SAFE TRAJECTORY TRACKING USING CERTAINTY-EQUIVALENCE CONTROL

Chang Zhang

University of California San Diego, La Jolla, CA 92093, USA

ABSTRACT

This paper examined the performance of the certainty-equivalence control (CEC) in solving a safe trajectory tracking for a ground differential-drive robot.The performance was evaluated based on the accumulated deviation from the target reference trajectory to the robot trajectory. This performance was then compared with the general policy iteration (GPI) algorithm. We found that CEC with moderate tuning is able to track the reference target with small errors while avoiding obstacles. GPI algorithm in comparison could not track the agent.

1. INTRODUCTION

The trajectory tracking problem is to guide a ground differential robot to follow a target agent. The environment the robot and the agent are in contains two circular obstacles that the robot must avoid. The target agent is moving in a predictable pattern and sometimes it will go into one of the obstacles. When the agent goes into an obstacle, our robot will have to find a path to avoid the obstacle while keeping itself as close to the agent as space allowed. We can control the robot by two types of velocity inputs, linear and angular. At every iteration, we will generate the most optimal input and this input will change the position state and direction state of the robot. To simulate a real-life environment, such as unpredictable ground resistance or wind, we injected noise into our control motion. The noise is has a Gaussian distribution $\mathcal{N}(0, diag(\sigma)^2$, and $\sigma = [0.04, 0.04, 0.004]$. To solve this type of stochastic optimal control problem, we first used CEC. CEC is a suboptimal control scheme that simplifies this problem in two ways. One it ignores the noise and thus changes the problem from stochastic to deterministic. Two it uses a constant finite time horizon hence there is no need to make assumptions about the future costs past the time horizon. The second approach is to use the GPI algorithm, specifically policy iteration. This requires us to discretize the continuous state space and control space. This process of finding the optimal path has then become a Markov decision process, in which we also need to obtain the transition probability between the states. With the discretized space/control space and transition probability on hand, we can iterate over the value function and the policy function until the policy converges.

This problem is important to study because trajectory tracking problem exists in a vast variety of fields. These problems are often difficult to solve because of unpredictable noise and complicated constraints. To be able to solve the more difficult problems, we have to first understand how to solve our simplified problem.

2. PROBLEM STATEMENT

To have a mathematical formulation of our problem, we have to define our control state $u_t = (v_t, \omega_t)$, where $v_t \in \mathbb{R}$ is the linear velocity at time t and $\omega_t \in \mathbb{R}$ is the angular velocity at time t. We denote our state of the robot at time of t as $x_t = (p_t, \theta_t)$, where $p_t \in \mathbb{R}$ is the position in relation to the origin (0,0) and $\theta_t \in [-\pi, \pi)$ is its orientation. Time step is discrete and $t \geq 0$. Using Euler discretization, we obtain our transition for the state of the robot as:

$$x_{t+1} = \begin{bmatrix} p_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} cos(\theta_t)dt & 0 \\ sin(\theta_t)dt & 0 \\ 0 & dt \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + w_t$$
$$= f(x_t, u_t, w_t), \text{ where } w_t \text{ is the noise.}$$

As we track the target position r_t and target orientation α_t, we can calculate the error as:

$$e_t = (\tilde{p_{t+1}}, \tilde{\theta_{t+1}}) = (p_t - r_t, \theta_t - \alpha_t)$$

We use this error as our state space for convenience. Our state transition model can then be defined in the Euler discretization form as well.

$$e_{t+1} = e_t + \begin{bmatrix} cos(\tilde{\theta_t} + \alpha_t)dt & 0 \\ sin(\tilde{\theta_t} + \alpha_t)dt & 0 \\ 0 & dt \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \begin{bmatrix} r_t - r_{t+1} \\ \alpha_t - \alpha_{t+1} \end{bmatrix} + w_t$$

$$= g(t, e_t, u_t, w_t)$$

Now with all the building blocks, we can now formulate our stochastic optimal control problem as the following:

$$V^*(\tau, e) = \min_{\pi} \mathbb{E}\left[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} (\tilde{p_t}^T Q \tilde{p_t} + q(1 - cos(\tilde{\theta_t}))^2 + u_t^T R u_t) | e_\tau = e\right]$$

$$\text{s.t. } e_{t+1} = g(t, e_t, u_t, w_t), w_t \sim \mathbb{N}(0, diag(\alpha)^2)$$

$t = \tau, \tau + 1, ...$
$u_t = \pi(t, e_t)$
$\tilde{p}_t + r_t \in$ Free space

Note that our stage cost is controlled by three terms: $Q \in \mathbb{R}^{2\times2}, R \in \mathbb{R}^{2\times2}$, and q. R and q punish the position and orientation deviation of the robot from the trajectory respectively. R punishes the control input. In English, the problem is to find the policy that would produce controls that can minimize the deviation of the robot from the trajectory using the least amount of control inputs.

