NYU

TANDON
SCHOOL OF
ENGINEERING

REINFORCEMENT LEARNING AND OPTIMAL CONTROL
FOR ROBOTICS ROB-GY 6323

Project 1

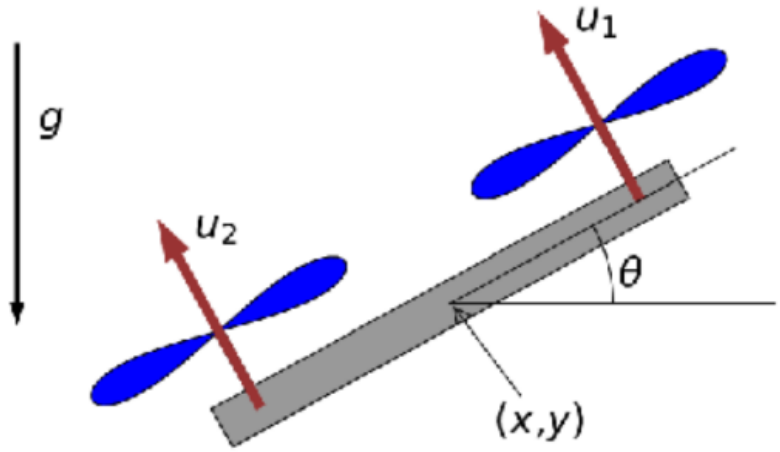
Author:
NAVEEN KUMAR
Student ID: N19068326

Instructor:
Prof. Ludovic Righetti

1 Introduction

In this project, we have been provided a 2D quadrotor and we are supposed to control it using different controllers and in different ways in different parts of the project. In part 1, we just need to calculate u1 and u2 such that the quadrotor will remain stationary at (0,0) position with 0°orientation. We also need to discretize the system. In part 2, we are supposed to write an LQR controller that will be going to keep the quadrotor stationary at (0,0) with 0°orientation. Because it is an LQR so even after the disturbance it comes to the position (0,0) position with 0°orientation. In part 3, the quadrotor is supposed to follow a circular trajectory of unit radius around (0,0) coordinates using LQR controller. In part 4, we have to make the quadrotor go to (3,3) with $\pi/2$ orientation with an optimal trajectory calculated by iLQR. in the second portion of part 4, we have to make a quadrotor flip.

2 Part 1



We have been provided a 2D quadrotor according to the diagram above. u_1 & u_2 are control inputs. x is horizontal and y is vertical position. θ is the orientation of the quadrotor. The quadrotor model is

$$\dot{x} = v_x \tag{1}$$

$$m\dot{v_x} = -(u_1 + u_2)sin\theta \tag{2}$$

$$\dot{y} = v_y \tag{3}$$

$$m\dot{v_y} = (u_1 + u_2)cos\theta - mg \tag{4}$$

$$\dot{\theta} = \omega \tag{5}$$

$$I\dot{\omega} = r(u_1 - u_2) \tag{6}$$

here v_x and v_y are the linear velocities ω is the angular velocity of the quadrotor. u_1 & u_2 are the forces produced by the rotors. m is the quadrotor mass. I is the moment of inertia. r is the distance from the center of the robot frame to the propellers. g is the gravity constant. We are denoting the entire state with $z = [x, v_x, y, v_y, \theta, \omega]^T$ and similarly $u = [u_1, u_2]^T$.

2.1 Part 1.1

In this part, we have to discretize the system dynamics. Below is the discretized dynamics. Here Δt is the discritization step.

$$x_{n+1} = x_n + \Delta t.v_x \tag{7}$$

$$v_{x_{n+1}} = v_{x_n} + \Delta t.(\frac{-(u_1 + u_2)Sin\theta}{m}) \tag{8}$$

$$y_{n+1} = y_n + \Delta t.v_y \tag{9}$$

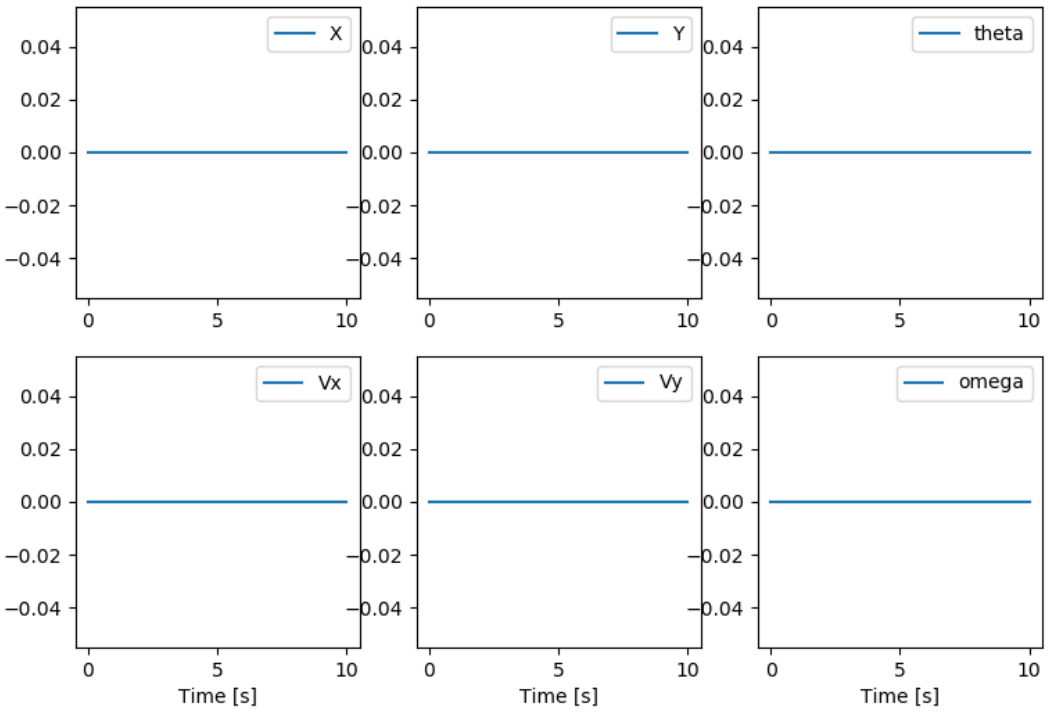
$$v_{y_{n+1}} = v_{y_n} + \Delta t.(\frac{(u_1 + u_2)Cos\theta - mg}{m}) \tag{10}$$

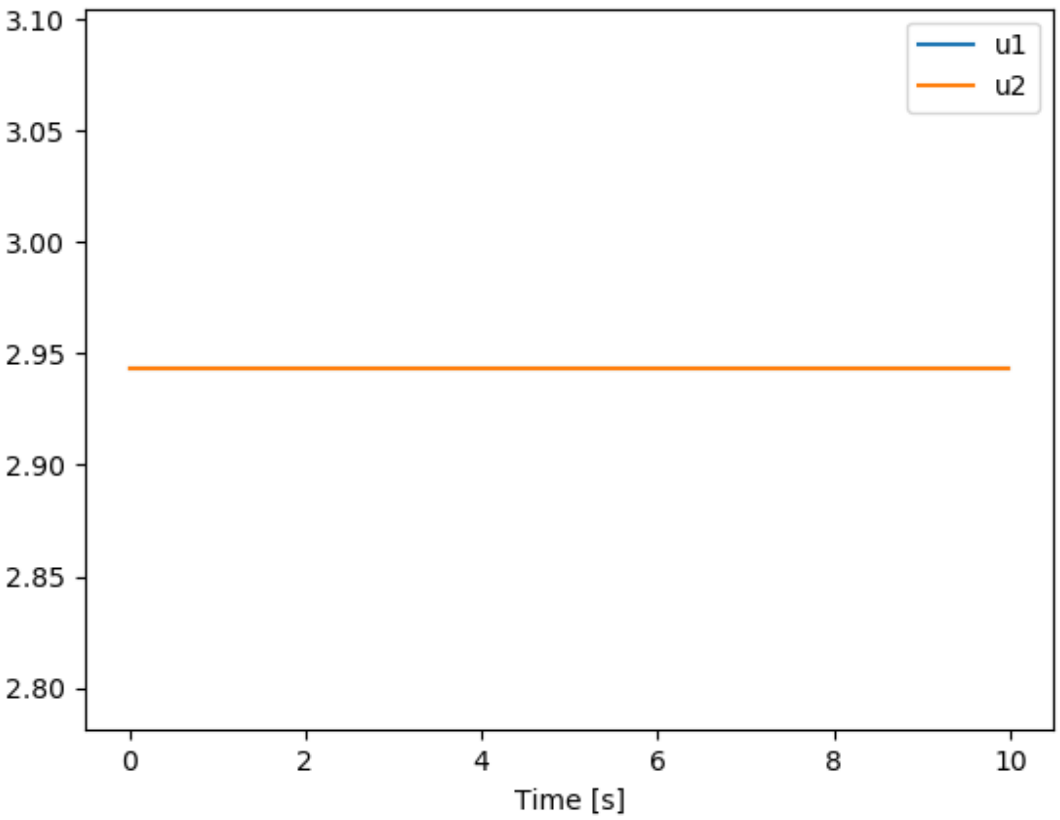
$$\theta_{n+1} = \theta_n + \Delta t.\omega \tag{11}$$

$$\omega_{n+1} = \omega_n + \Delta t.(\frac{r(u_1 - u_2)}{I}) \tag{12}$$

2.2 Part 1.2

We have to make robot starts at (0,0) with 0°orientation with 0 velocities and u_1^* and u_2^* should be such that the robot should be at this position without disturbance. So in this case u_1^* and u_2^* are going to be equal. From 4, we can say $u_1 + u_2 = mg$ and from 6 $u_1^* = u_2^*$ So, it should be $u_1^* = u_2^* = mg/2$ to make it hover.


