Trajectory Tracking

Sambaran Ghosal
Department of Electrical and Computer Engineering
UC San Diego
La Jolla, USA
sghosal@ucsd.edu

Nikolay Atanasov
Department of Electrical and Computer Engineering
UC San Diego
La Jolla, USA
natanasov@ucsd.edu

***Abstract*—Trajectory tracking in robotics is a very important problem. Many tasks require that robots stick to a desired path while carrying out tasks. Safe trajectory tracking is one of the most important problems in modern day robotics research. In this project we try to implement a Certainty Equivalent Finite Horizon Control and Infinite Horizon Discounted Control problem to enable a given robot move along a specified path without hitting any obstacles.**

***Index Terms*—Trajectory tracking, Finite Horizon, Infinite Horizon, Value Iteration, CEC**

I. Introduction

A crucial problem in robotics is trajectory tracking. Trajectory tracking is the problem where a given robot or group of robots have to move along a specified path while carrying out tasks. Additionally it also has to avoid obstacles while following this path. In this project, we try to implement a **receding-horizon certainty equivalent control (CEC)** and a **generalized Policy Iteration (GPI)** to solve a tracking problem for a given differential drive robot and compare their performances.

II. Problem Formulation

We have a ground differential drive robot and a reference trajectory that we need to follow. Assuming the state of the robot as $x_t := (p_t, \theta_t)$ where $p_t \in R^2$ is the position of the robot and $\theta_t \in [-\pi, \pi)$ describes the orientation of the robot at any time t. Differential drive robots require a linear velocity in XY plane and an angular velocity about the Z axis to move around. We consider these as the control inputs to the robot $u_t := (v_t, \omega_t)$ where v_t is the linear velocity and ω_t is the angular velocity/yaw rate of the robot at a given time. Discretizing the continuous time differential drive kinematics using a time step $\Delta = 0.5$ gives us the discrete time kinematics of the differential drive robot as follows :

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

for $t = \{1, 2, 3, ...\}$. w_t is a gaussian noise to capture the inaccuracy of actuators to move to a specific exact position. The mean of the gaussian noise is considered to be 0 and the covariance of the noise is considered to be $\sigma = [0.04, 0.04, 0.004]$. The control input to the robot is restricted between 0 to 1 for v_t and -1 to 1 for ω_t. Hence the control space can be defined as $\mathcal{U} \in [0, 1] \times [-1, 1]$.

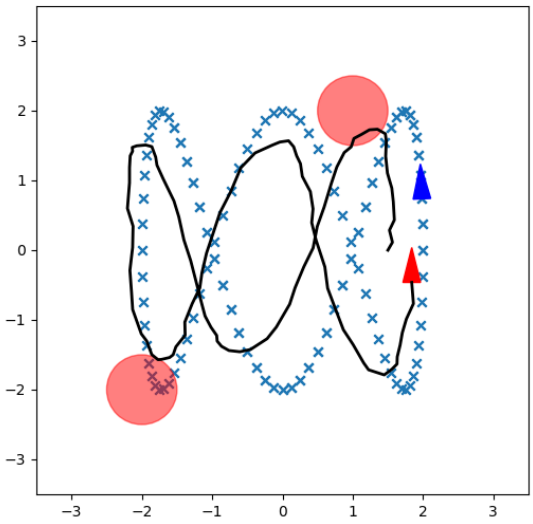

Fig. 1: The blue x's indicate the desired reference trajectory, the red circles are the obstacles and the black curve represents the robot trajectory produced by the controller

Given the above description of the differential drive robot, the objective is to track a given reference trajectory $r_t \in R^2$ and orientation trajectory $\alpha_t \in [-\pi, pi)$ while avoiding two obstacles present in the environment as circles at centeres (1,2) and (-2,-2) of radius 0.5. The free space for the tracking problem is hence described as $\mathcal{F} := [-3, 3]^2 - C_1 \cup C_2$ where C_1, C_2 are the two circles.

We define the error state of the robot as $e_t := (\tilde{p_t}, \tilde{\theta_t})$, where $\tilde{p_t} = p_t - r_t$ captures the error in tracking the position, and $\tilde{\theta_t} = \theta_t - \alpha_t$ captures the error in tracking the orientation. Note that $\tilde{\theta_t}$ has to lie between $[-\pi, \pi)$ too and hence we have to enforce angle wrap up on it. Using the error state, the differential drive model of the robot can be converted from actual state to error state kinematics as follows

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

We will represent this equation as $e_{t+1} = g(e_t, t, u_t, w_t)$ from now on. Given the error dynamics formulation, we can finally frame our objective problem for tracking as follows : Given the initial time τ and error at this time e_τ, the total sum of tracking error is given by

$$\begin{aligned} V(\tau, e_\tau) = E[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} (\tilde{p_t}^T Q \tilde{p_t} + q(1 - cos\tilde{\theta_t})^2 + u_t^T R u_t) | e_\tau] \\ \text{s.t. } e_{t+1} = g(t, e_t, u_t, w_t), w_t \sim \mathcal{N}(0, diag(\sigma^2)) \\ u_t = \pi(t, e_t) \in \mathcal{U} \\ \tilde{p_t} + r_t \in \mathcal{F} \end{aligned} \tag{3}$$

where $Q \in R^{2\times2}$ is a positive-definite matrix that penalizes tracking error in the position, $q > 0$ penalizes tracking error for orientation and $R \in R^{2\times2}$ is a positive-definite matrix that penalizes using extra control effort.

