Project 3: Infinite-Horizon Stochastic Optimal Control

Behrad Rabiei
Dept. of Electrical and Computer Engineering
University of California San Diego
Email: brabiei@ucsd.edu

***Abstract*—This paper is the report for Project 3 of ECE 276B (Planning and Learning in Robotics). The main objective of the project is to implement algorithms that will derive optimal control policies for a differential drive vehicle to trace a given trajectory. The two algorithms that we explored in this project are receding-horizon Certainty Equivalent Control (CEC) and Generalized Policy Iteration (GPI).**

I. Introduction

In the rapidly evolving field of robotics, the ability to ensure safe and accurate trajectory tracking is extremely important, especially for ground differential-drive robots widely used in various applications. These robots, characterized by their two independently driven wheels, require precise movement control to navigate complex environments effectively. This capability is not only critical in enhancing the operational efficiency of autonomous systems but also crucial in ensuring safety in interactions with humans and the surrounding infrastructure. From autonomous delivery services navigating busy city streets to robots performing tasks in hazardous or inaccessible areas, the importance of reliable trajectory tracking should not be under estimated. This project explores advanced algorithms and control systems to achieve safe trajectory tracking, aiming to significantly improve the adaptability and safety of differential-drive robots.

A. Receding-Horizon Certainty Equivalent Control (CEC)

The Receding-Horizon Certainty Equivalent Control (CEC) is a sophisticated algorithm that plays a crucial role in trajectory tracking for differential-drive robots. At its core, CEC operates by continually updating its control decisions based on a predictive model over a finite future horizon. This "receding horizon" approach involves optimizing control actions at each step based on current state estimates, and then shifting the window forward as time progresses. In practice, the CEC algorithm treats the future states and inputs as known and fixed (hence, "certainty equivalent"), ignoring uncertainty beyond the immediate planning horizon. This simplifies the computational complexity, making the algorithm more practical for real-time applications. It recalculates the optimal path and control inputs at each step, based on the latest available data, which allows it to adapt dynamically to changes in the environment or robot state.Receding-Horizon CEC is widely utilized in fields where precise control and adaptability are necessary, such as in autonomous vehicles, unmanned aerial vehicles (UAVs), and robotic manipulators in industrial automation. These applications benefit from the algorithm's ability to handle complex, dynamic environments while maintaining a high level of performance and safety.

B. General Policy Iteration

Generalized Policy Iteration (GPI) is a robust framework used in the field of reinforcement learning, which is well-suited for complex decision-making tasks such as trajectory tracking in robotics. GPI iteratively improves both the policy (the robot's strategy for action selection) and the value function (an estimate of the expected long-term return from each state under the current policy). This iterative process consists of two main phases: policy evaluation, where the value function is updated to accurately reflect the returns obtained by following the current policy, and policy improvement, where the policy is refined to yield higher returns based on the updated value function. The strength of GPI lies in its adaptability and convergence properties. By continually updating the policy and the corresponding value function, GPI can adapt to a wide range of environments and tasks. It systematically explores and exploits the solution space, ensuring that the robot's trajectory becomes increasingly optimal with each iteration. This makes GPI particularly effective for tasks where the environment may change or where the robot must learn from interactions with the environment without explicit programming for every possible scenario. In robotics, GPI is employed in navigation and obstacle avoidance, among other tasks, enabling robots to learn from their experiences and improve their performance over time autonomously. This capability is crucial for developing robots that operate effectively in dynamic, unpredictable settings such as urban landscapes, disaster sites, and varied industrial environments.

II. Problem Formulation

This project considers the design of a safe trajectory tracking control system for a ground differential-drive robot. Define the state of the robot at any discrete time step $t \in \mathbb{N}$ as $x_t := (p_t, \theta_t)$, where $p_t \in \mathbb{R}^2$ represents the position and $\theta_t \in [-\pi, \pi)$ represents the orientation. The control inputs to the robot are its linear velocity $v_t \in \mathbb{R}$ and angular velocity (yaw rate) $\omega_t \in \mathbb{R}$, represented together as $u_t := (v_t, \omega_t)$.

