COMS W4733: Computational Aspects of Robotics, Fall 2021

Homework 4

Problem 1: Bayes Filter (20 points)

We have a robot in an infinitely long hallway located somewhere between $x = -1$ and $x = 1$. We will use a uniform distribution over this domain as our prior belief distribution $B(x_0)$.

(a) The robot takes a noisy step in the positive x direction. Its motion model is given by a Gaussian distribution with unit variance: $p(x_1 \mid x_0, u_0) \sim \mathcal{N}(x_0 + 0.5, 1)$. Compute the new belief distribution $B'(x_1)$ after this step and generate a plot of the pdf. (The error function may be useful here.)

(b) The robot now picks up a sensor reading z_1 that can only be read between $x = -0.5$ and $x = 1.5$. The observation model $p(z_1 \mid x_1)$ is given by a uniform distribution over this domain. Compute the posterior belief distribution $B(x_1)$ after accounting for this sensor update and generate a plot of the pdf.

(c) Now suppose that we use a discrete Bayes filter instead. We discretize the uniform prior distribution $B(x_0)$ into four "cells" centered at $x = -0.75$, $x = -0.25$, $x = 0.25$, and $x = 0.75$. The probability of each cell simply takes on the value of the probability at each cell center. Compute the discretized distribution $B'(x_1)$ given the same motion model as in part (a).

(d) Combine the discretized $B'(x_1)$ distribution above with the observation model in part (b) to compute the discretized posterior $B(x_1)$.

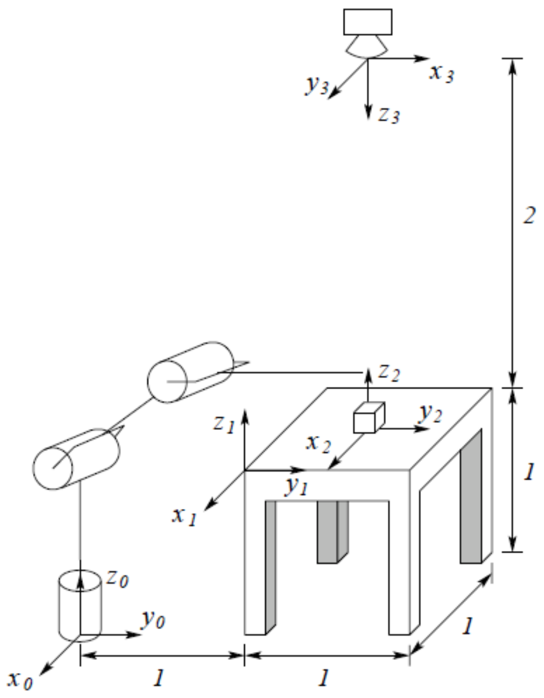
Problem 2: Homogeneous Transformations (15 points)

In the diagram below, a robot with base frame 0 is located 1 m away from a table, which is 1 m high and 1 m square. Frame 1 is rigidly attached to the table at the corner closest to the robot, and the two frames' y axes are coincident. A cube is located at the center of the table, with frame 2 rigidly attached to the center of its bottom face. A camera is located 2 m directly above the center of the table, with frame 3 rigidly attached to it.

(a) Find the homogeneous transformations relating each frame to the previous one: A_1^0, A_2^1, A_3^2.

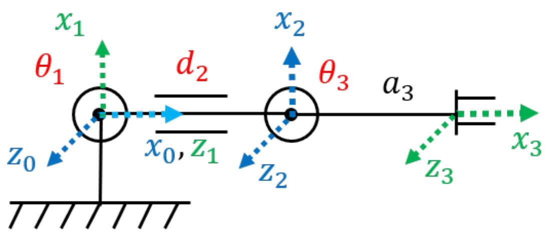
(b) Suppose that the robot performs the following two actions. It pulls the table toward itself, translating the table and cube by -0.5 m along the y_0 axis. It then rotates the cube 90 degrees clockwise about the z_2 axis. Recompute each of the previous homogeneous transformations.

(c) Compute the composite transformation A_3^0 relating the camera frame to the robot frame. Would you get a different answer depending on whether you use the transformations from part (a) or part(b)? Describe the translational displacement and rotation between the two frames (the latter may be more sophisticated than a basic coordinate axis rotation).



Problem 3: Forward Kinematics (15 points)

Consider the planar RPR manipulator shown below. Its joint variables are $q = (\theta_1, d_2, \theta_3)$, the link between the last joint and end effector has length a_3, and a set of coordinate frames from the first joint to the end effector are shown. Note that the link between the two revolute joints has length equal to d_2.



