CSE 571 - Robotics
Homework 2 - EKF and Particle Filter Localization

Due Thursday, November 3 at 11:59 PM

The key goal of this homework is to get an understanding of the properties of Kalman filters and Particle filters for state estimation. There are two parts to the homework - a written assignment and a programming assignment. There are two questions in the written assignment. For the programming assignment, you will be implementing an Extended Kalman Filter (EKF) and a Particle Filter (PF) for landmark based localization. You will also analyze their performance under various conditions. The zip file containing the code for this homework can be found on the class website (`https://courses.cs.washington.edu/courses/cse571/16au/`).

Useful reading material: Lecture notes, Chapters 3,4,5,7 & 8 of Probabilistic Robotics, Thrun, Burgard and Fox (pdf shared with class)

1 Writing assignments

1.1 Kalman Gain

Recall that if the random variables x_1 and x_2 are distributed according two Gaussian distributions:

$$p(x_1) = \mathcal{N}(\mu_1, \sigma_1^2)$$
$$p(x_2) = \mathcal{N}(\mu_2, \sigma_2^2)$$

then, their product distribution is still a Gaussian:

$$p(x_1)p(x_2) \sim \mathcal{N}\left(\frac{\sigma_2^2}{\sigma_1^2+\sigma_2^2}\mu_1 + \frac{\sigma_1^2}{\sigma_1^2+\sigma_2^2}\mu_2, \frac{1}{\sigma_1^{-2}+\sigma_2^{-2}}\right)$$

Show that in 1D, the Kalman filter correction step shown below is equivalent to a multiplication of the predicted state ($\mathcal{N}(\bar{\mu}, \bar{\sigma}^2)$) and observation ($\mathcal{N}(z, \sigma_{obs}^2)$) Gaussians with mean and variance given by:

$$\mu = \bar{\mu} + K(z - \bar{\mu})$$
$$\sigma^2 = (1-K)\bar{\sigma}^2$$

where $K = \frac{\bar{\sigma}^2}{\bar{\sigma}^2+\sigma_{obs}^2}$.

1.2 Motion Model Jacobian

The Kalman Filter tracks the state by repeatedly predicting the next state distribution given the current state distribution and the control (prediction step) followed by correcting the prediction based on the observations

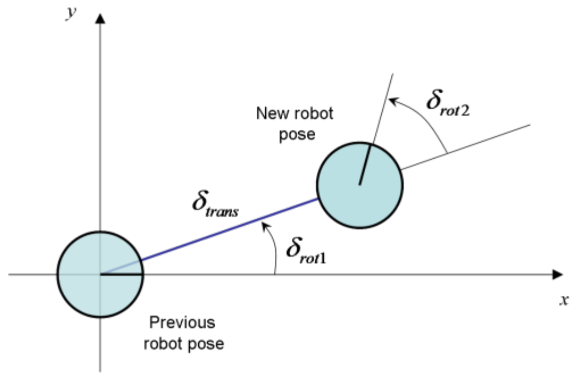

Figure 1: Odometry motion model.

(correction step). Given the current state distribution (μ_t, Σ_t) and the current control (u_t), we use a motion model (g) to generate predictions of the next state (μ'_{t+1}):

$$\mu'_{t+1} = g(\mu_t, u_t)$$

While the Linear Kalman Filter assumes a linear motion model, the model can be non-linear for the Extended Kalman Filter. We then linearize this model around the current state and control for predictions using the EKF. For this problem, we will try to linearize the non-linear odometry motion model we discussed in class (Section 5.4 of the book).
Our state is the 2D position of the robot and it's orientation ($\mu_t =< x_t, y_t, \theta_t >$). The applied control is of the form: ($u_t =< \delta_{rot1}, \delta_{trans}, \delta_{rot2} >$). The odometry motion model assumes that the robot rotates in place first by an angle equal to δ_{rot1}, followed by a translation of δ_{trans} and a final rotation by δ_{rot2}. Fig: 1 shows a pictorial representation of the motion model.

