Infinite Horizon Stochastic Optimal Control

Surya Prakash Thoguluva Kumaran Babu
Department of Mechanical and Aerospace Engineering
University of California San Diego
stk222@ucsd.edu

***Abstract*—The objective of this paper is to implement infinite horizon optimal control algorithms. In this project, we have explored receding-horizon certainty equivalent control (CEC) to convert the infinite horizon problem into an equivalent deterministic finite horizon and also the Generalized Policy Iteration method to directly solve the infinite horizon stochastic control problem.**

***Index Terms*—Infinite Horizon Optical Control, Receding-horizon certainty equivalent control (CEC), Generalized Policy Iteration**

I. Introduction

Robotic planning is an important area of research in the robotics domain. Given sensor data about the environment, the robot needs to know what it should do next to achieve a specific task. This is the exact objective of which planning and learning problems in robotics. Planning in robotics enables robots to operate in complex environments, handle uncertain situations, and accomplish tasks with a high degree of accuracy and precision. A subset of this is the robot path planning problem which can be solved in various ways. There has been a lot of research over the past few decades which tried to solve this problem. However, even today, although the high-level problem formulation is the same, we have different challenges. For example, the current auto-pilot control in some cars is still not 100% reliable and there is a lot of room for improvement.

In this paper, explored ways to design a trajectory-following algorithm as an infinite horizon optical control problem. We have explored 2 ways to solve it. The first is with Receding-horizon certainty equivalent control (CEC) where the infinite horizon problem is converted to a finite horizon problem to get control inputs of which we just apply the first input and then replan at every timestep. The second method is to use Generalized policy iteration to solve the infinite horizon problem directly.

II. Problem Formulation for Infinite Horizon Stochastic Optimal Control

A. Given Problem

We are given a ground differential drive robot, a reference trajectory, and an environmental obstacle, we need to design a control that minimizes the error between the reference trajectory and the actual trajectory. The robot states are given as $x_t = (p_t, \theta_t)$ where $p_t \in \mathbb{R}^2$ and orientation $\theta_t \in [-\pi, \pi]$ at discrete-timesteps $t \in \mathbb{N}$. The control input to the robot is a velocity vector of $u_t = (v_t, \omega_t)$ where $v_t \in \mathbb{R}$ and $\omega_t \in \mathbb{R}$ where v_t is the linear velocity and ω_t is the angular velocity. The discrete-time kinematic model of this differential drive robot is given as

$$x_{t+1} = \begin{bmatrix} p_{t+1} \\ \theta_{t+1} \end{bmatrix} = f(x_t, u_t, w_t) = \begin{bmatrix} p_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \Delta cos(\theta_t) & 0 \\ \Delta sin(\theta_t) & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + w_t, t = 0, 1, 2, .. \tag{1}$$

where $w_t \in \mathbb{R}^3$ models the motion noise with guassian distribution $N(0, diag(\sigma)^2)$ with standard deviation as $\sigma = [0.04, 0.04, 0.004] \in \mathbb{R}^3$. The noise is independent of the states.

The reference trajectory is specified with position trajectory as $r_t \in \mathbb{R}^2$ and orientation trajectory $\alpha_t \in (-\pi, \pi)$. There are 2 circular obstacles C_1 with center at $(-2, -2)$ with radius 0.5 and C_2 centered at $(1, 2)$ with radius 0.5. The free space of the environment is given as $F = [-3, 3]^2 - (C_1 \cup C_2)$. An image of the given environment is shown below in figure 1.

