CS 4610/5335: Robotic Science and Systems (Spring 2022) — Lawson L.S. Wong
Northeastern University — Due March 11, 2022

Exercise 3: Kalman Filtering, Localization, and Mapping

Please remember the following policies:

- Exercises are due at **11:59 PM** Boston time (ET).
- **Submissions should be made electronically on Canvas, as a single .pdf file for written parts, and as a single .zip file for code.** Check your submission and make sure you have included all files! You can make as many submissions as you wish, but only the latest one will be considered, and late days will be computed based on the latest submission.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 5% per day. Submissions later than this will not be accepted.
  There is no limit on the total number of late days used over the course of the semester.
- Written solutions may be handwritten or typeset. For the former, please ensure handwriting is legible. If you write your answers on paper and submit images of them, that is fine, but please put and order them correctly in a single .pdf file. One way to do this is putting them in a Word document and saving as a PDF file.
- For programming questions, we recommend leaving sufficient comments to explain the logic of your solution.
- Some questions are intended for CS 5335 students only. CS 4610 students may complete these for extra credit.
- You are welcome to discuss these problems with other students in the class, but you must understand and write up the solution yourself, *and* indicate who you discussed with (if any).
- Contact the teaching staff if there are *extenuating* circumstances.

KF. **5 points.** Formulate the following as Kalman filtering problems (write out the linear system equations). Use the provided control inputs and measurements to find the posterior distribution. Show your work.

(a) 1-*D displacement.* A robot is situated at an unknown position $x \in \mathbb{R}$ on a 1-D line.
At each time t, it is commanded to move by $\Delta x = u_t$,
perturbed by some unknown transition noise $v_t \sim \mathcal{N}\left(0, \sigma_v^2\right)$, such that $x_{t+1} = x_t + u_t + v_t$.
After it moves, we get a measurement z_t based on its new position x_{t+1},
perturbed by some unknown observation noise $w_{t+1} \sim \mathcal{N}\left(0, \sigma_w^2\right)$, such that $z_{t+1} = x_{t+1} + w_{t+1}$.
Using the following parameters and inputs, find the posterior distribution on x_3.

Noise parameters:	$\sigma_v^2 = 0.04$	$\sigma_w^2 = 0.01$		
Control inputs:	$u_0 = -0.5$	$u_1 = 1.2$	$u_2 = 0.3$	
Measurements:		$z_1 = -0.7$	$z_2 = 0.6$	$z_3 = 0.95$
Initial belief:	$x_0 \sim \mathcal{N}(0, 1)$			

(b) 2-*D displacement.* Now, the robot is situated on a 2-D plane, with position $\begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2$.
Similar to part (a), the robot is commanded via a displacement in each axis independently, with corresponding transition and observation noises in each axis.
Using the following parameters and inputs, find the posterior distribution at time $t = 3$.

Noise parameters:	$\Sigma_v = \begin{bmatrix} 0.04 & 0 \\ 0 & 0.09 \end{bmatrix}$	$\Sigma_w = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.02 \end{bmatrix}$		
Control inputs:	$u_0 = \begin{bmatrix} -0.5 \\ 0.3 \end{bmatrix}$	$u_1 = \begin{bmatrix} 1.2 \\ -0.6 \end{bmatrix}$	$u_2 = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$	
Measurements:		$z_1 = \begin{bmatrix} -0.7 \\ 0.3 \end{bmatrix}$	$z_2 = \begin{bmatrix} 0.6 \\ 0.0 \end{bmatrix}$	$z_3 = \begin{bmatrix} 0.95 \\ 0.15 \end{bmatrix}$
Initial belief:	$x_0 \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$			

(c) 1-*D motion.* Commanding robots using displacements is not too realistic. In practice, we may control velocities, which in turn lead to changes in position. Returning to the robot on the 1-D line in part (a), consider a new control model, where we can command changes in *velocity* $\Delta v = u_t$, such that $v_{t+1} = v_t + u_t +$ velocity noise, and positions get changed indirectly by $x_{t+1} = x_t + v_t +$ position noise. (Note that, for simplicity, we assume changes in position occur before changes in velocity.)
As before, we can only measure the resulting position (with noise); velocity cannot be directly observed. Using the following parameters and inputs, find the posterior distribution at time $t = 3$.

