École Centrale de Nantes
EMARO-CORO M1

MOBRO

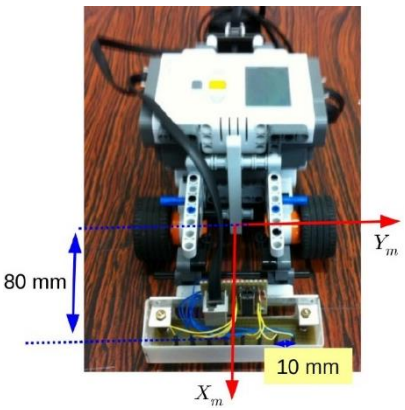
Lab 1-2

Students:
Bergantin Lucia
Tartari Michele

Introduction

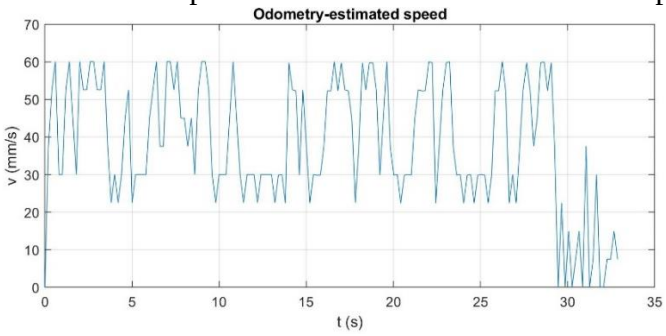
The aim of this lab was to apply the localization concept previously analyzed in class and to go in depth in the analysis of noise and its choice of values.

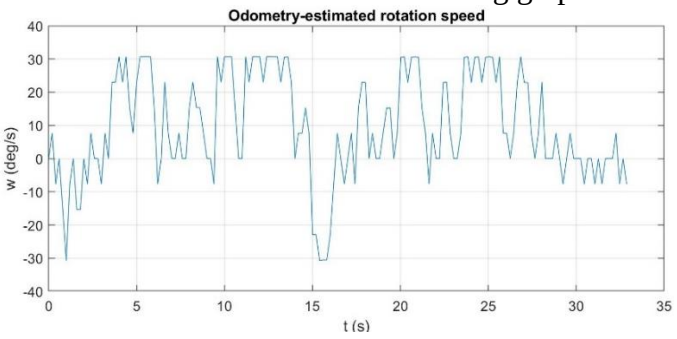
To do so we worked with the data retrieved from a real robot, which was equipped with a set of magnetic sensors to enable it to perform a localization in a certain area by the means of the Extended Kalman filter (EKF). This is preferable to the non-extended version because we handle noise.

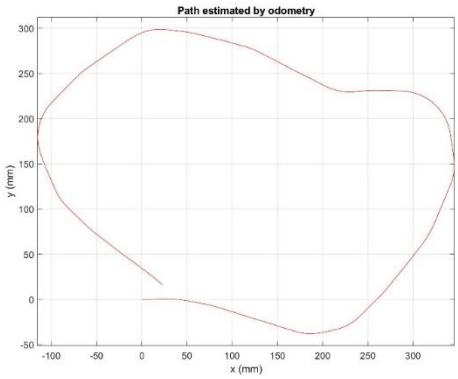


Running "ShowOdometry.m"

The Matlab script "ShowOdometry.m" shows the path estimated by odometry and can be applied to all the different cases presented. To test it we used 'oneloop.txt' data file and obtained the following graphs:







The plots show the estimated twist, dividing it into the linear and the rotation speed components. The third plot shows the estimated path travelled by the robot, calculated by the means of the evolution model.

