NAVARCH/EECS 568, ROB 530 Mobile Robotics: Methods & Algorithms (W2018) PS2

Maani Ghaffari
University of Michigan
Courtesy: R. Eustice

February 18, 2018

You are encouraged to talk at the conceptual level with other students, but you must complete all work individually and may not share any non-trivial code or solution steps. See the syllabus for the full collaboration policy.

Submission Instructions

Your assignment must be received by 11:55p on Monday, March 19th. You are to upload your assignment directly to the Canvas website as two attachments:

1. A `.tar.gz` or `.zip` file *containing a directory* named after your uniqname with the structure shown below.

```
alincoln_ps2.tgz:
alincoln_ps2/
alincoln_ps2/runsim.m
alincoln_ps2/runvp.m
alincoln_ps2/ekfpredict_sim.m
alincoln_ps2/ekfpredict_vp.m
alincoln_ps2/ekfupdate.m
alincoln_ps2/initialize_new_landmark.m
alincoln_ps2/da_known.m
alincoln_ps2/da_nn.m
alincoln_ps2/da_jcbb.m
alincoln_ps2/video_task1.{avi,mp4}
alincoln_ps2/video_task2.{avi,mp4}
alincoln_ps2/video_task3.{avi,mp4}
```

2. A PDF with the written portion of your write-up. 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_ps2.pdf`.

Homework received after 11:55p is considered late and will be penalized as per the course policy. The ultimate timestamp authority is the one assigned to your upload by Canvas. No exceptions to this policy will be made.

Important: For all (x, y) plots you will want to use the `axis equal` command to set the aspect ratio so that

equal tick mark increments on the x-,y- and z-axis are equal in size. This makes `SPHERE(25)` look like a sphere, instead of an ellipsoid.

Code

To begin, you will need to download `ps2-code.zip` from the Canvas course website. This `.zip` file contains some Matlab m-files and two subdirectories:

i) `slamsim` is a modified version of the localization simulator originally used in PS1. The number of landmarks is now configurable and multiple range/bearing/markerId tuples are produced at each time step. The odometry model remains the same as in PS1.

ii) `vicpark` contains 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 (Fig. 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`

Below you will find descriptions of the files included in the zip file. 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

- `slamsim/runsim.m` – Main loop for simulator; should call `ekfpredict_sim()` and `ekfupdate()`
- `vicpark/runvp.m` – Main loop for Victoria Park dataset; should call `ekfpredict_vp()` and `ekfupdate()`
- `ekfpredict_sim.m` – EKF prediction for simulator process model
- `ekfpredict_vp.m` – EKF prediction for Victoria Park process model
- `ekfupdate.m` – EKF landmark observation update; used by both simulator and Victoria Park
- `initialize_new_landmark.m` – augments the state vector with a new landmark element, called by `ekfupdate()`
- `da_known.m` – performs known data association, called by `ekfupdate()`, can be used in simulator
- `da_nn.m` – performs incremental maximum likelihood data association, called by `ekfupdate()`, can be used in simulator and Victoria Park
- `da_jcbb.m` – performs joint-compatibility branch and bound data association, called by `ekfupdate()`, can be used in simulator and Victoria Park

slamsim (essentially, the same files from PS1)

- `slamsim/deg2rad.m` – degrees to radians
- `slamsim/generateMotion.m` – simulates simple motion commands
- `slamsim/generateScript.m` – generates data according to initial mean and noise parameters
- `slamsim/prediction.m` – move robot according to specified motion
- `slamsim/observation.m` – generates noise free observation model
- `slamsim/sampleOdometry.m` – implements Table 5.6
- `slamsim/sample.m` – samples from a covariance matrix
- `slamsim/meanAndVariance.m` – returns the mean and variance for a set of non-weighted samples (illustrates handling of angles)
- `slamsim/getfieldinfo.m` – gets field information
- `slamsim/minimizedAngle.m` – normalizes an angle to $[-\pi, \pi]$
- `slamsim/plotcircle.m` – draws a circle
- `slamsim/plotcov2d.m` – draws a 2-D covariance matrix

- `slamsim/plotfield.m` – draws the field with landmarks
- `slamsim/plotmarker.m` – draws a dot at a specified 2D point (useful for plotting samples)
- `slamsim/plotrobot.m` – draws the robot
- `slamsim/plotSamples.m` – plots particles from the PF

vicpark (files for Victoria Park dataset)

- `vicpark/aa3_dr.mat` – dead-reckoned vehicle data
- `vicpark/aa3_gpsx.mat` – GPS data for ground-truthing
- `vicpark/aa3_lsr2.mat` – 2D laser scanner data
- `detectTreesI16.m` – routine for detecting tree-trunk feature points from laser data
- `load_vp_si.m` – function that loads and converts the Victoria Park `.mat` files into a single structure variable called `Data`
- `plotbot.m` – draws the robot as a triangle
- `viewLsr.m` – script for graphically displaying the laser data

Task 1: Simulator, Known Data Association (50 points)

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 $[\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}}]$ (radians, cm, radians). To run 200 time steps of the simulation execute `run(200,'sim')` 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. (10 pts) Write out your process and measurement models. You may use your models from PS1, 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. (15 pts) You will need to modify/implement functions for `runsim.m`, `ekfpredict_sim.m`, `ekfupdate.m`, `initialize_new_landmark.m`, and `da_known.m`. Your code should be easy to read and well-documented.