3. TECHNICAL APPROACH

3.1. CEC

CEC allows us to transform our problem into a deterministic finite horizon optimal control problem. To achieve this transition, we will set our noise w to 0. We then have to define a time horizon T, and we treat T as one of the hyperparameters we tune. Other hyperparameters that we need to tune are: Q,R, q,and γ. We then need to reformulate our minimization problem so that it is finite horizon and deterministic:

$$V^*(\tau, e) = \min_{u_\tau,..u_{\tau+T-1}} \min_{e_\tau,..e_{\tau+T-1}} c_T(e_{\tau+T}) + \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau}(\tilde{p}_t^T Q \tilde{p}_t + q(1 - cos(\tilde{\theta}_t))^2 + u_t^T R u_t))$$

The terminal cost $c_T = ||e_{\tau+T}||_2^2$, since we would like the end of our planned trajectory to be as close to the reference trajectory as possible. Our minimization problem is to optimize the controls and error states to minimize our defined value function.

We then have to decide the constraints base on the obstacles as well as the motion model. For the obstacles centered at Obs_1 and Obs_2 with radius of rad_1 and rad_2, we set:

$$norm(\tilde{p} + r_t - Obs_1) \geq rad_1$$
$$norm(\tilde{p} + r_t - Obs_2) \geq rad_2 + s$$

Note: s is a safety margin because it is possible that the noise "pushes" the robot into an obstacle. If the robot is in an obstacle, finding the optimal next path would become infeasible. To add our motion model, we set:

$$e_{t+1} = g(t, e_t, u_t, 0), t = \tau, ...\tau + T - 1 + s$$

Additional constraints incorporates the physical and control limitations of the robot:

$$0 \leq p_t \geq 1$$
$$-1 \leq \omega_t \geq 1$$
$$-3 \leq \tilde{p} + r_t \geq 3.$$

Because we ignore the noise in this minimization problem, yet the noise still exists in our test environment, we have to recompute the trajectory at every timestep and we only apply the first control u_t.

We use the Interior Point Optimizer to optimize our nonlinear program. The $\mathbb{O}(\sqrt{n}log(\frac{1}{\epsilon}))$ time complexity makes it a computationally cheap algorithm.

3.2. GPI

Another approach to this problem is using the general policy iteration algorithm. In our case, we choose to use the policy iteration. This allows us to pre-compute the policy and once the robot is deployed, the computation time to calculate the next move is $\mathbb{O}(1)$. The policy iteration algorithm is detailed below:

Algorithm 1 Policy Iteration

Initialize:$\pi_0 = [1, 1] \forall x$

$\mathbb{B}_*[V](x) = \min_{u\in\mathbb{U}(x)}\{l(x, u) + \gamma\mathbb{E}_{x'\sim p_f(|x,u)}[V(x)']\}$
for $k = 0, 1, 2...MaxIter$ do
  $V_{K+1} = \mathbb{B}^\infty_{\pi_{k+1}}[V_k]$
  $\pi_{k+1} = \text{argmin}_{u\in\mathbb{U}(x)} H[x, u, V_k]$

MaxIter is a hyperparameter for GPI. We set it equals to 30 to balance the time it takes to compute the policy and the accuracy of the result. To simplify our computation, our state space x is the state of the robot plus the reference trajectory. Transforming the new state space formulation to the error state we defined earlier is trivial. We define our stage cost l similar to the one in the CEC approach:

$$l(x, u) = \gamma(\tilde{p}^T Q \tilde{p} + q(1 - cos(\tilde{\theta}))^2 + u^T R u)$$

We solve $\mathbb{B}^\infty_{\pi_{k+1}}[V_k]$ as a linear system:

$$\mathbb{B}^\infty_\pi[V] = V^\pi_\infty = (I - \gamma P)^{-1} l$$
$$l_x = l(x, \pi(x)) \quad x \in X$$
$$P_{ij} = p_f(j|i, \pi(i)) \quad \text{for} \quad i, j = 1, ...n$$

Solving the problem this way will require us to discretize the state space as well as the control space. For position states, we split each axis (between -3 and 3) 12 times equally. For each orientation states, we divide the angle from $-\pi$ to π evenly.The trajectory is already discretized since there are only 100 distinct trajectory targets. We have 2 linear velocity input states and 5 angular velocity input states. Each divides the corresponding continuous input space evenly. For each continuous state/input space, we assign a discretized state/input that is the closest to that value. In total, the size of the space state $\mathbb{X}$ is $100 \times 12 \times 12 \times 5 = 72000$, and the size of the input state $\mathbb{U}$ is $5 \times 2 = 10$. To simplify the calculation, we also ignore the noise, hence the transition becomes deterministic and p_{ij} will either be 1 or 0, (not implemented) unless the transition is from a free-space to an obstacle. Due to our very large state space size, our matrix P can cause a memory issue. To alleviate our memory issue, we are using sparse matrices to store our deterministic transition probabilities. However, we still have to deal with an inverse operations, which its time complexity is $O(n^3)$.

