Safe Trajectory Tracking with Optimal Control

Jay Paek
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, California
jpaek@ucsd.edu

***Abstract*—This report presents an investigation into safe trajectory tracking for a ground differential-drive robot using infinite-horizon stochastic optimal control. The project aims to develop a control policy that enables the robot to track a desired reference trajectory while avoiding obstacles. The study explores two approaches: receding-horizon certainty equivalent control (CEC) and generalized policy iteration (GPI). The CEC method simplifies the stochastic problem into a deterministic one, while the GPI approach directly addresses the stochastic nature by discretizing the state and control spaces. Quantitative and qualitative results are provided to compare the performance of these methods in terms of computational complexity, tracking error, and collision avoidance.**

***Index Terms*—Robotics, dynamic programming, optimal control, searching, path-finding**

I. Introduction

Trajectory tracking is a fundamental problem in robotics, particularly for autonomous ground vehicles that must navigate dynamic environments with obstacles. Ensuring precise tracking of a reference trajectory while avoiding collisions is crucial for the safe and efficient operation of such robots. This report delves into the problem of infinite-horizon stochastic optimal control for a ground differential-drive robot, aiming to design a robust control policy that addresses the inherent uncertainties in the robot's motion.

The project focuses on a robot characterized by its position and orientation, controlled by linear and angular velocity inputs. The robot's dynamics are modeled using a discrete-time kinematic model that incorporates motion noise with Gaussian distribution. The control inputs are restricted to specific ranges, and the robot must navigate an environment with defined obstacles, ensuring it remains within the free space while tracking a reference trajectory.

II. Problem Formulation

This project considers the problem of safe trajectory tracking for a ground differential-drive robot. The state of the robot at discrete time $t \in \mathbb{N}$ is denoted by $\mathbf{x}_t := (\mathbf{p}_t, \theta_t)$, where $\mathbf{p}_t \in \mathbb{R}^2$ represents the position and $\theta_t \in [-\pi, \pi)$ represents the orientation of the robot. The control input $\mathbf{u}_t := (v_t, \omega_t)$ consists of the linear velocity $v_t \in \mathbb{R}$ and the 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 a time interval $\Delta > 0$, is given by:

$$\mathbf{x}_{t+1} = \begin{bmatrix} \mathbf{p}_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \Delta\cos(\theta_t) & 0 \\ \Delta\sin(\theta_t) & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \mathbf{w}_t,$$

where $\mathbf{w}_t \in \mathbb{R}^3$ models the motion noise with a Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathrm{diag}(\sigma^2))$ and standard deviation $\sigma = [0.04, 0.04, 0.004]^\top \in \mathbb{R}^3$. The motion noise is assumed to be independent across time and of the robot state $\mathbf{x}_t$. The kinematic model defines the probability density function $p_f(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_t)$ of $\mathbf{x}_{t+1}$ conditioned on $\mathbf{x}_t$ and $\mathbf{u}_t$ as the density of a Gaussian distribution with mean $\mathbf{x}_t + G(\mathbf{x}_t)\mathbf{u}_t$ and covariance $\mathrm{diag}(\sigma^2)$.

The control input $\mathbf{u}_t$ is limited to an allowable set of linear and angular velocities $U := [0, 1] \times [-1, 1]$. The objective is to design a control policy for the differential-drive robot to track a desired reference position trajectory $\mathbf{r}_t \in \mathbb{R}^2$ and orientation trajectory $\alpha_t \in [-\pi, \pi)$ while avoiding collisions with obstacles in the environment. There are two circular obstacles: C_1 centered at $(-2, -2)$ with a radius of 0.5 and C_2 centered at $(1, 2)$ with a radius of 0.5. Let $F := [-3, 3]^2 \setminus (C_1 \cup C_2)$ denote the free space in the environment.