Noise parameters:	Vel. noise $\sim \mathcal{N}(0, 0.03)$	Pos. noise $\sim \mathcal{N}(0, 0.02)$	$\sigma_w^2 = 0.01$	
Control inputs:	$u_0 = -0.2$	$u_1 = 0.0$	$u_2 = 0.1$	
Measurements:		$z_1 = 0.4$	$z_2 = 0.9$	$z_3 = 0.8$
Initial belief:	$x_0 \sim \mathcal{N}(0, 1)$	$v_0 \sim \mathcal{N}(0.5, 2)$		

Hint: The Kalman filter state space should be two-dimensional.
Optional: Can you propose a different model where both velocities and positions change simultaneously?
Optional: Even this model is not too realistic; in practice, we can only control *accelerations* (via forces/torques by Newton's second law of motion), which lead to changes in velocity, which in turn affects positions. Change the above model/formulation to reflect this observation.

(d) 2-*D displacement with nonlinear measurements.* Returning to the robot on the 2-D plane in part (b), commanded by x-y displacements, suppose we could no longer observe the position in each axis independently. Instead, there is a single sensor located at the origin, which can only measure the (squared) distance to the robot. Specifically, if the robot is located at $\begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2$, then $z = x^2 + y^2 +$ observation noise.
Using the following parameters and inputs, find the posterior distribution at time $t = 3$.

Noise parameters:	$\Sigma_v = \begin{bmatrix} 0.04 & 0 \\ 0 & 0.09 \end{bmatrix}$	$\sigma_w^2 = 0.01$		
Control inputs:	$u_0 = \begin{bmatrix} -0.5 \\ 0.3 \end{bmatrix}$	$u_1 = \begin{bmatrix} 1.2 \\ -0.6 \end{bmatrix}$	$u_2 = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$	
Measurements:		$z_1 = 0.6$	$z_2 = 0.4$	$z_3 = 1.0$
Initial belief:	$x_0 \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$			

(e) *Light-dark domain (Platt et al., RSS 2010).* Remaining on the 2-D plane with displacements:
"In the light-dark domain, a robot must localize its position in the plane before approaching the goal. The robot's ability to localize itself depends upon the amount of light present at its actual position. Light varies as a quadratic function of the horizontal coordinate.
Depending upon the goal position, the initial robot position, and the configuration of the light, the robot may need to move away from its ultimate goal in order to localize itself."
The system dynamics and controls are the same as in part (b), and the x-y position is also observed. However, in the light-dark domain, the *observation noise* is a function of the robot's x-position; specifically, the observation noise in each axis is given by $\sigma_w^2(x) = \frac{1}{2}(5 - x)^2 + 0.01$.
Using the following parameters and inputs, find the posterior distribution at time $t = 3$.

Noise parameters:	$\Sigma_v = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$	$\Sigma_w = \begin{bmatrix} \sigma_w^2(x) & 0 \\ 0 & \sigma_w^2(x) \end{bmatrix}$		
Control inputs:	$u_0 = \begin{bmatrix} 1.0 \\ -1.0 \end{bmatrix}$	$u_1 = \begin{bmatrix} 2.0 \\ -1.0 \end{bmatrix}$	$u_2 = \begin{bmatrix} -5.0 \\ 0.0 \end{bmatrix}$	
Measurements:		$z_1 = \begin{bmatrix} 3.5 \\ -1.0 \end{bmatrix}$	$z_2 = \begin{bmatrix} 5.3 \\ -0.5 \end{bmatrix}$	$z_3 = \begin{bmatrix} -2.0 \\ 0.0 \end{bmatrix}$
Initial belief:	$x_0 \sim \mathcal{N}\left(\begin{bmatrix} 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}\right)$			

