TTK4250 Sensor Fusion

Graded Assignment 3

Code and results hand in (5%): *Wednesday 11. November 16.00* on Blackboard as a single zip file.
Report hand in (10%): *Friday 13. November 16.00* on Blackboard as a single PDF file.

Pairs: This assignment is to be solved in pairs. These pairs will be the same for all the graded assignments.

Code hand in: Functioning python code that produces your results for task 3, without the data. This will account for one third of the possible points for this assignment (5% of total grade).

Report hand in: A 4 page (in total!) report is to be handed in by 13.November 16.00 as a PDF on Blackboard. Points will be withdrawn for reports longer than 4 pages. As the report is quite short, you should *consider carefully* what to have in it.

Report content: The algorithms are given in the book or this assignment, so you can simply refer to the book when what you want to say is already stated there. For task 2, we want to see

- 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.

for a selected parameter set. For task 3 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

due to the lack of ground truth.

We then expect you to discuss what these number and parameters means in practice, and what is gained or lost by changing the parameters and why. Points can also be given for other plots, numbers and considerations, but what to show is *up to you* to decide. Answers and analysis should connect theory and results to the real world, and show your understanding for the problem and solution. Try to connect the results on the simulated data to the results of the real data where it is possible.

To help tuning you can for instance split the NEES/NIS into the different axes and/or look at the biases, size and whiteness of the error or innovation sequence.

Note:

Again, this is not supposed to be a "try one million parameter sets" assignment. You are supposed to show your understanding of the algorithms in theory and practice. Yes, you do have to test a fair share of parameters, but doing just that will not give you the best grade. This is supposed to be an engineering/scientific assignment where in some sense you as a consultant have the task of showing a company what they can get from this algorithm on their datasets. That is, making sure that the algorithm is correct (you can get help with that), making hypotheses to test, selecting which parameters to try in a reasonable fashion, and documenting your findings in terms of consistency, the stated metrics and the estimated trajectory, is what will give you a good result. Any theoretical or practical insight from the lectures or book elaborated upon or applied to the data sets also counts positive.

Some background on template code

A skeleton and template code is handed out to give you a starting point to write your code. You are free to rewrite the template, or start a new if you wish. But this will give you a good idea of where to start, some checks to help avoid some possible mistakes in your functions, some documentation, and a starting point for the plots you need for the report. 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.

Note that the CatSlice class from graded assignment 2 is not provided, but can be used to slice matrices.

Task 1: *Implement EKFSLAM*

(a) Implement the motion prediction model,

```
def f(self, x: np.ndarray, u: np.ndarray) -> np.ndarray:
```

corresponding to equation (11.7) that takes a pose x and odometry u predicts it to the next time step,

(b) Implement the motion prediction Jacobian with respect to x,

```
def Fx(self, x: np.ndarray, u: np.ndarray) -> np.ndarray:
```

that is the Jacobian of the above function with respect to the pose, given by equation (11.13).

(c) Implement the motion prediction Jacobian with respect to u,

```
def Fu(self, x: np.ndarray, u: np.ndarray) -> np.ndarray:
```

that is the Jacobian of the above function with respect to the odometry, given by equation (11.14).

(d) Implement the predict function,

```
def predict(
        self,
        eta: np.ndarray,
        P: np.ndarray,
        z_odo: np.ndarray
) -> Tuple[np.ndarray, np.ndarray]:
```

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 the measurement function,

```
def h(self, eta: np.ndarray) -> np.ndarray:
```

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 the jacobian of the above measurement function with respect to η,

```
def H(self, eta: np.ndarray) -> np.ndarray:
```

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 the function

```
def add_landmarks(
        self, eta: np.ndarray, P: np.ndarray, z: np.ndarray
) -> Tuple[np.ndarray, np.ndarray]:
```

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 the update function,

```
def update(
        self, eta: np.ndarray, P: np.ndarray, z: np.ndarray
) -> Tuple[np.ndarray, np.ndarray, float, np.ndarray]:
```

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

Tune and run your EKFSLAM on this dataset.

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

Tune and run your EKFSLAM on the Victoria Park dataset. 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?

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