TTK4250 Sensor Fusion

Graded Assignment 3

Code hand in (5%): *Monday 22. November 16:00* in Blackboard as the zip file created by running `generate_handin.py`.
Report hand in (17.5%): *Monday 22. November 16:00* in Blackboard as a single PDF file.

Groups of two or three: This assignment is to be solved groups of two or three. These groups will be the same for both the graded assignments.

Code hand in: Hand in the zip file created by running `generate_handin.py`. Points will be withdrawn if you hand in anything more or anything less.

Report hand in: A report is to be handed. You are limited to 1000 words and a total of 6 pages (1 frontpage + 6 pages of content + 1 page of bibliography). The extra space is intended to allow you to have larger figures and more space between them.

Code content:

The score on the code will be based on the number of tests passed. If your code passes all the handed out tests, you should receive the full 5%. If you notice the tests having false negatives, false positives, or behaving weirdly, please let us know.

Report content:

As the report is quite short, you should *consider carefully* what you include. You can simply refer to equations and algorithms in the book when what you want to say is already stated there.

In task 2 and 3 you will be asked to try out different sets of tuning values. The following list should not be completed for every set of tuning values, but in total we want to see the following from running on the simulated data

- plot of trajectory and map.
- position estimate error over time
- plot of the NIS along with your chosen confidence bounds over time,
- plot of pose NEES along with your chosen confidence bounds over time,
- the averaged NIS along with your confidence bound, where the average can for instance be over number of associated landmarks in total,
- the averaged NEES along with your confidence bounds,
- the number of times NIS falls within your confidence bounds
- the number of times the NEES fall within your confidence regions
- and the RMSE (mean taken over time)
- (Optional) NEES of the map and total eta.

From running on real data this list reduces to

- plot of trajectory and map.
- position vs GPS distance over time.
- plot of the NIS along with your chosen confidence bound over time,
- the averaged NIS along with the confidence bound,
- the number of times NIS falls within your confidence bounds
- (Optional) show GPS in a NIS or NEES

We expect you to discuss what these plots and numbers mean in practice, and how they are connected to the different tuning parameters.

Answers and analysis should connect theory and results to the real world, and show your understanding of the problem and the solution. Theoretical or practical insight from the lectures or book elaborated upon or applied to the data sets counts positive.

Points can also be given for other plots, numbers, and considerations, but what to show is *up to you* to decide. We also invite you to be critical about the assumptions made and the implementation of the system. Try to connect the results on the simulated data to the results of the real data where it is possible.

We know the task description can be a bit vague, but we don't want you to end up writing different copies of the same report. Rest assured that two very different reports can both receive a top grade.

Some background on template code

A skeleton and template code is handed out to give you a starting point to write your code. The files handed out are:

- `utils.py`: The `wrapToPi` function that wraps an angle to $[-\pi, \pi)$ and `rotmat2d` function that forms the 2×2 rotation matrix that transforms from body frame to world frame, given a yaw angle ψ.
- `vp_utils.py`: Utility functions for real SLAM on Victoria Park dataset. The `detectTrees` function converts raw LiDAR measurements into range-bearing measurements. The function `odometry` calculates the odometry vector given by (11.4) from sensor measurements. The Car class is a convenience class for storing car parameters. **NB:** you do not need to do anything in this file.
- `JCBB.py`: The JCBB algorithm for data association.
- `plotting.py`: The ellipse function for drawing covariance ellipses.
- `EKFSLAM.py`: The `EKFSLAM` class skeleton, and related methods.
- `run_simulated_SLAM.py`: Template for running and analyzing the `EKFSLAM` methods on simulated data.
- `run_real_SLAM.py`: Template for running and analyzing the `EKFSLAM` methods on the Victoria Park data set.

Task 1: *Implement EKFSLAM*

(a) Implement `def f`, the motion prediction model, corresponding to equation (11.7) that takes a pose x and odometry u predicts it to the next time step,

(b) Implement `def Fx`, the motion prediction Jacobian with respect to x, that is the Jacobian of the above function with respect to the pose, given by equation (11.13).

(c) Implement `def Fu`, the motion prediction Jacobian with respect to u, that is the Jacobian of the above function with respect to the odometry, given by equation (11.14).

(d) Implement `def predict`, which takes the state η (pose and map), its covariance P and odometry u to predict the state and covariance, given by (11.18).

Note: The map is assumed static. This means that you only need to do something with the state and covariance that has to do with the pose. There is a huge save in computation if you take this into consideration for the covariance matrix when the problem gets large, and also do it in place (ie. reuse the input covariance matrix). This should also be evident from F in (11.17).

(e) Implement `def h`, the measurement function, that predicts the measurements of all the landmarks in the state, corresponding to equations (11.10) – (11.11).

Note that there is a offset between the robot center and the sensor location in the Victoria Park data set making us having to use $m^i - \rho_k - R(\psi_k)L$ instead of simply $m^i - \rho_k$, where L is the offset vector in body frame.

(f) Implement `def h_jac` the jacobian of the above measurement function with respect to η, given by equations (11.15) – (11.17). This matrix is dense in the three leftmost columns corresponding to the pose, and block diagonal for the landmarks.

