Project Report

June 15, 2024

1 Introduction

Generally speaking, the aim of this project is to design a controller that is able to keep track on a given trajectory of a differential drive. The trajectory is predetermined, while our motion model is stochastic which gave us the main problem of this project.

In part one, an algorithm called receding-horizon certainty equivalent control (CEC) is to be implemented. It is a suboptimal control algorithm with the main idea of repeatedly solving discounted finite horizon deterministic problems to approach an infinite horizon problem.

As for part two, the problem is going to be first being turned into a finite Markov-Decision-Problem (MDP) by discretizing the state space, and then being solved by generalized policy iteration (GPI).
Other useful tools are

- Detexify, which can convert your drawings to LaTeX commands,
- Mathpix tools, which you can use to screenshot an equation and it will convert into LaTeX equation form to be copied into your .tex document,
- Grammarly is a great help for grammar assistance on overleaf.

1.1 Sectioning

This is a subsection. You can also use sub-subsections, although it isn't recommended.

1.1.1 Subsubsection

Example of a sub-subsection. It isn't recommended to nest one more layer. However, if needed, the command is \paragraph{*insert section name*}. It should not be numbered.

1.1.2 Subsubsection

Example of a sub-subsection. It isn't recommended to nest one more layer. However, if needed, the command is \paragraph{*insert section name*}. It should not be numbered.

Finally, make sure to outline the report at the end of the introduction section.

2 Problem Statement

2.1 The Basic Setup

Before getting started, the basic information of the agent and the environment we are dealing with needed to given.

2.1.1 State

The state is composed with the position$P_t := [x_t, y_t]$ and orientation θ_t of the robot agent as follow:

$$x_t = \begin{bmatrix} P_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} \tag{1}$$

with position$P_t \in \mathbb{R}^2$ and orientation $\theta_t \in [-\pi, \pi)$. The time is also being considered in this project, for which we use discrete time $t \in \mathbb{N}$. There are also two circular obstacles inside the state space, respectively defined as C_1 centered at (-2,-2)with radius 0.5 and C_2 centered at (1, 2)with radius 0.5. Therefore, the free space is defined as: $F = [-3, 3]\ (C_1 \cup C_2)$

2.1.2 Control

The control in this problem is composed with linear velocity v and angular velocity (yaw rate) ω: $u_t = [v_t, \omega_t]^T$ And the control space is defined as: $U := [0, 1] \times [-1, 1]$

2.1.3 Motion Model

The motion model is defined as follow:

$$x_{t+1} = \begin{bmatrix} p_{t+1} \\ \theta_{t+1} \end{bmatrix} = f(x_t, u_t, w_t) = \begin{bmatrix} p_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \Delta cos(\theta_t),\ 0 \\ \Delta sin(\theta_t),\ 0 \\ 0 \qquad ,\Delta \end{bmatrix} + \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + w_t, \qquad t = 0, 1, 2, ... \tag{2}$$

The $\omega_t \in \mathbb{N}^3$is the motion noise, with Gaussian distribution $\mathcal{N}(0, diag(\sigma)^2)$ and standard deviation of $\sigma = [0.04, 0.04, 0.004] \in \mathbb{N}^3$. The kinematics model in (2) defines the probability density function $P_f(x_{t+1}|x_t, ut)$ of x_{t+1} conditioned on x_t and u_t as the density of a Gaussian distribution with mean $x_t + G(x_t)u_t$ and covariance diag$(\sigma)^2$.

