Coding Problem Set 5 - EKF SLAM

Joshua Mangelson (Modified from a lab by Ryan Eustice)

October 9, 2022

Purpose: The purpose of this lab is to give you hands on experience implementing and the ability to play around with landmark-based EKF SLAM.

Setup and Initial Instructions

Pulling the Code

You will be able to access the code for each of the labs in this course via Git. For this lab specifically, you will be able to access it via the following link: `https://bitbucket.org/byu_mobile_robotic_systems/lab5-ekf-slam`

Submission Instructions

Your assignment must be received by 11:55p on the deadline listed on LearningSuite. You are to upload your assignment directly to LearningSuite with the following three attachments:

1. A .tgz or .zip file *containing a directory* named after your netid with the structure shown below: `alincoln_lab5.tgz`:
   ```
   alincoln_lab5/
   alincoln_lab5/run.py
   alincoln_lab5/simSLAM.py
   alincoln_lab5/vpSLAM.py
   alincoln_lab5/da_known.py
   alincoln_lab5/da_nn.py
   alincoln_lab5/da_nndg.py
   alincoln_lab5/da_jcbb.py
   alincoln_lab5/detectTrees.py
   alincoln_lab5/generator.py
   alincoln_lab5/helpers.py
   alincoln_lab5/fieldSettings.py
   alincoln_lab5/__init__.py
   alincoln_lab5/video_task1.{avi,mp4,gif}
   alincoln_lab5/video_task2.{avi,mp4,gif}
   alincoln_lab5/video_task3.{avi,mp4,gif}
   ```
2. A PDF with the written portion of your writeup. Scanned versions of hand-written documents, converted to PDFs, are perfectly acceptable. No other formats (e.g. .doc) are acceptable. Your PDF file should adhere to the following naming convention: alincoln_lab5.pdf.
3. A 1 minute video of yourself demonstrating your completed lab and your understanding of the topics explored. Your video should adhere to the following naming convention: alincoln_lab5_summary.{avi,mp4}

Lab Overview

This assignment has three tasks. These tasks will give you experience implementing EKF-SLAM with and without data association in simulation and on a real-world dataset.

Code

Below you will find descriptions of the files included in the given git repository. You may end up not using every single file. Some are utilities for other files, and you don't really need to bother with them. Some have useful utilities, so you won't have to reinvent the wheel. Some have fuller descriptions in the files themselves.

Things to implement

- `simSLAM.py` – This file is a modified version of the localization simulator originally used in Lab4. You now receive multiple range/bearing/markerId tuples as measurements on each time step. The odometry model remains the same as in Lab4. You will need to implement multiple functions in this file including the `predict()`, `update()`, and `augmentState()` functions. You will also need to modify or complete some parts of the `run()` function.
- `vpSLAM.py` – This file implements the Victoria Park SLAM dataset. Historically, this dataset was often used in the research community for benchmarking the performance of new SLAM algorithms. This dataset is *real* data—it contains recorded velocity and steering angle information of a pickup truck instrumented with a 2D laser scanner as it drove around a public park in Sydney (Figure 1). The park has many trees and open space, and the laser scanner detected tree trunks to be used for perception. These detections will serve as point features in your SLAM implementation. Here's a URL where you can learn more: `http://www-personal.acfr.usyd.edu.au/nebot/victoria_park.htm`. You will need to implement multiple functions in this file including the `predict()`, `update()`, and `augmentState()` functions. You will also need to modify or complete some parts of the `run()` function.
- `da_known.py` – You will need to implement the `associateData()` function inside this file to associate landmarks (just based on the signature number already passed along with them).
- `da_nn.py` – You will need to implement the `associateData()` function inside this file to associate landmarks (using the nearest neighbor algorithm).
- `da_nndg.py` – You will need to implement the `associateData()` function inside this file to associate landmarks (using the nearest neighbor double gated algorithm).
- `da_jcbb.py` – You will need to implement the `associateData()` function inside this file to associate landmarks (using the joint compatibility branch and bound algorithm).

Other provided files

- `run.py` – The main function for the lab. Calls the run functions from `simSLAM.py` and `vpSLAM.py`.
- `detectTrees.py` – Detects trees from the VP laser data. You will not need to modify this file.
- `fieldSettings.py` – Stores the field settings for the simulated dataset. You will not need to modify this file.
- `generator.py` – Generates the simulated dataset as in Lab 4. You will not need to modify this file.
- `helpers.py` – Contains some useful helper functions. Take a look to see if there is anything useful to you.
- `testBench.py` – Useful to test individual functions if you desire.

