2D Grid Trajectory Tracking in a Nutshell: CEC and GPI Approaches

Zhuoqun Chen
Dept. Electrical and Computer Engineering
UC San Diego
La Jolla, USA
zhc057@ucsd.edu

***Abstract*—This report focuses on solving a reference trajectory tracking problem on 2D grid with obstacles. By formulating the problem into an infinite-horizon stochastic optimal control problem, we solved it with two suboptimal schemes: Receding-horizon Certainty Equivalent Control (CEC) and Generalized Policy Iteration (GPI). CEC approximates the problem by repeatedly formulating a finite-horizon Deterministic Optimal Control Problem (DOC) by fixing the noise at its expected value and solve it with Non-Linear Program (NLP) solvers like *CasADi*. GPI, however, solves the original problem by discretizing the continuous state and control spaces and designing a large state transition lookup table. Both schemes give promising results with small overall tracking errors in our experiments where reference motion is selected as a *Lissajous Curve* partially blocked by two circular obstacles. We then analyzed and compared the performance of these two methods in terms of computational complexity and total tracking errror.**

***Index Terms*—Trajectory Tracking, Infinite-horizon Stochastic Optimal Control, Receding-horizon Certainty Equivalent Control (CEC), Generalized Policy Iteration (GPI)**

I. Introduction

Trajectory Tracking is a fundamental problem in control systems where the goal is to control a robot to follow a desired trajectory or path. This problem arises in various fields such as robotics, autonomous vehicles, and aerospace systems. To tackle trajectory tracking problems, researchers have developed a variety of control algorithms, two of which are Receding-Horizon Certainty Equivalent Control (CEC) and Generalized Policy Iteration (GPI). In general, both algorithms will offer effective solutions and the choice of which algorithm to be deployed in real-world applications depends on the underlying model of the dynamic systems, the type of spefic tasks, computational resources, and the level of uncertainty in the system dynamics. The effectiveness of both algorithms can be validated in some simplified tasks, for example, in Figure 1, we can model the uncertainty in a 2D grid environment as random *Gaussian Noise* with zero mean and covariance $\mathcal{N}\left(\mathbf{0}, \mathrm{diag}(\boldsymbol{\sigma})^2\right)$ and let a differential-drive robot to track a *Lissajous* curve by using these algorithms.

In this report, we mainly focus on implementing CEC and GPI algorithms in a 2D reference trajectory tracking task described in Figure 1 and comparing their performance on various metrics. The organization of this report is as following: We formulate the trajectory tracking problem in Section II, then we present the implementation details in Section III, then we did experiments to show the effect of different hyper-parameters on our implemented algorithms in Section IV. At the end of the report, we also documented some experiment debugging logs while finishing this project.

II. Problem Formulation

Consider a 2D differential-drive robot with discrete-time Kinematic model in Euler discretiza-

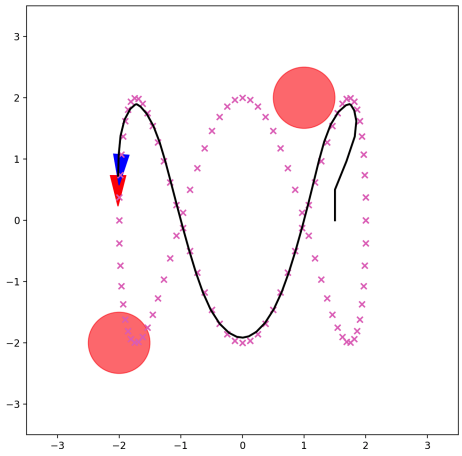

Fig. 1. Trajectory Following Task Example: The goal is to compute a control policy for a differential-drive robot (red triangle) to track a 2D reference trajectory (pink crosses). The trajectory is a periodic *Lissajous* curve. There are two circular obstacles (red circles) that partially block the reference trajectory thus the control policy will also need to be designed to avoid them.