X
Y
theta
Vx
Vy
omega
0.04
0.02
0.00
−0.02
−0.04
0
5
10
Time [s]



2.3 Part 1.3

When it comes to the system dynamics, it is not possible to move in x direction with keeping $\theta = 0$ initially but if anyhow we provide some accelaration in x direction, it will start moving towards x but in real world it is not possible as from 2 we can see that in order to give some accelaration in x direction, we should have some θ, to give it as an initial accelaration which can increase speed of quadrotor in x direction and it can move in the simulated world with $\theta = 0$ because of no losses but we need to provide some accelaration initially by tilting quadrotor to move it to the direction along x where it needs to go, but real world, it need continuous force to move in x because of friction and other losses.

2.4 Part 1.4

When it comes to the system dynamics, it is not possible to keep quadrotor at rest at $\theta = \pi/2$ as from equation 4 we can see that if $\theta = \pi/2$ the cos term will become 0, which causes the quadrotor to fall as no force will be acting to make it flying both u_1 and u_2 are parpendicular to the y direction.

Part 2

2.1 we have been given arbitrary operating point z^* and u^* and we have to linearize the dynamics using $\bar{z}_n = z_n - z^*$ and $\bar{u}_n = u_n - u^*$.

let $z_{n+1} = f(z_n, u_n)$

$$f(z) \approx f(z_n^*) + \frac{\partial f}{\partial z}\Big|_{z=z^*} (z - z^*)$$

$$z_{n+1} = f(z_n^*, u_n^*) + \frac{\partial f}{\partial z}\Big|_{\substack{z=z^* \\ u=u^*}} \cdot (z_n - z^*) + \frac{\partial f}{\partial u}\Big|_{\substack{z=z^* \\ u=u^*}} \cdot (u_n - u^*)$$

here $\left.\frac{\partial f}{\partial z}\right|_{\substack{z=z^* \\ v=v^*}} = A$

$\left.\frac{\partial f}{\partial v}\right|_{\substack{z=z^* \\ v=v^*}} = B$

$\bar{z}_n = z_n - z^*$

$\bar{v}_n = v_n - v^*$

After putting these values in above equation.

$$\bar{z}_{n+1} \approx A\bar{z}_n + B\bar{v}_n$$

here $A = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -\frac{\Delta t(u_1+u_2)\cos\theta}{m} & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & -\frac{\Delta t(u_1+u_2)\sin\theta}{m} & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

$$B = \begin{bmatrix} 0 & 0 \\ -\frac{\Delta t \sin\theta}{m} & -\frac{\Delta t \sin\theta}{m} \\ 0 & 0 \\ \frac{\Delta t \cos\theta}{m} & \frac{\Delta t \cos\theta}{m} \\ 0 & 0 \\ \Delta t \frac{\gamma}{I} & -\Delta t \frac{\gamma}{I} \end{bmatrix}$$

A & B depends on θ, u_1, and u_2 by seeing them

u_1 and u_2 are approximately equals to $\frac{mg}{2}$ and $\theta \approx 0$

So equation is going to be quite similar at every point.

Which means there are not many change in orientation and it is going to keep quadrotor stable at the point.

Part 2.3

Infinite horizon LQR

$$\min_{U_n} \sum_{n=0}^{\infty} (z_n - z^*)^T Q (z_n - z^*) + (U_n - U^*)^T Q (U_n - U^*)$$

Subject to

$$\bar{z}_{n+1} = A\bar{z}_n + B\bar{U}_n$$

so controller will be

while (True)

$$P = Q + A^T P A - A P B (B^T P B + R)^{-1} B^T P A$$

$$K = -(B^T P B + R)^{-1} B^T P A$$

$$\bar{U} = K\bar{z}$$

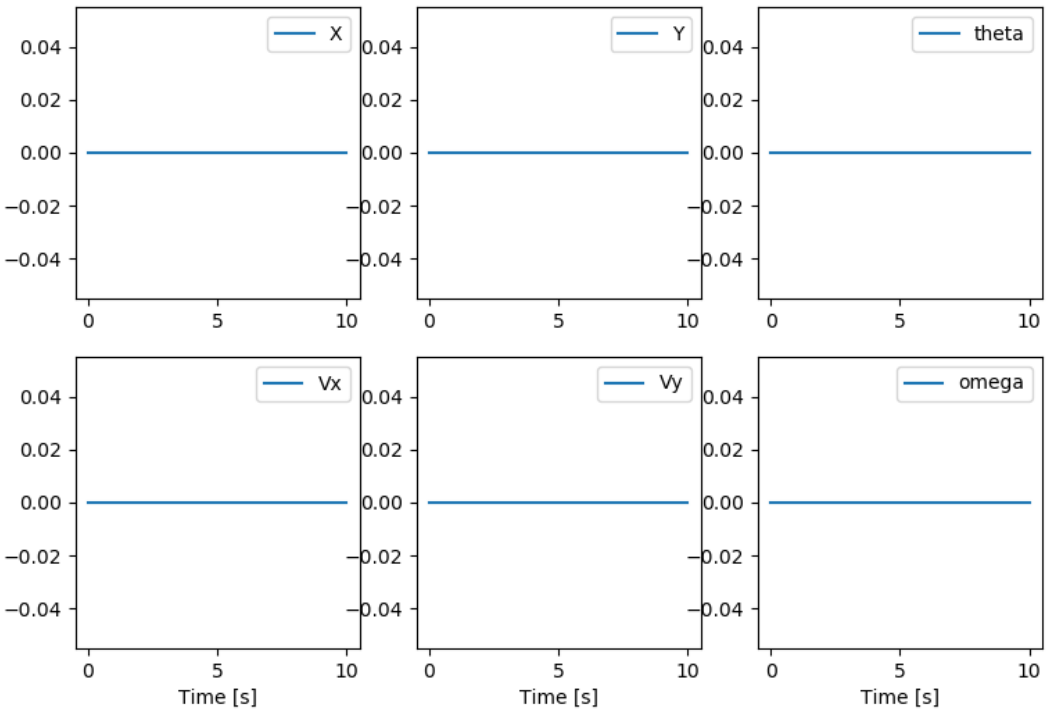
$$U_n = K\bar{z} + U^*$$

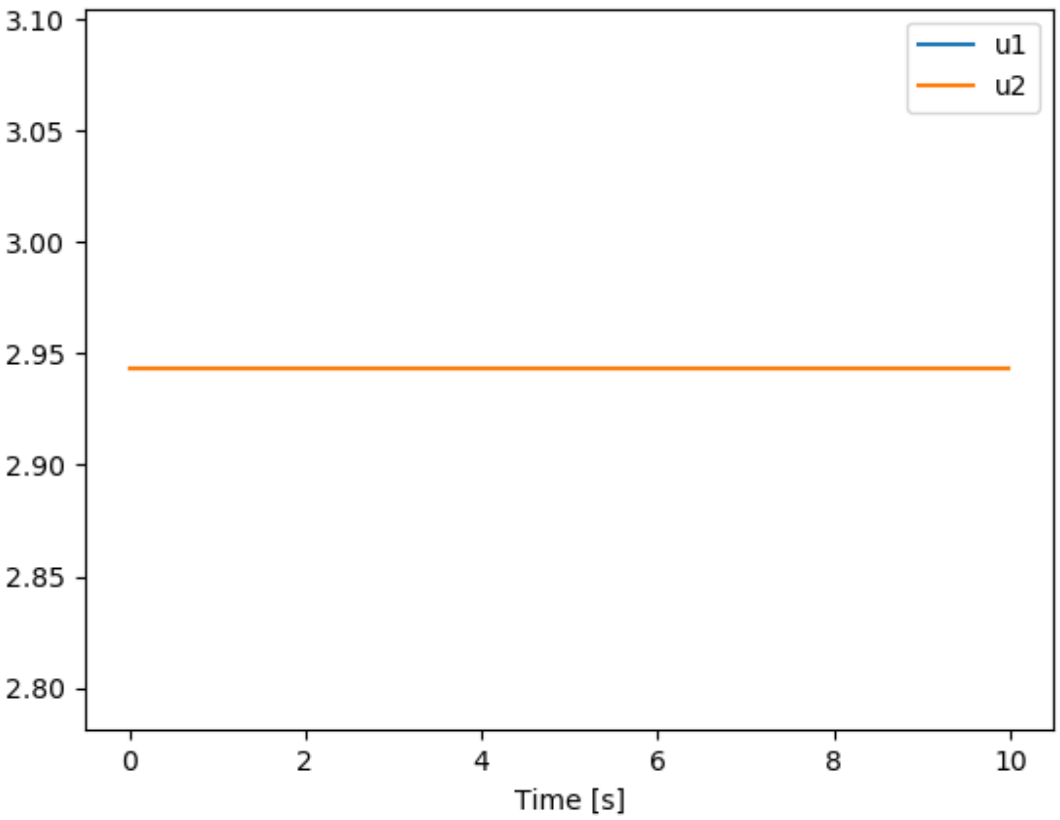
Above is the design of the infinite horizon controller. both the cost to go and the feedback gain are independent of n (i.e. constant for all stages). The control and cost-to-go are “stationary”.