The equations for the motion model (g) are as follows:

$$\begin{aligned} x'_{t+1} &= x_t + \delta_{trans} * cos(\theta_t + \delta_{rot1}) \\ y'_{t+1} &= y_t + \delta_{trans} * sin(\theta_t + \delta_{rot1}) \\ \theta'_{t+1} &= \theta_t + \delta_{rot1} + \delta_{rot2} \end{aligned}$$

where $\mu'_{t+1} =< x'_{t+1}, y'_{t+1}, \theta'_{t+1} >$ is the prediction of the motion model.

For the prediction step of the EKF, we need compute a linearized approximation of the non-linear motion model (g) around the current state (μ_t) and the current control (u_t). Specifically, **you need to compute** the Jacobians w.r.t state $G = \frac{\partial g}{\partial \mu_t}$ and control $V = \frac{\partial g}{\partial u_t}$, where:

$$G = \begin{pmatrix} \frac{\partial x'}{\partial x} & \frac{\partial x'}{\partial y} & \frac{\partial x'}{\partial \theta} \\ \frac{\partial y'}{\partial x} & \frac{\partial y'}{\partial y} & \frac{\partial y'}{\partial \theta} \\ \frac{\partial \theta'}{\partial x} & \frac{\partial \theta'}{\partial y} & \frac{\partial \theta'}{\partial \theta} \end{pmatrix}$$

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

with the subscripts t and $t + 1$ dropped out.

2 Programming assignment - Landmark Based Localization

In this assignment, you will implement an Extended Kalman Filter (EKF) and a Particle Filter(PF) for localizing a robot based on landmarks. The state of the robot is it's 2D position and orientation ($< x, y, \theta >$). We will model the motion of the robot using the odometry based motion model, with our control representation being ($u =< \delta_{rot1}, \delta_{trans}, \delta_{rot2}$). We assume that there are landmarks present in the robot's environment. The robot receives the bearings to the landmarks and the ID of the landmarks as observations: $z =$<bearing, landmark ID>. Additionally, we assume a noise model for the odometry motion model with parameters α (Table 5.6 of the book) and a separate noise model for the bearing observations with parameter β (Section 6.6 of the book). The landmark ID observation is noise free.

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. This information needs to be used by you to localize the robot over the whole time-sequence, by using an EKF or a PF.

2.1 Extended Kalman Filter

In the Extended Kalman Filter, you will estimate a Gaussian approximation of the robot state at each time $\mathcal{N}(\mu_t, \Sigma_t)$, based on the distribution at the previous time $\mathcal{N}(\mu_{t-1}, \Sigma_{t-1})$, the applied control ($u_{t-1}$) and the observation ($z_t$). Alg. 1 provides a high-level pseudocode for the EKF update (refer to the slides for a detailed explanation).

Algorithm 1 Extended Kalman Filter - TO BE IMPLEMENTED in ekfUpdate.m and run.m

```
Inputs: State: $\mu_{t-1}, \Sigma_{t-1}$, Control: $u_{t-1}$, Observation: $z_t$, Noise parameters: $\alpha, \beta$
Output: Next State: $\mu_t, \Sigma_t$
Prediction step:
  Prediction mean using motion model: $\bar{\mu}_t = g(\mu_{t-1}, u_{t-1})$        ▷ Use prediction.m
  Motion model Jacobians $G_t$ and $V_t$        ▷ From written assignment
  Motion noise $M_t$ using parameters $\alpha$        ▷ Use noiseFromMotion.m
  Prediction co-variance: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T$
Correction step:
  Expected observation at $\hat{z}_t$        ▷ Use observation.m
  Observation Jacobian $H_t$ and noise $Q_t = \beta$        ▷ $Q_t$ = bearing noise ($\beta$). ID noise = 0
  Predicted observation co-variance: $S_t = H_t \bar{\Sigma}_t H_t^T + Q_t$
  Kalman Gain: $K = \bar{\Sigma}_t H_t^T S_t^{-1}$
  Next state mean: $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$        ▷ Use minimizedAngle.m to limit angle differences to -pi to pi
  Next state co-variance: $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
  Compute likelihood of observation (pOfZ)        ▷ Use likelihood.m
```

You need to implement the EKF update steps in the file **ekfUpdate.m**. You also need to change the file **run.m** to ensure that the outputs of the EKF run are not reset. For additional experiments and plots, you can look at the files **run.m, runExperiments.m**.