tion form with time step $\Delta > 0$:

$$\begin{aligned}\mathbf{x}_{t+1} &= \begin{bmatrix} \mathbf{p}_{t+1} \\ \theta_{t+1} \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} \mathbf{p}_t \\ \theta_t \end{bmatrix}}_{\mathbf{x}_t} + \Delta \underbrace{\begin{bmatrix} \cos(\theta_t) & 0 \\ \sin(\theta_t) & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{G}(\mathbf{x}_t)} \underbrace{\begin{bmatrix} v_t \\ \omega_t \end{bmatrix}}_{\mathbf{u}_t} + \mathbf{w}_t \end{aligned} \tag{1}$$

where $\mathbf{w}_t \in \mathbb{R}^3$ models the motion noise with Gaussian distribution $\mathcal{N}\left(\mathbf{0}, \mathrm{diag}(\sigma)^2\right)$ with standard deviation $\sigma = [0.04, 0.04, 0.004] \in \mathbb{R}^3$. The robot is constrained to move inside a bounded configuration space $\mathcal{C} = [-3, 3]^2$ with two circular obstacles $\mathcal{C}_1$ and $\mathcal{C}_2$ with radius 0.5 centered at $\mathcal{O}_1 = (-2, -2)$ and $\mathcal{O}_2 = (1, 2)$ respectively. Thus the actual free space the robot can move is $\mathcal{F} := \mathcal{C} \backslash (\mathcal{C}_1 \cup \mathcal{C}_2)$. The control input $\mathbf{u}_t = [v_t, \omega_t]$ of the vehicle is also bounded inside control space $\mathcal{U} := [0, 1] \times [-1, 1]$, where v_t and ω_t are the linear and angular velocities respectively.

The reference trajectory to be followed is selected as a periodic *Lissajous* curve. The curve can be parameterized as a sequence of position trajectory $\mathbf{r}_t \in \mathbb{R}^2$ and orientation trajectory $\alpha_t \in [-\pi, \pi)$. The trajectory is partially blocked by obstacle $\mathcal{C}_1$ and is very close to $\mathcal{C}_2$. Because we mainly focus on how to reduce the gap between the computed path with reference path at each time step, it is convenient to define the error state $\mathbf{e}_t := \left(\tilde{\mathbf{p}}_t, \tilde{\theta}_t\right)$, where $\tilde{\mathbf{p}}_t := \mathbf{p}_t - \mathbf{r}_t$ and $\tilde{\theta}_t := \theta_t - \alpha_t$ measure the position and orientation deviation from the reference trajectory, and formulate an error state transition function as:

$$\begin{aligned}\mathbf{e}_{t+1} &= \begin{bmatrix} \tilde{\mathbf{p}}_{t+1} \\ \tilde{\theta}_{t+1} \end{bmatrix} = g\left(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{w}_t\right) \\ &= \underbrace{\begin{bmatrix} \tilde{\mathbf{p}}_t \\ \tilde{\theta}_t \end{bmatrix}}_{\mathbf{e}_t} + \underbrace{\begin{bmatrix} \Delta\cos\left(\tilde{\theta}_t + \alpha_t\right) & 0 \\ \Delta\sin\left(\tilde{\theta}_t + \alpha_t\right) & 0 \\ 0 & \Delta \end{bmatrix}}_{\tilde{\mathbf{G}}(\mathbf{e}_t)} \underbrace{\begin{bmatrix} v_t \\ \omega_t \end{bmatrix}}_{\mathbf{u}_t} \\ &\quad + \begin{bmatrix} \mathbf{r}_t - \mathbf{r}_{t+1} \\ \alpha_t - \alpha_{t+1} \end{bmatrix} + \mathbf{w}_t \end{aligned} \tag{2}$$

We then formulate the trajectory tracking with initial time τ and initial tracking error $\mathbf{e}$ as a discounted infinite-horizon stochastic optimal control problem:

$$\begin{aligned} V^*(\tau, \mathbf{e}) = \min_{\pi} \mathbb{E}\Bigg[&\sum_{t=\tau}^{\infty} \gamma^{t-\tau} \left(\tilde{\mathbf{p}}_t^\top \mathbf{Q} \tilde{\mathbf{p}}_t \right. \\ &\left. + q\left(1 - \cos\left(\tilde{\theta}_t\right)\right)^2 + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t\right) \mid \mathbf{e}_\tau = \mathbf{e}\Bigg] \\ \text{s.t.} \quad & \mathbf{e}_{t+1} = g\left(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{w}_t\right), \\ & \mathbf{u}_t = \pi\left(t, \mathbf{e}_t\right) \in \mathcal{U} \\ & \tilde{\mathbf{p}}_t + \mathbf{r}_t \in \mathcal{F} \end{aligned} \tag{3}$$

where $\mathbf{Q} \in \mathbb{R}^{2\times 2}$ in eq. (2) is a symmetric positive-definite matrix defining the stage cost for deviating from the reference position trajectory $\mathbf{r}_t$, $q > 0$ is a scalar defining the stage cost for deviating from the reference orientation trajectory α_t, $\mathbf{R} \in \mathbb{R}^{2\times 2}$ is a symmetric positive-definite matrix defining the

stage cost for using excessive control effort, and $\mathbf{w}_t \sim \mathcal{N}\left(\mathbf{0}, \mathrm{diag}(\boldsymbol{\sigma})^2\right), \quad t = \tau, \tau+1, \ldots$ is the Gaussian noise the same as in eq. (1).