To facilitate the control design, we define an error state $\mathbf{e}_t := (\tilde{\mathbf{p}}_t, \tilde{\theta}_t)$, where $\tilde{\mathbf{p}}_t := \mathbf{p}_t - \mathbf{r}_t$ and $\tilde{\theta}_t := \theta_t - \alpha_t$ measure the position and orientation deviation from the reference trajectory, respectively. The equations of motion of the error state are:

$$\mathbf{e}_{t+1} = \mathbf{x}_{t+1} - \begin{bmatrix} \mathbf{r}_{t+1} \\ \alpha_{t+1} \end{bmatrix} + \mathbf{w}_t.$$

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

$$V^*(\tau, \mathbf{e}) = \min_{\pi} \mathbb{E}\Bigg[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} \big(\tilde{\mathbf{p}}_t^\top \mathbf{Q} \tilde{\mathbf{p}}_t + q(1 - \cos(\tilde{\theta}_t))^2 + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t \big) \Big| \mathbf{e}_\tau = \mathbf{e} \Bigg],$$

where $\mathbf{Q} \in \mathbb{R}^{2\times 2}$ 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, and

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

III. Technical Approach

We will compare two different approaches for solving the problem: (a) receding-horizon certainty equivalent control (CEC) and (b) generalized policy iteration (GPI). These methods will be evaluated in terms of their ability to achieve safe and accurate trajectory tracking in the presence of motion noise and obstacles.

A. Certainty Equivalent Control

Certainty Equivalent Control (CEC) is a suboptimal control scheme that applies, at each stage, the control that would be optimal if the noise variables $\mathbf{w}_t$ 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:

$$V^*(\tau, \mathbf{e}) \approx \min_{\mathbf{u}_\tau, \ldots, \mathbf{u}_{\tau+T-1}} \mathfrak{q}(\mathbf{e}_{\tau+T}) \tag{1}$$

$$+ \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} \left(\tilde{\mathbf{p}}_t^\top \mathbf{Q} \tilde{\mathbf{p}}_t + q(1-\cos(\tilde{\theta}_t))^2 + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t \right) \tag{2}$$

subject to:

$$\begin{aligned} \mathbf{e}_{t+1} &= \mathbf{g}(t, \mathbf{e}, \mathbf{u}, 0), \quad t = \tau, \ldots, \tau+T-1 \\ \mathbf{u}_t &\in U \\ \tilde{\mathbf{p}}_t + \mathbf{r}_t &\in F \end{aligned}$$

where $\mathfrak{q}(\mathbf{e})$ is a suitably chosen terminal cost and $\mathbf{g}$ is an error update function. The receding-horizon CEC problem is now a non-linear program (NLP) of the form:

$$\begin{aligned} &\min_{\mathbf{U}} c(\mathbf{U}, \mathbf{E}) \\ &\text{subject to} \quad \mathbf{U}_{\text{lb}} \le \mathbf{U} \le \mathbf{U}_{\text{ub}} \\ &\mathbf{h}_{\text{lb}} \le \mathbf{h}(\mathbf{U}, \mathbf{E}) \le \mathbf{h}_{\text{ub}} \end{aligned}$$

where $\mathbf{U} := [\mathbf{u}_\tau^\top, \ldots, \mathbf{u}_{\tau+T-1}^\top]^\top$ and $\mathbf{E} := [\mathbf{e}_\tau^\top, \ldots, \mathbf{e}_{\tau+T}^\top]^\top$. An NLP program can be solved by an NLP solver, such as CasADi. Once a control sequence $\mathbf{u}_\tau, \ldots, \mathbf{u}_{\tau+T-1}$ is obtained, CEC applies the first control $\mathbf{u}_\tau$ to the system, obtains the new error state $\mathbf{e}_{\tau+1}$ at time $\tau+1$, and repeats the optimization online to determine the control input $\mathbf{u}_{\tau+1}$. This online re-planning is necessary because the CEC formulation does not take the effect of the motion noise into account.

