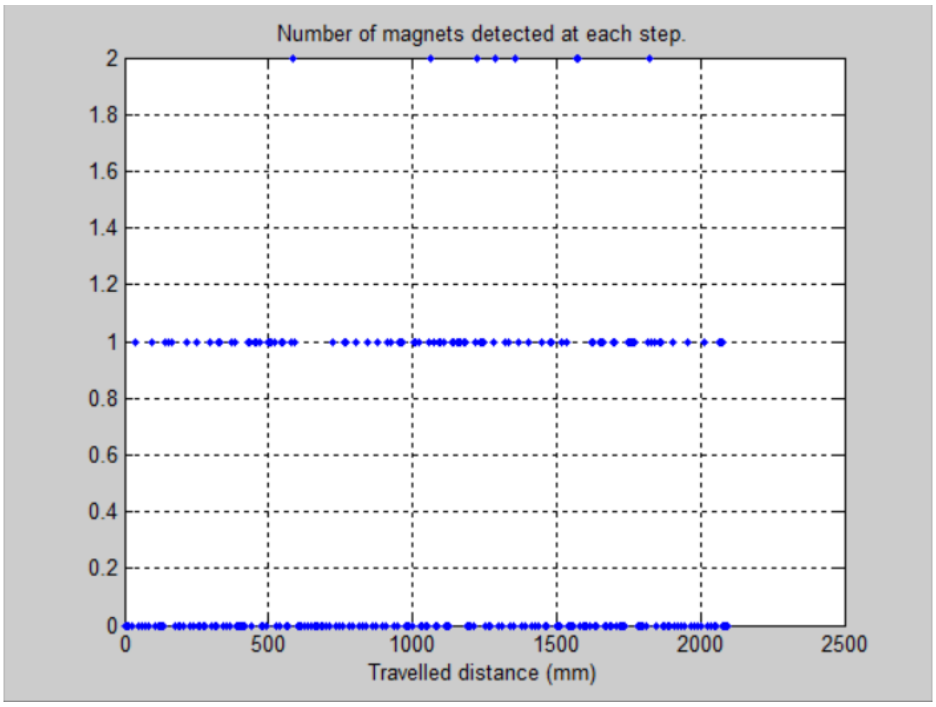

Figure 11: number of measurements per period.

