Trajectory Planning

Dhruv Talwar
Electrical and Computer Engineering
ECE 276B

I. Introduction

A critical challenge in the field of robotics is trajectory tracking, where robots are tasked with following a predetermined path while simultaneously avoiding obstacles. This problem holds great importance not only for mobile robots but also for manipulator applications, where robots must navigate paths, manipulate objects, and transport them to desired locations while ensuring obstacle avoidance.

In the scope of this project, our aim is to address the trajectory tracking problem for a specific differential drive robot. To achieve this, we explore the implementation of receding-horizon certainty equivalent control (CEC) and generalized Policy Iteration (GPI) techniques. By applying these control strategies, we strive to optimize the robot's ability to track a specified trajectory while effectively avoiding obstacles encountered along the way.

Throughout the project, we focus on fine-tuning various hyperparameters to enhance the tracking performance of the robot. By evaluating the efficacy of the CEC and GPI methodologies, we aim to understand their respective strengths and weaknesses in tackling the trajectory tracking problem. This comparative analysis will provide valuable insights into the performance and suitability of these control approaches for trajectory tracking applications in robotics.

II. Problem Formulation

We possess a ground-based differential drive robot that necessitates adhering to a designated trajectory. Let us assume the robot's state as $x_t := (p_t, \theta_t)$, where $p_t \in \mathbb{R}^2$ denotes the robot's position and $\theta_t \in [-\pi, \pi)$ describes its orientation at time t. To enable the robot's movement, linear velocity in the XY plane and angular velocity around the Z axis are crucial. Thus, we define the control inputs as $u_t := (v_t, \omega_t)$, where v_t represents the linear velocity, and ω_t signifies the angular velocity or yaw rate at time t.

In order to discretize the continuous-time kinematics of the differential drive, we employ a time step of Δ . This discrete-time approximation of the robot's kinematics can be expressed as follows:

$$x_{t+1} = \begin{bmatrix} p_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \Delta \cos\theta_t & 0 \\ \Delta \sin\theta_t & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + w_t \tag{1}$$

The term w_t represents Gaussian noise, which accounts for the imprecision of actuators in achieving an exact position. The noise follows a Gaussian distribution with a mean of 0. It is assumed that the motion noise is independent across time and independent of the robot's state x_t.Additionally, the noise is characterized by a covariance matrix $\sigma = \begin{bmatrix} 0.04 & 0.04 & 0.004 \end{bmatrix}$. It is important to note that the control inputs are bounded to specific ranges: v_t between 0 and 1, and ω_t between -1 and 1. Consequently, the control space can be defined as $U \in [0, 1] \times [-1, 1]$.

The primary goal is to achieve accurate tracking of a specified reference trajectory $r_t \in \mathbb{R}^2$ and an orientation trajectory $\alpha_t \in [-\pi, \pi)$. Simultaneously, the robot must navigate around obstacles within its environment, specifically two circular obstacles centered at coordinates (1, 2) and (-2, -2) with a radius of 0.5. This results in defining the free space for the tracking problem as $F := [-3, 3]^2 - C_1 \cup C_2$, where C_1 and C_2 represent the two circular obstacles.

The error state of the robot is defined as $e_t := (\tilde{p}_t, \tilde{\theta}_t)$. It helps us understand how much the robot deviates from the desired position and orientation. To calculate the position error, we subtract the reference position r_t from the actual position p_t, giving us $\tilde{p}_t$, that is $\tilde{p}_t = p_t - r_t$. Similarly, we calculate the orientation error by subtracting the reference orientation α_t from the actual orientation θ_t, resulting in $\tilde{\theta}_t$, that is $\tilde{\theta}_t = \theta_t - \alpha_t$

When dealing with the orientation error, we need to make sure it stays within the range of $[-\pi, \pi)$. To handle this, we apply a special method called "angle wrap-up" to adjust the orientation error appropriately. This ensures that the error is correctly represented and prevents any weird behavior due to exceeding the range.

Using the error state, we can analyze how well the robot is tracking the desired trajectory. By combining the error state with the control inputs, we can predict the robot's next state. This allows us to evaluate its performance and make adjustments if needed.

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

We formulate the trajectory tracking with initial time τ and initial tracking error e as a discounted infinite-horizon stochastic optimal control problem, and our goal is to find the optimal sequence of controls which minimizes the following eq:

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

subject to:

$$e_{t+1} = g(t, e_t, u_t, w_t), \quad w_t \sim \mathcal{N}(0, \operatorname{diag}(\sigma^2)), \quad t = \tau, \tau+1, \dots$$
$$u_t = \pi(t, e_t) \in U$$
$$\tilde{p}_t + r_t \in F$$

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.