Our objective is to find the sequence of controls that minimize the tracking error defined above i.e.

$$\begin{aligned} V^*(\tau, e_\tau) = \min_{\pi} E[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} (\tilde{p_t}^T Q \tilde{p_t} + \\ q(1 - cos\tilde{\theta_t})^2 + u_t^T R u_t) | e_\tau] \\ \text{s.t. } e_{t+1} = g(t, e_t, u_t, w_t), w_t \sim \mathcal{N}(0, diag(\sigma^2)) \\ u_t = \pi(t, e_t) \in \mathcal{U} \\ \tilde{p_t} + r_t \in \mathcal{F} \end{aligned} \tag{4}$$

The optimal policy is then π and we can access the policy at any given time instant and it will produce the best possible control input for the robot to track the reference trajectory.

III. Technical Approach

In this section, we will discuss two methods to solve the minimization problem for the tracking error described in the above section.

A. Receding Horizon Certainty Equivalent Control

Receding Horizon Certainty Equivalent Control(CEC) is a suboptimal control scheme that applies at each time step the control if the noise present in the motion model of the robot is fixed at the mean of the distribution, which for our case is 0 since we consider a zero mean gaussian noise as described in previous section. The fixing of noise converts the original stochastic problem to a deterministic optimal control problem which is slightly easier to solve. In addition, instead of solving the infinite horizon problem for the tracking error, CEC approximates the infinite horizon problem by a finite-horizon problem and then solves this finite-horizon deterministic optimal control problem repeatedly at each time step to account for the noise.

The CEC problem henc can be formulated as follows : at time τ with current error e_τ, if we consider a time horizon for the finite-horizon tracking error of T, the objective is to find the sequence of control actions $u_\tau, u_{\tau+1}, u_{\tau+2}, ..., u_{\tau+T-1}$ that minimizes the specified time horizon total tracking error as follows

$$\begin{aligned} V^*(\tau, e_\tau) = \min_{u_\tau, .., u_{\tau+T-1}} q(e_{\tau+T}) + \\ \sum_{t=\tau}^{t+T-1} (\tilde{p_t}^T Q \tilde{p_t} + q(1 - cos\tilde{\theta_t})^2 + u_t^T R u_t) \\ \text{s.t.} \quad e_{t+1} = g(t, e_t, u_t, 0) \\ u_t \in \mathcal{U}, \tilde{p_t} + r_t \in \mathcal{F} \end{aligned} \tag{5}$$

where $q(e_{\tau+T})$ is a terminal cost associated with the error at end of the current time horizon. $q(e_{\tau+T})$ can be a suitably chosen function that can capture the error from this time on wards till ∞. The problem is now a Non Linear Optimization program problem of the form

$$\begin{aligned} \min_{U,E} c(U, E) \\ \text{s.t. } U_{lb} \leq U \leq U_{ub} \\ h_{lb} \leq h(U, E) \leq h_{ub} \end{aligned} \tag{6}$$

U can be defined as an array of the sequence of the controls $[u_\tau^T, u_{\tau+1}^T, u_{\tau+2}^T, ..., u_{\tau+T-1}^T]^T$, E can be defined as an array of the error at the time steps $[e_\tau^T, e_{\tau+1}^T, ..., e_{\tau+T}^T]^T$. U_{lb}, U_{hb} define the lower and upper bounds of the control actions which for our case becomes $U_{lb} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ and $U_{hb} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. The function $c(U, E)$ is the total tracking error defined in Equation (5)

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

$h(U, E)$ express all other constraints that may be required for our problem. There can be multiple constraints of the form $h_{lb} \leq h(U, E) \leq h_{ub}$. For the CEC control, one class of h function is to describe the error dynamics equation $e_{t+1} = g(t, e_t, u_t, 0)$. Second class of h function describes the bounds on the error function defined as $\tilde{p_t} + r_t \in \mathcal{F}$

After we have formulated all the desired functions $c(U, E)$, h constraints, we feed this to a Non Linear Programming Language. For our project, we use the CasADi Solver to solve the above non linear minimization problem and get the control sequence with the following elements of the optimization problem

- The optimization variables will be U, E.
- constraints are the h equations described above as one class of constraints being the error dynamics equation $e_{t+1} = g(e_t, t, u_t, 0)$ and the other being the bounds of the error function as $\tilde{p_t} + r_t \in \mathcal{F}$.
- The objective of the NLP program is to minimize $c(U, E)$ described in Equation (7).

We return the value of u_τ obtained from CasADi solver and make the robot execute this to get to the next state, then we add some noise to the next state to account for the original

noise, and we call the CasADi solver to solve the optimization problem from this time step again. This process is repeated forever. Hence this is an Online Planner.