Part 1: single tuning variable

Modification of "EvolutionModel.m" and "MagnetLoc.m"

In the "EvolutionModel.m" script we coded a new function to calculate a new state vector and an input vector

$$Xnew = [x \quad y \quad \theta]^T$$
$$U = [\Delta q_r \quad \Delta q_l]^T$$

from which we extrapolated the evolution model using the odometry equations shown below:

$$\begin{cases} \Delta D_k = (r_r\, \Delta q_{r,k} + r_l\, \Delta q_{l,k})/2 \\ \Delta \theta_k = (r_r\, \Delta q_{r,k} - r_l\, \Delta q_{l,k})/e \\ \theta_{k+1} = \theta_k + \Delta\, \theta_k \\ x_{k+1} = x_k + \Delta D_k cos(\theta_k) \\ y_{k+1} = y_k + \Delta D_k sin(\theta_k) \end{cases}$$

where e is the track gauge. We used the simplified version of the equations due to the high amount of noise, that would make the extra computational effort useless.

```
function Xnew=EvolutionModel(Xold,U)

Xnew=zeros(3,1);
    Xnew(1)=Xold(1)+U(1)*cos(Xold(3));
    Xnew(2)=Xold(2)+U(1)*sin(Xold(3));
    Xnew(3)=Xold(3)+U(2);
end
```

EvolutionModel1.m	EvolutionModel.p
X =	X =
22.8971	22.8971
16.2193	16.2193
5.5483	5.5483

Comparing our code with the precompiled file, we obtained the same results.

In the "MagnetLoc.m" script we implemented the matrices A, B, C and ${}^{0}T_{m}$.

```
% Calculate linear approximation of the system equation
```

$$A = \frac{\partial f}{\partial X} \qquad B = \frac{\partial f}{\partial U} \qquad C = \frac{\partial g}{\partial X}$$

```
A=[1,0,-U(1)*sin(X(3));
   0,1, U(1)*cos(X(3));
        0,0,1];
B=[cos(X(3)),0;
   sin(X(3)),0;
          0,1];
[...]
C=[-cos(X(3)),-sin(X(3)),-sin(X(3))*(oPmagnet(1)-X(1))+cos(X(3))*(oPmagnet(2)-X(2);
    sin(X(3)),-cos(X(3)),-sin(X(3))*(oPmagnet(2)-X(2))-cos(X(3))*(oPmagnet(1)-X(1))];
```

```
% Calculate homogeneous transform of the robot with respect to the world frame
```

$${}^{0}T_{m} = \begin{bmatrix} cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix}$$

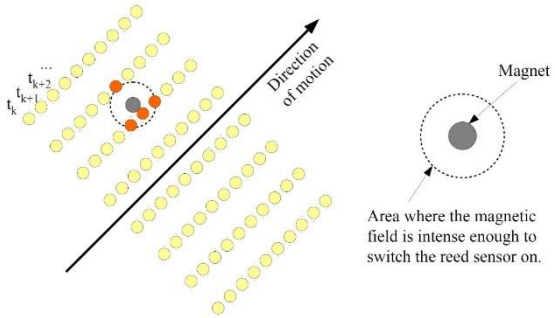
```
oTm = [cos(X(3)),-sin(X(3)),X(1);
       sin(X(3)),cos(X(3)),X(2);
                         0,0,1] ;
mTo = inv(oTm) ;
```