Part 2.4

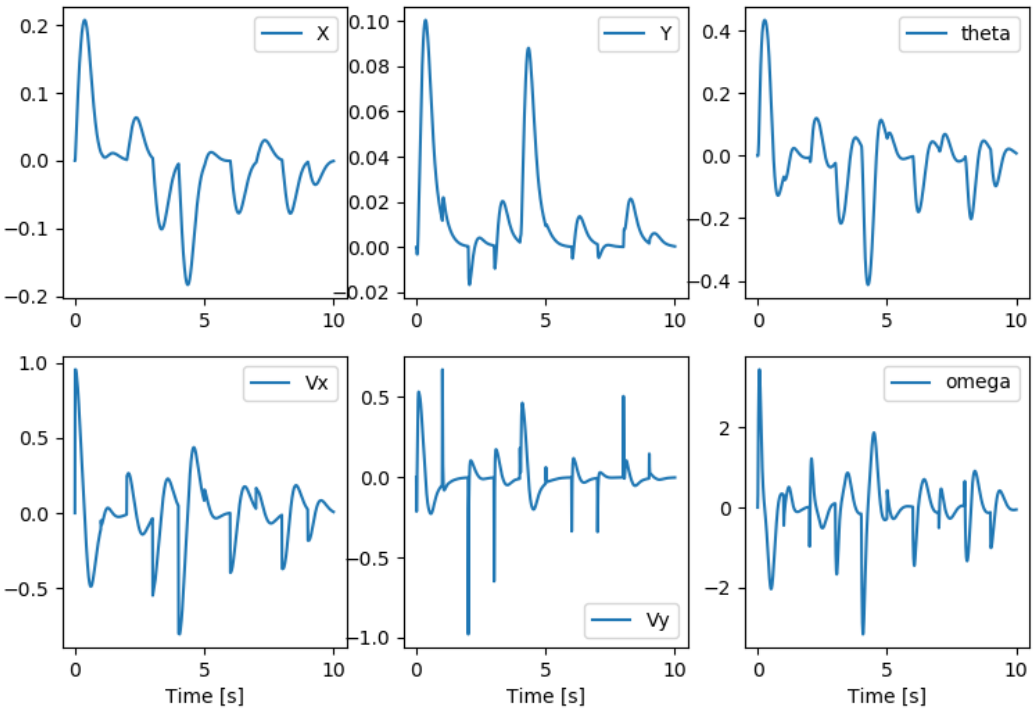
The equations are as same as mentioned in Part 2.3. Here $\overline{Z}$ is Z_n - Z^*, Z^* is [0,0,0,0,0,0] and Z_n is the current state.

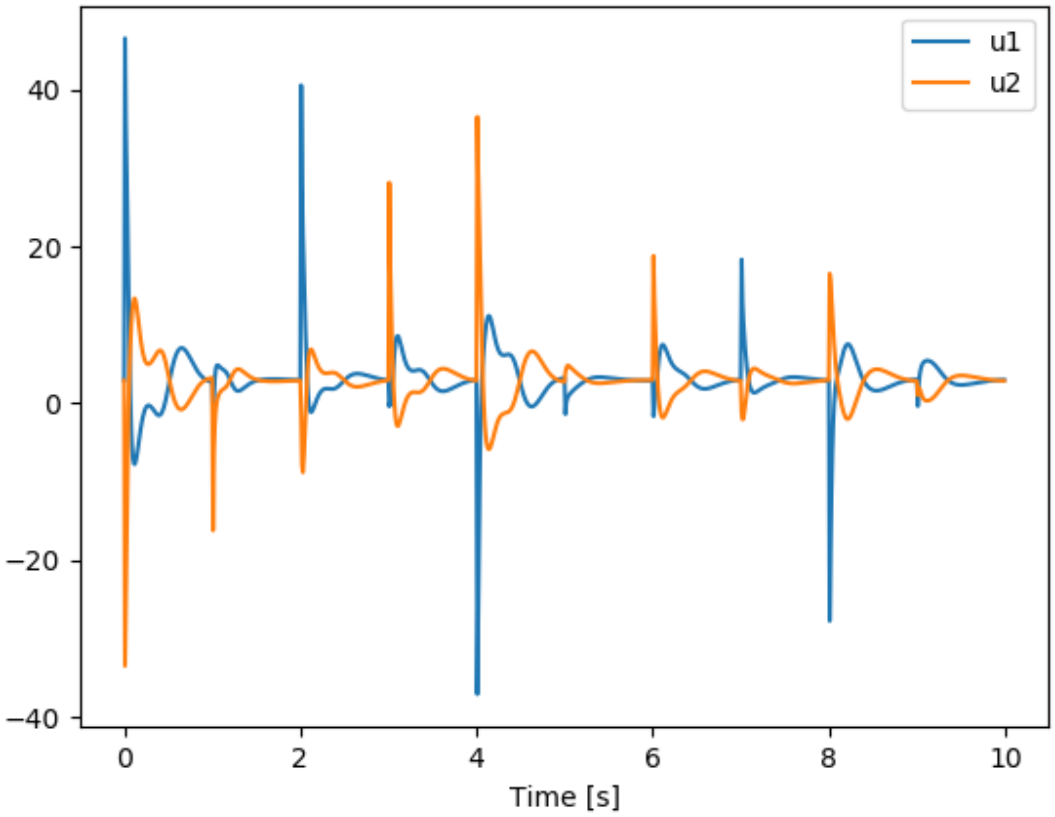
Plots - During no turbulence.





Plots with Turbulance


X
Y
theta
Vx
Vy
omega
Time [s]
Time [s]
Time [s]



So we can see that the controller is working fine to keep the quadrotor at (0,0) position with 0° orientation.

Part 3

Part 3.1

Part 3.1

the equation of circle is

$x = r\cos\frac{2\pi}{T}t$ here $T = 10$

$y = r\sin\frac{2\pi}{T}t$

$V_x = \dot{x} = \frac{dx}{dt} = -\frac{2\pi}{T}r\sin\frac{2\pi}{T}t$

$V_y = \dot{y} = \frac{dy}{dt} = \frac{2\pi}{T}r\cos\frac{2\pi}{T}t$

$\theta = 0$

$\omega = 0$

We have to linearize dynamics at every point of the trajectory

When it comes to the linearization of the dynamics, we can see that in A and B, it depends on θ, u_1 and u_2 in order to keep θ = 0, u_1 and u_2 will be approximately equal to mg/2 and there are also not much variations in θ from the plots so, equation is going to approximately same at every point but we have a different value of $\bar{Z}$ at every point.

Part 3.2

I have used below equations to derive the trajectory tracking controller.

