Infinite-Horizon Stochastic Optimal Control

Sandeep Chintada
A59015527
dchintada@ucsd.edu

***Abstract*—This project aims to design a control policy for a differential drive robot to safely track a desired reference position and orientation. We explore two commonly used methods, i.e., receding horizon certainty equivalence control (CEC) and generalised policy iteration (GPI) and compare their performance and compute requirements.**

***Index Terms*—optimal control, value function, receding horizon certainty equivalence control, generalised policy iteration**

I. Introduction

Trajectory tracking is a fundamental problem in robotics that deals with controlling the motion of a robot to accurately follow a predefined trajectory or path. The trajectory can represent a desired path in space or a sequence of specific poses that the robot needs to achieve over time. The objective is to minimize the error between the robot's actual trajectory and the desired trajectory, while considering various constraints and uncertainties.

Various control strategies and algorithms have been developed to address the trajectory tracking problem in robotics. These include classical control techniques such as proportional controller, model-based control methods, and more recent approaches such as optimal control and reinforcement learning. In this project, we consider two approaches, receding-horizon certainty equivalence control(CEC) and generalized policy iteration(GPI).

Receding Horizon CEC is a control strategy that combines two techniques: receding horizon control and certainty equivalence control. It involves an iterative optimization process to compute optimal control actions over a finite time horizon. This allows for real-time responsiveness as it solves optimization problems within a short time horizon. This enables the controller to quickly react to changes in the environment and adjust control actions accordingly, ensuring effective trajectory tracking even in dynamic scenarios.

GPI aims to find an optimal control policy by iteratively updating the value function and control policy for each state. When the system dynamics and cost function are known, as is the case here, GPI can converge to an optimal policy that minimizes the expected cumulative cost. This results in improved control accuracy and the ability to closely follow desired trajectories.

II. Problem Formulation

The objective of the project is to find the control sequence that minimizes the error between the reference trajectory, which represents the desired path for the system, and the actual path followed by the system. In order to achieve this, the project formulates an infinite horizon stochastic optimal control problem.

A. Mathematical primitives

Consider a robot whose state $\mathbf{x_t} := (\mathbf{p_t}, \boldsymbol{\theta_t})$ at discrete-time $t \in \mathbb{N}$ consists of its position $p_t \in \mathbb{R}^2$ and orientation $\theta_t \in [\pi, \pi)$. The robot is controlled by a velocity input $u_t := (v_t, \omega_t)$ consisting of linear velocity $v_t \in \mathbb{R}$ and angular velocity (yaw rate) $\omega_t \in \mathbb{R}$. The discrete-time kinematic model of the differential-drive robot obtained from Euler discretization of the continuous-time kinematics with time interval ¿ 0 is given by:

$$x_{t+1} = \begin{bmatrix} p_{t+1} \\ \theta_{t+1} \end{bmatrix} = f(x_t, u_t, w_t) = \underbrace{\begin{bmatrix} p_t \\ \theta_t \end{bmatrix}}_{x_t} + \underbrace{\begin{bmatrix} \Delta\cos\theta_t & 0 \\ \Delta\sin\theta_t & 0 \\ 0 & \Delta \end{bmatrix}}_{G(x_t)} \underbrace{\begin{bmatrix} v_t \\ t \end{bmatrix}}_{u_t} + w_t \quad (1)$$

where $w_t \in \mathbb{R}^3$ models the motion noise with Gaussian distribution $\mathcal{N}(0, \text{diag}(\sigma)^2)$ with standard deviation $\sigma = \left[0.04, 0.04, 0.004\right] \in \mathbb{R}^3$. The motion noise is assumed to be independent across time and the robot state x_t.

The kinematics model in (1) defines the probability density function $p_f(x_{t+1}|x_t, u_t)$ of x_{t+1} conditioned on x_t and u_t as the density of a Gaussian distribution with mean $x_t + G(x_t)u_t$ and covariance $\text{diag}(\sigma)^2$.

The objective of this project is to design a control policy for the differential-drive robot to track a desired reference position trajectory $r_t \in \mathbb{R}^2$ and orientation trajectory $\alpha_t \in [\pi, \pi)$ while avoiding collisions with obstacles in the environment. There are two circular obstacles $\mathcal{C}_1$ centered at $(2, 2)$ with radius 0.5 and $\mathcal{C}_2$ centered at $(1, 2)$ with radius 0.5.
Let $\mathcal{F} := [-3, 3]^2 \backslash (\mathcal{C}_1 \cup \mathcal{C}_2)$ denote the free space in the environment.