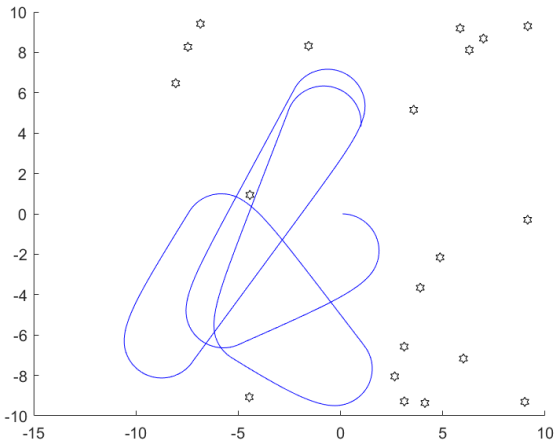
For more on this domain: `http://www.roboticsproceedings.org/rss06/p37.html`

(f) **Extra credit.** As mentioned in lecture, Kalman filters are widely applicable, including to various aspects of disease modeling. Choose and study a recent paper on applying Kalman filters to track some partially observable aspect of COVID-19 (e.g., reproduction number, number of cases, number of deaths, etc.). Explain the Kalman filter model used precisely, including what the system equations are (similar to the above parts). Use some numbers to perform several filtering steps to provide an illustrative example. *Example:* `https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0244474` You may choose any paper, as long as it applies Kalman filtering to tracking some aspect of COVID-19.

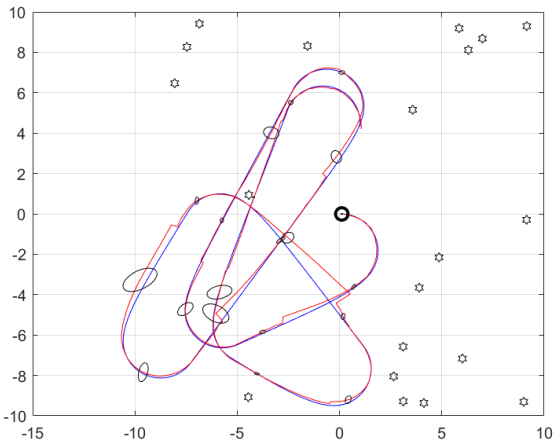
If Kalman filtering is still unclear to you, you should practice the above question again with different numbers.

Download the starter code (`ex3.zip`). In this file, you will find:

- `ex3_ekf.m`: Main function. Call `ex3_ekf(<questionNum>)` with an appropriate question number (0–4) to run the code for that question. Do not modify this file! (Feel free to actually make changes, but check that your code runs with an unmodified copy of `ex3_ekf.m`.)
- `E0.m` – `E4.m`: Code stubs that you will have to fill in for the respective questions.
- `e1.mat` – `e3.mat`, `e3_r8.mat`: Data files for each question.
- `parameters.m`: Parameters for EKF-SLAM.
- `visualize.m, visualize_path.m, visualize_map.m, average_error.m`: Helper functions for visualizing results and computing errors.



E0: Ground truth map and trajectory.



E0: Robotics toolbox EKF localization solution (in red).

E0. **0 points.** The Robotics toolbox actually provides an implementation of EKF-based SLAM already. Read through the code examples in Chapter 6 and learn how to use the existing functions provided by the toolbox. Run the code that is listed in the textbook for localization, mapping, and SLAM. **Use a maximum range of** 4 **and field-of-view of** $[-\frac{\pi}{2}, \frac{\pi}{2}]$ **for the sensor.** For example, the code for localization should be very similar to that shown on page 163 (the example code already has the correct maximum range and field of view). Return the EKF objects for each and save them in your workspace, because they will provide a way for you to check your own implementation in the upcoming questions.