2.1.4 Reference Trajectory

There is a predetermined reference trajectory with position $r_t \in \mathbb{R}^2$ and orientation $\alpha_t \in [-\pi, \pi)$

2.1.5 Error

The error e_t is defined as: $e_t := (\tilde{p}_t, \tilde{\theta}_t)$, where $\tilde{p}_t := p_t - r_t$ and $\tilde{\theta}_t := \theta - \alpha_t$. Moreover, there is also an error state motion model which help to acquire the position and orientation deviation with respect to the reference trajectory. Defined as:

$$e_{t+1} = \begin{bmatrix} \tilde{p}_{t+1} \\ \tilde{\theta}_{t+1} \end{bmatrix} = g(t, x_t, u_t, w_t) = \begin{bmatrix} \tilde{p}_t \\ \tilde{\theta}_t \end{bmatrix} + \begin{bmatrix} \Delta cos(\tilde{\theta}_t),\ 0 \\ \Delta sin(\tilde{\theta}_t),\ 0 \\ 0 \qquad ,\Delta \end{bmatrix} + \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \begin{bmatrix} r_t - r_{t+1} \\ \alpha_t - \alpha_{t+} \end{bmatrix} + w_t, \quad t = 0, 1, 2, ... \tag{3}$$

2.1.6 Value Function

The value function is defined as:

$$\begin{aligned} V^*(\tau, e) = &\min_{\pi} \mathbb{E}\left[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} \left(\tilde{p}_t^T Q \tilde{p}_t + q(1 - cos(\tilde{\theta}_t))^2 + u_t^T R u_t\right) | e = e_\tau\right] \\ &s.t.\ e_{t+1} = g(t, x_t, u_t, w_t), \quad w_t \sim \mathcal{N}(0, diag(\sigma)^2), t = \tau, \tau + 1, ... \\ &u_t = \pi(t, e_t) \in U \\ &\tilde{p}_t + r_t \in F \end{aligned} \tag{4}$$

with initial time τ and initial tracking error e. $Q \in \mathbb{R}^{2\times2}$ 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 for deviating from the reference orientation trajectory α_t, and $R \in \mathbb{R}^{2\times2}$ is a symmetric positive-definite matrix defining the stage cost for using excessive control effort.

2.2 Part One

As we mentioned, this part is implement by approximating the infinite horizon problem by repeatedly solving discounted finite-horizon deterministic optimal control problem which is defined as:

$$
\begin{aligned}
V^*(\tau, e) \approx \min_{u_{\tau,\dots,u_{\tau+T-1}}} & +q(e_{\tau+T}) + \sum_{t=\tau}^{\tau+T-1} \gamma^{t-\tau} \left(\tilde{p}_t^T Q \tilde{p}_t + q(1 - cos(\tilde{\theta}_t))^2 + u_t^T R u_t \right) \\
s.t.\ & e_{t+1} = g(t, x_t, u_t), \quad w_t \sim \mathcal{N}(0, diag(\sigma)^2), t = \tau, \tau+1, ..., \tau+T-1 \\
& u_t \in U \\
& \tilde{p}_t + r_t \in F
\end{aligned} \tag{5}
$$

Furthermore, this can be also seen as an optimization problem and can be solved by non-linear program, which can be generally formulated as follow:

$$
\begin{aligned}
& \min_u c(U, E) \\
s.t.\ & U_{lb} \leq U U_{ub} \\
& h_{lb} \leq h(U, E) h_{ub}
\end{aligned} \tag{6}
$$

Here the U is defined as the control from step τ to $\tau + T - 1$ and E is defined as the error state from step τ to $\tau + T$. Also the U_{lb} and U_{ub} are respectively defined as the lower bound and the upper bound of the control

2.3 Part Two

In this part, generally there's two things need to be done, discretization and solving the problem with GPI algorithm.

3 Technical Approach

3.1 Part 1

As mentioned in last section, we are going to solve the problem by applying the nonlinear program we defined in (6) In nonlinear program, we are going to minimize the function C(U,E) which in our case defined as(5) We initiate a few terms as follow:

- Discount factor: $\gamma = 1$
- Q and R are both 2×2 identity matrix
- $q = 1$
- terminal cost $q(e\tau + 1)$ is the same form of the stage cost
- T is the time horizon, which we first test it with T = 2. After make sure the algorithm is running we increase it

The way it is implemented, is by stacking all of the variables together in to one vertical vector with he size of $2n \times 1$. There is one thing need to be mentioned that even the function C is composed with both control u and error e from τ to $\tau + T$, the errors are not variable. Even we know nothing about the future error (from step τ to $\tau + T - 1$), we can still express them with respect to u_{t-1} for every e_t, as we can see from the error motion model.

