Guidelines – Read Carefully! You are provided a zip file containing a single folder named xyz007 containing the following folders and files.

xyz007
xyz007/data.txt
xyz007/target-actual.png
xyz007/target-observed.png
xyz007/laser.png
xyz007/mp440.py
xyz007/main.py

You should first rename the folder name xyz007 to be your NETID. If you are working in a team, the folder should be named with both group members' NETIDs, in the format of NETID1_NETID2. The folder name letters can be either upper or lower case. When you are ready to submit, remove any extra files (e.g., some python interpreters will create .pyc files) that are not required and zip the entire folder. The zip file should also be named with your NETID as NETID.zip (or NETID1_NETID2.zip for groups with two members). You may create additional python files though it should not be necessary; all your code can bit fit into `mp440.py`. This MP also requires the submission of a report in PDF format. Please name your PDF file as report.pdf and put it under the main submission folder, i.e.,

NETID/report.pdf or NETID1_NETID2/report.pdf

You are required to write your program adhering to Python 2.7 standards. In particular, DO NOT use Python 3.x. Beside the default libraries supplied in the standard Python distribution, you may use ONLY pygame, numpy, and matplotlib libraries (i.e., among the extra libraries installed after the standard python libraries).

As mentioned in class, you may form groups of up to two students. Only a single student should submit the solution.

Problem 1 [100 points]. 2D Kalman Filter and an Application.

a) Implementation of a 2D Kalman filter (60 points). You are to implement a (relatively simple) 2D Kalman filter. Suppose that there is a point mass moving in 2D with the system equation being

$$x_k = \begin{pmatrix} x_{1,k} \\ x_{2,k} \end{pmatrix} = \begin{pmatrix} x_{1,k-1} \\ x_{2,k-1} \end{pmatrix} + \begin{pmatrix} u_{1,k-1} \\ u_{2,k-1} \end{pmatrix} + \omega_{k-1},$$

and the observer being

$$z_k = \begin{pmatrix} z_{1,k} \\ z_{2,k} \end{pmatrix} = \begin{pmatrix} x_{1,k} \\ x_{2,k} \end{pmatrix} + \nu_k.$$

Assume that the system noise ω is a zero mean Gaussian with the covariance matrix

$$Q = \begin{pmatrix} 10^{-4} & 2 \times 10^{-5} \\ 2 \times 10^{-5} & 10^{-4} \end{pmatrix}$$

and the measurement noise is a zero mean Gaussian with the covariance matrix

$$R = \begin{pmatrix} 10^{-2} & 5 \times 10^{-3} \\ 5 \times 10^{-3} & 2 \times 10^{-2} \end{pmatrix}.$$

a.1 (20 points) To start, derive the required update equations for the 2D Kalman filter. For this task, you may simply write down the equations and provide the needed term with brief explanation of how these are obtained. This should go into your report.

a.2 (20 points) In the provided `main.py` file, the function `_load_data()` loads a set of 2D data with each item in the list being a list of the form $[u_{1,k-1}, u_{2,k-1}, z_{1,k}, z_{2,k}]$. You are to implement the equations you derived earlier in

`kalman2d(`*data*`)`

that computes the estimated $x_{1,k}, x_{2,k}$. Your function should return a list of list, e.g., in the form of

$[[x_{1,1}, x_{2,1}], \ldots, [x_{1,k}, x_{2,k}]]$

Your function will be tested on new data, with the same x_k, z_k, Q and R, to check its correctness. Note that you will need to fix the initial P matrix, which should be of the form of λI where λ is a scaling factor and I is the 2D identity matrix.

a.3 (15 points) Implement the function

`plot(`*data*`,`*output*`)`

using matplotlib that plots both the observed 2D points as well as the estimated ones. That is, for the set of $(z_{1,k}, z_{2,k})$ in the data, plot them and connect consecutive data points using straight line segments. Do the same for the estimated $(x_{1,k}, x_{2,k})$ on the same plot. Draw the observations with a blue line and the estimates with a red line.

a.4 (15 points) Report what value of $P = \lambda I$ that you found to work well for the given data. Also, include a plot produced by your plotting function for the included data set.

The data that is included is also listed in the table at the end of this document. As mentioned, your program should work for data that is different from this provided data set.

b) An "application" of a 2D Kalman filter. Suppose you are to shoot a cloaked target in a 2D region of 500×500 (pixels). You may shoot a laser anywhere along the bottom of the region and while the laser has a short burst, is quite powerful and travels through the region vertically instantaneously. At the same time, the laser uses lots of energy which means you only have a single shot. While the target is cloaked, as it moves, it sends energy signatures that allows the prediction of its next move. After the move is complete, the target also sends a signal that can be tracked with some Gaussian noise. Your goal is to track the target as it moves and decide when and where (alone the base) to fire the laser. If you enable the part of the code in the `main` function in `main.py`, the program will draw the observed target.