Linear - Quadratic tracking problems

What if I want the system to move along a certain path?

$$\min \frac{1}{2}(x_N - \bar{x}_N)^T Q_N (x_N - \bar{x}_N) + \sum_{n=0}^{N-1} \frac{1}{2}(x_n - \bar{x}_n)^T Q_n (x_n - \bar{x}_n) + \frac{1}{2} u_n R_n u_n$$

where $\bar{x}_n$ is a desired trajectory to follow

Is equivalent to solving

$$\min \frac{1}{2} x_N^T Q_N x_N + q_N^T x_N + \sum_{n=0}^{N-1} \frac{1}{2} x_n^T Q_n x_n + q_n^T x_n + \frac{1}{2} u_n R_n u_n$$

where $q_n = -Q_n \bar{x}_n$

$$\min \frac{1}{2} x_N^T Q_N x_N + q_N^T x_N + \sum_{n=0}^{N-1} \frac{1}{2} x_n^T Q_n x_n + q_n^T x_n + \frac{1}{2} u_n R_n u_n$$

subject to $x_{n+1} = A_n x_n + B_n u_n$

Initialize $P_N = Q_N \qquad p_N = q_N$

Iterate from N-1 to 0

$$K_n = -(R_n + B_n^T P_{n+1} B_n)^{-1} B_n^T P_{n+1} A_n$$

$$P_n = Q_n + A_n^T P_{n+1} A_n + A_n^T P_{n+1} B_n K_n$$

$$k_n = -(R_n + B_n^T P_{n+1} B_n)^{-1} B_n^T p_{n+1}$$

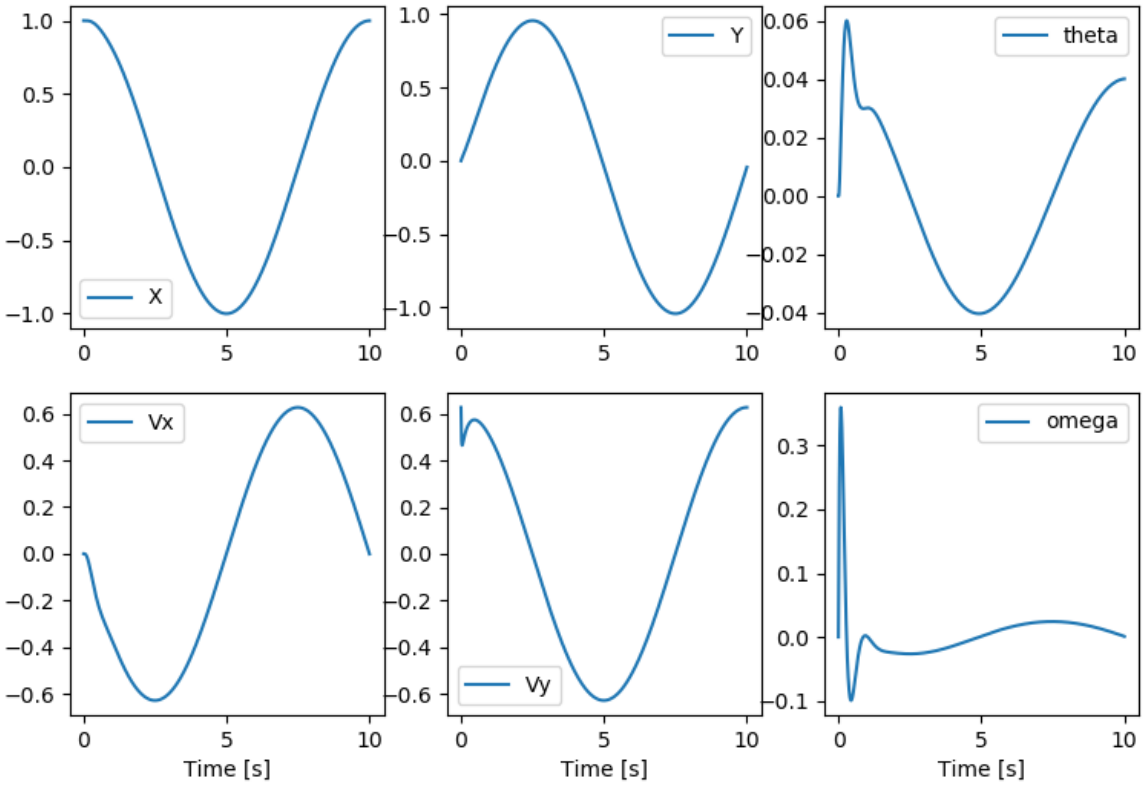
$$p_n = q_n + A_n^T p_{n+1} + A_n^T P_{n+1} B_n k_n$$

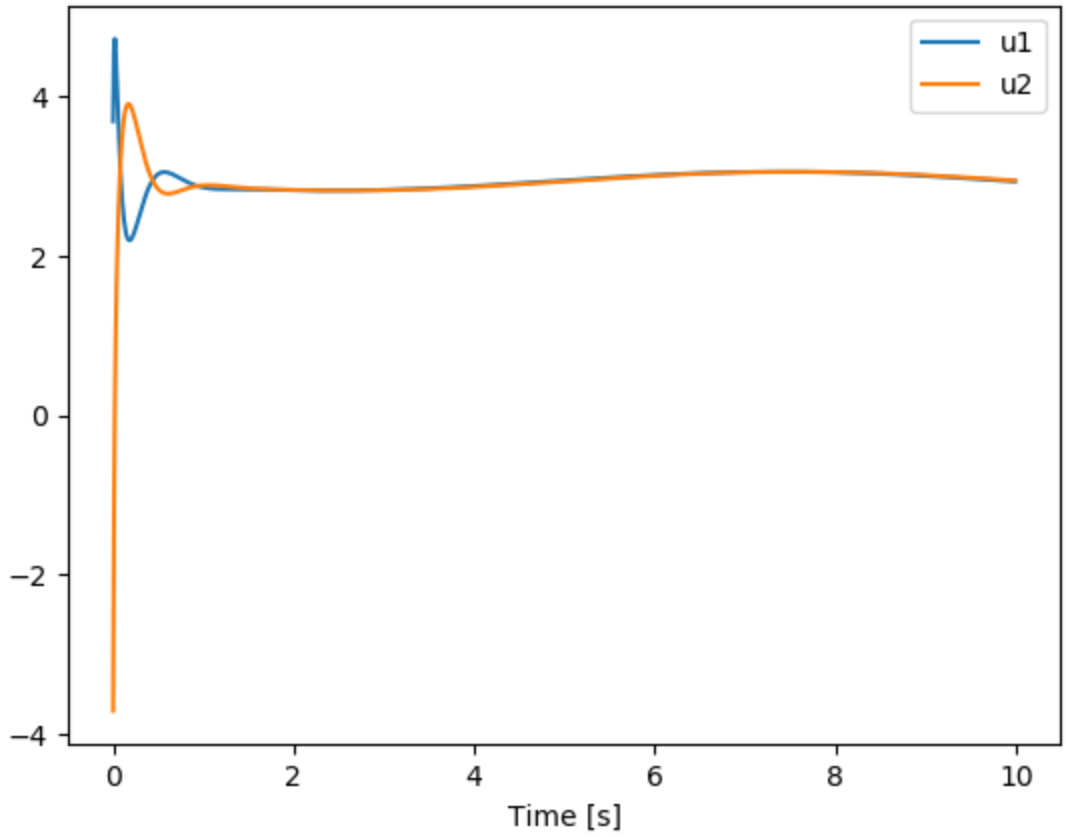
Optimal policy $\mu_n^*(x_n) = K_n x_n + k_n$

linear feedback + feedforward

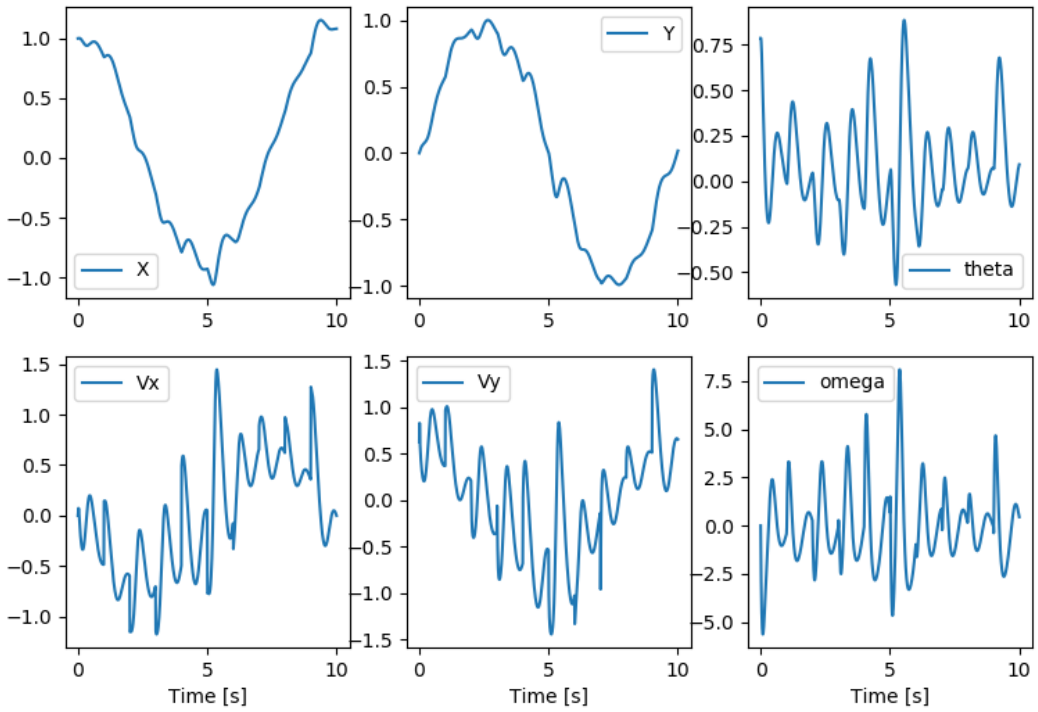
I have used Backward riccati and was able to found out values of K and k which I directly put in the controller and it worked.

