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.

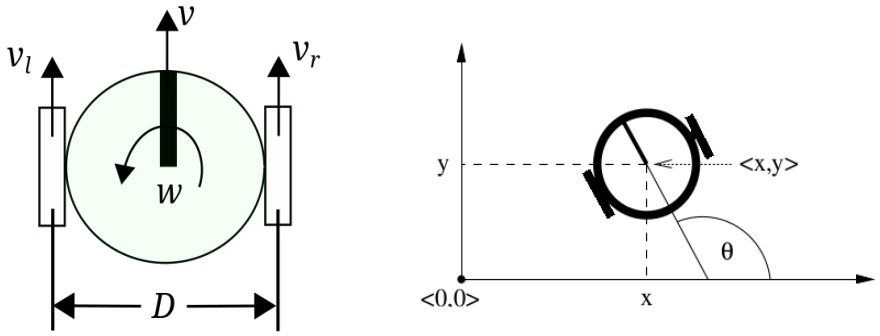


Figure 1: A simplified diagram of a mobile robot

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.

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

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

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 (1).

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

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.

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

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

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 $1 + (K-1) \times \frac{n}{\text{k}} : K \times \frac{n}{\text{k}}$ for generating k subsamples).

b) Store the parameters values learned in b) 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) sholud 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 (1). By using orientation error as mentioned in Equation 3, get a separate estimate for polynomial order $p2$ in (1).

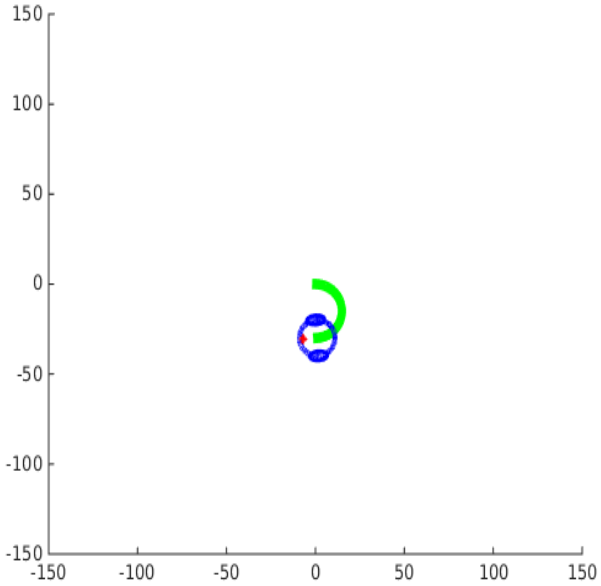


Figure 2: Robot trajectory simulation using the learned model parameters

Exercise 2: *Handwritten digits classification using Baysian classifier*

MNIST database contains images of handwritten digits. It has a training set of $60,000$ examples and a test set of $10,000$ examples. The digits have been centered and size-normalized to 28×28 pixels. The files "train-images.idx3-ubyte" and "train-labels.idx1-ubyte" contain the training images and the corresponding labels. The labels are from 0 to 9 which are the identity of the images. The files "t10k-images.idx3-ubyte" and "t10k-labels.idx1-ubyte" contain the test images and their corresponding labels, which you will use for the evaluation of the learned classifier. Load the data using the provided functions $loadMNISTImages$ and $loadMNISTLabels$.

$images = loadMNISTImages('train-images-idx3-ubyte');$
$labels = loadMNISTLabels('train-labels-idx1-ubyte');$

Reading the training images will yield a matrix of size $784 \times 60,000$ where $60,000$ is the number of images while reading the labels file will produce a vector of size $60,000 \times 1$. The i^{th} image in the dataset can be easily visualized by $imshow(reshape(images(:,i),28,28))$. Now you will model the distribution of each digit using a multivariate Gaussian distribution.

Since the data is already in high dimensional space, you will first project the data into a lower dimensional space, to avoid singularity problem assosiated with density estimation in high dimensional space. You will use

Figure 3: Some sample images from MNIST dataset.

Principal components Analysis (PCA) to find linear orthogonal basis that preserve the maximum variance in the data. To find the PCA basis:

1. Make the training data zero mean, by subtracting mean of the images from all training images.
2. Use matlab command cov to calculate the covariance matrix of the zero-mean data.
3. Calculate eigenvalues and eigenvectors of the cavariance matrix by using matlab command eig.
4. Now the d principal components of the data are the d eigenvectors with highest eigenvalues.
5. Project the data on these basis

Now you have the reduced dimensional data. As mentioned earlier, we will model each class by using a multivariate gaussian distribution. Now calculate the mean and covariance of each digit class separately. You can use the matlab functions $mean$ and cov for learning the means and covariances.

For a novel test input:

1. Subtract the mean vector of the training data from it.
2. After mean subtraction, project it on the already learned basis.
3. Calculate the likelihood value of the projected data for each class.

   $$p(x|class_j) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_j|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_j)'\Sigma_j^{-1}(x-\mu_j)}$$

   where $|\ |$ and $'$ represents the determinant and transpose respectively. You can use the matlab function $mvnpdf$ for calculating the likelihood value. The model parameters μ_j and Σ_j are the model parameters which you learned for the j^{th} class.
4. Assosiate the input to the class yielding highest likelihood value.

Now apply the maximum likelihood classifier on the **test data** by changing the value of d from 1 to 60. Attach the plot of classification error when varying d from 1 to 60. Report the optimal value of d producing smallest percentage of misclassified images and its corresponding classification error. If more than one d values give the lowest error then report the one with smallest d value. You can generate the confusion matrix with matlab command $confusionmat$ and plot it with $helperDisplayConfusionMatrix$. Also include the confusion matrix for optimal d value in the report. For sake of verification, the classification error for $d = 15$ is 7.03. You can compare your results with some well know classifiers for this dataset at http://yann.lecun.com/exdb/mnist/.

