ROB-GY 6323 First Project Report

PROJECT 1 - OPTIMAL CONTROL REPORT

Done by:

GORDON OBOH

Department of Mechanical & Aerospace Engineering

Tandon School of Engineering , NYU

November 29, 2022

Contents

1 Introduction

The goal of this project is to control a basic 2D quadcopter to perform some acrobatic moves.
The quadcopter is depicted below:

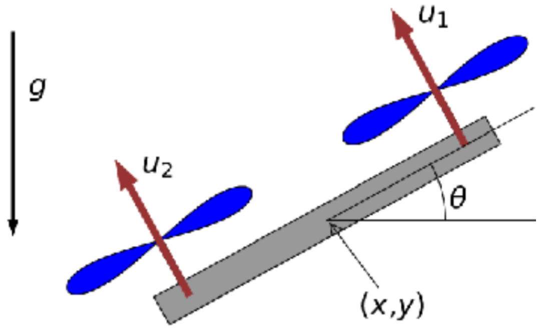

Figure 1: model of a 2D Quadcopter shown with some forces acting on it

This Project report serves to explain some of the steps taken in implementing the Optimal Control techniques used.
The drone is modeled with the following equations:

$$\dot{x} = V_x \qquad \dot{V_x} = \frac{-(u_1+u_2)sin\theta}{m}$$
$$\dot{y} = V_y \qquad \dot{V_y} = \frac{(u_1+u_2)cos\theta}{m} - g$$
$$\dot{\theta} = \omega \qquad \dot{\omega} = \frac{(u_1-u_2)r}{I}$$

where;
x and y are horizontal and vertical postions of the quadcopter.
V_x and V_y are the linear velocities along the x and y axis respectively.
θ is the orientation of the quadcopter with respect to the horizontal plane.
ω is the angular velocity of the quadcopter
u_1 and u_2 are our control inputs; the forces produced by the rotors.
m is the mass of the quadcopter.
I is the moment of Inertia.
r is half the length of the quadcopter; this is measured from the center of the quadcopter to tip of the propeller.
g is gravitational constant

Our State and control is denoted below:

$$\mathrm{z} = \Big[x, V_x, y, V_y, \theta, \omega\Big]^\top$$
$$\mathrm{u} = \Big[u_1, u_2\Big]^\top$$

The module *quadrotor.py* defines useful constants (mass, length, gravity, etc) and functions to simulate and animate the quadrotor in the Jupyter Notebook.

2 Part 1 - Setting Up

2.1 Discretizing the System

The first thing we have to do is discretize the system dynamics.

$$\begin{bmatrix} x_{n+1} \\ V_{xn+1} \\ y_{n+1} \\ V_{yn+1} \\ \theta_{n+1} \\ \omega_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ V_{xn} \\ y_n \\ V_{yn} \\ \theta_n \\ \omega_n \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -\frac{\sin\theta}{m}\Delta t & -\frac{\sin\theta}{m}\Delta t \\ 0 & 0 \\ \frac{\cos\theta}{m}\Delta t & \frac{\cos\theta}{m}\Delta t \\ 0 & 0 \\ \frac{r}{I}\Delta t & -\frac{r}{I}\Delta t \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \\ 0 \\ 0 \end{bmatrix}$$

$$z_{n+1} = z_n + \Delta * f(z_n, u_n)$$

2.2 Finding u^*

Our states are set to

$$z_0 = \begin{bmatrix} 0, 0, 0, 0, 0, 0 \end{bmatrix}^\top$$

With our set state, we compute u^* with the model equations.

$$\dot{V_x} = \frac{-(u_1 + u_2) sin\theta}{m} \qquad \dot{V_y} = \frac{(u_1 + u_2) cos\theta}{m} - g \qquad \dot{\omega} = \frac{(u_1 - u_2) r}{I}$$

$$0 = \frac{-(u_1 + u_2) sin\theta}{m} \qquad 0 = \frac{(u_1 + u_2) cos\theta}{m} - g \qquad 0 = \frac{(u_1 - u_2) r}{I}$$

$$0 = -(u_1 + u_2) sin\theta \qquad mg = (u_1 + u_2) cos\theta \qquad 0 = (u_1 - u_2)$$

