Safe Trajectory Tracking

1st Weixiao Zhan
weixiao-zhan[at]ucsd[dot]edu

I. Introduction

This project aims to develop a safe trajectory tracking policy for a ground differential-drive robot. The trajectory tracking problem is formulated as a discounted infinite-horizon stochastic optimal control problem. The control policy must track a periodic reference trajectory while managing motion model noise and avoiding obstacles.

This project explores two primary approaches: receding-horizon certainty equivalent control (CEC) and generalized policy iteration (GPI), specifically the Q-value iteration. The performance of both methods is evaluated based on tracking accuracy, computational complexity, and collision avoidance capabilities.

II. Problem Formulation

A. Environment

The environment, which the robot operates in, is a xy-plane bounded within $[-3,3]\times[-3,3]$. There are two circular obstacles: Obstacle C_1 centered at $(-2,-2)$ with a radius of 0.5. Obstacle C_2 centered at $(1,2)$ with a radius of 0.5. Denote the free space:

$$\mathcal{F} = [-3,3]^2 \setminus (C_1 \cup C_2)$$

The reference trajectory is a sequence of positions and orientations:

$$r_t \in [-3,3]\times[-3,3]\times[-\pi,\pi), t < \mathcal{T}$$

with period $\mathcal{T}$ and may overlap with obstacles.

B. Motion Model

The motion model of the differential-drive robot is defined as follows. The robot's state at time t, denoted by $x_t = [p_t \quad \theta_t]^T$, consists of position $p_t \in \mathcal{F}$ and orientation $\theta_t \in \mathbb{R}$. The control input $u_t = (v_t, \omega_t)$ consists of the linear velocity $v_t \in [0,1]$ and angular velocity $\omega_t \in [-1,1]$. The motion model, with time interval $\Delta > 0$, is:

$$x_{t+1} = f(x_t, u_t, w_t) = x_t + G(x_t)u_t + w_t$$

where

$$G(x_t) = \begin{bmatrix} \Delta\cos(\theta_t) & 0 \\ \Delta\sin(\theta_t) & 0 \\ 0 & \Delta \end{bmatrix}$$

and motion noise $w_t \sim \mathcal{N}(0,\Sigma)$.

C. Receding-horizon CEC

CEC is a suboptimal open-loop-feedback control scheme. At each stage, it applies the control that would be optimal, if the noise variable w_t were realized at their expected values. The main advantage of CEC is that it reduces a stochastic optimal control problem to a deterministic optimal control problem, which can be solved more efficiently.

In addition, receding-horizon CEC approximates infinite-horizon problem by repeatedly solving a discounted finite-horizon deterministic optimal control problem, with look ahead steps T, at each time step.

Formally, when given state x_τ, CEC solves following optimization problem to find control sequence $u_{\tau:\tau+T-1}$, applies the first control u_τ. At next stage, repeat this process with $x_{\tau+1}$.

$$\min_{u_\tau,\dots,u_{\tau+T-1}} q(e_{\tau+T}) + \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} \begin{pmatrix} \tilde{p}_t^\top Q \tilde{p}_t \\ + \\ q(1-\cos(\tilde{\theta}_t))^2 \\ + \\ u_t^\top R u_t \end{pmatrix}$$

$$s.t. \begin{cases} e_t = x_t - r_t \\ \quad = \begin{bmatrix}\tilde{p}_t & \tilde{\theta}_t\end{bmatrix}^T \\ x_t = f(x_{t-1}, u_{t-1}, 0) \\ \quad = \begin{bmatrix}p_t & \theta_t\end{bmatrix}^T \\ p_t \in \mathcal{F} \\ u_t \in \mathcal{U} \end{cases} \Bigg| \begin{array}{l} x_\tau \\ T \in \mathbb{N}^+ \\ Q \in \mathbb{R}^{[2\times 2]} \\ q \in \mathbb{R} \\ R \in \mathbb{R}^{[2\times 2]} \end{array}$$

in which, q is the terminal cost, Q, q, R are stage cost hyper-parameters.

D. Non-linear Programming (NLP)

An NLP program written in following standard form can be solved by an NLP solver.

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

where $U = [u_\tau^\top, \dots, u_{\tau+T-1}^\top]^\top$ and $E = [e_\tau^\top, \dots, e_{\tau+T}^\top]^\top$.