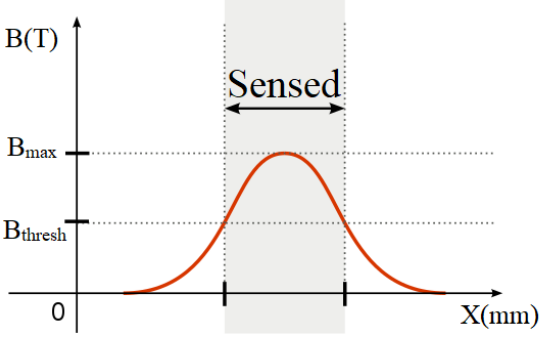
Evaluating the measurement noise

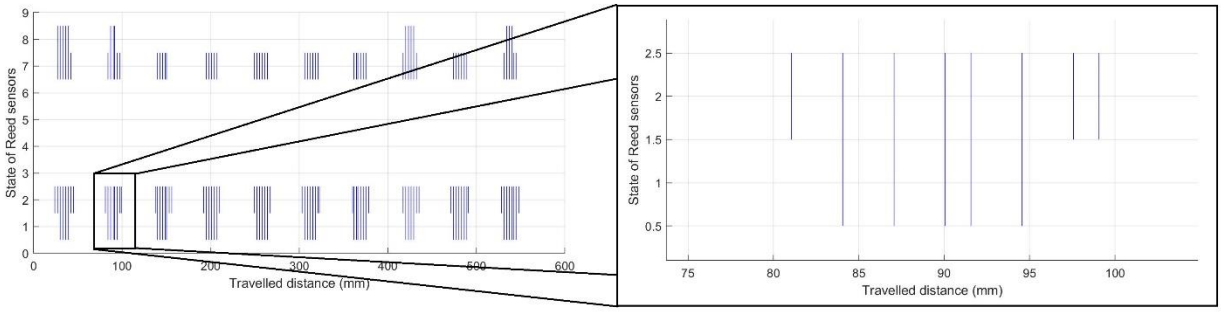
As for the uncertainty variances values, we considered the average human precision while performing the task of placing the robot in the starting pose. In our opinion a tolerance of 2mm could be plausible along x and y axis, while 2°deg could be acceptable for the initial rotation.

Differently from the ideal case (that considers a single infinite linear sensor), here the magnetic beams are detected in an area much larger than a single point as the reed switches are triggered by any magnetic field above a certain threshold. Consequently, the detection of the magnet by one or more sensor is strictly related to the distance between the magnet and each switch. Thus, the distribution has to be considered uniform.

Along the robot y axis, the position of the magnet could be estimated with a precision of 10mm (equal to the distance between two reed switches). As for the other axis, we can estimate for how long it is possible to detect the magnet along x (direction of motion in the robot frame) by running the simulation with the maximum sensor frequency and by looking at the graph below (plotted using *PlotResults.m*): the magnet was detected for a travel distance of roughly 20mm.







