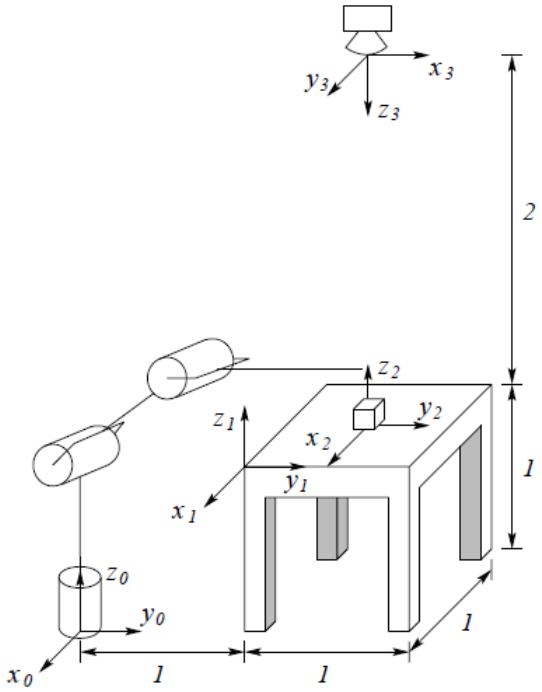
COMS W4733: Computational Aspects of Robotics

Homework 4

Problem 1: Homogeneous Transformations (18 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.



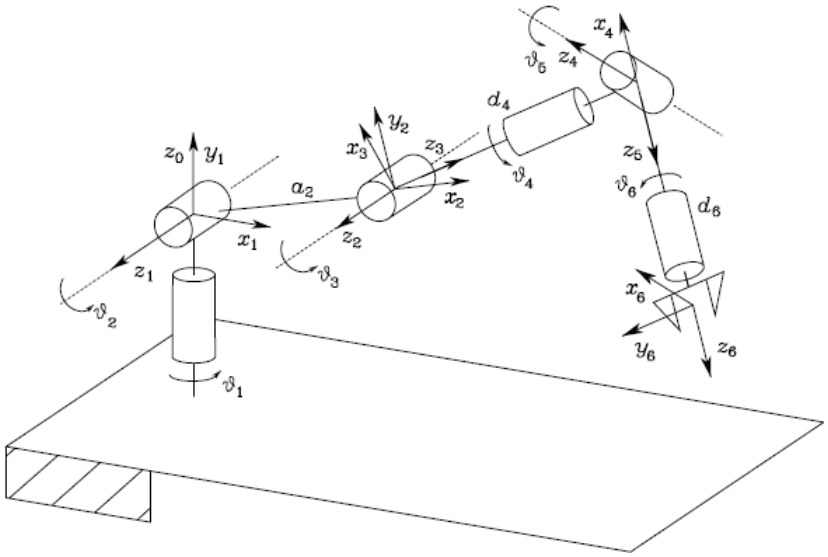
(a) Find the homogeneous transformations relating each frame to the previous one: A_1^0, A_2^1, A_3^2.

(b) Suppose the robot rotates the cube 90 degrees counterclockwise about the z_2 axis and then moves it to the position $(-0.2, 0.2, 0)$ relative to frame 1. Recompute the homogeneous transformations above that are changed by this action.

(c) Compute the composite transformation A_3^0 relating the camera frame to the robot frame by multiplying the intermediate transformation matrices. Why does it not matter whether you use the transformations from part (a) or part (b)?

Problem 2: Forward Kinematics (16 points)

We attach a spherical wrist to the end of the anthropomorphic arm as shown below. All coordinate frames are defined so as to conform to DH convention. Not all frame axes are shown; the origins of frame pairs 0 and 1, 2 and 3, and 4 and 5 are coincident. a_2, d_4, and d_6 are the robot's link lengths.



1. Find the complete DH parameter table describing the transformations between each successive frame. There should be six rows, with row i containing the transformation parameters between frames $i-1$ and i. Format your table with five columns: Link, a_i, α_i, d_i, θ_i.

   Hint: Refer to the anthropomorphic arm and spherical wrist examples in the lecture notes and identify the differences. Pay particular example to frame 3.