Note: For getting a better inside about PCA, you can plot mean image and top few eigenvectors of a class. Additionally, you can also experiment with reconstructing an image from top few eigenvectors. This part is not evaluated, it is just for your better understanding of PCA. Please do not include these results in the submission.

Exercise 3: *Human motion clustering*

Background

Learning by Demonstration (LbD) is a powerful tool widely used in robotics for acquiring new skills for robots. LbD has the advantage of learning new skills directly from demonstrations, this makes possible to avoid tedious hand programming of new tasks. Moreover, by the means of learning algorithm, the skill can be represented in a compact form reducing the amount of data to store.

LbD works in two steps. Firstly, an expert provides some demonstrations of a task to execute. There are two main ways to collect these demonstrations. The user can directly drive the robot from an initial configuration to the desired one (kinaesthetic teaching). Or, the user can execute the task several times while some sensors track its motion and collect data. Secondly, the demonstrations are encoded using machine learning algorithms. Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM) have been widely used to encode robot's skills from demonstrations and to retrieve the desired trajectory.

The set of parameters needed by GMM or HMM is usually learned using an iterative optimization technique, the so-called Expectation Maximization algorithm. The results of the Expectation Maximization algorithm are on the initial guess of the parameters. Typically, unsupervised clustering algorithms are use to determine the initial parameters.

Task

In this exercise you have to implement in Matlab two unsupervised clustering algorithms, namely the K-means (without using the *kmeans* Matlab function) and the Non-Uniform Binary Split Algorithm. These algorithms will be used to cluster the data in *gesture_dataset.mat*. After loading this file you will get three $60 \times 10 \times 3$ matrices, called *gesture_l*, *gesture_o* and *gesture_x* respectively. Each matrix contains 10 repetitions of the same gesture. Each gesture consists of 60 3D positions (x, y and z) of the user's right hand. The dataset is shown in Figure 4.

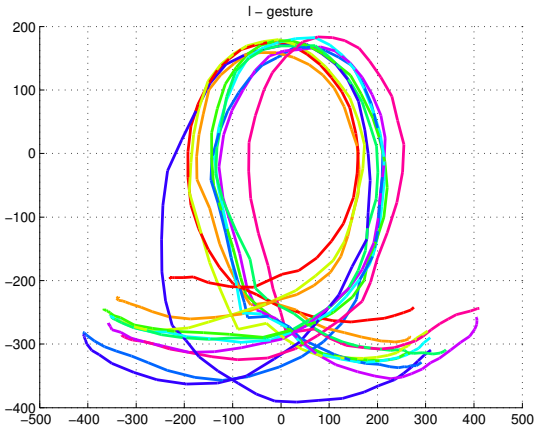
a) Classify the dataset points using the K-means ($k = 7$) algorithm, choosing the euclidean distance as distortion function. The initial cluster label are provided into the *gesture_dataset.mat* as three 7×3 matrices (*init_cluster_l*, *init_cluster_o* and *init_cluster_x*). Run the algorithm until the decrement of the total distortion function is less than 10^{-6}.

b) Classify the dataset points using the Non-Uniform Binary Split algorithm ($k = 7$) using the same split vector $\mathbf{v} = [0.08,\ 0.05,\ 0.02]^T$ for each iteration.

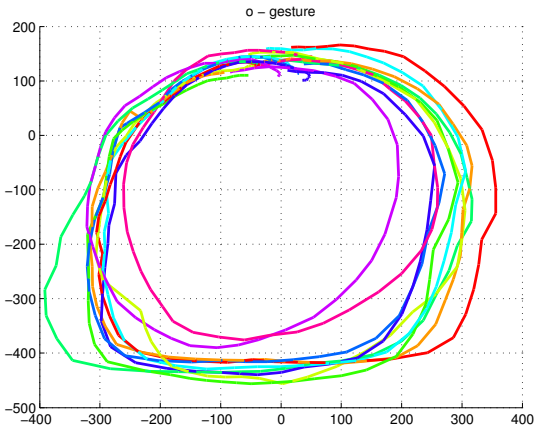
For both algorithms provide Matlab files and six figures (three for each question) in which the points belonging to each cluster are drawn with different colours. Use the mapping between clusters and colours provided in Table 1.

cluster	1	2	3	4	5	6	7
colour	blue	black	red	green	magenta	yellow	cyan

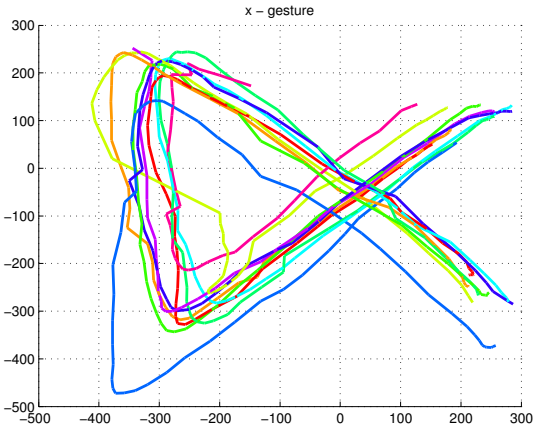
Table 1: Mapping between clusters and colours



(a)



(b)



(c)

Figure 4: 2D plots of the datasets in Exercise 3