$$\text{but } \theta = 0^\circ$$

$$0 = 0 \qquad mg = (u_1 + u_2) \qquad u_1 = u_2$$

$$0 = 0 \qquad u_1 = \frac{mg}{2} \qquad u_1 = u_2$$

$$u^* = \begin{bmatrix} \frac{mg}{2} \\ \frac{mg}{2} \end{bmatrix}$$

2.3 $\theta = 0$

If $\theta = 0$ then there will be no position change in the x direction.
On the x axis; $sin(0) = 0$ this results in an equilibrium on the forces acting on the quadrotor, the rotors(control inputs) require an angle change to break this equilibrium.

2.4 $\theta = \frac{\pi}{2}$

If $\theta = \frac{\pi}{2}$ the system will not be at rest.
On the x axis; $sin(\frac{\pi}{2}) = 1$ this results in maximum velocity in the -x direction.
On the y axis; $cos(\frac{\pi}{2}) = 0$ the system doesn't have any upward thrust to cancel out the force of gravity and will fall with maximum acceleration in the -y direction.

3 Part 2 - LQR to Stay in Place

3.1 Linearizing System Dynamics

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

where;

$$z^* = \Big[x_0, 0, y_0, 0, 0, 0\Big]^\top$$
x_0 is the desired x coordinate
y_0 is the desired y coordinate

$$u^* = \Big[\tfrac{mg}{2}, \tfrac{mg}{2}\Big]^\top$$

3.2 Linearization function

Now we get into the "`get_linearization`" function, it takes the state and control in symbolic form as parameters and returns a symbolic version of matrices A and B, A and B changes with each new state and control, this allows us to linearize on the new state and control values.

We use a combination of one step Euler integration

$$z_{k+1} = z_k + dt * f(z_k, u_k)$$

and first order Taylor-series expansion

$$z_{k+1} + \delta z_{k+1} = f(z_k + \delta z_k, u_k + \delta u_k) \approx f(z_k, u_k) + \tfrac{\partial f}{\partial z}|_{z_t,u_t}(z - z_k) + \tfrac{\partial f}{\partial u}|_{z_t,u_t}(u - u_k)$$
$$A = \tfrac{\partial f}{\partial z}|_{z_t,u_t} \; B = \tfrac{\partial f}{\partial z}|_{z_t,u_t}$$
$$\delta z_{k+1} = A(z_k, u_k)\bar{z}_k + B(z_k, u_k)\bar{u}_k$$

3.3 Infinite Horizon LQR Controller

The "`inf_LQR`" function takes A,B,Q and R matrices as parameters and returns gain(K), with the computed gain(K) we can get our optimal control(u).

$$\bar{u}_n = \bar{z}_n * K_n$$

The "`inf_LQR`" function is then used to compute the gain(K)

The cost function in equation (1) was implemented

$$J = min \sum_{i=0}^{N-1} (x_i^T Q_i x_i + u_i^T R_i u_i) + (x_N^T Q_N x_N) \tag{1}$$

subject to

$$z_{n+1} = Az_n + Bu_n$$

and the Discrete-time Riccati equation was used to solve the cost function and get the gain(K)

$$P_N = Q_N = Q$$
$$\text{for N-1 to 0}$$
$$K_n = -(B_n^\top P_{n+1} B_n + R_n)^{-1}(B_n^\top P_{n+1} A_n)$$
$$P_n = Q_n + (A_n^\top P_{n+1} A_n) + (A_n^\top P_{n+1} B_n K_n)$$

with the calculated K_n above, the optimal control policy is given in equation (2)

$$\mu_n^*(z_n) = K_n z_n \tag{2}$$

3.4 Analysis and Plots

When there are no disturbance, the control inputs converge, but with added disturbances the control inputs compensates and returns back to converged point, the spikes in figure 5 reflect this control compensation for disturbances.