To solve this infinite-horizon stochastic optimal control problem, CEC and GPI algorithms can be applied by adopting different strategies and approximations at different levels. More details are presented at Section III. In later sections, we focus more on the specific approaches to solve the above trajectory tracking problem with different algorithms.

III. Technical Approach

In this section, we discuss two specific methods developed that can be used to solve eq. (3).

A. Receding-horizon Certainty Equivalent Control (CEC)

The original formulation eq. (3) modeled the uncertainty in the system as a stochastic process. However, Receding-horizon Certainty Equivalent Control (CEC) scheme uses a simple idea, i.e., to treat it to be a fixed value of expectation by reformulating the equation to be:

$$
\begin{aligned}
V^*(\tau, \mathbf{e}) \approx \min_{\mathbf{u}_\tau, \ldots, \mathbf{u}_{\tau+T-1}} & \mathrm{q}\left(\mathbf{e}_{\tau+T}\right) + \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} \left(\tilde{\mathbf{p}}_t^\top \mathbf{Q} \tilde{\mathbf{p}}_t + q\left(1 - \cos\left(\tilde{\theta}_t\right)\right)^2 + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t \right) \\
\text{s.t.} \quad & \mathbf{e}_{t+1} = g\left(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{0}\right), \quad t = \tau, \ldots, \tau+T-1 \\
& \mathbf{u}_t \in \mathcal{U} \\
& \tilde{\mathbf{p}}_t + \mathbf{r}_t \in \mathcal{F}
\end{aligned}
\tag{4}
$$

In eq. (4), the error state at the next time step is always determined by fixing the noise term at its mean value, in our settings, the mean of the gaussian noise is set to be $\mathbf{0}$. We can also notice that at each time step, the formulation approximates an infinite-horizon problem to a discounted finite-horizon deterministic optimal control (DOC) problem with continous state space and control space and time horizon T. T is a hyper-parameter we tune to get better performance in our task. This new formulation is easier to solve with a *non-linear program* (NLP) solver like *CasADi*[1]. The pipeline is thus designed as a repeated *reformulate-take one step further-reformulate* process. We can view this process as a dynamic online strategy, the reason why we only take one step is because we somehow ignored the stachatic nature of the noise, thus to get overall smaller accumulated error, we need to reformulate the problem when we get to the new state and evaluate and replan the control policy.

To solve eq. (4) at each time step, we rewrite it as a more compact non-linear program form:

$$
\begin{aligned}
\min_{\mathbf{U}} \quad & c(\mathbf{U}, \mathbf{E}) \\
\text{s.t.} \quad & \mathbf{U}_{lb} \leq \mathbf{U} \leq \mathbf{U}_{ub} \\
& \mathbf{h}_{lb} \leq \mathbf{h}(\mathbf{U}, \mathbf{E}) \leq \mathbf{h}_{ub}
\end{aligned}
\tag{5}
$$

where $\mathbf{U} := \left[\mathbf{u}_\tau^\top, \ldots, \mathbf{u}_{\tau+T-1}^\top\right]^\top$ and $\mathbf{E} := \left[\mathbf{e}_\tau^\top, \ldots, \mathbf{e}_{\tau+T}^\top\right]^\top$ are vectorized control sequence and error state sequence. In our settings, the constrains in eq. (5), explicitly parametrized as *control in-bounds constraint*, *robot position in-bounds constraint*, *error state transition constraint*, *collision avoidance constrain*, along with the cost function, are forwarded to a *CasADi* NLP solver to get a numeric solution of $\mathbf{U}$ and $\mathbf{E}$.

