ECE 276B PR3: Infinite-Horizon Stochastic Optimal Control

Varun Vupparige
Mechanical and Aerospace Engineering
University of California San Diego
La Jolla, USA

I. Introduction

With ever increasing use of robots in the different fields such as aviation, marine, automotive, healthcare, and military exploration, the human-robot interaction has increase exponentially. An autonomous system consists of various subsystems such as sensing, perception, path planning and control systems, and the study of trajectory tracking control has been an important research topic. The control task is to enforce the system state to follow a prescribed desired trajectory as closely as possible. One of the major challenges on the way to obtaining robot autonomy is trajectory tracking control which is capable of producing an optimal control input and thus an optimal trajectory which is safe, reliable and obstacle-free.

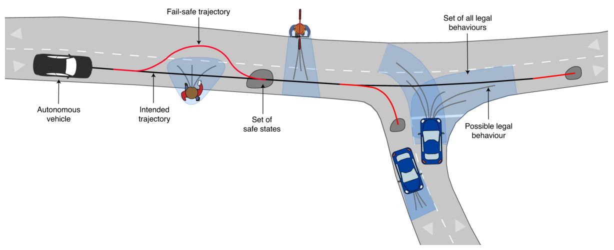


Fig. 1. Trajectory Tracking in an Autonomous car

Fig 1 shows different scenarios where the autonomous car has to track a trajectory in different traffic or road conditions. Thus, these problems are widely applied in different contexts and varying operational design domains. It is extremely important to make sure that the agent is navigating as intended, thus much research has be done on their motion control such as path-following control, trajectory tracking control, stabilization control and formation control. Few of the popular trajectory tracking techniques include back-stepping approach, sliding mode control, adaptive control, and state feedback method. Recently there also been increasing adoption of deep learning techniques to solve these problems.

In this project, the safe tracking of a ground differential robot is considered which has stochastic motion model by formulating it has optimal control problem.

II. Problem Formulation

In this section, the problem of trajectory tracking for a ground differential-drive robot is formulated as a infinite-horizon stochastic optimal control problem.

Consider a robot whose state $x_t := (p_t,\ \theta_t)$ consists of its (x, y) position $p_t \in R^2$ and orientation $\theta \in [-\pi, \pi]$. If we can imagine the environment as a discretized space, then each grid has a position as defined by p_t and can have any orientation $\theta \in [-\pi, \pi]$. Now, lets defined the control space U of the ground differential robot. The robot is controlled by a linear velocity input $v_t \in [0, 1]$ and angular velocity input $w_t \in [-1, 1]$.

Based on the definition of state space X and control space U, we can now define the motion model. Given a robot's state x_t, and control input u_t, the state x_{t+1} can be modelled as a probabilistic/deterministic function.

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

The discrete-time kinematic model of the differential-drive robot obtained from Euler discretization of the continuous-time kinematics with time interval ¿ 0 is:

$$\mathbf{x}_{t+1} = \underbrace{\begin{bmatrix} \mathbf{p}_t \\ \theta_t \end{bmatrix}}_{\mathbf{x}_t} + \underbrace{\begin{bmatrix} \Delta\cos\left(\theta_t\right) & 0 \\ \Delta\sin\left(\theta_t\right) & 0 \\ 0 & \Delta \end{bmatrix}}_{\mathbf{G}(\mathbf{x}_t)} \underbrace{\begin{bmatrix} v_t \\ \omega_t \end{bmatrix}}_{\mathbf{u}_t} + \mathbf{w}_t \tag{2}$$

where, $w_t \in R^3$ is the Gaussian zero-mean white noise with standard deviation σ = [0.04,0.04,0.004] $\in R^3$. Thus, the motion model can be interpreted as a probability density function $p_f(x_{t+1}|x_t, u_t)$ of state x_{t+1} conditioned on x_t and u_t.Furthermore, in eqn (2) δ is the discrete time step, θ_t is the orientation of the robot at time t.

