RBE 502 Robot Conrol - Project

Fall 2024

Sai Hitesh Viswasam

Department of Robotics Engineering

Worcester Polytechnic Institute

Email: sviswasam@wpi.edu

1 Introduction

The Problem Statement of this project is to design a controller for a quadrotor travelling in restricted airspace as shown in 1.

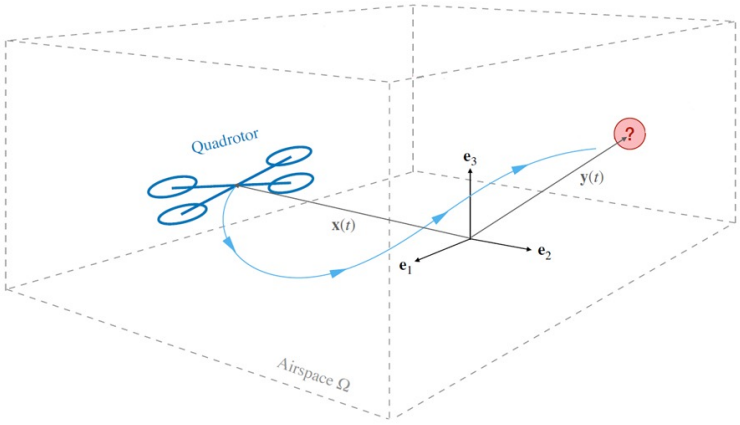

Figure 1: Quadrotor navigating in the airspace following a target position/trajectory

The quadrotor system is defined by a frame and four propellers that can provide forces in the direction perpendicular to the frame. The rotors also provide moments about the CoM of the quadrotor due to their own rotation, given by the equations

$$F_i = k_f \omega_i^2 \tag{1}$$

$$M_i = \pm k_m \omega_i^2 \tag{2}$$

The state variables for this system can be described as the vector $q \in \mathbb{R}^{12}$.

$$q = \begin{bmatrix} r & \dot{r} & \gamma & \dot{\gamma} \end{bmatrix}^T$$

where $r = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is the position of the CoM of the quadrotor and $\gamma = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ is the vector containing the roll, pitch and yaw angles of the quadrotor respectively.
The rigid body dynamics of the system can be represented as follows:

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{3}$$

and

$$\mathbf{I} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \mathbf{I} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4}$$

Here, $\mathbf{R}$ represents the rotation matrix that converts the orientation of the quadrotor from the body frame to components of vectors in the world frame which is given by:

$$[R] = \begin{bmatrix} \cos\psi\cos\theta - \sin\phi\sin\psi\sin\theta & -\cos\phi\sin\psi & \cos\psi\sin\theta + \cos\theta\sin\phi\sin\psi \\ \cos\theta\sin\psi + \cos\psi\sin\phi\sin\theta & \cos\phi\cos\psi & \sin\psi\sin\theta - \cos\psi\cos\theta\sin\phi \\ -\cos\phi\sin\theta & \sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{5}$$

And $\mathbf{L}$ is the distance of the rotors from the CoM, $\begin{bmatrix} p & q & r \end{bmatrix}^T$ is the vector representing $\dot{\gamma}$, the angular velocities of the drone in the body frame.

2 Methodology

In this project, two different controllers are implemented for the drone to track trajectories around the space. The trajectories that are provided are the circular helix and a diamond helix. The controllers used are the PD controller and LQR controller, which are further described below.

2.1 PD Controller

To implement the PD controller, the property of **differential flatness** of the quadrotor system is used so as to make the control of the system easier. Through this property, we observe that even though the system has 12 state variables, we can simply the control by

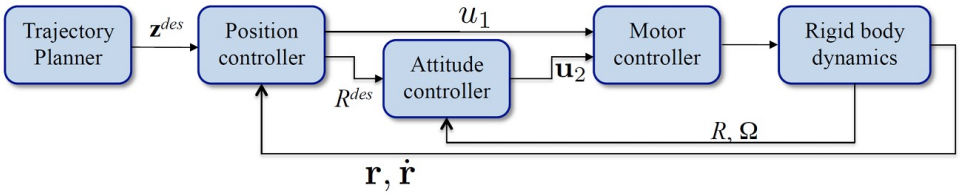

Figure 2: Implementation of the PD controller using differential flatness of the system

only controlling the flat outputs of the system, which are given by

$$x = \begin{bmatrix} \mathbf{r} \\ \psi \end{bmatrix}$$

where $\mathbf{r}$ represents the x, y and z positions of the quadrotor and ψ is the yaw angle. The trajectory can be directly defined in terms of only these variables and the whole system can still be controlled. [1]

Using this property, the PD controller of this system can be implemented through two seperate controllers namely the position and the attitude controller. The attitude controller works in the inner loop which controls the orientation of the drone and runs at a high frequency and based on this, the outer loop position controller controls the position and velocity of the system and runs at a lower frequency, this is depicted in 2.