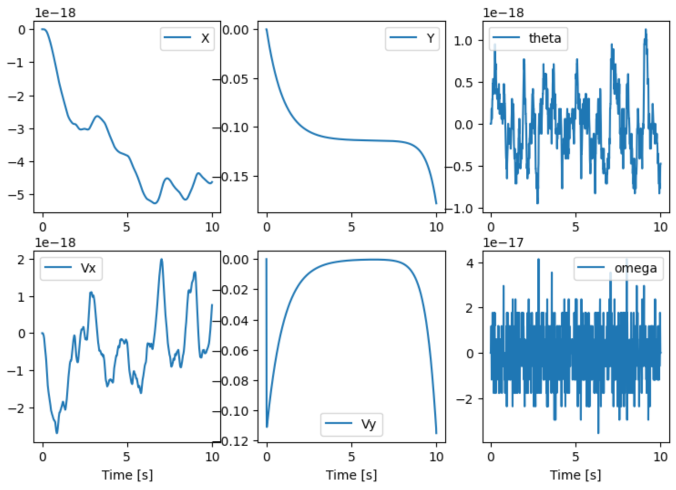

Figure 2: Quadrotor stabilized at z=0 with no disturbance

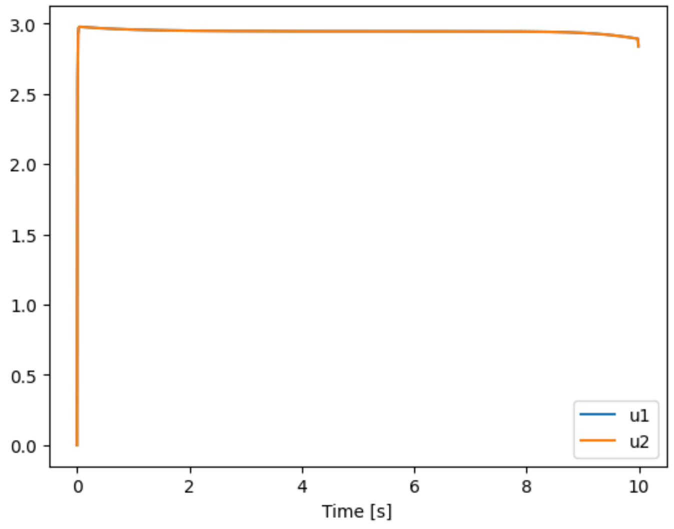

Figure 3: control inputs of Quadrotor stabilized at z=0 with no disturbance

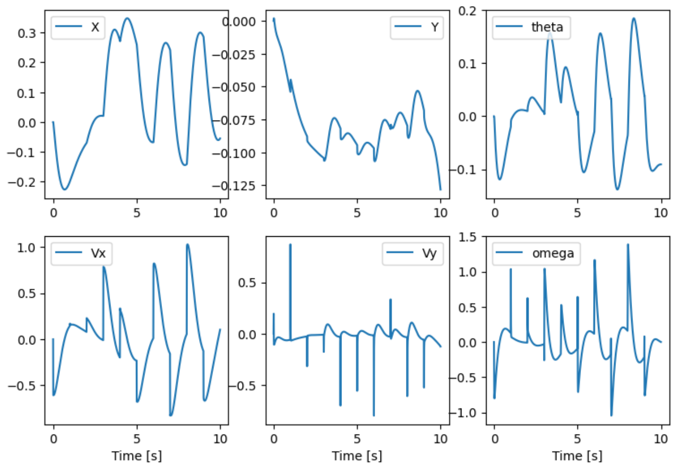

Figure 4: Quadrotor stabilized at z=0 with some disturbance

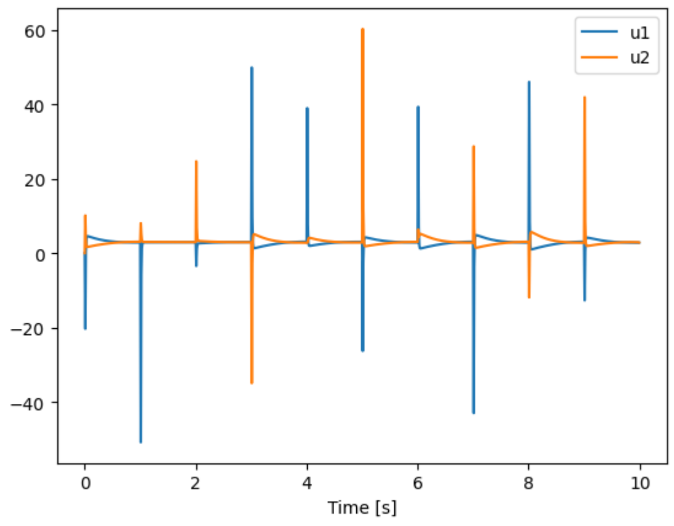

Figure 5: control inputs of Quadrotor stabilized at z=0 with some disturbance

4 Following a Circular Trajectory

4.1 Following a circular path

The linearization changes with state and control inputs values.
Why? as the quadrotor moves, it's state changes and it's control inputs have to change to keep it on path.

4.2 Design

The first step in designing the controller was to get $\bar{z}$ that would tracking around a circular path, Cartesian coordinate and polar coordinate forms were considered, and I used the polar coordinate equation of a circle to define the position and the derivatives of theses positions with respect to time to get the velocity.

$$x = rcos(\theta)$$

$$y = rsin(\theta)$$

but $\theta = \omega \Delta t$ and $r = 1$

$$x = cos(\omega \Delta t) \tag{3}$$

$$y = sin(\omega \Delta t) \tag{4}$$

The derivatives of (3) and (4) with respect to time would would give us the respective velocities, which are presented below:

$$\dot{x} = -\omega sin(\omega \Delta t)$$

$$\dot{y} = \omega cos(\omega \Delta t)$$

with the above equations $\bar{z}$ can be given as:

$$\bar{z} = \Big[cos(\omega \Delta t), -\omega sin(\omega \Delta t), sin(\omega \Delta t), \omega cos(\omega \Delta t), 0, 0\Big]^\top$$

The next step would be to get A and B matrices, this is provided with `"get_linearization"` function as the state changes with previously defined u^* as the initial control.
A `"solve_LQR controller_trajectory"` function is then implemented to calculate K gain and k feedforward.

Initializing

$$P_N = Q_N = Q$$

$$p_N = q_N = Q * \bar{z}_n$$

for N-1 to 0

$$K_n = -(B_n^\top P_{n+1} B_n + R_n)^{-1}(B_n^\top P_{n+1} A_n) \tag{5}$$

$$P_n = Q_n + (A_n^\top P_{n+1} A_n) + (A_n^\top P_{n+1} B_n K_n)$$

$$k_n = -(B_n^\top P_{n+1} B_n + R_n)^{-1}(B_n^\top P_{n+1} A_n) \tag{6}$$

$$p_n = Q_n + (A_n^\top P_{n+1} A_n) + (A_n^\top P_{n+1} B_n K_n)$$

with the calculated K_n and k_n in equation (5) and (6), the optimal control policy is given below as:

$$\mu_n^*(z_n) = K_n z_n + k_n$$

4.3 Analysis and Plots

When there are no disturbance, the control inputs are fairly flat, but with added disturbances the control inputs compensates and returns back to the defined circular path, the spikes in figure 9 reflect this control compensation for disturbances.