And for the constrain function, we will be using the error motion model, as we already defined previously. In this case, we will have $e_{t+1} - g(t, e_t, u_t, 0)$ for $t = \tau$ to $\tau + T^\smile 1$, and stack them up vertically, equate all of them to 0. However, in code we can only set them in inequality way. So the lower bound and the upper bound would both be 0. About the bound, since the variables are control u_t, the bound would be just the bound of the control space $U := [0, 1] \times [-1, 1]$ as we already discussed. However we also need to stack them together into a big vertical vector, so we can put them into code.

3.2 Part 2

There's few parts needed to be done: discretization, define transition matrix and find the policy.

3.2.1 Discretization

In this case, the discretization would be not only over states but also over controls. The dimension would be (100,10,10,10,5,10) with each of them correspond to (time, number of the gird in x, number of the gird in y, number of the gird in theta,number of the gird in v, number of the gird in ω). Also functions convert the states and control from original coordinate to cells coordinate needs to be define vice versa.

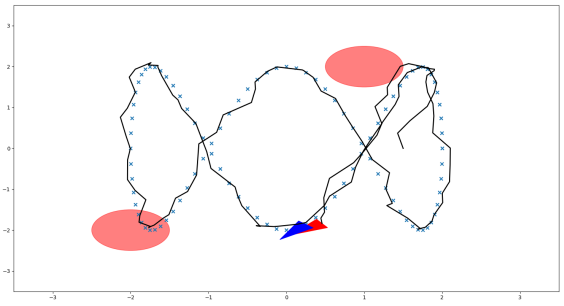

Figure 1: horizon T=2

3.2.2 Define Transition Matrix

The size of the transition matrix would be (100,10,10,10,5,10,6,4) the first six were already defined. The 6 is for the six neighbor we have since there is no diagonal move. Since our motion model is stochastic and even without noise the agent is not guarantee to land exactly on a grid after a movement. Therefore information about the six neighbor of the next state needs to be kept in recorded. As for the 4, is related to the information that store in the matrix, which is the error state and the probability.

Now after we defined the dimension of the transition matrix, we need to fill it with certain information. The way of calculating the error had already been discussed previously, then all we need is to find the probabilities. As we know, the node that probability represent is the one of the neighbors of the result of certain move, and the reason we need this probability is that certain move is not deterministic because we have noise in Gaussian distribution. Therefore, we take a step back, compute the result of the move without the noise,use it as a mean. Then the error can be seen as a deviation from the mean which can be used as a input of the following function to generate a likelihood:

$$\phi(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right) \tag{7}$$

where x is the vector of random variables, μ is the mean vector, Σ is the covariance matrix, n is the dimension of x in this case is 3, and $|\Sigma|$ denotes the determinant of Σ.

However, this likelihood can not straightly using as the probability before it been normalize as follow:

$$\phi_{normalized} = \frac{\phi_i}{\sum_{i=0}^{5} \phi_i} \tag{8}$$

It will be summed up with all of the neighbors of its parent node, and the probability will be generated by divide the likelihood with the sum.

3.2.3 Find The Optimal Policy

Now, with the transition matrix and the discretizaion, we are able to generate the optimal policy pi. Here we are going to use value iteration which is defined as the pesudocode as followed:

The iteration time here is set at 50 times.

4 Results

4.1 Part 1

In the first part we were able to track the given trajectory perfectly. Even there our own trajectory does not look that smooth and still have some small deviation. But, generally we can seen it as working well.