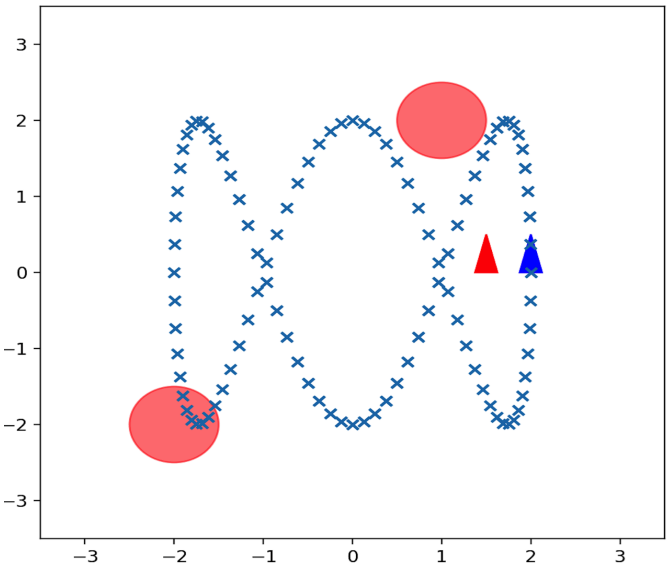

Fig. 1. Given Environment Figure

B. Markov Descision Process

This section provides the MDP setup for this project.

- State Space: The given problem is a trajectory tracking problem and since the reference trajectory can be different in different time steps, we could use the error between the reference trajectory and the current state as the state space along with time time index.

$$\begin{aligned} \chi \in [e_t, t]^T, e_t \in [\overline{p_t}, \overline{\theta_t}], \\ \overline{p_t} = p_t - r_t, \overline{\theta_t} = \theta - \alpha_t \end{aligned} \tag{2}$$

where $\overline{p_t} \in \mathbb{R}^2$ is the error in position and $\overline{\theta_t} \in [-\pi, \pi]$ is the error in orientation, and $t \in \mathbb{N}$ is the time step.
- Control Space U: The control space is a vector with 2 elements as linear velocity v_t and angular velocity ω_t

$$U = [v_t, \omega_t]^T, \ v_t \in [0, 1], \omega_t \in [-1, 1] \tag{3}$$

- Motion Model $p_f(.|x_t, u_t)$: The kinematic model described in equation 1 defines the probability density function $p_f(x_{t+1}|x_t, u_t)$ as a gaussian distribution with mean $x_t + G(x_t)u_t$ and covariance $diag(\sigma)^2$. Since the state space we have defined is in terms of error, the updated motion model $g(e_t, t, u_t, w_t$ is given as:

$$\begin{aligned} x_{t+1} = \begin{bmatrix} \overline{p_{t+1}} \\ \overline{\theta_{t+1}} \end{bmatrix} = f(x_t, u_t, w_t) = \begin{bmatrix} \overline{p_t} \\ \overline{\theta_t} \end{bmatrix} + \\ \begin{bmatrix} \Delta cos(\overline{\theta_t} + \alpha) & 0 \\ \Delta sin(\overline{\theta_t} + \alpha) & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \begin{bmatrix} r_t - r_{t+1} \\ \alpha_t - \alpha_{t+1} \end{bmatrix} + w_t \end{aligned} \tag{4}$$

- Time Horizon T: Since this is an infinite horizon problem, T is set to infinite.
- Stage Cost $l(x_t, u_t)$: As this is a control problem and we are looking to minimize the error between the current position and desired position, any deviation from the reference state should be penalized. For our problem, we have assumed a quartic cost. In order to have a smooth trajectory, the control input that is applied must be as close to 0 as possible. Hence a penalization term for the input is also taken into account.

$$l([\overline{p_t}, \overline{\theta_t}], u_t) = \overline{p_t}^T Q \overline{p_t} + q(1 - cos(\overline{\theta}_t))^2 + u_t^T R u_t \tag{5}$$

where $Q \in \mathbb{R}^{2\times2}$ is a symmetric positive definite matrix that penalizes the error in position, $q \in \mathbb{R}$ penalizes the error in orientation and $R \in \mathbb{R}^{2\times2}$ symmetric positive definite matrix that penalizes the effort put in by the controller.
- Terminal Cost $q(x_T)$: Since this is an infinite horizon problem, the terminal cost is not defined.
- Discount factor γ: The value of $\gamma \in [0, 1]$ depends on the priority of long-term gain versus short-term gain.
- Optimal Policy $\pi^*(.)$: It is a function that maps the state space to control space which gives the minimum long term cost.
- Optical Value Function $V^*(.)$: It is the long term cost of following an optimal control policy.