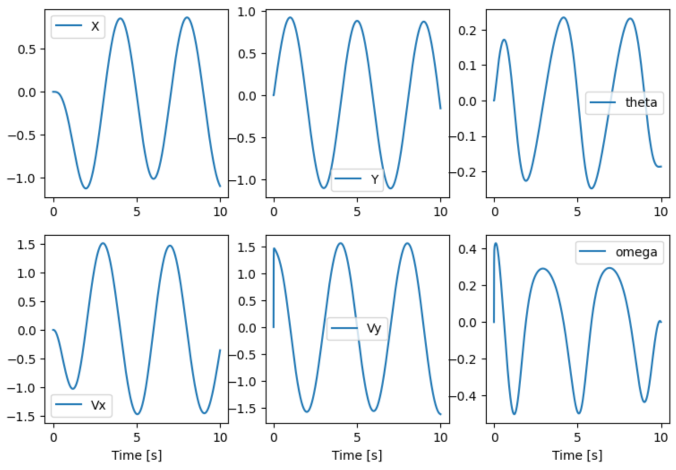

Figure 6: Quadrotor following a circular path with no disturbance

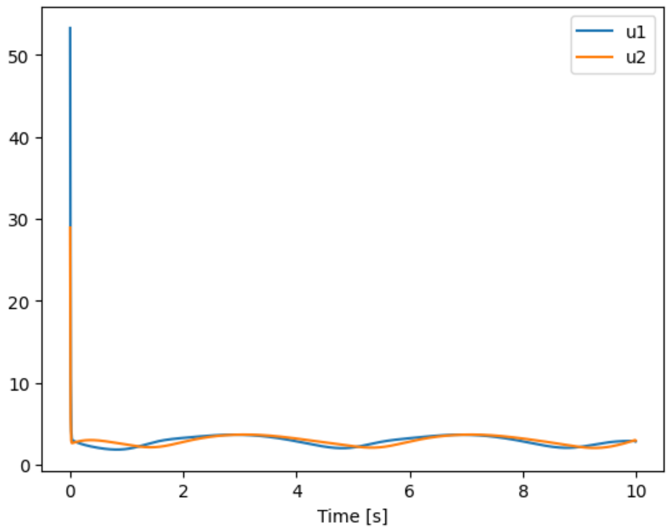

Figure 7: control inputs of Quadrotor following a circular path with no disturbance

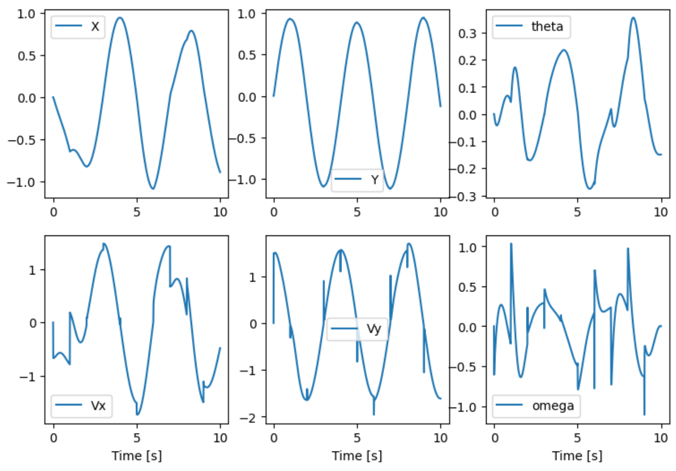

Figure 8: Quadrotor following a circular path with some disturbance

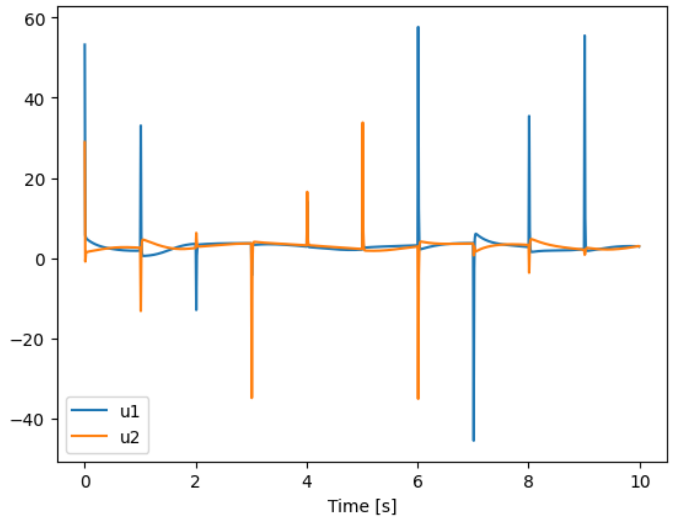

Figure 9: control inputs of Quadrotor following a circular path with some disturbance

4.4 $\theta = \frac{\pi}{4}$

The quadrotor starts out with $\theta = \frac{\pi}{4}$ but the rotors makes the quadrotor to return back to $\theta = 0$ orientation and it follows the circular path in this orientation.

The advantage of this controller implementation is that this design is relatively simple.