E. GPI

GPI is a powerful technique to approach infinite horizon planing problem. The standard GPI repeats following two steps until converged:

1) policy evaluation: try to find value function V^π of a given policy π.

2) policy improvements: greedy improve the policy.

To use GPI in contiguous state space, discretization or parametrization of value function are necessary.

F. Q-value iteration

Q-value iteration is an equivalent variation of general policy iteration. Its routines is shown in Algorithm 1.

Algorithm 1 Q-value iteration

1: **while** $\delta Q > \epsilon$ **do** ▷ Q-value iteration
2: for $\forall x \in \mathcal{X}, u \in \mathcal{U}$
3: $Q(x,u) = l(x,u) + \gamma \mathbb{E}_{x' \sim p_f} \left[\min_{\forall u' \in \mathcal{U}} Q(x',u')\right]$
4: **end while**
5: $\pi(x) \leftarrow \arg\min_u Q(x,u), \forall x$ ▷ extract policy

III. Technical Approach

A. Receding-horizon CEC: NLP Formulation

In this project, the receding-horizon CEC problem is converted into standard NLP form and solved using Casadi library.

1) optimization variable:

$$U = \begin{bmatrix} u_\tau & ... & u_{\tau+T-1} \end{bmatrix} \quad \in R^{[2\times T]}$$
$$U_{lb} = \begin{bmatrix} 0 & \cdots & 0 \\ -1 & \cdots & -1 \end{bmatrix}$$
$$U_{ub} = \begin{bmatrix} 1 & \cdots & 1 \\ 1 & \cdots & 1 \end{bmatrix}$$

2) objective:

$$C(U, x_\tau) = \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} \cdot \left(loss_e\left(x_t - r_t\right) + loss_u\left(u_t\right)\right)$$

in which
$$\begin{cases} loss_e(e_t) = \underbrace{\tilde{p}_t^T Q \tilde{p}_t}_{A} + \underbrace{q\left(1 - \cos\left(\tilde{\theta}_t\right)\right)^2}_{B} \\ loss_u(u_t) = \underbrace{u_t^T R u_t}_{C} + \underbrace{\left(u_t - u_{t-1}\right)^T r \left(u_t - u_{t-1}\right)}_{D} \\ x_t = f(x_{t-1}, u_t) \\ e_t = \begin{bmatrix} \tilde{p}_t \\ \tilde{\theta}_t \end{bmatrix} \end{cases}$$

The above Formulation hardcoded the motion model f, reference trajectory r_t, and error state e_t's calculation into the objective function.

Term A and B regulates the position and orientation error between states and reference trajectory. Term C regulates large controls.

Term D is an added smoothing term. It regulates the controls' change rate to avoid sharp turns. Its impact is further discussed in Results section.

3) constrains:

$$h(U, x_\tau) = \begin{bmatrix} p_{\tau+1} & \cdots & p_{\tau+T-1} \\ \|p_t - c_1\|_2 & \cdots & \|p_{\tau+T-1} - c_1\|_2 \\ \|p_t - c_2\|_2 & \cdots & \|p_{\tau+T-1} - c_2\|_2 \end{bmatrix}$$
$$h_{lb} = \begin{bmatrix} \begin{bmatrix} -3 \\ -3 \end{bmatrix} & \cdots & \begin{bmatrix} -3 \\ -3 \end{bmatrix} \\ c_1.radius & \cdots & c_1.radius \\ c_2.radius & \cdots & c_2.radius \end{bmatrix}$$
$$h_{ub} = \begin{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} & \cdots & \begin{bmatrix} 3 \\ 3 \end{bmatrix} \\ \infty & \cdots & \infty \\ \infty & \cdots & \infty \end{bmatrix}$$

in which, $x_t = \begin{bmatrix} p_t \\ \theta_t \end{bmatrix} = f(x_{t-1}, u_{t-1})$

Similar to objective function, the motion model f is hard-coded into the constrain function. The constrains enforce states $x_{\tau+1:\tau+T-1}$ to be within the free space $\mathcal{F}$.

4) formation considerations: Converting CEC into the above NLP without explicitly introducing optimization variable on state x_t or error states e_t has several benefits.

- avoids motion model constrains in following form:

$$-\epsilon < x_{t+1} - f(x_t, u_t) < \epsilon$$

- avoids implementing error state motion model.
- avoids boundary and collision check on error states e_t.

