CS 4610/5335: Robotic Science and Systems (Spring 2022) Lawson L.S. Wong
Northeastern University Due March 25, 2022

Exercise 4: Simultaneous 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.

Download the starter code (`ex4.zip`). In this file, you will find:

- `ex4_slam.m`: Main function. Call `ex4_slam(<questionNum>)` with an appropriate question number (0–9) 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 `ex4_slam.m`.)
- `E0.m` – `E9.m`: Code stubs that you will have to fill in for the respective questions.
  You should copy over your completed `E0.m` and `E1.m` (and `E2.m` for CS 5335 only) from Ex3.
- `e1.mat` – `e3.mat`, `e3_r8.mat`: Data files for each question.
- `parameters.m`: Parameters for EKF-SLAM.
- `visualize.m, visualize_histogram.m, visualize_path.m, visualize_map.m, average_error.m`: Helper functions for visualizing results and computing errors.

E0. **0 points.** Completed in Ex3. EKF objects returned by E0 can be used to check your answers in E1–E4.

E1. **0 points.** Completed in Ex3. Implement EKF-based localization, given a known map.

E2. **[CS 4610 only for Ex4; CS 5335 completed this already in Ex3.] 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.

E3. **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. **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; `ex4_slam.m` will run `E3` on the new data, and your estimate should once again be similar to the toolbox estimate.

E5. **2 points.** In order to further investigate SLAM, we need to have more control over how our data is generated, and generate it much faster than running code in E0. Implement a noisy "sensor" that takes in a true odometry sequence and true map, and produces noisy odometry readings and noisy range/bearing measurements of landmarks, where the amount of noise is determined by the V and W noise covariance matrices.

In this question, implement the 'o' mode of the noisy sensor: at each time step, if there are multiple visible landmarks (within the given maximum range and field of view), one is chosen uniformly at random to produce a noisy measurement. This corresponds to the behavior of the `RangeBearingSensor` in the toolbox. You may want to refer to the source code of `Vehicle` and `RangeBearingSensor` to see how noisy odometry readings and sensor measurements are generated by the toolbox.

Useful functions: `randn`, `sqrtm`

Hint: Note that the `zind` output is now a cell array instead of a vector. In future questions, you will implement more sensor modes, which may output more than one observation per time step.

Hint: If your noisy data generator is implemented correctly, the resulting behavior on SLAM (E3) will typically be similar compared to the data provided in `e3.mat`. However, since the noisy data is randomly generated, variation in performance is expected, so results will not be exactly the same, and sometimes not even close. You may want to run your code multiple times to see if the results are typically reasonable.

E6. **3 points.** In E3, the vehicle trajectory only detected half of the landmarks in the map, and some landmarks had large error/uncertainty in their positions. If we knew roughly where the landmarks were, we could potentially *design* trajectories that visited more landmarks, collected more data, and therefore produce better / more complete maps. Assume that we are using the same landmark map from before.

Design trajectories to achieve the following (write them in E6):

(a) Estimate all landmarks in the map, possibly with greater error than E3.

(b) Estimate at least as many landmarks as E3, but with less error than E3.

To ensure that your trajectories satisfy vehicle constraints, your true odometry readings should have maximum translation of 0.1 and maximum rotation of ± 0.0546 (these constraints are satisfied by the original trajectory `odo_m` in `e2.mat`). It is unlikely the same trajectory can satisfy both of the above requirements.

Useful functions: `repmat`

Hint: It may be easier to design the true odometry sequence, ensuring that it satisfies motion constraints, and use that to compute the true trajectory.

Hint: Since the data generated by E5 is noisy, the results of each run of SLAM (E3) will be different. To ensure that your trajectories are actually better than the original one, you should change `num_trials` in `parameters.m` to 100 or more and make sure the average values / error-statistic histograms are better.

It may also be possible to improve SLAM results by making changes to the map (the placement of landmarks).

(c) Based on the insights obtained from your experimentation above, if you can make one change to the map to improve SLAM, what would that change be? (You do not need to implement this, just explain.)

E7. **2 points.** Another way to improve SLAM is to obtain more observations.
Recall that the noisy sensor in the toolbox and E5 can only observe one visible landmark, even if more are visible (within maximum range and field of view).

(a) Go back to E5 and implement the 'a' mode, where *all* landmarks within range and field of view generate noisy measurements. This will result in at least as many observations compared to before.

(b) Next, copy your E3 code into E7, and modify it to process multiple observations per time step.

(c) This should lead to slightly better performance, but the error-statistic histograms will likely be quite similar to those from the 'o' mode. Why is that? Can you find different sensor parameters that lead to a more dramatic improvement in SLAM performance?

If you are taking the undergraduate version of the course (CS 4610), you may stop here. You are welcome and invited to complete the remaining questions and bonus questions, for extra credit.