5 iterative LQR

5.1 Task 1

My cost function to encourage the behaviour is defined below:

$$J = \sum_{n=0}^{400}(\bar{z}_n^\top Q_n \bar{z}_n + u_n^\top R_n u_n) + \sum_{401}^{599}(\bar{z}_n^\top Q_n \bar{z}_n + u_n^\top R_n u_n) + \sum_{600}^{999}(\bar{z}_n^\top Q_n \bar{z}_n + u_n^\top R_n u_n) + (\bar{z}_N^\top Q_N \bar{z}_N) \tag{7}$$

where:

$$\bar{z}_n = z_n - z_n^* \tag{8}$$

$$\bar{u}_n = u_n - u_n^* \tag{9}$$

for 0 to 400 and 600 to 999:

$$\bar{z}_n = z_n$$

$$\bar{u}_n = u_n$$

for 401 to 599:

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

$$z_n^* = \left[3, 0, 3, 0, \tfrac{\pi}{2}, 0\right]^\top$$

$$\bar{u}_n = u_n \tag{10}$$

5.2 Compute Cost function

From the above equations (8) and (11) the initial u_n is set as $\frac{mg}{2}$ and initial z_0 is set as 0. the z is calculated with `"get_next_state"` function using z_0 as the first state and u_i as the successive control inputs.

$$z_{n+1} = f(z_n, u_n) \tag{11}$$

with the calculated z_{n+1} we then use it and the respective u_n to calculate the cost for each trajectory

5.3 Quadratic Approximation function

Getting the Jacobians and Hessians require first and second order partial derivatives of the cost function on equation (7) with respect to the state and control input.

$$\frac{\partial J}{\partial z}|_{z_n,u_n} \rightarrow Q_n z_n + q_n$$
$$\frac{\partial J}{\partial u}|_{z_n,u_n} \rightarrow R_n u_n + r_n$$
$$\frac{\partial^2 J}{\partial z^2}|_{z_n,u_n} \rightarrow Q_n$$
$$\frac{\partial^2 J}{\partial u^2}|_{z_n,u_n} \rightarrow R_n$$

z_n is the calculated states from the "`compute_cost`" function and u_n is the respective control used to calculate the state.

5.4 iLQR

Using the Jacobians and Hessians calculated in the "`get_quadratic_approximation`" function the A and B matrices linearized at each z_n and u_n is passed into a "`iLQR`" function and it returns the gains(K) and feedforward(k).

Initializing

$$P_N = Q_N$$

$$p_N = q_N = Q_N \bar{z}$$

for N-1 to 0

$$K_n = -(R_n + B_n^\top P_{n+1} B_n)^{-1} B_n^\top P_{n+1} A_n \tag{12}$$
$$P_n = Q_n + A_n^\top P_{n+1} A_n + A_n^\top P_{n+1} B_n K_n$$
$$k_n = -(R_n + B_n^\top P_{n+1} B_n)^{-1} B_n^\top p_{n+1} + r_n \tag{13}$$
$$p_n = q_n + A_n^\top p_{n+1} + A_n^\top P_{n+1} B_n k_n$$

using the calculated gains and feedforward as shown in equations (12) and (13), the optimal control policy is calculated below.

$$u_n = u^* + K_n(z_n - z^*) + \alpha k_n$$

where α is used to perform line search. α is initialized at 1 as the differences between the old cost and new cost are compared to a tolerance value, α is halved when the difference between the old cost and new cost is below the tolerance value. α stops decrementing when it gets below 0.01.

