Introduction

In this assignment you have two options to choose from. Choose only one!

Submission details

- This assignment can be done individually or in pairs (though we strongly encourage you to work in pairs).
- Programs should be in Python (preferably Python 2) or Matlab.
- For submission, package up your code as a zip or tar file. Include your written answers as a pdf or word file named writeup.[pdf|docx]. There's a bunch of graphs we want you to plot, include these in your writeup, and please label them so we know which figures go with which sub-problem. Don't forget to explain your work!
- If you have any questions about this assignment, please contact the TA (stinger@tx.technion.ac.il). Your email subject should start with CS5660.

Option 1 – Kalman Filtering

The problem setup

The problem we will address is tracking of a moving target. We first need to write down the equation describing the target's motion. We will use a standard second order model.
Notation:
k = discrete time
x, y = target position (horizontal and vertical)
$\dot{x}, \dot{y}$ = target velocity (horizontal and vertical)
a_x, a_y = target acceleration (horizontal and vertical)
We will assume the acceleration in each direction behaves as white noise:
$a_x = \frac{d^2x}{dt^2}$ = white noise, $a_y = \frac{d^2y}{dt^2}$ = white noise.
In other words:
$w(k) = \begin{bmatrix} w_x(k) \\ w_y(k) \end{bmatrix}$ = zero mean white Gaussian "acceleration noise" with covariance σ_w^2 (in both directions).

We measure the position of the target every time interval T. This will allow us to work with a discrete system.

The discrete state vector is:

$$x(k) = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}$$

The discrete state-space model:

$$x(k+1) = Ax(k) + Bw(k)$$

Where

$$A = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0.5T^2 & 0 \\ T & 0 \\ 0 & 0.5T^2 \\ 0 & T \end{bmatrix}$$

We measure only the position, hence, the observation equation is:

$$z(k) = Cx(k) + v(k)$$

Where

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and $v(k) = \begin{bmatrix} v_x(k) \\ v_y(k) \end{bmatrix}$ = zero mean Gaussian measurement noise with covariance σ_v^2 (in both directions). The measurement vector is: $z(k) = \begin{bmatrix} z_x(k) \\ z_y(k) \end{bmatrix}$

Tasks

1. **Simulation: generate the ground-truth and observations**
   Assume the target was initially at rest $x(0) = y(0) = 0$.
   a. Generate the following ground-truth trajectories of the target to be tracked:
      1. Between $k = 0$(sec) and $k = 200$(sec) the target moves with constant velocity $\dot{x} = 300\ m/sec,\ \ \dot{y} = 0\ m/sec$
      2. Between $k = 200$(sec) and $k = 300$(sec) the target accelerates at $a_x = a_y = 2\ m/sec^2$ along each axis.
      3. Between $k = 300$(sec) and $k = 500$(sec) the target moves with constant velocity $\dot{x} = 0\ m/sec,\ \ \dot{y} = 300\ m/sec$
   b. Plot the exact trajectory of the target using $T = 1$(sec).
   c. Next you will make measurements (also with $T = 1$(sec)). The measurement process (described above) is noisy with $\sigma_v = 1000(m)$, simulating a real-world scenario where measuring is rarely perfect. This means your measurements will not be identical to the ground-truth.
      Plot the observations (measurements) on the same x-y plot as the exact trajectory.
   d. Generate also a plot of the velocity along each axis vs. time.

2. **Tracking**
   Implement a Kalman Filter for estimation of the target state.
   a. Let $\sigma_w = 0.3\left(\frac{m}{sec^2}\right)$. Generate a plot of the estimated trajectory, the raw measurements, and the true trajectory. Generate a plot of the estimation and measurement errors versus time. Repeat the same for the velocity. (Do not square the errors). Explain your results for all parts of the trajectory. In your

explanation refer to the model we chose. Remember that errors are measured against the ground-truth.

b. In the previous item you obtained large estimation errors at some part of the trajectory. Modify the filter process noise σ_w to reduce this phenomenon. Plot the position data (true and estimated trajectories and the raw measurements) as well as the corresponding errors. Explain your observations.
c. Set σ_w to its original value. Repeat the first task with the filter parameter σ_v reduced by three orders of magnitude. Plot the position data only. Explain your results. Compare and discuss the three experiments. Note: the measurements are still generated with the same standard deviation as before. It is the filter parameters that change!
d. Set σ_w and σ_v to their original values. Initialize your filter with the state vector $[0 \quad -100 \quad 0 \quad -100]^T$. Set the initial error covariance to be the identity matrix. Repeat the experiment, plot and discuss your results.
e. Adjust the initial error covariance such that the undesired phenomenon you've obtained in the previous item is reduced. Repeat the experiment, plot and discuss your results.
f. Now we shall consider the case of an imperfect sensor, i.e., we will not always get a measurement. Assume that the sensor detects the target with a known probability P_d. If at time k the target is not detected, there is no measurement provided to the filter at that time. Propose a (slightly) modified filter that is capable of dealing with such cases. Using previously generated measurements, test your filter for $P_d = 0.5$. Generate a plot of the estimation results for the new filter and the corresponding estimates of the original filter (i.e. for $P_d = 1$). Explain and discuss the results.

Option 2 – dimensionality reduction

This option is more open-minded (and possibly more rewarding).

1. Collect data: Come up with a data-set that you're interested in. For example, it could be music files, artists, stocks, images of animals – you're welcome to approach Lihi to discuss the choice of data-set. It is your job to collect the data (or find a readymade data-set online). Ideally, the data-set you chose will have expected 2 or 3 underlying variables. For example we saw in class that for face images the variables could be viewing angle and lighting direction. A good size of dataset would be a few hundred entries.
2. Extract features: You will need to represent each entry in your data-set as a feature vector. You get to choose what feature vector you will use. Please explain and motivate your choice.
3. Find Nearest-Neighbors: Select 10 entries in your data-set and find for each entry its Nearest-Neighbor (NN). The NN is the one whose feature vector is the most similar. You get to define the distance measure you are using. Please explain your choice clearly and motivate it. Include in your report the 10 entries you chose and their NN.

4. Dimensionality reduction: Apply 3 methods for dimensionality reduction to reduce to 2 dimensions. You could choose from PCA, MDS, LDA, LLE, Isomap, t-SNE, random projection. All of these have open source Matlab implementations. I am not sure about Python. It is ok to use code you find online.
    a. Visualize the 2D results. Explain your observations. Are these the results you thought you'd get? Do they make sense? Can you interpret what is the underlying variable for each axis?
    Bonus: Visualize the connection between the 2D points to the higher dimension. This isn't easy, so be careful.
    b. Compare the results of the 3 dimensionality reduction methods. Which one makes most sense to you?

5. Find Nearest-Neighbors in low dimensions: Find the NN of the 10 entries you chose in item 3, this time after dimensionality reduction. Use the same distance measure you chose before, if possible. You might choose to use a different distance measure now that you are working with a reduced dimension. Please explain your choice of distance measure clearly and motivate it (even if you chose to keep the same distance measure please explain why).
    a. Include in your report the NN of the 10 selected entries for each dimensionality reduction method.
    b. Are these the same as before dimensionality reduction? Do they make more sense or less?