Optimal Control

1st Marcus Schaller
Department of Electrical and Computer Engineering (ECE 276B)
University of California San Diego
La Jolla, U.S.A
mschalle@ucsd.edu

***Abstract*—Optimal control is crucial in robotics as it allows a robot to follow a trajectory or path with minimal wasted movements. In order to implement an optimal control scheme one must first select from a variety of methods that can be used to successfully solve such problems. This paper will discuss the use of receding-horizon certainty equivalent control and generalized policy iteration to solve these problems.**

I. Introduction

Optimal control is a very broad subject and therefore implementing an effective optimal control algorithm can be done in a variety of ways. The two methods that will be discussed in this paper include receding-horizon certainty equivalent control (CEC) and generalized policy iteration (GPI). This project will involve following a trajectory while at the same time avoiding obstacles. As CEC uses a powerful NLP library to be solved it will be more accurate as the state and control space will be continuous. GPI involves using a discrete state and control space but it is easier to see what the algorithm is doing during each iteration.

II. Problem Formulation

In order to implement an optimal algorithm one must first formulate the necessary equations needed to solve this problem.

A. Motion model

The robot in the project is a ground differential robot. The state of this robot is $x_t = (p_t, \theta_t)$ with a discrete to,e $t \in \mathbb{N}$. $pt \in \mathbb{R}$ is the robots position and $\theta_t \in [-\pi, \pi)$ is the orientation. The robot is controlled by a velocity input $u_t = (v_t, \omega_t)$ consisting of the linear velocity $v \in \mathbb{R}$ and angular velocity or 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 kinematic with $\tau > 0$ and t = 0,1,2...is:

$$\mathbf{x}_{t+1} = \begin{bmatrix} \mathbf{p}_{t+1} \\ \theta_{t+1} \end{bmatrix} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \tag{1}$$

$$= \underbrace{\begin{bmatrix} \mathbf{p}_t \\ \theta_t \end{bmatrix}}_{\mathbf{x}_t} + \underbrace{\begin{bmatrix} \tau\cos(\theta_t) & 0 \\ \tau\sin(\theta_t) & 0 \\ 0 & \tau \end{bmatrix}}_{\mathbf{G}(\mathbf{x}_t)} \underbrace{\begin{bmatrix} v_t \\ \omega_t \end{bmatrix}}_{\mathbf{u}_t} + \mathbf{w}_t, \tag{2}$$

$\mathrm{w}_t \in \mathbb{R}^3$ models the motion noise with Gaussian distribution $\mathcal{N}\left(\mathbf{0}, \mathrm{diag}(\sigma)^2\right)$ with standard deviation $\boldsymbol{\sigma} = [0.04, 0.04, 0.004] \in \mathbb{R}^3$. The motion noise is independent across time and of the robot state 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 $\mathbf{x}_t + \mathbf{G}(\mathbf{x}_t)\mathbf{u}_t$ and co variance $\mathrm{diag}(\sigma)^2$. The control input u_t is limited to an allowable set of linear and angular velocities $\mathcal{U} := [0, 1] \times [-1, 1]$.

B. Trajectory and Configuration Space

A trajectory for which the robot must follow is given by position trajectory $\mathbf{r}_t \in \mathbb{R}^2$ and orientation trajectory $\alpha_t \in [-\pi, \pi)$. The robot must also avoid 2 circular obstacles C_1 centered at (-2,-2) with radius 0.5 and C_2 centered at (1,2) with radius 0.5. The free space of the environment can be denoted as $\mathcal{F} := [-3, 3]^2 \backslash (\mathcal{C}_1 \cup \mathcal{C}_2)$ The following figure shows the environment.