The kinematic model of the robot, derived via Euler discretization with a timestep $\Delta > 0$, is formulated as:

$$x_{t+1} = \begin{bmatrix} p_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} p_t \\ \theta_t \end{bmatrix} + \Delta \begin{bmatrix} \cos(\theta_t) & 0 \\ \sin(\theta_t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + w_t, \quad (1)$$

where $w_t \in \mathbb{R}^3$ represents the motion noise with Gaussian distribution $N(0, \text{diag}(\sigma)^2)$ and $\sigma = [0.04, 0.04, 0.004] \in \mathbb{R}^3$. The motion noise is assumed independent of the robot's state and across different times.

The control inputs are bounded within the set $U := [0, 1] \times [-1, 1]$. The primary objective is to design a control policy for the appropriate robot to follow a desired trajectory consisting of a reference position trajectory $r_t \in \mathbb{R}^2$ and orientation trajectory $\alpha_t \in [-\pi, \pi)$, while avoiding obstacles and collisions. The environment includes two circular obstacles: C_1 centered at $(-2, -2)$ with radius 0.5, and C_2 centered at $(1, 2)$ with radius 0.5. The free space F in the environment is thus defined as:

$$F := [-3, 3]^2 \setminus (C_1 \cup C_2).$$

To measure deviation from the reference trajectory, we define the error state $e_t := (\tilde{p}_t, \tilde{\theta}_t)$ where $\tilde{p}_t := p_t - r_t$ and $\tilde{\theta}_t := \theta_t - \alpha_t$. The error dynamics are given by:

$$e_{t+1} = \begin{bmatrix} \tilde{p}_{t+1} \\ \tilde{\theta}_{t+1} \end{bmatrix} = \begin{bmatrix} \tilde{p}_t \\ \tilde{\theta}_t \end{bmatrix} + \Delta \begin{bmatrix} \cos(\tilde{\theta}_t + \alpha_t) & 0 \\ \sin(\tilde{\theta}_t + \alpha_t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \begin{bmatrix} r_t - r_{t+1} \\ \alpha_t - \alpha_{t+1} \end{bmatrix} + w_t, \quad (2)$$

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

$$V^*(\tau, e) = \min_{\pi} \mathbb{E}\Bigg[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} \big(\tilde{p}_t^\top Q \tilde{p}_t + q(1 - \cos(\tilde{\theta}_t))^2 + u_t^\top R u_t \big) \mid e_\tau = e \Bigg] \quad (3)$$

subject to:

$$\begin{aligned} e_{t+1} &= g(t, e_t, u_t, w_t), \quad w_t \sim N(0, \text{diag}(\sigma)^2) \\ t &= \tau, \tau + 1, \ldots \\ u_t &= \pi(t, e_t) \in U, \\ \tilde{p}_t + r_t &\in F, \end{aligned}$$

where $Q \in \mathbb{R}^{2\times2}$ is a symmetric positive-definite matrix defining the stage cost for deviating from the reference position trajectory r_t, $q > 0$ is a scalar defining the stage cost for deviating from the reference orientation trajectory α_t, and $R \in \mathbb{R}^{2\times2}$ is a symmetric positive-definite matrix defining the stage cost for using excessive control effort.

A. Certainty Equivalent Control (CEC)

Certainty Equivalent Control (CEC) represents a suboptimal control scheme that applies, at each control stage, the control input that would be optimal if the noise variables w_t were assumed to be fixed at their expected values (zero in this case). The primary advantage of CEC is its simplification of a stochastic optimal control problem to a deterministic optimal control problem, which is generally more tractable.

Moreover, receding-horizon CEC approximates an infinite-horizon problem by iteratively solving the following discounted finite-horizon deterministic optimal control problem at each time step:

$$V^*(\tau, e) \approx \min_{u_\tau, \ldots, u_{\tau+T-1}} \Bigg\{ q(e_{\tau+T}) + \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} (\tilde{p}_t^\top Q \tilde{p}_t + q(1 - \cos(\tilde{\theta}_t))^2 + u_t^\top R u_t) \Bigg\} \quad (4)$$

subject to the following constraints:

$$\begin{aligned} e_{t+1} &= g(t, e_t, u_t, 0), \qquad t = \tau, \ldots, \tau + T - 1, \\ u_t &\in U, \\ \tilde{p}_t + r_t &\in F, \end{aligned}$$

where $q(e)$ is a suitably chosen terminal cost function.

The receding-horizon CEC problem thus forms a non-linear programming (NLP) problem expressed as:

$$\min_{U} c(U, E) \quad \text{s.t.} \quad U_{\text{lb}} \le U \le U_{\text{ub}}, \quad h_{\text{lb}} \le h(U, E) \le h_{\text{ub}}, \quad (5)$$

where $U := [u_\tau^\top, \ldots, u_{\tau+T-1}^\top]^\top$ and $E := [e_\tau^\top, \ldots, e_{\tau+T}^\top]^\top$.

