EECE 5550 Mobile Robotics HW #4

Due: March 28, 2023

Please upload a single pdf with your answers and code to Gradescope.

Problem 1: Extended Kalman Filter

Assume that you have a robot that moves with constant velocity over a 2D space. Assume further that you have two landmarks $\mathbf{l}_1 = [l_x^1, l_y^1]$ and $\mathbf{l}_2 = [l_x^2, l_y^2]$ at *known positions* in the environment. At each time step, the robot observes both landmarks with some sensor noise. You will formulate an Extended Kalman Filter (EKF) estimator for this problem by answering the following parts. (There is no implementation required for this question.)

(a) Formally define the discrete-time dynamical model of a robot moving with **constant** velocity over a 2D space. Its state vector is $\mathbf{x_t} = [p_x, v_x, p_y, v_y]$, where p_x, v_x are the position and velocity in x axis, p_y, v_y are the position and velocity in y axis. You should also consider some (Gaussian) noise in the dynamical model. For any variable you define, write their domain as well (for example, if you define a matrix R, you should indicate it as $R \in \mathbb{R}^{n \times n}$.)

(b) Formally define the observation model which consists of the Euclidean distances from the robot's current position (p_x, p_y) to each of the two landmarks (l_x^i, l_y^i for $i = 1, 2$) plus (Gaussian) noise.

(c) As we saw in class, an EKF requires linearization of the nonlinear (dynamical and/or observation) models and then uses these linearized models in the main steps of the Kalman Filter. Based on your findings from parts (a) and (b), define the Jacobians of the models to obtain the linearized models.

(d) Finally, based on your findings from parts (a), (b) and (c), apply the propagation and update steps of EKF and write the updates of the covariances.

Problem 2: Scan matching using Iterative Closest Point

We saw in class that laser scan matching can be used to recover high-precision estimates of the relative transformation between two sensor frames. In practice, this is often used to produce estimates of robot motion (i.e. odometry) that are *far* more accurate than what can be achieved using e.g. wheel odometry or IMU integration.

In this exercise, you will implement the Iterative Closest Point (ICP) algorithm, and use it to estimate the rigid transformation that optimally aligns two 3D pointclouds. Recall that given two pointclouds $X, Y \subset \mathbb{R}^d$ and an initial guess $(t_0, R_0) \in \mathrm{SE}(d)$ for the optimal rigid registration $y = Rx + t$ aligning X to Y, ICP (Algorithm 1) proceeds by alternating between the following two computations:

- Estimating a set of correspondences $C = \{(i_k, j_k)\}_{k=1}^K$ between points in the two pointclouds, given a guess for the optimal registration $(\hat{t}, \hat{R}) \in \mathrm{SE}(d)$ (lines 5–10),
- Computing a rigid registration $(\hat{t}, \hat{R}) \in \mathrm{SE}(d)$ that optimally aligns the set $\{(x_{i_k}, y_{j_k})\}_{k=1}^K$ of corresponding points in the least-squares sense:

$$(\hat{t}, \hat{R}) \in \operatorname*{argmin}_{(t,R)\in \mathrm{SE}(d)} \sum_{k=1}^{K} \|y_{j_k} - (Rx_{i_k} + t)\|_2^2. \tag{1}$$

  Recall that Horn's method (Algorithm 2) provides an efficient algorithm for computing the optimal registration $(\hat{t}, \hat{R}) \in \mathrm{SE}(d)$ in (1).

Algorithm 1 Iterative closest point

Input: Pointclouds $X = \{x_i\}_{i=1}^{n_X} \subset \mathbb{R}^d$ and $Y = \{y_i\}_{i=1}^{n_Y} \subset \mathbb{R}^d$, initial guess $(t_0, R_0) \in \mathrm{SE}(d)$ for a rigid registration aligning pointcloud X to pointcloud Y, maximum admissible distance $d_{\max} > 0$ for associating points.