Fig 2 shows the environment with obstacles (red circles), agent(red triangle marker), blue dots(ref trajectory). The environment of the agent is a two dimensional space and the free space can be defined as $\mathcal{F} := [-3, 3]^2 \backslash (\mathcal{C}_1 \cup \mathcal{C}_2)$. Obstacles $C1$ and $C2$ are 0.5m radius circular obstacles centered at (-2,2) and (1,2)

In this project we are required to design a control policy for the robot differential-drive kinematic to track the reference trajectory as indicated by the blue markers in fig 2. Lets define the position of the reference trajectory to be as $r_t \in R^2$ and orientation of the reference trajectory to be $\alpha_t \in R^2$. Since, we are trying to track the reference trajectory and that is equivalent to minimising difference between the reference trajectory and

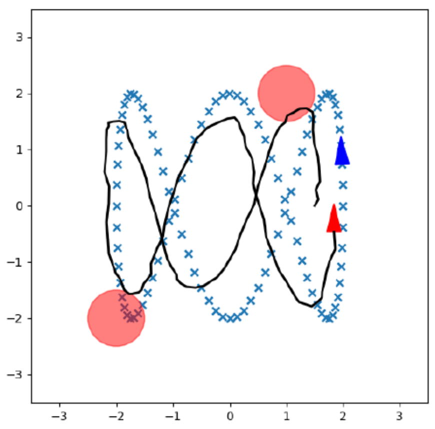


Fig. 2. Environment, ref Trajectory and agent

actual robot trajectory, it would be logical to define the error state and its dynamics. Let $e_t := (\tilde{p_t}, \tilde{\theta_t})$, where $\tilde{p_t} = p_t - r_t$ and $\tilde{\theta_t} = \theta_t - \alpha_t$ measure the position and orientation deviation from the reference trajectory respectively. Similar to motion model, we can define the error dynamics $g(t, e_t, u_t, w_t)$ as follows:

$$g(t, e_t, u_t, w_t) = e_t + G(\tilde{e_t})(u_t) + e_t + 1 + w_t \qquad (3)$$

where $G(e_t) = \begin{bmatrix} \Delta \cos\left(\tilde{\theta}_t + \alpha_t\right) & 0 \\ \Delta \sin\left(\tilde{\theta}_t + \alpha_t\right) & 0 \\ 0 & \Delta \end{bmatrix}$

$e_{t+1} = \begin{bmatrix} \mathrm{r}_t - \mathrm{r}_{t+1} \\ \alpha_t - \alpha_{t+1} \end{bmatrix}$

Thus, we formulate the trajectory tracking with initial time τ and initial tracking error e as a discounted infinite horizon optimal control problem:

mathtools amsmath

$$V^*(\tau, \mathbf{e}) = \min_{\pi} \mathbb{E}\left[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} \left(\tilde{\mathbf{p}}_t^\top \mathbf{Q} \tilde{\mathbf{p}}_t + \right.\right. \qquad (4)$$
$$\left. q\left(1 - \cos\left(\tilde{\theta}_t\right)\right)^2 + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t \mid \mathbf{e}_\tau = \mathbf{e}\right]$$

where Q and R are positive definite matrices which define the stage cost for deviating from reference trajectory and stage cost for using excessive control effort. In the next section, we explain two possible approaches to solving this infinite horizon stochastic optimal control problem to track the reference trajectory of ground differential robot.

III. Technical Approach

In this section, the implementation of two different approaches namely receding horizon certainty equivalent control is being discussed.

A. Receding Horizon CEC

Receding Horizon Control, also known as Model Predictive Control has been a popular paradigm for designing control policies for the past two decades especially for deterministic systems with constraints. This general purpose control scheme involves repeatedly solving a optimization problem, using predictions of future costs, disturbances, and constraints over a receding time horizon to compute an optimal control policy. The basic analogy behind REC is:

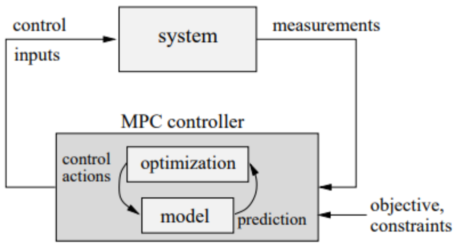


Fig. 3. Illustration of REC/MPC Controller

Lets consider a plant which can be modelled as a linear dynamical system. At time t, the current state of the system is sampled and based on the stage costs, control actions, a cost minimizing policy is computed for next t steps in the future. This particular calculation to compute the cost minimizing strategy is done real time when this particular control scheme is applied on real system. According the sampling frequency of the controller, the state of the plant is sampled again, and the calculations are repeated starting from the current state. Since, the predict horizon keeps shifting time horizon, it is called as receding horizon control.