Task 1: Simulator, Known Data Association

In this task, you will implement EKF SLAM for point features. Your robot is driving around an environment obtaining observations to a number of landmarks. The position of these landmarks is not initially known, nor is the number of landmarks. For now, we'll simplify the problem: when the robot observes a landmark, you know which landmark it has observed (i.e., you have perfect data association).

For this problem, your landmark observations are $[range, bearing, markerId]$ tuples. Your robot state is $[x, y, \theta]$ (cm, cm, radians) and the motion control is $[\delta_{rot1}, \delta_{trans}, \delta_{rot2}]$ (radians, cm, radians). To run 200 time steps of the simulation with known data association execute `./run.py -t sim -n 200 -d known` at the command line.

You are encouraged to implement the Kalman equations in the simplest and most literal way possible: don't worry about computational or memory efficiency.

1. Write out your process and measurement models. You may use your models from Lab4, but your measurement model will now have components for both the robot and a feature. In your implementation, your state vector and covariance will grow in the augment step. Write out the steps and model matrices for augmentation of a new feature to your state vector from an observation.

2. You will need to modify/implement functions in `simSLAM.py` and `da_known.py`. Your code should be easy to read and well-documented.

3. In your writeup, include a plot of the final map showing 3-σ error ellipses for the feature positions and also the true beacon locations. Generate a movie of your SLAM filter operating, with each frame showing the position and (x, y) uncertainty of the robot and all the landmarks.

   Note: The lab has some built in options for generating GIFs from your visualizations.

4. The paper "A Solution to the SLAM Problem" by Dissayanake et al. (IEEE TRA, 2001) examines the properties of the SLAM covariance matrix and the way in which the correlation coefficients in the covariance matrix evolve over time. Specifically, it shows that the correlation of all landmarks increases with time and that the uncertainty of a landmark only decreases after initialized. Use your results to verify the claims in this paper. Graph the correlation cofficients between the x coordinates of the feature locations for a number of map elements. Also graph the correlation coefficient between the robot x position and the x position of several map elements. Finally, graph the determinant of the feature covariance matrix for all map features. Are the claims in the paper by Dissayanake et al. (IEEE TRA, 2001) validated? Comment on the title of the Dissayanake paper (does this constitute a "solution" to SLAM? What aspects of the general SLAM problem are not accounted for in their analysis?)

5. Implement batch updates for measurements. For a given observation step, all simultaneous measurements are processed together by stacking the innovations and Jacobians, and using a block diagonal measurement noise matrix.

   If you already implemented batch updates in step 2, implement sequential updates. Do you notice any changes in algorithm stability? Why might batch updates be preferable?

Task 2: Simulator, Unknown Data Association

For this problem, our landmark observations are $[range, bearing]$, we will ignore the fact that the simulator gives us `landmarkSignatures`. Hence, our algorithm must infer *which* landmarks in the environment generated the set of measurements at each time step.

1. You will need to modify/implement functions in `simSLAM.py`, `da_nn.py`, and `da_nndg.m`. Your code should be easy to read and well-documented.

2. Rerun your simulation and use incremental maximum likelihood data association (i.e., nearest neighbor in a Mahalanobis sense) in file (`da_nn.py`) to infer the landmark correspondences. In your writeup, include a plot of the data associations inferred by your algorithm as compared to the ground-truth `landmarkSignatures` known by the simulator. Are there any discrepancies? If so, how may this impact your results?

3. Now implement data association using the double gated nearest neighbor (`da_nndg.py`) method described in the slides where two thresholds are used and measurements between those two thresholds are ignored. Regenerate the plots from the last step and include them in your lab writeup. How do these results differ from the basic nearest neighbor algorithm?

4. Generate a movie of your SLAM filter operating, with each frame showing the position and (x, y) uncertainty of the robot and all the landmarks.

5. Repeat the task above but now for different landmark densities and maximum number of simultaneous observations. You can increase the maximum number of observations per time step and the default number of landmarks in the environment by changing the parameters `maxObs` in the `simSLAM.py` file and `numMarkersX` in the `fieldSettings.py` file. As the landmarks become more cluttered, how does nearest neighbor data association perform? How does the double gated method perform? How might this impact your SLAM results?

Task3: Victoria Park

For this problem, we tackle the Victoria Park benchmark dataset. Our landmark observations are $[range, bearing]$ detections of tree trunk point features. Our robot state is $[x, y, \theta]$ (m, m, radians) and the motion control is $[v_e, \alpha]$ (m/s, radians). To run 1000 time steps of the dataset execute `./run.py -t vp -n 1000 -d nndg` at the command line. For this dataset, it is likely that nearest-neighbor landmark correspondence may not perform well and you may want to consider implementing joint-compatibility branch and bound in the file `da_jcbb.py`.