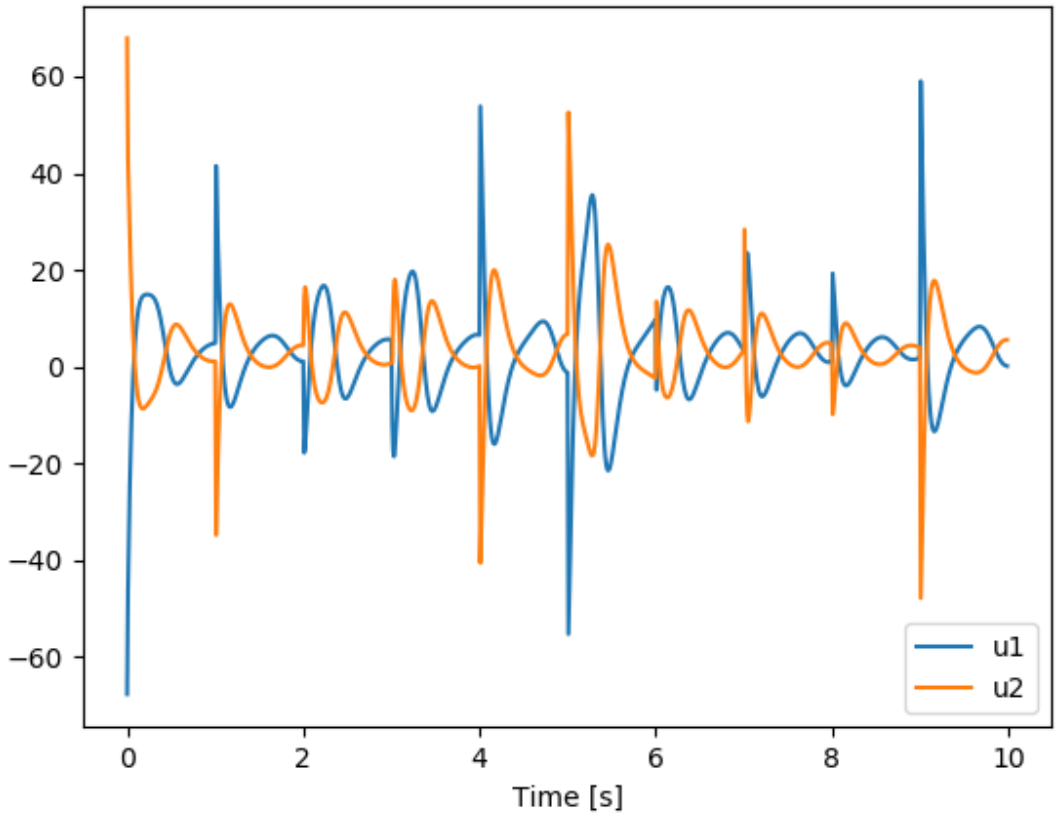
These are the plots





With Disturbance


X
Y
theta
Vx
Vy
omega
Time [s]
Time [s]
Time [s]



Part 3.3

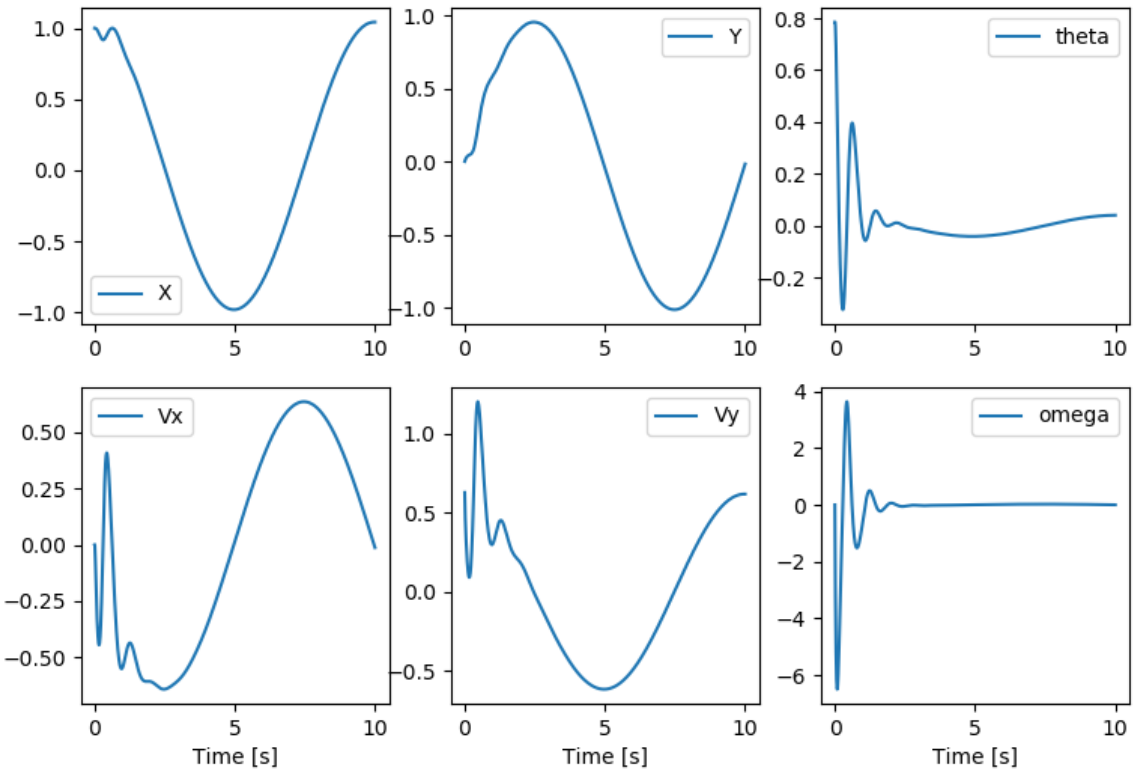
From the simulation, I was able to track the circular trajectory. Along with that, we can also see there are very small variations in the θ. We are quite close to 0 but in order to move in x direction, drone needs to somewhat tilt. So, controller tried to keep θ = 0 but we also need quadrotor to go in x direction so it is not completely zero.

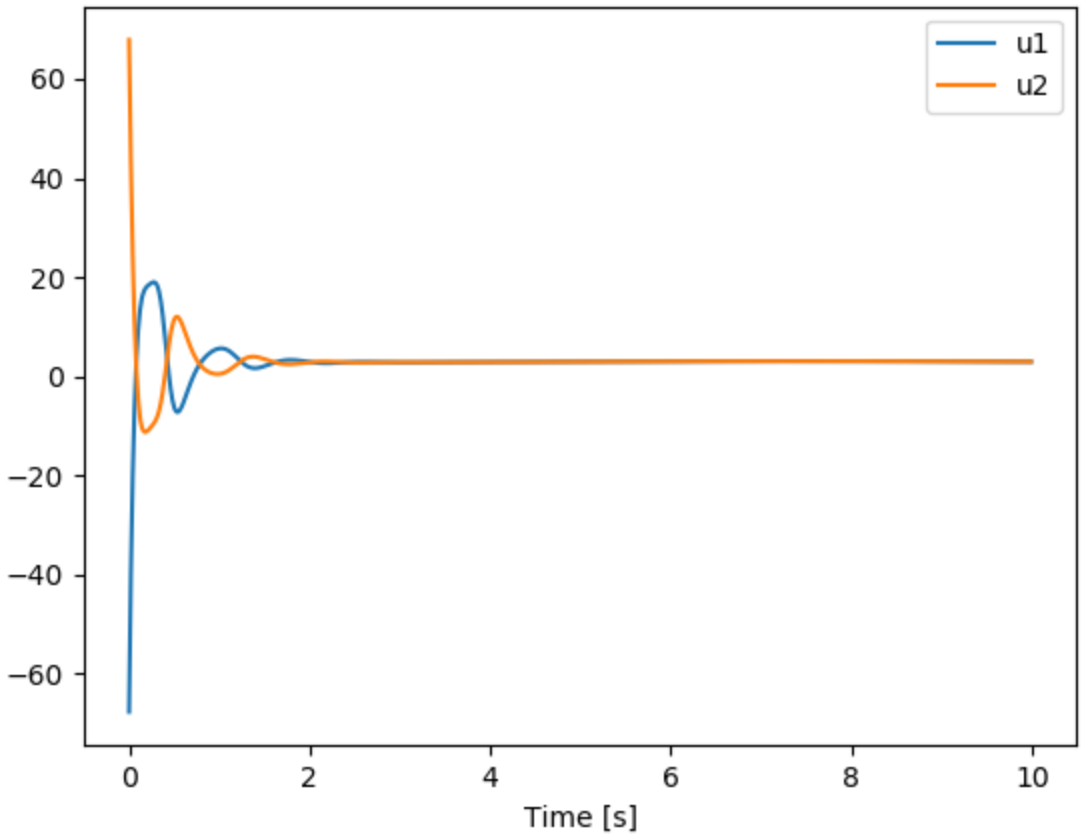
Part 3.4

After analyzing the plots and results, I can say that it is a good controller to achieve the task of trajectory tracking in a better way cause we see that u_1 and u_2 have become equal after a particular time. Not much variations in θ as well. I don't see any issue with this controller. Apart from that, even after providing disturbances, controller was able to make quadrotor follow the trajectory.