Upon deriving a control sequence $u_\tau, \ldots, u_{\tau+T-1}$, CEC applies the first control u_τ to the system, updates the error state to $e_{\tau+1}$ at the subsequent time $\tau + 1$, and repeats the optimization process defined in Equation (4) online to determine the next control input $u_{\tau+1}$. This ongoing re-planning is essential, as the CEC approach does not inherently account for the effects of motion noise.

B. General Policy Iteration

In the second part of this project, we aim to solve the stochastic optimal control problem described in Equation (3) using the Generalized Policy Iteration (GPI) algorithm. To apply GPI, it is necessary to discretize the continuous state and control spaces. The state space is discretized into $(n_t, n_x, n_y, n_\theta)$ grid points, while the control space is represented by (n_v, n_ω) grid points. It is important to note that $\tilde{\theta}$, representing an angle, requires wrap-around handling to maintain continuity over 2π. The temporal discretization parameter n_t is set to 100, aligning with the periodicity of the provided reference trajectory.

The position error $\tilde{p}$ and the control inputs u are discretized over the regions $[-3, 3]^2$ and U, respectively. To establish the transition probabilities in the discretized Markov Decision Process (MDP), for each discrete state e and each control action u, we select the next grid points e' centered around the expected state transition $g(t, e, u, 0)$. The transition likelihoods, governed by $N(0, \mathrm{diag}(\sigma)^2)$, are evaluated at these points and normalized so that the sum of outgoing probabilities equals one.

Furthermore, the safety constraint $\tilde{p}_t + r_t \in F$ must be enforced by either restricting transitions to non-colliding states or incorporating an additional stage-cost term to penalize potential collisions. Generally, a denser discretization of the state and control spaces can yield a more accurate approximation of the continuous problem, though at the cost of an increased computational burden for the GPI algorithm. Adaptive discretization strategies are employed to refine the grid in areas closer to the reference trajectory and coarsen it further away, enhancing computational efficiency while maintaining solution fidelity.

Similar considerations apply to the control space, where finer control actions are beneficial near the reference trajectory or near obstacles. It is also crucial to maintain compatibility between the densities of the state space and control space grids; for instance, a dense control discretization is ineffectual with a sparse state space grid as minute controls would not result in discernible state transitions.

To expedite the computation, pre-computing certain values can be advantageous. Specifically, the transition probabilities $p_f(e'|e, u)$ can be precomputed and stored in a multidimensional matrix of dimensions $(n_t, n_x, n_y, n_\theta, n_v, n_\omega, 8, 4)$, considering 8 neighboring states. This approach can similarly be applied to pre-compute and store stage costs $\ell(e, u)$, facilitating faster algorithm execution.

1) GPI Details: The Generalized Policy Iteration (GPI) serves as the principal mechanism for solving stochastic infinite-horizon problems over Markov Decision Processes (MDPs) with known models. GPI operates in two main phases: Policy Evaluation and Policy Improvement.

Policy Evaluation

In the policy evaluation step, given a policy π, the value function V^π is computed for every state $x \in \mathcal{X}$. The value function at each state is defined by:

$$V^\pi(x) = \ell(x, \pi(x)) + \gamma \mathbb{E}_{x' \sim p_f(\cdot|x, \pi(x))} \left[V^\pi(x') \right], \quad (6)$$

where $\ell(x, \pi(x))$ denotes the immediate cost of taking action $\pi(x)$ in state x, and γ is the discount factor which balances the importance of immediate and future rewards. The expectation is taken over the state transitions according to the policy π.

Policy Improvement

During policy improvement, a new policy π' is derived by optimizing the expected total cost, updating the policy at every state as follows:

$$\pi'(x) \in \arg \min_{u \in U(x)} \left\{ \ell(x, u) + \gamma \mathbb{E}_{x' \sim p_f(\cdot|x, u)} \left[V^\pi(x') \right] \right\}, \quad (7)$$

for all $x \in \mathcal{X}$. This step involves determining the action u that minimizes the sum of the immediate cost and the discounted value of the next state, as expected under the current policy.This iterative process of evaluating and improving policies continues until the policy converges, resulting in an optimal policy that minimizes the long-term expected cost for all states in the MDP.