It will be convenient to define an error state $e_t := (\tilde{p}_t, \tilde{\theta}_t)$, where $\tilde{p}_t := p_t - r_t$ and $\tilde{\theta}_t := \theta_t - \alpha_t$ measure the position and orientation deviation from the reference trajectory respectively. The equations of motion of the error state are as follows:

$$e_{t+1} = \begin{bmatrix} \tilde{p}_{t+1} \\ \tilde{\theta}_{t+1} \end{bmatrix} = g(t, e_t, u_t, w_t) = \underbrace{\begin{bmatrix} \tilde{p}_t \\ \tilde{\theta}_t \end{bmatrix}}_{e_t} + \underbrace{\begin{bmatrix} \Delta\cos(\tilde{\theta}_t + \alpha_t) & 0 \\ \Delta\sin(\tilde{\theta}_t + \alpha_t) & 0 \\ 0 & \Delta \end{bmatrix}}_{\tilde{G}(e_t)} \underbrace{\begin{bmatrix} v_t \\ t \end{bmatrix}}_{u_t}$$

$$+ \begin{bmatrix} r_t - r_{t+1} \\ \alpha_t - \alpha_{t+1} \end{bmatrix} + w_t \quad (2)$$

B. Optimal Control Formulation

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

$$\begin{aligned}
V^*(\tau, e) = &\min_{\pi} \mathbb{E}\Bigg[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} \big(\tilde{p}_t^{\mathrm{T}} Q \tilde{p}_t + q(1-\cos\tilde{\theta}_t)^2 \\
&\qquad + u_t^{\mathrm{T}} R u_t\big) \Bigg| e_\tau = e\Bigg] \\
\text{s.t} \quad & e_{t+1} = g(t, e_t, u_t, w_t),\ w_t \sim \mathcal{N}(0, \mathrm{diag}(\sigma)^2), \\
& t = \tau, \tau+1, \ldots, \\
& u_t = \pi(t, e_t) \in \mathcal{U} \\
& \tilde{p}_t + r_t \in \mathcal{F}
\end{aligned} \tag{3}$$

where $Q \in \mathbb{R}^{2\times 2}$ 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 $R \in \mathbb{R}^{2\times 2}$ is a symmetric positive-definite matrix defining the stage cost for using excessive control effort.

III. Technical Approach

To solve the discounted infinite horizon stochastic optimal control problem formulated in (3), we shall implement receding horizon certainty equivalence control(CEC) and generalized policy iteration(GPI) algorithms and compare these two different approaches.

A. Receding Horizon CEC

CEC is a suboptimal control scheme that applies, at each stage, the control that would be optimal if the noise variables w_t were fixed at their expected values (zero in our case). The main attractive characteristic of CEC is that it reduces a stochastic optimal control problem to a deterministic optimal control problem, which can be solved more effectively. Receding-horizon CEC, in addition, approximates an infinite-horizon problem by repeatedly solving the following discounted finite-horizon deterministic optimal control problem at each time step. This is mathematically similar to (3) and is represented as follows:

$$\begin{aligned}
V^*(\tau, e) = &\mathfrak{q}(e_{t+T}) + \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} \big(\tilde{p}_t^{\mathrm{T}} Q \tilde{p}_t + q(1-\cos\tilde{\theta}_t)^2 \\
&+ u_t^{\mathrm{T}} R u_t\big) \\
\text{s.t} \quad & e_{t+1} = g(t, e_t, u_t, w_t),\ w_t \sim \mathcal{N}(0, \mathrm{diag}(\sigma)^2), \\
& t = \tau, \tau+1, \ldots, T-1 \\
& u_t = \pi(t, e_t) \in \mathcal{U} \\
& \tilde{p}_t + r_t \in \mathcal{F}
\end{aligned} \tag{4}$$

where $\mathfrak{q}(e)$ is the chosen terminal cost. In this case, we just choose the distance between the agent and the reference at the end of the horizon of our terminal cost. The receding horizon CEC is now a non-linear program(NLP) of the form:

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

where $U = [u_1^{\mathrm{T}}, \ldots, u_{\tau+T-1}^{\mathrm{T}}]^{\mathrm{T}}$ and $E = [e_1^{\mathrm{T}}, \ldots, e_{\tau+T}^{\mathrm{T}}]^{\mathrm{T}}$.

B. Receding Horizon CEC Implementation Details

Essentially, starting from the given position, we compute the initial error with the reference trajectory and obtain the next control by solving a Nonlinear Programming (NLP) problem using CasADi.