b.1 (20 points) In this problem, you are to implement the function

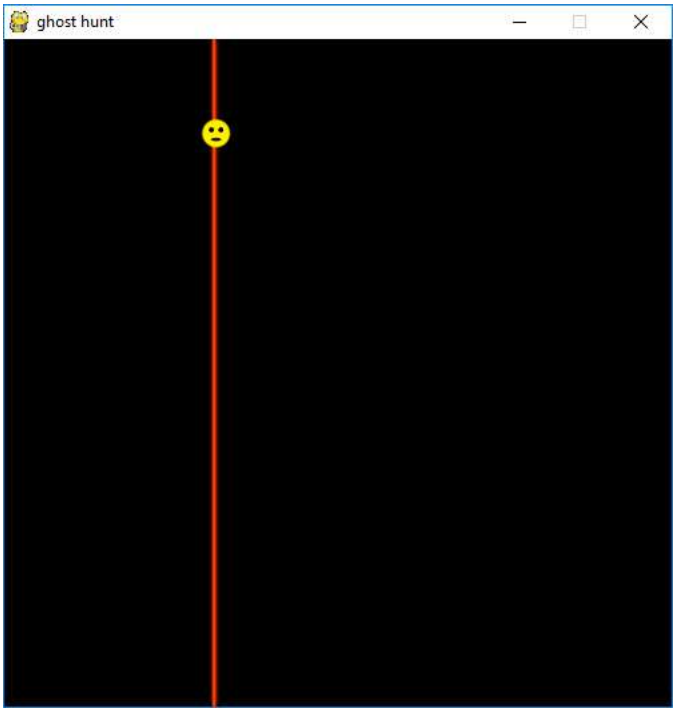
`kalman2d_shoot(`*ux*`, `*uy*`, `*ox*`, `*oy*`, `*reset*`)`

that takes the current $u_{1,k-1}, u_{2,k-1}, z_{1,k}, z_{2,k}$ values and based on your estimate, decide whether to fire the laser and where. The argument *reset* is set to true when the function is called the first time in a new iteration (i.e., you should in such case reset your internal states). Note that in the python code we use x and y to represent the two coordinates instead of using x_1 and x_2, to make it easy to represent the two coordinates. The return value for the function is a 3-tuple of the format

$(ex, ey, shoot)$

where ex and ey are the estimated x and y location and *shoot*, a boolean variable, indicates whether

the laser should be fired (at ex). Whenever *shoot* is set to be true when returning, the program will stop and show the laser and the actual target location. On the console screen, the relevant information will also be displayed. You are to fire a shot within 200 iterations, which is about 40 seconds since the system update is performed at around 5Hz. The absolute error in the x coordinate, which will be averaged over about 50 runs, should be below 8 (pixels) to get 80% of the credits. For full credits, an accuracy of 6 pixels is needed. A screenshot is provided in the figure below showing what you would see after firing the shot.



Note that your program will be tested along five trajectories; the provided trajectory is only one of them. They will have the same noise settings.

b.2 (10 points) Note that you may do anything you want in `kalman2d_shoot`, i.e., unlike in `kalman2d`, you are not limited to only using Kalman filter in making your decision in `kalman2d_shoot`. In the report, state how you decide when to shoot.

b.3 (7 bonus points) Finally, implement another version of your shooting routine in

`kalman2d_adv_shoot`(ux, uy, ox, oy, $reset$)

which will be evaluated based on the time that has passed and the accuracy. For each trial, let the time passed before you decide to shoot be t and the absolute error in x be e_x, then the score for the trial (you must make a decision within 200 iterations as before) is given by

$$score = t + e^{\frac{e_x}{2}}.$$

Your goal is to minimize this score; the code will be subject to the same 50 tests used for evaluating `kalman2d_shoot` and the average will be taken. The top 10 teams will be awarded 7–4–4–4–2–2–2–2–2–2 bonus points with better performing implementation given higher score.

Table 1: Sample control input and observations for running the Kalman filter.

k	$u_{1,k-1}$	$u_{2,k-1}$	$z_{1,k}$	$z_{2,k}$
1	0.258286754	-0.323660615	2.275352754	0.973676385
2	0.167658082	-0.173943802	2.230101836	0.701036583
3	0.213711214	-0.096513636	2.77765305	0.664235947
4	0.257826742	-0.153303882	2.929903792	0.725217065
5	0.227367085	-0.226875214	3.192038877	0.604513851
6	0.29707499	-0.125012662	3.347010866	0.278565189
7	0.253380049	0.040957103	3.900306915	0.333608292
8	0.434690544	-0.179539834	3.95920546	0.266202459
9	0.233358557	-0.110426773	4.513029017	0.264158685
10	0.309397654	-0.039241497	4.681395671	0.128679188
11	0.268100576	-0.052630663	4.944550247	-0.183817474
12	0.336295529	0.050916321	5.380023776	0.314923847
13	0.383568488	-0.263521311	5.816818264	-0.115412464
14	0.271740175	-0.316833884	5.892258439	-0.677581348
15	0.37813998	-0.100109434	6.35130042	-0.551171782
16	0.32708519	-0.124288625	6.50389161	-0.955727407
17	0.318597789	-0.216869042	6.992389399	-0.951967449
18	0.215326213	-0.334744463	7.043205612	-1.221487912
19	-0.035040957	-0.077543457	7.006712655	-1.476399369
20	0.170197008	-0.062307673	7.076019663	-1.702334042
21	0.325171499	-0.008000277	7.534015162	-1.481437319
22	0.414100752	-0.043273351	8.001067914	-1.49148567
23	0.197609976	-0.391999135	8.211005889	-2.015420805
24	0.409390148	-0.114261647	8.649643037	-2.122464452
25	0.334974442	-0.065947082	8.925195479	-2.247646534