Output: An estimated set $C = \{(i_k, j_k)\}_{k=1}^K$ of correspondences between points in X and Y, and the rigid registration $T = (\hat{t}, \hat{R}) \in \mathrm{SE}(d)$ that optimally aligns corresponding points of X and Y in the least-squares sense (1).

```
1: function ICP($X, Y, t_0, R_0, d_{\max}$)
2:     Initialization: $\hat{t} \leftarrow t_0$, $\hat{R} \leftarrow R_0$.
3:     repeat
4:         Initialize list of empty correspondences: $C \leftarrow \varnothing$, $K \leftarrow 0$.
5:         for $i = 1, \ldots, n_X$ do:                    ▷ Estimate point correspondences
6:             Find closest point $y_j$ in $Y$ to image of $x_i$ under transformation $(\hat{t}, \hat{R})$:
                   $$y_j = \operatorname*{argmin}_{y \in Y} \|y - (\hat{R}x_i + \hat{t})\|_2^2. \tag{2}$$
7:             if $\|y_j - (\hat{R}x_i + \hat{t})\|_2 < d_{\max}$ then
8:                 Add $x_i$ and $y_j$ to the list of point correspondences: $C \leftarrow C \cup \{(i, j)\}$.
9:             end if
10:        end for
11:        Compute optimal registration given the point correspondences $C$:
                   $$(\hat{t}, \hat{R}) \leftarrow \textsc{ComputeOptimalRigidRegistration}(X, Y, C) \tag{3}$$
12:    until convergence return $(\hat{t}, \hat{R}, C)$
13: end function
```

(a) Implement a function `EstimateCorrespondences`(X, Y, t, R, $d_{\max}$) that constructs the list $C = \{(i_k, j_k)\}_{k=1}^K$ of estimated point correspondences, using the procedure of lines 5–10 in Algorithm 1. Your function should take as input the pointclouds $X, Y \subset \mathbb{R}^d$, the estimated rigid transformation $(t, R) \in \mathrm{SE}(d)$ aligning X to Y, and the maximum admissible distance $d_{\max}$ for associating two points, and return a list $C = \{(i_k, j_k)\}_{k=1}^K$ of estimated point correspondences.

(b) Implement the function `ComputeOptimalRigidRegistration`(X, Y, C) shown in Algorithm 2.

Algorithm 2 Optimally aligning pointclouds with *known* point correspondences

```
Input: Pointclouds $X = \{x_i\}_{i=1}^{n_X} \subset \mathbb{R}^d$ and $Y = \{y_j\}_{j=1}^{n_Y} \subset \mathbb{R}^d$, list of point correspondences
    $C = \{(i_k, j_k)\}_{k=1}^{K}$.
Output: Rigid transformation $T = (\hat{t}, \hat{R}) \in \mathrm{SE}(d)$ that optimally aligns corresponding points of
    $X$ and $Y$ in the least-squares sense (1).
1: function ComputeOptimalRigidRegistration($X, Y, C$)
2:     Calculate pointcloud centroids:
           $$\bar{x} \triangleq \frac{1}{K}\sum_{k=1}^{n} x_{i_k}, \qquad \bar{y} \triangleq \frac{1}{K}\sum_{k=1}^{K} y_{j_k}. \tag{4}$$
3:     Calculate deviations of each point from the centroid of its pointcloud:
           $$x'_{i_k} \triangleq x_{i_k} - \bar{x}, \qquad y'_{j_k} \triangleq y_{j_k} - \bar{y}. \tag{5}$$
4:     Compute cross-covariance matrix $W$:
           $$W \triangleq \frac{1}{K}\sum_{k=1}^{K} y'_{j_k}(x'_{i_k})^{\mathsf{T}}. \tag{6}$$
5:     Compute singular value decomposition: $W = U\Sigma V^{\mathsf{T}}$.
6:     Construct optimal rotation:
           $$\hat{R} \triangleq U\,\mathrm{Diag}(1, \dots, 1, \det(UV)))V^{\mathsf{T}}. \tag{7}$$
7:     Recover optimal translation: $\hat{t} \triangleq \bar{y} - \hat{R}\bar{x}$.
8:     return $(\hat{t}, \hat{R})$.
9: end function
```

(c) With the aid of your results from parts (a) and (b), write a function that implements the ICP algorithm (Algorithm 1). Your implementation should accept as input the two pointclouds X and Y, an initial guess $(t_0, R_0) \in \mathrm{SE}(d)$ for the rigid transformation aligning X to Y, the maximum admissible distance $d_{\max}$ for associating points, *and* the number `num_ICP_iters` of ICP iterations (lines 3–12 of Algorithm 1) to perform.