We need to compensate for the sensor offset in the jacobian as well. Using $\frac{\partial}{\partial x}||x|| = \frac{x^T}{||x||}$, and $\frac{\partial}{\partial x}\angle(x) = \frac{x^T R(\pi/2)^T}{||x||^2}$ along with $z_c = m^i - \rho_k - R(\psi_k)L$ and $z_b(x) = R(-\psi)z_c = R(-\psi)(m^i - \rho_k) - L$, it can be shown that the measurement Jacobians can be written as

$$\frac{\partial}{\partial x}||z_b(x)|| = (\frac{\partial}{\partial z_b}||z_b(x)||)(\frac{\partial}{\partial x}z_b(x)) = \frac{z_c^T}{||z_c||}\begin{bmatrix}-I_2 & -R(\frac{\pi}{2})(m^i - \rho_k)\end{bmatrix},$$
$$\frac{\partial}{\partial x}\angle z_b(x) = (\frac{\partial}{\partial z_b}\angle(z_b(x)))(\frac{\partial}{\partial x}z_b(x)) = \frac{z_c^T R(\frac{\pi}{2})^T}{||z_c||^2}\begin{bmatrix}-I_2 & -R(\frac{\pi}{2})(m^i - \rho_k)\end{bmatrix}.$$

(g) Implement `def add_landmarks`, that inverts the measurement function and creates new landmarks and their covariances in the function.

The covariances are generated using

$$G_x^j = \frac{\partial}{\partial x}h^{-1}(z,x) = \begin{bmatrix} I_2 & z_r^j \begin{bmatrix}-\sin(z_\phi^j + \psi) \\ \cos(z_\phi^j + \psi)\end{bmatrix} + R(\psi + \frac{\pi}{2})L \end{bmatrix}$$
$$G_z^j = \frac{\partial}{\partial z}h^{-1}(z,x) = R(z_\phi^j + \psi)\mathrm{diag}(1, z_r^j)$$
$$P_{\text{new}} = \begin{bmatrix} P & P_{:,x}\{G_x^j\}^T \\ \{G_x^j\}P_{x,:} & \{G_x^j\}P_{xx}\{G_x^j\} + \mathrm{blkdiag}(G_z^j R(G_z^j)^T)\end{bmatrix}$$

(h) Implement `def update`, that takes the prior mean and covariance and a set of measurements to update the map and pose estimates as well as calculating NIS and creating new landmarks. The data association is done for you so you only need to make make the EKF update.

Task 2: *Run EKFSLAM on simulated data*

You are given a data set consisting of

- odometry (K, 3): the measured movement of the robot in body frame
- z (K,): Python list of detections at each time step. Each detection is (#measurements, 2).
- poseGT (K, 3): the pose ground truth at each time step
- landmarks (M, 2): the ground truth landmarks

We have given you a set of pretty good initial tuning values (marked with `# TODO tune`). We want you to find 3 different sets of tuning values that differ in different ways and explain the results you get using them.

An example of three different sets of tuning values could be

- a set that gives better performance than the initial tuning values in some way
- a set that gives similar performance in terms of error but bad performance in terms of consistency
- a set that yields very few landmarks, but still manages to keep track to some degree

Note that NIS has varying degrees of freedom over time. The confidence intervals are therefore varying as well. We therefore propose to normalize the plots by dividing by either the amount of associated landmarks or the number of degrees of freedom in each time step.

Some questions to help discussion (not mandatory to answer):

- Which types of errors you see?
- Why do think the errors are like this?
- Would you say that there are any shortcomings on the algorithm on this data set?

Note: The runtime can be quite sensitive to the alphas of JCBB. This is due both to the depth of search and creation of new landmarks, creating sort of a tradeoff. This is especially true when other poor parameters are chosen, so that too few or too many landmarks fit the prediction. This is usually seen quite quickly by the associations, and how many new landmarks are created.

Task 3: *Run EKFSLAM on the Victoria Park dataset*

The Victoria Park data set is a SLAM data set available along with some additional information here [1]. All times are converted to seconds for you in the script, and a function to create odometry and detections from the data are also provided.

The dataset consists of

- LASER (mK, 361): Laser scanner returns. There are 361 returns per scan evenly divided in a half circle.
- speed (K,): measured speed from a wheel encoder at the rear left wheel.
- steering (K,): measured wheel steering angle
- Lo_m (Kgps,): GPS longitude in meters
- La_m (Kgps,): GPS latitude in meters
- timeLsr (mK,): Time of laser scans in milliseconds
- timeOdo (K,): Time of odometry measurements in milliseconds

[1] http://www-personal.acfr.usyd.edu.au/nebot/victoria_park.htm

- timeGps (Kgps,): Time of GPS measurements in milliseconds

Similar as in the previous task we have given you pretty good initial tuning values (marked with `# TODO tune`). In the same way as previously, we want you to find 3 different sets of tuning values that differ in different ways and explain the results you get using them.

See the linked webpage and the files in the Victoria Park folder for further information. The same reasoning applies to NIS and tuning here as in the simulated dataset.

Note that you can use the GPS as a sort of ground truth, but it is very much less than perfect. You might want to use a NIS calculation instead of NEES calculation if you do that. Some comparison to the GPS is, however, expected.

Some questions to help discussion (not mandatory to answer):

- Which types of errors you see?
- Is this different on the real data set compared to the simulated one?
- Why do think this is so?
- Would you say that there are any shortcomings on the algorithm on this data set?
- What is your general take on the EKFSLAM algorithm on data like this?
- Can you think of any improvement strategies?