Human-centered Assistive Robotics Technische Universität München Prof. Dr.-Ing. Dongheui Lee	**MACHINE LEARNING IN ROBOTICS** Assignment 1

Exercise 1: *Estimating velocity motion model of a mobile robot through linear regression*

Background

In this exercise you will write a matlab program for estimating the pose x, y and θ (position and orientation) of a mobile robot from control inputs v and w (velocity and angular velocity) through black box modeling. Black box modeling is primarily useful when the aim is to fit the data regardless of particular mathematical structure of the model. The control inputs are usually applied as velocities to each wheel v_l and v_r. These can be transformed as resultant velocity $v = \frac{v_r+v_l}{2}$ and angular velocity $w = \frac{v_r-v_l}{D}$ where D is the distance in between two wheels. θ is measured from x-axis and a counter clockwise rotation of mobile robot correspond to positive w.

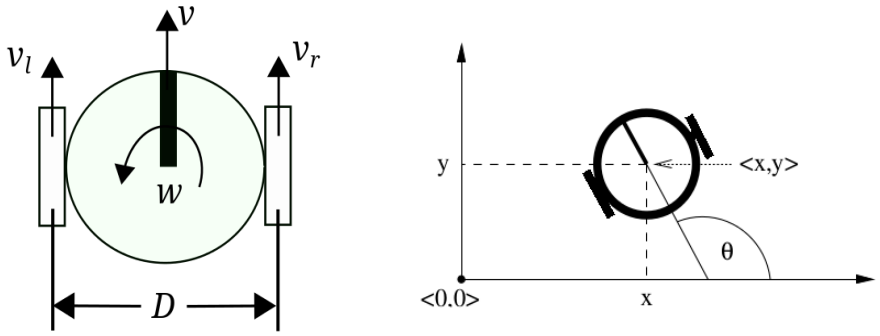


Figure 1: A simplified diagram of a mobile robot

The response of the control input is recorded with respect to a body frame of reference such that the pose is $(0, 0, 0)$ in body frame before applying control input. Now the regression doesn't have to learn the global transformation of state variables, instead a local change of pose can be learned which can then be transformed into global pose by applying appropriate rotation and transformation.

Task

A dataset is provided as a matlab file *Data.mat*. After loading this file you will get two matrices Input and Output. Input has a dimension of 2xn and Output has a dimension of 3xn. Input contains the control inputs while the corresponding columns in Output contains the change of state w.r.t body frame for these inputs.

$$\text{Input} = \begin{pmatrix} v^{(1)} & v^{(2)} & \dots & v^{(n)} \\ w^{(1)} & w^{(2)} & \dots & w^{(n)} \end{pmatrix}$$

$$\text{Output} = \begin{pmatrix} \Delta x^{(1)} & \Delta x^{(2)} & \dots & \Delta x^{(n)} \\ \Delta y^{(1)} & \Delta y^{(2)} & \dots & \Delta y^{(n)} \\ \Delta \theta^{(1)} & \Delta \theta^{(2)} & \dots & \Delta \theta^{(n)} \end{pmatrix}$$

Since the sensors were not perfect, each column in Output is corrupted by zero mean Gaussian noise $\mathcal{N}(0, \Sigma)$. Now you will apply linear regression to learn Input, Output mapping as mentioned in equation (1).

$$
\begin{aligned}
\Delta x &= a_{11} + \sum_{p=1}^{p1} \left(a_{1\ 2+3(p-1)} v^p + a_{1\ 3+3(p-1)} w^p + a_{1\ 4+3(p-1)} (vw)^p \right) \\
\Delta y &= a_{21} + \sum_{p=1}^{p1} \left(a_{2\ 2+3(p-1)} v^p + a_{2\ 3+3(p-1)} w^p + a_{2\ 4+3(p-1)} (vw)^p \right) \qquad (1) \\
\Delta \theta &= a_{31} + \sum_{p=1}^{p2} \left(a_{3\ 2+3(p-1)} v^p + a_{3\ 3+3(p-1)} w^p + a_{3\ 4+3(p-1)} (vw)^p \right)
\end{aligned}
$$

Since increasing the order of polynomial will always give better prediction results you will apply cross validation to avoid over fitting. Among different techniques for cross validation, you have to implement k-fold cross validation. In k-fold cross validation the data is randomly divided into k equal sized subsamples. Training is performed k times (folds) where in each fold (k-1) subsamples are used for training and the remaining subsample is used for testing. Since every observation is passed through the testing phase, the overall estimate of error can be combined to get a single estimate of error for the given model complexity (polynomial order / free parameters). The model complexity which result in lowest error is selected. At the end the model parameters are re-estimated for the selected model complexity by using the entire dataset.

