Autonomous Planning Under Uncertainty (0970252)
Homework #2

Submission guidelines:

- Submission in pairs by: 29 November 2025, 23:59.
- Email submission as a *single zip or pdf file* to moshiktech@gmail.com.
- Make sure to name the submitted file ID1-ID2.zip (or .pdf) with ID1,ID2 - the students' ids.
- For hands-on parts, your source code should be submitted as well (include it in the above zip file).

Theoretical questions

1. Consider a function $f : \mathbb{R}^n \to \mathbb{R}$, and a random variable $X \in \mathbb{R}^n$ with probability density function $p(x)$. In the following sections we will be interested in estimating $\mathbb{E}[f(X)]$.

   (a) Assume $\{x^i\}_{i=1}^N$ are sampled independently from $p(x)$. Show that the vanila Monte Carlo estimator $\hat{\mathbb{E}}_N^{MC}[f(x)] = \frac{1}{N}\sum_{i=1}^N f(x^i)$ is unbiased [1].

   (b) Assume $\{x^i\}_{i=1}^N$ are sampled independently from some proposal distribution $q(x)$. Consider the importance sampling estimator $\hat{\mathbb{E}}_N^{IS}[f(x)] = \frac{1}{N}\sum_{i=1}^N w_i \cdot f(x^i)$, where $w_i = \frac{p(x^i)}{q(x^i)}$.

      i. which conditions should $q(x)$ satisfy in order to ensure that $\hat{\mathbb{E}}_N^{IS}[f(x)]$ is valid?

      ii. show that the importance sampling estimator is unbiased if the conditions in the previous section are satisfied.

   (c) Consider the self normalized importance sampling estimator $\hat{\mathbb{E}}_N^{SN}[f(x)] = \sum_{i=1}^N \tilde{w}_i \cdot f(x^i)$ where $\tilde{w}_i = \frac{w_i}{\sum_{j=1}^N w_j}$. Is the self normalized importance sampling estimator unbiased? Justify your answer.

[1] An estimator is said to be unbiased if its expected value is equal to the true value of the parameter being estimated. i.e. $\mathbb{E}[\hat{\theta}] = \theta$.

Hands-on tasks

You are highly encouraged to use the Julia language for all hands-on tasks. However, this is not compulsory. In case you use Julia, the supplied skeleton may be helpful.

- **Note 1:** Your implementations might be useful in future homeworks as well, therefore it is recommended to keep them organized and well documented.
- **Note 2:** In all sections that require sampling from random distributions or random number generators, use a consistent random number generator with one of your IDs as the fixed seed from the beginning of your code, and use it in sampling methods. Make sure your code is reproducible (outputs the same results every time it runs).

In this exercise, we will implement a Bootstrap Particle Filter from scratch, do not use the Julia `ParticleFilters` package or similar packages in this exercise.

1. Consider a mobile robot navigating in a continuous 2D environment. For simplicity, we consider only the position state of the robot, i.e. $X_i \in \mathbb{R}^2$ denotes the x-y robot position at time instant i. The motion and observation models, $\mathbb{P}(X_{k+1} \mid X_k, a_k)$ and $\mathbb{P}(z_k \mid X_k)$, are given by

$$X_{k+1} = f(X_k, a_k) + w, \tag{1}$$

$$z_k = h(X_k) + v \tag{2}$$

where w and v are transition and observation noise terms, with $w \sim \mathcal{N}(0, \Sigma_w)$ and $v \sim \mathcal{N}(0, \Sigma_v)$.

In this exercise, we shall consider simple linear models, i.e. $f(X_k, a_k) = F \cdot X_k + a_k$ with $F = I_{2\times 2}$ and $a_k \in \mathbb{R}^2$ is the commanded position displacement. We also consider position measurements such that $z_k \in \mathbb{R}^2$ and $h(X_k) \equiv X_k$. Consider a prior over robot position at time instant 0 is available: $b(X_0) \doteq \mathbb{P}(X_0) = \mathcal{N}(\mu_0, \Sigma_0)$.

   (a) Implement a function `InitParticleBelief` that initializes a particle belief.
   - Input: initial belief b_0, number of particles N.
   - Output: a particle belief.

   (b) Implement a function `SampleMotionModel` that, given a robot state X and action a, generates the next state, i.e. $X' \sim \mathbb{P}(X' \mid X, a)$, according to the motion model in Eq. (1).
   - Input: State X, action a
   - Output: Next state X'

   (c) Implement a function `GenerateObservation`, which generates an observation according to the observation model in Eq. (2), i.e. $z \sim \mathbb{P}(z \mid X)$.
   - Input: State X
   - Output: Observation z

   (d) Implement a function `PropagateParticleBelief`, which updates a particle belief according to a (given) performed action, considering a recursive setting.
   - Input: a particle belief from some time instant k , action a_k.
   - Output: an updated particle belief at the next time step.

   (e) Implement a function `ObsLikelihood` that calculates the likelihood of an observation given the state of the robot.
   - Input: observation z, robot state X.
   - Output: likelihood of the observation.

   (f) Implement a function `PosteriorParticleBelief`, which updates a particle belief according to a (given) performed action and captured observation, considering a recursive setting.
   - Input: a particle belief from some time instant k , action a_k and observation z_{k+1}.
   - Output: an updated posterior particle belief at the next time step.

