NA 568 Mobile Robotics: Methods & Algorithms Winter 2022 – Homework 5 – Localization

Maani Ghaffari
University of Michigan

February 18, 2022

This is a reminder that no late HW is accepted. We drop your lowest grade from HW 1-6. It is perfectly fine to drop a zero as a HW grade. We are using Gradescope for turning in HW; see relevant information on the course Canvas site.

This problem set counts for about 7% of your course grade. 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 11:55 pm on Friday, March 11th (Anywhere on Earth Time). This corresponds to 6:55 AM on March 12th in Eastern Time. This is selected out of fairness to all our students, including those who take the course remotely. You are to upload your assignment directly to the Gradescope website as two attachments:

1. A `.tar.gz` or `.zip` file *containing a directory* named after your uniqname (such as `alincoln_hw5.zip` or `alincoln_hw5.tar.gz`). Includes all codes in this directory. Also, include a video for each of the four filters in this directory.
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_hw5.pdf`.

1 Velocity Motion Model

This is the velocity motion model from Chapter 5 of Probabilistic Robotics book. The discrete dynamical equations for the state of a robot given an input of linear and angular velocities (v and ω respectively) are as follows.

$$x_{k+1} = x_k - \frac{\hat{v}}{\hat{\omega}}\sin(\theta_k) + \frac{\hat{v}}{\hat{\omega}}\sin(\theta_k + \hat{\omega}\Delta t), \tag{1}$$

$$y_{k+1} = y_k + \frac{\hat{v}}{\hat{\omega}}\cos(\theta_k) - \frac{\hat{v}}{\hat{\omega}}\cos(\theta_k + \hat{\omega}\Delta t), \tag{2}$$

$$\theta_{k+1} = \theta_k + \hat{\omega}\Delta t + \hat{\gamma}\Delta t. \tag{3}$$

The variable γ is considered a third input to the system but its commanded value is always zero. It is still necessary to take this command into consideration when deriving the Jacobians for this motion model. This input is necessary because without it the posterior pose will be located on a two-dimensional manifold and thus will be degenerate (causing the filter to not be able to estimate the heading correctly). This is described further on page 129 of Probabilistic Robotics. This type of circular motion model as seen in Probabilistic Robotics is used so the robot can rotate and translate concurrently.

$$\hat{v} = v + \epsilon_v \ , \quad \epsilon_v \sim \mathcal{N}(0, \alpha_1 v^2 + \alpha_2 \omega^2), \tag{4}$$

$$\hat{\omega} = \omega + \epsilon_\omega \ , \quad \epsilon_\omega \sim \mathcal{N}(0, \alpha_3 v^2 + \alpha_4 \omega^2), \tag{5}$$

$$\hat{\gamma} = \epsilon_\gamma \ , \quad \epsilon_\gamma \sim \mathcal{N}(0, \alpha_5 v^2 + \alpha_6 \omega^2). \tag{6}$$

In this velocity motion model, the motion noise terms are indicated in (4), (5), and (6), where ϵ_v, ϵ_ω and ϵ_γ are uncorrelated Gaussian noises with 0 mean, and $\alpha_1, \ldots, \alpha_6$ are robot-specific error parameters. This is described further on pages 127-129 of Probabilistic Robotics.

2 Measurement Model

For the implementation of EKF, UKF, and PF, We use the following measurement model that is similar to that shown on the Nonlinear Kalman Filtering slides, where (m_x, m_y) is the known position of the observed landmark.

$$z_k = \begin{bmatrix} \operatorname{atan2}(m_y - y_k, m_x - x_k) - \theta_k \\ \sqrt{(m_y - y_k)^2 + (m_x - x_k)^2} \end{bmatrix} + q_k \ , \quad q_k \sim \mathcal{N}(0, Q_k). \tag{7}$$

For the implementation of RI-EKF, we use the one shown on Slide 33 of the Invariant EKF slides.

$$Y_k = \bar{X}_k^{-1} b + \tilde{V}_k$$

$$\begin{bmatrix} y_k^1 \\ y_k^2 \\ 1 \end{bmatrix} = \begin{bmatrix} \bar{R}_k^T & -\bar{R}_k^T p_k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} m \\ 1 \end{bmatrix} + \begin{bmatrix} v_k \\ 0 \end{bmatrix}, \quad v_k \sim \mathcal{N}(0, V_k) \tag{8}$$