C. Infinite Horizon Stochastic Optimal Control

The infinite horizon optimal stochastic optimal control in an MPD is defined:

$$V^*([e_t, t]) = \min \mathbb{E}[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} l([\overline{p_t}, \overline{\theta_t}], u_t)]$$

$$s\ t.\ e_{t+1} = g(e_t, t, u_t, w_t)\ u_t = \pi(e_t, t) \in U\ \overline{p_t} + r_t \in F$$

The objective is to find an optimal value and policy using the above formulation. It can be solved in two ways as in CEC and generalized policy iteration.

III. Technical Approach for CEC

A. Receding Horizon CEC Formulation

Certainty equivalent control (CEC) is a suboptimal control sequence that will be optimal if the noise in the motion model were to be zero. The main idea is to reduce a stochastic optimal control problem into a deterministic optimal control problem. Receding horizon CEC further converts the deterministic optimal control problem into finite horizon deterministic optimal control problem. A receding CEC for a relatively small time horizon is solved in order to predict a control input. Then the first input is applied and using the next state the control sequence is again recomputed and the first optimal control is applied. This is done repeatedly over and over. This is a form of model predictive control where a control now is taking into account a small prediction into the future and gives an optimal control for this small window into the future.

Mathematically the receding horizon CEC can convert the objective function in our formulation into the following discounted finite horizon determinsitc optimal control problem at each time step:

$$V^*([e_t, \tau]) = \min_{u_\tau, \ldots u_{\tau+T-1}} q(e_{\tau+T}) +$$

$$\sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} l([\overline{p_t}, \overline{\theta_t}], u_t)]\ s\ t.\ e_{t+1} = g(e_t, t, u_t, 0)$$

$$q(e_{\tau+T}) = \overline{p_{\tau+T}}^T Q \overline{p_{\tau+T}} + q(1 - cos(\overline{\theta}_{\tau+T}))^2$$

$$u_t \in U\ \overline{p_t} + r_t \in F,\ t = \tau, \tau + 1..T - 1$$

where the terminal cost $q_{e_{\tau+T}}$ is defined now. This further can be reduced to a nonlinear probleming (NLP) of the form:

$$\min_U c(U, E)$$

$$s.t.\ U_{lb} \leq U \leq U_{ub}\ and$$

$$h_{lb} \leq h(U, E) \leq h_{ub}$$

where $U = [u_\tau^T, ..., u_{\tau+T-1}^T]$ and $E = [e_\tau^T, ...e_{\tau+T}^T]$ represents all the states and control sequences vertically concatenated into one big vector respectively. CasADi is used to solve such NLP to obtain the control sequence. After solving the NLP, the first input is applied to move the agent and a new state is obtained which is then used to solve a new NLP and the process is repeated.

B. Receding Horizon NLP Implementation

An NLP is setup by providing an objective function, constraints and decision variables. Given below is a brief of the problem setup:

- Time Horizon T: This is a hyperparameter that tells us how far into the future are we supposed to look while performing receding horizon CEC. The effect of these are discussed in the discussion section.
- Decision Variables: A $T+1$ number of states and T a number of control inputs are the decision variables for which optimized values are supposed to be found. State decision variable will be of size $3 \times T$ where each column corresponds to x position error, y position error, and angle position error. The control decision variable will be of size $2 \times T$ where each column corresponds to linear velocity and angular velocity.
- States Boundary Constraints: The constraints that make the x position, and y position of the states to be within bounds of $[-3, 3]$ is applied. There are no constraints on the angle state as all possible angles are covered from $[-\pi, \pi]$. The actual position and orientation are computed by taking a sum of the error states with the reference state at that instant and that resultant sum is now used to apply this states boundary condition.
- Control Boundary Constraints: The constraints that limit the linear velocity to $[0, 1]$ and angular velocity to $[-1, 1]$ are applied.
- Initial Condition Constraints: The initial error when the program is initialized is known. That will be the error between the reference trajectory and the current state of the agent. A constraint is added to fix the initial condition of the state variable at time $t = \tau$ to be the current error.
- Motion Model Constraint: The error states need to respect the error states motion model as in equation 4. At each iteration, the next error states are computed based on the current error state, control input, and reference trajectories. The next iteration error state is a constraint to the computed new state error.
- Objective Function: A running γ weighted sum of the stage cost is added to a variable throughout the time horizon T during the iteration. The terminal cost on the state is computed and added at the end of the iteration to that same variable. This variable is passed as an objective function for the NLP to minimize.
- Obstacle Constraint: An constraint that computes the current x,y position of the robot from its error states and reference trajectory and checks the distance between the robot and obstacle and constraints the distance to be more than the radius of the obstacle.
- Discount factor γ: The gamma value can be in the range of $[0, 1]$ where 0 will penalize the current stage cost more than the future stage cost. For our case a discount factor of $\gamma = 0.85$ worked better.
- Cost penalization: The Q matrix is set to be $0.7I$ and q is set to 1 and R is set to be I. The factor of multiplication tells us the weightage of each error in the objective function. A higher factor for Q, q, and R means we are penalizing heavily for the error in the position, error in the orientation, and effort in the input respectively, compared to other terms. Currently, we have set equal weightage for all the errors except for the position error which is set to be 0.7. This is due to the fact that the displacement is incremental and even a slightly larger error in displacement can be rectified over time.

IV. Generalized Policy Iteration

A. Generalized Policy Iteration Formulation

GPI is a way to directly solve for the optimal value function and optimal control problem formulated as discounted infinite horizon stochastic optimal control problem. The given objective function can be converted to a bellman equation of the form:

$$V^*([e_t, \tau]) = \min_u (l(e_t, u) + \gamma \mathbb{E}_{e_{t+1,t+1} \sim p_f(.|e_t,u)}[V^*([e_{t+1}, \tau + 1])]$$

B. Bellman Equations Algorithms

The above equation can be solved by value iteration or policy iteration or generalized policy iteration. Policy iteration is a two-step process as described below:

- Policy Evaluation: Given a policy π compute the value function associated with it V^π. Initialize V^π to be all zeroes and then iterate through the below equation for n times. If $n = 1$ then the Bellman equation algorithm is called value iteration. If $n \to \infty$ then the bellman algorithm is called policy iteration. If $1 \leq n \leq \infty$ then the algorithm is called generalized policy iteration. The policy evaluation step of generalized policy iteration for the discounted problem is given as

$$V_{k+1}(e_t, \tau) = l(e_t, \pi(e_t)) + \gamma \mathbb{E}_{e_{t+1} \sim p_f(.|e_t,\pi(e_t))}[V_k(e_{t+1}, \tau + 1)]$$

- Policy Improvement: Given V^π, obtain a new policy π' using the below step:

$$\pi' \in arg\min_u l(e_t, u) + \gamma \mathbb{E}_{e_{t+1} \sim p_f(.|e_t,u}[V^\pi(e_{t+1}, \tau + 1)]$$

C. Generalized Policy Iteration Implementation

The continuous space is discretized in order to perform policy iteration.

- State space discretization: The error in position, error in orientations, and the time stamp are the states. Since the space is continuous the error in position and orientation will also be continuous. The error in position is discretized as 10 points between $[[-3, 3]$ that are equally spaced. The orientation error is split as 20 discrete points between $-\pi, \pi$ that are equally spaced. All possible combinations of the discrete values are used to create the state space.
- Control space discretization: The linear velocity is continuous between $[0, 1]$ and angular velocity is continuous

between $[-1, 1]$. Both the control inputs are discretized into 10 equally spaced values. All combinations of these 10 points will give us the control space.
- Stage Cost: A matrix of size all states and all control inputs are constructed and the stage cost is evaluated for all control inputs and all control states.