III. Technical Approach

We shall now move on to the Technical Approached followed to implement the two types of methodologies namely receding-horizon certainty equivalent control (CEC) and generalized policy iteration (GPI)

A. Receding-Horizon Certainty Equivalent Control (CEC)

CEC is a suboptimal control scheme that, at each stage, applies the control that would be optimal if the noise variables $\mathbf{w}_t$ were fixed at their expected values (zero in our case). This approach simplifies a stochastic optimal control problem into a deterministic one, which can be solved more effectively. Receding-horizon CEC approximates an infinite-horizon problem by repeatedly solving a discounted finite-horizon deterministic optimal control problem at each time step.

The objective function for the receding-horizon CEC problem is defined as: The objective function for the receding-horizon CEC problem is defined as:

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

Here, $V^*(\tau, \mathbf{e})$ represents the optimal value function, $\mathbf{e}$ denotes the error states, $\mathbf{u}_\tau, \dots, \mathbf{u}_{\tau+T-1}$ represent the control inputs, and $Q(\mathbf{e}_{\tau+T})$ is the terminal cost defined as:

$$Q(\mathbf{e}_{\tau+T}) = \mathbf{p}_{\tau+T}^T \mathbf{Q} \mathbf{p}_{\tau+T} + q(1-\cos\theta_{\tau+T})$$

The receding-horizon CEC problem can be formulated as a non-linear program (NLP) with the following general form:

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

In the above formulation, $\mathbf{U} = [\mathbf{u}_0, \dots, \mathbf{u}_{T-1}]^T$ represents the control inputs and $\mathbf{E} = [\mathbf{e}_0, \dots, \mathbf{e}_T]^T$ denotes the error states. $c(\mathbf{U}, \mathbf{E})$ is the cost function to be minimized, and $h(\mathbf{U}, \mathbf{E})$ represents the constraint functions. $\mathbf{U}_{\text{lb}}$ and $\mathbf{U}_{\text{ub}}$ are the lower and upper bounds on $\mathbf{U}$, while $\mathbf{h}_{\text{lb}}$ and $\mathbf{h}_{\text{ub}}$ are the lower and upper bounds on $h(\mathbf{U}, \mathbf{E})$.

In the receding-horizon CEC approach, the lower and upper bounds of the control actions, denoted as Ulb and Uhb respectively, are defined. In our specific case, Ulb is set to [0, -1] and Uhb is set to [1, 1]. The function c(U, E) represents the total tracking error, as defined in Equation (5), given by:

$$c(U, E) = q(e_{\tau+T}) + \sum_{t=\tau}^{\tau+T-1} \left(\tilde{p}_t^T Q \tilde{p}_t + q(1-\cos\tilde{\theta}_t)^2 + u_t^T R u_t \right) \tag{7}$$

The function h(U, E) encompasses any additional constraints that may be required for our problem. These constraints can take the form of inequalities, such as hlb h(U, E) hub. In the case of CEC control, one class of h functions describes the error dynamics equation, given by et+1 = g(t, et, ut, 0). Another class of h functions represents the bounds on the error function, defined as $\tilde{p}_t + \tilde{r}_t \in \mathcal{F}$.

The optimization problem can be solved using an NLP solver such as CasADi, which provides an interface to IPOPT. CasADi formulates the NLP problem and solves it iteratively, yielding the optimal control sequence $\mathbf{U}^*$. The first control input u_0^* is applied, and the process is repeated at the next time step with the updated error state to obtain the subsequent control inputs.

Optimization variables: The control inputs $\mathbf{u}[u_0, \dots, u_{T-1}]^T$ and error states $\mathbf{e}[e_0, \dots, e_{T-1}]^T$ are defined as the variables to be optimized. The various parameters are initialized as follows:

- $\mathbf{Q} = \mathbf{I}$
- $\mathbf{R} = \mathbf{I}$
- $q = 1$
- $\gamma = 1$

Constraints: The constraints and boundary conditions for the Nonlinear Programming (NLP) problem are set using the Opti package. The constraints for the control inputs $\mathbf{u}_t \in \mathcal{U}$ are defined as:

- Lower bound: $0 \le v \le 1$

- Upper bound: $-1 \leq \omega \leq 1$

The constraints for the robot space coordinates $(x, y) = \tilde{p}_t + \tilde{r}_t \in \mathcal{F}$ are defined as:

- $-3 \leq x \leq 3$
- $-3 \leq y \leq 3$
- $(x+2)^2 + (y+2)^2 > 0.5^2$
- $(x-1)^2 + (y-2)^2 > 0.5^2$
- The constraint for the error $\mathbf{e}_{t+1} = g(t, \mathbf{e}_t, \mathbf{u}_t, 0)$ is also set.