The formulation does not normalize the orientation θ_t into the reference trajectory's orientation space $[-\pi, \pi)$, as the objective function has taken periodic cos over $\tilde{\theta}$.

The formulation also skipped stage cost. A conventional stage cost $q(x_{\tau+T}) = \gamma^T(loss_e(e_{\tau+T}) + loss_u(u_{\tau+T}))$ would have the same effect of increasing look ahead step by 1, thus skipped.

5) hyper-parameter tuning: In practice, following hyper-parameters produced best results:

$$T = 5 \qquad \gamma = 0.9$$
$$Q = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad q = 2$$
$$R = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix} \quad r = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix}$$

Different hyper-parameters and their effect are further discussed in Results section.

B. GPI: GPU Batch Q-value iteration

The computation of Q-value iteration is highly parallelizable, making GPUs the ideal hardware to leverage. In this project, a GPU-accelerated batch Q-value iteration is implemented with PyTorch.

1) state space and control space discretization: The orientation is first normalized into $[-\pi, \pi)$. The state space $\mathcal{X} = [-3, 3]^2 \times [-\pi, \pi)$ is discretized into (nx, ny, nth) equally-spaced points in each dimension. The control space $\mathcal{U} = [0, 1] \times [-1, 1]$ is discretized into (nv, nw) equally-spaced controls values. Since the trajectory has period $\mathcal{T}$, the time horizon is discretized in $nt = \mathcal{T}$ steps.

2) f:transition matrix and probability: The continuous and stochastic motion model

$$p_f(\cdot|x, u, w)$$

is discretized and approximated using the following method:

- For all discrete state x and control u, a neighborhood with K discrete states, denote as $\{x'_k|k < K\}$, around the next state's expectation, $x' = f(x, u, 0)$ is considered possible next states.
- The neighborhood selection make sure the positions p'_k are clamped between $[-3, 3]^2$ and the orientations θ'_k are wrapped around.
- The likelihood of transitioning into x'_k is evaluated using Gaussian distribution [1] $N(x', \Sigma)$.
- The transition probability distribution is normalized [2] within the neighborhood, so that $\sum_k Prob(x'_k) = 1$.

The discretization of f can be efficiently computed with PyTorch batch and broad cast operations. The results are saved in two tensors: transition tensor P with shape $(K, nx, ny, nth, nv, nw, 3)$ stores the **indices** of neighborhood points x'_k; transition probabilities tensor P_p with shape $(K, nx, ny, , nth, nv, nw)$ stores their probabilities.

The transition tensors are discretization of f, thus is **time invariant**.

3) stage cost:

$$l(x_t, u_t) = \tilde{p}_t^T Q \tilde{p}_t + q\left(1 - \cos\left(\tilde{\theta}_t\right)\right)^2 + u_t^T R u_t$$
$$s.t. \quad \begin{bmatrix}\tilde{p}_t \\ \tilde{\theta}_t\end{bmatrix} = x_t - r_t$$

The stage cost can be discretized and saved to tensor L with shape $(nt, nx, ny, nth, nv, nw)$. For those states that are on the boundary and causing collision, their stage costs are set to ∞. The stage cost dependents on the reference trajectory and is **time variant**.

4) Q-value iteration and policy extraction: implements Algorithm 1 using batch operations. It first computes current V-values $V = \min_{\text{dim}=[-2,-1]} Q$. The expectation $E_{x' \sim p_f}$ is approximated by the weighted sum of neighborhoods' V-values at next time step $t' = (t + 1) \mod nt$. Q values are stored in tensor Q, with shape $(nt, nx, ny, nth, nv, nw)$.

Q value iteration is executed until $\|Q_k - Q_{k-1}\| < \delta$. Then the policy π is extracted: $\pi = \arg\min_{\text{dim}=[-2,-1]} Q$.

5) implementation considerations: Similar to CEC approach, standard states and motion model f is discretized instead of error states. The primary considerations is memory consumption. The standard motion model need to discrete $6 * 6 * 2\pi * 1 * 2$ hyper-volume but the error motion model need to discrete $12 * 12 * 2\pi * 1 * 2$ hyper-volume which is 4 times greater.

[1]The motion model f is not linear, but we will assume it is, and approximated the result distribution with Gaussian distribution.