B. Generalized Policy Iteration

In this section we try to solve the infinite horizon stochastic optimal control problem for the total tracking error directly using Value Iteration algorithm. To use the Value Iteration algorithm, we discretize the state space of ours into discrete grids. Our reference trajectory is a periodic motion with a period of 100 time steps and hence we consider that time is a set of 100 points of increment 0.5 ranging from 0-50. The x, y coordinate ranges from -3 to 3, hence we consider an adaptive grid for the error in position of the robot as follows : $e_x, e_y = [-3, -0.5, -0.25, -0.15, -0.05, 0.05, 0.15, 0.25, 0.5, 3]$. Similarly the orientation error of the robot is discretized as follows : $\tilde{\theta} = [-180^o, -90^o, -45^o, -27^o, -9^o, 9^o, 27^o, 45^o, 90^o, 180^o]$. Hence, we have discretization $n_x = n_y = 9$, $n_t = 100$, $n_\theta = 9$ for the state space and hence the size of our state space becomes $|\mathcal{X}| = 9 \times 9 \times 9 \times 100 = 71000$.

The space of linear velocity is discretized as follows : $v_t = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1$. The angular velocity is discretized as follows : $\omega_t = [-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1]$. Hence the control space has been discretized with $n_v = 10, n_\omega = 10$ and hence size of control space $|\mathcal{U}| = 10 \times 10 = 100$.

Since the GPI attempts to solve the tracking error problem in presence of noise, effectively this means that from a given error, a control sequence can land us up in several possible states. The GPI algorithm computes the value function of a state using the expected value function of all possible transitions from the current state, hence we need to find the probability of transition to possible states. Since our noise is a gaussian random variable with covariance diag(σ^2), the state transition matrix can be found as follows :

- The expected next state is $e_{t+1} = g(t, e_t, u_t, 0)$.
- Due to noise, we will possibly land at some area around e_{t+1}. We select the closest 5 points near this point in our discretized state space grid.
- We find the likelihood of getting these points x_i for i = 1,2,3,4,5 by assuming a gaussian noise with mean $\mu = g(t, e_t, u_t, 0)$ and covariance $\Sigma = \text{diag}(0.04, 0.04, 0.004)$ as
$$p(x_i) = \frac{1}{\sqrt{(2\pi)^3|\Sigma|}} e^{-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)} \tag{8}$$
- The transition probability is obtained by normalizing the likelihood of the above points
$$p = [\frac{p(x_1)}{\sum_{i=1}^5 p(x_i)}, \frac{p(x_2)}{\sum_{i=1}^5 p(x_i)}, \frac{p(x_3)}{\sum_{i=1}^5 p(x_i)}, \frac{p(x_4)}{\sum_{i=1}^5 p(x_i)}, \frac{p(x_5)}{\sum_{i=1}^5 p(x_i)}] \tag{9}$$

The above steps can be represented pictorially as shown in Figure (2)

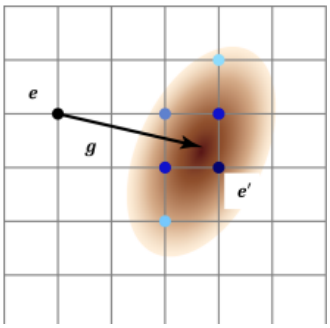

Fig. 2: Transition probabilities using gaussian noise

The transition probability for our discretized state space is going to be of dimension $R^{|\mathcal{X}|\times|\mathcal{U}|\times|\mathcal{X}|}$, which is an enormous matrix. But most of the elements of this matrix are 0's. Hence we store this matrix in disk as a sparse matrix. This transition matrix serves as our motion model of the Markov Decision Process(MDP) for the tracking error problem that we aim to solve using the Value Iteration algorithm.

The step cost for our MDP process is defined as

$$l(x_t, u_t) = \tilde{p_t}^T Q \tilde{p_t} + q(1 - cos\tilde{\theta_t})^2 + u_t^T R u_t) + \text{penalty} \tag{10}$$

where penalty is decided as follows

$$\text{penalty} := \begin{cases} 25 & ||\tilde{p_t} + r_t||_2^2 < 0.55^2 \\ 0 & ||\tilde{p_t} + r_t||_2^2 > 0.55^2 \end{cases} \tag{11}$$

which captures the condition whether the trajectory falls inside any of the two circles or not. We inflate the radius of the circles a bit to account for noise. The step cost is stored as a matrix L of size $R^{|\mathcal{X}|\times|\mathcal{U}|}$.

We now have all the elements for the MDP problem associated with the tracking error optimal control problem. We can now implement a Value Iteration algorithm to obtain the optimal policy and value function that would minimize the long term expected tracking error. The value iteration algorithm is described below

Algorithm 1 Value Iteration

Require: L step cost matrix, P stochastic transition matrix

$V_0 = R^{|\mathcal{X}|} \leftarrow 0, \gamma \leftarrow 0.95$
for k = 0,1,2,.... **do**
  $\mathcal{Q} \leftarrow L + \gamma P V_k$
  $V_{k+1} \leftarrow \min_{u \in \mathcal{U}} \mathcal{Q}$
  $\pi \leftarrow \text{argmin}_{u \in \mathcal{U}} \mathcal{Q}$
  if $||V_{k+1} - V_k||_2 \leq 1e-9$ **then**
    return π
  end if
end for

After the Value Iteration converges, we get the optimal policy function and the optimal value function. We can then

just call the optimal policy from the current time and error and we get the optimal policy. Also since this was modelled as a infinite-horizon problem, we don't need to call the planner again and again from different times. This is an offline planner.