2. Use the forward kinematics to find the pose T_6^0 of the end-effector when all joint angles are 0 (we recommend that you write a small program to do this). Briefly describe the result as a sequence of elementary transformations of the base frame.

Problem 3: Bayes Filter (16 points)

We have a robot in an infinitely long hallway, located somewhere between $x = 0$ and $x = 1$. We will use a uniform distribution over this domain as our prior belief distribution $B(x_0)$.

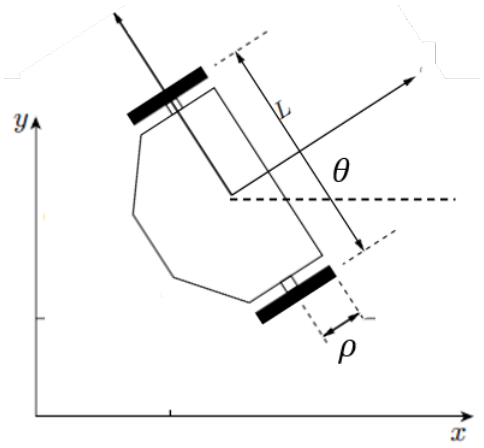
(a) The robot takes a (noisy) step one unit in the positive x direction. Its motion model is given by a Gaussian distribution with unit variance: $p(x_1 \mid x_0, u_0) \sim \mathcal{N}(x_0 + 1, 1)$. Compute the new belief distribution $B'(x_1)$ after this step and generate a plot of the pdf. (You will probably want to use a program to compute any integrals rather than doing them by hand. The error function may be useful here.)

(b) The robot now picks up a sensor reading y_1 that can only be read between $x = 0$ and $x = 1$. The observation model $p(y_1 \mid x_1)$ 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 (remember to normalize!). Briefly compare and contrast $B(x_1)$ and $B(x_0)$.

Problem 4: Particle Filter Localization

Problem and Code Description

You will be implementing a particle filter to perform localization for the differential-drive vehicle shown below. Many mobile robots use this drive mechanism, which consists of two wheels mounted on one axis, each independently driven forward or backward with velocities $\dot{\psi}_1$ and $\dot{\psi}_2$.



The robot will navigate the plane, which is populated with several known landmarks. It is also equipped with 360-degree range and bearing sensors. Both the robot's true motion and sensor models are noisy, although the particle filter does not have access to this information. The provided code is based on the particle filter localization example from PythonRobotics. A description of some of the relevant variables and functions is provided below:

- `Q` and `R` are covariance matrices used by the particle filter.
- `RHO`, `L`, `WHEEL1_NOISE`, `WHEEL2_NOISE`, and `BEARING_SENSOR_NOISE` are parameters describing the actual robot. The first two correspond to the parameters shown in the figure.
- `RFID` is a list of landmark positions. The remaining variables describe the time interval, total simulation time, maximum range of the range sensor, number of particles to use, and the animation plot limits.
- The functions `input`, `move`, `measure` implement the "true" physics and behavior of the robot. These are implemented for you and unknown by the robot.
- The functions `generate_particles`, `localization`, `predict`, `update`, and `resample` implement the particle filter. You will write the last three of these.
- The `main` function runs the simulation and displays an animation of the robot, particles, and landmarks, as well as an error plot at the end of the animation. If you run the code without implementing anything, you should see an animation pop up. The blue trajectory shows the robot's "ground truth" position in the plane. Note that it does not traverse a perfect circle even though the inputs are constant, due to the physical imperfections of the robot. As the robot moves, it is able to sense landmarks, shown as `*` symbols in the plot and connected to the robot with black segments if sensed. At the end of the animation, a plot shows the error between the state estimate and true state over time.