The objective of the NLP program is to minimize c(U, E)

Once the NLP formulation is complete, the optimization is performed by setting the solver to IPOPT using solver('ipopt'), and then solving the NLP problem and computing all the required variables using solve() function. The predicted controls for only the next step, $\mathbf{u}_{\tau+1}$, are then returned for executing the robot motion using that input at the next instant.

B. Generalized Policy Iteration

In this section now, we will implement Value iteration for tracking the error thus solving the infinite horizon stochastic optimal control problem. The approach to solve the problem is as follows: The steps to be followed are as follows:

1: The first step in applying the algorithm is to discretize the state space into a set of discrete grids.
2: The reference trajectory is a periodic motion with a period of 100 time steps, and time is defined as a set of 100 points incrementing by 0.5, ranging from 0 to 50.
3: Define the adaptive grid for the position of the robot by creating x-coordinate error ($error_x$) and y-coordinate error ($error_y$). Initialize the values.
4: Discretize the orientation error of the robot (θ) using a predefined set of values.
5: Discretize the state space into $x_{steps} = y_{steps} = 9$, $time_{steps} = 100$, and $\theta_{steps} = 9$, resulting in a total state space size of $|X| = 71,000$.
6: Discretize the control space by defining grids for linear velocity (v_t) and angular velocity (ω_t). I decided to discretize them into steps of 10 each, resulting in the cardinality of the control space as $|U| = 100$.
7: Use expectation to calculate the probabilities to reach the next state as now we have noise in our model.
9: Utilize the GPI algorithm to compute the value function and derive the optimal policy for minimizing the tracking error, considering the influence of noise.

TABLE I
DISCRETIZATION OF VARIABLES

Variable	Discretization
Time (t)	0, 0.5, 1, ..., 49, 49.5, 50
x-coordinate error	[-3, -0.5, -0.25, -0.15, -0.05, 0.05, 0.15, 0.25, 0.5, 3]
y-coordinate error	[-3, -0.5, -0.25, -0.15, -0.05, 0.05, 0.15, 0.25, 0.5, 3]
Orientation error (θ)	[-180°, -90°, -45°, -27°, -9°, 9°, 27°, 45°, 90°, 180°]
Linear velocity (v_t)	[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
Angular velocity (ω_t)	[-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1]

1) Calculation of State Transition Probabilities:

To account for the presence of noise in the system, we need to calculate the state transition probabilities. The following steps outline the process:

Calculate the expected next state: Given the current state e_t and control u_t, we compute the expected next state using the transition function $g(t, e_t, u_t, 0)$, assuming zero noise. This provides us with an estimate of the next state without considering any disturbances.

Incorporate noise and select neighboring points: Due to the presence of noise, the actual next state can deviate from the expected next state. We consider a Gaussian noise distribution to model this uncertainty. We select the nearest neighboring points in the discretized state space grid to account for the potential variability in the next state. in our case we have used 10 nearest points.

Determine the likelihood of reaching neighboring points: To determine the probabilities of transitioning to the neighboring points, we assume a Gaussian noise distribution with a mean $\mu = g(t, e_t, u_t, 0)$ and a covariance matrix $\Sigma = \text{diag}(0.04, 0.04, 0.004)$. The probability density function of the Gaussian distribution is given by:

$$p(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

where x represents the variable, n is the dimensionality of x, μ is the mean vector, and Σ is the covariance matrix.

We evaluate this probability density function for each neighboring point, representing the likelihood of reaching that specific point considering the noise.

Step 4: Normalize the probabilities: After determining the likelihoods of reaching the neighboring points, we sum up these probabilities and normalize them. This normalization ensures that the probabilities form a valid probability distribution, with all probabilities summing up to 1.

By following these steps, we obtain the state transition probabilities, which can be represented as a three-dimensional matrix of dimensions $R|X| \times |U| \times |X|$.

Here, R represents the number of possible transitions from a given state, $|X|$ denotes the size of the state space, and $|U|$ signifies the size of the control space. Each entry in the transition probability matrix corresponds to the probability of transitioning from a specific state-action pair to a subsequent state, thus this matrix corresponds to our motion model. By considering the entire matrix, we can comprehensively model the stochastic nature of the system and derive effective control policies that minimize tracking errors.

The step cost for our Markov Decision Process (MDP) is formulated as follows:

$$l(x_t, u_t) = \tilde{p}_t^T Q_p \tilde{p}_t + q(1 - \cos\tilde{\theta}_t)^2 + u_t^T R_u u_t + \text{penalty}$$

where $\tilde{p}_t$ denotes the position error, $\tilde{\theta}_t$ represents the orientation error, Q_p and R_u are weighting matrices, and q is a scalar parameter.