In the next section, we show the results of using the CEC controller, the GPI controller and try to show the effect of changing the value of parameters like Q,q,R.

IV. Results

In this section we present the results obtained for the trajectory tracking problem using both the Receding Time Horizon CEC controller and the infinite horizon Generalized Policy Iteration Controller. We also try to study the effects of the cost parameters Q,q and R.

In the plots presented, the blue curves represent the trajectory executed by the robot using the optimal policy computed using either the CEC or GPI controller, the red x's indicate the desired reference trajectory and the red circles represent the circular obstacles.

A. Receding Time Horizon Certainty Equivalent Control

As described in previous section, the CEC controller solves the infinite time horizon stochastic optimal control tracking minimization problem by converting into a finite time horizon deterministic optimal control problem. The key aspects of the CEC controller are the tracking hyper parameters Q, q and R as well as the time horizon for the CEC controller.

1) Effect of Q: The parameter Q in the tracking error cost penalizes the controller for error in the position tracking. It is expected that as we increase the value of the parameter Q, we see an improved performance in tracking the reference position. To test this claim, we keep the parameter q = 30 and R = 5 fixed and change the value of Q and see the error in tracking. The results are shown in Figure (3).

From the graphs, we observer that the error in tracking the reference trajectory was the minimum for Q = 2, slightly better with Q = 5 and the best tracking performance was achieved when Q = 10. Hence we have a support to our claim that using a higher value of Q achieves to better tracking performance. But during running the CEC control, I ran into a problem that if the value of Q was increased more, the CasADi solver threw the **Infeasible Problem Detected**. My reasoning is that when the reference trajectory is close to the obstacles and the robot is a little bit far from the reference trajectory, since the emphasis is high on minimizing the position tracking error, the controller is may end up putting the robot almost inside the obstacle and this leads to the infeasible problem error.

2) Effect of q: The parameter q puts emphasis on tracking the reference orientation. If the orientation is not tracked properly, even if our Q value is high to ensure proper position tracking, we will not be able to get good tracking. We present results of varying the value of q by keeping Q = 5 and R = 5 fixed and see the tracking performance. The plots are shown in Figure (4).

We observe from the plots that as expected, increasing the value of q improved the tracking performance, but the improvement was not as much significant as we had observed when we changed the values of Q. Hence we can conclude that although a higher value of q is better, there is not much significant improvement in the tracking performance.

TABLE I: Performance using CEC controller

Q	q	R	T	Avg Time per control(ms)	Tracking Error
10	30	5	5	31	62
10	30	5	15	86	74.22
10	30	5	25	121	112.86
2	30	5	15	72	105
5	30	5	15	72	71
5	5	2	15	74	58
5	1	5	15	72	81
5	5	5	15	72	79
5	15	5	15	72	75

3) Effect of R: The parameter R penalizes the controller for putting in too much control effort. While it is useful to not put in too much effort since it may drain the robot battery power, it is also equally important to not penalize it too much otherwise the controller will not put enough control input to reduce the tracking error. We see the effect of different values of R with Q = 5 and q = 5. The plots are shown in Figure (5).

From the plots shown, we can see that we get the best performance using R = 2 and the performance keeps decreasing as we increase the value of R. Hence we can conclude that a slightly higher value of R is okay to use to penalize the use of excessive controls, but not too high otherwise the tracking performance will decrease.

4) Effect of time horizon T: Since the CEC controller converts the infinite time horizon optimal control problem to a finite time horizon optimal control problem, choosing the time horizon for the optimal control problem is important. Choosing a longer time horizon will produce a sequence of controls that minimize the tracking error over a longer time and hence perhaps produce a better performance but it will take computationally longer time. Whereas a CEC control with lower time horizon may not produce the best tracking performance on the long run, it is computationally less expensive. The effect of different time horizon for the CEC controller is presented in Figure (6).

We observe that a time horizon of 5 and 15 work better than a time horizon of 25 and that with time horizon of 25 we see a strange behaviour in the bottom sinusoid where the robot shows a 360 donut kind of behaviour. Also, the average time taken for the CEC controller to produce the next control sequence is 31, 72 and 121 milliseconds respectively for $T = 5, 15, 25$. Hence we can conclude that it is not necessary to have a very long time horizon for the CEC controller and a reasonable time horizon produces satisfactory tracking performance with reasonable computation time.

A summary of the above results is presented in Table I.

B. Generalized Policy Iteration

We can directly solve the infinite horizon stochastic optimal control problem by running a Generalized Policy Iteration algorithm to find the optimal policy and optimal value function.

TABLE II: Performance using Generalized Policy Iteration controller

Q	q	R	Tracking Error
10	5	2	276
30	5	2	226
60	5	2	136
60	15	2	187
60	30	2	126
60	30	5	103
60	30	10	212
75	30	1	69

We implement the Value Iteration algorithm as part of the Generalized Policy Iteration. As we have shown in the CEC controller, we try to see the effects on tracking performance by changing the Q, q, R values.