To implement the controller to follow a trajectory, we make the assumptions that the system stays in near hover conditions while following the trajectory. Under these assumptions, we can linearise the system around the equilibrium point of hover, where the quadrotor operates with small roll and pitch angles. The equations of dynamics reduce as follows:
The position error can be written as:

$$e_i = (r_{i,T} - r_i)$$

To reduce this error to 0, we have:

$$\ddot{r}_{i,T} - \ddot{r}_i^{\text{des}} + k_{d,i}(\dot{r}_{i,T} - \dot{r}_i) + k_{p,i}(r_{i,T} - r_i) = 0 \tag{6}$$

The desired positions can be related to the desired roll and pitch values as:

$$\ddot{r}_1^{\text{des}} = g\left(\theta^{\text{des}} \cos\psi_T + \phi^{\text{des}} \sin\psi_T\right) \tag{7}$$

$$\ddot{r}_2^{\text{des}} = g\left(\theta^{\text{des}} \sin\psi_T - \phi^{\text{des}} \cos\psi_T\right) \tag{8}$$

$$\ddot{r}_3^{\text{des}} = \frac{1}{m} u_1 - g. \tag{9}$$

The desired yaw is given by:

$$\psi^{\text{des}} = \psi_T(t) \tag{10}$$

$$r^{\text{des}} = \dot{\psi}_T(t) \tag{11}$$

The above equations are implemented in MATLAB directly and by tuning the control gains, the trajectory can be tracked with minimum error.

2.2 LQR Controller

To implement the LQR controller, we first need to convert the system into a linear state space form. Using the above equations for hover assumptions and using small angle approximations, the state space equation for the above system is given by the equation shown in [2].

$$\dot{\mathbf{q}} = A\mathbf{q} + B\mathbf{u}$$

Where

$$A = \begin{bmatrix} 0_{3\times3} & I_{3\times3} & 0_{3\times1} & 0_{3\times1} & 0_{3\times1} & 0_{3\times3} \\ 0_{1\times3} & 0_{1\times3} & g_s(\psi) & g_c(\psi) & 0 & 0_{1\times3} \\ 0_{1\times3} & 0_{1\times3} & -g_c(\psi) & g_s(\psi) & 0 & 0_{1\times3} \\ 0_{1\times3} & 0_{1\times3} & 0 & 0 & 1 & 0_{1\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times1} & 0_{3\times1} & 0_{3\times1} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times1} & 0_{3\times1} & 0_{3\times1} & 0_{3\times3} \end{bmatrix} \tag{12}$$

and

$$B = \begin{bmatrix} 0_{5\times1} & 0_{5\times1} & 0_{5\times1} & 0_{5\times1} \\ \frac{1}{m} & 0 & 0 & 0 \\ 0_{3\times1} & 0_{3\times1} & 0_{3\times1} & 0_{3\times1} \\ 0 & 1/J_{xx} & 0 & 0 \\ 0 & 0 & 1/J_{yy} & 0 \\ 0 & 0 & 0 & 1/J_{zz} \end{bmatrix} \tag{13}$$

The control inputs can be written in the form

$$\mathbf{u} = K(\mathbf{q}_{des} - \mathbf{q})$$

where K is the gain matrix.

This can be directly implemented in MATLAB using the *lqr* command

$$[KS] = lqr(A, B, Q, R)$$

3 Results

Using the PD controller for the trajectory tracking, the results obtained are as follows:

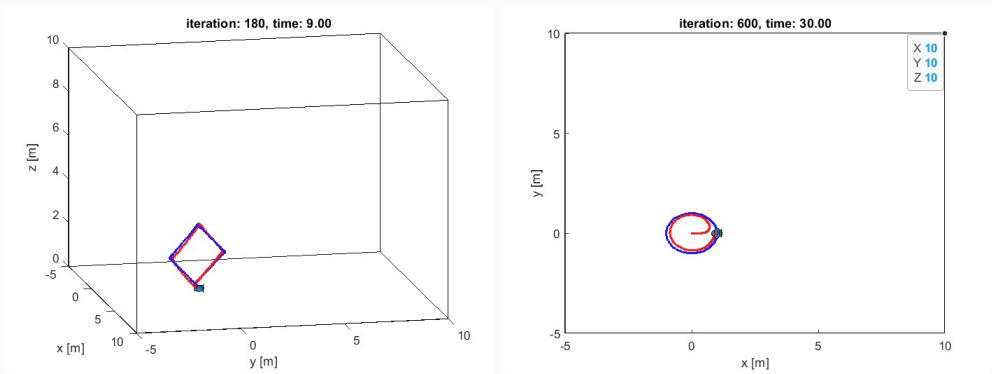

Figure 3: Tracking the Diamond and Circle Trajectories using PD controller

The Diamond trajectory is executed in 180 iterations and the controller took 9 seconds to track the trajectory. Similarly, the Circle trajectory took 600 iterations and 30 seconds to

execute.