1) Online Numeric Solution with NLP Solver: As the final NLP form is formulated as eq. (5) for each time step, we can repeatedly parameterize it and feed to a CasADi solver to get numeric solution on the fly. The optimization objective $c(\mathbf{U}, \mathbf{E})$ can be expressed as

$$
\mathrm{c(U,\,E)}=\mathrm{q}(e_{\tau+T}) + \sum_{t=\tau}^{t+T-1} \left(\tilde{p}_t^T Q \tilde{p}_t + q\left(1 - \cos\tilde{\theta}_t\right)^2 + u_t^T R u_t \right)
\tag{6}
$$

For simplicity, we designed $\mathbf{Q}$ and $\mathbf{R}$ as:

$$
\mathbf{Q} = \mathrm{Qdiag} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\tag{7}
$$

and

$$
\mathbf{R} = \mathrm{Rdiag} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\tag{8}
$$

where Qdiag and Rdiag in eq. (7) and eq. (8) are both scalar served as hyper-parameters.

2) control in-bounds constraint: this constraint states that each control input element $\mathbf{u}$ in $\mathbf{U} = \left[\mathbf{u}_\tau^\top, \ldots, \mathbf{u}_{\tau+T-1}^\top\right]^\top$, can't exceed the minimum and maximum bounds $\mathbf{U}_{lb} = [v_{\min}, w_{\min}]^T$ and $\mathbf{U}_{ub} = [v_{\max}, w_{\max}]^T$ within time-horizon T, where $v_{\min} = 0$, $w_{\min} = -1$ and $v_{\max} = 1$, $w_{\max} = 1$.

3) robot position in-bounds constraint: this constraint states that at any time, the robot's position $\mathbf{x}_t = \tilde{\mathbf{p}}_t + \mathbf{r}_t$ can't exceed the boundaries of the 2D grid:

$$\begin{bmatrix} -3 \\ -3 \end{bmatrix} \leq \tilde{\mathbf{p}}_t + \mathbf{r}_t \leq \begin{bmatrix} 3 \\ 3 \end{bmatrix} \tag{9}$$

4) error state transition constraint: this constraint corresponds to the error state transition function in eq. (4) with time step $\Delta = 0.5$:

$$\mathbf{e}_{t+1} = g\left(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{0}\right), \quad t = \tau, \ldots, \tau+T-1 \tag{10}$$

we need to pay attention that this constraint holds for all the time step within T, thus there are T number of constraint when forwarding to the solver.

5) collision avoidance constrain: this constraint corresponds the defination of free space $\mathcal{F} := \mathcal{C} \backslash (\mathcal{C}_1 \cup \mathcal{C}_2)$ in our settings. To make the generated path safer, we also specified a hyper-parameter *safety margin* to shrink the $\mathcal{F}$ a little bit away from the boundaries of the obstacles:

$$\|\tilde{\mathbf{p}}_t + \mathbf{r}_t - \mathcal{O}_i\|_2 \geq \text{radius} + \text{safety margin} \tag{11}$$

where $\mathcal{O}_i,\ i = 1, 2$ are the center position vector of two circular obstacles, $\text{radius} = 0.5$ and we set the $\text{safety margin} = 0.05$.

With the above defined objective and constrains at each time step, the NLP solver will give us a numerical solution of $\mathbf{U} = \left[\mathbf{u}_\tau^\top, \ldots, \mathbf{u}_{\tau+T-1}^\top\right]^\top$. We then retrieve $\mathbf{u}_\tau$ as the optimal control input for current robot state and execute it once with `car_next_state`, which is the Kinematics model of the robot state and get to the next state and repeat this process until the end of simulation time. Ideally, without the presence of the gaussian noise incoorporated in the robot state transition model, the tracking results can be perfectly smooth in visualization, we will discuss more details about experiment results in Section IV.

B. Generalized Policy Iteration (GPI)

Unlike the CEC scheme which constantly reformulates a new finite-horizon deterministic control problem at each time step, Generalized Policy Iteration (GPI) scheme solves the original eq. (3) directly but need to discretize the state and control spaces into finite collections in a tabular manner. The idea behind this is pretty simple: since we can directly access the underlying environment model, maintaining a state transition lookup table in an offline manner is available. Generally, GPI combines elements of both policy evaluation and policy improvement in an iterative process. It learns an approximation of the optimal control policy by iteratively updating the value function and the policy. This iterative process continues until the policy converges to an optimal or near-optimal solution.