Once the above matrices are precomputed, value iteration function is called with current robot state and current reference to find and optimal policy. An overview of the value function algorithm is as shown below:

- The current error is estimated based on the reference position and KDTree is used to find the nearest discrete errror state.
- A state transition matrix function is called which takes the current time index as input. Based on the time index, for all states and for all control inputs, based on the reference index, all the next states are computed using the motion model.
- All the next states are passed into collisioncheck function. This returns a boolean vector which returns true if the next states is outside the limits and if it is colliding with an obstacles. Else false is returned. For all the next states that are collision free, a guassian function is called with these states. The guassian function takes each of the valid next states as mean and likelihood of all the discrete states locations is found and assigned. Using this functions a state transition matrix for all states, for all control inputs to all next states the probabilities are estimated and sent as the state transition matrix.
- Using the precomputed stage cost, the motion model and intial value function which was initialized to zero, value iteration equation is repeatedly applied for 50 iterations to get the optimal value function and the optimal policy for all states at a particular time step.
- The optimal policy corresponding to the nearest error state corresponding to the actual error is returned as output

The value iteration is then repeated for all the current states of the robot.

V. Discussion for Receding Horizon CEC

A. Effect of Time Horizon

A brief summary of changing the time horizon in CEC are mentioned below based on table 1:

- As we increase the time horizon the final error almost remains constant. This is not as per expectation because now we can see far more into the future and make an informed decision.
- However when we increase the time horizon, the iteration time increases. This will have an effect on the response speed of the controller.
- There is an upper limit and lower limit on the time horizon as the solver throws errors if the horizon is outside the feasible bounds.

B. Effect of Discount Factor

The effects of the discount factor are discussed below based on table 2:

- The final error keeps decreasing as we reduce the discount factor. This is as per expectation because, if we reduce the discount factor we give lesser and lesser priority for the future states. This is close to reality because the motion model has noise and not trusting the future states is beneficial.
- The computation time is almost similar for all the test cases except there is an anomaly when the discount factor is 0.5.
- The best discount factor in our case was bound to be 0.8. However while varying other parameters, a $\gamma = 0.85$ gave feasible results and $\gamma = 0.8$ was infeasible.

C. Effect of Tuning Penalty Variables

The NLP has many parameters to be tuned. In this section, a few observations are discussed based on the Table 3, 4, 5:

- There is no significant difference between computation time while changing Q, q parameters except for R parameter.
- The errors across all the parameter changes are not significantly different except that we got the best error when $R = 0.6I$.
- There was no significant effect of varying the penalties like Q, q. However decreasing R has positive effects on imporving the accuracy and reducing final error. However the obtained trajectory of the best case of R does not follow closely to the desired trajectory based on visual inspection.
- In conclusion, we can justify this by saying that all the penalties are weighted almost equally but the CEC is very sensitive to R effect of varying penalty for input effort.

D. Effect of No Noise in Motion Model

When the motion model noise was removed, the trajectory was very smooth and there was no abrupt interventions in orientations.

VI. Discussion for GPI

A. Effect of discretization

A few points based on an attempt to implement GPI was listed below:

- The state space and control space size is directly proportional to the discretization resolution. A resolution of 30 resulted in 800000 elements and performing matrix operations on them was time-consuming.
- It was impossible to perform matrix inversion of this huge size. Hence using linear systems of equation method for policy evaluation is not feasible.
- Using GPU by converting numpy arrays to pytorch tensors would have resulted in a significant saving on the computation time.

TABLE I
EFFECT OF TIME HORIZON. γ = 0.85, $Q = 0.7I$, $R = I$,$q = 1$

S.No	Parameter T	Iteration Time(ms)	Final Error
1	5	Infeasible	
2	10	153.37	2189.47
3	12	186.37	2187.55
4	15	229.56	2190.01
5	20	Infeasible	

B. Expectations compared to CEC

I am listing below a few of my expectations of GPI compared to CEC.