If you just run the code from the textbook, you will probably not get the same map and trajectory as illustrated in Figure 6.7 on page 163. This is because the map, trajectory, and noise (both odometry and sensing) are generated *randomly*. To ensure that your EKF objects use the same data that we have provided in our `*.mat` files, reset the random number generator's seed by calling `rng(0)`. In your code, should do this *every* time before you call `LandmarkMap` (which randomly generates a map) and `ekf.run` (which randomly generates the trajectory and noisy odometry/sensor readings).

If you reset the seed correctly, you should get the same map as shown in Figure 6.7, but the path is probably still different. (We actually do not know what random seed generates the trajectory shown in the textbook.) You can visualize both by calling: `visualize({}, {}, [], map, veh, 'n', [])`
which produces the left figure above.
Visualizing the result of EKF localization (following the code in the textbook) gives the figure on the right.

E1. **5 points.** Implement EKF-based localization, given a known map.

Observe how the `E1` function is called in `ex3_ekf.m`; data is loaded from `e1.mat`, which contains odometry and observation data (`odo`, `zind`, `z`) as well as the ground truth map and vehicle objects (`map`, `veh`). The known map is passed to `E1`, but the true trajectory is not – it is only given to `visualize` for comparison. If you saved the EKF object `ekf_l` from `E0` and pass it in to `ex3_ekf.m`, then it will also be used for comparison in `visualize`, so you can see the ground truth, your estimate, and the Robotics toolbox EKF estimate. It is okay if your estimate is different from the toolbox EKF estimate, but they should be similar.

Useful functions: `atan2`, `angdiff` – make sure you are using the Robotics toolbox version of `angdiff`, which is *different* from the one provided in the official Robotics system toolbox. Correct version:
`https://github.com/petercorke/spatial-math/blob/master/angdiff.m`

E2. **5 points.** Implement EKF-based mapping, given a known vehicle trajectory.

The given odometry readings `odo` are assumed to be accurate, unlike in the previous question. Note that the order of the landmarks in the state vector is dependent on when they are first observed in the path, therefore the state vector will be scrambled – use the `indices` vector to keep track of the alignment. Also, for the path taken, some of the landmarks will never be observed, therefore they will never appear in the state estimate.

If you are taking the undergraduate version of the course (CS 4610), you may stop here. However, the following questions will be assigned for Ex4, so you may continue to get a head start.

E3. **[CS 5335 only for Ex3.] 5 points.** Mathematically derive the Jacobians in EKF-SLAM, and then implement EKF-SLAM.

The exposition in the textbook is rather light on derivations, and in particular for SLAM, is not very explicit about the Jacobians to be used in the algorithm. On a separate written document, first derive the Jacobians used in EKF-SLAM. This includes the Jacobians for the transition function (F_x and F_v), the observation function (H_x and H_w), and the insertion Jacobian (denoted Y_z in the textbook). Think about the dimensions of each of these Jacobian matrices, and make sure your matrices have the correct shape.

Once you have explicitly derived the Jacobians, it should be fairly straightforward to transfer to code, using your code from `E1` and `E2` as the foundation.

E4. **[CS 5335 only for Ex3.] 0 points.** The trajectory and map found by SLAM in `E3` should have substantial uncertainty, although it is likely better than that found by the Robotics toolbox implementation of EKF-SLAM. If you observe the vehicle collecting data in `E0`, you can see the sensor in action, and may notice that many points along the true trajectory have no visible landmarks. We hypothesize that if the sensor had greater maximum range, the estimates will be more accurate and less uncertain.

Copy the code from `E0` for generating SLAM data into `E4` and run the toolbox EKF on it. The maximum range passed has been doubled from 4 to 8 units. This generates the same data as provided in `e3_r8.mat`, and the solution found in the EKF object should be significantly better than before. If you implemented `E3` correctly, no further implementation is necessary; `ex3_ekf.m` will run `E3` on the new data, and your estimate should once again be similar to the toolbox estimate.