Perception in Robotics
Term 2, 2018. PS3

Gonzalo Ferrer
Skoltech

November 27, 2018

This problem set is a single task comprising 10% of your course grade, and it is to be done individually. You are encouraged to talk at the conceptual level with other students, discuss the equations and even the results, but you may not show/share/copy any non-trivial code.

Submission Instructions

Your assignment must be received by 11:59p on Monday, December 10th. You are to upload your assignment directly to the Canvas website as two attachments:

1. A PDF with the written portion of your document, solving the tasks proposed below. 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_ps3.pdf`.
2. A `.tgz` or `.zip` file *containing a directory* named after your uniqname with the structure shown below.

```
alincoln_ps3.tgz:
alincoln_ps3/run.py
alincoln_ps3/field_map.py
...
alincoln_ps3/slam/* //new files created by you
alincoln_ps3/ekf.{avi,mp4}
alincoln_ps3/sam.{avi,mp4}
alincoln_ps3/ekf_da.{avi,mp4}
alincoln_ps3/sam_da.{avi,mp4}
```

Homework received after 11:59p 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.

SLAM problem set

Code

To begin, you will need to download `ps3-code.zip` from the Canvas course website. This `.zip` file contains some Python files. The project is a modified version of the localization simulator originally used in PS2. 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 PS2.

Below you will find descriptions of the files included in the zip file. You may end up not using every single function. 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

- Include all those necessary calls on the `run.py` file to update and correct the SLAM problem. Plotting is also necessary.
- Implement an online SLAM algorithm using EKF with known correspondences and for unknown correspondences, maximum likelihood data association.
- Implement the square root SAM with known correspondences and unknown.

Utilities (essentially the same from PS2)

- `README.md` – Some commands examples for installing the environment, testing and evaluating the task.
- `run.py` – Main routine, with multiple options, allowing you to solve the task with no need to modify the file.
- `field_map.py` – for plotting the map.
- `tools/task.py` – General utilities available to the filter and internal functions
- `tools/data.py` – Routines for generating, loading and saving data .
- `tools/objects.py` – Data structures for the project
- `tools/plot.py` – All utilities for plotting data.
- `slam/slamBase.py` – An abstract base class to implement the various SLAM algorithms.

Task 1: Simulator, Known Data Association (70 points)

In this task, you will implement EKF SLAM and $\sqrt{\text{SAM}}$ for landmarks. 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).

You are encouraged to implement the Kalman and $\sqrt{\text{SAM}}$ equations in the simplest and most literal way possible: don't worry about computational or memory efficiency.

1. (30 pts) Implement the functions for `efkPrediction` and `ekfUpdate`, `initializeNewLandmark`, and `daKnown`. Here, you have freedom to create new files and modify existing ones (mainly `run.py`) if it is required. Your code should be easy to read and well-documented. In the document indicate which files you have created or modified. Also, 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) 3-sigma uncertainty of the robot and all the landmarks.

   Note: Follow the example code provided to you in PS2 `run.py` for generation of the movie `python run.py -s -i slam-evaluation-input.npy -m ekf`

2. (5 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?

3. (10) Generate the adjacency matrix A and the vector of residuals b for any given number of observations. Include in your document a plot of A after executing the full evaluation sequence, by using the `plt.spy` function.

4. (25) Choose one of the methods explained in class for solving the $\sqrt{\text{SAM}}$ problem (Cholesky, QR or Schur) and solve the problem at each iteration. Efficiency here is not an issue, so for this task reordering is not necessary nor the use of sparse matrices. In your document, 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 same as described before.

   Note: Do not use direct solvers from linear algebra libraries. We are asking you to call a factorization method, implement back-substitution and update of your solution.

Task 2: Simulator, Unknown Data Association 30 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. (15 points) You will need to modify/implement functions for the data association problem in the EKF SLAM. Your code should be easy to read and well-documented. Rerun your simulation and use maximum likelihood data association 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) 3-sigma uncertainty of the robot and all the landmarks.

2. (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 `–num-landmarks-per-side` and `–max-obs-per-time-step` when calling `run.py`. As the landmarks become more cluttered, how does nearest neighbor data association perform? How might this impact your SLAM results?

3. (10 pts) Propose and implement a version of the ML data association for the Factor Graph SLAM (SAM) algorithm. Remember that here we can undo wrong associations and many different techniques/heuristics could be used, so this solution is open. Generate a video with the results.