You should get the following results for a successful implementation of the EKF on MATLAB/Octave (results not same due to random number generators):

```
MATLAB:
```

```
run(200, false, 0.001, true) -->
meanPositionError = 7.2436
meanMahalanobisError = 3.0394
ANEES = 1.0131
meanPOfZ = 2.1214

Octave:
run(200, false, 0.001, true) -->
meanPositionError = 5.2527
meanMahalanobisError = 1.9864
ANEES = 0.66212
meanPOfZ = 2.1756
```

2.2 Particle Filter

Unlike the EKF which represents the state as a Gaussian, the particle filter represents the state distribution via sampled particles. At each time t, the particle filter takes as input the set of particles from the previous time p_{t-1}, uses the control u_t and the (noisy) motion model to propagate these particles forward to generate candidate particles for the next step ($\bar{p}_t$). These particles are then weighted based on the observed measurement z_t and a new set of particles are sampled from this weighted set of particles. These particles are used as the estimates for the state distribution at the current step p_t. Alg: 2 provides a high-level pseudocode for the EKF update (refer to the slides for a detailed explanation).

Algorithm 2 Particle Filter - TO BE IMPLEMENTED in pfUpdate.m, resample.m and run.m

```
**Inputs**: Particles: $p^1_{t-1}, p^2_{t-1}, \ldots, p^n_{t-1}$, Control: $u_{t-1}$, Observation: $z_t$, Noise parameters: $\alpha, \beta$
**Output**: Particles at next step: $p^1_t, p^2_t, \ldots, p^n_t$
**Prediction step:**
  Propagate particles through noisy motion model $\forall i,\ \bar{p}^i_t = g(p^i_{t-1}, u_{t-1}, \alpha)$    ▷ Use sampleOdometry.m
**Correction step:**
  Compute importance weight for all particles    ▷ Use observation.m and likelihood.m
  Normalize importance weights
  Resample based on importance weights - Systematic Resampling    ▷ To be implemented in resample.m
  Compute mean, var and avg. observation likelihood (pOfZ) for predicted / final particles ▷ Use meanAndVariance.m
```

For the particle filter, **you need to implement** the overall algorithm in **pfUpdate.m** as well as the resampling step in **resample.m**. We will use the systematic resampling technique discussed in class for this assignment. As before, you also need to change the file **run.m** to ensure that the outputs of the EKF run are not reset. For additional experiments and plots, you can look at the files **run.m, runExperiments.m**.

You should get the following results for a successful implementation of the PF on MATLAB/Octave (results not same due to random number generators):

```
MATLAB:
run(200, true, 0.001, true) -->
meanPositionError = 8.2722
meanMahalanobisError = 8.2194
ANEES = 2.7398
```

```
meanPOfZ = 2.1447

Octave:
run(200, true, 0.001, true) -->
meanPositionError = 7.1791
meanMahalanobisError = 7.5702
ANEES = 2.5234
meanPOfZ = 2.1626
```

2.3 Writeup

In addition to the code, **you need to provide a writeup** analyzing the performance of your filters under various conditions. The file **runExperiments.m** can be quite useful for generating these tables and plots. Your writeup **must** include the following:

- A plot showing the noise free path, real robot path, and filter path for each filter under the default (provided) parameters. *(2 plots total).*
- A table of values and corresponding plots of the mean position error as the alpha and beta factors range over $r = [1/64, 1/16, 1/4, 1, 4, 16, 64]$ (note that this is between 1/8 and 8 times the default noise values) for both filters. This means that one run should be, for example: **run(200, false, 0.001, true, false, [1/64,1/64,1/64,1/64])** *(2 tables, 2 plots).*
- Tables and plots of mean position error and ANEES as noise for data + filter vary over r and the number of particles varies over $[20, 100, 300]$. For example, one run should be **run(200, true, 0.001, true, false, [1/64,1/64,1/64,1/64], 20)**. Note that you have already produced values for $n = 100$ earlier. *(2 additional tables, 2 additional plots).*
- **Extra Credit:** Tables and plots for both filters of mean position error, ANEES, and pOfZ in which the actual data has the default noise, but the filter noise estimates range over r. For example, one run should be **run(200, false, 0.001, true, false, [1,1/64,1,1/64])**. *(6 tables, 6 plots).*
- **Extra Credit:** For each set of tables and plots, you should comment on what you see! What trends do you see? Which filters perform better under which conditions?
- **Extra Credit:** Give an example of when particle starvation can occur for the particle filter. How can this be avoided?

2.4 Code overview

This section gives a brief overview of the functions available to you in the code:

```
Things to implement:
run.m              -- Main update loop, should call ekfUpdate and pfUpdate
ekfUpdate.m        -- EKF update
pfUpdate.m         -- Particle filter update
resample.m         -- Particle filter resampling, called by pfUpdate
runExperiments.m   -- Useful later for running multiple experiments
```

```
Tools:
You should not need to use these files, but look at them if you like:
generateScript.m   -- Generates data according to initial mean and noise parameters
generateMotion.m   -- Simulates simple motion commands

You may find these files useful:
prediction.m       -- Deterministically move robot according to given motion (odometry motion model)
observation.m      -- Returns the observation of the specified marker given the current state
sampleOdometry.m   -- Noisy samples from the odometry motion model (Implements Table 5.6 from book)
sample.m           -- Generates samples from a covariance matrix
meanAndVariance.m  -- Mean and var for a set of unweighted samples (illustrates handling angles)
getfieldinfo.m     -- Gets field information
minimizedAngle.m   -- Normalizes an angle to [-pi, pi] (use this when dealing with angles)
endPoint.m         -- Returns the location of an observation
noiseFromMotion.m  -- Generate motion noise variances based on alphas
matlab.el          -- Customization file for emacs

Display functions:
plotcircle.m       -- Draws a circle
plotcov2d.m        -- Draws a 2-D covariance matrix
plotfield.m        -- Draws the field with landmarks
plotmarker.m       -- Draws an 'x' at a specified point (useful for drawing samples)
plotrobot.m        -- Draws the robot
plotSamples.m      -- Plots particles from the pf
plotLine.m         -- Plot a ray (origin, angle, length)

Data format (see run.m and generateScript.m):
State              : [x,y,θ];
Observation        : [bearing to landmark, landmark ID];
Control            : [δ_rot1,δ_trans,δ_rot2];
```

2.4.1 Hints

- Make sure to call **minimizedAngle()** any time an angle or angle difference could exceed [-pi,pi].
- Try visualizing the extra co-variance matrices returned by the EKF.
- Turn off plotting for a significant speedup. Enclose all plotting commands within blocks so they can be turned off with the run parameter.
- It's easy to visualize multiple plots. You can also zoom in on a plot (when it's static, for example when the pause time is negative).
- Make sure to use the low variance systematic sampler from the textbook / slides. It gives you smoother particle distribution, and also requires only a single random number per resampling step. This will make your runs consistent with the reference implementation.

3 Submission

You will be using Catalyst dropbox (`https://catalyst.uw.edu/collectit/dropbox/summary/barun/38890`) for submitting the homework. You need to submit the code and plots including any helper functions you use. More instructions for submission will be available in the dropbox. The written assignments can be submitted either in writing or electronically along with the code.