To further ensure the effectiveness of the cost function, we introduce a penalty term based on the proximity of the position error to the reference trajectory. The penalty is determined by the following condition:

Condition	Penalty
$\|\|\tilde{p}_t + r_t\|\|^2 < 0.52$	80
$\|\|\tilde{p}_t + r_t\|\|^2 > 0.52$	0

I have used 0.52 which acts as a buffer so that the agent does not collide into the obstacle.

where r_t denotes the reference trajectory.

By incorporating these elements into the step cost equation, we account for the impact of position error, orientation error, control input, and potential collisions. This comprehensive approach allows us to assess the cost associated with specific state-action pairs within our MDP process.Using our formulation we can now utilize the VI algorithm with all its elements

The Value Iteration algorithm can be defined as

Value Iteration Algorithm

```
1: MDP with state space $X$, action space $U$, transition probabilities $P$, cost $L$, discount factor $\gamma$
2: Initialize V$_0$
3: for k = 0,1,2... do
4:     for each state $x \in X$ do
5:         for each action $u \in U$ do
6:             Cal expected value for action $u$ in state $x$:
7:             $Q(x,u) = L(x,u) + \gamma \sum_{x'} P(x'|x,u)V(x')$
8:             Update the value function for state $x$:
9:             $V_{k+1} = \min_{u \in U} Q(x,u)$
10:            Derive policy $\pi$ from value function $V$:
11:            $\pi(x) = \arg\min_{u \in U} Q(x,u)$ for $x \in X$
12:        end for
13:    end for
14:    if $\|V_{k+1} - V_k\|_2 \leq 1 \times 10^{-6}$ then
15:        return $\pi^*(x)$
16:    end if
17: end for
```

Once the Value Iteration algorithm converges, we obtain the optimal policy function and the optimal value function. We can then directly use the optimal policy given the current time and error to determine the optimal policy.

In the following section, we present the results obtained using the CEC controller and the GPI controller.

IV. Results

We have successfully implemented both the Receding Time Horizon CEC controller and the infinite horizon Generalized policy Iteration Controller. In this section we shall present the results from varying different parameters in both controllers.

1) Receding Time Horizon Certainty Equivalent Control: The CEC controller addresses the problem of stochastic optimal control tracking in an infinite time horizon by transforming it into a deterministic optimal control problem with a finite time horizon.

Please note that I have used the parameter ”False” for noise in the motion model as it was given in the main question.

The following table shows the effects of parameters used in the controller

TABLE II
Effect of T

Q	q	R	T	Avg Iteration Time	Total Time	Final Error
10	10	5	10	184.35	44.26	103.03
10	10	5	15	426.164	102.301	102.301
10	10	5	25	1313.362	315.229	103.104

From the table we can conclude that as the value of T increases from 10 to 15 to 25 the total time and Avg Iteration time increases drastically. Also we can see that there is a slight effect of increasing T in the CEC controller as a higher T may produce almost the same or better tracking performance but will take longer to compute. A shorter time horizon will be less computationally expensive.

TABLE III
Effect of Q

Q	q	R	T	Avg Iteration Time	Total Time	Final Error
10	10	5	10	184.35	44.26	103.03
20	10	5	10	194.692	46.74	81.2417
21	10	5	10	188.54	45.27	82.728

From the table above we can conclude that as the value of Q increases keeping all other parameters same, the trajectory error decreases. The parameter Q in the tracking error cost penalizes the controller for error in the position tracking. As the value of Q increases, the controller is more heavily penalized for errors, which leads to improved tracking performance.

Also we cannot increase Q too much as when I used Q value as 30 the code was not able to find a solution.

TABLE IV
Effect of small Q

Q	q	R	T	Avg Iteration Time	Total Time	Final Error
10	10	5	10	184.35	44.26	103.03
10	30	5	10	179.70304	43.14	94.1875
10	80	5	10	192.08	46.19	87.088

From the table above we can conclude that as the value of q increases keeping all other parameters same, the trajectory error decreases. This is because a higher value of q penalizes the controller more for errors in orientation tracking. The parameter q in the tracking error cost puts emphasis on tracking the reference orientation. This puts emphasis on the orientation as tracking only the position, though precisely can lead to an increase in the error. Thus we can see in the table that for q value 80 we have the lowest final error.

