Lehrstuhl für STEUERUNGS- UND REGELUNGSTECHNIK Technische Universität München Prof. Dongheui Lee	**MACHINE LEARNING IN ROBOTICS** Assignment 1

Exercise 1: *Estimating velocity motion model of a mobil 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.

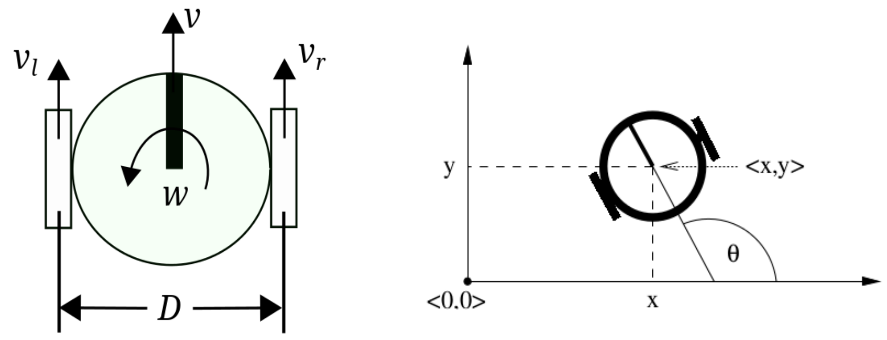


Figure 1: A simplified diagram of a mobile robot

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.

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

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

a) Now apply k-fold cross validation with k=2 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) Again 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.

c) 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 arrays). 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 $(-1, -0.05)$ and attach the generated plots.

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.

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).

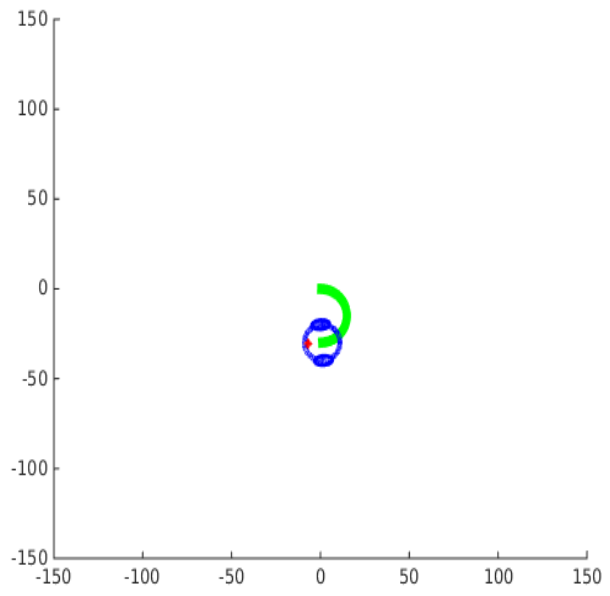


Figure 2: Robot trajectory simulation using the learned model parameters

Exercise 2: *Skin color detection using Baysian classifier*

Here you will model the skin color using a multivariate Gaussian distribution. The skin color of a pixel (for instance its rgb values) can be represented by a d dimensional vector x. We assume that x follows the distribution:

$$p(x|skin\ model) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_s|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_s)'\Sigma_s^{-1}(x-\mu_s)} \tag{4}$$

$|\ |$ and $'$ represents the determinant and transpose respectively. The model parameters μ_s and Σ_s are unknown which you will learn through training data.

Task

Learning the multivariate Gaussian

For learning the model parameters we will use the dataset available at: `http://vis-www.cs.umass.edu/lfw/lfw-bush.tgz`. Extract and place these images in a seprate directory. These images contain the photographs of former U.S president George W. Bush. In these images the center of the face is at the center of the image and approximately corresponds to 30% of the image. If you extract pixels from a rectangular region placed at the center of the image then you can be very confident that these pixels correspond to skin color. Now you will write a matlab function

function CenterPixels = ExtractCenterPixels(ImageName, p)

where *p* is in [0,1]. The returned *CenterPixels* are pixels from the rectangular region placed at the center of the image and has the dimensions (floor(p*H) x floor(p*W) x 3) where H and W are the image's height and width respectively. The pixels data can be reshaped in a better form with the matlab command:

*rgbValues = reshape(CenterPixels, [size(CenterPixels,1)*size(CenterPixels,2), 3]);*

The *rgbValues* has a dimension of (size(rgbValues,1)*size(rgbValues,2) x 3) and it's each row correspond to the rgb value of a skin pixel.