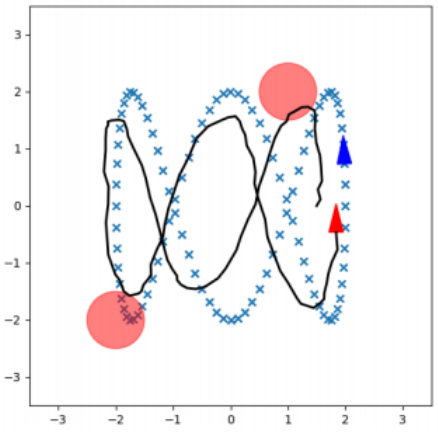

Fig. 1. Problem space

The error state which can be defined as $\mathbf{e}_t := \left(\tilde{\mathbf{p}}_t, \tilde{\theta}_t\right)$, where $\tilde{\mathbf{p}}_t := \mathbf{p}_t - \mathbf{r}_t$ and $\tilde{\theta}_t := \theta_t - \alpha_t$ measures the position and orientation deviation from the trajectory. The equation of motion for the error dynamics are:

$$\mathbf{e}_{t+1} = \begin{bmatrix} \tilde{\mathbf{p}}_{t+1} \\ \tilde{\theta}_{t+1} \end{bmatrix} = g(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{w}_t) = \tag{3}$$

$$\underbrace{\begin{bmatrix}\tilde{\mathbf{p}}_t\\ \tilde{\theta}_t\end{bmatrix}}_{\mathbf{e}_t}+\underbrace{\begin{bmatrix}\tau\cos\left(\tilde{\theta}_t+\alpha_t\right) & 0\\ \tau\sin\left(\tilde{\theta}_t+\alpha_t\right) & 0\\ 0 & \tau\end{bmatrix}}_{\mathbf{G}(e_t)}\underbrace{\begin{bmatrix}v_t\\ \omega_t\end{bmatrix}}_{\mathbf{u}_t}+\begin{bmatrix}\mathbf{r}_t-\mathbf{r}_{t+1}\\ \alpha_t-\alpha_{t+1}\end{bmatrix}+\mathbf{w}_t \quad (4)$$

C. Discounted Horizon Problem

The trajectory tracking problem can be formulated with initial time τ and initial tracking e as a discounted infinite-horizon stochastic optimal control problem.

$$V^*(\tau,\mathbf{e})= \quad (5)$$

$$\min_\pi \mathbb{E}\left[\sum_{t=\tau}^{\infty}\gamma^t\left(\tilde{\mathbf{p}}_t^\top\mathbf{Q}\tilde{\mathbf{p}}_t+q\left(1-\cos\left(\tilde{\theta}_t\right)\right)^2+\mathbf{u}_t^\top\mathbf{R}\mathbf{u}_t\right)\mid\mathbf{e}_\tau=\mathbf{e}\right] \quad (6)$$

$$\begin{aligned}\text{s.t.}\quad & \mathbf{e}_{t+1}=g\left(t,\mathbf{e}_t,\mathbf{u}_t,\mathbf{w}_t\right),\quad \mathbf{w}_t\sim\mathcal{N}\left(\mathbf{0},\mathrm{diag}(\boldsymbol{\sigma})^2\right),\quad t=\tau,\tau+1,\dots\\ & \mathbf{u}_t=\pi\left(t,\mathbf{e}_t\right)\in\mathcal{U}\\ & \tilde{\mathbf{p}}_t+\mathbf{r}_t\in\mathcal{F}\end{aligned} \quad (7)$$

where $\mathbf{Q}\in\mathbb{R}^{2\times2}$ 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 of deviating from trajectory α_t, and $\mathbf{R}\in\mathbb{R}^{2\times2}$ is a symmetric positive-definite matrix defining the stage cost for using excessive control effort.

III. Technical Approach

This project contains multiple parts which will be individually discussed within this section.

