Project 3: Infinite-Horizon Stochastic Optimal Control

Chen-Yeh Lin
Department of Electrical and Computer Engineering
University of California, San Diego
PID: A69036794

***Abstract*—This paper tackles the challenge of accurately following a desired trajectory with a differential-drive robot in cluttered environments subject to stochastic disturbances. We formulate the problem as a discounted infinite-horizon stochastic optimal control task with safety constraints and propose two solution methods: (1) a receding-horizon certainty-equivalent controller (CEC) that repeatedly solves a deterministic finite-horizon optimization by substituting noise with its expectation, and (2) a generalized policy iteration (GPI) scheme that constructs a discretized Markov decision process and refines the feedback law through iterative evaluation and improvement. We validate both approaches in numerical examples and MuJoCo simulations, revealing their relative strengths in terms of computational load, tracking precision, and disturbance rejection.**

I. Introduction

Autonomous mobile robots must reliably execute planned paths in real-world settings where imperfect models, sensor noise, and unmodeled disturbances can compromise performance. In applications such as warehouse logistics, agricultural inspection, or search-and-rescue, even small deviations from the intended trajectory may lead to collisions, mission failure, or excessive energy consumption. Designing controllers that enforce collision avoidance, mitigate stochastic motion errors, and respect actuation limits is therefore essential for safe, efficient robot deployment.

In this study, we focus on a differential-drive platform and cast trajectory tracking as a stochastic optimal control problem with a discounted infinite horizon. The cost functional penalizes both state tracking error and control effort while enforcing static obstacle avoidance. We explore two complementary solution paradigms: receding-horizon CEC, which reduces online complexity by replacing random disturbances with their mean and solving a nonlinear program at each time step; and GPI, which approximates the true optimal policy via tabular discretization and iterative policy evaluation/improvement. Through comparative experiments, we demonstrate how each method trades off between real-time feasibility, robustness to noise, and overall tracking accuracy, providing practical guidelines for controller selection.

II. Problem Statement

The objective is to design a control policy for a differential-drive robot to track a reference trajectory while satisfying system dynamics and environmental constraints.

A. System Dynamics and Variables

The robot's state at discrete-time $t \in \mathbb{N}$ is $\mathbf{x}_t := (\mathbf{p}_t, \theta_t)$, where $\mathbf{p}_t \in \mathbb{R}^2$ is the position and $\theta_t \in [-\pi, \pi)$ is the orientation. The control input is $\mathbf{u}_t := (v_t, \omega_t)$, consisting of linear and angular velocities. The discrete-time kinematic model is given by the following equations, which separates the state vector $\mathbf{x}_{t+1}$ into its position and orientation components for clarity:

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \Delta \left(\frac{\omega_t \Delta}{2}\right) \begin{bmatrix} \cos(\theta_t + \frac{\omega_t \Delta}{2}) \\ \sin(\theta_t + \frac{\omega_t \Delta}{2}) \end{bmatrix} v_t + \mathbf{w}_{p,t} \tag{1}$$

$$\theta_{t+1} = \theta_t + \Delta\omega_t + w_{\theta,t} \tag{2}$$

where the Gaussian motion noise $\mathbf{w}_t \sim \mathcal{N}(0, (\sigma)^2)$ has been decomposed into its position component $\mathbf{w}_{p,t} \in \mathbb{R}^2$ and orientation component $w_{\theta,t} \in \mathbb{R}$.

The error state is defined as $\mathbf{e}_t := (\tilde{\mathbf{p}}_t, \tilde{\theta}_t)$, where $\tilde{\mathbf{p}}_t := \mathbf{p}_t - \mathbf{r}_t$ and $\tilde{\theta}_t := \theta_t - \alpha_t$ are the position and orientation deviations from the reference trajectory $(\mathbf{r}_t, \alpha_t)$. The error dynamics, which describe the evolution of the tracking error, are given by:

$$\tilde{\mathbf{p}}_{t+1} = \tilde{\mathbf{p}}_t + \Delta \left(\frac{\omega_t \Delta}{2}\right) \begin{bmatrix} \cos(\tilde{\theta}_t + \alpha_t + \frac{\omega_t \Delta}{2}) \\ \sin(\tilde{\theta}_t + \alpha_t + \frac{\omega_t \Delta}{2}) \end{bmatrix} v_t + (\mathbf{r}_t - \mathbf{r}_{t+1}) + \mathbf{w}_{p,t} \tag{3}$$

$$\tilde{\theta}_{t+1} = \tilde{\theta}_t + \Delta\omega_t + (\alpha_t - \alpha_{t+1}) + w_{\theta,t} \tag{4}$$