The approach ensures that the control policy adapts at each time step by solving a finite-horizon optimization problem, thereby making the system more robust to changes and uncertainties in the environment. However, it is crucial to note that while CEC simplifies the problem by ignoring the stochastic nature of the noise, it does not explicitly handle the effects of motion noise, which may lead to suboptimal performance in highly uncertain environments.

1) CEC Class Initialization: The CEC class is initialized with the following parameters:

- `horizon`: The number of future time steps to consider in the optimization problem.
- `err`: The initial error state, initialized to $\mathbf{0}$.
- `time_step`: The discrete time step size, set to 0.5.

We set the time horizon to 13 for the best performance.

2) Optimization Variables: The optimization variables are defined with a CasADi symbolic variable. Each row in `sol` represents the control inputs and the subsequent state, structured as:

- `sol[i, 0]`: Linear velocity v_t.
- `sol[i, 1]`: Angular velocity ω_t.
- `sol[i, 2]`: Next position x_t.
- `sol[i, 3]`: Next position y_t.
- `sol[i, 4]`: Next orientation θ_t.

3) Cost Function: The cost function is set up as follows:

- `R`: Control effort weighting matrix.
- `Q`: Position error weighting matrix.
- `gamma`: Discount factor, set to 0.6.
- `q`: Orientation error weighting scalar, set to 0.7.

The objective function includes terms for position error, orientation error, and control effort as described in (1) and (2). We will use

$$Q = \begin{bmatrix} 0.8 & 0 \\ 0 & 0.3 \end{bmatrix}, R = \begin{bmatrix} 0.4 & 0 \\ 0 & 0.1 \end{bmatrix}$$

The logic behind these configurations come from the fact that tight tracking in the y-direction could lead to collisions due to the unsafe nature of the initial trajectory. The error in the x-direction is prioritized to minimize error while allowing the robot to keep up. As for the orientation error, it can be seen that minimizing orientation error and x-direction error is sufficient in keeping the robot in the correct trajectory. For the control input minimizations, in order to encourage the robot to make sharper turns, the cost for the angular velocity is lowered. However, too much momentum can lead to collisions, hence the higher cost for linear velocity.

4) Constraints: The motion model equality constraints ensure that the robot's state evolves according to its kinematics. We add an additional constraint that ensures the robot avoids collisions with obstacles by penalizing states near the obstacle center at $(-2, -2)$ and $(1, 2)$. We make it so that the motion model equalty constraints are tight, with 0.0001 from each other.

5) Bounds: The bounds for the optimization variables are defined as follows:

- `lb`: Lower bounds for the optimization variables.
- `ub`: Upper bounds for the optimization variables.

The lower and upper bounds for the variables are set to ensure the control inputs and states remain within allowable ranges.

- Linear velocity $v_t \in (0, 1)$.
- Angular velocity $\omega_t \in (-1, 1)$.
- Next position $x_t \in (-3, 3)$.
- Next position $y_t \in (-3, 3)$.
- Next orientation $\theta_t \in \mathbb{R}$.

B. Generalized Policy Iteration

In this section, we present the formulation of the Generalized Policy Iteration (GPI) algorithm to solve the stochastic optimal control problem (3) directly. Since the state and control spaces are continuous, we discretize them into a finite number of grid points.

1) Discretization of State and Control Spaces: The state space, consisting of position error $\tilde{\mathbf{p}}_t = (\tilde{x}_t, \tilde{y}_t)$ and orientation error $\tilde{\theta}_t$, is discretized into $(n_t, n_x, n_y, n_\theta)$ grid points. The control space, consisting of linear velocity v_t and angular velocity ω_t, is discretized into (n_v, n_ω) grid points. The state space discretization is defined over the regions $\tilde{\mathbf{p}} \in [-3, 3]^2$ and $\tilde{\theta} \in [-\pi, \pi)$, while the control input space U is discretized over the allowable set $U := [0, 1] \times [-1, 1]$.