- The final error is expected to be lesser than CEC as we have taken the motion noise into the formulation.
- The computation time for each iteration will be still higher than CEC even if we precompute the values.
- Adaptive discretization needs to be implemented if GPI is to be implemented in real-time.

VII. RESULTS FOR RECEDING HORIZON CEC

The effect of varying various parameters of the CEC algorithms are shown in this section. In all the tables the fixed constants are mentioned in the header. The best values are visualized and shown in the figures. Figure 8 shows the best hyperparameter-tuned receding horizon CEC for no obstacle case. Figure 9 shows the best hyperparameter-tuned receding horizon CEC for the case with obstacles case. Figure 9 shows the output if the motion model noise was removed.

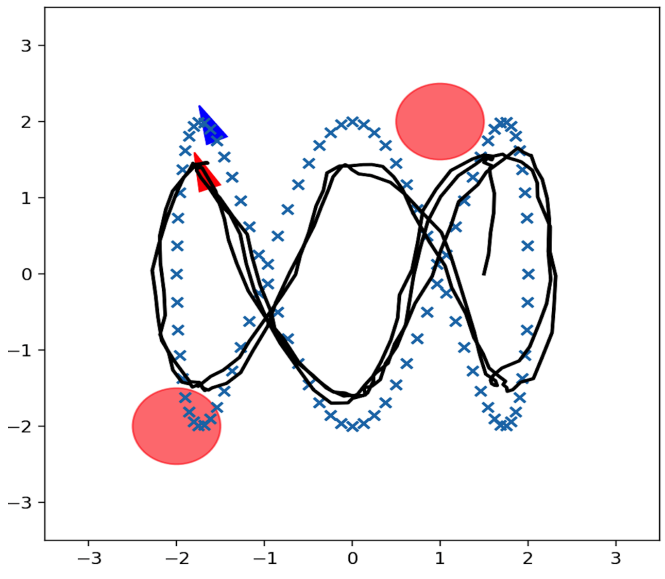

Fig. 2. Best Controller For Time Horizon case

TABLE II
EFFECT OF DISCOUNT FACTOR γ. $Q = 0.7I$, $R = I$,$q = 1$,$T = 15$

S.No	Parameter γ	Iteration Time(ms)	Final Error
1	0.99	235.51	2323.37
2	0.95	219.69	2240.95
3	0.9	303.31	2211.26
4	0.85	260.44	2190.01
5	0.8	270.62	2187.50

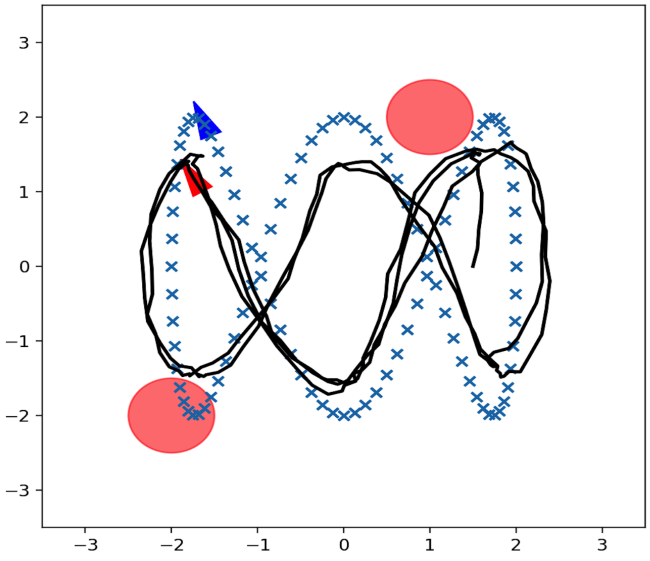

Fig. 3. Best Controller For Discount Factor case

TABLE III
EFFECT OF DISPLACEMENT PENALIZATION Q. γ = 0.85, $R = I$,$q = 1$,$T = 15$

S.No	Parameter Q	Iteration Time(ms)	Final Error
1	1I	Infeasible	
2	0.9I	Infeasible	
3	0.8I	Infeasible	
4	0.7I	239.35	2190.01
5	0.6I	254.14	2189.24