1) Effect of Q: The effect of changing Q when using Value Iteration is shown in Figure (7). As with the CEC controller, we observe that increasing Q improves the tracking performance. With Q = 10, we had a tracking error of 275, with Q = 30 the tracking error was 225 and with Q = 60 it was 136.

2) Effect of q: The effect of changing q when using Value Iteration is shown in Figure (8). We observe that with q = 5, we have tracking error of 136, with q = 15 a tracking error of 187 and with q = 30, a tracking error of 125. So the performance did improve as we increased q to 30 but we also see that there was a slight dip in performance. So we can conclude that in general increasing q will improve tracking performance but we may need to do some fine tuning.

3) Effect of R: The effect of changing R is presented in Figure (9). As we have seen previously, R penalizes for using extra control effort. While this may be benefitial in saving energy, a very high value of R will unnecessarily penalize the controller and will lead to bad tracking performance. This is the behaviour we see in Figure (9). We see an improved performance from using R = 2 to R = 5. But then the performance was drastically bad for R = 10 because the controller was penalized unnecessarily.

A summary of tracking performance using different parameters using the GPI controller is presented in Table II

Based on the above trends, I did some fine tuning and got some values that worked well. The final performance with these parameters are shown in Figure (10).

From the above experiments using both the CEC and GPI controller, we present the following conclusions :

- Since the CEC controller is continuous state and control space whereas the GPI implemented in this project was over discrete state and control space, the CEC controller was able to perform better than the GPI controller. But it can be claimed that if a GPI controller was implemented using value function approximation methods that could be implemented on continuous control and state space, the performance would be almost similar.
- The major challenges faced in implementing a discretized state space and control space GPI algorithm was space and memory. To achieve significantly well tracking performance, the state space and control space needs to be discretized into fine grids, but this increases the size of the state and the control space, hence the size of the transition probability matrix and the step cost matrix increase significantly. For this project, to create the transition matrix alone took like 40 minutes even after implementing MultiThread Processing. The transition probability matrix cannot be created as a numpy matrix because it requires more than 4Tb memory if we were to save floating point numbers as numpy arrays of the size of the transition matrix. Hence it was saved as a sparse matrix.
- The CEC controller can be implemented as an online controller, where we set up the CasADi controller with the step cost definition, the optimization variables and constraints, and at each time we feed in the current error and the controller will output the next command to execute.
- The GPI controller needs to be implemented as an offline controller where we decide the state space and control space discretization, the number of children to consider for a given transition, then store the transition matrix in disk and then solve the infinite-horizon discounted optimal control problem and store the recieved the optimal policy function in the disk. We can then call this at each step of the robot motion and reload the next control action to be executed.

V. Conclusion and Future Work

In this project, we solved a trajectory tracking problem by converting it to a Markov Decision Process and get the optimal value and policy functions using the Generalized Policy Iteration and a CEC controller.

In the future, we can try to use the Generalized Policy Iteration by using Functional Approximation to the value function and not having to discretize the state space and control space. We can use a Gaussian RBF functional approximation to get the value function for any possible state and control, and we can train the value function approximator by the data we have and using these, we can find the optimal policy. We can also frame this tracking problem as full Reinforcement Learning problem such as DQN, DDPG, SAC to get the optimal policy.

VI. Acknowledgement

I would like to thank Professor Nikolay Atanasov for his help and suggestions throughout this project. His assistance helped me overcome some crucial problems encountered during the project.

References

[1] Nikolay Atanasov, "https://natanaso.github.io/ece276b/ref/ECE276B_11_BellmanEquatio
[2] Nikolay Atanasov, "https://natanaso.github.io/ece276b/ref/ECE276B_3_MDP.pdf"
[3] "https://web.casadi.org/docs/"
[4] "https://sparse.pydata.org/en/stable/"