   (g) Implement a function `ResampleParticles`, which resamples particles according to their weights.
   - Input: a particle belief, an ESS threshold.

- Output: a resampled particle belief.

(h) Consider $b(X_0) = \mathcal{N}(\mu_0, \Sigma_0)$ with $\mu_0 = (0,0)^T$ and $\Sigma_0 = I_{2\times2}$, $\Sigma_w = 0.1^2 \cdot I_{2\times2}$ and $\Sigma_v = I_{2\times2}$. Let the actual initial robot location (ground truth) be $X_0 = (-0.5, -0.2)^T$. The belief is represented by a fixed number of particles $N = 10$.

  i. Generate a possible robot trajectory $\tau \doteq \{X_0, X_1, \ldots, X_T\}$ where the robot starts at X_0 and follows a given action sequence $a_{0:T-1} \doteq \{a_0, a_1, \ldots, a_{T-1}\}$. Assume $T = 10$, and $a_i \doteq (1,1)^T$ for $i \in [0, T-1]$.

  ii. Generate observations along the trajectory τ, by generating a single z_i for each $X_i \in \tau$.

  iii. Calculate propagated beliefs along the trajectory τ while only considering a motion model (without observations). Draw on a plot the actual trajectory and the propagated particle beliefs[2] of each time step.

  iv. Calculate the *posterior* beliefs along the trajectory τ, this time also considering observations (generated observations in 1(h)ii), but *without resampling*. Draw on a separate plot the actual trajectory, and the posterior particle beliefs of each time step.

  v. Calculate the *posterior* beliefs along the trajectory τ, this time also considering observations and *resampling*. Draw on a separate plot the actual trajectory, and the posterior particle beliefs of each time step.

2. We now modify the problem setting as follows. Several beacons are scatted in the environment. The locations of these beacons are known to the robot (denote it by X_j^b for the jth beacon), and represented by a matrix $X^b \in \mathbb{R}^{2\times n}$, i.e. $X^b = [X_1^b, \ldots, X_n^b]$, where n is the number of beacons.

   Assume the robot gets a relative position observation $z^{rel} \in \mathbb{R}^2$ from a beacon if it is sufficiently close to it (distance less than d). For simplicity, consider the beacons are positioned sufficiently apart such that an observation from only one of the beacons can be obtained for every possible robot location. The measurement covariance Σ_v is fixed.

(a) Write the corresponding observation model for the measurement z^{rel}, as a function of robot location X, beacon location X^b, and distance threshold d.

(b) Implement a function `GenerateObservationFromBeacons`, which generates an observation z^{rel} according to the observation model from clause 2a.

  - Input: Robot location X, beacon locations X^b, beacon distance threshold d.
  - Output: Measurement z^{rel} and index of beacon within range; null if no beacon is within range.

(c) Implement a function `ObsFromBeaconsLikelihood` that calculates the likelihood of an observation given the state of the robot and the beacon locations.

  - Input: observation z^{rel}, robot state X, beacon locations X^b.
  - Output: likelihood of the observation.

(d) Implement a function `PosteriorParticleBeliefBeacons`, which updates a particle belief according to a (given) performed action and a captured observation from the observation model from clause 2a.

  - Input: a particle belief from some time instant k , action a_k and observation z_{k+1}^{rel}.
  - Output: an updated posterior particle belief at the next time step.

(e) Consider $n = 9$ beacons equally scattered in a 9×9 grid, and let the coordinate of its bottom left corner be $(0,0)$. Assume $b(X_0) = \mathcal{N}(\mu_0, \Sigma_0)$ with $\mu_0 = (0,0)^T$ and $\Sigma_0 = I_{2\times2}$, $\Sigma_w = 0.1^2 \cdot I_{2\times2}$ and $\Sigma_v = I_{2\times2}$. Let the actual initial robot location (ground truth) be $X_0 = (-0.5, -0.2)^T$. Consider an action sequence $a_{0:T-1} = \{a_0, \ldots, a_{T-1}\}$, with $a_i \doteq (1,1)^T$ for $i \in [0, T-1]$, for $T = 10$. Further, let $d = 1$. The belief is represented by a fixed number of particles $N = 10$, and should be initialized again at the start of this clause.

  i. Generate new observations from the observation model from clause 2a, along the trajectory τ from clause 1(h)i, by generating a single z_i for each $X_i \in \tau$.

  ii. Calculate the *posterior* beliefs using observations *without resampling* along the trajectory τ. Draw on a separate plot the beacons, the actual trajectory, and these posterior particle beliefs.

  iii. Calculate the *posterior* beliefs using observations *with resampling* along the trajectory τ. Draw on a separate plot the beacons, the actual trajectory, and these posterior particle beliefs.

[2] You can use the supplied `scatterParticles` function to draw the particle belief.

(f) We now consider the implications of the resampling stage in the particle filter algorithm.

   i. Why would we need a resampling stage in the particle filter algorithm?
   ii. What happens if we resample too often or too rarely?
   iii. Bonus (5 points): What would you suggest as a more efficient resampling method? Implement it and compare the results.