(d) Now you will apply your ICP implementation to register two 3D pointclouds. Download the files `pclX.txt` and `pclY.txt` containing the pointclouds from the course Canvas website, and run your ICP implementation with the following parameters:

- Set the initial estimate for the optimal registration to the identity: $(t_0, R_0) = (0, I_3) \in \mathrm{SE}(3)$.
- Set $d_{\max} = .25$
- Set the number of ICP iterations to 30: `num_ICP_iters` $= 30$.

Check your results by plotting the pointcloud Y and the image of X under the estimated transformation $(\hat{t}, \hat{R}) \in \mathrm{SE}(3)$ on the same axes, using different colors for the two pointclouds, and compute the root-mean-squared error for the associated points under this registration:

$$\mathrm{RMSE} = \sqrt{\frac{1}{K}\sum_{k=1}^{K} \|y_{j_k} - (\hat{R}x_{i_k} + \hat{t})\|_2^2}. \tag{8}$$

Then report and submit the following:

- The parameters $(\hat{t}, \hat{R}) \in \mathrm{SE}(3)$ for the estimated rigid transformation,
- The RMSE (8) for the estimated point correspondences,
- The plot showing the co-registered pointclouds,
- Your code.

Problem 3: State estimation by particle filtering on a Lie group

We saw in class that the Particle Filter provides a simple yet highly versatile algorithm for performing recursive Bayesian estimation, and is especially well-suited to state estimation problems with nonlinear state transition or measurement functions or non-Gaussian models of uncertainty.

In this exercise, you will apply particle filtering to perform state estimation over a Lie group: specifically, you will design and implement a particle filter to track the pose of a differential-drive ground robot.

Algorithm 3 Particle filter propagation

```
Input: Particle set $X_t = \{x_t^{[i]}\}_{i=1}^n$ sampled from belief $p(x_t)$ over initial state $x_t$, control $u_t$.
Output: Particle set $X_{t+1} = \{x_{t+1}^{[i]}\}_{i=1}^n$ sampled from belief $p(x_{t+1}|u_t)$ over subsequent state
    $x_{t+1}$ after applying control $u_t$.
1: function ParticleFilterPropagate($X_t, u_t$)
2:     Initialize empty particle set: $X_{t+1} \leftarrow \varnothing$.
3:     for $i = 1, \ldots, n$ do
4:         Draw sample $x_{t+1}^{[i]}$ from the motion model $p(x_{t+1}|x_t, u_t)$:
               $x_{t+1}^{[i]} \sim p(x_t|x_t^{[i]}, u_t).$        (9)
5:         Add sample $x_{t+1}^{[i]}$ to the output particle set: $X_{t+1} \leftarrow X_{t+1} \cup \{x_{t+1}^{[i]}\}$.
6:     end for
7:     return $X_{t+1}$.
8: end function
```

(a) The state propagation step of the particle filter (Algorithm 3) requires a procedure for sampling from the motion model $p(x_{t+1}|x_t, u_t)$ (cf. line 4). Recall from our earlier discussion (in Lecture 5) that differential drive robots are actuated by controlling the velocities $(\dot{\varphi}_l, \dot{\varphi}_r)$ of their left and right wheel speeds.

Suppose that we can only control the wheel speeds of our robot *imprecisely*; that is, the *true* left and right wheel speeds $(\tilde{\varphi}_l, \tilde{\varphi}_r)$ are related to the *commanded* wheel speeds $u \triangleq (\dot{\varphi}_l, \dot{\varphi}_r)$ according to:

$$\begin{aligned} \tilde{\varphi}_l &= \dot{\varphi}_l + \epsilon_l, \quad \epsilon_l \sim \mathcal{N}(0, \sigma_l^2), \\ \tilde{\varphi}_r &= \dot{\varphi}_r + \epsilon_r, \quad \epsilon_r \sim \mathcal{N}(0, \sigma_r^2). \end{aligned} \tag{13}$$

Using (13), derive a generative description (i.e., a list of steps that provide you with samples from) for the motion model $p(x_{t_2}|x_{t_1}, \dot{\varphi}_l, \dot{\varphi}_r, r, w, \sigma_l, \sigma_r)$ that parameterizes the distribution of the pose of the robot $x_{t_2} \in \mathrm{SE}(2)$ at time t_2 as a function of its pose $x_{t_1} \in \mathrm{SE}(2)$ at time t_1 given the *commanded* wheel speeds $(\dot{\varphi}_l, \dot{\varphi}_r)$, its wheel radius r and track width w, and the variances σ_l and σ_r for the true wheel speeds.