Since $\tilde{\theta}$ is an angle, wrap-around should be enforced in its discretization. The position error $\tilde{\mathbf{p}}$ and control input $\mathbf{u}$ are discretized over the regions $[-3, 3]^2$ and U, respectively. The reference trajectory is periodic with a period of 100, so we set $n_t = 100$.

2) Transition Probabilities in the Discretized MDP: To create transition probabilities in the discretized Markov Decision Process (MDP), for each discrete state $\mathbf{e}$ and each discrete control $\mathbf{u}$, we choose the next grid points $\mathbf{e}'$ around the mean $\mathbf{g}(t, \mathbf{e}, \mathbf{u}, 0)$, where $\mathbf{g}$ represents the deterministic part of the state transition. We evaluate the likelihood of the motion noise $\mathcal{N}(\mathbf{0}, \text{diag}(\sigma^2))$ at the chosen grid points and normalize the probabilities so that the outgoing transition probabilities sum to 1.

3) Safety Constraints: To account for safety constraints, ensuring $\tilde{\mathbf{p}}_t + \mathbf{r}_t \in F$, we can either:

- Disallow transitions to states that may lead to collisions, or
- Introduce an additional stage-cost term to penalize collisions.

4) Pre-computation for Efficiency: To speed up computation, certain results can be pre-computed and stored for downstream use. For example, the transition probabilities $p_f(\mathbf{e}' \mid \mathbf{e}, \mathbf{u})$ can be pre-computed and stored in a matrix of size $(n_t, n_x, n_y, n_\theta, n_v, n_\omega, 8, 4)$, considering 8 neighbors. The same technique can be used for the stage cost $\ell(\mathbf{e}, \mathbf{u})$ and other relevant terms. Furthermore due to the sheer volume of values to maintain, we will proceed with parallel computation by using the Python ray library.

5) Generalized Policy Iteration: The GPI algorithm iteratively performs the following steps:

1. **Policy Improvement:** Given $V_k(\mathbf{x})$, obtain a policy:

$$\pi(\mathbf{x}) \in \arg\min_{\mathbf{u} \in U} \left\{ \ell(\mathbf{x}, \mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}' \sim p_f(\cdot \mid \mathbf{x}, \mathbf{u})} \left[V_k(\mathbf{x}') \right] \right\}, \quad \forall \mathbf{x} \in \mathcal{X}$$

2. **Value Update:** Given $\pi(\mathbf{x})$ and $V_k(\mathbf{x})$, compute:

$$V_{k+1}(\mathbf{x}) = \ell(\mathbf{x}, \pi(\mathbf{x})) + \gamma \mathbb{E}_{\mathbf{x}' \sim p_f(\cdot \mid \mathbf{x}, \pi(\mathbf{x}))} \left[V_k(\mathbf{x}') \right], \quad \forall \mathbf{x} \in \mathcal{X}$$

Value Update is a single step of the iterative Policy Evaluation algorithm. GPI assumes the Value Update and Policy Improvement steps are executed an infinite number of times for all states, ensuring convergence.

IV. Results

The Fig. 1 is the trajectory for the best performance of the CEC algorithm with the following statistics:

- Total time: 9.282578945159912
- Average iteration time: 38.57044080893199 ms
- Final error trains: 122.46106030050775
- Final error rotation: 59.75442865267129

Fig. 2 is when the time horizon is increased to 30.

- Total time: 25.07937717437744
- Average iteration time: 104.39033508300781 ms
- Final error trains: 126.15584287968042
- Final error rotation: 70.41631007180045

Clearly, more foresight into the future trajectory isn't necessarily good. On the left hand side of Fig. 2, the robot fails to make an optimal turn and forces itself to turn the opposite direction in order to make up for the lost ground. Furthermore, more time horizon requires more linearly more constraints for the NLP. This would significantly increase computation time, which is not very desirable for a online task.

Fig. 3 is when the angular error scalar is set to 2.