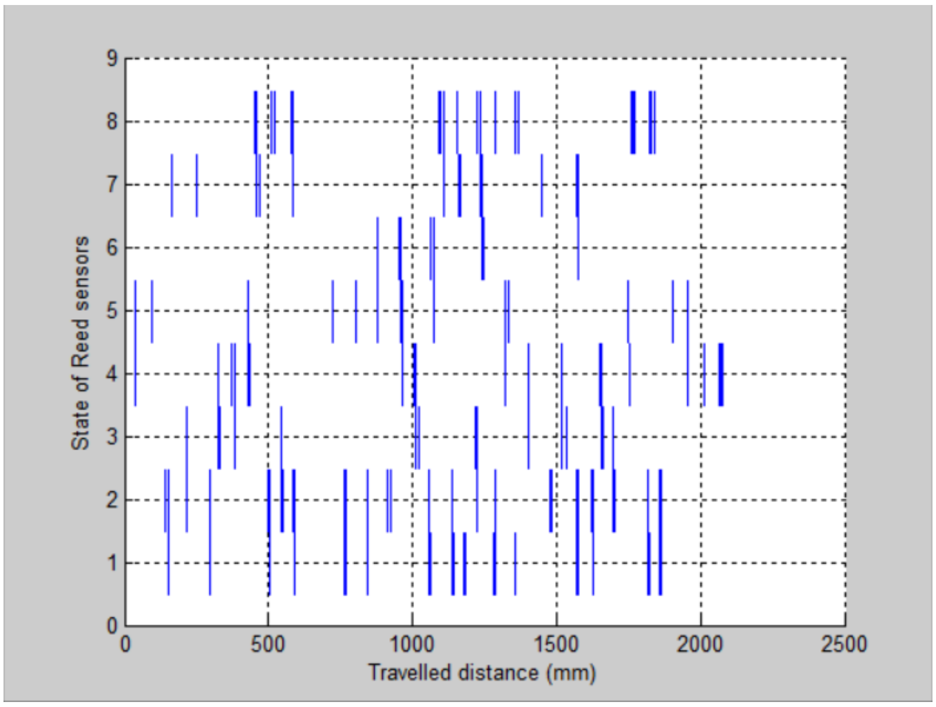

Figure 12: Raw sensor data

3. Lab 2 Exercises

Because of the X = [*0 0 0*pi/180 rwheel+0 rwheel+0*].' in the lab 2, we need to modify some matrix to apply the different.
At the first, we define the new *U*, *v* and *w* in the lab 2.

```
U = deltaq / period ;
v = 0.5*( X(4)*U(1) + X(5)*U(2) ) ;
w = X(4)/trackGauge*U(1) - X(5)/trackGauge*U(2) ;
```

Therefore, we are able to utilize the parameters *rwheel+0* which different from the lab1 in the matrix, for example, the matrix A and B

$$A=\begin{bmatrix} 1 & 0 & \text{-v*period*sin(X(3))} & \text{0.5*U(1)*period*cos(X(3))} & \text{0.5*U(2)*period*cos(X(3));} \\ 0 & 1 & \text{v*period*cos(X(3))} & \text{0.5*U(1)*period*sin(X(3))} & \text{0.5*U(2)*period*sin(X(3));} \\ 0 & 0 & 1 & \text{U(1)*period/trackGauge} & \text{-U(2)*period/trackGauge} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B = period * \begin{bmatrix} \text{X(4)*cos(X(3))/2} & \text{X(5)*cos(X(3))/2} \\ \text{X(4)*sin(X(3))/2} & \text{X(5)*sin(X(3))/2} \\ \text{X(4)/trackGauge} & \text{-X(5)/trackGauge} \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The equation of A and B is the same as which defined in the lab1, so we did not mention it again. And then, we update the equation in the evolution model which is similar to what we did in the lab1.

$$\hat{X}_{k+1/k} = f(\hat{X}_{k/k}, U_k) = \begin{bmatrix} \text{Xold(1)+v*period*cos(X(3))} \\ \text{Xold(2)+v*period*sin(X(3))} \\ \text{Xold(3)+w*period} \\ \text{Xold(4)} \\ \text{Xold(5)} \end{bmatrix}$$

Other parameters almost the same as the Lab1. And in the noise definition, we need to modify the *Qbeta* and *Qalpha*

$$Qbeta = Qwheels$$
$$Qalpha = zeros(5)$$

This is because that we have 5 parameters in X, so *Qalpha* also should be a matrix possess 5 dimensions. According to *Qbeta*, because:

$Qbeta = \begin{bmatrix} \Delta q_{r,k} \\ \Delta q_{l,k} \end{bmatrix} = \begin{bmatrix} \sigma_{wheel} & 1 \\ 1 & \sigma_{wheel} \end{bmatrix}$, So we are able to directly use $Qbeta = Qwheels$ in lab2 rather

than $Qbeta = \text{jointToCartesian} * Qwheels * \text{jointToCartesian}^{-1}$

According to the noise tuning, we set the *sigmaRR=1* at first. Then we tune the other noise parameters to a satisfactory value.