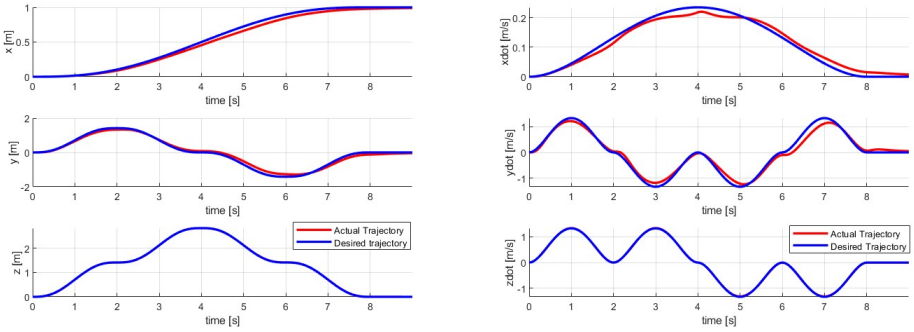

Figure 4: Tracking Performance of the PD controller for the diamond trajectory.

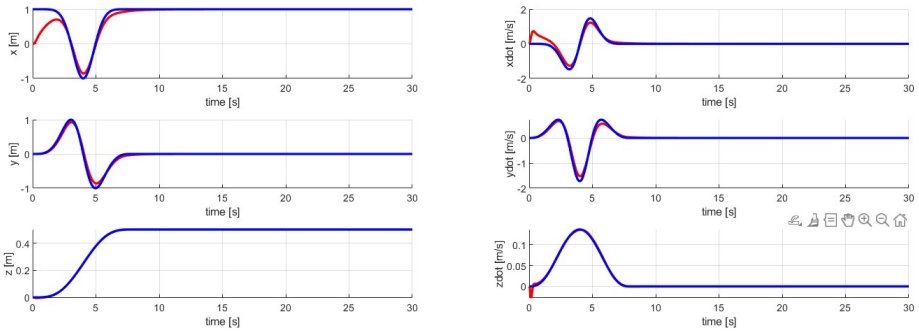

Figure 5: Tracking Performance of the PD controller for the circle trajectory.

The fig. 4 and 5 show the tracking performance of the PD controller for different trajectories. The Mean Square Error of the trajectories tracked are 0.012 m^2 and 0.027 m^2 for the diamond and the circle trajectories respectively. This performance was obtained from the following choice of hyperparameters:

Gain	**x**	**y**	**z**	**Roll (f)**	**Pitch (q)**	**Yaw (y)**
K_p	6.19	9	18.5	1000	1000	300
K_d	6.9	7	22	80	80	60

Table 1: Gain values for x, y, z positions and Roll, Pitch, Yaw

In comparision, the performance of the LQR controller is as follows in fig.6:
With the LQR controller, the Diamond trajectory is executed in 193 iterations and the controller took 9.63 seconds in MATLAB to track the trajectory. Similarly, the Circle trajectory

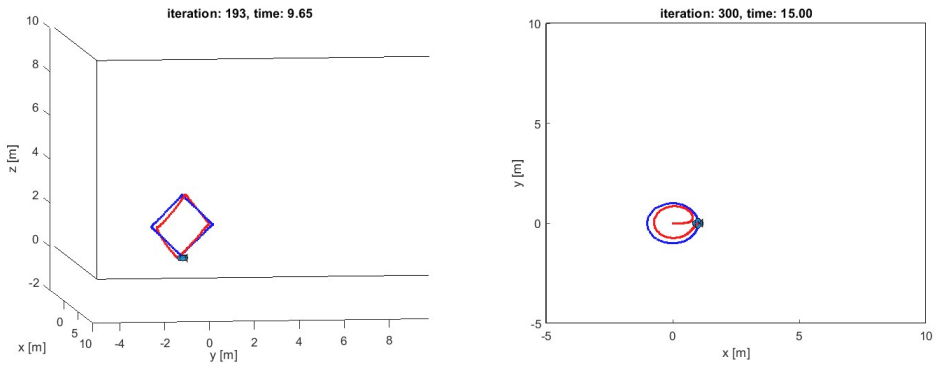

Figure 6: Tracking the Diamond and Circle Trajectories using LQR controller

was run until 300 iterations and 15 seconds in MATLAB.