A. Receding-horizon certainty equivalent control

The receding horizon problem can be characterized with the terminal cost $\mathfrak{q}\left(\mathbf{e}_{\tau+T}\right)$, the stage cost $l(e_t,u_t)$ and γ. The stage cost in this problem can be represented as:

$$\left(\tilde{\mathbf{p}}_t^\top\mathbf{Q}\tilde{\mathbf{p}}_t+q\left(1-\cos\left(\tilde{\theta}_t\right)\right)^2+\mathbf{u}_t^\top\mathbf{R}\mathbf{u}_t\right) \quad (8)$$

When combined the value at each time step can be formulated:

$$V^*(\tau,\mathbf{e})=\min_{\mathbf{u}_\tau,\dots,\mathbf{u}_{\tau+T-1}}\mathfrak{q}\left(\mathbf{e}_{\tau+T}\right)+\sum_{t=\tau}^{\tau+T-1}\gamma^t\left(\tilde{\mathbf{p}}_t^\top\mathbf{Q}\tilde{\mathbf{p}}_t+q\left(1-\cos\left(\tilde{\theta}_t\right)\right)^2+\mathbf{u}_t^\top\mathbf{R}\mathbf{u}_t\right) \quad (9)$$

$$\begin{aligned}\text{s.t.}\quad & \mathbf{e}_{t+1}=g\left(t,\mathbf{e}_t,\mathbf{u}_t,\mathbf{0}\right),\quad t=\tau,\dots,\tau+T-1\\ & \mathbf{u}_t\in\mathcal{U}\\ & \tilde{\mathbf{p}}_t+\mathbf{r}_t\in\mathcal{F}\end{aligned} \quad (10)$$

T represents the time horizon or how many stages the value function takes into account when calculating the optimal value at time τ. By selecting the control values u that minimize the value function the optimal control policy can be returned. At each time step the an array of controls U and errors E are created so the following non linear programming objective function can be formed:

$$\begin{aligned}\min_{\mathbf{U}}\quad & c(\mathbf{U},\mathbf{E})\\ \text{s.t.}\quad & \mathbf{U}_{lb}\le\mathbf{U}\le\mathbf{U}_{ub}\\ & \mathbf{h}_{lb}\le\mathbf{h}(\mathbf{U},\mathbf{E})\le\mathbf{h}_{ub}\end{aligned} \quad (11)$$

where $\mathbf{U}:=\left[\mathbf{u}_\tau^\top,\dots,\mathbf{u}_{\tau+T-1}^\top\right]^\top$ and $\mathbf{E}:=\left[\mathbf{e}_\tau^\top,\dots,\mathbf{e}_{\tau+T}^\top\right]^\top$.

In this project a CasADi was used. CasADi is an open source tool which is able to solve non linear programming problems. In this project the Opti stack was used to help setup the problem when programming it. The first step involves creating the U and E matrices as opti variables. This allows the program to change these variables in order to optimize the problem. After this the value function was created using equation 9. A for loop would iterate each create the Error matrix in order to sum the Value function. The state was also calculated as it was determined that the terminal cost equation would check if the state was within the obstacle and make that value non optimal. This was completed by subjecting the the state to the following equation in the Opti NLP solver.

$$(X_{state}-X_{obs})^2+(Y_{state}-Y_{obs})^2>r_{obs}^2 \quad (12)$$

Where r represents the obstacle radius and X and Y the state and obstacles position's. After the loop added to a singular value V, V was sent to the optimizer as the value to be minimized. It is important to note that the control function U was constrained so that $0<=v<=1$ and $-1<=\omega<=1$. In addition the position states were constrained to be between -3 and 3. Finally the optimizer solved the NLP and returned the estimated control at time τ. This was completed for each time step until the trajectory was completed. Finally the stage cost values of Q, R, and q were diag(5.0), diag(1.0), and 0.5 respectively.

B. Generalized Policy Iteration

The second part of the project involved implementing the general policy iteration in order to follow the trajectory. However, do to the sheer size of the state space, unlike general policy iteration the policy was calculated using online methods so as to save computational time.