After the solver converges to a solution, we extract the first control from the optimized sequence and apply it to the system. The system's motion model is described by equation (1). Due to the presence of motion noise, the actual position of the system may deviate from the predicted next state by the receding horizon CEC. As a result, we solve the CEC problem again and iterate this process throughout the entire simulation time. This iterative approach allows us to continually update and adjust the control inputs based on the system's actual behavior, accounting for the effects of motion noise.

To model the system accurately, we introduce 3 types of constraints in addition to the control constraints.

First, the agent is required to explicitly follow the formulated motion model. This constraint ensures that the agent adheres to the desired motion behavior specified in the model.

Second, there is a collision avoidance constraint. This constraint arises from the presence of two circular obstacles located at coordinates $[-2, -2]$ and $[1, 2]$. It is essential to impose constraints on the trajectory to ensure that the agent avoids colliding with these obstacles.

Lastly, there is a constraint that the new position of the agent must always remain within the given grid. This constraint guarantees that the agent's position stays within the boundaries of the defined grid throughout its trajectory.

The constraints can be explicitly formulated as follows:

1) **Control Constraints**:
   a) Velocity $0 \leq v \leq 1$
   b) Angular Velocity $-1 \leq \omega \leq 1$
2) **Obstacles**:
   a) $\mathrm{dist}(AgentPosition - [-2, -2]) > 0.5$
   b) $\mathrm{dist}(AgentPosition - [1, 2]) > 0.5$
3) **Grid Constraint**:
   a) $-3 \leq AgentPositionx \leq 3$
   b) $-3 \leq AgentPositiony \leq 3$
4) **Motion model**: Given by equation (2)

To ensure that our orientation value lies in the range of $[-\pi, \pi)$, we perform angle wrapping wherein we shift the angle appropriately to stay within the angle bounds.

Since we need to optimize the cost, we need to penalize the control and the error appropriately. We choose Q that penalizes the error to be $I^{2\times 2}$ and R that penalizes the control value to be $I^{2\times 2}$ as well. q that penalizes the control error is chosen to be 1 initially and then increased to 2 since the solver was unable to converge to a solution.

The time horizon was chosen to be 10 initially and increased to 20 since the solver was able to converge to the solution for a wider range of Q, R, q values.

C. Generalized Policy Iteration

Policy iteration is an iterative algorithm used to calculate the optimal value function and policy that minimizes the value function. Unlike value iteration, which focuses on iterating over values, policy iteration operates by iterating over policies.

In this iterative process, we start with an initial policy and evaluate its corresponding value function. Then, we improve the policy by selecting actions that lead to higher value estimates. This improvement step is based on the previously evaluated value function. We continue this process of policy evaluation and improvement until convergence. This algorithm can be mathematically described as follows:

1) Policy evaluation: given a policy π compute V^π
$$V^\pi(x) = \ell(x, \pi(x)) + \gamma \mathbb{E}_{x' \sim p_f(|x,\pi(x))}[V^\pi(x')] \quad \forall x' \in \mathcal{X}$$
2) Policy improvement: given V^π, obtain a new policy π'
$$\pi'(x) \in \arg\min_{u \in \mathcal{U}} [\ell(x, \pi(x)) + \gamma \mathbb{E}_{x' \sim p_f(\cdot|x,\pi(x))}[V^\pi(x')]] \quad \forall x' \in \mathcal{X}$$

In the context of policy evaluation, the objective is to compute the value function, denoted as V^π, until it converges in each iteration. However, instead of checking for convergence in every policy evaluation step, an alternative approach is to iterate a fixed number of times. This is where generalized policy iteration (GPI) becomes relevant.

Generalized policy iteration is a framework that combines policy evaluation and policy improvement steps in an iterative manner. In GPI, the policy evaluation step is performed iteratively for a predetermined number of iterations, regardless of convergence. This allows for a more structured and controlled approach to updating the value function.

The algorithm is presented below:

Algorithm 1 GPI algorithm

```
1: Initialize $V_0$
2: for $k = 0, 1, \dots$ do
3:     $\pi_{k+1}(x) = \arg\min_{u \in \mathcal{U}(.)} H[x, u, V_k(\cdot)]$
4:     $V_{k+1}(x) = \mathcal{B}^n_{\pi_{k+1}}[V_k]$
5: end for=0
```

D. GPI implementation

1) Discretization: In GPI, we need to find out the value function at every possible state. But since our state space is continuous, it is not possible to do so and hence, we need to discretize state space. Also, since the motion model depends on the reference trajectory which in turn depends on the time, we need to ensure that time is now part of the state space as well. So, we end up discretizing the state space into $n_t, n_x, n_y, n_{th} = 100, 20, 20, 20$ points for time, error in x-position, error in y-position, and the orientation difference respectively. Additionally, we discretize the control inputs as well with $n_v, n_w = 10, 20$ points. We choose a simple linear spacer like linspace to aid in our naive discretization. It is to be noted that in more practical implementations, techniques such as adaptive discretization are used. Once we have the state values and their respective discrete indices, we store the state space as a $[4, nx \times ny \times nth \times nt]$ sized array wheres the control space is stored as a $[2, nv \times nw]$ array where each column represents the discrete coordinates.

