Stanford **Spring 2026**

AA 203: Optimal and Learning-based Control
Homework #3
Due Friday May 22nd by 5:00pm

Problem 1: To solve a stochastic optimization problem with value iteration by formulating it as an MDP.

Problem 2: Gain familiarity with tools for HJ reachability and develop an understanding of sub-level sets in the context of backward reachability.

Problem 3: Understand the basics of feasibility in MPC.

Problem 4: Introduce algorithmic details of designing terminal ingredients for MPC.

3.1 Markovian drone. In this problem, we will apply techniques for solving a Markov Decision Process (MDP) to guide a flying drone to its destination through a storm. The world is represented as an $n \times n$ grid, i.e., the state space is

$$\mathcal{S} \coloneqq \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1, x_2 \in \{0, 1 \ldots, n-1\}\}.$$

In these coordinates, $(0, 0)$ represents the bottom left corner of the map and $(n-1, n-1)$ represents the top right corner of the map. From any location $x = (x_1, x_2) \in \mathcal{S}$, the drone has four possible directions it can move in, i.e.,

$$\mathcal{A} \coloneqq \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}.$$

The corresponding state changes for each action are:

- `up`: $(x_1, x_2) \mapsto (x_1, x_2 + 1)$
- `down`: $(x_1, x_2) \mapsto (x_1, x_2 - 1)$
- `left`: $(x_1, x_2) \mapsto (x_1 - 1, x_2)$
- `right`: $(x_1, x_2) \mapsto (x_1 + 1, x_2)$

Additionally, there is a storm centered at $x_{\text{eye}} \in \mathcal{S}$. The storm's influence is strongest at its center and decays farther from the center according to the equation $\omega(x) = \exp\left(-\frac{\|x - x_{\text{eye}}\|_2^2}{2\sigma^2}\right)$. Given its current state x and action a, the drone's next state is determined as follows:

- With probability $\omega(x)$, the storm will cause the drone to move in a uniformly random direction.
- With probability $1 - \omega(x)$, the drone will move in the direction specified by the action.
- If the resulting movement would cause the drone to leave $\mathcal{S}$, then it will not move at all. For example, if the drone is on the right boundary of the map, then moving right will do nothing.

The quadrotor's objective is to reach $x_{\text{goal}} \in \mathcal{S}$, so the reward function is the indicator function $R(x) = I_{x_{\text{goal}}}(x)$. In other words, the drone will receive a reward of 1 if it reaches the $x_{\text{goal}} \in \mathcal{S}$, and a reward of 0 otherwise. The reward of a trajectory in this infinite horizon problem is a discounted sum of the rewards earned in each timestep, with discount factor $\gamma \in (0, 1)$.

(a) Given $n = 20$, $\sigma = 10$, $\gamma = 0.95$, $x_{\text{eye}} = (15, 15)$, and $x_{\text{goal}} = (19, 9)$, write code that uses value iteration to find the optimal value function for the drone to navigate the storm. Recall that value iteration repeats the Bellman update

$$V(x) \leftarrow \max_{a \in \mathcal{A}} \left(\sum_{x' \in \mathcal{S}} p(x'; x, a)(R(x') + \gamma V(x')) \right)$$

until convergence, where $p(x'; x, a)$ is the probability distribution of the next state being x' after taking action a in state x, and R is the reward function. Plot a heatmap of the optimal value function obtained by value iteration over the grid $\mathcal{S}$, with $x = (0, 0)$ in the bottom left corner, $x = (n - 1, n - 1)$ in the top right corner, the x_1-axis along the bottom edge, and the x_2-axis along the left edge.

(b) Recall that a policy π is a mapping $\pi : \mathcal{S} \to \mathcal{A}$ where $\pi(x)$ specifies the action to be taken should the drone find itself in state x. An optimal value function V^* induces an optimal policy π^* such that

$$\pi^*(x) \in \arg\max_{a \in \mathcal{A}} \left(\sum_{x' \in \mathcal{S}} p(x'; x, a)(R(x') + \gamma V^*(x')) \right)$$

Use the value function you computed in part (a) to compute an optimal policy. Then, use this policy to simulate the MDP for $N = 100$ time steps with the state initialized at $x = (0, 19)$. Plot the policy as a heatmap where the actions $\{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$ correspond to the values $\{0, 1, 2, 3\}$, respectively. Plot the simulated drone trajectory overlaid on the policy heatmap, and briefly describe in words what the policy is doing.