TABLE IV
EFFECT OF ORIENTATION PENALIZATION Q. γ = 0.85, $Q = 0.7I$, $R = I$,$T = 15$

S.No	Parameter q	Iteration Time(ms)	Final Error
1	2	248.00	2193.55
2	3	225.82	2197.88
3	5	Infeasible	
4	10	Infeasible	
5	1	240.00	2190.01

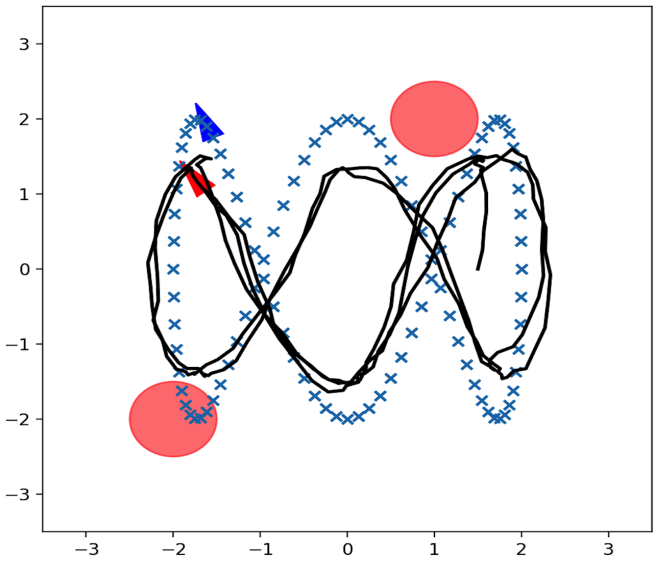

Fig. 4. Best Controller For Displacement Penalty case

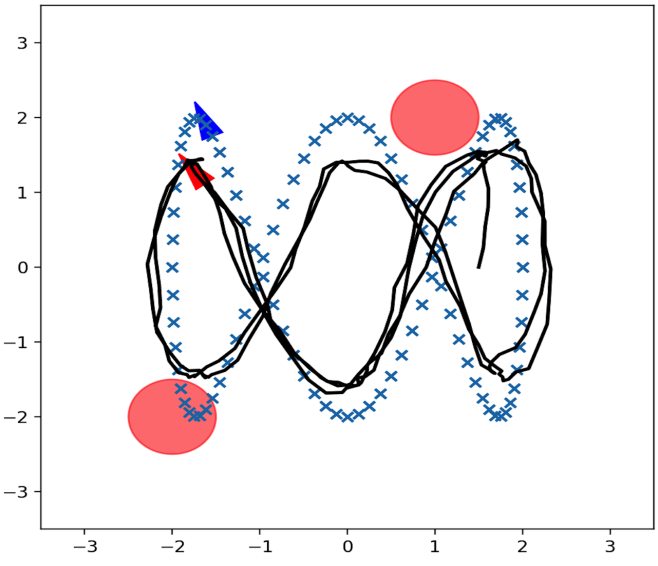

Fig. 5. Best Controller For q case

TABLE V
EFFECT OF EFFORT OF INPUT PENALIZATION R. $\gamma = 0.85$, $Q = 0.7I$, $q = 1$,$T = 15$

S.No	Parameter R	Iteration Time(ms)	Final Error
1	1.5I	232.37	2188.90
2	1I	239.81	2190.01
3	0.8I	Infeasible	
4	0.7I	191.10	1000.33
5	0.6I	159.45	429.81