A ground differential robot has non-linear system dynamics and has noise variables modelled as a Gaussian distribution as explained in the problem formulation section, and the above explained REC method is typically for linear deterministic dynamical system. Since, a non-linear objective function is no longer convex and hence no longer guaranteed to produce a feasible solution,is we need to make certain modifications to the existing optimization problem. Lets define the objective function:

$$V^*(\tau, \mathbf{e}) = \min_{\mathbf{u}_\tau, \ldots, \mathbf{u}_{\tau+T-1}} \mathrm{q}\left(\mathbf{e}_{\tau+T}\right) +$$
$$\sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} \left(\tilde{\mathbf{p}}_t^\top \mathbf{Q} \tilde{\mathbf{p}}_t + q\left(1 - \cos\left(\tilde{\theta}_t\right)\right)^2 + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t\right) \qquad (5)$$

As clearly evident, the objective function formulated is non-linear and stochastic. For such systems, Certainty Equivalent Control (CEC) is implemented. This control scheme computes a sub-optimal policy that applies a control which is optimal when the noise variables are assumed to be zero. Hence, it does not guarantee an optimal path but a sub-optimal one. The certainty equivalence principle states that: a model of the system is fit by observing its time evolution, and a control policy is then designed by treating the fitted model as the truth. Thus, the stochastic optimal control problem is converted to a deterministic optimal control problem. The receding horizon CEC problem can be treated as non-linear program of the form:

$$\begin{array}{ll} \min_{\mathbf{U}} & c(\mathbf{U}, \mathbf{E}) \\ \text{s.t.} & \mathbf{U}_{lb} \leq \mathbf{U} \leq \mathbf{U}_{ub} \\ & \mathbf{h}_{lb} \leq \mathbf{h}(\mathbf{U}, \mathbf{E}) \leq \mathbf{h}_{ub} \end{array} \tag{6}$$

In this project we use an optimization solver called as CasADi to compute the trajectory tracking control policy. Now, lets discuss the details of implementation of this optimization problem in python.

Algorithm 1 Receding-Horizon CEC

```
Require: E, U to be initialized as a optimization variable
  $v \leftarrow 0$
  for $i \leftarrow 1$ to $N$ do
      $v \leftarrow v + \gamma^i(\tilde{\mathbf{p}}i^\top \mathbf{Q}\tilde{\mathbf{p}}i + q(1-\cos(\tilde{\theta}i)^2 + \mathbf{u}i^\top \mathbf{R}\mathbf{u}_i)$
      $E_{i+1} \leftarrow E_i + \tilde{G}Ei.u_i + e_{t+1} + w_t$
      $Constraint_1(\tilde{\mathbf{p}}i + \mathbf{r}i \in \mathcal{F})$
      $Constraint_2(||\tilde{\mathbf{p}}_i - \mathbf{C}1||_2 > 0.5)$      ▷ Avoid Circle 1
      $Constraint_3(||\tilde{\mathbf{p}}_i - \mathbf{C}2||_2 > 0.5)$      ▷ Avoid Circle 2
  end for
  $Constraint_4(\mathbf{U} \in \mathcal{U})$
  $minimize(v)$
  $solve()$
  return U
```

We set the planning horizon T to be some initial value. This value determines how far the horizon is traversed to compute a cost minimising open loop control strategy. We define the error estimate as a function which takes previous error e_t, control policy u_t and computes the estimate of next error e_{t+1}. The equation governing this function is specified in eqn (3) for reader's reference. In the main controller function, we initialize the error from time t to time $t+T+1$, e_t as an optimization decision variable of matrix type and control from time t to time $t+T$, u_t as an another optimization variable.

The syntax as specified by the casADi helper docs is as follows: x = opti.variable(). After defining the decision variables, we initialize the value function to be zero. Then we compute the value function of next T steps by using the error estimate function. To each of these iteration, we impose the constraints of free space as specified by constraints 1, 2 and 3 in psuedo algorithm above. After computing the summation of value function over T steps, we add the terminal cost to it.

Thus, we have computed the objective function which needs to be minimised. Furthermore, we define the remaining controls constraints for linear velocity and angular velocity inputs and solve this formulated to optimization problem. The solved variables are the single step control input. The same process is repeated again by sampling the next state of the robot.