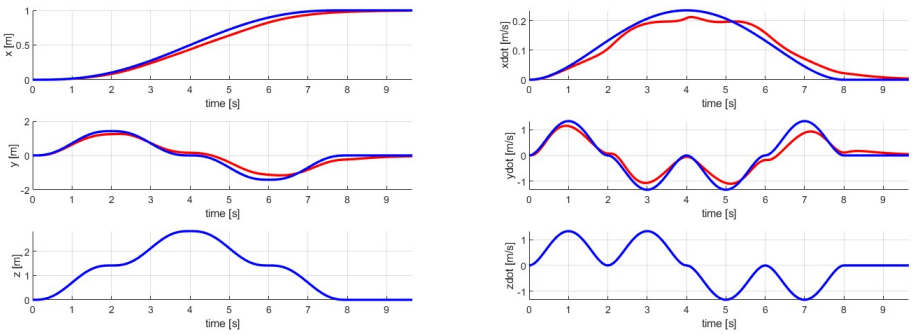

Figure 7: Tracking Performance of the LQR controller for the diamond trajectory.

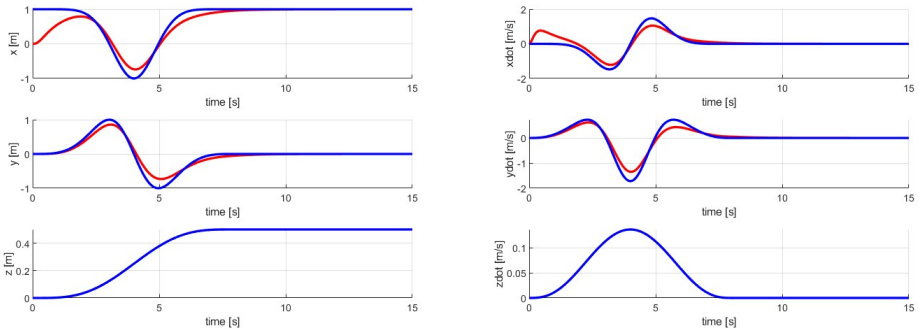

Figure 8: Tracking Performance of the LQR controller for the circle trajectory.

The fig. 7 and 8 show the tracking performance of the LQR controller for different trajectories. The Mean Square Error of the trajectories tracked are 0.035 m^2 and 0.064 m^2 for the

diamond and the circle trajectories respectively. This performance was obtained from the following choice of hyperparameters:

Matrix	Values
Q	$\text{diag}(500, 500, 500, 100, 100, 100, 1000, 1000, 1000, 10, 10, 10)$
R	$\text{diag}(0.00001, 0.00001, 0.00001, 0.00001)$

Table 2: Diagonal matrices Q and R

4 Inferences

From the above results, it can be concluded that both the controllers were successful in being able to track a trajectory by minimising the error. The PD controller could minimise the error better as reported by the MSE values for the trajectories in both the controllers. This could be because of the fact that the LQR controller also regulates and optimises the control input for the trajectories as well, thereby making it difficult for the quadrotor to navigate sharp turns in the trajectories, which was depicted clearly by the circle trajectory. Additionally, the trajectory tracking was performed by both the controllers using near-hover assumptions with modest accelerations, and hence the trajectories panned out were not exactly the same as the desired trajectories, however the error was minimal.

Coming to tuning the controllers, in the case of the PD controller, high gains were given to the attitude controller since it runs at a higher frequency and the outer loop position controller depends on the attitude controller. Hence high gain values stabilise the attitude of the drone first and then track the position. In the case of the LQR controller, high values in the **Q** matrix are responsible for the fast convergence of the system and low values in **R** matrix are responsible for the maximum limit of the control input that can be given to the system.

It can be concluded that a PD controller is simple to implement and can track and minimize errors well while following a trajectory, however it is really senstive to the gains and can only perform trajectory tracking using near-hover assumptions. For this purpose, a LQR

controller can be used to track complex trajectories without using hover assumptions. However, even in the case of an LQR controller, the cost function includes a cost on the input and hence an LQR controller puts a limit on the maximum input that can be given to the system, which makes it hard to navigate through steep curves, as shown in the circular trajectory. Also, an LQR controller needs the system dynamics to be in a linear form and hence a better controller is needed for Non-Linear systems.

References

[1] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP multiple micro-UAV testbed," *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010. doi: 10.1109/MRA.2010.937855.

[2] J. R. Thomas, "Trajectory generation and control for quadrotor grasping and perching," in *Proceedings*, 2012. [Online]. Available: `https://api.semanticscholar.org/CorpusID:208284343`