The previously given URL contains links to vehicle and landmark observation models for this dataset. The noise parameters in the `vpSLAM.py` should be a good starting point, but you can increase them if you think you should. Note: The online documentation contains two typos in the process model, it has a $\tan(\phi)$ where a $\tan(\alpha)$ should be. The correct process model is

$$\begin{bmatrix} x[k+1] \\ y[k+1] \\ \phi[k+1] \end{bmatrix} = \mathbf{f}(\mathbf{x}[k], \mathbf{u}[k]) + \mathbf{w}_f[k] = \begin{bmatrix} x[k] + \Delta T \left(v_c[k] \cos\phi[k] - \frac{v_c[k]}{L} \tan\alpha[k] \left(a \sin\phi[k] + b \cos\phi[k] \right) \right) \\ y[k] + \Delta T \left(v_c[k] \sin\phi[k] + \frac{v_c[k]}{L} \tan\alpha[k] \left(a \cos\phi[k] - b \sin\phi[k] \right) \right) \\ \phi[k] + \Delta T \frac{v_c[k]}{L} \tan\alpha[k] \end{bmatrix} + \mathbf{w}_f \quad (1)$$

where $\mathbf{u}[k] = \left[v_c[k], \alpha[k] \right]^\top \sim \mathcal{N}(\mu_u, \Sigma_u)$ is the noisy (measured) velocity and steering angle used as a pseudo control input, and $\mathbf{w}_f[k] = \left[w_x[k], w_y[k], w_\phi[k] \right] \sim \mathcal{N}(0, \Sigma_f)$ is a fictitious additive white noise term used to account for the inaccuracy of our simple Ackerman steering model.

1. You will need to modify/implement functions in `vpSLAM.py` and optionally `da_jcbb.m` (10% extra credit). Your code should be easy to read and well-documented.
2. In your writeup, include a plot of the final map showing 3-σ error ellipses for the feature positions. Generate a movie of your SLAM filter operating, with each frame showing the position and (x, y) uncertainty of the robot and all the landmarks. Run your filter for 500 time steps.
3. Since the VP dataset potentially contains hundreds of features, this is a good opportunity to gain some practical appreciation for what it means for an algorithm to have $\mathcal{O}(n^2)$ complexity (i.e., the quadratic computational complexity of EKF SLAM inference.) Using the python TIME.TIME() function, record the amount of CPU time spent at each prediction and update step. Make a plot versus time of the prediction delta CPU time, update delta CPU time, and number of landmarks in your map at each iteration.
4. (Optional) Overlay your trajectory and map on a Google Earth image of Victoria Park. You can use `http://www.gpsvisualizer.com/` for this.

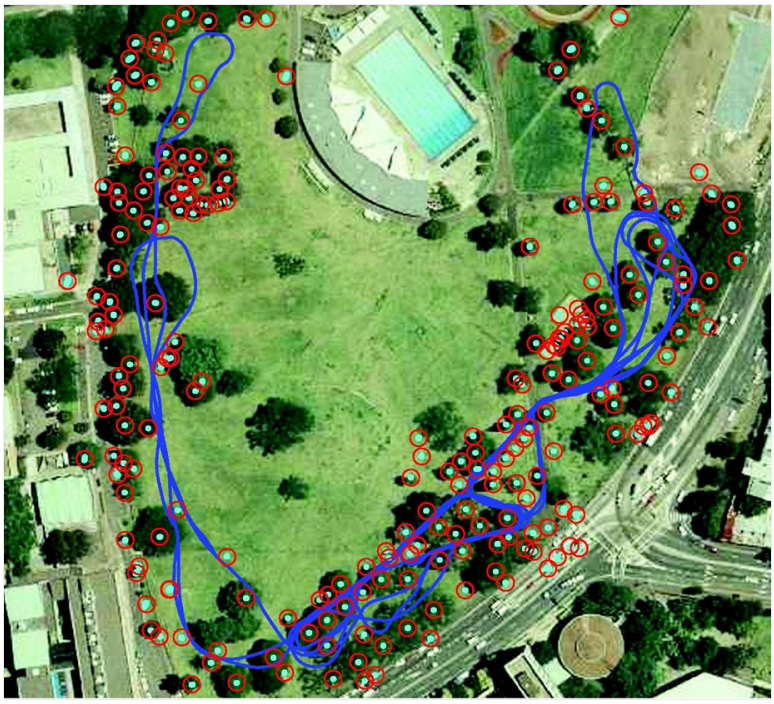

Figure 1: Victoria Park experimental platform and EKF SLAM result. (courtesy E. Nebot)