Motion and Sensor Models

The robot's state is its position and orientation (x, y, θ). The control inputs are the velocities of the left and right wheels $(\dot{\psi}_1, \dot{\psi}_2)$. However, due to physical imperfections, the effective velocity inputs (u_1, u_2) include Gaussian noise. The motion model is as follows:

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

The sensor model returns the range and bearing between the robot and all landmarks within the sensor's threshold distance. These measurements are imperfect, with the range value rounded to the nearest integer and the bearing including a Gaussian distributed error. The corresponding landmark locations are known. The return structure is a matrix, in which each row contains the measured range and bearing as well as the corresponding true landmark position. The number of rows varies depending on the number of landmarks observed at a given time.

4.1: Particle Prediction (10 points)

The prediction (or motion) of the particles will be slightly different from the physical motion model of the robot, reflecting our ignorance about some of the physical aspects. Instead of modeling individual uncertainties in the inputs, we will simply add Gaussian noise to the entire state vector. The particle motion model is thus the following:

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

where $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, Q)$. Implement this in `predict()`.

4.2: Particle Update (10 points)

When accounting for measurements, each particle gets reweighted according to the cumulative likelihood of each measurement. As with the motion model, the particle filter does not know the true measurement model. For each landmark observed, compute the difference between the predicted range and bearing and the measured range and bearing. We will assume that this "observation error" $\begin{bmatrix} \tilde{r} & \tilde{\phi} \end{bmatrix}^T$ is distributed according to a zero-mean Gaussian with covariance matrix R. We will further assume that the measurements of each landmark are independent of each other. The cumulative likelihood modifying a particle's current weight is thus given by

$$\prod_{landmarks} \frac{1}{2\pi\sqrt{|R|}} \exp\left(-\frac{1}{2} \begin{bmatrix} \tilde{r} & \tilde{\phi} \end{bmatrix} R^{-1} \begin{bmatrix} \tilde{r} & \tilde{\phi} \end{bmatrix}^T \right). \tag{2}$$

Implement this in `update()`.

4.3: Particle Resampling (10 points)

When the magnitude of the particle weights is small enough, the particles are all resampled so that we can reset their weights. This can be done very easily in Python using `numpy.random.choice`. The weight vector contains the likelihoods, and we can sample `NP` particles from the original set of particles. The new particle set is then returned. Implement this in `resample()`.

4.4: Analysis and Discussion (20 points)

If you have successfully implemented all the functions above, you can uncomment the two lines in the main function that plot the particles and state estimate. When running the animation, the particles are represented by green dots, and the red trajectory keeps track of their mean position over time. At the very beginning, the particles are initialized uniformly over the entire state space, but they should quickly converge to near the robot's true state. Given the default configuration and parameters, the state estimate should stay fairly close to the ground truth throughout the simulation.

1. Run the full length of the simulation with the default parameters. Include the two figures in your writeup, one with the last frame of the animation and the other with the plot of the state errors over time. Briefly describe the behavior of the particles. Do they generally provide a good approximation of the robot's true state? Provide an explanation of why they sometimes diverge and at other times immediately jump to a more certain location.

2. What happens if `MAX_RANGE` for sensing landmarks is increased to 15 or 20? What if it is decreased to a value like 5? (For the latter, note that when all particle weights become 0, the `localization` function throws away all current particles and generates a completely new set of particles.) Show plots for both cases of increasing and decreasing `MAX_RANGE` and explain how the sensed landmarks affect the accuracy of state estimation.

3. Now let's explore the effect of different numbers of particles (reset `MAX_RANGE` to 10). Try two different experiments of 50 and 30 particles (you can try an even lower value if you like), and show the plots for both. Why is it important to have a large number of particles for accurate state estimation? Explain how it may be possible to completely lose track of the true state forever for a sufficiently small number of particles.

4. Reset all parameters to their default values. Recall that Q and R are "user-defined"; they approximate the true underlying imperfections in the robot physics. Explain what happens if either parameter is much smaller or much larger than they currently are. You can try scaling these matrices by 10^{-2} or 10^{2} (or even 0, but be careful about R!). You do not have to show plots, but please separately address each of the four scenarios (Q much smaller or much larger, R much smaller or much larger). How do we interpret the uncertainties in our particle filter in those scenarios, and why do they lead to the results you observe?

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 file). Please tag your pages, including the relevant parts of the appendix for problem 4. Submit both your document and code archive separately on Gradescope.