From the table above we can conclude that as the value of R increases keeping all other parameters same, the trajectory

TABLE V
EFFECT OF R

Q	q	R	T	Avg Iteration Time	Total Time	Final Error
10	30	5	10	179.70304	43.14	94.1875
10	30	10	10	185.291	44.48	127.48
10	30	15	10	41.45	172.649	154.03

error also increases. The R term penalizes the controller for exerting too much control effort. Thus if the value of R is too high the controller will penalize it for putting too much effort resulting in a poor overall trajectory.

The trajectory from the CEC controller can be seen in figure 1. I had run the controller for 2 loops (till the agent runs at least once on all parts of the track. The black line shows the trajectory of the agent. The red triangle is the agent and the blue one is the reference path to follow.

2) Generalized Policy Iteration: The current situation involves the usage of the GPI (Generalized Policy Iteration) controller, which is designed to optimize the agent's behavior based on a given reference path. However, it is observed that the agent's movements deviate from the expected path, following a rather strange trajectory.

Figure 2 will show the output I got from this as I was not able to get the correct path in the due time. The code is working fine all the equations are there.

Even though I am able to make the state transition matrix, the stage cost matrix and am able to run the value iteration function well and minimize the V function over all pi, when I am running the code the difference between the subsequent V is not the same, in one iteration it was 210, then it went down to 209.99 but then it suddenly jumped to 546.88 and then after 2 iterations it again went down to 209 then a few iterations after again this happened

Several factors can contribute to this behavior. Insufficient training or exploration during the policy iteration process may lead to suboptimal policies that do not accurately capture the desired actions for different states.

In addition, the representation of the problem may not adequately capture the underlying dynamics, resulting in a mismatch between the learned policy and the actual system behavior. Moreover, modeling inaccuracies or unanticipated dynamics can introduce uncertainties, causing the agent to exhibit unexpected behavior. Addressing these challenges requires careful analysis of the training process, representation choices, hyperparameter settings, and model accuracy to align the GPI controller with the desired reference path and improve the agent's behavior accordingly.

I ran the code for Q 10, q,10 R, 2 and 5. Both results are shown with the report. Due to time constraint I was unable to correct this.

Here are the functions I used in the final code

check_collision(x, y): This function checks for collisions between the system and obstacles. It takes the coordinates of the system, x and y, as input and computes the penalty for collision based on the distance between the system and predefined obstacle positions.

stage_cost(error, control, time, ref=cur_ref): This function calculates the cost of taking a step in the Markov Decision Process (MDP). It considers the current state of the system, error, the control input, control, the current time step, time, and the reference trajectory, ref. The cost is computed based on the position and orientation error, control input, and penalty for collision.

create_L(i): This function creates the cost-to-go vector for a specific state in the MDP. It takes the current state index, i, as input and calculates the cost-to-go for each possible control input at that state using the stage_cost function.

value_iteration_matrix(V, L, p): This function performs value iteration on the MDP to find the optimal policy. It takes the current value function, V, the cost-to-go matrix, L, and the transition matrix, p, as input. It iteratively updates the value function and computes the optimal policy until convergence. The convergence criterion is based on the difference between the new value function and the previous value function.

V. CONCLUSION

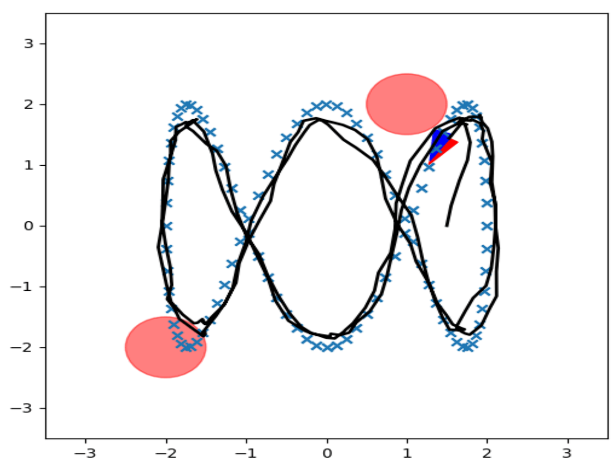
n this project, we implemented the CEC (Control Error Compensation) controller and explored its performance in the trajectory tracking problem. The CEC controller showcased impressive results in terms of trajectory tracking accuracy. This can be attributed to its deterministic nature and the utilization of a tailored control strategy designed specifically for the task at hand. The CEC controller directly leverages known information without the need for an exploration phase, allowing for precise control decisions based on the system's dynamics.

