CS 4758/6758 Robot Learning: Homework 3

Due March 14 at 4:00 PM (in class)

1 Kalman Filters: Multiple Sensors (25 points)

Suppose we have a continuous system with state $x \in \Re^N$. The state at time k is determined by the equation:

$$x_{k+1} = x_k + w_{k+1} \quad \text{where } w_{k+1} \sim N(0, Q)$$

Now we have two sensors z_k and y_k. One sensor gives data at $k = 1, 3, 5, \ldots$, and the other sensor gives data at $k = 2, 4, 6, \ldots$.

$$\begin{aligned} z_k &= x_k + v_k \quad \text{where } v_k \sim N(0, R_1) \\ y_k &= x_k + e_k \quad \text{where } e_k \sim N(0, R_2) \end{aligned}$$

Derive the time and measurement update equations for this situation.

2 Kalman Filter Implementation (50 points)

You are designing a ground robot that will operate autonomously and use a Kalman filter for localization. The robot moves according to $x_k = Ax_{k-1} + w_k$, $w_k \sim \mathcal{N}(0, Q)$ and makes measurements $z_k = Hx_k + v_k$, $v_k \sim \mathcal{N}(0, R)$. You can choose between three chassis/dynamics and two sensors arrays but need to decide which pairing is best. Your options are:

Dynamics

$$(a): \quad A = 1.01 \begin{bmatrix} \cos .1 & -\sin .1 \\ \sin .1 & \cos .1 \end{bmatrix} \quad Q = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

$$(b): \quad A = \begin{bmatrix} .99 & 0 \\ 0 & .99 \end{bmatrix} \quad Q = \begin{bmatrix} .4 & -.2 \\ -.2 & .4 \end{bmatrix}$$

$$(c): \quad A = \begin{bmatrix} 1.01 & 0 \\ 0 & 1.01 \end{bmatrix} \quad Q = \begin{bmatrix} .1 & -.05 \\ -.05 & .25 \end{bmatrix}$$

Sensors

$$(d): \quad H = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad R = \begin{bmatrix} .1 & 0 \\ 0 & .1 \end{bmatrix}$$

$$(e): \quad H = \begin{bmatrix} 1 & .5 \\ .5 & -1 \\ 1 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 1 & 0 & -.5 \\ 0 & 2 & .5 \\ -.5 & .5 & 1 \end{bmatrix}$$

(a) The expected squared error of the Kalman filter estimate $\hat{x}_k$ is given by $\mathbb{E}\left[(x_k - \hat{x}_k)^T(x_k - \hat{x}_k)\right]$. Express this quantity in terms of the estimate covariance P_k. (10 points)

(b) For each of the six combinations of dynamics and sensors, plot the expected squared error with respect to time. Assume you know the robot's starting position; that is, $P_0 = 0$. Attach the code you used for your calculations. (20 points)

(c) If you only plan on operating the robot for $t < 50$, which combination has the lowest error? What is its expected error at $t = 50$? (5 points)

(d) If you plan to operate the robot indefinitely, which combination is best? What is its expected error as $t \to \infty$? (5 points)

(e) If the robot instead starts from a completely unknown position, how do your answers to (c) and (d) change? Repeat them for this case. (10 points)

3 Pose classification (25 points)

For this question, you will use skeleton tracking data from Kinect to classify human poses. This data was taken from the NiTE skeleton tracking software. Refer to `http://pr.cs.cornell.edu/humanactivities/data.php` for more information on the data format. Link indices are as follows:

```
Joint number -> Joint name
   1 -> HEAD
   2 -> NECK
   3 -> TORSO
   4 -> LEFT_SHOULDER
   5 -> LEFT_ELBOW
   6 -> RIGHT_SHOULDER
   7 -> RIGHT_ELBOW
   8 -> LEFT_HIP
   9 -> LEFT_KNEE
  10 -> RIGHT_HIP
  11 -> RIGHT_KNEE
  12 -> LEFT_HAND
  13 -> RIGHT_HAND
  14 -> LEFT_FOOT
  15 -> RIGHT_FOOT
```

The attached `setupLinkInds` script will populate these indices as global variables in your MATLAB workspace, and should be run before running any of the other provided functions.

The data is given in the form of a .mat file with the following fields (for N different skeletons):

P	Nx15x3	Points for each joint, in X,Y,Z world-space coords
R	Nx15x3x3	Rotation matrices for each joint, relative to world-space
person	Nx1	Person index, from 1 to 4, for each case
class	Nx1	Pose class index, from 1 to 5
links	14x2	Pairs of joint indices with links between them, used for visualization

The poses are:

1. Chopping food
2. Holding a cup
3. Opening a pill bottle
4. Talking on the phone
5. Writing on a whiteboard

(a) Use the provided `showSkelFromInd` function to plot one skeleton for each pose class and attach these plots. Use the `links` variable provided in poseData.mat

(b) For this part, you will complete `trainPoseClassifiers.m` and `eval/confToPrecRec.m` to train multinomial logistic classifiers for a series of cross-validation splits of the given pose data. To get started, read `trainPoseClassifiers.m` and make sure you understand what it's doing.

For this problem, we want to have four different *per-person* cross-validation splits of the data. In each split, the data from a different subject is held out as test data, and the data from the others is used for training. Methods such as this are a good way of evaluating the generalization performance of a feature set/learning algorithm.

The starter code in `trainPoseClassifiers.m` is mostly complete, but you need to add code to properly split the pose data into training and testing sets as described above. The `splitData` function will probably be useful for this.

For evaluation, we want to look at the *confusion*,*precision*, and *recall* for each split of the data. Confusion will be given in the form of a confusion matrix – for N classes, this will be an NxN matrix C such that $C_{i,j}$ is the number of cases which are actually in class i, but were predicted to be in class j. Precision is defined as the fraction of cases which are predicted to have a given class which actually belong to that class. Recall is defined as the fraction of cases of a given ground-truth class which are correctly classified as belonging to that class.

Code to evaluate a set of weights B learned by `mnrfit` is provided in `eval/evaluateMNRClass`. This code will use the given weights and data to compute predictions, and produce a confusion matrix comparing the predicted and ground-truth classes. However, the `eval/confToPrecRec` function, which is supposed to compute precision and recall given a confusion matrix, is currently a stub, and you will have to implement this function to properly compute precision and recall.

Fix the code in the indicated locations in `trainPoseClassifiers.m` and `eval/confToPrecRec.m`, then use `trainPoseClassifiers.m` to train classifiers and report performance for each cross-validation split. Either fill in the table on the next page, or produce a similar table reporting precision and recall for each split and class, as well as split-wise and class-wise averages.

What do you notice about the split-wise averages? Does one split significantly underperform the others? Why do you think this is? *Hint: take a look at some poses for each person*

Results table for problem 3B

	Chopping		Hold cup		Open bottle		Use phone		Whiteboard		Average	
Test person	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
1												
2												
3												
4												
Average												