III. Technical Approach

CEC

This section describes the implementation of the Receding-Horizon Certainty Equivalent Control (CEC) strategy using CasADi for nonlinear optimization. The CEC method is employed to determine the optimal control inputs for a system given its current and reference states. The primary objective is to minimize a cost function over a finite time horizon while adhering to specified constraints.

A. Initialization

The CEC class initializes with a set time horizon T of 15 time steps and a discount factor γ of 0.95. These parameters determine the temporal scope of the optimization and the weighting of future costs, respectively. The state cost for position, $\mathbf{Q}$, is defined as a 2×2 identity matrix, and the control cost, $\mathbf{R}$, is also a 2×2 identity matrix. Additionally, the state cost for orientation, q, is set to 1. These cost matrices are crucial for formulating the optimization objective function, balancing the trade-offs between state deviations and control efforts.

B. Control Calculation

The core functionality of the CEC strategy is encapsulated in the `__call__` method, which computes the control input based on the current time step, current state, and reference state. The method begins by generating a sequence of reference states over the time horizon T using a Lissajous trajectory function. This trajectory serves as the desired path the system should follow.

Optimization variables, specifically the velocity v_t and angular velocity w_t, are defined using CasADi symbolic expressions. These variables represent the control inputs at each time step. The objective function is formulated to minimize the cost, which is a combination of state and control costs. The state cost includes the position error weighted by $\mathbf{Q}$ and the orientation error weighted by q. The control cost is weighted by $\mathbf{R}$. The objective function is given by:

$$\text{objective} = \sum_{i=1}^{T} \gamma^i \left(\mathbf{p}_t^\top \mathbf{Q} \mathbf{p}_t + q \left(1 - \cos(e_t[2])\right)^2 + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t \right)$$

where $\mathbf{p}_t$ represents the position error, $e_t[2]$ represents the orientation error, and $\mathbf{u}_t$ represents the control input vector.

C. Constraints and Optimization

In the Receding-Horizon Certainty Equivalent Control (CEC) strategy, constraints play a critical role in ensuring the feasibility and safety of the control inputs. These constraints are incorporated into the nonlinear programming problem (NLP) to restrict the optimization variables within acceptable limits and to address obstacle avoidance.

1) Bounds on Optimization Variables: The optimization variables, specifically the control inputs v_t (velocity) and w_t (angular velocity), are subject to lower and upper bounds. These bounds are defined to ensure that the control actions are physically realizable and to prevent excessive or unrealistic control efforts. The lower bounds are set to $[0.5, 0.5]$ and the upper bounds are set to $[100, 100]$ for each time step from 1 to $T - 1$. This results in a constraint formulation as follows:

$$0.5 \leq v_t, w_t \leq 100 \quad \forall t \in \{1, \ldots, T-1\}$$

2) State Constraints for Obstacle Avoidance: To incorporate obstacle avoidance, the state variables are constrained to maintain a safe distance from predefined obstacles. Two specific constraints are included to ensure the system avoids collision with obstacles at given locations. These constraints are formulated as:

$$(e_t[0]+\text{ref\_states}[i][0]+2)^2+(e_t[1]+\text{ref\_states}[i][1]+2)^2 \geq d_{\text{min}}^2$$

$$(e_t[0]+\text{ref\_states}[i][0]-1)^2+(e_t[1]+\text{ref\_states}[i][1]-2)^2 \geq d_{\text{min}}^2$$

where $e_t[0]$ and $e_t[1]$ are the position errors in the x and y directions, respectively, and ref_states$[i]$ represent the reference states at time step i. The value d_{min} denotes the minimum distance required to avoid the obstacle. These constraints are appended to the list of optimization constraints to ensure the system's trajectory remains collision-free.

3) Formulating the NLP: The nonlinear programming problem (NLP) is formulated by combining the objective function and the constraints. The CasADi solver `nlpsol` with the 'ipopt' algorithm is employed to solve the NLP. The solver is configured with specific options to control the level of output and computational performance.

The complete NLP formulation includes:

- The objective function, which aims to minimize the cumulative cost over the time horizon.
- The bounds on the optimization variables v_t and w_t.
- The state constraints for obstacle avoidance, ensuring safe navigation.