E8. **[CS 5335 only.] 4 points.** Sometimes robots do not have the luxury of a sensor that returns both bearing and range. For example, if the robot is navigating using the stars (at night) or the sun (during the day), these landmarks only provide bearing information (no range). Implement *bearing-only* SLAM.

(a) When a new landmark is detected, we need to set its initial distribution. What should this (possibly non-Gaussian) initial distribution ideally be? (Remember that the sensor is still subject to maximum range and field of view limits.) Since we are restricted to Gaussian distributions, how should we set the initial mean and covariance matrix?

(b) How should the Jacobian matrices be set to achieve this initial Gaussian belief of a new landmark? You may want to try running bearing-only SLAM for a small number of time steps to ensure that the initial belief is set properly (e.g., stop after the new landmark is first detected and processed).
Hint: If the `x_est` and `P_est` cell arrays are shorter than T (= 1000 by default), then `visualize.m` will only display the partial trajectory and map estimates passed in.

(c) Copy your E7 code into E8 and modify it to perform bearing-only SLAM.

(d) Experiment with the parameters of the bearing sensor (its maximum range and field of view). What parameters work well? Do you observe any trends? Is a sensor that captures more information always better? Why or why not?

E9. **[CS 5335 only.] 4 points.** Similarly, some landmarks / sensors only provide range information (no bearing). For example, robots can perform indoor localization and mapping using the Wi-Fi signal strength from multiple access points; the signal strength only depends on range (unless using a directional antenna).
Implement *range-only* SLAM.

(a) When a new landmark is detected, we need to set its initial distribution. What should this (possibly non-Gaussian) initial distribution ideally be? (Remember that the sensor is still subject to maximum range and field of view limits.) Since we are restricted to Gaussian distributions, how should we set the initial mean and covariance matrix?

(b) How should the Jacobian matrices be set to achieve this initial Gaussian belief of a new landmark? You may want to try running range-only SLAM for a small number of time steps to ensure that the initial belief is set properly (e.g., stop after the new landmark is first detected and processed).

(c) Copy your E7 code into E9 and modify it to perform range-only SLAM.

(d) Experiment with the parameters of the range sensor (its maximum range and field of view). What parameters work well? Do you observe any trends? Is a sensor that captures more information always better? Why or why not?

(e) Which do you find better for SLAM, a bearing-only sensor or range-only sensor?

The following questions are bonus. They are completely optional and potentially time-consuming.

EC0. **From Ex3 KF(f); you may complete this if you did not already do so in Ex3.**
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.

EC1. How is the toolbox EKF different from yours in E1–E3? Inspect the source code of the toolbox EKF (type `EKF` in the MATLAB editor, right-click on it, and select "Open "EKF"").

What are the differences in the mathematical equations being used, and what differences in modeling assumptions do they imply? Which version is better?

EC2. So far, we have assumed that sensing errors add a small amount of Gaussian noise to the true values. Sometimes, errors can be far greater, for example a landmark that does not actually exist is detected, i.e., a false positive.

(a) Go back to E5 and implement the 'f' mode, where all landmarks within range and field of view generate noisy measurements, but occasionally some measurements get randomly flipped into false positives. For false positives, the measurement should be uniformly sampled from within the range and field of view.

(b) You can see the results of this new "sensor" by running `ex4_slam(10)`.
Demonstrate that a small number of false positive errors in the sensor data can drastically affect results from SLAM. What is the smallest false positive rate / number of errors you can find that leads to a clearly incorrect result? Show the example.

(c) Devise and implement methods to handle false positives in the data. Illustrate with examples.

EC3. So far, we have assumed that we know the identity of each detected landmark, i.e., data association is known. Demonstrate that a small number of data association errors in the sensor data can drastically affect results from SLAM. In some cases, the identity of landmarks (`zind`) is not provided, and we have to deduce data association from the sensor data itself. Devise and implement methods to handle unknown data association. Illustrate the effectiveness / limitations of your approach with multiple examples and histograms.

EC4. Implement particle-filter / sequential Monte-Carlo localization (Ch. 6.7) and Rao-Blackwellized SLAM (FastSLAM) (Ch. 6.5). Apply these to the data in `e3.mat`. Investigate the following:

(a) How will you aggregate the maps corresponding to the various particles?

(b) How does your output compare to the solution found by EKF-SLAM?

(c) How does the number of particles affect solution quality and computation speed?

(d) Can you devise scenarios where particle-filter localization will outperform EKF-based localization? Similarly, under what scenarios will Rao-Blackwellized SLAM outperform EKF-SLAM?

EC5. Implement pose-graph SLAM (Ch. 6.6). Apply this to the data in `e3.mat`. Compare with EKF-SLAM.