5.5 iLQR Algorithm

iLQR as the name implies is an iterative algorithm that provides a solution for nonlinear control problems. The iLQR algorithm can be set to stop at a defined criteria.

```
 1: $z_0 = [0,0,0,0,0,0]^\top$
 2: for $i$ in (0 to N+1) do
 3:     if $i = 0$ then
 4:         $A, B = getlinerization(z_0, u_0)$
 5:         $z_1 = A_n z_0 + B_n u_0$
 6:     else
 7:         $A, B = getlinerization(z_n, u_n)$
 8:         $z_{n+1} = A_n z_n + B_n u_n$
 9:     end if
10: end for
11: if $|Cost_{old} - Cost_{new}| > tol$ then
12:     $Cost_{old} = Cost_{new}$
13:     for $i$ in (N to 0) do
14:         if $i =$ N then
15:             $P_N = Q_N$
16:             $p_N = q_N = Q_N \bar{z}$
17:         else
18:             $K_n = -(R_n + B_n^\top P_{n+1} B_n)^{-1} B_n^\top P_{n+1} A_n$
19:             $P_n = Q_n + A_n^\top P_{n+1} A_n + A_n^\top P_{n+1} B_n K_n$
20:             $k_n = -(R_n + B_n^\top P_{n+1} B_n)^{-1} B_n^\top p_{n+1} + r_n$
21:             $p_n = q_n + A_n^\top p_{n+1} + A_n^\top P_{n+1} B_n k_n$
22:         end if
23:     end for
24:     for $i$ in (0 to N-1) do
25:         $u_n = u^* + K_n(z_n - z^*) + \alpha k_n$
26:         $z_{n+1} = A_n z_n + B_n u_n$
27:     end for
28: else
29:     $\alpha = \frac{\alpha}{2}$
30: end if
31: if $Cost_{new} < tol$ or $alpha < 0.01$ then
```

```
32:     $u = u_n$          ▷ At the end of the Algorithm the returned control is set to $u_n$
33: break:
34: end if
```

5.6 Analysis and Plots

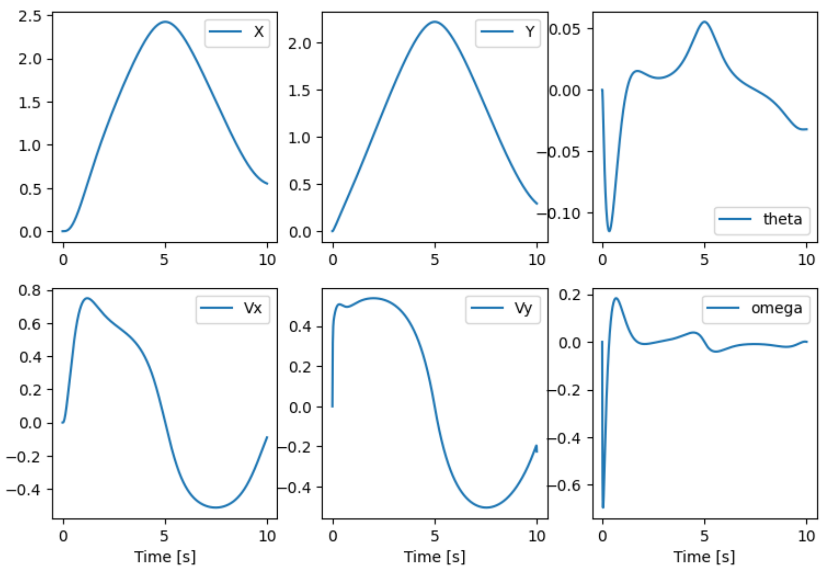

Figure 10: Quadrotor following defined path with no disturbance

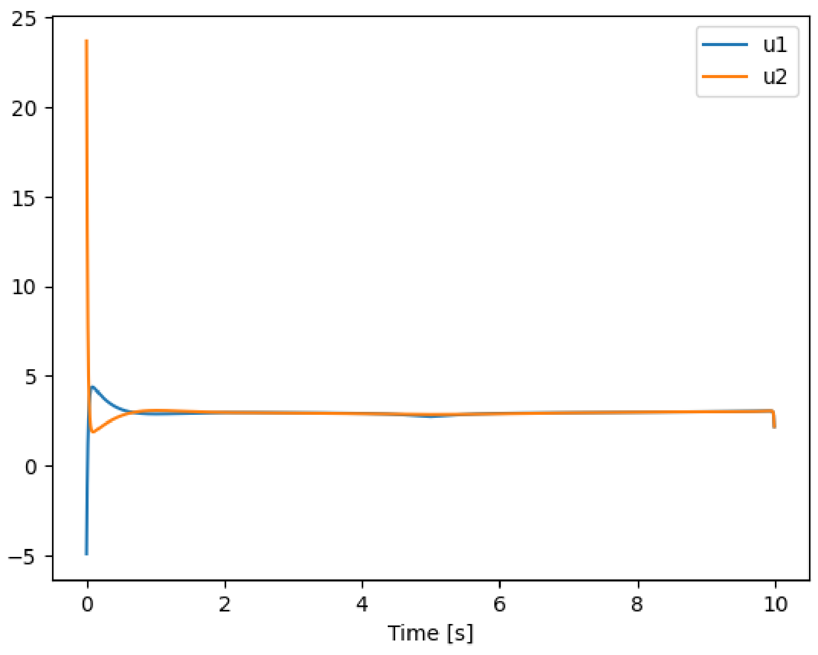

Figure 11: control inputs of Quadrotor following defined path with no disturbance