$$
\text{Position error} = \frac{\sum_{i=1}^{n} \left(\left(\Delta x^{(i)} - \Delta x_{pred}^{(i)} \right)^2 + \left(\Delta y^{(i)} - \Delta y_{pred}^{(i)} \right)^2 \right)^{\frac{1}{2}}}{n} \qquad (2)
$$

$$
\text{Orientation error} = \frac{\sum_{i=1}^{n} \left(\left(\Delta \theta^{(i)} - \Delta \theta_{pred}^{(i)} \right)^2 \right)^{\frac{1}{2}}}{n} \qquad (3)
$$

a) Now apply k-fold cross validation with k=5 and report the optimal values for $p1$ and $p2$ by varying them from $1 \rightarrow 6$. Also provide the learned parameter values. Since your data has already been shuffled, you don't have to worry about random partition for cross validation (for $K = 1 \rightarrow$k use $1 + (K-1) \times \frac{n}{\mathsf{k}} : K \times \frac{n}{\mathsf{k}}$ for generating k subsamples).

b) Store the parameters values learned in a) as a cell array *'par'* of size 1x3. Each cell *par*$\{i\}$ contains the learned parameters values $a_{i1} a_{i2} \ldots a_{im_i}$ (as column vectors).

c) Save this cell array by matlab command *save('params','par')*. Now run the provided matlab function *Simulate_robot* for the (v, w) values of $(0, 0.05), (1, 0), (1, 0.05)$ and $(-1, -0.05)$ to get a visualization of learned dynamics.

If you have learned the model parameters correctly then the plot by *Simulate_robot* for (0.5,-0.03) should be as in Figure 2. Also for the sake of verification, the optimal values for k=4 are $p1 = 4$ and $p2 = 1$.

Note: In your code combine position error as mentioned in Equation (2) to get a single estimate of polynomial order $p1$ in Equation (1). By using orientation error as mentioned in Equation (3), get a separate estimate for polynomial order $p2$ in Equation (1).

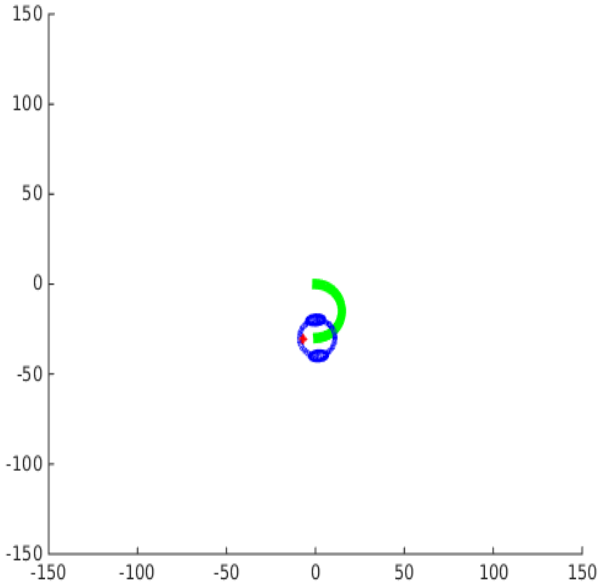


Figure 2: Robot trajectory simulation using the learned model parameters

Solution 1: *Estimating velocity motion model of a mobile robot through linear regression*

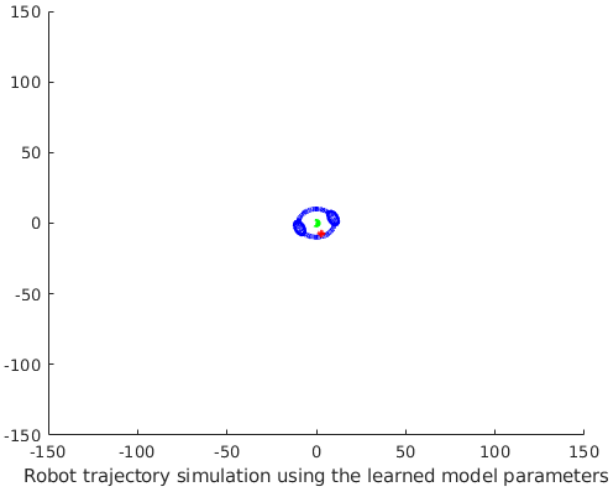
a) In attached code (main.m), check the divide cell with comment "Calculate error with varying p and different folds"

b) If you run the attached code (main.m), the value of 'par' would be generated as following FIgure 3.