Setting the Mahalanobis distance threshold and tuning

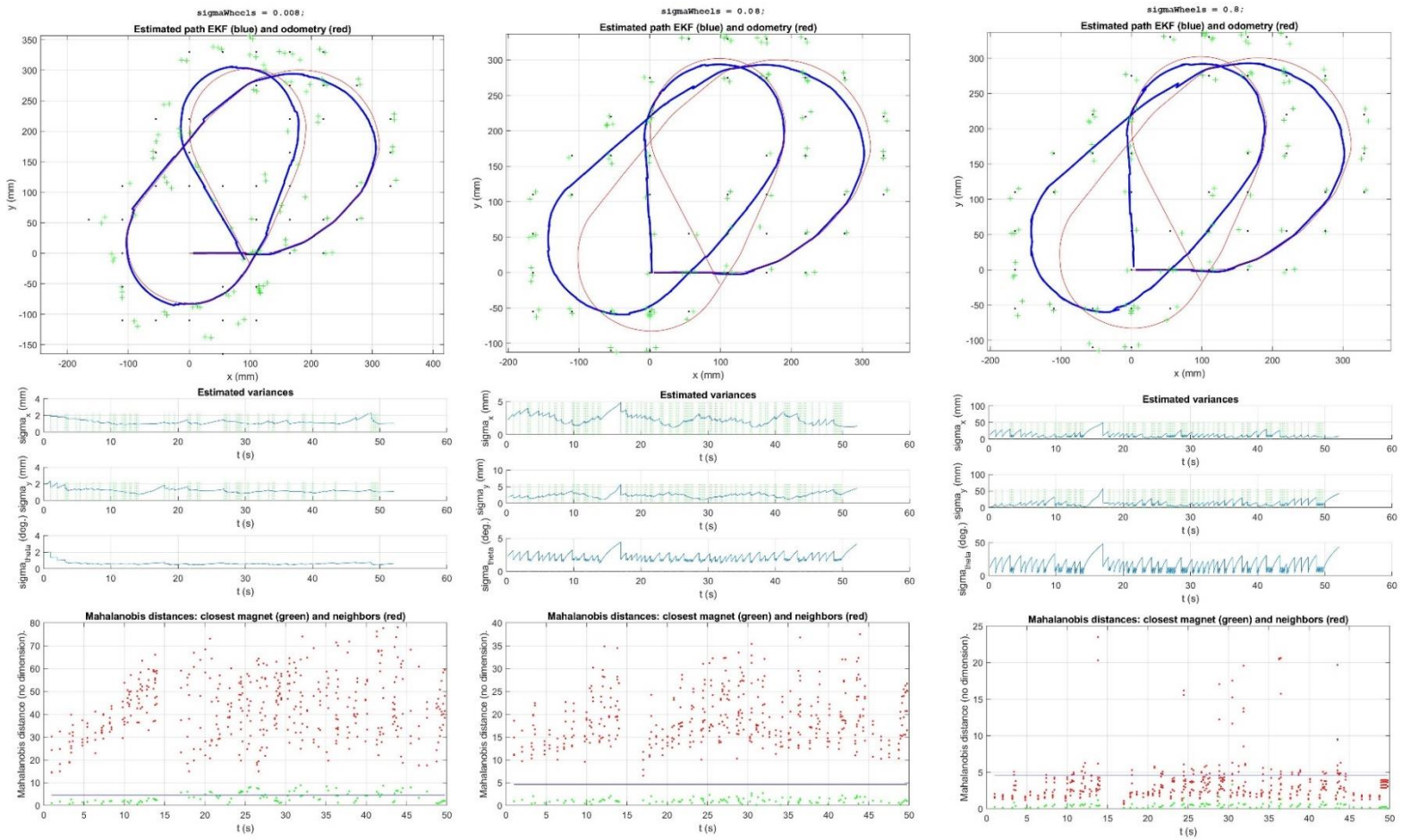
We finally set the Mahalanobis distance threshold. To preform that task we used the MATLAB function X = *chi2inv(P, V)*, where P is the probability that a Mahalanobis distance between Y and $\hat{Y}$ is below a certain threshold, while V is the number of d.o.f. of the robot. We opted for $P=0.9$, as it would imply that, assuming a gaussian noise, 10% of the data would be over the threshold. Note that these results differ from reality, since in this application the noises are not gaussian.

Tuning

To calibrate the filter, we used a single tuning parameter called *sigmaWheels*, that sums up all the variances of the non-ideal aspects of the robot and of the environment, such as non-thin wheels, radii variation, skidding, slipping, unperfect surface, etc. To set its value it's fundamental to remember that errors should fall in an interval of +/-2**sigmaWheels*, therefore we performed some test.

If *sigmaWheels* is too low (e.g. *sigmaWheels=0.008*) we have low values of variances, but in the EKF localization the starting point does not coincide with the arriving one and several of the Mahalanobis distances of correct detections are above the threshold and thus discharged.

If *sigmaWheels* is too high (e.g. *sigmaWheels=0.8*) we have quasi-matching start/end poses, but variances are extremely high and many Mahalanobis distances of incorrect detections are below the threshold.



For this lab we set *sigmaWheels=0.08*, since it gave a balance between a close match of start/end poses and low variances (roughly within 5mm and 5°deg tolerances, even for long periods without detections). The threshold is also to be considered satisfying, since it lies exactly in-between Mahalanobis distances of correct and incorrect detections.
We also verified the value with other data sets, as each one focuses on a different aspect of the motion and has a different tolerance on its variation.