3.2 Reach-avoid flight. Consider the goal of developing a self-righting quadrotor, i.e., a flying drone that you can throw into the air at various poses and velocities which will autonomously regulate itself to level flight while obeying dynamics, control, and operational-envelope constraints. For this problem, we consider the 6-D dynamics of a planar quadrotor described by

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ \frac{-(T_1+T_2)\sin\phi - C_D^v v_x}{m} \\ v_y \\ \frac{(T_1+T_2)\cos\phi - C_D^v v_y}{m} - g \\ \omega \\ \frac{(T_2-T_1)\ell - C_D^\phi \omega}{I_{yy}} \end{bmatrix}, \quad T_1, T_2 \in [0, T_{\max}], \tag{1}$$

where the state is given by the position in the vertical plane (x, y), translational velocity (v_x, v_y), pitch ϕ, and pitch rate ω; the controls are the thrusts (T_1, T_2) for the left and right prop respectively. Additional constants appearing in the dynamics above are gravitational acceleration g, the quadrotor's mass m, moment of inertia (about the out-of-plane axis) I_{yy}, half-length ℓ, and translational and rotational drag coefficients C_D^v and C_D^ϕ, respectively (see `starter_hj_reachability.py` for precise values of these constants in `PlanarQuadrotor.__init__`).

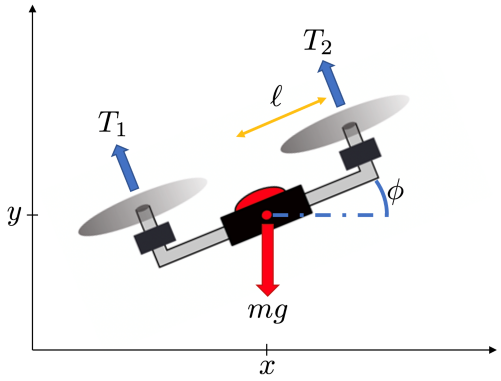

Figure 1: A planar quadrotor.

We will approach the problem of self-righting through continuous-time dynamic programming, specifically a Hamilton-Jacobi-Bellman (HJB) formulation.[1] To help mitigate the curse of dimensionality, we ignore the lateral motion (irrelevant to achieving level flight) and consider reduced 4-D dynamics with state vector $\mathbf{x} := (y, v_y, \phi, \omega) \in \mathbb{R}^4$. For these reduced dynamics, we define the target set

$$\mathcal{T} = [3, 7] \times [-1, 1] \times [-\pi/12, \pi/12] \times [-1, 1] \subset \mathbb{R}^4.$$

We assume that once the planar quadrotor reaches this set, we have another controller (e.g., an LQR controller linearized around hover) that can take over to maintain level flight.

To bound the domain of our dynamic programming problem (and also to ensure that our quadrotor doesn't plow into the ground), in addition to the dynamics and control constraints given in (1) we would also like to constrain our planar quadrotor to stay within the operational envelope

$$\mathcal{E} = [1, 9] \times [-6, 6] \times [-\infty, \infty] \times [-8, 8].$$

[1] One might also consider an HJI-based extension to handle worst-case disturbances (e.g., wind), but for simplicity in this exercise we just consider the undisturbed dynamics.