For each step of policy iteration algorithm the v could be between 0 and 1 and ω could be between -1 and 1. Each of these values had 101 different possible discrete values each linearly spaced between their range. In order to determine the optimal policy equation 4 was used to calculate the error given a policy u_t and the future error was returned, e_{t+1}. These values were then used in the following equation and the control input that returned the minimum value was extracted.

$$\gamma^t\left(\tilde{\mathbf{P}}_t^\top\mathbf{Q}\tilde{\mathbf{p}}_t+q\left(1-\cos\left(\tilde{\theta}_t\right)\right)^2+\mathbf{u}_t^\top\mathbf{R}\mathbf{u}_t\right) \quad (13)$$

Equation 2 was also used to calculate the future state of each control policy and equation 12 was used to determine if the future state would be in an obstacle. If it was the corresponding value would be ∞. This would be considered the terminal cost at each time and helped the algorithm select a control policy that would not result in a path through an obstacle. Once an optimal control was extracted, equation 2 was used to return the non discrete future state. From there any discrete state that was within a 5 percent range of the non discrete state was extracted. This would be 0.3 for the x and y state values and .314 for the angle state value. This is because the range for x and y is -3 to 3 and $-\pi$ to π for theta. This also helps assist with run time as it would not need to sample from all discrete values. Figure 2 illustrates this. For each discrete in this range the normal pdf was taken with mean equal to the

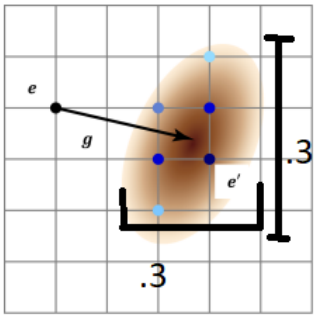

Fig. 2. Sampling area

non discrete point and linear standard deviation equal to .04 and angular standard deviation .004. The probabilities for each of the discrete points were then normalized to add to one and the the x,y, and theta values were randomly sampled using the normalized probabilities. The x, y, and theta values were sampled individually to result in an $O(3n)$ run time rather than a $O(n^3)$ further saving computational time. The resulting sampled values were set to the cars next state and the cycle was repeated for the next time step.

It is important to note that the state space involved was size 61x61x100 at time t for the x, y, and theta range respectively. Additionally as noted before the control range was 101x101. The Q, R, and q values were diag(10), diag(0), and diag(1) respectively

IV. Results

Unfortunately I struggled to get optimal results for this project. In order to calculate the effectiveness of each algorithm the error was calculated by taking the norm of the current state minus the current reference.

A. Part 1, CEC

I was able to generate a path with a minimum error of 107. It appears the algorithm struggled on the turns and would not always reach the maximum and minimum y positions of the path. When implementing obstacle avoidance the error increased to 156. All of these trajectories were calculated with a time horizon of 10 as this balanced the run time of the algorithm and the accuracy of the computed path. I found that introducing noise into the system made the algorithm less accurate and resulted in a score of 137 when obstacles were not avoided. The following figures represent these paths and the gifs were submitted with the code.

1) Future improvements for CEC: Given more time I would like to try to solve this CEC problem without using the Opti CasADi stack. While it made it easier to setup the problem it was hard to debug and determine why my path was inaccurate. Additionally, I would like to improve upon the obstacle avoidance as the method I used was not optimal.

B. Part 2 GPI

I was able to generate a path with an error of 98 when not including obstacle avoidance. However, I was unable to implement obstacle avoidance as I found when the future states were calculated all states were inside the obstacle. This obstacle avoidance resulted in a much messier path with an error of 170. Once again adding noise to the system made it more inaccurate and resulted in an error of 137. The following figures demonstrate this.

