State Estimation

Submission Guidelines

Writeups must be submitted as a PDF via Gradescope. LaTeX is preferred, but other typesetting methods are acceptable. Code for the programming component must be submitted in a zip archive via Canvas. Plots generated as part of the programming component should be included in the writeup.

0 Collaborators

List the names of all collaborators and which questions you collaborated on.

1 Bayes Rule (5 points)

A vacuum cleaning robot is equipped with a cleaning unit to clean the floor. The robot has a binary sensor that detects whether a floor tile is clean or dirty. However, neither the cleaning unit nor the sensor are perfect; from previous experience, you know the robot successfully cleans a dirty tile with probability

$$p(x_{t+1} = \text{clean} | x_t = \text{dirty}) = 0.6$$

where x_t is the state of the floor tile at time t and x_{t+1} is the resulting state after the cleaning unit has been activated. Activating the cleaning unit when the tile is clean will never make it dirty.

The probability that the sensor indicates that the tile is clean when it is actually dirty is $p(z = \text{clean} | x = \text{dirty}) = 0.2$. The probability that the sensor correctly detects a clean tile is given by $p(z = \text{clean} | x = \text{clean}) = 0.6$.

You have no knowledge about the true state of the floor tile, except that after cleaning the tile, the robot's sensor indicates that it is clean. Compute the probability that the floor tile is now clean, assuming that the prior distribution at time t is $p(x_t = \text{clean}) = c$. Then, plot $p(x_{t+1} = \text{clean})$ for $0 \leq c \leq 1$.

2 Correlated Noise (15 points)

The Kalman filter formulation assumes independent additive Gaussian noise in the transition and observation model. However, this assumption may not hold in many practical situations. Consider the following uncontrolled system:

$$\begin{aligned} x_0 &\sim \mathcal{N}(\mu_0, \Sigma_0) \\ x_{t+1} &= Ax_t + w_t \\ w_t &= 0.1w_{t-1} + 0.2w_{t-2} + p_{t-1} \\ p_t &\sim \mathcal{N}(0, \Sigma_{pp}) \\ z_t &= Cx_t + v_t \\ v_t &= 0.8v_{t-1} + q_{t-1} \\ q_t &\sim \mathcal{N}(0, \Sigma_{qq}) \\ p_{-1} &= q_{-1} = v_{-1} = w_{-1} = w_{-2} = 0 \end{aligned}$$

Describe a new state representation, transition model, and observation model such that the problem can be transformed into the standard uncorrelated noise Kalman filtering setup.

3 Gaussian Conditioning (20 points)

Let X and Y denote two random variables that are jointly Gaussian:

$$\begin{aligned} p(x, y) &= \mathcal{N}(\mu_{XY}, \Sigma_{XY}) \\ &= \frac{1}{2\pi|\Sigma_{XY}|^{1/2}} \exp\left\{-\frac{1}{2}((x, y)^\top - \mu_{XY})^\top \Sigma_{XY}^{-1}((x, y)^\top - \mu_{XY})\right\} \end{aligned}$$

where $\mu_{XY} = (\mu_X, \mu_Y)^\top$ and $\Sigma = \begin{pmatrix} \sigma_X^2 & \sigma_{XY}^2 \\ \sigma_{XY}^2 & \sigma_Y^2 \end{pmatrix}$.

Prove that the conditional distribution $p(x|y)$ is also Gaussian:

$$p(x|y) = \mathcal{N}(\mu_{X|Y}, \sigma_{X|Y}^2) = \frac{1}{\sqrt{2\pi}\sigma_{X|Y}} \exp\left\{-\frac{1}{2}\frac{(x - \mu_{X|Y})^2}{\sigma_{X|Y}^2}\right\}$$

where $\mu_{X|Y} = \mu_X + \frac{\sigma_{XY}^2}{\sigma_Y^2}(y - \mu_Y)$ and $\sigma_{X|Y}^2 = \sigma_X^2 - \frac{\sigma_{XY}^4}{\sigma_Y^2}$.

Hints and Reminders

- Use the definition of the conditional distribution and complete the square
- Since $p(x, y)$ is jointly Gaussian, the marginal distribution of y is also Gaussian: $p(y) = \mathcal{N}(\mu_Y, \sigma_Y^2)$
- For 2×2 invertible matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $A^{-1} = \frac{1}{|A|}\begin{pmatrix} d & -c \\ -b & a \end{pmatrix}$ and $|A| = ad - bc$.

Landmark-Based Localization

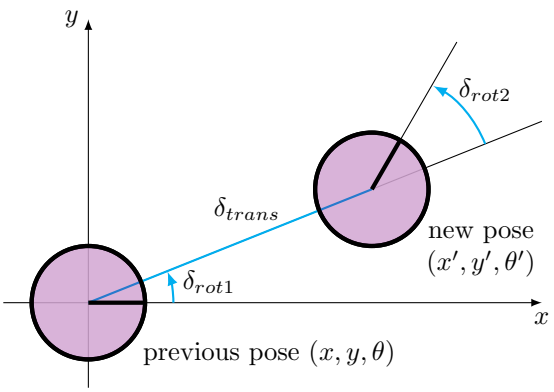

Figure 1: Odometry-based motion model

In the programming component of this assignment, you will implement an Extended Kalman Filter (EKF) and Particle Filter (PF) for localizing a robot based on landmarks. The state of the robot is its 2D position and orientation: (x, y, θ). We will use the odometry-based motion model in Figure 1 (i.e. the robot rotates by δ_{rot1}, drives straight forward δ_{trans}, then rotates again by δ_{rot2}). We assume that there are landmarks present in the robot's environment. The robot receives the bearings (angles) to the landmarks and the ID of the landmarks as observations: $(\text{bearing}, \text{landmark ID})$.