Algorithm 4 Particle filter update

Input: Particle set $X_t = \{x_t^{[i]}\}_{i=1}^n$ sampled from prior belief $p(x_t)$ over x_t, measurement z_t.

Output: Particle set $\bar{X}_t = \{\bar{x}_t^{[i]}\}_{i=1}^n$ sampled from posterior belief $p(x_t|z_t)$ over x_t after incorporating measurement z_t.

```
1:  function ParticleFilterUpdate($X_t, z_t$)
2:      for $i = 1, \ldots, n$ do                              ▷ Calculate importance weights
3:          Calculate importance weight for $x_t^{[i]}$ using measurement likelihood function:
                $$w_i \triangleq p(z_t|x_t^{[i]}). \quad (10)$$
4:      end for
5:      Initialize empty particle set: $\bar{X}_t \leftarrow \varnothing$.
6:      for $i = 1, \ldots, n$ do                              ▷ Importance-weighted resampling
7:          Sample $i$th particle $\bar{x}_t^{[i]}$ from prior particle set $X_t$ with replacement:
                $$\bar{x}_t^{[i]} \sim p\left(\bar{x}_t | X_t, \{w_k\}_{k=1}^n\right), \quad (11)$$
            with sampling probabilities proportional to importance weights:
                $$p\left(\bar{x}_t = x_t^{[k]} \mid X_t, \{w_k\}_{k=1}^n\right) \propto w_k. \quad (12)$$
8:          Add sample $\bar{x}_t^{[i]}$ to the posterior particle set: $\bar{X}_t \leftarrow \bar{X}_t \cup \{\bar{x}_t^{[i]}\}$.
9:      end for
10:     return $\bar{X}_t$.
11: end function
```

For the noiseless case, we have derived the following expressions (14)-(15) for you. Your job is to extend these to the noisy case above. The robot's velocity, $\dot{\Omega}$, at $I \in \mathrm{SE}(2)$ as a function of the wheel speeds is

$$\dot{\Omega} \colon \mathbb{R}^2 \to \mathrm{Lie}(\mathrm{SE}(2))$$
$$\dot{\Omega}(\dot{\varphi}_l, \dot{\varphi}_r) = \begin{pmatrix} 0 & -\frac{r}{w}(\dot{\varphi}_r - \dot{\varphi}_l) & \frac{r}{2}(\dot{\varphi}_r + \dot{\varphi}_l) \\ \frac{r}{w}(\dot{\varphi}_r - \dot{\varphi}_l) & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{14}$$

and the robot's trajectory $\gamma(t)$ is the *integral curve* that starts at the pose $X_0 \in \mathrm{SE}(2)$ at time $t = 0$,

$$\gamma(t) = X_0 \exp\left(t\dot{\Omega}(\dot{\varphi}_l, \dot{\varphi}_r)\right). \tag{15}$$

(b) Similarly, the measurement update step of the particle filter (Algorithm 4) requires a procedure for evaluating the measurement likelihood function $p(z_t|x_t)$ (cf. line 3).

Suppose that our robot is equipped with a sensor that is capable of producing a noisy measurements of its position in the plane (for example, a GPS receiver). Specifically, the measurement $z_t \in \mathbb{R}^2$ at time t is related to the robot's pose $x_t \triangleq (l_t, R_t) \in \mathrm{SE}(2)$ at time t according to:

$$z_t = l_t + \epsilon_p, \quad \epsilon_p \sim \mathcal{N}(0, \sigma_p^2 I_2). \tag{16}$$

Derive a closed-form expression (i.e., PDF) for the measurement likelihood function $p(z_t|x_t)$ under the measurement model (16).