- Total time: 7.309280157089233
- Average iteration time: 30.34007449944814 ms
- Final error trains: 121.7834535355296
- Final error rotation: 67.25850716760307

Although yielding decent performance, the bottom left of the trajectory can be seen to be sharp. At this moment, the arrow hesitates to follow the trajectory and then turns clockwise in order to realign with the trajectory. This is the main issue for the angular error. Poor configurations would always lead to such rapid turn arounds at the sharper turns. There is only one configuration such that a smooth trajectory is possible, which is Fig. 1.

In conclusion, for hyperparameter tuning for safety, it is important to prioritize the realistic physical interactions that the robot would face. Smooth trajectories at a slow pace will always be more optimal than a faster unsafe travel.

There are no results for the GPI implementation.

V. Acknolwedgements

Thank you to Professor Nikolay Atanasov and the ECE 276B staff for such a great quarter!

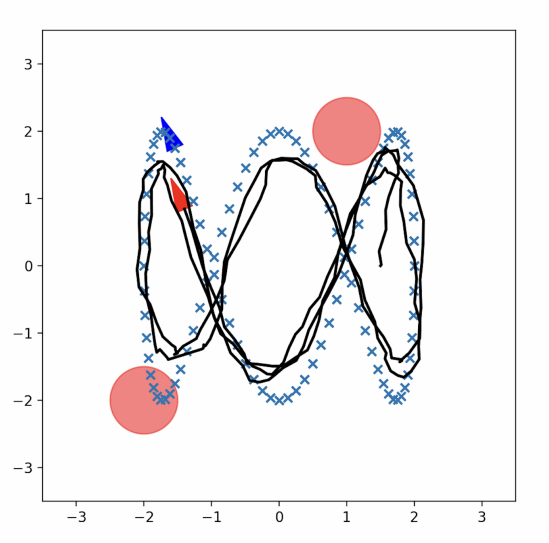

Fig. 1: Best configuration trajectory

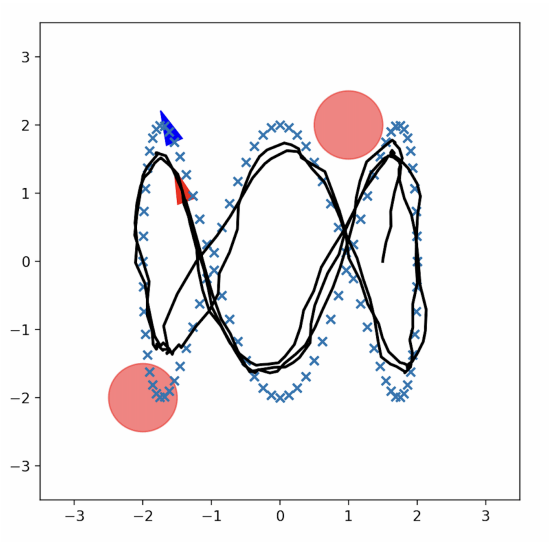

Fig. 2: Time horizon set to 30

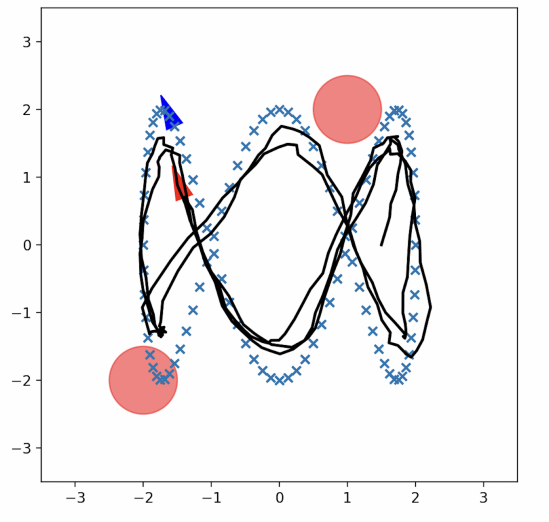

Fig. 3: Orientation error scale set to 2