Part 2: the introduction of wheel radius estimation

Evolution model

In this second part, we considered a state vector *Xnew* including also the radii of the right and left wheel, setting a difference of 14% between the two wheels as an initial guess. The input vector *U* contains the joint derivatives that coincide with the angular speed of the wheels.

$$Xnew = [x \quad y \quad \theta \quad r_r \quad r_l]^T$$
$$U = [\dot{q_r} \quad \dot{q_l}]^T$$

We extrapolated the evolution model using the simplified odometry model shown below:

$$\begin{cases} \Delta D_k = (r_r\,\Delta q_{r,k} + r_l\,\Delta q_{l,k})/2 \\ \Delta\theta_k = (r_r\,\Delta q_{r,k} - r_l\,\Delta q_{l,k})/e \\ \theta_{k+1} = \theta_k + \Delta\,\theta_k \\ x_{k+1} = x_k + \Delta D_k cos(\theta_k) \\ y_{k+1} = y_k + \Delta D_k sin(\theta_k) \\ r_{r,k+1} = r_r \\ r_{l,k+1} = r_l \end{cases}$$

```
function Xnew=EvolutionModel(Xold,U)
global trackGauge period;
dD=(Xold(4)*U(1)+Xold(5)*U(2))/2*period;
dT=(Xold(4)*U(1)-Xold(5)*U(2))/trackGauge*period;

Xnew=zeros(5,1);
Xnew(1)=Xold(1)+dD*cos(Xold(3));
Xnew(2)=Xold(2)+dD*sin(Xold(3));
Xnew(3)=Xold(3)+dT;
Xnew(4)=Xold(4);
Xnew(5)=Xold(5);
end
```

In the "MagnetLocWithRadius.m" script we calculated the matrices *A*, *B* and *C*. Note that to obtain the values of q_r and q_l for the computation of the model we had to multiply for the period, since U is a speed vector.

$$A = \frac{\partial f}{\partial X}$$
$$B = \frac{\partial f}{\partial U}$$

```
A=[1,0,-(X(4)*U(1)*period+X(5)*U(2)*period)/2*sin(X(3)), U(1)/2*cos(X(3))*period, U(2)/2*cos(X(3))*period;
   0,1, (X(4)*U(1)*period+X(5)*U(2)*period)/2*cos(X(3)), U(1)/2*sin(X(3))*period, U(2)/2*sin(X(3))*period;
   0,0,                                               1,          U(1)/e*period,         -U(2)/e*period;
   0,0,                                               0,                      1,                      0;
   0,0,                                               0,                      0,                      1];

B= period*[ X(4)/2*cos(X(3)),   X(5)/2*cos(X(3)) ;
            X(4)/2*sin(X(3)),   X(5)/2*sin(X(3)) ;
                     X(4)/e,           -X(5)/e ;
                          0,                 0 ;
                          0,                 0];
```

The measurement equation doesn't change in comparison with the first part, since the ${}^{est}P_e$ matrix only depends on the sensor which remains the same. Thus, matrix C is similar to the previous one if not for two columns of 0s, corresponding to the radii columns.

$$C = \frac{\partial g}{\partial X}$$

```
C = [ -cos(X(3)),    -sin(X(3)),  -sin(X(3))*(oPmagnet(1)-X(1))+cos(X(3))*(oPmagnet(2)-X(2)), 0,  0 ;
       sin(X(3)),    -cos(X(3)),  -sin(X(3))*(oPmagnet(2)-X(2))-cos(X(3))*(oPmagnet(1)-X(1)), 0,  0];
```