At first we were testing the algorithm with only horizon T=2, we can see here in the figure 1:

As we can see here is not too bad, but with some error at the same time

After we were able to implement with T=2, we start to test the case with higher T. We increase it to 50 in figure2

We can see here it got better at tracking the trajectory, but still not good enough and it did not improve a lot if we compare it to T=2 case. But the good news is, it did not consume much more time compare to the previous case.

After that, we try to implement T = the rest of the time in the horizon for every step. However that is just too time consuming so we did not end up having any result.

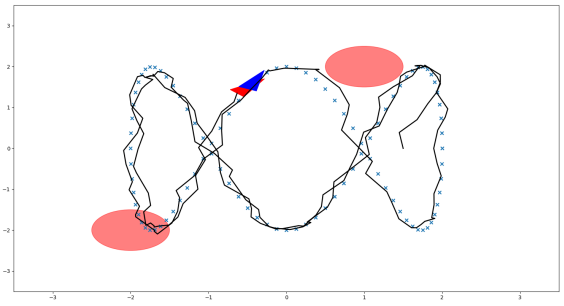

Figure 2: horizon T=50

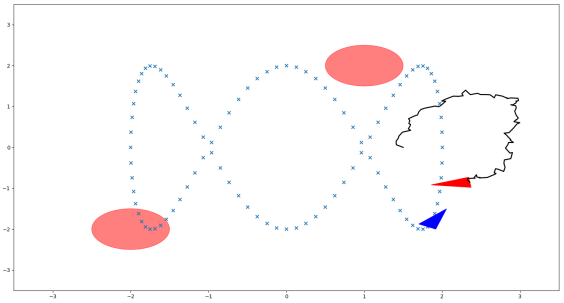

Figure 3: Part 2 result

4.2 Part 2

In this part we were not able to develop an GPI algorithm that can eventually track the robot. Our main problem here is we finding ourselves out side of the state space when we try to find the trajectory from the policy we generated. Moreover, even we used some little tricks to erase those points we were not even able to find one. But, with help of those images, we were able to spot problems. We found that the agent was moving with way smaller steps than the given one, as it shows here in figure 3. As we can see here it barely generate result.

There's one important thing needs to be mentioned which is how we cope with the insanely large state space.

4.2.1 Vectorization

Basically the idea of vectorize a function is to make the input of a function from a scalar to a vector, matrix or even a high dimension tensor. So we can deal with the data all at once. At first, we were inspired by ECE276A project, which when we were using jax, we tried to vecotrize everything to avoid for loop. In this case, we were testing np.vectorize at first since it is a function from numpy library itself that can vectorize functions. However, it did not go well. Then we took a step back, simply just modify the matrix altogether. A simply intuition is we were always trying to make usre the matrix about the same size.

4.2.2 Save Temporary Result

This was inspired by the library joblib. Basically, we save some result to local so there will have enough working memory. For example, there is a function delicate to save the model after I initialize it. This would not only give me enough memory to run it, but also help me when coding and debuging.

4.2.3 Deleting Used up Variable

There is a lot of temporary variable gone useless after we used it. It could take a huge amount of memory if it is with huge amount and could slow down our computation. Therefore,we would just use del to delete those variables.

4.2.4 Other Possible Solution

There are some other possibly feasible way we did not implement or tried but did not eventually use it

- Parallel Computing
  The basic idea of parallel computing is to execute tasks using multiple processors or computational units, such as CPU cores, to achieve faster computation. Can be easily understand as divide large task into smaller subtask and have them being done simultaneously. Libraries like Ray and Joblib probive us with powerful code for parallel computing.

- Jit from Numba
  Jit is a decorator from numba library, which can do parallel computing too. However it is way eaier than those we just mentioned. But at the same time, there's also down side. It can only deal with simple data structure, which means it can not deal with self defined class and its method.That was the reason we did not end uop using it

- Cuda
  Cuda is also a way to do parallel computing. It is a parallel computing platform and application programming interface (API) model created by NVIDIA. We can use it to make our code being parallel compute by powerful GPU, which can be hunderds of times faster.