C. State and Control Spaces Discretization

Although the problem itself has infinite-horizon, but the state we designed need to be parameterized with time dimention. The main reason behind this because the reference trajectory we need to follow is already known and periodic and with period $n_t = 100$ so this is necessary. For simplicity, we interpolate the notation here and denote the disretized finite state $\mathbf{x}_i$ as a tuple with four elements:

$$\mathbf{x}_i = (s_t, s_x, s_y, s_a) \tag{12}$$

Natually, because the period of reference trajectory is $n_t = 100$, s_t should be ranged from 0 to 50 descretized by time step 0.5. In eq. (12), the latter 3 elements of the state tuple can be also viewed as a discretization verison of the error state $\mathbf{e}_i = (s_x, s_y, s_a)$ as defined in eq. (2).

1) Adaptive Finer-coarser Resolution for State Space: In practice, we didn't discretize the latter 3 dimensions using the same resolution evenly. Instead, we followed the adaptive discretization of $\mathbf{e}_i$ with a finer grid closer to the reference trajectory and a coarser grid further away. The main reason for this is maintaining the same small and even resolution along each dimension is so expensive in both computation and memory consumption. For example, the bounds for both s_x and s_y are $[-3, 3]$, if we evenly divide them to 10 elements and evenly divide s_a every 15 degree $(360/15 = 24)$, then the cardinality of the state space $|\mathcal{X}| = 100 \times 10 \times 10 \times 24 = 240000$, which is too computationally entensive to process during value iteration or policy iteration. Our method is based on such observation that when the error of potential current state $\mathbf{x}_i$ deviates the corresponding

reference state $\mathbf{x}_{\text{ref}}$ too much, the policy made for that state is of less importance and some standard control strategy will be discovered eventually for such *target lost* situation. So in practice, we descretize s_x, s_y, and s_a into 3 levels and 9 values in total, respectively. For s_x and s_y, the finest-level resolution is the evenly distance from -0.25 to 0.25 divided into 5 parts; the middle-level is descretized to ± 0.5; and the coarsest-level, ± 3.0. Similarly for s_a (descretized error of orientation angle), the finest-level resolution is the evenly distance from $-\pi/4$ to $\pi/4$ divided into 5 parts; the middle-level is descretized to $\pm\pi/2$; and the coarsest-level, $\pm\pi$.

2) Even Resolution for Control Space: For control space, we use even resolution within $[v_{\min}, v_{\max}]$, and $[\omega_{\min}, \omega_{\max}]$ with $\text{res}_v = 0.1$ and $\text{res}_\omega = 0.2$:

$$v_i \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \tag{13}$$

and

$$\begin{aligned} w_i \in \{&-0.2, -0.4, -0.6, -0.8, -1.0, 0.0, \\ &0.2, 0.4, 0.6, 0.8, 1.0\} \end{aligned} \tag{14}$$

3) State and Control Space Candinality: Section III-C1 and Section III-C2 discussed our design for the descritization strategy, we can calculate the corresponding state space and control space cardinality to further assist the design for transition function:

$$\begin{aligned} |\mathcal{X}| &= 100 \times 9 \times 9 \times 9 = 72900 \\ |\mathcal{U}| &= 11 \times 11 = 121 \end{aligned} \tag{15}$$

D. Transition Function Design

Since we no longer model the Gaussian noise as fixed values as in CEC method, we now need to analyze the distribution $p_f(\cdot|\mathbf{x}_i, \mathbf{u}_i)$ of the next state. For simplicity, we separate $\mathbf{x}_i$ into 2 parts: s_t and $\mathbf{e}_i = (s_x, s_y, s_a)$. The transition of s_t is relatively easier to analyze:

$$s_j = \Delta * ((s_i/\Delta + 1) \mod 100) \tag{16}$$

This is because the periodicity nature of the reference motion, once the reference trajectory goes to the last point along the path, the next referenced point will the original point 100 steps before. The transition of $\mathbf{e}_i$ is actually the descretized version of eq. (2) with stochastic noise considered. For a given discretized error state $\mathbf{e}$ and control $\mathbf{u}$, The next discretized error state $\mathbf{e}\prime$ should also be a Gaussian random variable with mean and covariance:

$$\mathbf{e}' \sim \mathcal{N}(g(t, \mathbf{e}, \mathbf{u}, \mathbf{0}), \text{diag}(\sigma^2)) \tag{17}$$

This process can be visualized as in Figure 2 where the next error state $\mathbf{e}'$ can be at some random candidate locations around its mean value.