Also note that the ${}^0T_m$ matrix does not change.

```
P =

    0.6507    0.0545   -0.0063         0         0
    0.0545    0.5441    0.0059         0         0
   -0.0063    0.0059    0.0002         0         0
         0         0         0         0         0
         0         0         0         0         0

P =

    0.6899    0.0393   -0.0067         0         0
    0.0393    0.5818    0.0064         0         0
   -0.0067    0.0064    0.0002         0         0
         0         0         0         0         0
         0         0         0         0         0

P =

    0.7149    0.0377   -0.0069         0         0
    0.0377    0.6063    0.0067         0         0
   -0.0069    0.0067    0.0002         0         0
         0         0         0         0         0
         0         0         0         0         0
```

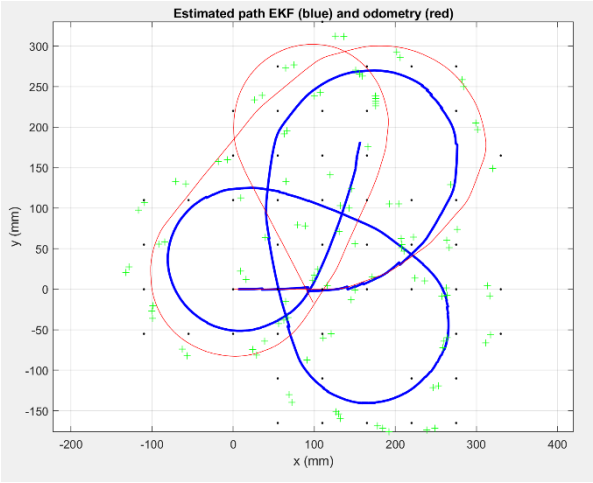
In the new "DefineVariances.m" script we first set the wheels variances (*sigmaRR* and *sigmaRL*) and the state noise matrix *Qalpha* to zero, while we set *sigmaWheels* to a non-zero value. Executing the program, we observed that the last two columns and rows of the matrix P were all 0s. This means that with these settings we know exactly the radius of each wheel, as in the first part. However, the two radii don't necessarily coincide.

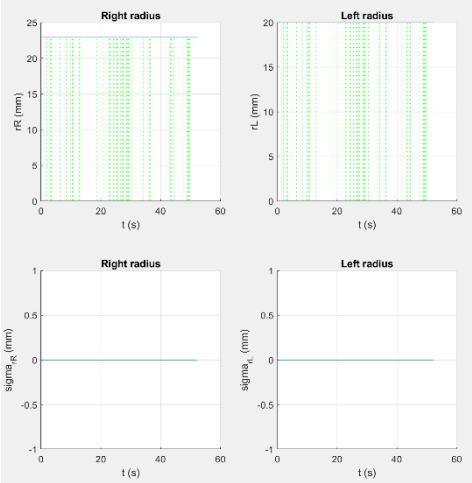
Aside, we see an example of what explained above, obtained by the execution of the program with the described settings using the oneloop.m data set. This happens for all the given data sets.

We then tuned the noises using a methodology similar to the first part as for the values of *sigmax*, *sigmay*, *sigmatheta*, *sigmameasurementX* and *sigmameasurementY*. As for *sigmaRR* and *sigmaRL* we accepted a wheel tolerance on the initial guess of 2mm.