Reaching the target set $\mathcal{T}$ while *avoiding* the obstacle set $\mathcal{E}^\complement$ (i.e., the set complement of $\mathcal{E}$) is referred to as a *reach-avoid* problem. If we can construct two real-valued, Lipschitz continuous functions $h(\mathbf{x}), e(\mathbf{x})$ defined over the state domain such that

$$\mathbf{x} \in \mathcal{T} \iff h(\mathbf{x}) \leq 0, \quad \mathbf{x} \in \mathcal{E} \iff e(\mathbf{x}) \leq 0,$$

i.e., $\mathcal{T}, \mathcal{E}$ are the zero-sublevel sets of h, e respectively, then it may be shown (see, e.g., (FCTS15, Theorem 1)) that the value function $V(\mathbf{x}, t)$ defined as

$$\begin{aligned} V(\mathbf{x}_0, t_0) = \min_{\mathbf{u}(\cdot)} \min_{\tau \in [t_0, 0]} & h(\mathbf{x}(\tau)) \\ \text{s.t. } & \dot{\mathbf{x}}(\tau) = f(\mathbf{x}(\tau), \mathbf{u}(\tau)) \quad \forall \tau \in [t_0, 0] \\ & \mathbf{x}(\tau) \in \mathcal{E} \quad \forall \tau \in [t_0, 0] \\ & \mathbf{x}(t_0) = \mathbf{x}_0 \end{aligned}$$

(where f is the relevant portion of the full dynamics (1)) satisfies the HJB PDE[2]

$$\begin{aligned} \max \left\{ \frac{\partial V}{\partial t}(\mathbf{x}, t) + \min\{0, H(\mathbf{x}, \nabla_{\mathbf{x}} V(\mathbf{x}, t))\}, e(\mathbf{x}) - V(\mathbf{x}, t) \right\} = 0 \\ \text{where } H(\mathbf{x}, \mathbf{p}) = \min_{\mathbf{u}} \mathbf{p}^T f(\mathbf{x}, \mathbf{u}), \\ V(\mathbf{x}, 0) = \max\{h(\mathbf{x}), e(\mathbf{x})\}. \end{aligned}$$

Implementing an appropriate solver for this type of PDE is somewhat nontrivial (see, e.g., (Mit02) for details); for this exercise we will use an existing solver – you will be responsible for setting the problem up and interpreting the results.

If you are running your code locally on your own machine, install the solver at the command line using `pip` via the command:

```
pip install --upgrade hj-reachability
```

Otherwise, if you are using Google Colab, run a cell containing:

```
!pip install --upgrade hj-reachability
```

For this problem, you will fill parts of `starter_hj_reachability.py` in with your own code. When submitting code, only provide the methods or functions that you have been asked to modify.

(a) Subject to the control constraints $T_1, T_2 \in [0, T_{\max}]$, derive the locally optimal action that minimizes the Hamiltonian, i.e., for arbitrary $\mathbf{x}, \mathbf{p}$ compute

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \mathbf{p}^T f(\mathbf{x}, \mathbf{u}),$$

where f denotes the last four rows of the dynamics defined by (1). Use this knowledge to implement the method `PlanarQuadrotor.optimal_control`.

[2] This is similar to the backward reachable tube HJI PDE mentioned in class (omitting the disturbance), where as before the inner min ensures the value function is nondecreasing in time (so that as BRT computation proceeds backward in time, the value function is nonincreasing at successive iterations, i.e., you get to "lock in" the lowest value you ever achieve). The outer max is the new addition in this formulation compared to what we saw in class, and may be interpreted as always making sure $V(\mathbf{x}, t) \geq e(\mathbf{x})$ so that if $e(\mathbf{x}) > 0$ (i.e., the state is outside of the operating envelope) then also $V(\mathbf{x}, t) > 0$ (i.e., the state is outside the BRT of states that can reach the target collision-free).

(b) Write down a functional form for $h(\mathbf{x})$ such that $\mathbf{x} \in \mathcal{T} \iff h(\mathbf{x}) \leq 0$. Implement the function `target_set`.

Hint: Note that $a(\mathbf{x}) \leq 0 \ \wedge \ b(\mathbf{x}) \leq 0 \iff \max\{a(\mathbf{x}), b(\mathbf{x})\} \leq 0$. This means that if you have multiple constraints represented as the zero-sublevel sets of multiple functions, then the conjunction of the constraints may be represented as a pointwise maximum of the functions.

(c) Write down a functional form for $e(\mathbf{x})$ such that $\mathbf{x} \in \mathcal{E} \iff e(\mathbf{x}) \leq 0$. Implement the function `envelope_set`.