After we evaluate the mean for the final transition after plugging into the input discretized error state and control input, we select and pick the *top-4 nearest* candidate states $\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3, \mathbf{e}'_4$ by comparing the absolute difference alone each dimension and then evaluate the probability density function (pdf) at these states $p(\mathbf{e}'_1), p(\mathbf{e}'_2), p(\mathbf{e}'_3), p(\mathbf{e}'_4)$. These values are then normalized by dividing by the sum of these values, and selected as the new transition probabilities corresponding the next 4 new candidate states under specific discretized error state and control input at current iteration. And for the states not among the top-4 candidates, we simplity init the transition probability to them as 0. For each input state and control, the output will be of the new states of the same number of the state space along with corresponding possibilities. But most of the elements in the output will be 0 thus resulting a super large sparse matrix and we used this nature to store such transitions to be more memory-efficient. More specificly, the output transition will be of the scale $|\mathcal{X}| \times |\mathcal{U}| \times |\mathcal{X}|$.

E. Stage Cost Design

The stage costs between transitions are basicly the same as each single term of the stage costs defined in eq. (3) with addition cost to penalize the entering the collision area:

$$\begin{aligned} l(x, u) = \tilde{p}^T Q \tilde{p} + q\left(1 - \cos\tilde{\theta}_t\right)^2 \\ + u^T R u + \text{cost}_{\text{collision}} \end{aligned} \tag{18}$$

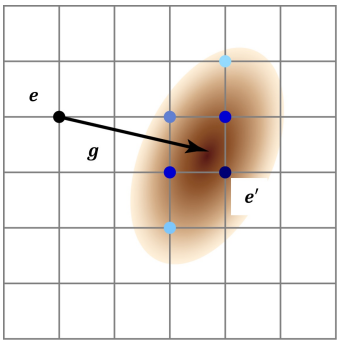

Fig. 2. An illustration of the discretized transition possibilities that is sampled from its distribution that is used in GPI scheme to solve the reference trajectory tracking problem

where we use fixed scalar cost for collision avoidance:

$$\text{cost}_{\text{col}} := \begin{cases} k_{\text{col}} & \|\tilde{p}_t + r_t - \mathcal{O}_i\|_2 < \text{rad} + \text{mar} \\ 0 & \|\tilde{p}_t + r_t - \mathcal{O}_i\|_2 \geq \text{rad} + \text{mar} \end{cases} \tag{19}$$

where k_{col} is a hyper-parameter that can be tuned.

After all the preparation of getting a discretized MDP is done, we now present our Generalized Policy Iteration algorithm that operates on it as Algorithm 1:

Algorithm 1 Generalized Policy Iteration (GPI)

Input: discretized MDP with $\mathcal{X}$, $\mathcal{U}$, transitions P, stage costs L, discounted factor γ
Output: near-optimal policy π

$V_0 \leftarrow 0$, policy evaluation iter set to $n = 1$
for $k = 0, 1, 2, \cdots$ **do**
  $Q \leftarrow L + \gamma P V_k$
  $V_{k+1} \leftarrow \min_{u \in \mathcal{U}} Q$
  $\pi \leftarrow \arg\min_{u \in \mathcal{U}} Q$
  if $\|V_{k+1} - V_k\|_2 \leq$ terminal condition **then**
    return π
  end if
end for
return near-optimal policy π

In our practice, Algorithm 1 usually will converage to less or equal than $1e-4$ in a couple of hundreds of iterations. After we get a converged policy function, we can use it to query the control strategy iteratively.

IV. Experiment Results

In section III, we discussed about CEC algorithm and GPI algorithm. In this section, we visualize the followed path returned by above algorithms with different hyper-parameters and we give some analysis on the results and conclude in the end of this section.

A. Experiment Settings

We have a reference trajectory to track with period $T = 100$. We simulate the processing to be 60 seconds.

We first evaluate the effectiveness of both CEC algorithm and GPI algorithm to solve this trajectory tracking problem. We then compared and visualized the followed path performance of both algorithms to get some intuitions.

In practice of implementing GPI algorithm, because to get a full state transition lookup table is too time-consuming, which in general, will took about 50 to 60 minutes, thus we first run the files to generate the transitions first and save them as pickle files in the form of sparse matrix objects. Then when we need to compute policies for specific hyper-parameter sets, we will load them manually and save the policy with the settings together to be used as the backend database to be queried while doing the tracking task.

Experiment results are as following:

B. CEC Algorithm

The overall tracking error is rounded to *.2f*.