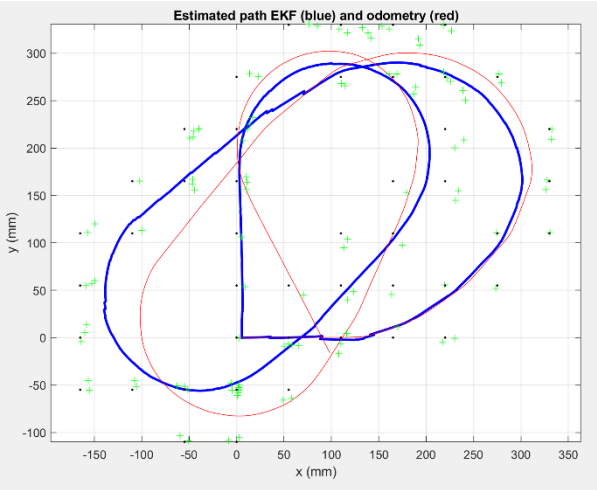
```
% Uncertainty on initial position of the robot.
sigmaX     = 2 ;              % Determined by student
sigmaY     = 2 ;              % Determined by student
sigmaTheta = 2 *pi/180;       % Determined by student
sigmaRR    = 2 ;              % 2mm tollerance from real value
sigmaRL    = sigmaRR ;
Pinit = diag( [ sigmaX^2 sigmaY^2 sigmaTheta^2 ...
                sigmaRR^2 sigmaRL^2 ] ) ;

% Measurement noise.
sigmaXmeasurement = 5.7735 ;  % approximate to gaussian noise using uniform ditribution
                              % 20/sqrt(12), uniform distribution
sigmaYmeasurement = 2.8868 ;  % 10/sqrt(12), uniform distribution
Qgamma = diag( [sigmaXmeasurement^2 sigmaYmeasurement^2] ) ;
```

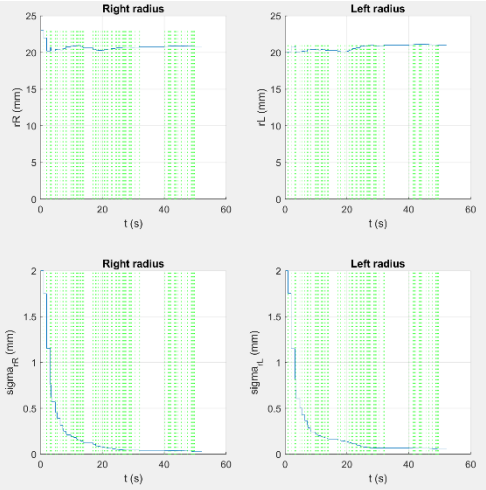
Executing the program, we checked the convergences of all the different noises. It's particularly important to check the behavior of the radii and their variances, to analyze how this affects the localization. We temporarily set the radii variances to zero and execute the program using the twoloops data set. What obtained is shown in the figures below. Note that the EFK localization is extremely inaccurate, while the radii and their variances remain always constant.