(d) Run the rest of the script/cells to compute $V(\mathbf{x}, -5)$ and take a look at some of the controlled trajectories; hopefully they look reasonable (see the note below if you're picky/have extra time, though if the quad rights itself and gets to the target set $\mathcal{T}$ that's sufficient for our purposes). Do not submit any trajectory plots; instead include a 3D plot of the zero isosurface (equivalent of a contour/isoline, but in 3D) for a slice of the value function at some fixed y value (e.g., $y = 7.5$ as pre-selected in the starter code). Explain why one of the bumps/ridges (e.g., as highlighted by the red or blue arrow in Figure 2, which you may also use to check your work) has the shape that it does.

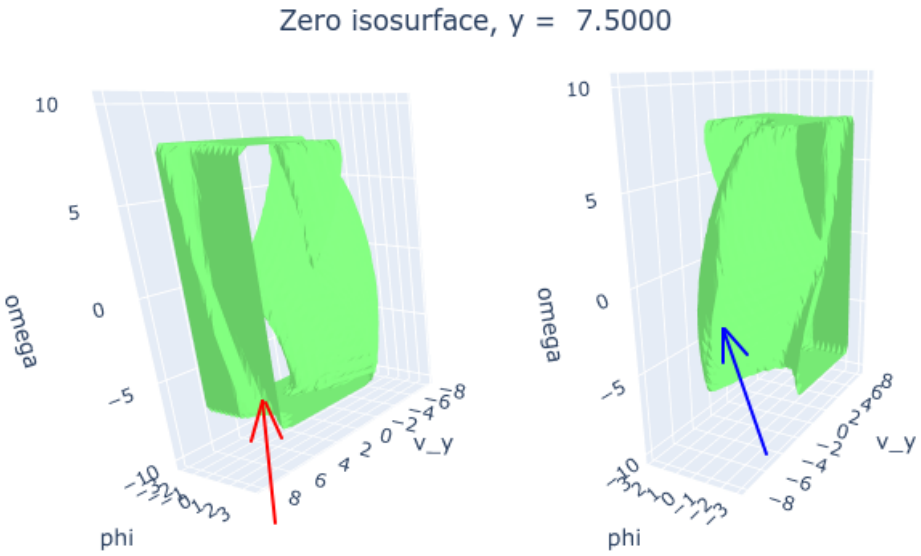

Figure 2: Example zero isosurface views. Can you explain why the red valley (outside the isosurface, i.e., unrecoverable initial conditions) or blue ridge (inside the isosurface, i.e., initial conditions that can reach the target collision-free) exhibit the "tilt" they have by considering the corresponding states?

Note: If the behavior of your control policy isn't as nice as you'd like (e.g., height/pitch oscillations), consider modifying your target set function $h(\mathbf{x})$ (e.g., by scaling how you account for each dimension in your construction). For the purpose of reachable set computation, at least theoretically[3] the zero-sublevel set of the value function V (corresponding to the set of feasible initial states) is unaffected by the details of h as long as $h(\mathbf{x}) \leq 0 \iff \mathbf{x} \in \mathcal{T}$. In the context

[3]With a relatively coarse grid discretization and not-particularly-high-accuracy finite difference schemes/time integrators for PDE solving (sacrifices made so you don't have to wait for hours to see results), for numerical reasons the BRT may have some dependence on your formulation of $h(\mathbf{x})$.

of dynamic programming to compute an optimal control policy, however, $h(\mathbf{x})$ also defines the terminal cost in a way that materially affects the policy once the set is reached (though in practice, this is where we'd have some other stabilizing controller take over).

(e) In a few sentences, write down some pros/cons of this approach (i.e., computing a policy using dynamic programming) for a self-righting quadrotor vs. alternatives, e.g., applying model-predictive control. Potential things to think/write about: computational resources (time, memory) required for online operation, local/global optimality, flexibility to accommodate additional obstacles in the environment, bang-bang controls, etc.