1) Without the Presence of Random Noise: Ideally, when the environment has no noise at all, the original trajectory tracking problem will be reduced to a infinite-horizon Deterministic Optimal Control Problem, in this case, CEC algorithm can achieve even higher tracking accuracy, after tuning and selecting one of the hyper-parameter set as $\text{Qdiag} = 5, q = 5, Rdiag = 2, T = 15$ with safety margin set to 0.005, we have nearly perfectly followed path with overall error 2209.77, the average iteration time is about 40.6ms and the visualization can be seen in Figure 3.

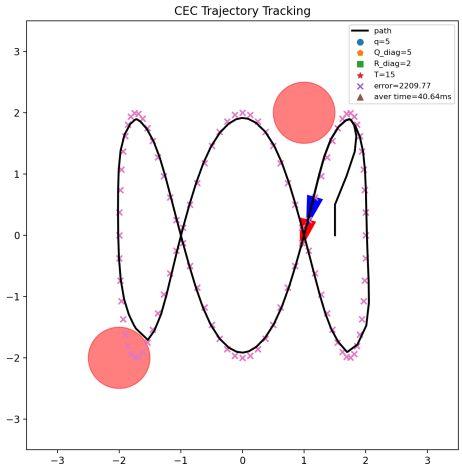

Fig. 3. CEC without environment noise can have nearly perfect tracking result

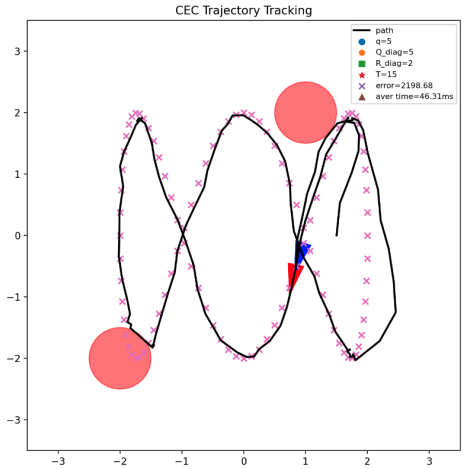

Fig. 4. CEC with environment noise deviates the reference trajectory

2) With the Presence of Random Noise: With the same set of parameters as Section IV-B1, we add the environment noise into it and get the following visualization result as in Figure 4. We can see that the tracking error is now 2298.68 and the deviation is even larger and the followed path is not that smooth as previous results.

3) The Role of Terminal Cost in CEC: The terminal cost in CEC formulation can be viewed as some kind of approximation of the infinite-horizon discounted sum of stage cost. It can not perfectly modeled, but it's better to add this term.

4) Increasing Q Results Smaller Position Tracking Error: As we can see in Figure 5, keeping other parameters still, and increasing Q results in smaller position tracking error because essentially this parameter is larger, the more the controller will try to follow more and reduce the gap on the positional with reference motion.

5) Increasing q Results Smaller Orientation Tracking Error: As we can see in Figure 6, keeping other parameters still, and increasing Q results in smaller position tracking error because essentially this parameter is larger, the more the controller

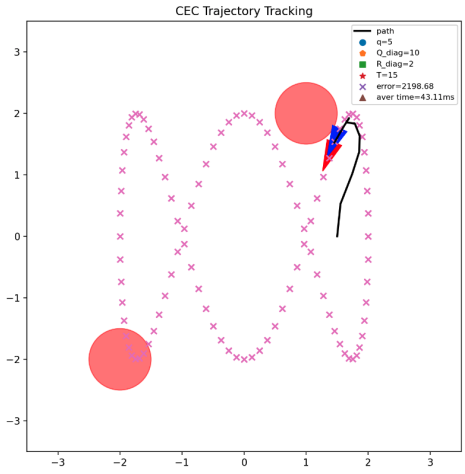

Fig. 5. CEC Q is 10

will try to follow more and reduce the gap on the orientation with reference motion.