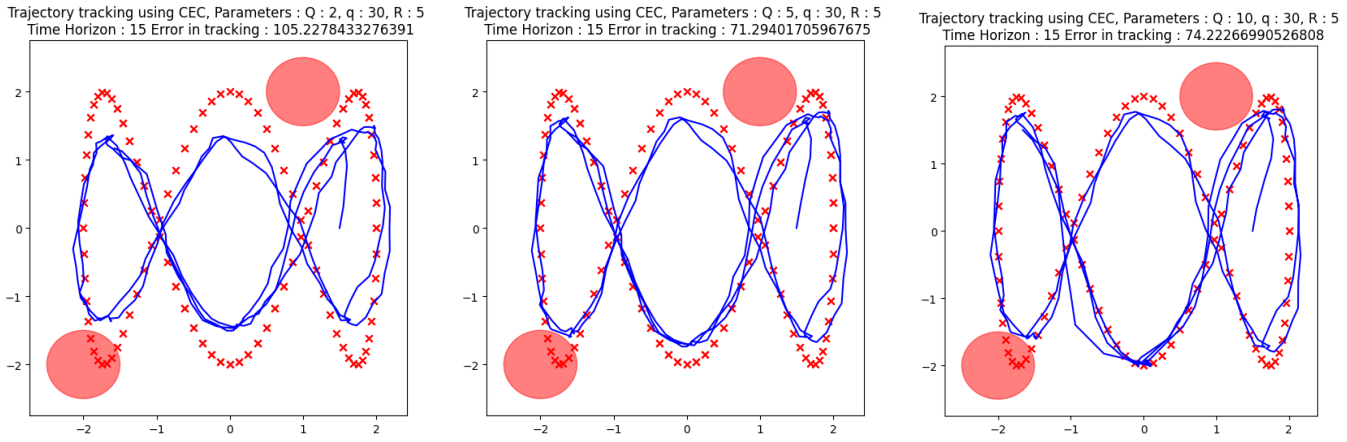

Fig. 3: Effect of Q on tracking performance using CEC controller

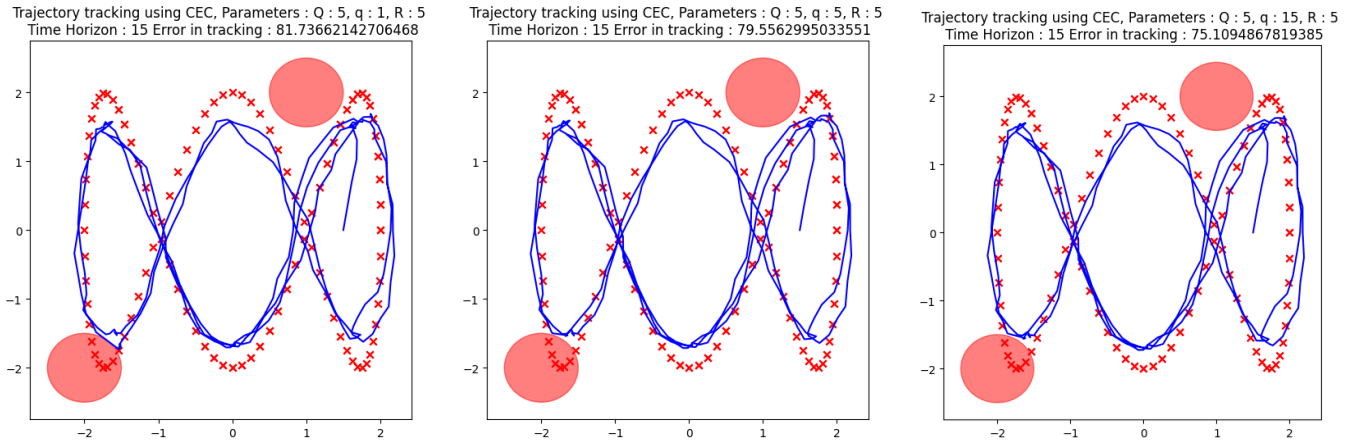

Fig. 4: Effect of q on tracking performance using CEC controller

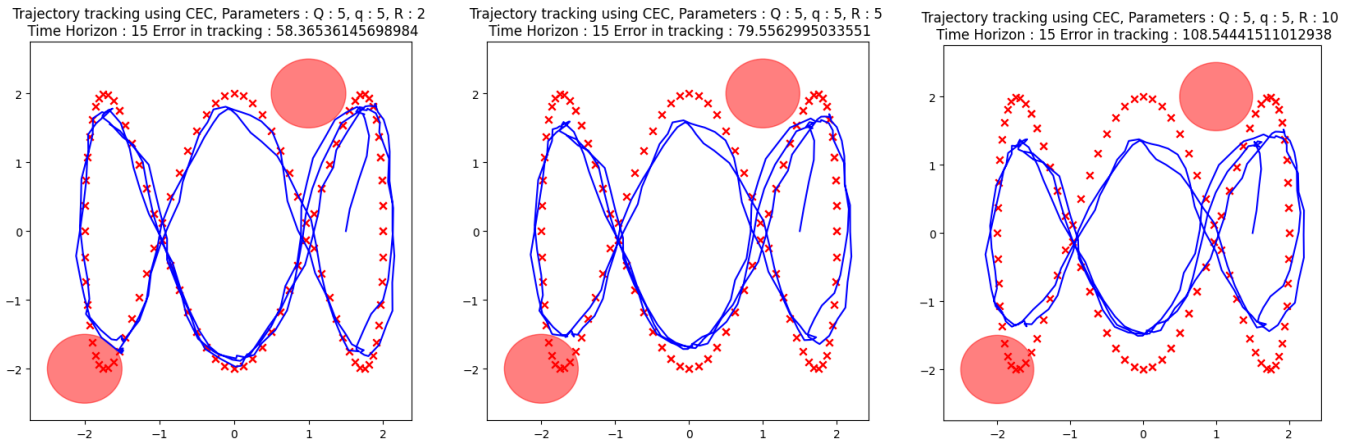