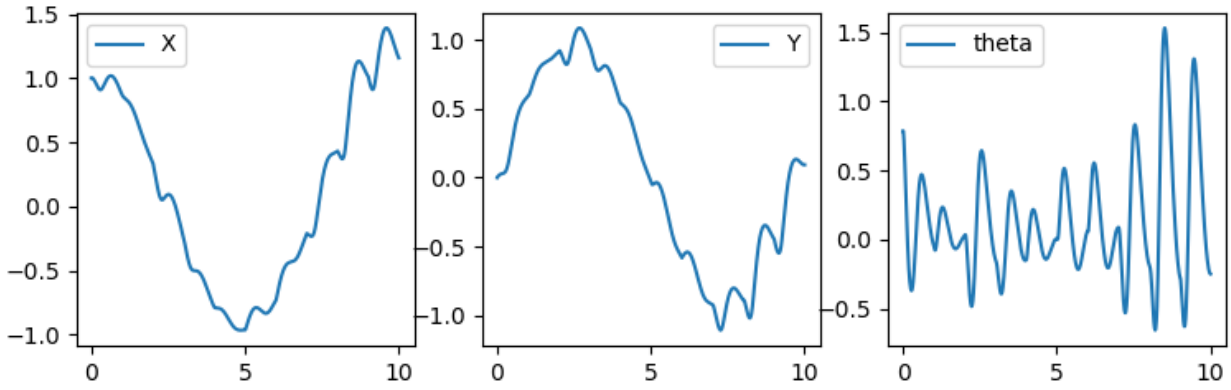
Part 3.5

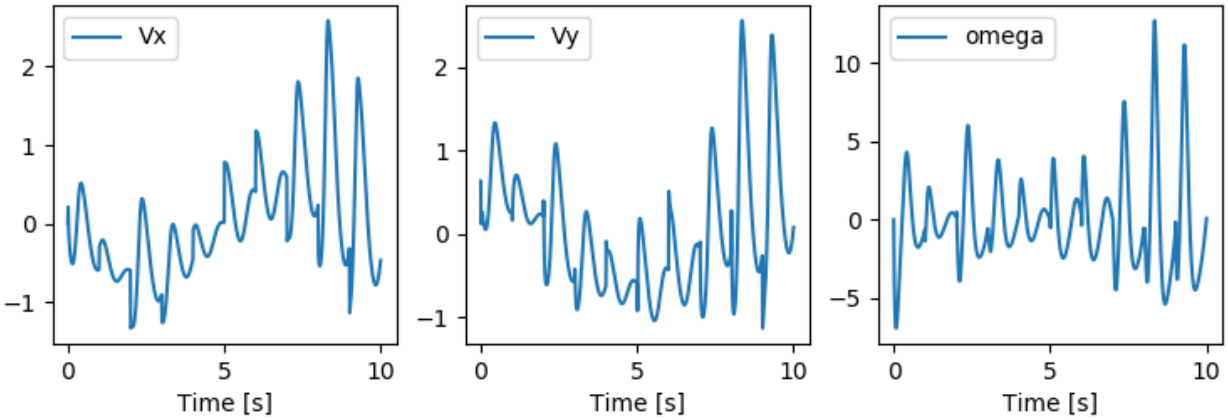
When I kept desired orientation θ = pi/4. Initially, there are some disturbances but at the end quadrotor behaves normally and followed the circular trajectory.
These are the plots

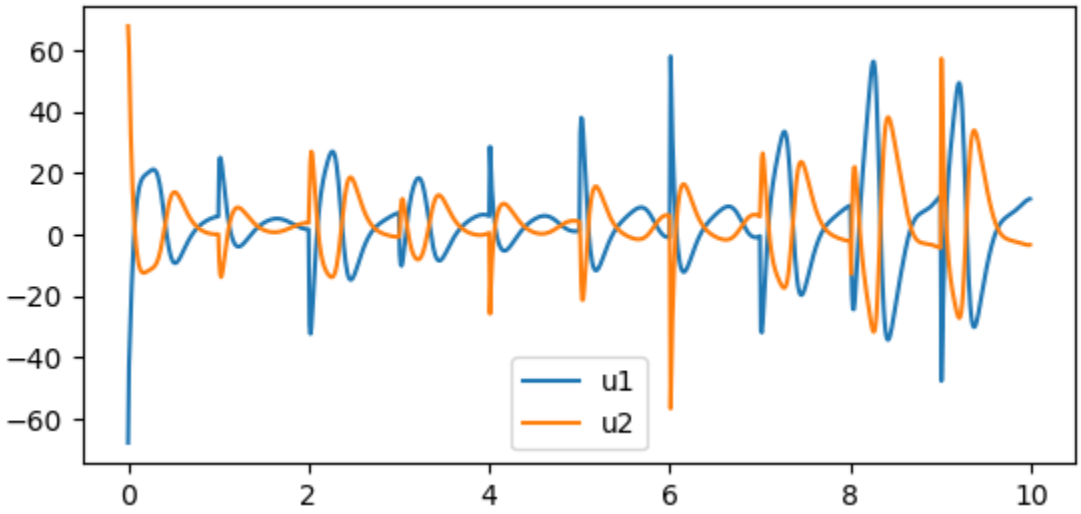




With disturbance and and desired angle = pi/4

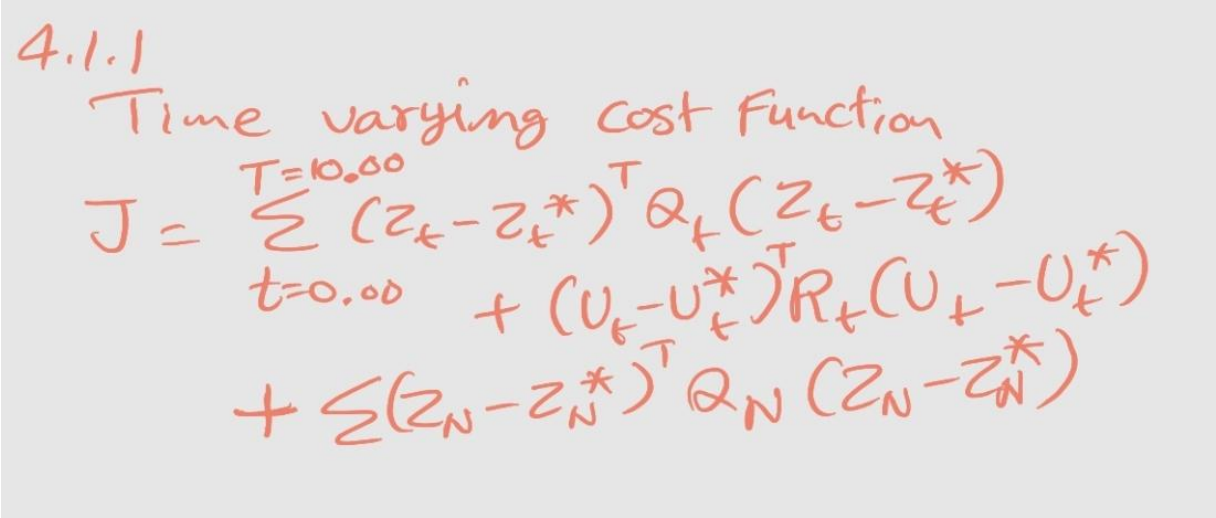



Vx
Vy
omega
Time [s]
Time [s]
Time [s]


u1
u2

Part 4

Part 4.1.1 Time varying cost function will be



Here t = time, z = state, u = control, N = last state.

Part 4.1.2

Compute_cost function is written in Jupyter notebook which is similar to above function.