4) Solver Execution and Solution Extraction: Upon defining the NLP, the solver is executed with an initial guess (all zeros in my case) for the optimization variables and the defined bounds and constraints. The solution provided by the solver includes the optimal control inputs for each time step. These inputs are then extracted and used to control the system, ensuring it follows the reference trajectory while minimizing the cost and avoiding obstacles.

GPI

D. State Discretization

Instead of directly using world coordinates, we utilize a state space for the error between the actual position and the reference, denoted as e_t. This state space is discretized into grids to make the problem computationally feasible for the GPI algorithm. The state space X is discretized into a grid with dimensions $(N_t, N_x, N_y, N_\theta)$, where:

- N_t represents discretized time steps, set to 100 to match the reference trajectory period.
- N_x, N_y, and N_θ represent the number of discretized points in the position error and orientation error spaces, set to (21, 21, 10) respectively. (These are the values the TA used on Piazza)

The control space U is discretized into (N_v, N_w), where:

- N_v and N_w represent the number of discretized points in the control input space for linear and angular velocities, respectively set to (6, 11). (These are the values the TA used on Piazza)

To achieve this, the position error e_t and control input u are adaptively discretized over predefined regions. This is implemented using `np.linspace` and the hyperbolic tangent function by solving $y = \tanh(x)$ for y within the bounds of our space $[-3, 3]$. Using `np.linspace`, an even distribution between these points is found. Plugging these values back into the hyperbolic tangent function results in a denser discretization towards the origin, providing finer control near the reference trajectory.

Transition Matrix and Stage Cost Precomputation: Transition probabilities for the discretized Markov Decision Process (MDP) are precomputed. For each discrete state e and control u, the next grid points e' are chosen around the mean $g(t, e, u, 0)$, considering only the immediate 8 neighbors. Their likelihood of transitioning to each grid point is evaluated using a multivariate normal distribution with a mean of $g(t, e, u, 0)$ and a diagonal covariance matrix σ^2. The probabilities are normalized to ensure they sum to 1.Stage costs are similarly precomputed. Precomputing these elements is crucial as it allows for faster computations and implementation of the algorithm.

E. Collision Avoidance Handling

Collision avoidance is handled by introducing additional stage costs for grid points that fall within the collision margin of defined obstacles. Specifically: A collision margin is defined around obstacles to create a buffer. During the stage cost computation, any state within this margin incurs a higher cost, effectively discouraging the policy from selecting paths through these regions.

The stage costs are precomputed for all state-action pairs, taking into account position and orientation errors as well as control inputs. This precomputation allows for efficient lookup during the GPI iterations. The cost function includes terms for position error, orientation error, and control effort, weighted by matrices Q, R, and a scalar q for the orientation error.

F. Implementation Details of the GPI Algorithm

The implementation of the General Policy Iteration (GPI) algorithm involves several key steps:

1) **Initialization**: Initialize the value function $V(e)$ arbitrarily (e.g., to zero) for all states e.

2) **Policy Evaluation**: For the current policy π, evaluate the value function $V(e)$ iteratively:

$$V(e) = \sum_{e',u} P(e'|e,u)\left[C(e,u) + \gamma V(e')\right] \quad (8)$$

where $P(e'|e,u)$ are the transition probabilities, $C(e,u)$ is the stage cost, and γ is the discount factor.

3) **Policy Improvement**: Update the policy π by choosing the control action u that minimizes the expected cost:

$$\pi(e) = \arg\min_u \sum_{e'} P(e'|e,u)\left[C(e,u) + \gamma V(e')\right] \quad (9)$$

4) **Convergence Check**: Repeat the policy evaluation and improvement steps until the policy converges, i.e., there are no significant changes in the value function $V(e)$.

IV. Results

NOTE: I have provided two reference images demonstrating how the implementations in parts A and B track the reference trajectory in Figures 1 and 3. I have also provided some statistics for the sample in part A in Figure 3. Unfortunately, I forgot to screenshot the statistics for part B while I was working on the code, and it takes an unbelievable amount of time to rerun the algorithm. For reference, my transition matrices take nearly 4 hours to pre-compute.

A. Part A

In Figure 1, we can see that the CEC implementation does a good job of following the reference trajectory. One subtle point is that the robot sometimes gets very close to the obstacles. This is probably due to the environmental noise that we are not accounting for in the CEC implementation. In terms of runtime, the CEC implementation is exceptionally fast compared to GPI. This could be because Casadi is most likely written in a way that is fully optimized compared to my suboptimal implementation of GPI. Certain flaws I have noticed with CEC are that it sometimes scrapes obstacles and also dramatically deviates from the reference trajectory.