Fig. 5: Effect of R on tracking performance using CEC controller

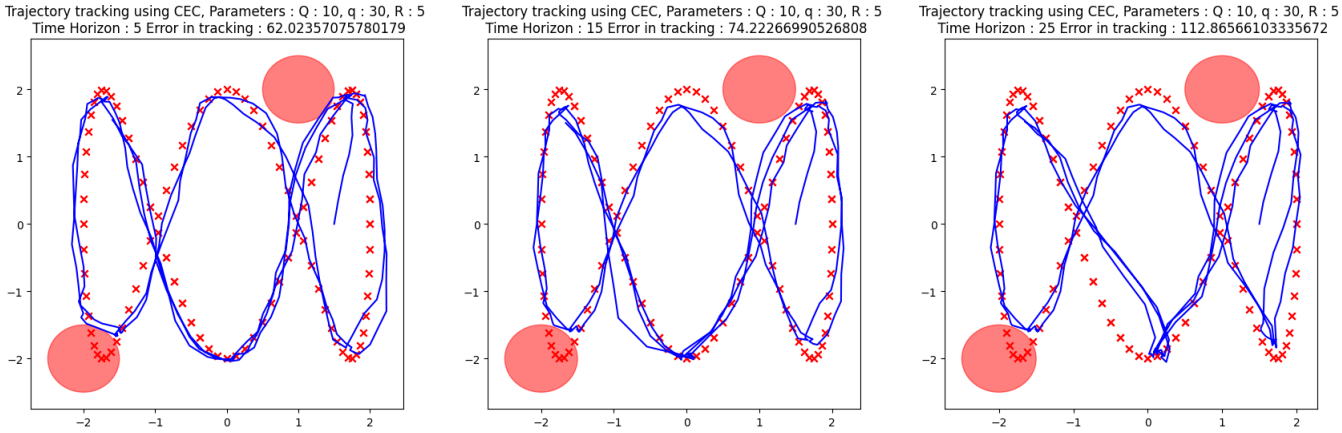

Fig. 6: Effect of time horizon T on tracking performance using CEC controller

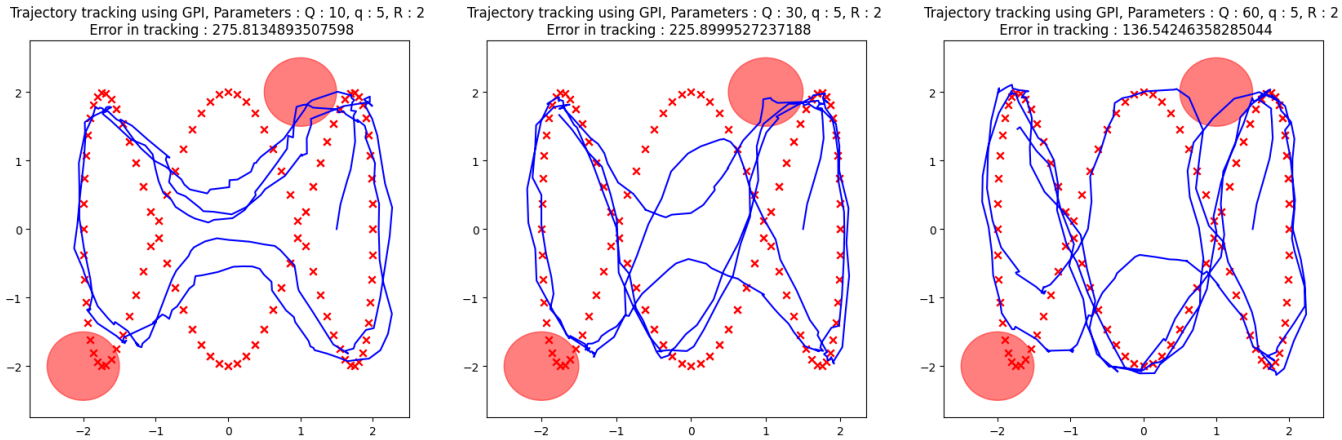

Fig. 7: Effect of Q on tracking performance using GPI controller