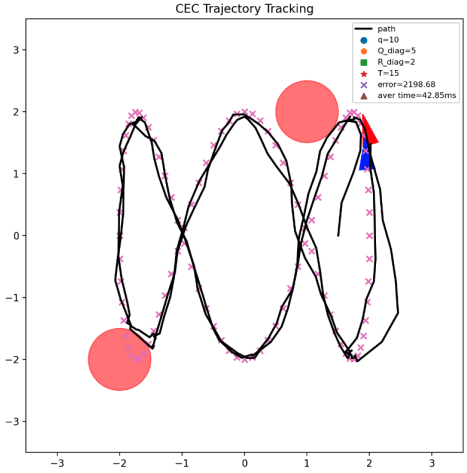

Fig. 6. CEC q is 10

6) Increasing R Results smaller Tracking Radius: As we can see in Figure 7, keeping other parameters still, and increasing R results in smaller followed path using smaller tracking radius because essentially this parameter is larger, the more the controller will try to use smaller force and velocities to track the trajectory.

7) CEC Computation Efficiency: Generally, CEC is a online planner and it can have relative good online planning performace, especially when the environment is changing fast and plan is happning at a high frequency. And on time it consumes is about 50ms average, which is enough for basic control.

C. GPI Algorithm

1) GPI Trajectory is not smooth compared to CEC: In generall, for example, in Figure 8, GPI followed path is not that smooth compared to CEC scheme.

2) GPI Trajectory is not smooth compared to CEC: In generall, for example, in Figure 8, GPI followed path is not that smooth compared to CEC scheme.

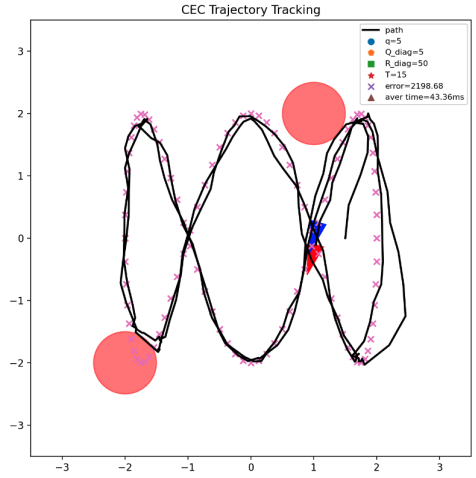

Fig. 7. CEC R is 50

3) GPI Trajectory sometimes can't avoid obstacle: For example, in Figure 8, GPI followed path can't prevent the obstacles.

4) GPI Spent most of the computation time offline thus online faster: Compaerd to CEC, GPI can have online speed 2.6ms, but most of the computation is done offline(1h even).

D. Comparison with CEC and GPI Algorithms

CEC is a model-based control approach that solves trajectory tracking problems by predicting the future behavior of the system over a finite time horizon and optimizing the control inputs accordingly. CEC assumes that the system dynamics are known and employs a certainty equivalent approach, where the uncertain parameters of the system are replaced with their nominal values. By solving an optimization problem at each time step, CEC determines the optimal control sequence for the given time horizon, and only the first control action is implemented. This process is repeated at each time step, leading to a receding-horizon control strategy.

On the other hand, Generalized Policy Iteration (GPI) is an offline planner that combines elements of both policy evaluation and policy improvement in an iterative process. It learns an approximation of

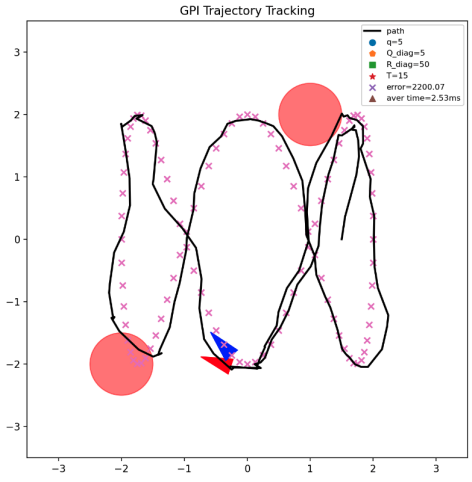

Fig. 8. GPI Q=75 q=30 R=1

the optimal control policy by iteratively updating the value function and the policy. By interacting with the system and collecting data, GPI improves the control policy based on the observed costs and states. This iterative process continues until the policy converges to an optimal or near-optimal solution. The computational consumption for this scenario can be extensive, for our 2D task with many state and control space to discretize.

For most

E. Conclusions

We performed extensive experiments to compare the different performance between CEC and GPI algorithms with different hyper-parameter sets and the results basically align with the conclusion in technical approach part.

ACKNOWLEDGMENT

Thanks Prof. Nicolay for such meaningful lectures and the commitment of TA Zhirui Dai.

REFERENCES

[1] *Casadi*. [Online]. Available: https : / / web . casadi.org/.

APPENDIX