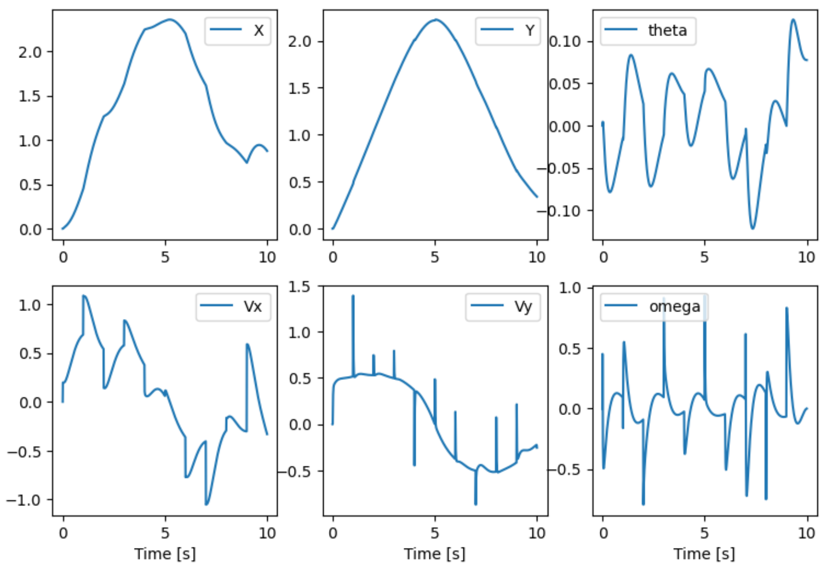

Figure 12: Quadrotor following defined path with some disturbance

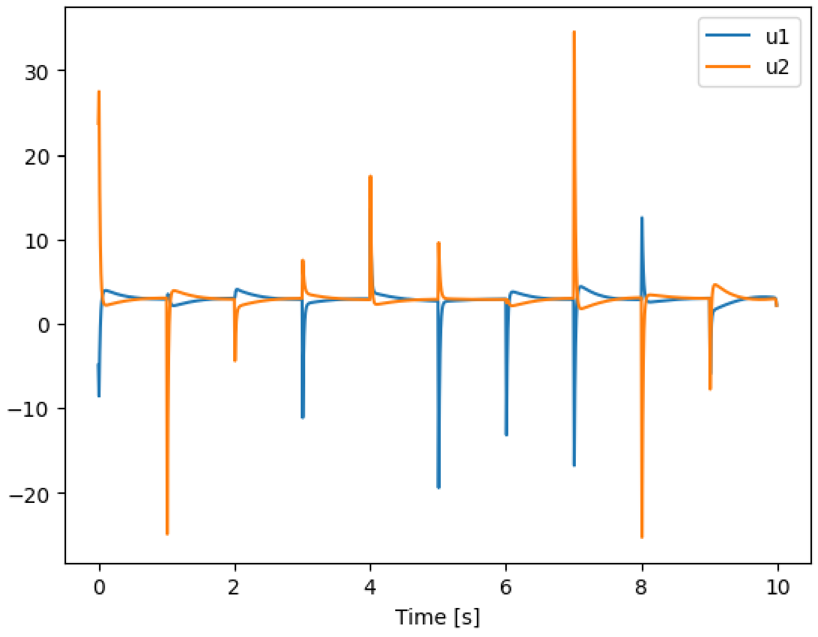

Figure 13: control inputs of Quadrotor following defined path with some disturbance

5.7 Benefits and Issues

Benefits:

- iLQR is a useful algorithm for nonlinear dynamic systems

Issues:

- Sudden changes in dynamics or disturbance may not be easily handled by iLQR, causing the system to lose control
- iLQR has higher computational cost when the dynamics keeps changing suddenly.