Project 3: Infinite-Horizon Stochastic Optimal Control

Collaboration in the sense of discussion is allowed, however, the assignment is individual and the work you do should be entirely your own. See the collaboration and academic integrity statement here: https://natanaso.github.io/ece276b. *Books, websites, and papers may be consulted but not copied from. It is absolutely forbidden to copy or even look at anyone else's code.* ***Please acknowledge in writing people you discuss the problems with.***

Submission

Please submit the following files on **Gradescope** by the deadline shown at the top right corner.

1. **Programming assignment**: upload all code you have written for the project and a README file with a clear, concise description of the main file and how to run your code.
2. **Report**: upload your report in pdf format. You are encouraged but not required to use an IEEE conference template[1] for your report. Please refer to Slide 13 and 14 of ECE 276A Lecture 1 for the expected report structure and content[2].

Project Description

This project considers safe trajectory tracking for a ground differential-drive robot. Consider a robot whose state $\mathbf{x}_t := (\mathbf{p}_t, \theta_t)$ at discrete-time $t \in \mathbb{N}$ consists of its position $\mathbf{p}_t \in \mathbb{R}^2$ and orientation $\theta_t \in [-\pi, \pi)$. The robot is controlled by a velocity input $\mathbf{u}_t := (v_t, \omega_t)$ consisting of linear velocity $v_t \in \mathbb{R}$ and angular velocity (yaw rate) $\omega_t \in \mathbb{R}$. The discrete-time kinematic model of the differential-drive robot obtained from Euler discretization of the continuous-time kinematics with time interval $\Delta > 0$ is:

$$\mathbf{x}_{t+1} = \begin{bmatrix} \mathbf{p}_{t+1} \\ \theta_{t+1} \end{bmatrix} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) = \underbrace{\begin{bmatrix} \mathbf{p}_t \\ \theta_t \end{bmatrix}}_{\mathbf{x}_t} + \underbrace{\begin{bmatrix} \Delta\cos(\theta_t) & 0 \\ \Delta\sin(\theta_t) & 0 \\ 0 & \Delta \end{bmatrix}}_{\mathbf{G}(\mathbf{x}_t)} \underbrace{\begin{bmatrix} v_t \\ \omega_t \end{bmatrix}}_{\mathbf{u}_t} + \mathbf{w}_t, \quad t = 0, 1, 2, \dots \tag{1}$$

where $\mathbf{w}_t \in \mathbb{R}^3$ models the motion noise with Gaussian distribution $\mathcal{N}(\mathbf{0}, \text{diag}(\boldsymbol{\sigma})^2)$ with standard deviation $\boldsymbol{\sigma} = [0.04, 0.04, 0.004] \in \mathbb{R}^3$. The motion noise is assumed to be independent across time and of the robot state $\mathbf{x}_t$. The kinematics model in (1) defines the probability density function $p_f(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ of $\mathbf{x}_{t+1}$ conditioned on $\mathbf{x}_t$ and $\mathbf{u}_t$ as the density of a Gaussian distribution with mean $\mathbf{x}_t + \mathbf{G}(\mathbf{x}_t)\mathbf{u}_t$ and covariance $\text{diag}(\boldsymbol{\sigma})^2$. The control input $\mathbf{u}_t$ of the vehicle is limited to an allowable set of linear and angular velocities $\mathcal{U} := [0, 1] \times [-1, 1]$.

The objective is to design a control policy for the differential-drive robot to track a desired reference position trajectory $\mathbf{r}_t \in \mathbb{R}^2$ and orientation trajectory $\alpha_t \in [-\pi, \pi)$ while avoiding collisions with obstacles in the environment. There are two circular obstacles $\mathcal{C}_1$ centered at $(-2, -2)$ with radius 0.5 and $\mathcal{C}_2$ centered at $(1, 2)$ with radius 0.5. Let $\mathcal{F} := [-3, 3]^2 \setminus (\mathcal{C}_1 \cup \mathcal{C}_2)$ denote the free space in the environment. The environment, the obstacles, and the reference trajectory are illustrated in Fig. 1.