2) Stage Cost: We then use the discretized state space and control space to compute the stage cost. To begin with, we check if the coordinates fall within the obstacle and set their stage cost to ∞ irrespective of the control. The stage cost can be computed using the value function description in equation (3) i.e.,
$$\ell(x, u) = \tilde{p}_t^{\mathrm{T}} Q \tilde{p}_t + q(1 - \cos\tilde{\theta}_t)^2 + u_t^{\mathrm{T}} R u_t \tag{6}$$

3) Motion Model: Given the time instance, state and the control input, we can compute the mean of the next state using the equation (2) . In this process, we put additional constraints on the agent movement. If the next state lies within the obstacle or if it goes outside the grid, the stage cost for that action will be set to ∞. Also, we perform angle wrapping as described in CEC section to ensure that it stays within $[-\pi, \pi)$.

4) Transition Probabilities: We can use the next state mean from the motion model and the noise covariance matrix to model the transition probabilities. Here, in the motion model we need to ensure the next time state to be 0 if the current time is 99. For every time t, we have $n_x \times n_y \times n_{th}$ states and upon application of the motion model, the new state could be in any of these states at the next time-stamp. So, we use a multivariate normal distribution function that calculates the probabilities and normalize them to get the transition probabilities for every state and every control.

5) Generalized Policy iteration: Since we have everything we need to compute the value function, we can now proceed to initialize a random policy (all zeros in our case) and then improve it. We iterate the value function only once as in value iteration and perform improvement over 100 iterations. The algorithm was described in algorithm 1 in the previous section.

6) Collision avoidance:: As mentioned earlier, we check each state in the state cost computation to see if it collides with an obstacle and set its stage cost to ∞ thereby asking the agent not to choose any action that moves into the obstacle. We perform the same check in the motion model as well.

7) Scaling to large subspaces: In my initial implementation of the generalized policy iteration (GPI) algorithm, the only scaling operation employed was discretization. However, for future implementations, it would be beneficial to incorporate adaptive discretization to enhance scalability. Additionally, considering the computational demands involved, an alternative approach such as utilizing kernel functions to approximate the value function could have been a more advantageous choice rather than getting entangled in the complexities of discretizing the state space. This approach would help address the computational requirements while maintaining accuracy and efficiency in approximating the value function.

IV. Results

A. CEC

A number of experiments were conducted for testing CEC since I had to tune a lot of parameters and they could be seen in the fig directory of the code attachment. As mentioned in the implementation section, I began setting everything to 1 and I and the time horizon as 10. While it was tracking decently, the error was a bit higher at 395. So, I increased the time horizon and as expected the error reduced(to 170) but something strange happened. Because I have increased the time horizon, it tried to fit an average path instead of properly tracking the reference. Also, there seems to be a weird orientation issue wherein the agent moves left instead of right at corners. It might be because angle is not sufficiently penalized or maybe because of angle wrapping. What I had found is that angle wrapping increased the error to 2200. It could be that my implementation of the wrapping was faulty but people I've consulted with had similar issues. So, I couldn't be very sure. I tried to penalize control more in one experiment and it resulted in tracking that tries to move as little as possible. In another experiment, I tried to penalize the error but it couldn't return a feasible solution even after multiple trials. Penalizing the angle error didn't result in much of a difference.