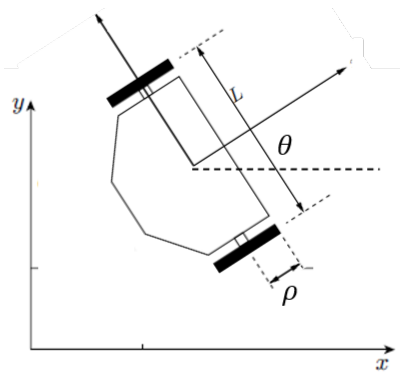
(a) Derive the complete DH table. There should be five columns (link, a_i, α_i, d_i, θ_i) and four rows (link, 1, 2, 3).

(b) Find the homogeneous transformations A_i^{i-1} corresponding to each row of the DH table.

(c) Find the composite forward kinematics T_3^0. What are the position and orientation of the end effector as functions of the joint variables?

Problem 4: Differential-Drive Car Localization

The differential-drive car shown below is a common mobile robot configuration. By independently controlling the velocities of its two wheels, the robot can translate and reorient on the plane. Its state vector is $\mathbf{x} = (x, y, \theta)$, it has two inputs $\mathbf{u} = (u_1, u_2)$ corresponding to the angular velocities of its two wheels, ρ and L are fixed parameters, and Δt is the timestep duration.



You will be implementing two different localization algorithms, one using the extended Kalman filter and one using the particle filter. The robot itself behaves according to models with nonlinear noise components and unknown to your estimators (it essentially moves around the plane in an imperfect circle). Instead, the filters will assume a standard nonlinear motion model with additive Gaussian noise $\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k$, where

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \Delta t \begin{bmatrix} \frac{\rho}{2}\cos\theta_{k-1}(u_{1,k} + u_{2,k}) \\ \frac{\rho}{2}\sin\theta_{k-1}(u_{1,k} + u_{2,k}) \\ \frac{\rho}{L}(u_{2,k} - u_{1,k}) \end{bmatrix} + \mathbf{w}_k \tag{1}$$

and $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, Q)$. The vehicle has a range and bearing sensor, which measures its distance r_k and angle ϕ_k from p fixed landmarks within the sensor's maximum range. Again, these will have both nonlinear model and noise components, but your estimators will assume the standard model $\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k$, where $h = \begin{bmatrix} h_1 \cdots h_p \end{bmatrix}^T$,

$$h_i(\mathbf{x}_k) = \begin{bmatrix} r_{ik} \\ \phi_{ik} \end{bmatrix} = \begin{bmatrix} \sqrt{(x_{li} - x_k)^2 + (y_{li} - y_k)^2} \\ \text{atan2}(y_{li} - y_k, x_{li} - x_k) - \theta_k \end{bmatrix} + \mathbf{v}_k \tag{2}$$

and $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, R_k)$.

4.0: Code Description

We provide skeleton files based on localization examples from PythonRobotics. You will be filling out the necessary functions for EKF localization and particle filter localization. A description of the relevant variables and functions common to both files is provided below:

- `WHEEL1_NOISE`, `WHEEL2_NOISE`, and `BEARING_SENSOR_NOISE` are parameters describing the true robot model noise. *Your estimators should not make reference to these.*
- `RHO` and `L` are known physical parameters of the robot as shown on the above figure. `MAX_RANGE` is the maximum range of the robot's sensor. `RFID` is a list of known landmark positions.
- `Q` and `R` are covariance matrices used by your estimators.
- The remaining variables describe the time interval, total simulation time, number of particles in the particle filter, and the animation plot limits.
- The functions `input`, `move`, `measure` implement the "true" physics and behavior of the robot.

- The `main` function runs the simulation and displays an animation of the true robot trajectory in blue, estimated robot trajectory in red, and landmarks indicated by `*` symbols connected to the robot with black segments if sensed. A second plot shows the errors in the estimated state components at the end of the animation.

4.1: EKF Localization (15 points)

Recall that the EKF uses *linearizations* of the motion and observation models in Equations (1) and (2). Before you write any code, derive the F_k matrix by linearizing the motion model and show the result on your pdf. The linearization form of H_k is the same as the example from lecture.

Implement the EKF in `ekf_localization.py`. The `predict()` function takes in the state mean, covariance, and velocity inputs. It computes and returns the predicted state mean and covariance.