We assume a noise model for the odometry motion model with parameters α (PR Table 5.6) and a separate noise model for the bearing observations with parameter β (PR Section 6.6). The landmark ID observation is noise-free. See the provided starter code for implementation details.

At each timestep, the robot starts from the current state and moves according to the control input. The robot then receives a landmark observation from the world. You will use this information to localize the robot over the whole time sequence with an EKF and PF.

4 Linearized Motion Model (10 points)

The EKF prediction step uses a linearized approximation of a nonlinear motion model g around the current mean of the state distribution μ and control u. Derive the Jacobians with respect to the mean $G = \frac{\partial g}{\partial \mu}$ and control $V = \frac{\partial g}{\partial u}$.

$$G = \begin{pmatrix} \frac{\partial x'}{\partial x} & \frac{\partial x'}{\partial y} & \frac{\partial x'}{\partial \theta} \\ \frac{\partial y'}{\partial x} & \frac{\partial y'}{\partial y} & \frac{\partial y'}{\partial \theta} \\ \frac{\partial \theta'}{\partial x} & \frac{\partial \theta'}{\partial y} & \frac{\partial \theta'}{\partial \theta} \end{pmatrix} \qquad V = \begin{pmatrix} \frac{\partial x'}{\partial \delta_{rot1}} & \frac{\partial x'}{\partial \delta_{trans}} & \frac{\partial x'}{\partial \delta_{rot2}} \\ \frac{\partial y'}{\partial \delta_{rot1}} & \frac{\partial y'}{\partial \delta_{trans}} & \frac{\partial y'}{\partial \delta_{rot2}} \\ \frac{\partial \theta'}{\partial \delta_{rot1}} & \frac{\partial \theta'}{\partial \delta_{trans}} & \frac{\partial \theta'}{\partial \delta_{rot2}} \end{pmatrix}$$

Code Overview

The starter code is written in Python and depends on NumPy and Matplotlib. This section gives a brief overview of each file.

- `localization.py` – This is your main entry point for running experiments.
- `soccer_field.py` – This implements the dynamics and observation functions, as well as the noise models for both.
- `utils.py` – This contains assorted plotting functions, as well as a useful function for normalizing an angle between $[-\pi, \pi]$.
- `policies.py` – This contains a simple policy, which you can safely ignore.
- `ekf.py` – Add your extended Kalman filter implementation here!
- `pf.py` – Add your particle filter implementation here!

Command-Line Interface

To visualize the robot in the soccer field environment, run

```
$ python localization.py --plot none
```

The blue line traces out the robot's position, which is a result of noisy actions. The green line traces the robot's position assuming that actions weren't noisy. After you implement a filter, the filter's estimate of the robot's position will be drawn in red.

```
$ python localization.py --plot ekf
$ python localization.py --plot pf
```

To see other command-line flags available to you, run

```
$ python localization.py -h
```

Data Format

- state: $[x, y, \theta]$
- control: $[\delta_{rot1}, \delta_{trans}, \delta_{rot2}]$
- observation: $[\theta_{bearing}]$

Hints

- Make sure to call `utils.minimized_angle` any time an angle or angle difference could exceed $[-\pi, \pi]$.
- Make sure to use the low-variance systematic sampler from lecture. It gives you a smoother particle distribution and also requires only a single random number per resampling step.
- Turn off plotting for a significant speedup.

5 EKF Implementation (25 points)

Implement the extended Kalman filter algorithm in `ekf.py`. You will need to fill in `ExtendedKalmanFilter.update` and the `Field` methods `G`, `V`, and `H`. Your results from a successful EKF implementation should be comparable to the following results.

```
$ python localization.py ekf --seed 0
...
Mean position error: 8.9983675360847
Mean Mahalanobis error: 4.416418248584298
ANEES: 1.472139416194766
```

(a) Plot the real robot path and the filter path under the default (provided) parameters.

(b) Plot the mean position error as the α and β factors range over $r = [1/64, 1/16, 1/4, 4, 16, 64]$[1] and discuss anything interesting you observe.

You should run 10 trials per value of r. One run might look something like:

```
$ python localization.py ekf --data-factor 4 --filter-factor 4
```

(c) Plot the mean position error and ANEES (average normalized estimation error squared) as the filter α, β factors vary over r (as above) while the data is generated with the default. Discuss anything interesting you observe.

6 PF Implementation (25 points)

Implement the particle filter algorithm in `pf.py`. You will need to fill in `ParticleFilter.update` and `ParticleFilter.resample`.

```
$ python localization.py pf --seed 0
...
Mean position error: 8.567264372950905
Mean Mahalanobis error: 14.742252771106532
ANEES: 4.914084257035511
```

(a) Plot the real robot path and the filter path under the default parameters.

(b) Plot the mean position error as the α, β factors range over r and discuss.

(c) Plot the mean position error and ANEES as the filter α, β factors vary over r while the data is generated with the default.

(d) Plot the mean position error and ANEES as the α, β factors range over r and the number of particles varies over $[20, 50, 500]$.

[1] Since the factors are multiplied with variances, this is between 1/8 and 8 times the default noise values.