4. RESULT

4.1. CEC

CEC is able to track our reference trajectory both visually and in term of the tracking error. The tracking trajectory that produced the least total error is shown in figure 1.Because of its hard obstacle constraint, the robot can successfully avoid obstacles. Another interesting observation is that our robot has the tendency to cut corners as it anticipates the motion of the target.This is likely due to a long planning horizon.

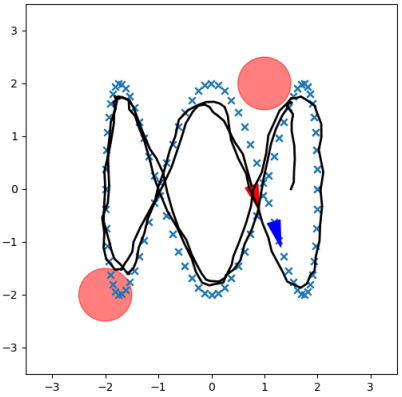

Fig. 1

4.1.1. The effect the time horizon T and γ

Increasing time horizon allows the robot to plan further ahead, yet it does not always lead to a better performance when increasing T as it potentially promotes more corner cutting behaviors. Increasing T also increases the time it takes to compute the next step. From figure 2 we see that the average computation time increases exponentially as the time horizon increases. Picking the right T is important if we need our robot to be able to make quick decisions.

Increasing γ has a similar effect as increasing T but without increasing the computation complexity. γ also can shifts the focus of the minimization to either the present time or to the distance future by changing the weights. Choosing the correct γ is important as it has a big impact on the total error.

4.1.2. The effect of Q and R

Ratio between Q and R is to balance how much we want to focus on minimizing the input and how much we want to minimize the position deviation. When the ratio between the two is not one, the problem becomes infeasible. Once we remove the obstacle constrain, the problem becomes feasible. The exact reason why this is happening is unknown.

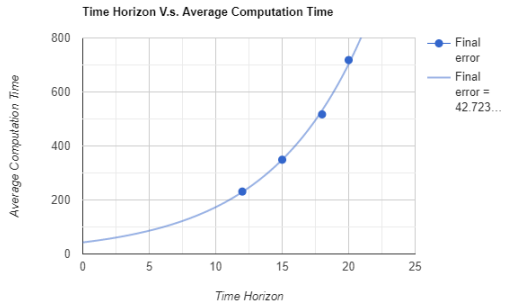

Fig. 2

4.1.3. The effect of q

Parameter q is to control how much we like your robot orientation to match the target trajectory. The optimal q we found was 50000, which was higher than other hyperparameters by 4-5 magnitudes. This means it is extremely crucial to have the direction correct as any small deviation in angles will result in large position error.

Parameters					Results	
T	Q	R	q	γ	Average iteration time(ms)	Final error
20	5	5	50000	0.7	718	107.63
18	5	5	50000	0.7	517	99.04
15	5	5	50000	0.7	349	100.03
12	5	5	50000	0.7	231	103.63
10	5	5	50000	0.7	N/A	infeasible
18	4	5	50000	0.7	N/A	infeasible
18	5	4	50000	0.7	N/A	infeasible
18	4	4	50000	0.7	531	103.88
18	6	6	50000	0.7	N/A	infeasible
18	5	5	1.00E+05	0.7	542	101.76
18	5	5	10000	0.7	514	104.47
18	5	5	1000	0.7	499.9	109.78
18	5	5	1000	0.5	508.74	140.6
18	5	5	1000	0.9	686.2	210.93

Fig. 3: The effect of parameter settings on the total error and average computation time.

4.2. GPI

We are not able to use GPI to track the trajectory since the policy for every state has converged to $u = [0, 0]$. This is possibly due to having a low discretization density. Since each discretized position state is so large, for every input the policy produces, the next state stays unchanged. Increasing the space resolution will drastically increase the time it takes to compute the policy. At the current resolution, it already takes about 3 hours to compute. Nevertheless, even the policy does not converge to the true optimal policy, we can still analyze the average time it takes to plan a step. After generating a policy, planning the next step takes a time complexity of $\mathbb{O}(1)$.

On average, each steps takes 2.37ms. This showcases GPI's ability to rapidly make online adjustment to its robot trajectory.

5. CONCLUSION

In summary, CEC can track a given trajectory decently well without colliding with any of the obstacles but it is at the cost of spending a longer time to plan for each step. Although our implementation of the GPI algorithm failed to track the trajectory, it is worth noting that GPI requires us to spend a considerable amount of time pre-computing the optimal policy. Once the policy is computed, the robot will be able to make rapid changes to its trajectory in real time.