[2]The torch normalization function will correctly return all zero if the input is all zero

However, the error state allows adaptive discretization: discrete finer near the origin and coarse on the peripheral, which is left for future work.

6) hyper-parameter tuning: In practice, GPI used the same Q, q, R as CEC. And following discretization perform the best:

$$\begin{array}{lll} nt = 100 & & \\ nx = 31 & ny = 31 & nth = 11 \\ nv = 6 & nw = 11 & \\ K = 1 & \delta = 10 & \end{array}$$

Because the discretization is relatively sparse compare to the noise variance, most of the probability weight is concentrated on one or two points in the neighborhood. A 7-point spindle neighborhood ($K = 7$) would have $7\times$ peak memory consumption [3] compare to $K = 1$ neighborhood in batch Q iteration, which unfortunately exceeded my GPU GRAM capacity. Thus the nearest point neighborhood ($K = 1$) is used.

Those infinite stage costs are set to a large number of 10^{10}, as infinity cause `nan` in computation. With $\delta = 10$, Q-value iteration takes around 50 iterations to converge.

IV. Results

Best results of using P-controller CEC, and GPI are present in Figure 1. Both CEC and GPI are able to track the reference trajectory and avoid obstacles. Full animated results can be found at `fig/*.gif`.

A. CEC

This section compares results produced by different hyper-parameters of CEC.

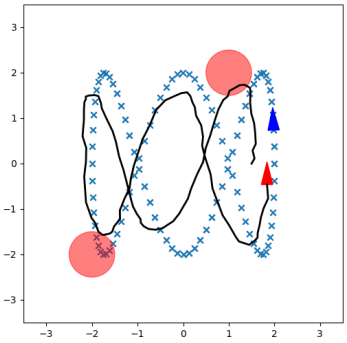
1) look ahead steps: Figure 2 shows the effect of changing T. Compare to $T = 5$ (Figure 2b), Too small $T = 1$ (Figure 2a) caused the policy can't avoid obstacles in advance. Too large $T = 20$ (Figure 2a) caused the solver converged to a different optimal, it first bypassed the obstacles then chase the trajectory. This could due the choose of $\gamma = 0.9$, the solver didn't weight the near stages enough.

In addition, larger T cause CEC slower to solve. $T = 25$ is around the limit of finding next control in time on my machine.

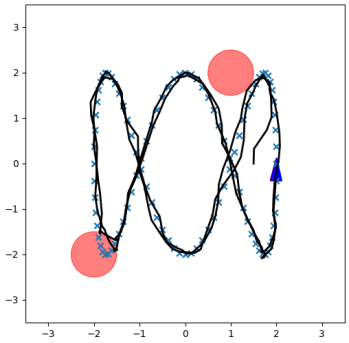
2) smoothing term: In the CEC formation, a smoothing term D is added. Figure 3 shows the result without smoothing and with $r = 1$ compare to $r = 0.05$ at 3b. With the smoothing term, the turns are less sharp. For a different less smooth reference trajectory, such smoothing term could become useful.

3) less important: γ, q: Figure 4 shows two less important parameters. Non-discounted result ($\gamma = 1$ at Figure 4c) is really close to the baseline result with $\gamma = 0.9$. Figure 4b shows the orientation term B is also less import in this project, because the reference orientation is always the tangent direction of reference trajectory. For a different motion model with degree of freedom in orientation, term B would be more significant.

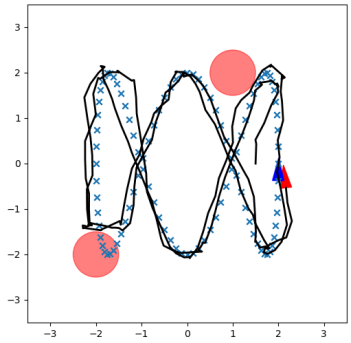
[3]A non-batch or subdivided batch Q iteration would have smaller peak memory consumption, but due to time limits and left for future work.