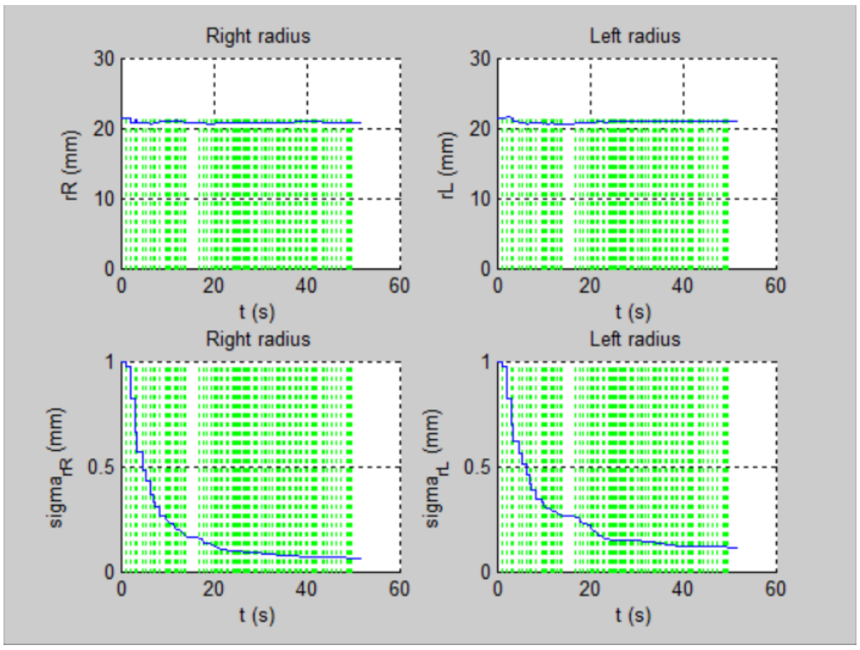

Figure 13: Variances

After finished tuning work, we define the *sigmaRR=0* which means we have already optimized the noise in the *rwheel* and tune the other noise parameters again. In conclusion, those parameters eventually became nearly as the same as what we got in the lab1.
Then, we are able to get the graphs and those graphs are almost similar to the graphs which we got in the lab1, for example, the following graphs are about one loop movement.