Now read 'n' face images and extract center pixels using the function you have just written. Perform this task in the function:

function [mu, Sigma] = LearnModelParameters(DirectoryName, n)

Typecast the data to 'double' and use Matlab commands 'mean' and 'cov' to calculate mu and Sigma which are the mean and covariance values of the rgb values of the extracted pixels data. The output mu and Sigma have the size (3 x 1) and (3 x 3) respectively.

Matlab Hint: An array A of size(nA x 3) and an array B of size B (nB x 3) can easily be concatenated in matlab with command C = [A;B] where the size of C is $\big($(nA+nB) x 3$\big)$. Matlab command 'Dir' can be used to get the names of files in a directory.

After this write a matlab function 'EvaluateLikelihood'

function LikValues = EvaluateLikelihood(Image, mu,Sigma)

The input to this function is an image of size (H x W x 3) and the learned parameters. The Output LikValues has dimension (H x W) and contains the likelihood value of each pixel (rgb data) of the Image. Use Equation 4 to evaluate the likelihood value.

a) Now using n=20 and p=0.2 read the first n images from 'George_W_Bush' directory and learn the values for μ_s=mu and Σ_s=Sigma. Report the learned parameter values for μ_s and Σ_s.

b) Learn the model of background pixels by using all images from the directory 'BackgroundImages' and use p=1 to learn the values for μ_b=mu and Σ_b=Sigma. Report the learned parameter values for μ_b and Σ_b.

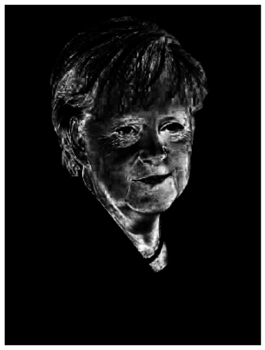
c) Apply EvaluateLikelihood on provided 'SampleImage.jpg' by using the learned μ_s and Σ_s (n=20 and p=0.2) and attach the display of your LikValues (p(x|skin model)) by using imshow(LikValues/max(max(LikValues))). Your result should look like as in Figure 4b.

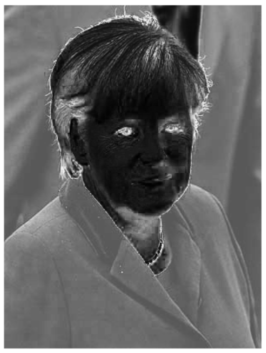
d) Again apply EvaluateLikelihood on provided 'SampleImage.jpg' but now by using the learned μ_b and Σ_b (all images and p=1) and attach the display of your LikValues (p(x|background model)) by using imshow(LikValues/max(max(LikValues))). Your result should look like as in Figure 4c.

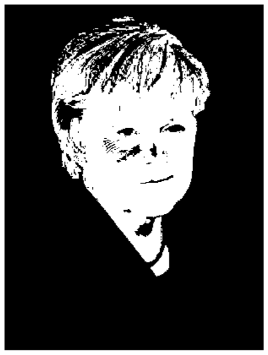
e) Since now you have the likelihood value of each pixel, you can easily apply likelihood ration test to classify a pixel as skin or non-skin pixel. A pixel will be classified as a skin pixel if $\frac{\text{p}(x|\text{skin model})}{\text{p}(x|\text{background model})} >= 1$ and non skin otherwise. Apply the likelihood ration test on the results obtained from c) and d). In matlab elementwise division of two equally sized matrices can be readily performed by using A./B. Replace the pixles classified as skin by 1 and non-skin by 0 to create a binary image and attach the result of the binary image. The binary image should look like as in Figure 3d.

f) Your skin classifier can also be used as a naive face detector. Use the provided matlab function 'FindBiggestComp.m' to find the largest connected component in the result obtained in e). This function returns the x-coordinates and y-coordinates of the bounding box. Plot this bounding box and see if it overlapps the face. If every thing has been done correctly then the the result will look like as in Figure 3e.