B. Generalized Policy Iteration

Generalized policy iteration is a policy iteration algorithm assuming that estimate of value function converges in finite steps in the policy evaluation part. Namely, it approximates infinite-horizon problem by finite horizon problem as receded-horizon CEC does. The main drawback of GPI algorithm is time-complexity. To tackle this,the motion of reference point at sufficiently small time step is assumed to be negligible

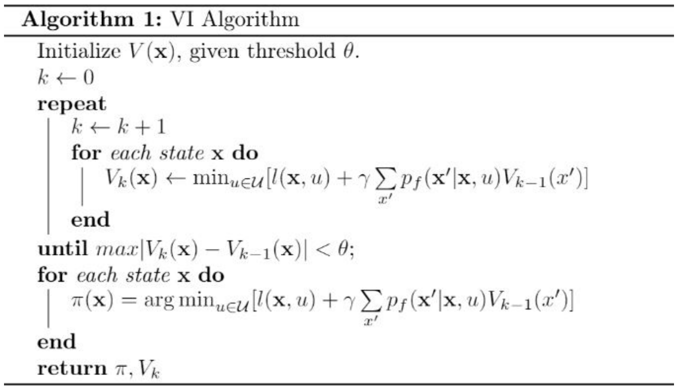
1) Discretization: As the above problem is continuous, the state and the control space has to be to discretized. The state is represented by: -

- $\tilde{p}_{x,dis}$: Discretized x-coordinate of position deviation $\tilde{\mathbf{p}}$
- $\tilde{p}_{y,dis}$: Discretized y-coordinate of position deviation $\tilde{\mathbf{p}}$
- $\tilde{\theta}_{dis}$: Discretized orientation deviation
- $\tilde{\alpha}_{dis}$: Discretized reference point angle

Let n_x, n_y, n_θ, and n_α be number of these discretized state. We set n_x = 21, n_y = 21, n_θ= 11, and n_α = 11. That leads to 21 X 21 X 11 X 11 = 53,361 total states. Each value is continuously split-ted i.e $\tilde{p}_{x,dis} \in [-3, 3]$ into 21 parts = $(-3, -2.7, ...3)$. For θ a wrap-around is implemented to enforce it stays between $[-\pi, \pi)$. The policy of the robot is also discretized. $u = [v, \omega]^T$ to 4 v_{dis} and 4 ω_{dis} making it a 4 x 4 = 16 action space.

2) Value Iteration: Value Iteration starts from the end and then works backwards (Dynamic Programming), although there is no real terminal. VI utilizes V_0 and updates V for convergence. The algorithm for value iteration follows as show in Algorithm 2. There is a thresh hold implemented to terminate the algorithm.

$$V_k(\mathbf{x}) \leftarrow \min u \in \mathcal{U} \left[l(\mathbf{x}, u) + \gamma \sum x' p_f\left(\mathbf{x}' \mid \mathbf{x}, u\right) V_{k-1}\left(x'\right)\right]$$



Algorithm 1: VI Algorithm

```
Initialize $V(\mathbf{x})$, given threshold $\theta$.
$k \leftarrow 0$
repeat
    $k \leftarrow k+1$
    for each state x do
        $V_k(\mathbf{x}) \leftarrow \min_{u\in\mathcal{U}}[l(\mathbf{x},u) + \gamma \sum_{x'} p_f(\mathbf{x}'|\mathbf{x},u)V_{k-1}(x')]$
    end
until $max|V_k(\mathbf{x}) - V_{k-1}(\mathbf{x})| < \theta$;
for each state x do
    $\pi(\mathbf{x}) = \arg\min_{u\in\mathcal{U}}[l(\mathbf{x},u) + \gamma \sum_{x'} p_f(\mathbf{x}'|\mathbf{x},u)V_{k-1}(x')]$
end
return $\pi, V_k$
```

IV. Results

In this section, we will report the results obtained from running various experiments to tune the hyper-parameters, γ, T, x_{start}, y_{start} and θ_{start}.

GIF visualization of few environments can found in the following google drive link: click here

The table summarises the different simulations performed to check the efficiency of the receding horizon CEC using the

Iter	γ	T	Q	r	R	x_star	y_start	error
1	0.9	3	100	50	8	1.5	0	92.516
2	0.9	5	100	50	8	1.5	0	108
3	0.9	7	100	50	8	1.5	0	110.57
4	0.9	10	100	50	8	1.5	0	111.96
5	0.9	3	1	50	8	1.5	0	364
6	0.9	3	100	1	8	1.5	0	969
7	0.9	3	100	50	1	1.5	0	107
8	0.9	3	300	150	24	1.5	0	94
9	0.95	3	100	50	8	1.5	0	94
10	1	3	100	50	8	1.5	0	96.59
11	0.8	3	100	50	8	1.5	0	88
12	0.9	3	100	50	8	2	0	91.103
13	0.9	3	100	50	8	-2	-2.5	107
14	0.9	3	100	50	8	3	-3	107

casADi opti solver. From iteration 1 to 4, I changed the values of T and observed the accumulated error and simulation time. With increase in T, the accumulated error and average iteration time increased but not in a linear way. Since T determines the how much forward the costs are computed, it makes sense for error and sim time to increase. Then I iterated with different hyper-parameter values.