B. Objective Function

The goal is to solve a discounted infinite-horizon stochastic optimal control problem. This is achieved by finding a policy π that minimizes the expected cumulative cost. First, we define the stage cost $L(\mathbf{e}_t, \mathbf{u}_t)$ which penalizes state deviation and control effort:

$$L(\mathbf{e}_t, \mathbf{u}_t) = \tilde{\mathbf{p}}_t^\top Q \tilde{\mathbf{p}}_t + q(1 - \cos(\tilde{\theta}_t))^2 + \mathbf{u}_t^\top R \mathbf{u}_t \tag{5}$$

The optimal value function $V^*(\tau, \mathbf{e})$ is then the solution to the following optimization problem:

$$V^*(\tau, \mathbf{e}) = \min_{\pi} \mathbb{E}\left[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} L(\mathbf{e}_t, \mathbf{u}_t) \,\middle|\, \mathbf{e}_\tau = \mathbf{e}\right] \tag{6}$$

subject to the constraints for $t = \tau, \tau+1, \ldots$:

$$\begin{aligned} \mathbf{e}_{t+1} &= g(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{w}_t), \quad \mathbf{w}_t \sim \mathcal{N}(0, (\sigma)^2) \\ \mathbf{u}_t &= \pi(t, \mathbf{e}_t) \in \mathcal{U} \\ \tilde{\mathbf{p}}_t + \mathbf{r}_t &\in \mathcal{F} \end{aligned}$$

Here, γ is the discount factor, Q and R are positive-definite cost matrices, $q > 0$ is a scalar cost, $\mathcal{U}$ is the set of admissible controls, and $\mathcal{F}$ is the obstacle-free space.

Technical Approach

This section details the two distinct methodologies employed to solve the stochastic optimal control problem for trajectory tracking: (a) a model-based receding-horizon (RHC) control, and (b) a model-free Generalized Policy Iteration (GPI) algorithm on a discretized state-action space.

(a) Receding-Horizon Certainty Equivalent Control (CEC)

The CEC approach provides a suboptimal but computationally tractable solution by simplifying the stochastic problem into a deterministic one. This is achieved by assuming the process noise is always zero ($\mathbf{w}_t = \mathbf{0}$). At each time step τ, a finite-horizon optimal control problem with a horizon of $T = 15$ steps is solved. From the resulting optimal control sequence, only the first control input, $\mathbf{u}_\tau^*$, is applied to the system. The state is then updated, and the entire optimization process is repeated at the next time step, $\tau+1$. This online re-planning makes the system reactive to disturbances and modeling errors not captured in the deterministic formulation. The core of this approach is a Non-Linear Program (NLP).

NLP Formulation: The NLP is structured to find the best sequence of controls over a finite horizon $T = 15$.

- **Optimization Variables:** The decision variables for the NLP are the sequence of control inputs $\mathbf{U} \in \mathbb{R}^{30}$ and the corresponding sequence of error states $\mathbf{E} \in \mathbb{R}^{45}$ over the planning horizon.

$$\begin{aligned} \mathbf{U} &= [\mathbf{u}_\tau^T, \ldots, \mathbf{u}_{\tau+14}^T]^T \\ \mathbf{E} &= [\mathbf{e}_{\tau+1}^T, \ldots, \mathbf{e}_{\tau+15}^T]^T \end{aligned}$$

  These are stacked into a single vector $\mathbf{X} = [\mathbf{U}^T, \mathbf{E}^T]^T$ for the solver.
- **Objective Function:** The objective is to minimize the sum of discounted stage costs over the horizon T, using a discount factor of $\gamma = 0.98$. The stage cost $L(\mathbf{e}_t, \mathbf{u}_t)$ penalizes tracking error and control effort with specific weights. A terminal cost q_f is added to penalize deviation at the end of the horizon.

$$\min_{\mathbf{X}} \left(q_f(\mathbf{e}_{\tau+T}) + \sum_{k=0}^{T-1} \gamma^k L(\mathbf{e}_{\tau+k}, \mathbf{u}_{\tau+k}) \right)$$

  The cost matrices and weights are set to $Q = \text{diag}(8, 8)$, $R = \text{diag}(1, 1)$, and $q = 5$. The stage cost and terminal cost are:

$$\begin{aligned} L(\mathbf{e}_t, \mathbf{u}_t) &= \tilde{\mathbf{p}}_t^T Q \tilde{\mathbf{p}}_t + q(1 - \cos(\tilde{\theta}_t))^2 + \mathbf{u}_t^T R \mathbf{u}_t \\ q_f(\mathbf{e}_{\tau+T}) &= \tilde{\mathbf{p}}_{\tau+T}^T Q \tilde{\mathbf{p}}_{\tau+T} + q(1 - \cos(\tilde{\theta}_{\tau+T}))^2 \end{aligned}$$