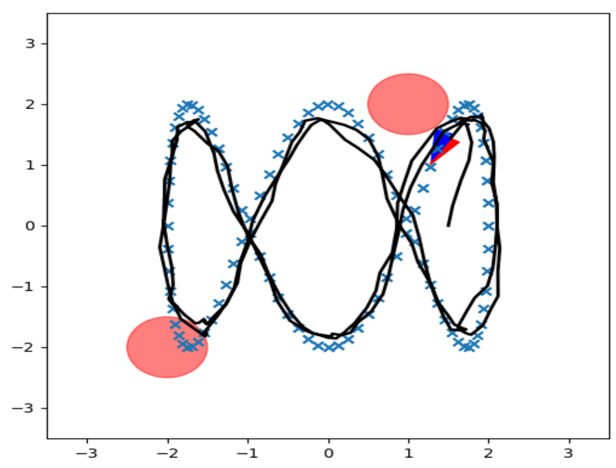
However, due to unforeseen challenges and limitations encountered with the Generalized Policy Iteration (GPI) algorithm, we were unable to successfully run and evaluate its performance in this project. The GPI algorithm requires a careful balance between exploration and exploitation, along with accurate value function approximation techniques. Despite our best efforts, these challenges hindered the implementation and execution of the GPI algorithm, preventing a comprehensive comparison between the two approaches.

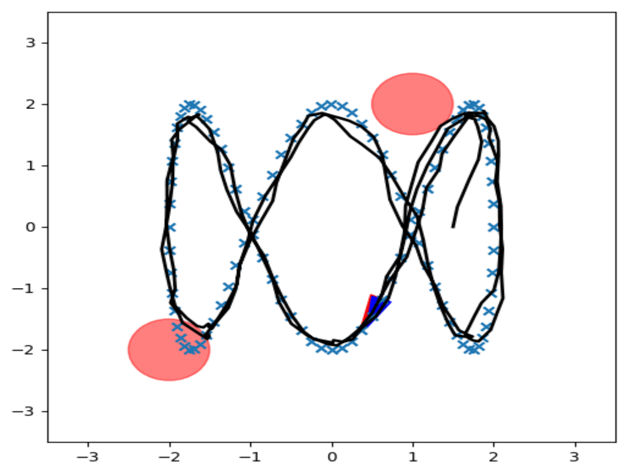
As a result, our focus remained primarily on the CEC controller, which demonstrated promising trajectory tracking accuracy. Future work could involve further investigation and refinement of the GPI algorithm to overcome the encountered challenges and enable a more comprehensive evaluation of its performance. Nevertheless, the CEC controller's deterministic nature and tailored control strategy make it a compelling choice for trajectory tracking tasks, showcasing its potential for effective control and accurate trajectory following.