(c) Write a (code) function that implements the particle filter propagation step (Algorithm 3) for a differential-drive ground robot. Your function should accept as input the following data:

- The current time t_1
- The particle set $X_{t_1} = \{x_{t_1}^{[i]}\}_{i=1}^n \subset \mathrm{SE}(2)$ describing the robot's belief over its position at time t_1
- The commanded wheel speeds $(\dot{\varphi}_l, \dot{\varphi}_r)$
- The time t_2 at which to predict its next pose $x_{t_2} \in \mathrm{SE}(2)$
- The parameters r, w, σ_l, and σ_r of the robot

and return a particle set $X_{t_2} = \{x_{t_2}^{[i]}\}_{i=1}^n \subset \mathrm{SE}(2)$ describing the robot's belief over its pose at time t_2, sampled according to the generative model you derived in part (a).

(d) Write a (code) function that implements the particle filter update step (Algorithm 4). Your function should accept as input:

- The particle set $X_t = \{x_t^{[i]}\}_{i=1}^n \subset \mathrm{SE}(2)$ representing its prior belief over its pose
- A noisy position z_t measurement sampled according to the generative model (16),
- The magnitude σ_p of the measurement noise,

and return the particle set $\bar{X}_t = \{x_t^{[i]}\}_{i=1}^n \subset \mathrm{SE}(2)$ modeling the robot's *posterior* belief after incorporating the measurement z_t.

The following parts (e)-(g) are extra credit. In the next series of experiments, we will assume the following (fixed) parameter values: $\dot{\varphi}_l = 1.5$, $\dot{\varphi}_r = 2$, $r = .25$, $w = .5$, $\sigma_l = .05$, $\sigma_r = .05$, and $\sigma_p = .10$.

(e) Using the particle filter propagation function that you implemented in part (c), generate $N = 1000$ realizations of x_{10} (the pose of the robot at time $t = 10$) from $p(x_{10} | x_0, \dot{\varphi}_l, \dot{\varphi}_r, r, w, \sigma_l, \sigma_r)$, assuming that $x_0 = (0, I_2) \in \mathrm{SE}(2)$ (i.e. that the robot starts at the origin at time $t = 0$). Calculate the empirical mean and covariance of the positions of the points in this particle set, and plot the positions of the sampled particles in the plane.

(f) In this part of the exercise, you will simulate the evolution of our differential drive robot's belief over its pose while navigating using *dead reckoning.*

Starting with an initial particle set X_0 consisting of $N = 1000$ copies of $I = (0, I_2) \in \mathrm{SE}(2)$ (indicating absolute certainty that the robot's initial pose x_0 is the origin), apply your particle filter propagation function from part (c) to recursively generate sample-based approximations to the robot's belief over its pose $x_t \in \mathrm{SE}(2)$ at times $t \in \{5, 10, 15, 20\}$.

For each of these sample sets, report the empirical mean and covariance of the particles' positions. Finally, plot the positions of the particles in each of these sample sets in a *single* plot, using a different color for each sample set.

(g) Now let us consider the effect of incorporating noisy measurements z_t from the model (16) on the robot's uncertainty over its position.

Staring with an initial particle set X_0 containing $N = 1000$ copies of $I = (0, I_2) \in \mathrm{SE}(2)$ (indicating absolute certainty that the robot's initial pose x_0 is the origin), apply your particle filter propagation and update functions from parts (c) and (d) to recursively generate sample-based approximations of the robot's *posterior* beliefs over its pose $x_t \in \mathrm{SE}(2)$ at times $t \in \{5, 10, 15, 20\}$ obtained *after* incorporating the sequence of measurements:

$$z_5 = \begin{pmatrix} 1.6561 \\ 1.2847 \end{pmatrix}, \quad z_{10} = \begin{pmatrix} 1.0505 \\ 3.1059 \end{pmatrix}, \quad z_{15} = \begin{pmatrix} -0.9875 \\ 3.2118 \end{pmatrix}, \quad z_{20} = \begin{pmatrix} -1.6450 \\ 1.1978 \end{pmatrix}. \tag{17}$$

Report the empirical mean and covariance of the particles' positions in each of the posterior sample sets, and plot the positions of all of these particles in a *single* plot, using a different color for each sample set.