It will be convenient to define an error state $\mathbf{e}_t := (\tilde{\mathbf{p}}_t, \tilde{\theta}_t)$, where $\tilde{\mathbf{p}}_t := \mathbf{p}_t - \mathbf{r}_t$ and $\tilde{\theta}_t := \theta_t - \alpha_t$ measure the position and orientation deviation from the reference trajectory, respectively. The equations of motion of the error state are:

$$\mathbf{e}_{t+1} = \begin{bmatrix} \tilde{\mathbf{p}}_{t+1} \\ \tilde{\theta}_{t+1} \end{bmatrix} = g(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{w}_t) = \underbrace{\begin{bmatrix} \tilde{\mathbf{p}}_t \\ \tilde{\theta}_t \end{bmatrix}}_{\mathbf{e}_t} + \underbrace{\begin{bmatrix} \Delta\cos(\tilde{\theta}_t + \alpha_t) & 0 \\ \Delta\sin(\tilde{\theta}_t + \alpha_t) & 0 \\ 0 & \Delta \end{bmatrix}}_{\tilde{\mathbf{G}}(\mathbf{e}_t)} \underbrace{\begin{bmatrix} v_t \\ \omega_t \end{bmatrix}}_{\mathbf{u}_t} + \begin{bmatrix} \mathbf{r}_t - \mathbf{r}_{t+1} \\ \alpha_t - \alpha_{t+1} \end{bmatrix} + \mathbf{w}_t. \tag{2}$$

[1] https://www.ieee.org/conferences_events/conferences/publishing/templates.html
[2] https://natanaso.github.io/ece276a/ref/ECE276A_1_Introduction.pdf

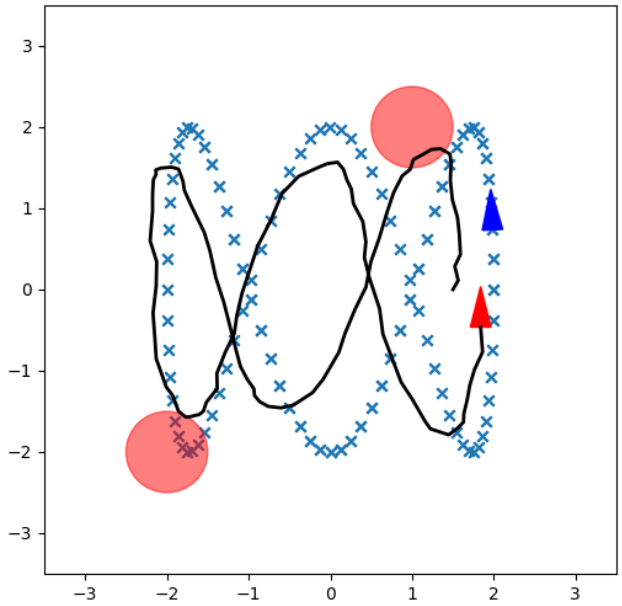

Figure 1: The objective of this project is to design a control policy for a differential-drive robot (red triangle) to track a reference trajectory (cyan crosses), while avoiding two circular obstacles (red). A baseline proportial-error controller is provided but the trajectory (black curve) it generates has a large tracking error and does not avoid obstacles.

We formulate the trajectory tracking with initial time τ and initial tracking error $\mathbf{e}$ as a discounted infinite-horizon stochastic optimal control problem:

$$\begin{aligned} V^*(\tau, \mathbf{e}) = \min_{\pi} \mathbb{E}\left[\sum_{t=\tau}^{\infty} \gamma^{t-\tau}\left(\tilde{\mathbf{p}}_t^\top \mathbf{Q}\tilde{\mathbf{p}}_t + q(1-\cos(\tilde{\theta}_t))^2 + \mathbf{u}_t^\top \mathbf{R}\mathbf{u}_t\right) \mid \mathbf{e}_\tau = \mathbf{e}\right] \\ \text{s.t.} \quad \mathbf{e}_{t+1} = g(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{w}_t), \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(\boldsymbol{\sigma})^2), \quad t = \tau, \tau+1, \dots \\ \mathbf{u}_t = \pi(t, \mathbf{e}_t) \in \mathcal{U} \\ \tilde{\mathbf{p}}_t + \mathbf{r}_t \in \mathcal{F} \end{aligned} \tag{3}$$