(a) p-controller

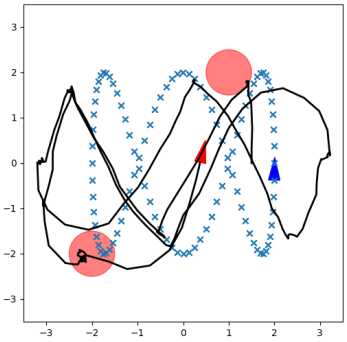


(b) CEC

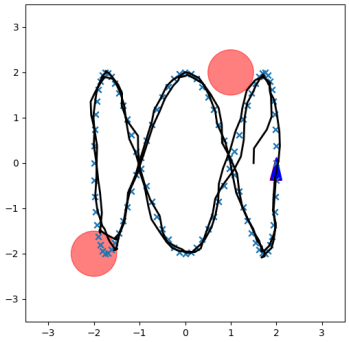


(c) GPI

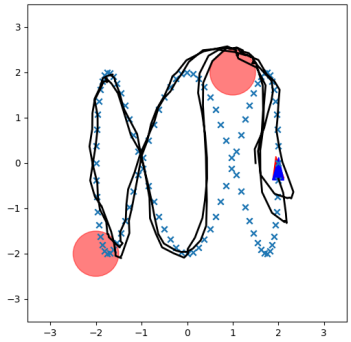
Fig. 1: Best results from CEC, GPI



(a) $T = 1$



(b) base line $T = 5$

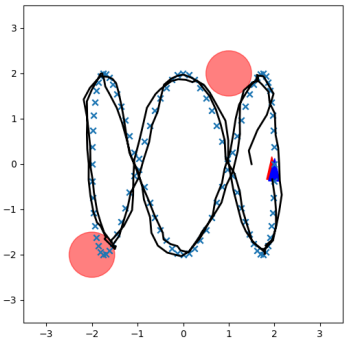


(c) $T = 20$

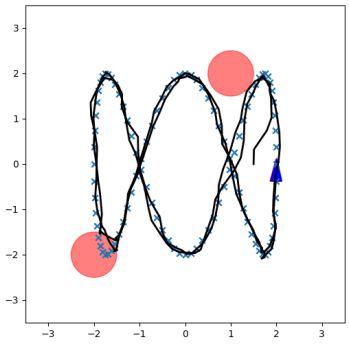
Fig. 2: look ahead steps

B. GPI

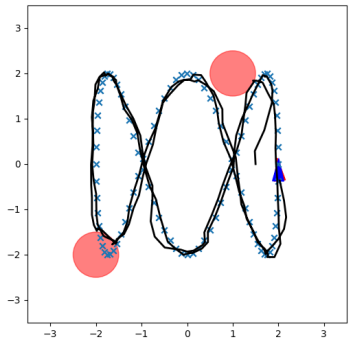
Figure 1c shows the result of optimal policy extracted from Q-value iteration. Compare to CEC results, it is clear that due to discretization granularity the GPI's control is less refined. However, the advantage of GPI is it online runtime. Only one table lookup can decide the current control, which can be significant in latency sensitive and small time interval applications.



(a) no smoothing $r = 0$

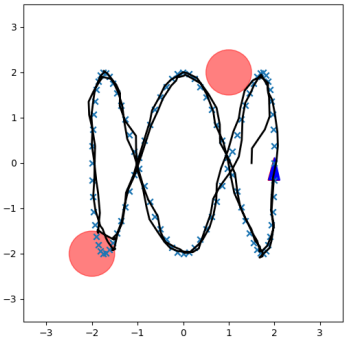


(b) base line $r = 0.05$

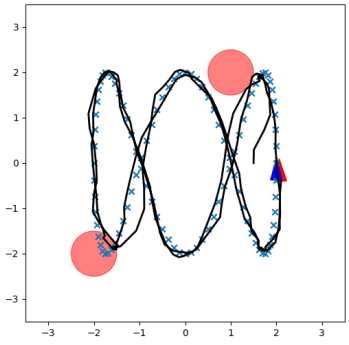


(c) more smoothing $r = 1$

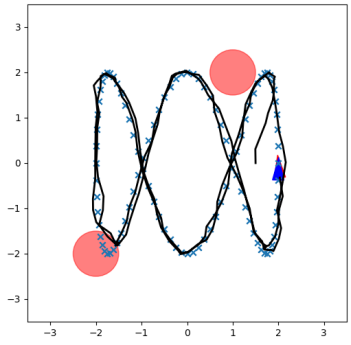
Fig. 3: smoothing term



(a) base line, $q = 2, \gamma = 0.9$



(b) $q = 0$



(c) $\gamma = 1$

Fig. 4: less important parameters