B. Part B

The GPI implementation appears to follow the reference trajectory, but it struggles in certain scenarios. For example, on the outside edges, the robot curves out in the opposite direction and meets the reference robot at the other end of the curve. I'm not quite sure what is causing this, but it could be a result of my control and input space discretization. Given more time, I could have debugged the implementation further, but given how time-consuming running the code is and the limited time to study for finals, this was the best result I could produce.

V. Conclusion

The implementation of the receding-horizon Certainty Equivalent Control (CEC) and Generalized Policy Iteration (GPI) algorithms showcased the strengths and limitations of each approach in trajectory tracking for differential-drive

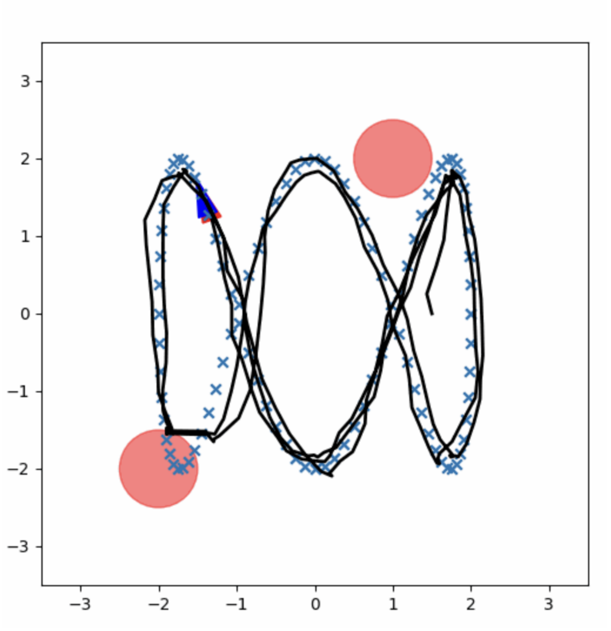
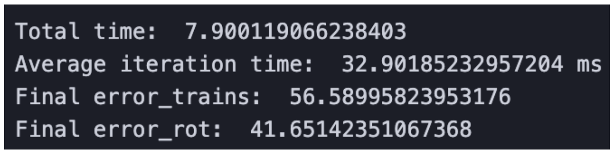


Fig. 1. Sample trajectory from part A



```
Total time:  7.900119066238403
Average iteration time:  32.90185232957204 ms
Final error_trains:  56.58995823953176
Final error_rot:  41.65142351067368
```

Fig. 2. Performance and Runtime from part A

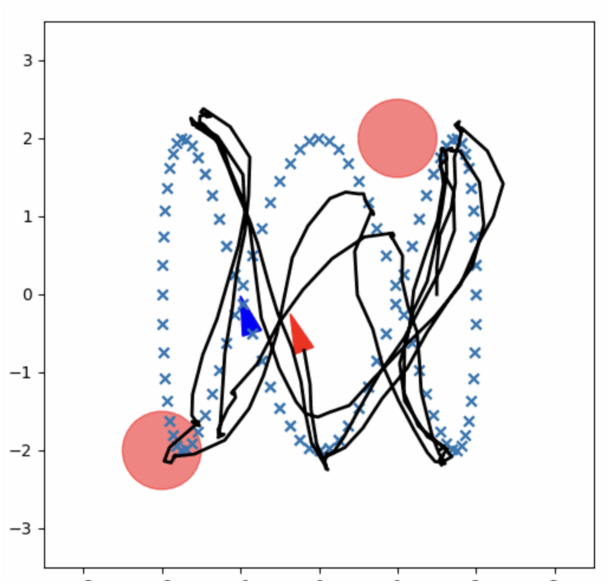


Fig. 3. Sample trajectory from Part B

robots. The CEC method demonstrated efficient runtime performance and effective trajectory following, although it occasionally failed to account for environmental noise, resulting in minor deviations and obstacle collisions. On the other hand, GPI, while computationally intensive, provided a robust framework for policy improvement through adaptive discretization, yet struggled with certain trajectory edge cases. These results highlight the trade-offs between computational efficiency and control accuracy, suggesting that future work could focus on integrating noise management in CEC and refining discretization methods in GPI to enhance overall performance in dynamic and unpredictable environments.

References

[1] https://natanaso.github.io/ece276b
[2] https://web.casadi.org

Contents