The parameters Q, R and r are the stage cost of position, stage cost of control inputs and stage cost of orientation respectively. For example, increasing Q significantly would results in controller placing more emphasis on decreasing the positional error compared to other values. We observed similar behaviour where decreasing the parameter would result in more accumulated error of that parameter. Please refer to relevant gifs in the google drive for reference.

Furthermore, I iterated with different values γ and observed no significant performance changes apart from accumulated error which is expected since we are scaling the computed long term cost by parameter which is less than one. Lastly, I iterated with extreme start positions of x and y. During these iterations, in some cases the controller was not able to track the ref trajectory at all, but in some cases it would follow the trajectory afte some initial delay.

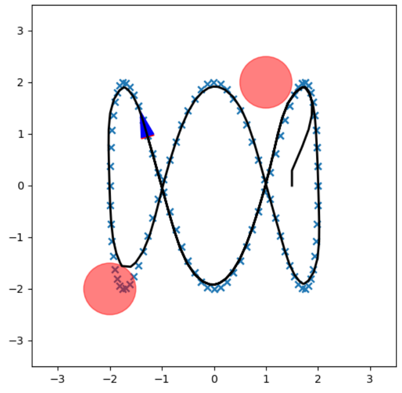


Fig. 4. Illustration of iteration 3

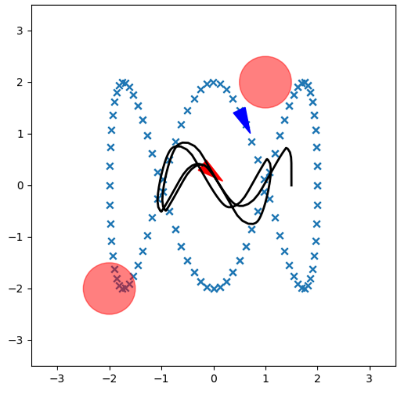


Fig. 5. Illustration of iteration 5

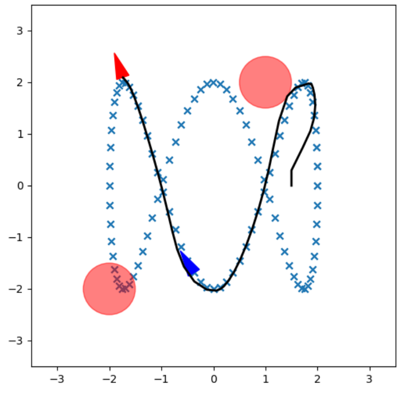


Fig. 6. Illustration of iteration 6

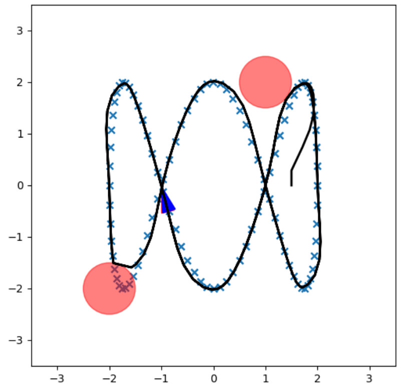


Fig. 7. Illustration of iteration 9

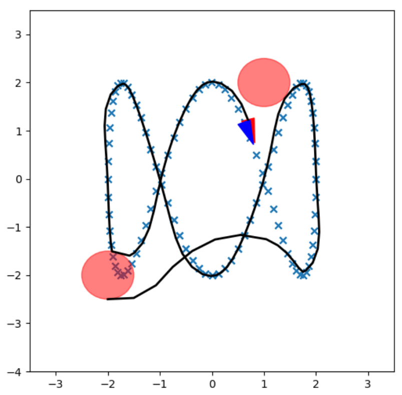


Fig. 8. Illustration of iteration 13

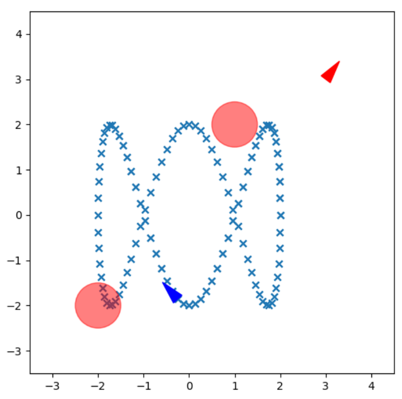


Fig. 9. Illustration of iteration 14