The `update()` function takes in the state mean and covariance, as well as sensor measurements in the form of a $p \times 3$ array. The first two columns contain an observed landmark's sensed range and bearing, and the third column contains the landmark row index in `RFID`. From these you can compute and form the $2p$ innovation vector $\tilde{\mathbf{y}}_k$, the $2p \times 3$ measurement linearization H_k, and the associated $2p \times 2p$ measurement covariance matrix R_k. R_k contains p copies of the R matrix (for a single landmark measurement) along its diagonal blocks. You can then implement the update equations and return the updated state mean and covariance.

4.2: EKF Discussion (10 points)

Once you have implemented the above procedures, you can run the simulation loop in the main function, which will animate the robot's true and predicted state trajectories. The state covariance is shown as a green dashed ellipse centered at the robot's current location, although if your EKF is working properly the ellipse will generally be small enough to appear dot-sized. Answer each of the questions below using a few sentences each along with associated plots.

1. Run the full length of the simulation with the default parameters a few times. You should observe that the estimation is generally able to track the robot's true state fairly well. Explain any portions in which the errors increase by a noticeable amount and hypothesize why this occurs. Include two figures showing these observations, one with the last frame of the animation and the other with the plot of the state errors over time.

2. How are the covariance and accuracy of the state estimate affected if `MAX_RANGE` is decreased to a value like 10? Show the resultant plots and explain how the number of sensed landmarks affects the accuracy of state estimation.

3. Now let's explore the effect of starting with different initial estimates for the state mean and covariance (reset `MAX_RANGE` to 18), which are set near the top of the `main()` function. First try increasing and decreasing `P` by several orders of magnitude and comment on whether you see any different initial behaviors. Then set the initial mean `x_est` to something like $(20, 20, 20)$ and show the resultant plots. Explain your observations.

4.3: Particle Filter Localization (15 points)

Now moving on to `pf_localization.py`, the `localization()` procedure implements the main loop of the particle filter. It calls several helper functions: `generate_particles()` to initialize (or re-initialize) the filter, `predict()` and `update()` in each timestep, and `resample()` when sample weights become sufficiently low. Implement the last three functions. `predict()` returns a particle's new state by sampling from the motion model in Equation (1). You can generate random Gaussian values to simulate $\mathbf{w}_k$.

`update()` returns the weight for a given particle based on the measurement provided. As in the EKF, the measurement is a $p \times 3$ array. Loop through each known landmark i in `RFID`. If you encounter either situation in which the predicted range $\hat{r}_i \leq$ `MAX_RANGE` but landmark i is not in `z`, or conversely $\hat{r}_i >$ `MAX_RANGE` but it is in `z`, you can stop and immediately return the observation likelihood $w = 0$. Otherwise, compute the innovation $\tilde{\mathbf{y}}_i$ for landmark i. Assuming that $\tilde{\mathbf{y}}_i$ is distributed according to a zero-mean Gaussian with covariance R and that each landmark measurement is mutually independent, the observation likelihood is given by

$$w = \prod_i \frac{1}{2\pi\sqrt{|R|}} \exp\left(-\frac{1}{2}\tilde{\mathbf{y}}_i R^{-1} \tilde{\mathbf{y}}_i^T\right). \tag{3}$$

Finally, `resample()` performs the resampling process: given a current set of particles and their associated weights, return a new set of particles generated by sampling from the provided information. This can be done very easily in Python using `numpy.random.choice`.

4.4: Particle Filter Discussion (10 points)

Once you have implemented the above procedures, you can run the simulation loop in the main function. The particles are represented by green dots, while the red trajectory represents the mean of their positions. Answer each of the questions below using a few sentences each along with associated plots.

1. Run the full length of the simulation with the default parameters a few times. You should observe that the estimation looks somewhat erratic with particles everywhere at the beginning, followed by a convergence to near the true robot state if the filter is working properly. Explain this particle behavior (and any other observations that you have) and include the two figures showing these observations.

2. How is the overall accuracy of the state estimate affected if `MAX_RANGE` is decreased to a value like 10? Show the resultant plots and explain how the number of sensed landmarks affects the accuracy of state estimation.

3. Now let's explore the effect of different numbers of particles (reset `MAX_RANGE` to 20). Show what happens when we have only 10 or so particles. Why is it important to have a sufficiently large number of particles for accurate state estimation?

Submission

You should have one document containing your solutions, responses, and figures for all written questions. At the end of the document, create an appendix with printouts of all code that you wrote or modified for problem 4 (you should not include the entire files). Submit both your document and code archive separately on Gradescope.