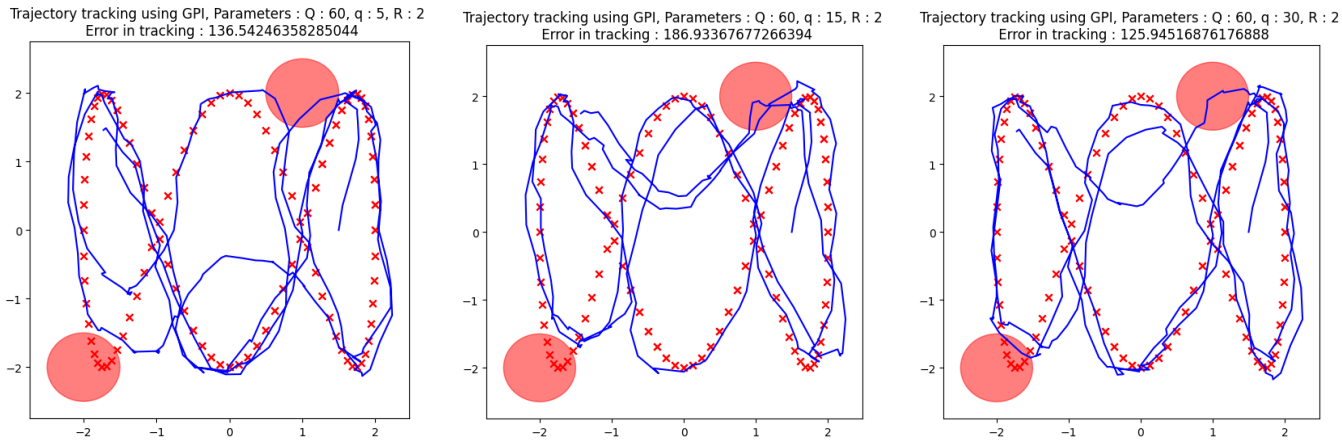

Fig. 8: Effect of q on tracking performance using GPI controller

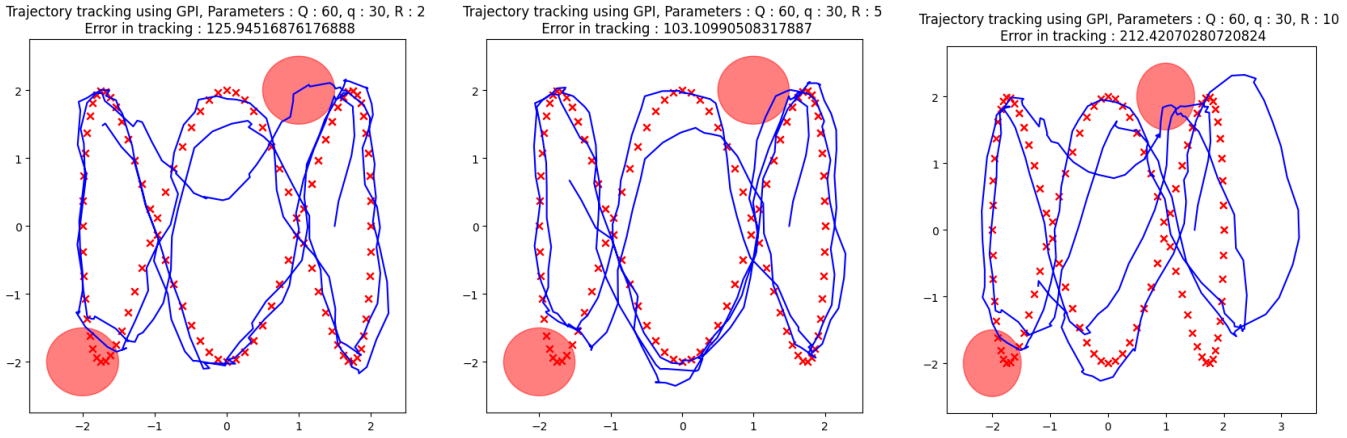

Fig. 9: Effect of R on tracking performance using GPI controller

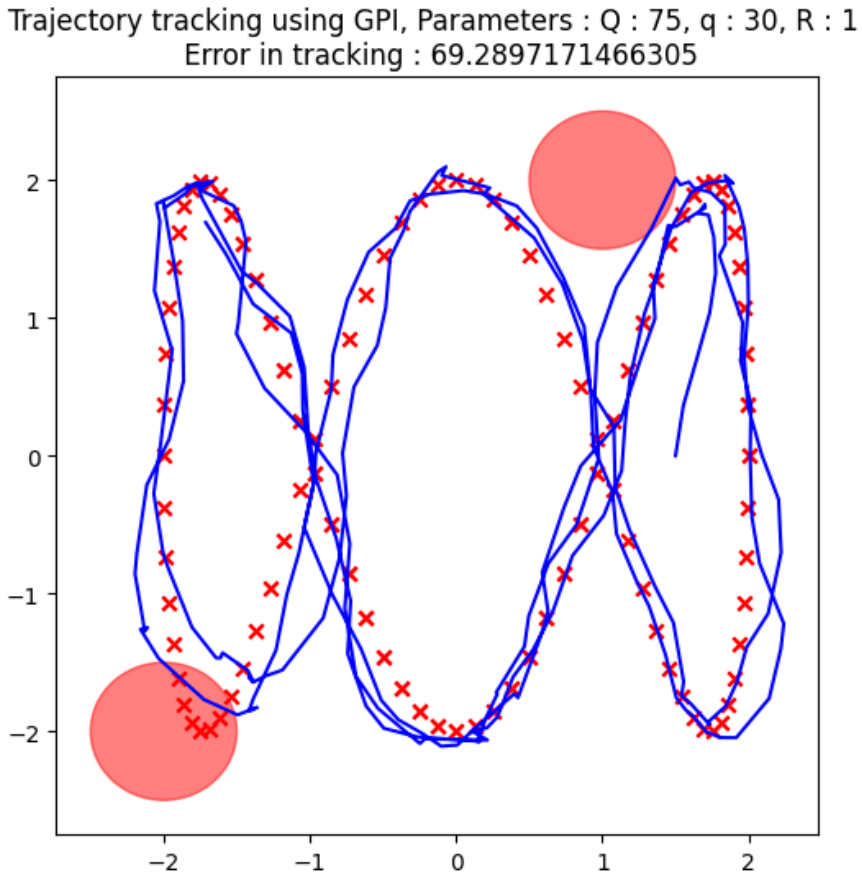

Fig. 10: Tracking performance using GPI controller