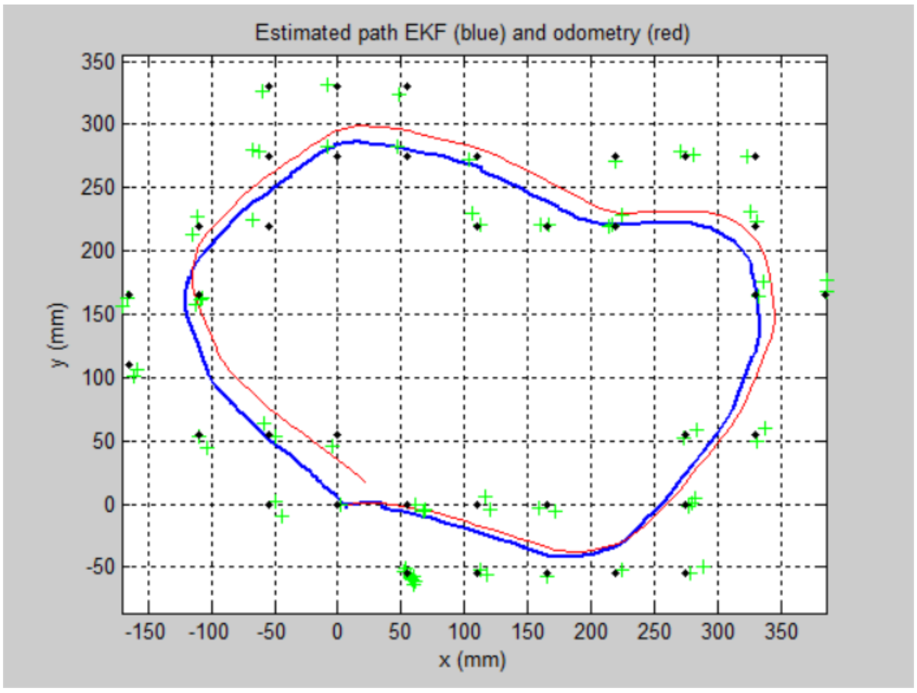

Figure 14: Paths

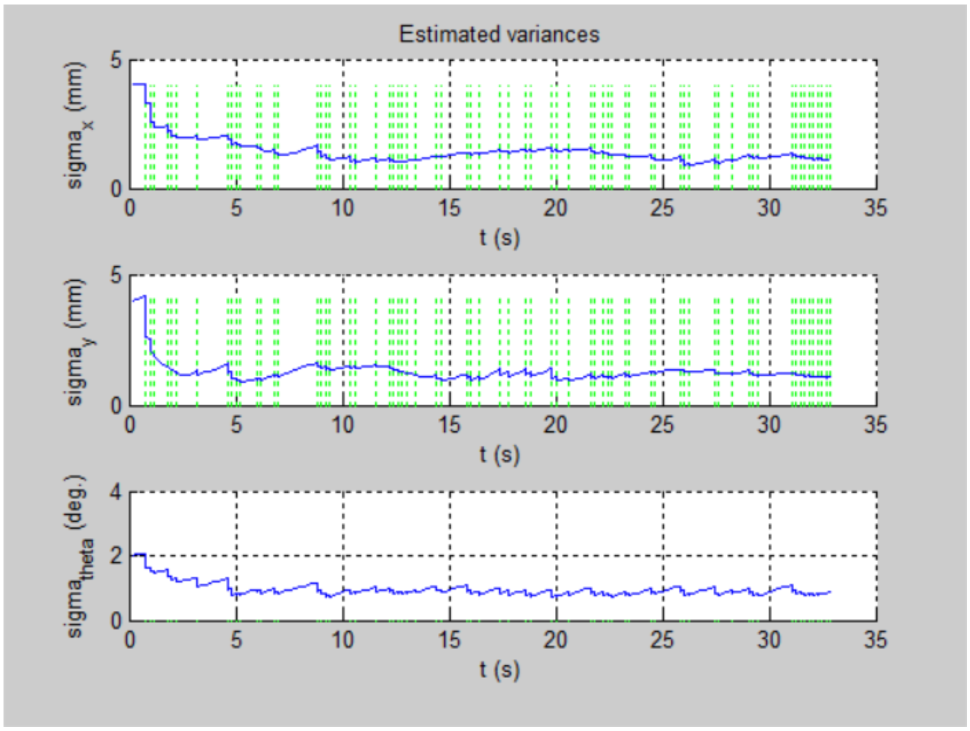

Figure 15: Variances

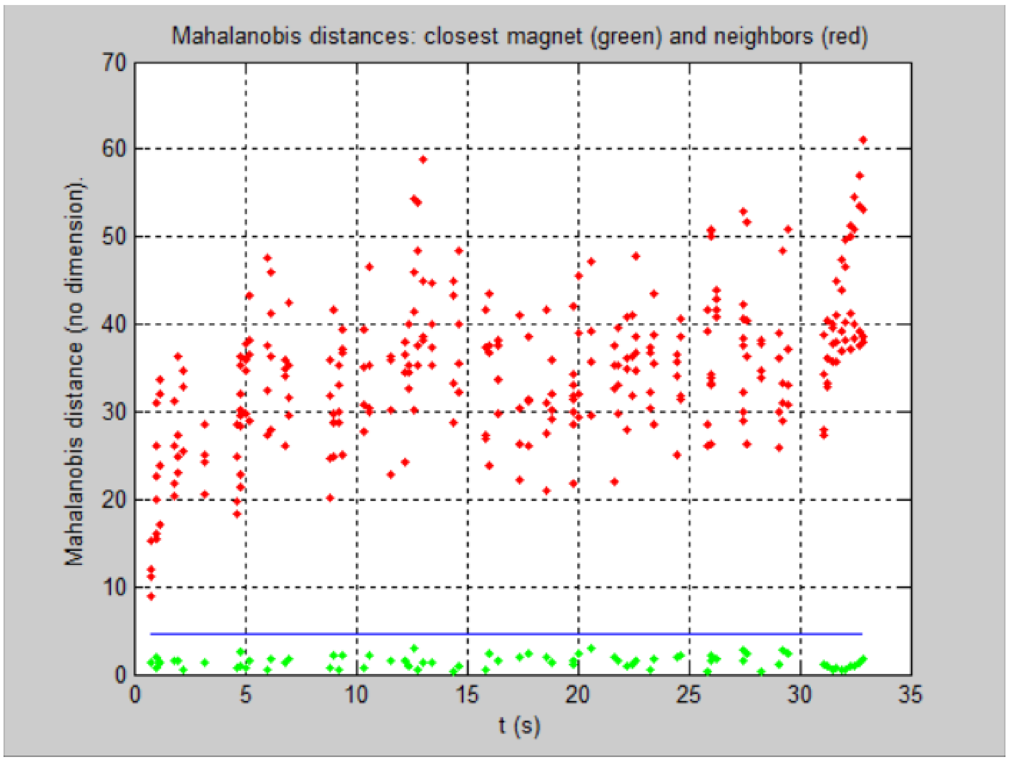

Figure 15: Variances

Obviously, the lab2's graphs are up to the standards and succeeded in tuning noise. Therefore, we assume that our lab2 MATLAB code accessed our goal.