(a) Original Image


(b) p(x|skin model)


(c) p(x|background model)


(d) Binary image

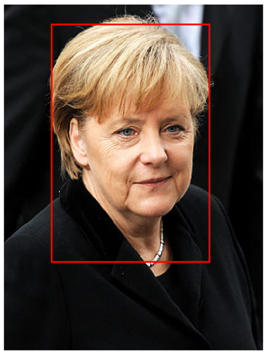

(e) Face recognition

Figure 3: Results for Exercise 2

<u>Exercise 3</u>: *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.

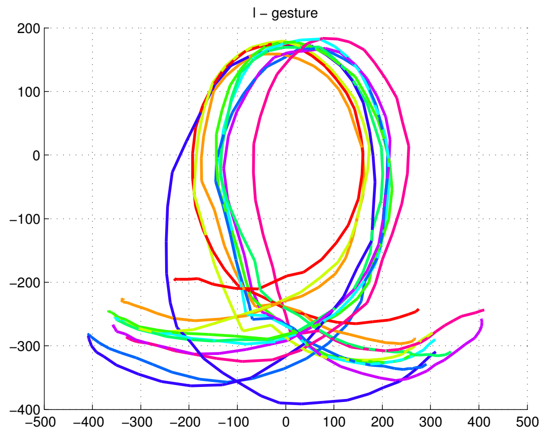
Task

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 $60 \times 10 \times 3$, called *gesture_l*, *gesture_o* and *gesture_x* respectively, containing 10 repetitions of the same gesture. Each gesture consists of 60 right hand positions. The dataset is shown in Figure 4.

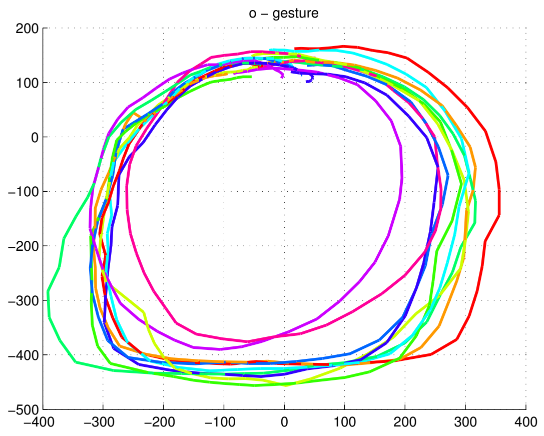
a) Classify the dataset points using the K-means ($k = 7$) algorithm, choosing the euclidean distance as distortion function. The initial cluster label are provided into the *gesture_dataset.mat* as three 7×3 matrices (*init_cluster_l*, *init_cluster_o* and *init_cluster_x*). Run the algorithm until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

b) Classify the dataset points using the Non-Uniform Binary Split algorithm. Compare the results with those obtained using K-means.

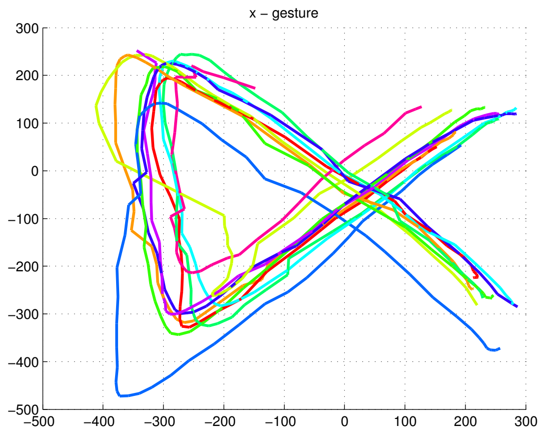
For both algorithms provide Matlab files and six figures (three for each question) in which the points belonging to each cluster are drawn with different colours. Use the mapping between clusters and colours provided in Table 1.



(a)



(b)



(c)

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

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

Table 1: Mapping between clusters and colours