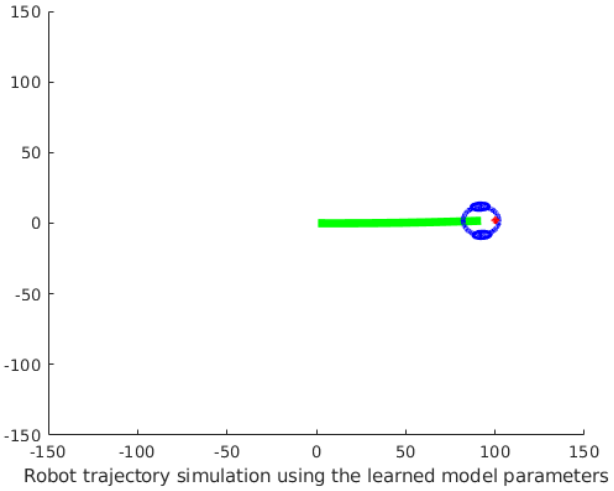
	par{1, 1}		par{1, 2}		par{1, 3}
	1		1		1
1	0.0025	1	-0.0043	1	8.0784e-04
2	0.9198	2	-0.0010	2	-3.1902e-04
3	-0.0029	3	0.0014	3	0.9987
4	-7.4385e-04	4	0.4680	4	3.2142e-04
5	-0.0010	5	5.6850e-...		
6	0.0014	6	-0.0025		
7	0.0025	7	-0.0010		
8	1.3601e-04	8	1.9246e-...		
9	-2.6908e-04	9	-0.0017		
10	6.6926e-05	10	-6.7254e...		
11	1.3061e-05	11	-7.8462e...		
12	-0.0043	12	0.0035		
13	-4.5174e-05	13	8.7155e-...		

Figure 3: Value of Learned Parameters.

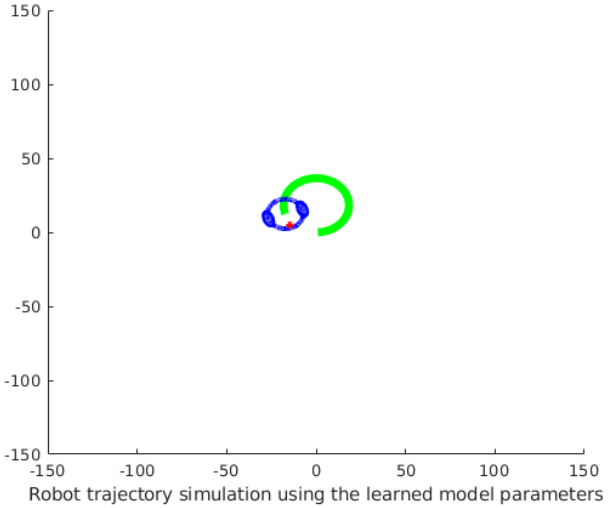
c) Below images show the result of Simulate_robot for different (v, w) values.



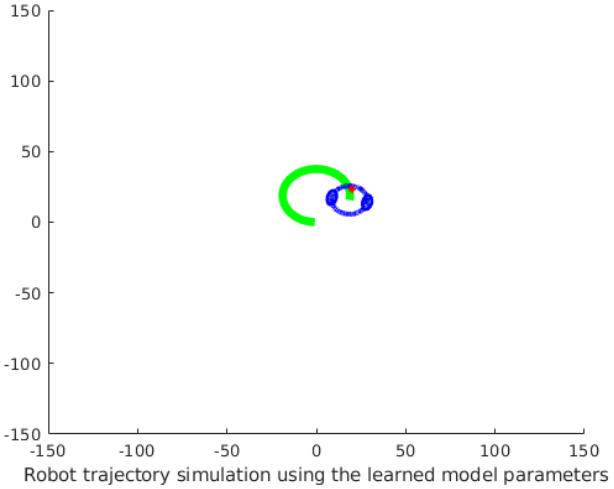
(a) $(v, w) = (0, 0.05)$



(b) $(v, w) = (1, 0)$



(c) $(v, w) = (1, 0.05)$



(d) $(v, w) = (-1, -0.05)$