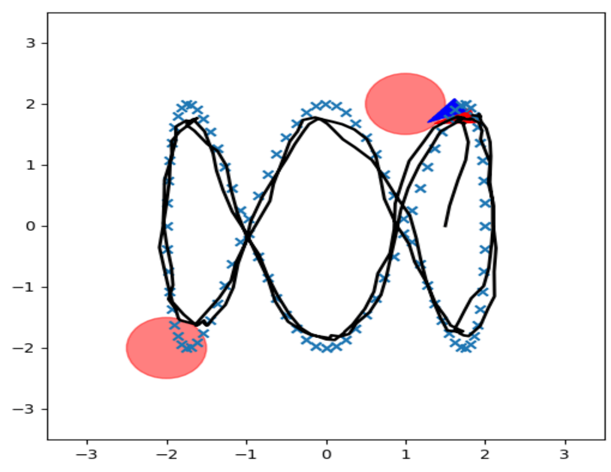
A. Collaboration

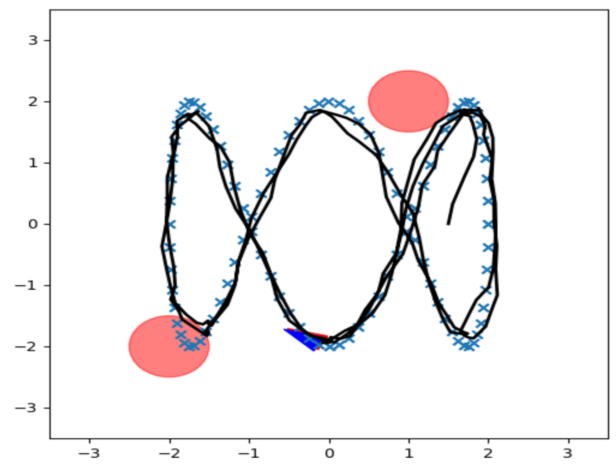
Surya Pilla


(a) Q = 10, q = 10, R = 5, T = 10


(b) Q = 10, q = 10, R = 5, T = 10


(c) Q = 20, q = 10, R = 5, T = 10


(d) Q = 10, q = 30, R = 5, T = 10


(e) Q = 21, q = 10, R = 5, T = 10

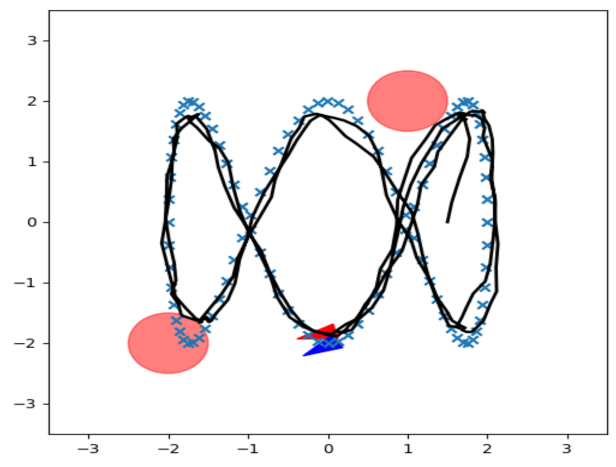
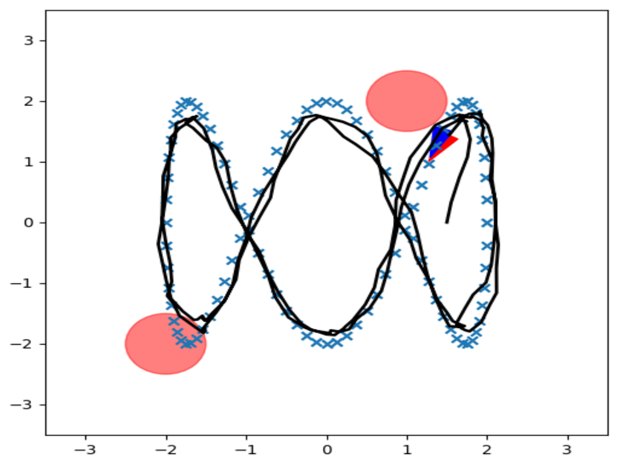
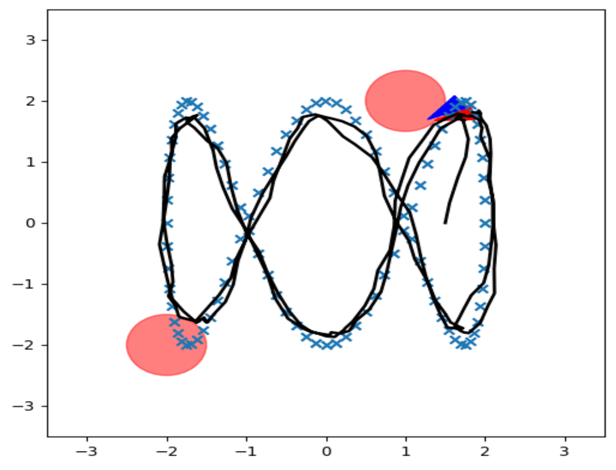

(f) Q = 10, q = 80, R = 5, T = 10

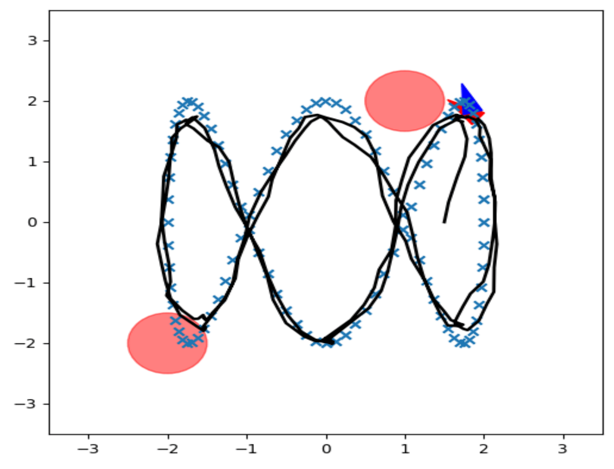
Figure 1: CEC controller: Effects of Q and q respectively