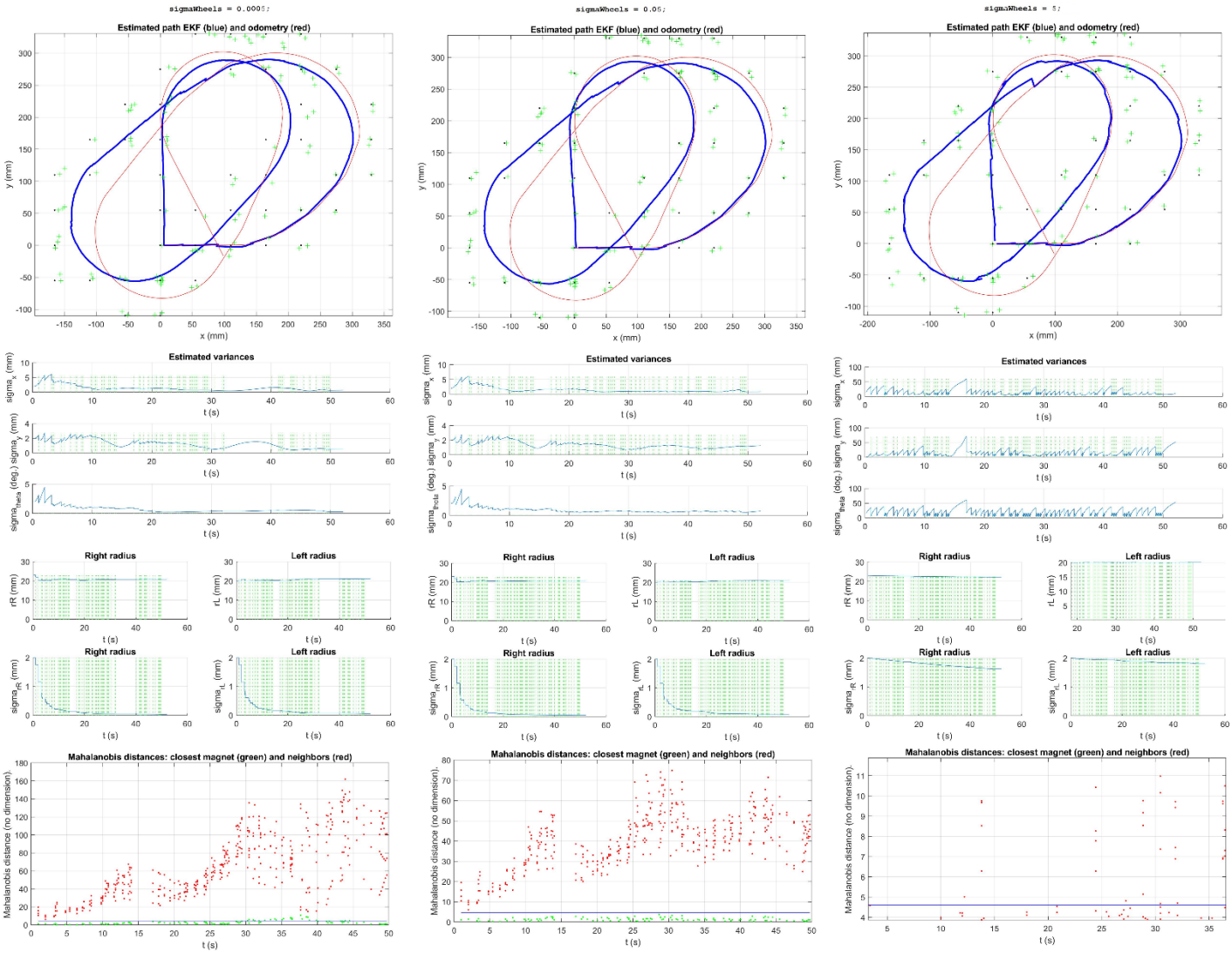
Setting the value to 2 instead, we saw that the radii then tend to converge to a value very close to their nominal one, while the variances tend to zero (never reaching it due to a too short execution time). The figures below are obtained using the new settings on the twoloops data set: the difference is evident as the localization quality is definitely increased.





Tuning

As in the previous part we used a single tuning parameter called *sigmaWheels*, that sums up all other variances of the non-ideal aspects of the environment, such as non-thin wheels, skidding, slipping, unperfect surface, etc.
If *sigmaWheels* is too low (e.g. *sigmaWheels=0.0005*) we have low values of variances (for both pose components and radii) but several of the Mahalanobis distances of correct detections are above the threshold and thus discharged.
If *sigmaWheels* is too high (e.g. *sigmaWheels=5*) the variances are extremely higher and many Mahalanobis distances of incorrect detections are below the threshold.
For this lab we set *sigmaWheels=0.05* since it gave low variances (roughly within 5mm and 5deg. tolerance, even for long period without detections). The threshold is satisfying, since it lies exactly in-between Mahalanobis distances of correct and incorrect detections.



Comparison

As we can see, the *sigmaWheels* value is smaller than the one set in the first part. This is a direct consequence of the isolation of the noise component due to the radii, that is therefore not included anymore in the *sigmaWheels* value. This has of course a positive effect on the localization since it reduces the sources of uncertainty compensated by *sigmaWheels*.

As for the tuning of both cases, note that we reported only one example, but we had to execute the program on all the data sets and verify the correctness of the results obtained. Each data set has a different tolerance on the variation of the *sigmaWheels* value, still in general we saw that in the second part the tolerance was higher, since *sigmaWheels* itself had less influence on the result due to the isolation of the radii variance problem.