Part 4.1.3

4.1.3. Quadratic approximation of the cost

$$\text{cost} = \sum_{t=0}^{10} (z_t - z_t^*)^T Q_t (z_t - z_t^*) + (u_t^T - u_t^*) R_t (u_t - u_t^*)$$

$$q = \frac{\partial \text{cost}}{\partial z} = 2 Q_t (z_t - z_t^*)$$

$$r = \frac{\partial \text{cost}}{\partial u} = 2 R_t (u_t - u_t^*)$$

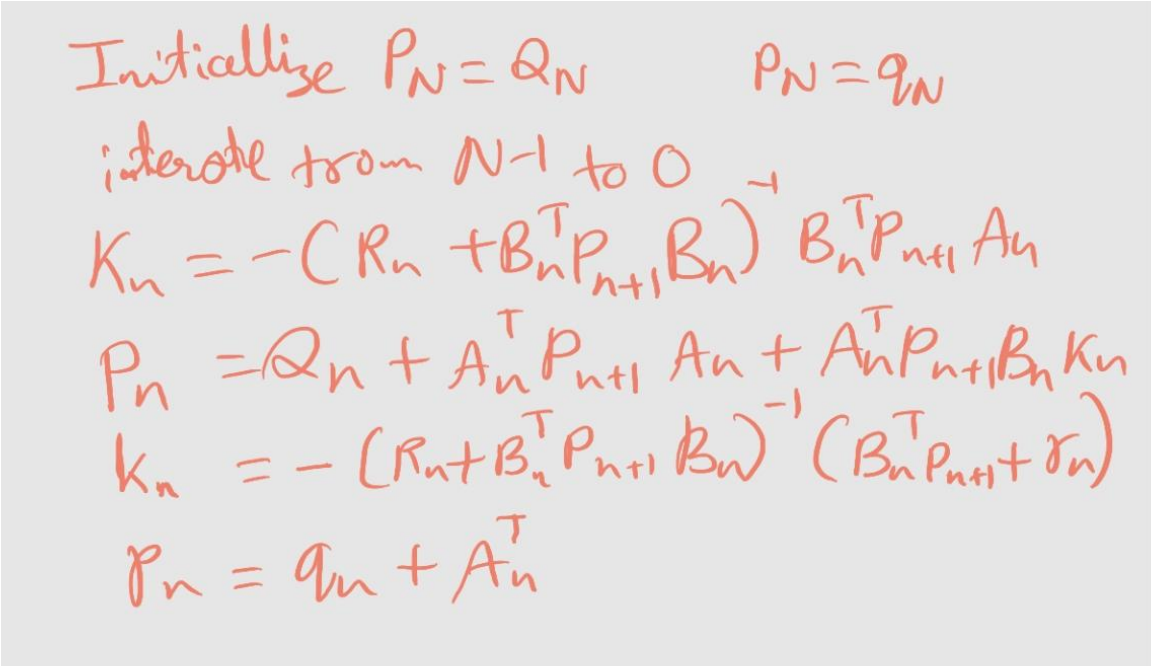
$$Q = 2 Q_t$$

$$R = 2 R_t$$

the formula for Quadratic approximation cost is

$$J \simeq \text{Const} + \sum_{n=0}^{N-1} q_n^T \bar{z}_n + \frac{1}{2} \bar{z}_n Q_n \bar{z}_n + r_n^T \bar{u}_n + \frac{1}{2} \bar{u}_n^T R_n \bar{u}_n$$

the function quadratic approximation should return Q, R, q, r for the following riccati recursion.



Part 4.1.5

The main iLQR algorithm uses above backward reiccati equations to find such a control to achieve such trajectory having desired points with robot orientation. The main iLQR algorithm that I tried to do is mentioned below.

4.1.4 Algorithm

$\alpha = 1$

Initial guess

$Z_i = zeros(6, 1001)$ $\quad U_g = \frac{mg}{2}$

$U_i = \frac{mg}{2} \times ones(2, 1000)$

Previous cost = Infinity

Cost = 0

while(1)

q, r, Q, R = quadratic_approx cost($Z_i, U_i, 1000$)

$K_{gains}, k_{feedfor}$ = trajectory function($q, r, Q, R, Z_i, U_i, 1000$)

while($\alpha > 0.01$)

for i in 1000

$U_n = U_g + K(Z_n) + \alpha t$

Z_n = nextState(Z_n, U_n)

Cost = computeCost(Z_n, U_n)

if(Cost < previous cost)