where $\mathbf{Q} \in \mathbb{R}^{2\times 2}$ is a symmetric positive-definite matrix defining the stage cost for deviating from the reference position trajectory $\mathbf{r}_t$, $q > 0$ is a scalar defining the stage cost for deviating from the reference orientation trajectory α_t, and $\mathbf{R} \in \mathbb{R}^{2\times 2}$ is a symmetric positive-definite matrix defining the stage cost for using excessive control effort. We will compare two different approaches for solving the problem in (3): (a) receding-horizon certainty equivalent control (CEC) and (b) generalized policy iteration (GPI).

Part 1: CEC is a suboptimal control scheme that applies, at each stage, the control that would be optimal if the noise variables $\mathbf{w}_t$ were fixed at their expected values (zero in our case). The main attractive characteristic of CEC is that it reduces a stochastic optimal control problem to a deterministic optimal control problem, which can be solved more effectively. Receding-horizon CEC, in addition, approximates an infinite-horizon problem by repeatedly solving the following discounted finite-horizon deterministic optimal control problem at each time step:

$$\begin{aligned} V^*(\tau, \mathbf{e}) \approx \min_{\mathbf{u}_\tau, \dots, \mathbf{u}_{\tau+T-1}} \mathfrak{q}(\mathbf{e}_{\tau+T}) + \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau}\left(\tilde{\mathbf{p}}_t^\top \mathbf{Q}\tilde{\mathbf{p}}_t + q(1-\cos(\tilde{\theta}_t))^2 + \mathbf{u}_t^\top \mathbf{R}\mathbf{u}_t\right) \\ \text{s.t.} \quad \mathbf{e}_{t+1} = g(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{0}), \quad t = \tau, \dots, \tau+T-1 \\ \mathbf{u}_t \in \mathcal{U} \\ \tilde{\mathbf{p}}_t + \mathbf{r}_t \in \mathcal{F} \end{aligned} \tag{4}$$

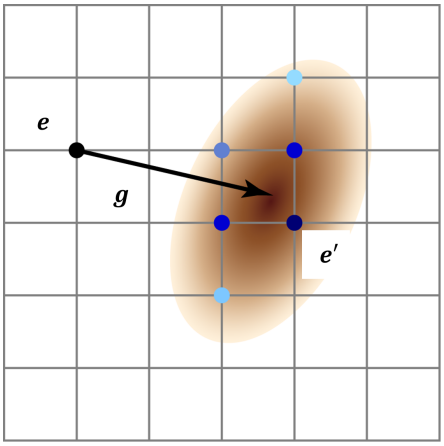

Figure 2: To create transition probabilities in the discretized MDP, for each discrete state $\mathbf{e}$ and control $\mathbf{u}$, we choose the next grid points $\mathbf{e}'$ around the mean $g(t, \mathbf{e}, \mathbf{u}, 0)$, evaluate the likelihood of $\mathcal{N}(\mathbf{0}, \text{diag}(\boldsymbol{\sigma})^2)$ at the chosen grid points, and normalize so that the outgoing transition probabilities sum to 1.