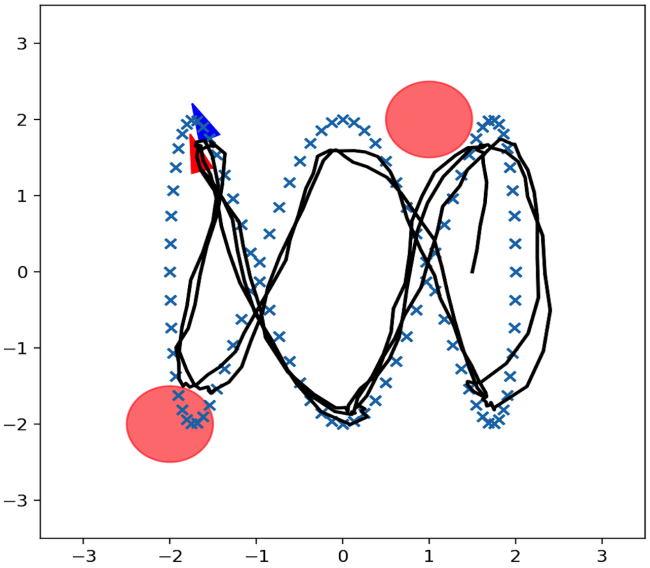

Fig. 6. Best Controller For Input Penalization case

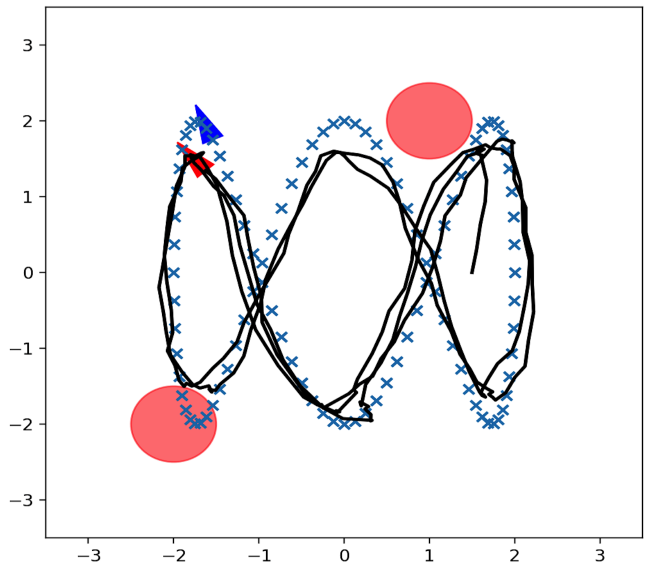

Fig. 7. Best Controller case

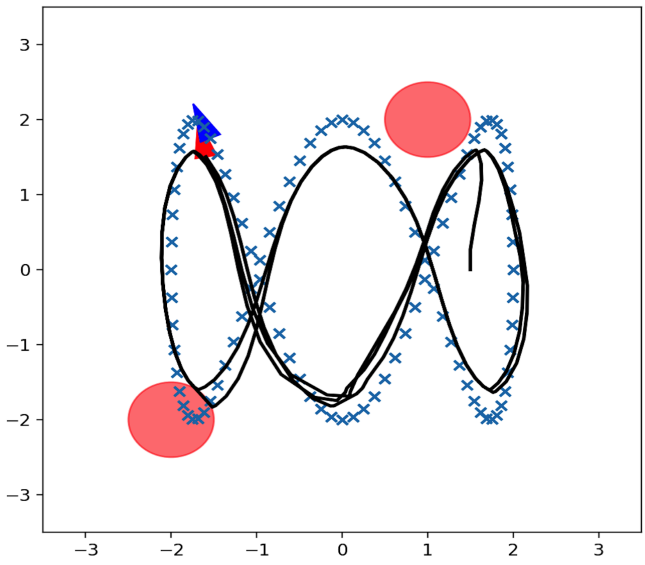

Fig. 8. Best Controller case with obstacles and no motion noise