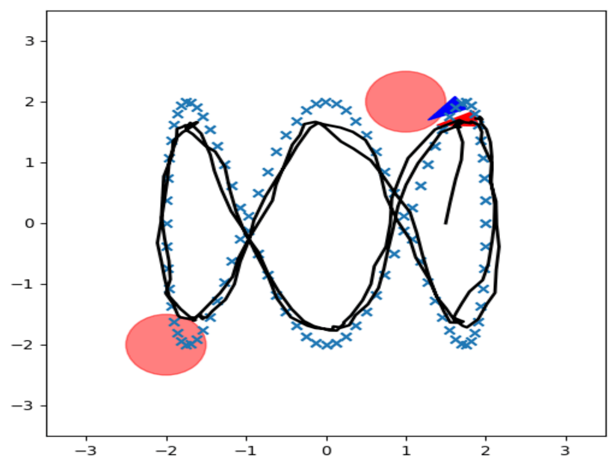
(a) Q = 10, q = 10, R = 5, T = 10



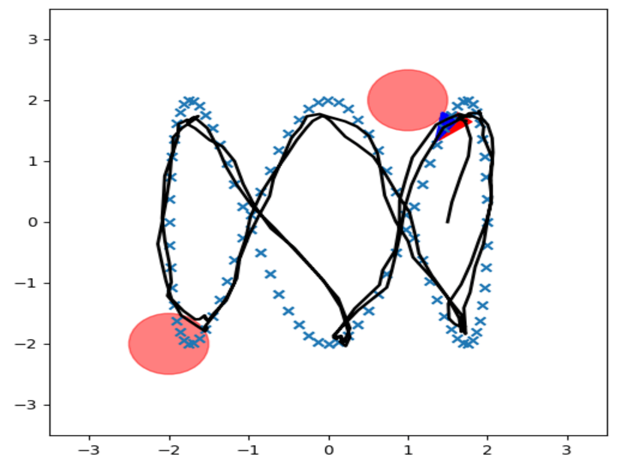
(b) Q = 10, q = 30, R = 5, T = 10



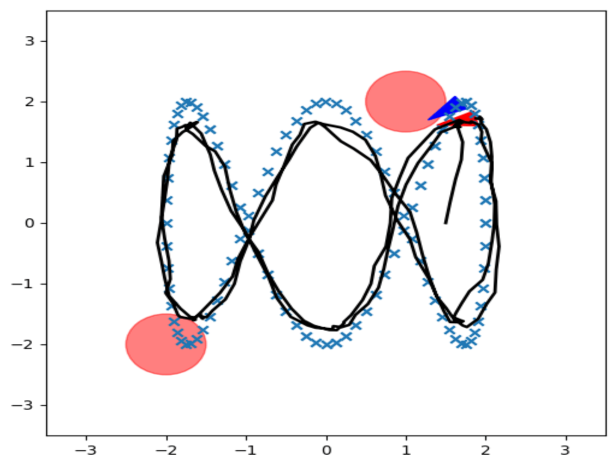
(c) Q = 10, q = 10, R = 5, T = 15



(d) Q = 10, q = 30, R = 10, T = 10

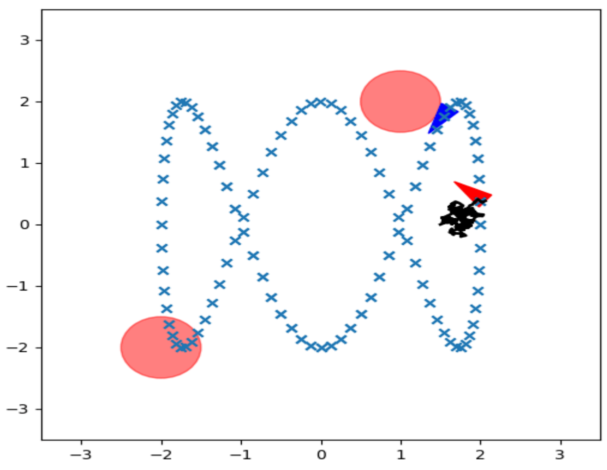


(e) Q = 10, q = 10, R = 5, T = 25

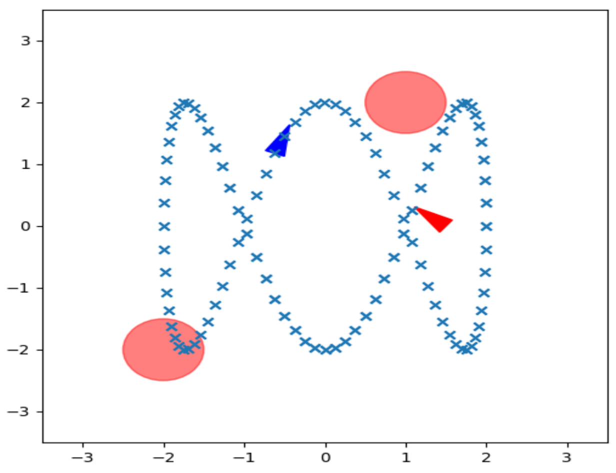


(f) Q = 10, q = 30, R = 15, T = 10

Figure 2: CEC controller: Effects of T and R respectively



(a) Q = 10, q = 10, R = 2



(b) Q = 10, q = 30, R = 5

Figure 3: GPI controller