where $\mathfrak{q}(\mathbf{e})$ is a suitably chosen terminal cost. The receding-horizon CEC problem is now a non-linear program (NLP) of the form:

$$\begin{aligned} \min_{\mathbf{U}} \; & c(\mathbf{U}, \mathbf{E}) \\ \text{s.t.} \; & \mathbf{U}_{lb} \leq \mathbf{U} \leq \mathbf{U}_{ub} \\ & \mathbf{h}_{lb} \leq \mathbf{h}(\mathbf{U}, \mathbf{E}) \leq \mathbf{h}_{ub}, \end{aligned} \tag{5}$$

where $\mathbf{U} := [\mathbf{u}_\tau^\top, \ldots, \mathbf{u}_{\tau+T-1}^\top]^\top$ and $\mathbf{E} := [\mathbf{e}_\tau^\top, \ldots, \mathbf{e}_{\tau+T}^\top]^\top$. An NLP program can be solved by an NLP solver, such as CasADi. Details about CasADi and examples can be found at https://web.casadi.org/. CasADi can be installed via:

```
pip install casadi
```

but other options are available on the website. Once a control sequence $\mathbf{u}_\tau, \ldots, \mathbf{u}_{\tau+T-1}$ is obtained, CEC applies the first control $\mathbf{u}_\tau$ to the system, obtains the new error state $\mathbf{e}_{\tau+1}$ at time $\tau + 1$, and repeats the optimization in (4) online to determine the control input $\mathbf{u}_{\tau+1}$. This online re-planning is necessary because the CEC formulation does not take the effect of the motion noise into account.

Part 2: In this part, we will use the GPI algorithm to solve (3) directly but we need to discretize the state and control spaces since they are continuous. We can discretize the state and control spaces into $(n_t, n_x, n_y, n_\theta)$ and (n_v, n_ω) number of grid points, respectively. Be careful that $\tilde{\theta}$ is an angle and wrap-around should be enforced, while $n_t = 100$ since the provided reference trajectory is periodic with period of 100. The position error $\tilde{\mathbf{p}}$ and control input $\mathbf{u}$ may be discretized over the regions $[-3, 3]^2$ and $\mathcal{U}$. To create transition probabilities in the discretized MDP, for each discrete state $\mathbf{e}$ and each discrete control $\mathbf{u}$, we can choose the next grid points $\mathbf{e}'$ around the mean $g(t, \mathbf{e}, \mathbf{u}, 0)$, evaluate the likelihood of $\mathcal{N}(\mathbf{0}, \text{diag}(\boldsymbol{\sigma})^2)$ at the chosen grid points, and normalize so that the outgoing transition probabilities sum to 1. See Fig. 2 for an example. You should also be careful to account for the safety constraint $\tilde{\mathbf{p}}_t + \mathbf{r}_t \in \mathcal{F}$, either by not allowing transitions to states that may collide or by introducing an additional stage-cost term to penalize collisions. Generally, denser discretization produces more accurate solutions in the original continuous-space problem but this might lead to a very large state/action space for the GPI algorithm. Since we use error state in (3), it is easy to implement an adaptive discretization of $\mathbf{e}$ with a finer grid closer to the reference trajectory and a coarser grid further away. Similar idea can be also applied to the control space such that the robot can take slower actions when getting closer to the reference trajectory or the obstacles. Also pay attention to the compatibility of the state space grids and the control space grids. For example, dense discretization for the control space is redundant

when using sparse discretization for the state space since small control will not lead to transitions in the state space grids.

To speed up the computation, some results could be pre-computed and stored for downstream use. For example, the transition probabilities $p_f(\mathbf{e}'|\mathbf{e},\mathbf{u})$ can be pre-computed and stored in a matrix of size $(n_t, n_x, n_y, n_\theta, n_v, n_\omega, 8, 4)$, when 8 neighbors are considered. The same technique can also be used for the stage cost $\ell(\mathbf{e},\mathbf{u})$ and so on.

Parallel computing can also be used. Some Python packages like `ray` provide easy-to-use parallel computing:

```
import ray
import numpy as np

@ray.remote  # you may specify the number of CPU cores to use
def worker(start, end):
  ...

def parallel_computing():
  ray.init()
  jobs = [worker.remote(start, end) for start, end in ...]
  results = []
  while len(jobs) > 0:
    done_id, jobs = ray.wait(jobs, num_returns=1)
    results.append(ray.get(done_id[0]))
  ray.shutdown()
```