3 Motion Model using 2D Special Euclidean Group

Now we model the the motion model using SE(2). This model correctly respects the geometry of the 2D plane and 3 DOF of the robot motion. The discrete-time motion model of the robot is given by (using the right-invariant definition of the error)

$$\begin{aligned} X_{k+1} &= X_k \exp(\xi_k^\wedge \Delta t) \\ \Sigma_{k+1} &= \Sigma_k + \mathrm{Ad}_{X_k} W_k \mathrm{Ad}_{X_k}^\mathsf{T} \end{aligned} \tag{9}$$

where the robot pose $X_k = \begin{bmatrix} R_k & p_k \\ 0 & 1 \end{bmatrix} \in \mathrm{SE}(2)$, the twist $\xi_k^\wedge = \begin{bmatrix} \omega_k^\wedge & v_k \\ 0 & 0 \end{bmatrix} \in \mathfrak{se}(2)$ (or $\xi_k = \mathrm{vec}(\omega_k, v_k) \in \mathbb{R}^3$), with the motion model noise $w_k \sim \mathcal{N}(0, W_k)$.

Task 1: Derive the propagation and correction equations for the following filters. (40 points)

A. (10 pts) Extended Kalman Filter (EKF) using the velocity motion model and measurement model displayed on (7).

B. (10 pts) Unscented Kalman Filter (UKF) using the velocity motion model and measurement model displayed on (7).

C. (10 pts) Particle Filter (PF) using the velocity motion model and measurement model displayed on (7).

D. (10 pts) Right-Invariant EKF (RI-EKF) using the SE(2) motion model and right-invaraint measurement model displayed on (8).

Task 2: Use the derived filters to localize the robot within the given map. Include the plots generated at the end of the run and a video for each filter. (60 points)

The environment is composed of six landmarks and shown in Figure 1. The robot has a desired trajectory which can be seen in green. Due to noise in the robot actuators, it does not follow the given trajectory. The filtering process for the EKF, UKF, and PF is as follows,

- The robot moves, and an on-board sensor measures the linear and angular velocities in the body frame of the robot. The motion measurements are input to the filter as a 3 x 1 vector where the values are $[v, \omega, \gamma]$ as described in the motion model. The additional angular velocity, γ, is also an input because of the degeneracy issue described in the motion model.
- The robot obtains noisy range and bearing measurements to two given landmarks in the environment. For each landmark, the measurement order is [bearing, range, Landmark_ID].
- The velocity and sensor readings are used for the prediction and correction steps of the filter, respectively.
- The robot moves again, and the process repeats.

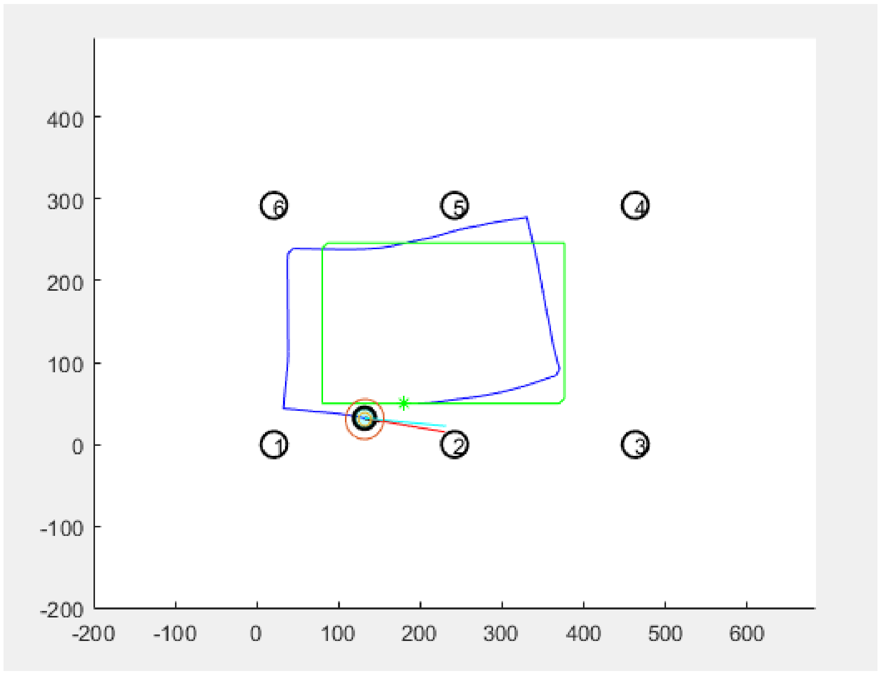

Figure 1: The simulated environment for landmark localization.