break

else

$\alpha = \alpha/2$

error = Previous cost − Cost

if error < 10

break

$Z_i = Z_{new} = Z_n$

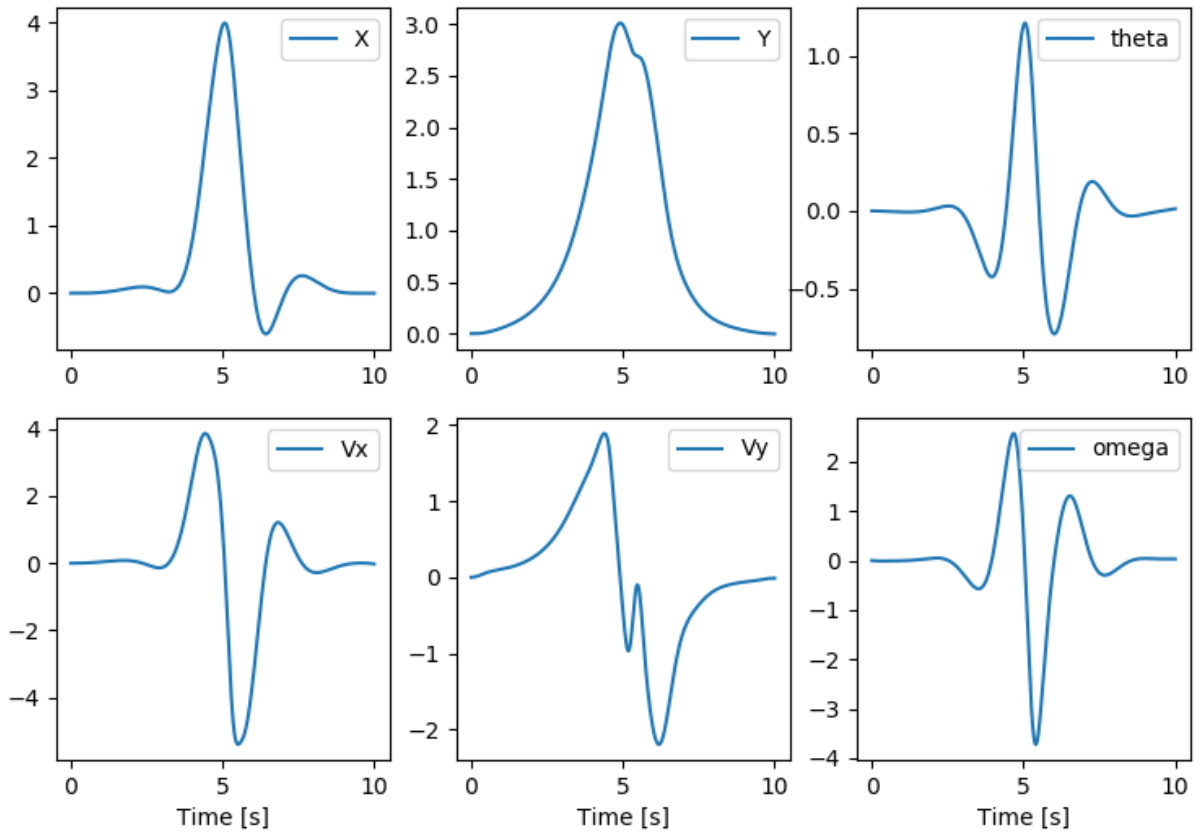
$U_i = U_{new} = U_n$

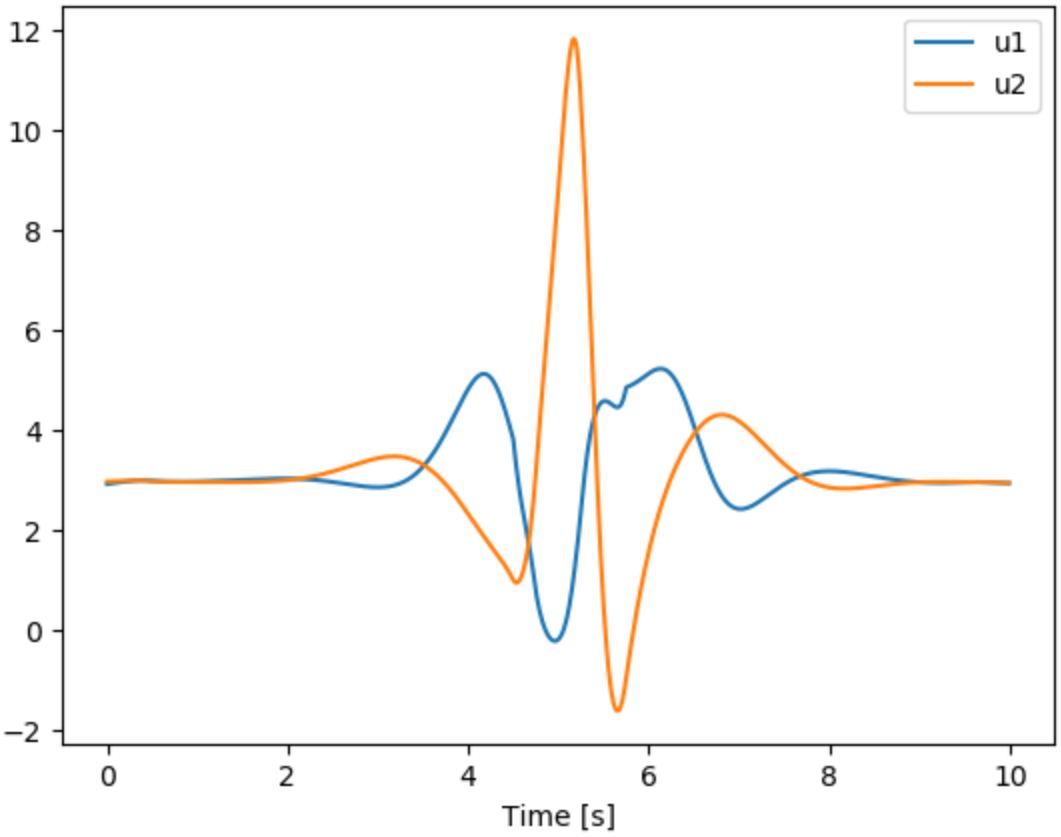
Previous cost = cost

Part 4.1.7

This algorithm is able to find an optimal trajectory to go from one point to another with the required orientation; however setting up the Q and R gains at different points is a really challenging task here. You have to set Q and R high at you desired points and less in other points of the trajectory. If you have set appropriate Q and R at different points then this algorithm can give you an optimal trajectory having way points with the required orientation.

Plots

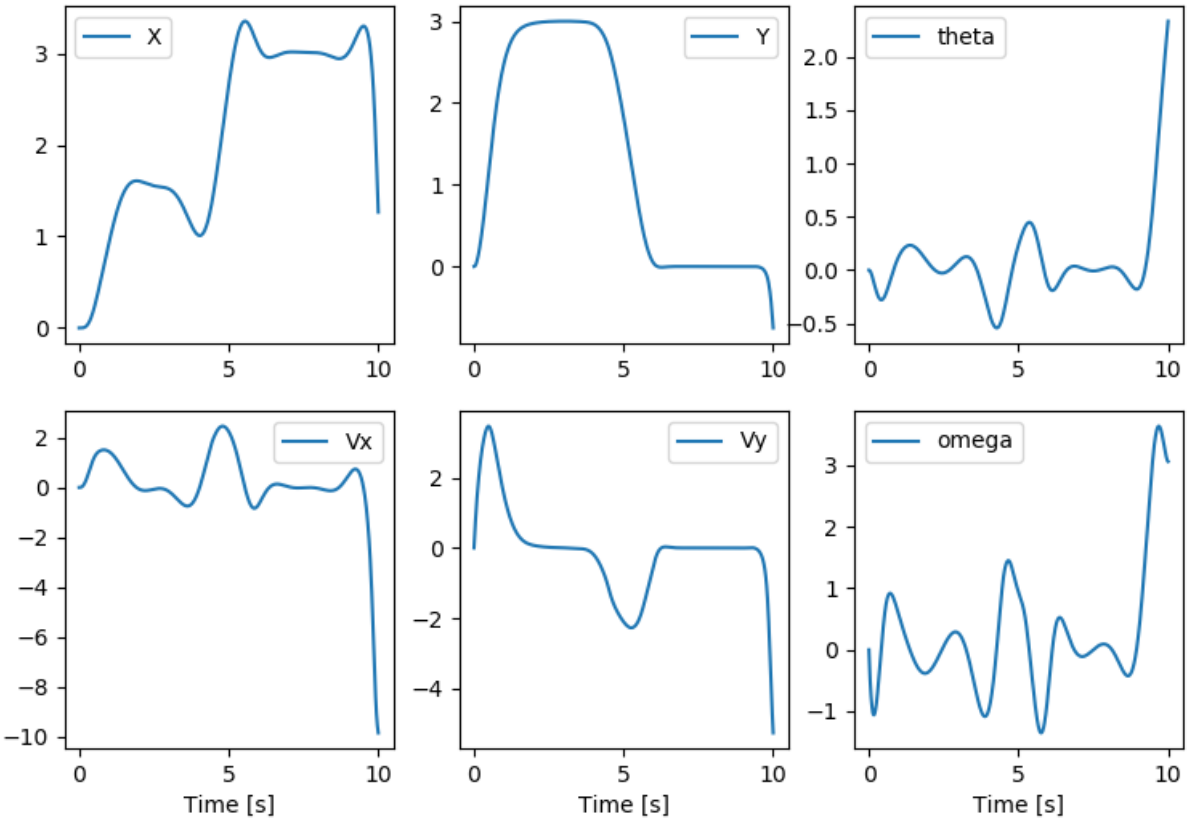


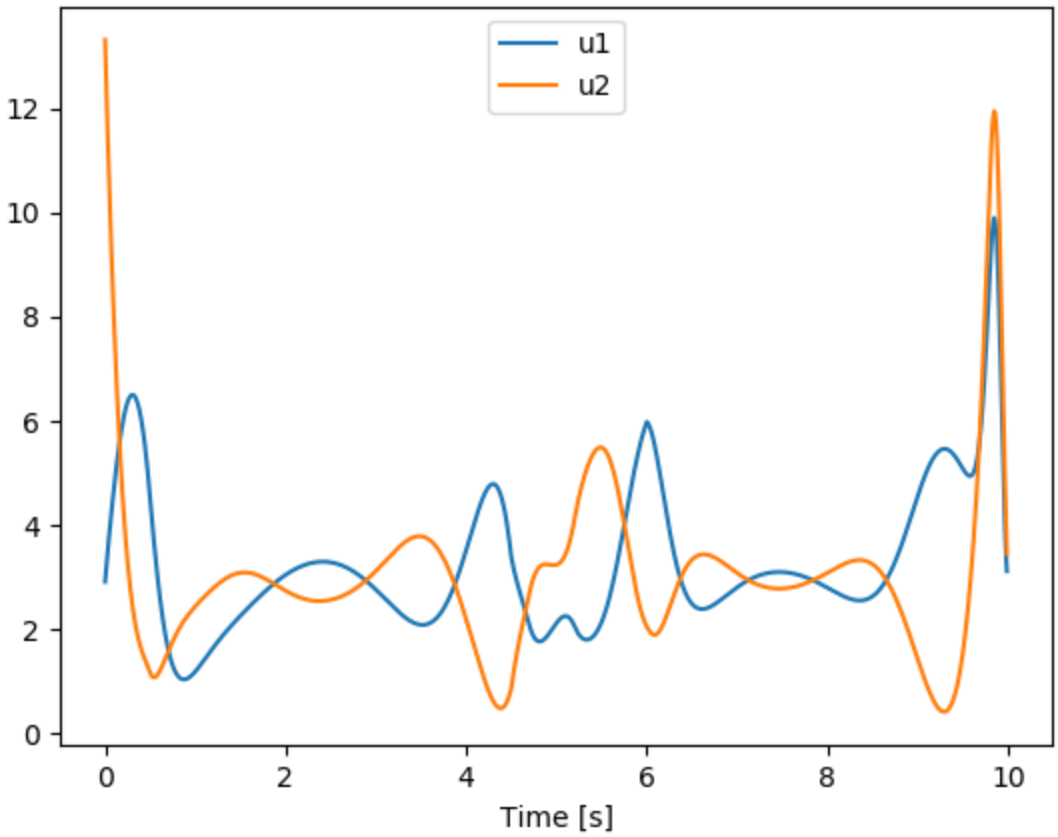


Part 4.2

I tried to do this task using current approach but couldn't set the appropriate Q and R gains at desired points to make it flip. Quadrotor is able to go to desired positions however I am not able to flip because I couldn't fine appropriate gains as well as appropriate intermediate points.

If I had more time and if I try to tune the Q and R gains at different points and setting up various types of points, I were able to make it flip.


X
Y
theta
Vx
Vy
omega
Time [s]
Time [s]
Time [s]


u1
u2
12
10
8
6
4
2
0
0
2
4
6
8
10
Time [s]