3. (10 pts) 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: Follow the example code provided to you in PS1 `run.m` for generation of the movie.

4. (8 pts) 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. 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?

5. (7 pts) 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 (25 points)

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

1. (10 pts) You will need to modify/implement functions for `runsim.m`, `ekfupdate.m`, and `da_nn.m`. Your code should be easy to read and well-documented.
2. (10 pts) Rerun your simulation and use incremental maximum likelihood data association (i.e., nearest neighbor in a Mahalanobis sense) 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 `markerIds` known by the simulator. Are there any discrepancies? If so, how may this impact your results? 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.
3. (5 pts) 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 `Param.maxObs` and `Param.nLandmarksPerSide` in the `runsim.m` file. As the landmarks become more cluttered, how does nearest neighbor data association perform? How might this impact your SLAM results?

Task3: Victoria Park (25 points)

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(1000,'vp')` at the command line. To run the whole dataset, execute `run(inf,'vp')`. 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.m`.

The previously given URL contains links to vehicle and landmark observation models for this dataset. The noise parameters in the `runvp.m` should be a good starting point. 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

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

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. (10 pts) You will need to modify/implement functions for `runvp.m`, `ekfpredict_vp.m`, `ekfupdate.m`, and optionally `da_jcbb.m` (10% extra credit). Your code should be easy to read and well-documented.
2. (10 pts) 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. (5 pts) 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 MATLAB's 'tic' and 'toc' commands, 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.

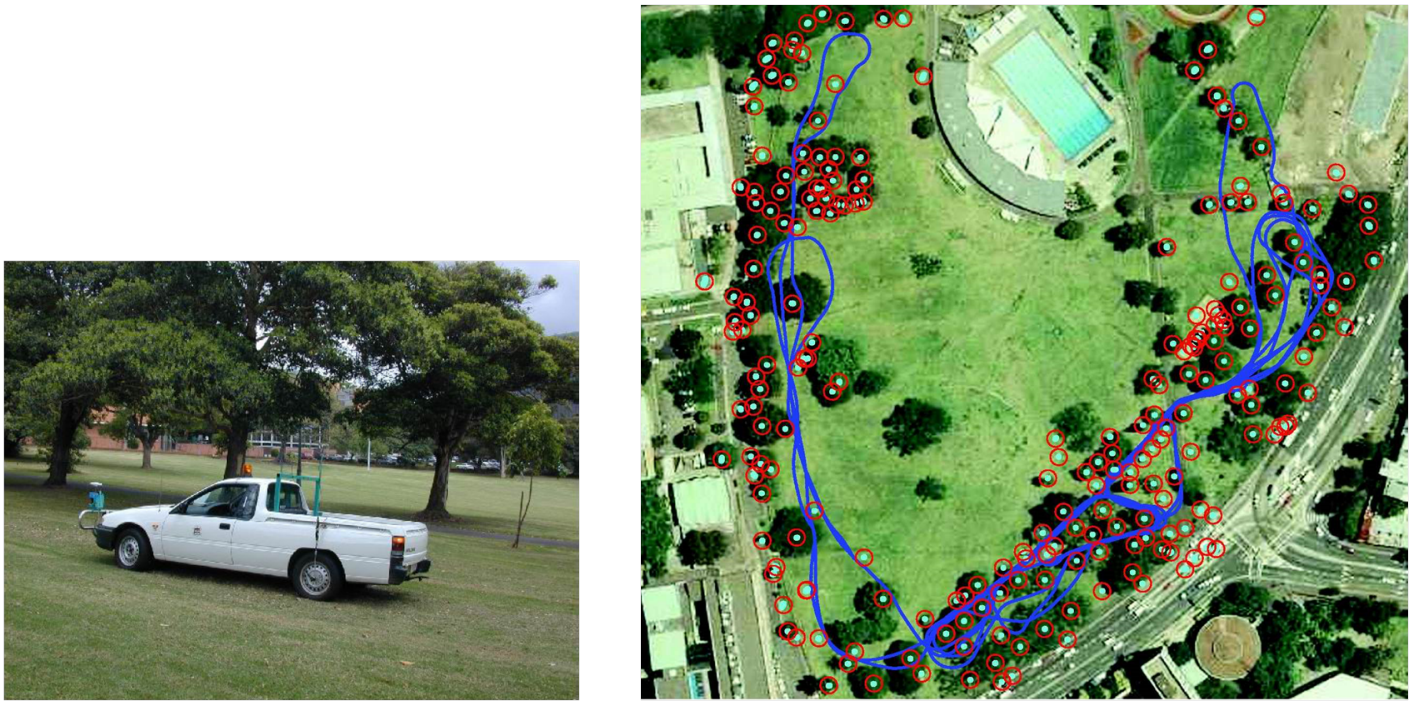

Figure 1: Victoria Park experimental platform and EKF SLAM result. (courtesy E. Nebot)