The filtering process when using the Invariant EKF is slightly different. It is as follows.

- The robot moves, and an on-board sensor measures the linear and angular velocities in the body frame of the robot. The motion measurements are input to the filter as a 3 x 1 vector where the values are $[v, \omega, \gamma]$ as described in the motion model. The additional angular velocity, γ, is also an input because of the degeneracy issue described in the motion model.
- The angular and linear velocity measurements are converted to the format of lie algebra of SE(2) via the velocity motion model (provided in the code). This describes the motion of the robot in the body frame and is needed for the prediction step of the filter.
- The robot obtains two noisy measurements which are the relative X and Y locations of two separate landmarks in the body frame of the robot. The marker ID for each landmark is also received. Two landmarks are necessary since the Observability Gramian is not full rank when only one landmark is viewed (see slides 35-36 of the Invariant EKF slides).
- The skew symmetric matrix describing relative motion is used for the prediction step. The relative landmark measurements (denoted Y1 and Y2 in the code) and global landmark locations (denoted landmark_x, landmark_y, landmark_x2, landmark_y2 in the code) are used in the correction.
- The robot moves again and the process repeats.

Other important information is as follows,

- The Δt seen in the discrete dynamical equations is one second.
- You will have to try different values for sensor noise to find the optimal values for the filters.
- The filters are considered to be consistent if the ground truth robot pose is always maintained within the 3-sigma contour of the multivariate Gaussian distribution over the state. You should see your robot stay within the ellipse.

You can choose to implement your code in either **MATLAB** or **Python**. Below list some instruction for running the code.

MATLAB

- The filters are initialized using the functions `filter_intialization.m` and `system_initialization.m`. You may need to view these functions since the initialized properties are used in the filtering process.
- An example of how to run the code is as follows, `run(100,0,0,"EKF")`. This will run the environment for 100 time-steps and localize using the EKF filter. To switch between filters enter either `"EKF"`, `"UKF"`, `"PF"` or `"InEKF"` into the `run` function. Enter a 1 into the thrid input of the run function to create a video.

Python

- The filters are initialized using the functions `utils/filter_intialization.py` and `utils/system_initialization.py`. You may need to view these functions since the initialized properties are used in the filtering process.
- An example of how to run the code is as follows, `python3 run.py`. This will run the environment for 100 time-steps and localize using a dummy filter. To switch between filters, change `filter_name` in `config/settings.yaml` to either `"EKF"`, `"UKF"`, `"PF"` or `"InEKF"`. You can use a screen recording tool like `Kazam` to record a video.

A. (15 pts) Fill in the prediction and correction functions within the EKF class located in the file:
MATLAB: `EKF.m` or
Python: `filter/EKF.py`.
Include a video in your zip file.

B. (15 pts) Fill in the prediction and correction functions within the UKF class located in the file:
MATLAB: `UKF.m` or
Python: `filter/UKF.py`.
Include a video in your zip file.

C. (15 pts) Fill in the prediction and correction function within the PF class located in the file:
MATLAB: `PF.m` or
Python: `filter/PF.py`.
Include a video in your zip file.

D. (15 pts) Fill in the prediction, correction and propagation functions within the InEKF class located in the file:
MATLAB: `InEKF.m` or
Python: `filter/InEKF.py`.
Include a video in your zip file.

E. (10 Extra credits) **The covariance while using the InEKF is kept within the Lie algebra of the matrix group.** We wish to map the covariance from Lie algebra to Cartesian coordinates, (x, y, θ).

MATLAB

Update the function `lieToCartesian.m` in order to map the covariance from Lie algebra to Cartesian coordinates, (x, y, θ), and then use the function `mahalanobis.m` to measure the performance of the filter. Include a plot of the filter performance. Save the Cartesian mean and covariance into `filter.mu_cart` and `filter.sigma_cart`, respectively. (If you choose to complete this uncomment mahalanobis and Lie to Cartesian functions in the InEKF filter switch.)

Python

Update the function `lieToCartesian()`, `func()` in `utils/utils.py` to map the covariance from Lie algebra to Cartesian coordinates, and then use the function `mahalanobis()` to measure the performance of the filter. Include a plot of the filter performance. Save the Cartesian mean and covariance into `mu_cart` and `Sigma_cart`, respectively. (If you choose to complete this, change `Lie2Cart` in `config/settings.yaml` to `True`.)

Remark 1. *Please pay attention to the size of the recorded videos. Use a reasonable quality that produces a small size for uploading the zip file.*