3.3 MPC feasibility. Consider the discrete-time LTI system

$$x_{t+1} = Ax_t + Bu_t.$$

We want to compute a receding horizon controller for a quadratic cost function, i.e.,

$$J(x,u) = x_T^\mathsf{T} P x_T + \sum_{t=0}^{T-1} \Big(x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t \Big),$$

where $P, Q, R \succ 0$ are weight matrices. We must satisfy the state and input constraints $\|x_t\|_\infty \leq r_x$ and $\|u_t\|_\infty \leq r_u$, respectively. Also, we will enforce the terminal state constraint $\|x_T\|_\infty \leq r_T$, where we will tune $r_T \geq 0$. For $r_T = 0$, the terminal state constraint is equivalent to $x_T = 0$, while for $r_T \geq r_x$ we are just left with the original state constraint $\|x_T\|_\infty \leq r_x$.

For this problem, you will work with the starter code in `mpc_feasibility.py`. Carefully review *all* of the code in this file before you continue. Only submit code you add and any plots that are generated by the file.

(a) Implement a receding horizon controller for this system using CVXPY in the function `do_mpc`. Run the remaining code to simulate closed-loop trajectories with $r_T \geq r_x$ from two different initial states, each with either $P = I$ or P as the unique positive-definite solution to the discrete algebraic Riccati equation (DARE)

$$A^\mathsf{T} P A - P - A^\mathsf{T} P B (R + B^\mathsf{T} P B)^{-1} B^\mathsf{T} P A + Q = 0.$$

Submit your code and the plot that is generated, which displays both the realized closed-loop trajectories and the predicted open-loop trajectories at each time step. Discuss your observations of any differences between the trajectories for the different initial conditions and values of P.

(b) Finish the function `compute_roa`, which computes the region of attraction (ROA) for fixed $P \succ 0$ and different values of N and r_T. Submit your code and the plot of the different ROAs. Compare and discuss your observations of the ROAs.

Hint: While debugging your code, you can set a small `grid_dim` to reduce the amount of time it takes to compute the ROAs. However, you must submit your plot of the ROAs with at least `grid_dim = 30`.

3.4 Terminal ingredients. Consider the discrete-time LTI system $x_{t+1} = Ax_t + Bu_t$ with

$$A = \begin{bmatrix} 0.9 & 0.6 \\ 0 & 0.8 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

We want to synthesize a model predictive controller to regulate the system to the origin while minimizing the quadratic cost function

$$J(x, u) = x_T^\mathsf{T} P x_T + \sum_{t=0}^{T-1} \Big(x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t \Big),$$

with $Q \succ 0$, $R \succ 0$, and $P \succ 0$, subject to $\|x_t\|_2 \leq r_x$, $\|u_t\|_2 \leq r_u$, and $x_T \in \mathcal{X}_T$. For this problem, set $N = 4$, $r_x = 5$, $r_u = 1$, $Q = I$ and $R = I$.

Recall from lecture that the terminal ingredients $\mathcal{X}_T$ and P are critical to recursive feasibility and stability of the resulting closed-loop system under receding horizon control.

(a) For this particular problem, explain why and how we can design $\mathcal{X}_T$ and P in an open-loop manner, i.e., by only considering the uncontrolled system $x_{t+1} = Ax_t$. You only need to describe what properties of $\mathcal{X}_T$ and P your method must ensure to guarantee recursive feasibility and stability of the resulting closed-loop system with MPC feedback.

For the remainder of this problem, set $P = I$ for simplicity.

We want to find as large of a positive invariant set $\mathcal{X}_T$ for $x_{t+1} = Ax_t$ as possible that satisfies the state constraints. While maximal positive invariant sets may be computed via iterative methods using tools from computational geometry[4], we restrict our search to ellipsoids of the form

$$\mathcal{X}_T = \{x \in \mathbb{R}^n \mid x^\mathsf{T} W x \leq 1\}$$

with $W \succ 0$. Since $\text{vol}(\mathcal{X}_T) \sim \sqrt{\det(W^{-1})}$, we can formulate our search for the largest ellipsoidal $\mathcal{X}_T$ as the semi-definite program (SDP)

$$\begin{aligned} \underset{W \succ 0}{\text{maximize}} \quad & \log\det(W^{-1}) \\ \text{subject to} \quad & A^\mathsf{T} W A - W \preceq 0 \cdot \\ & I - r_x^2 W \preceq 0 \end{aligned}$$

Critically, each constraint in a convex or concave SDP is a *linear matrix inequality (LMI)*.