- **Constraints:** The optimization is subject to three types of constraints:
  1) **System Dynamics:** The error state must evolve according to the deterministic model ($\mathbf{w}_t = \mathbf{0}$).
  2) **Control Limits:** Control inputs must remain within $\mathcal{U} := [0.1, 1] \times [-1, 1]$.
  3) **Collision Avoidance:** The robot (radius $r_{\text{robot}} = 0.3$) must not collide with the obstacles centered at $(-2, -2)$ and $(1, 2)$, both with radius 0.5.

  This NLP is solved at each time step using the CasADi framework with the IPOPT solver.

(b) Generalized Policy Iteration (GPI)

The GPI approach directly tackles the stochastic problem by finding an optimal policy on a discretized version of the state and action spaces. It iterates between policy evaluation and policy improvement for 10 cycles until the policy converges.

- **State and Action Discretization:** A finite Markov Decision Process (MDP) is constructed by discretizing the continuous spaces.
  - Position error $\tilde{p}_x, \tilde{p}_y \in [-3, 3]$ m with $N_x = 29, N_y = 29$ points $\Rightarrow N_{xy} = 841$ cells.
  - Heading error $\tilde{\theta} \in [-\pi, \pi)$ is discretized into $N_\theta = 20$ bins.
  - Total states per time index is $|S| = 841 \times 20 = 16820$.
  - Linear velocities $v \in \text{linspace}(0.1, 1.0, 5)$ and angular velocities $\omega \in \text{linspace}(-1, 1, 5)$ give $|A| = 25$ actions.
  - Time (t) is discretized based on the trajectory's period, $t \in \{0, 1, \ldots, 99\}$.
- **Collision Avoidance Handling:** A "soft" penalty term $C(\mathbf{p}_t)$ is added to the stage cost to discourage collisions. The total cost function uses the same Q, R, and q values as the CEC controller.
- **Scaling the GPI Algorithm:** The enormous state-action space ($100 \times 4851 \times 25$) requires computational scaling strategies.
  1) **Pre-computation:** Transition probabilities $P(\mathbf{e}'|t, \mathbf{e}, \mathbf{u})$ and stage costs $L_{\text{total}}(\mathbf{e}, \mathbf{u})$ are pre-computed for all state-action pairs.

2) **Parallelization:** The `Ray` library parallelizes the pre-computation and the GPI steps. In each of the 10 main iterations, policy evaluation is run twice, followed by one policy improvement step.

- **Policy Evaluation:** For a given policy π_k, the value function is updated.

$$V_{k+1}(t, \mathbf{e}) = L_{\text{total}}(\mathbf{e}, \pi_k(t, \mathbf{e})) + \gamma \sum_{\mathbf{e}' \in \mathcal{E}} P(\mathbf{e}'|t, \mathbf{e}, \pi_k(t, \mathbf{e})) V_k(t+1, \mathbf{e}')$$

- **Policy Improvement:** A new policy π_{k+1} is formed by acting greedily with respect to V_{k+1}. This involves computing the action-value function $Q_{k+1}(t, \mathbf{e}, \mathbf{u})$ and finding the minimizing action.

$$Q_{k+1}(t, \mathbf{e}, \mathbf{u}) = L_{\text{total}}(\mathbf{e}, \mathbf{u}) + \gamma \sum_{\mathbf{e}' \in \mathcal{E}} P(\mathbf{e}'|t, \mathbf{e}, \mathbf{u}) V_{k+1}(t+1, \mathbf{e}')$$

$$\pi_{k+1}(t, \mathbf{e}) = \arg \min_{\mathbf{u} \in \mathcal{U}_d} Q_{k+1}(t, \mathbf{e}, \mathbf{u})$$

All large data arrays are placed in Ray's shared object store via `ray.put()` to avoid expensive data transfers between processes.

III. Results

A. Qualitative Assessment

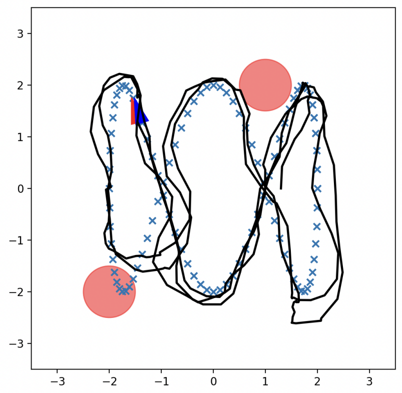

Fig. 1: Closed-loop trajectory produced by the CEC controller. The solid black line is reference way-points ; the blue $\times$ indicate the the realised path.