Part 3: (Optional) Alternatively, you may consider representing the value function with a linear combination of features, $V^*(t,\mathbf{e}) \approx \hat{V}(t,\mathbf{e};\boldsymbol{\theta}) := \boldsymbol{\theta}^\top \boldsymbol{\phi}(t,\mathbf{e})$, as discussed in Lecture 13[3]. This is an extension of the direct discretization of $(t,\mathbf{e}) \in \mathbb{N}\times\mathbb{R}^2\times[-\pi,\pi)$ into a finite set $(t,\mathbf{e}) \in \{t_i,\mathbf{e}_i\}_{i=1}^{n_t n_x n_y n_\theta}$ described above. Instead of a discrete value function approximation, we can use a kernel function, such as the radial basis function $k(t,\mathbf{e},t_i,\mathbf{e}_i) = \alpha\exp(-\beta_t(t-t_i)^2 - \beta_{\mathbf{e}}\|\mathbf{e}-\mathbf{e}_i\|_2^2)$ with parameters $\alpha,\beta_t,\beta_{\mathbf{e}}$, to construct a feature:

$$\boldsymbol{\phi}(t,\mathbf{e}) = [k(t,\mathbf{e},t_1,\mathbf{e}_1),\ldots,k(t,\mathbf{e},t_{n_t n_x n_y n_\theta},\mathbf{e}_{n_t n_x n_y n_\theta})]^\top \tag{6}$$

The dimension of $\boldsymbol{\theta}$ can be reduced compared to the naive discretization by using a sparser grid for $\mathbf{e}$, e.g., $\mathbf{e}_1,\mathbf{e}_3,\ldots,\mathbf{e}_{2k+1}$ with $k = \lfloor\frac{n_x n_y n_\theta - 1}{2}\rfloor$ when downsampling with a stride of 2. Using a feature-based implementation of GPI will perform better but this part is optional. In this part, you may need GPU to accelerate the computation of the weights $\boldsymbol{\phi}(t,\mathbf{e})$ and the gradients $\delta\boldsymbol{\theta}$. You may visit https://datahub.ucsd.edu to get access to GPU resources. The interface of Datahub by UCSD is similar to Jupyter Notebook and you can install packages like `pytorch` or `tensorflow` to use GPU.

Project Report and Implementation

Write a project report describing your approach to the trajectory tracking problem and provide your implementation of the three parts above.

1. [5 pts] **Introduction:** Present a brief overview of the trajectory tracking problem and the techniques used in this project.
2. [5 pts] **Problem Statement:** State the problem you are trying to solve in mathematical terms. The project description already defines the variables, constraints, and objective in (1) - (3), and it is acceptable to restate this formulation.

[3] https://natanaso.github.io/ece276b/ref/ECE276B_13_ValueFunctionApproximation.pdf

3. [30 pts] **Technical Approach:** Describe your approach in two sections: (a) dedicated to the receding-horizon CEC control and (b) dedicated to the GPI algorithm. Describe the NLP formulation in part (a), including the optimization variables, constraints, and objective function. Describe the state discretization, collision avoidance handling, and how you managed to scale the GPI algorithm to a large state space in part (b). If you implemented the optional part 3, you may add a third section describing the feature-based value function approximation or you may describe the details in part (b).

4. [30 pts] **Results:** Show qualitative and quantitative results for both the CEC and GPI trajectory tracking techniques and compare their performance in terms of computational complexity, collision avoidance, and tracking error. You may discuss the effect of ignoring the noise in the CEC formulation and of the discretization in the GPI formulation. Discuss also the performance and any interesting details you observe about the two techniques.

5. [30 pts] **Code:** Upload your code to Gradescope. It will be evaluated based on correctness and efficiency. Correctness refers to whether your code implements the algorithms you present in your report and whether the algorithms you present actually solve the trajectory tracking problem. Efficiency refers to your code being able to execute and produce control inputs for the vehicle in reasonable time. The results presented in the report should be consistent with the output of your code. The code should have a clear structure and comments. **Copying, rephrasing, or even looking at anyone else's code is strictly forbidden. Writing your own code is the best way to avoid plagiarism.**