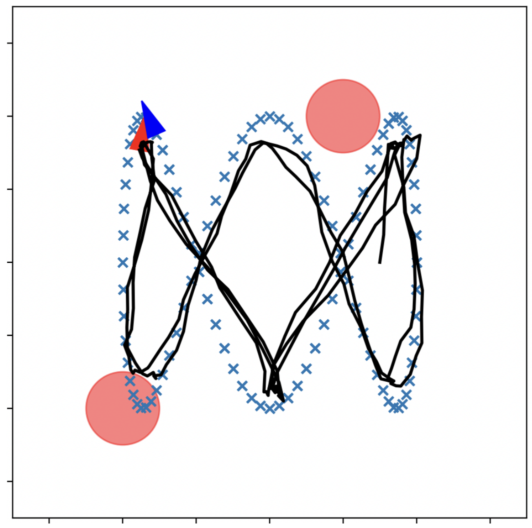
Fig. 1: Q=R=I, q=1, T=20

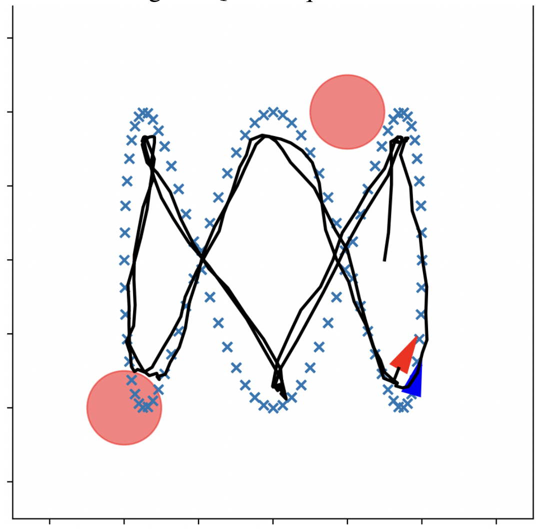
Fig. 2: Q=R=I, q=1, T=20

H

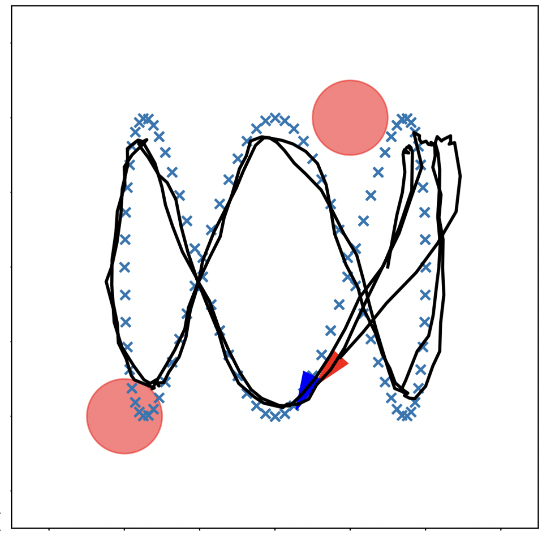
Fig. 3: Q=I, R = 0.5I, q=1, T=10

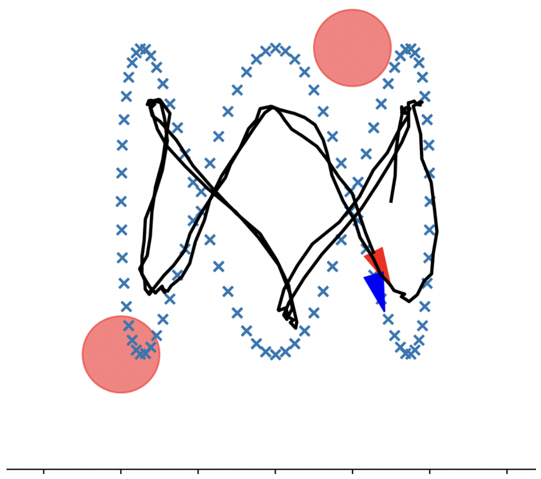
Fig. 4: Q=I, R=3I, q=1, T=20

B. GPI

While my code is running properly without any issue, it is taking way too long to run. For context, transition probability computation estimate is showing about 12 hours and hence I couldn't test my code adequately for comparison with CEC. Even stage cost computation took me around 35 mins. What this seems to suggest is that receding horizon CEC, at least for this particular scenario seems to be much much better computationally and otherwise.

V. Acknowledgements

I'd like to thank the professor Nikolay Atanasov for teaching two great courses in ECE 276A and ECE 276B and it will no doubt be a great addition to my skillset and help me in my career. The homeworks needs thinking and the projects reinforce the concepts taught brilliantly and I will be forever thankful to the professor for igniting the passion in robotics for me. I'd also like to thank the TA Zhirui Dai for help ful suggestions on Piazza and for being the sole TA of a demanding course.

Additionally, for this project, I relied on help from 2 people. One, Renukrishna gutta for help in understanding how to setup

CasADi solver and two, Prithwiraj Paul for suggesting me to use one class for the implementation and save the motion model and transition probabilities in pickle files. Though I couldn’t get the results by the time I am submitting the project, his help has been valuable in me finishing at least the code part.