4. Additional lab mission

In the additional lab mission, we will only use the measurement in y coordinate to implement localization work. Undoubtedly, the location strategy which only utilizes the measurement value to locate the robot's position will occur the mistake when the robot's trajectory is a horizon line. Because if we only use the measurement value Y, we are impossible to know the magnets' x coordinate value. In other words, we do not know the precise position of the closest magnet's real position. This is because that we will mix this magnet with those magnets which has the same y coordinate value and different x coordinate. In case of the 45 degree trajectory. Because that each magnet has different x coordinate value. So this strategy will not cause any problem.

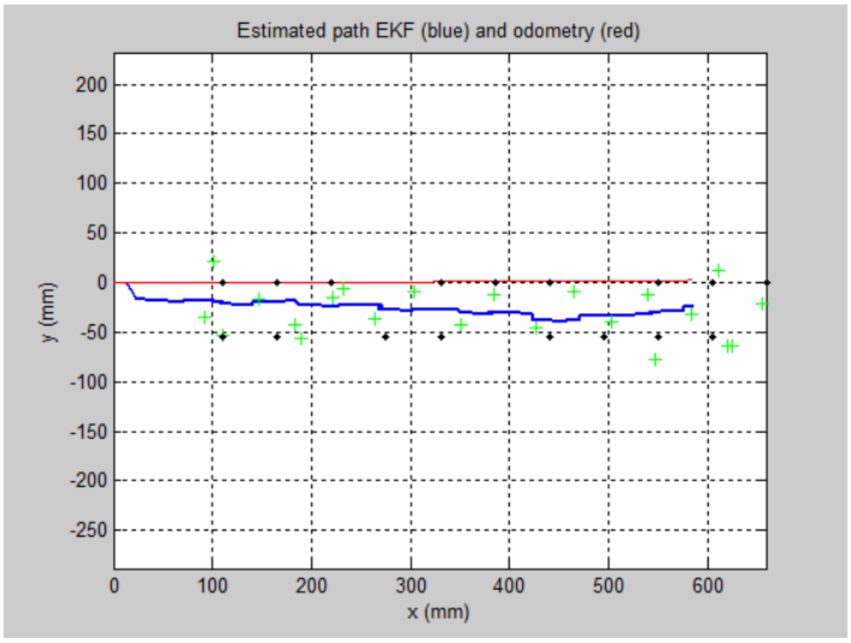

Figure 16: Estimated path (trial 1)

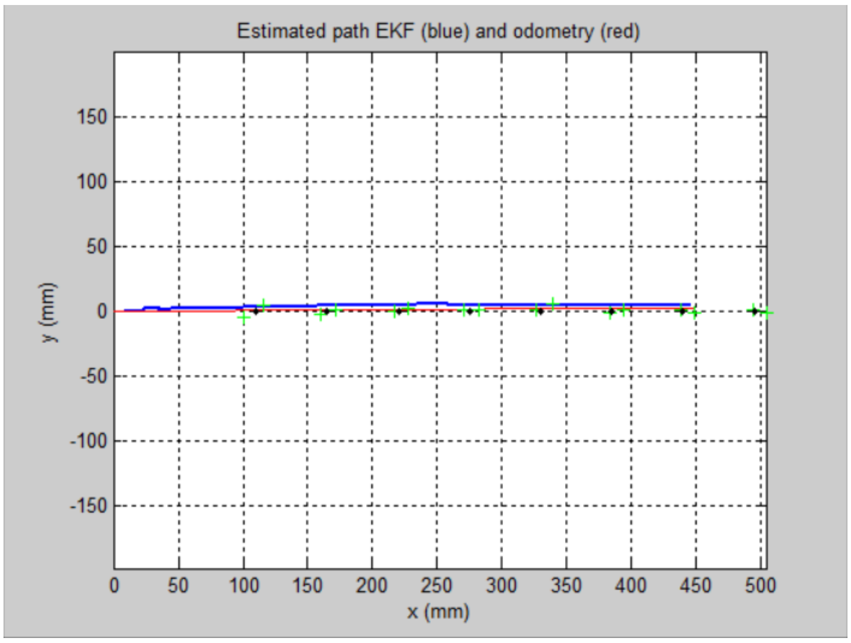

Figure 16: Estimated path(trial 2)

However, if we use the line1 or line2 which is the horizon line. This strategy will make mistake because all magnets surround trajectory have similar x coordinate value and robot can't recognize the which one is the closest magnet.