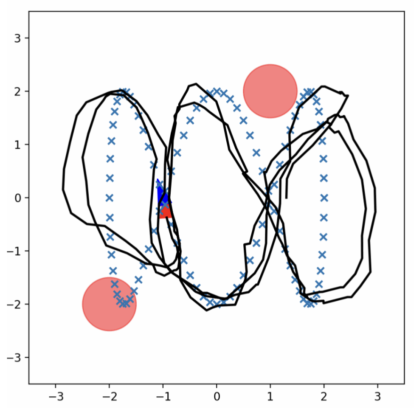

Fig. 2: Closed-loop trajectory produced by the GPI controller.

a) Trajectory shape.: Both controllers successfully follow the general shape of the desired path (Figs. 1–2) and avoid the obstacles. However, the CEC path is noticeably smoother, especially in the tight interior loop, thanks to the continuous control search in the NLP. The GPI trajectory exhibits small zig-zag artefacts—an expected consequence of the coarse discretisation of $\mathcal{U}$ and (x, y, θ) grids.

b) Collision avoidance.: Neither controller violates the 0.5 m safety margin. CEC occasionally drives closer to the upper obstacle apex, which can be attributed to the use of a myopic $N = 15$ horizon and the fact that process noise is ignored during optimisation; a slightly larger horizon or chance-constrained formulation would likely restore a larger clearance. GPI produces a more conservative clearance because the policy was trained on an explicit occupancy mask that already encapsulates the obstacle buffer.

B. Quantitative Comparison

TABLE I: Performance metrics for CEC vs. GPI. Errors are *cumulative* translation and rotation tracking errors over the entire rollout.

Method	Total CPU time [s]	Avg. iter. time [ms]	$\text{err}_{\text{trans}}$	err_{rot}
CEC	42.09	175.14	109.32	103.34
GPI	0.134	0.301	102.20	89.93

a) Computational cost.: Table I shows a $\approx 315\times$ **reduction** in wall-clock time when switching from CEC to GPI, and a $\approx 580\times$ **drop** in per-iteration latency (0.30 ms vs. 175 ms). This is expected: CEC solves a nonlinear program online, whose worst-case complexity scales roughly $\mathcal{O}(N\, n_{\text{var}}^3)$ with horizon length and decision variables, whereas GPI requires only a constant-time table lookup and nearest-grid interpolation at run time. (Recall that GPI's heavy lift—the value-iteration sweep over a $7 \times 7 \times 12 \times 5 \times 5$ grid—was executed offline and is **not** included in the timings.)

b) Tracking accuracy.: GPI achieves a 6.5 % lower cumulative translation error and a 13.0 % lower rotation error compared with CEC, despite using a significantly coarser action set. This counter-intuitive result is mainly due to the way process noise is handled:

- **CEC (noise ignored).** The optimiser assumes deterministic dynamics, producing an *optimistic* open-loop plan every 15 steps. Once noise pushes the system off the predicted state, the plan becomes sub-optimal until the next optimisation window, inflating the integrated error.
- **GPI (noise-aware).** The value-function was trained by explicitly simulating noisy transitions, so the resulting policy is inherently robust to moderate disturbances and re-optimises at every time step.

c) Effect of discretisation (GPI).: While robustness improves, discretisation introduces two drawbacks:

1) *Resolution-limited precision.* The coarse grids (0.5 m spacing in x, y and 30° in θ) lead to the choppy control visible in Fig. 2. Finer grids would smooth the trajectory but exponentially inflate the lookup table.
2) *Memory footprint.* The 5-D table used here already occupies $\approx$ 2.1 MB. Doubling each axis would raise it to 130 MB, quickly exhausting on-board cache on embedded hardware.

C. Discussion

- CEC provides smoother, constraint-aware motion but is computationally heavy; its performance degrades when the process noise it ignores becomes significant relative to the horizon length.
- GPI offers real-time control with minimal on-line overhead and better noise resilience, at the cost of grid-induced path roughness and a non-negligible offline training burden.
- In scenarios where *hard real-time* reaction is critical and memory is ample (e.g. micro-UAVs), GPI is preferable. Conversely, when on-board optimisation time is available and the environment is highly structured, CEC's continuous control space yields superior path smoothness and tighter constraint handling.

Overall, both methods remain collision-free on this benchmark, but their trade-offs are markedly different: compute vs. smoothness for CEC, memory vs. precision for GPI.