(b) Prove that $A^\mathsf{T} W A - W \preceq 0$ and $I - r_x^2 W \preceq 0$ together are sufficient conditions for $\mathcal{X}_T$ to be a positive invariant set satisfying the state constraints.

(c) For a maximization problem, we want the objective to be a concave function. Unfortunately, the given SDP is not concave since $\log\det(W^{-1}) = -\log\det(W)$ is convex with respect to its argument $W \succ 0$. Reformulate the given SDP in W as a concave SDP in $M \coloneqq W^{-1}$.

Hint: You should use

- the fact that $B \preceq C$ if and only if $ABA \preceq ACA$ for symmetric A, B, and C where $A \succ 0$,
- the fact that $A \preceq \gamma I$ if and only if $A^{-1} \succeq \frac{1}{\gamma} I$ for $A \succ 0$ and $\gamma > 0$, and

[4] See the MPT3 library (https://www.mpt3.org/) for some examples tailored to model predictive control.

- Schur's complement lemma, which states that

$$C - B^\mathsf{T} A^{-1} B \succeq 0 \iff \begin{bmatrix} A & B \\ B^\mathsf{T} & C \end{bmatrix} \succeq 0$$

  for any conformable matrices A, B, and C where $A \succ 0$.

(d) Use NumPy and CVXPY to formulate and solve the SDP for M. Plot the ellipsoids $\mathcal{X}_T$, $A\mathcal{X}_T$, and $\mathcal{X} := \{x \mid \|x\|_2^2 \leq r_x^2\}$ in the same figure. You should see that $A\mathcal{X}_T \subseteq \mathcal{X}_T \subseteq \mathcal{X}$. Submit your code and plot, and report $W := M^{-1}$ with three decimal places for each entry.

Hint: Consult the CVXPY documentation to help you write your code. Specifically, look at the list of functions in CVXPY (https://www.cvxpy.org/tutorial/functions/index.html) and the SDP example (https://www.cvxpy.org/examples/basic/sdp.html). You can write any definite constraints in the SDP with analogous semi-definite constraints (i.e., treat "$\succ$" as "`>>`" for the purposes of writing your CVXPY code).

For plotting purposes, you can use the following Python function to generate points on the boundary of a two-dimensional ellipsoid.

```python
import numpy as np

def generate_ellipsoid_points(M, num_points=100):
    """Generate points on a 2-D ellipsoid.

    The ellipsoid is described by the equation
        `{ x | x.T @ inv(M) @ x <= 1 }`,
    where `inv(M)` denotes the inverse of the matrix argument `M`.

    The returned array has shape (num_points, 2).
    """
    L = np.linalg.cholesky(M)
    θ = np.linspace(0, 2*np.pi, num_points)
    u = np.column_stack([np.cos(θ), np.sin(θ)])
    x = u @ L.T
    return x
```

(e) Use NumPy and CVXPY to setup the MPC problem, then simulate the system with closed-loop MPC from $x_0 = (0, -4.5)$ for 15 time steps. Overlay the actual state trajectory and the planned trajectories at each time on the plot from part (d). Also, separately plot the actual control trajectory over time in a second plot. Overall, you should have two plots for this entire question. Submit both plots and all of your code.

Hint: Instead of forming the MPC problem in CVXPY during each simulation iteration, form a single CVXPY problem parameterized by the initial state and replace its value before solving (https://www.cvxpy.org/tutorial/intro/index.html#parameters).

References

[FCTS15] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, *Reach-avoid problems with time-varying dynamics, targets and constraints*, Hybrid Systems: Computation and Control, 2015.

[Mit02] I. M. Mitchell, *Application of level set methods to control and reachability problems in continuous and hybrid systems*, Ph.D. thesis, Stanford University, 2002.