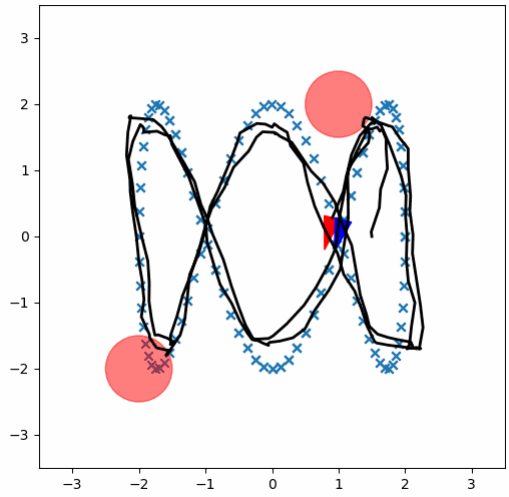

Fig. 3. CEC with no obstacle avoidance

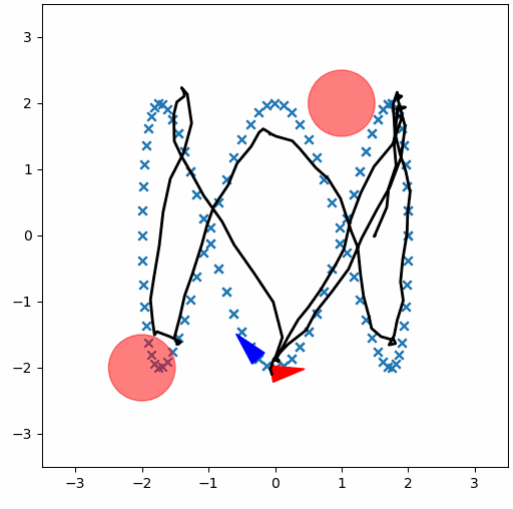

Fig. 4. CEC with obstacle avoidance

C. GPI future improvements

First I would like to get the obstacle avoidance working perhaps by implementing a time horizon greater than 1 to allow the algorithm to prepare for avoiding the obstacle. I would also like to further vectorize the code as it currently has a runtime of about 3 minutes.

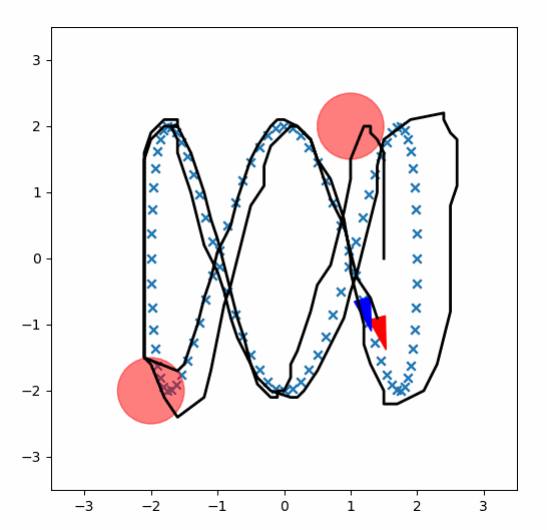

Fig. 5. GPI with no obstacle avoidance

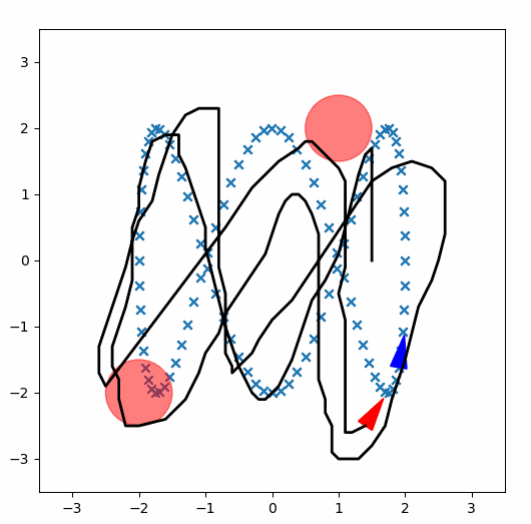

Fig. 6. GPI with obstacle avoidance

D. CEC vs GPI

In this project CEC appeared to run much quicker. Although this could be contributed to the powerful library that was used to solve this part. CEC also appeared to follow the path better except at the y extremes. If I had to spend time to improve one I would like to improve the CEC algorithm as this should result in a much more accurate result.

ACKNOWLEDGMENT

Thank you Professor Atanasov as well as Thai. I appreciate all the effort you put into making this project. I learned a lot and had a great time solving this problem.

REFERENCES

[1] N. Atanasov, “ECE 276B PR3 PDF” in ECE276B, 08-June-2021.