Simulator Refinements and Quadcopter Control

Jett Vanbrocklin

February 2026

1 Introduction

As stated in [1], The motion of a quadrotor can be defined by a union of linear and attitude dynamics, kinematics, and their relationship. The governing equations for these relationships can be found in [2].

However, to create a high fidelity control model, system and environmental factors must be accounted for. In this laboratory assignment, a control algorithm is implemented that transcribes the desired pose and position into the commanded voltage upon the motors of the quadrotor. The report follows the theoretical analysis, specific implementation, and exerimental results of the model.

2 Theoretical Analysis

2.1 The Drag Coeffecient

The value of $\boldsymbol{d_a}$ (The Drag Coefficient) is affected by both elements of the quad state and environmental conditions. For instance, the amount of air the quad "runs into" is dependent on the cross-sectional area that is perpendicular to the velocity vector, and the magnitude of the velocity vector itself. If the plane of the quad rotors is parallel to the velocity vector, the cross-sectional area is minimal. However, as the quad rotor plane alters its roll or pitch angle, the area normal to the velocity increases, causing the quad to push through "more air." The quad's velocity also has effect on the amount of air displaced, and in turn an effect on the drag coefficient.

Looking into the environmental variables, the level of humidity and the density of the air both have affects on the quad's drag. This again relates to the intuitive notion that drag is related to how much air or molecules the quad displaces whilst traveling.

To develop a model for $\boldsymbol{d_a}$, a controlled environment where the recently described factors of velocity, attitude, and air/moisture density can be accurately defined and replicated. For example, a wind tunnel could be used. By fastening a quad to a rigid body equipped with a force sensor at desired pitch angles. Then air could be blown out at a predisposed density and velocity (simulating

the quad traveling at said velocity). The determined force is the drag force, otherwise expressed as $-\boldsymbol{d_a v_I^u}$, where $\boldsymbol{v_I^u}$ is the unit vector of the velocity of the quad.

2.2 Expressing an Arbitrary Drag Function

As per the lab document, the drag coefficient $\boldsymbol{d_a}$can be defined as

$$d_a = \frac{1}{2} C_d A_d \rho f_d(z_I, v_I)$$

where $\boldsymbol{C_d}$ is a constant drag coefficient, $\boldsymbol{A_d}$ is the area of a infinitely thin disk approximation of the quad, $\boldsymbol{\rho}$ is the mass density of moist air, and $\boldsymbol{f_d(z_I, v_I)}$ is some function of $\boldsymbol{z_I}$, the quad's body z axis expressed in I, and $\boldsymbol{v_I}$, the velocity of the quad expressed in I.

To analytically express $\boldsymbol{f_d(z_I, v_I)}$, one could project the velocity vector onto the z_I axis, to determine the component of velocity aligned with z_I. The projection can be described as

$$z_I^T v_I$$

To properly model drag, it is desirable to express drag as a quadratic function of velocity. Intuitively, this is because the transfer of momentum between the quad and surrounding air and the rate at which the quad encounters air are both proportional to velocity. As result, the total momentum transfer per unit time scales with the square of velocity. The following model expresses this notion:

$$f_d = (z_I^T v_I)|z_I^T v_I|$$

where the projection of v_I onto z_I is squared whilst maintaining the sign information.

2.3 Step Response of Transfer Function

The following coupled differential equations for angular rate of each rotor were derived in class.

$$J_m \dot{\omega} + b\omega = K i_a$$
$$L_a \frac{di_a}{dt} + R_a i_a = e_a - K\omega$$

A description of these equations can be found in [3]. After converting these equations to the Laplace domain (and assuming $L_a \to 0$), we found the transfer function

$$\frac{\Omega(s)}{E_a(s)} = \frac{c_m}{\tau s + 1} \tag{1}$$

Using Matlab's built-in step response tool kit, the step response can be taken for this specific transfer function:

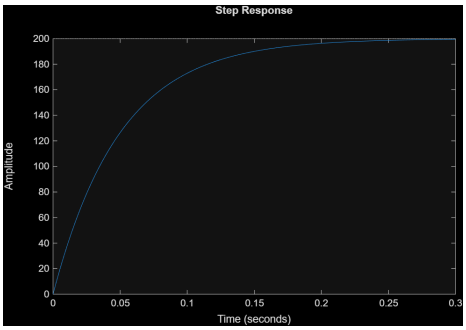


Figure 1: Voltage to Angular Rate Step Response

In **Figure 1** above, it can be noted that the time it takes for the amplitude to reach 90% is 0.125 seconds, marking the 90% rise time of the system.

Analyzing the transfer function (1), the rise time does not directly depend on the step magnitude. The output would only scale vertically, not effecting the time it takes to reach a percentage of the magnitude.

In practice however, it can be expected that the magnitude of the step has an effect on the output. There are a few reasons for this. First, the transfer function assumes perfect linearity, which is often not obtainable in the real world.

Second, systematic limits are not accounted for in this transfer function. Real motors have voltage, current, thermal, and torque limits. If these limits are surpassed, motors often act with unexpected behavior. Thus, given a step magnitude that over extends the motor would have direct effect on the rise time.

Lastly, real systems experience consequential effects from driving a motor. An example of this is back-emf, caused by the induced current from the acceleration of the motor. Additionally, forces such as friction decrease the energy transfer between a power source and the motor, where this friction can be viscous and directly dependent on the magnitude of acceleration.

For the reasons above, in real practice step magnitude has an effect on rise time.

2.4 Step Response of Equivalent ODE

The transfer function (1) can be converted into a first order differential equation in the angular rate ω as shown below:

$$\begin{aligned} \frac{\Omega(s)}{E_a(s)} &= \frac{c_m}{\tau_m s + 1} \\ \Omega(s)(\tau_m s + 1) &= E_a(s) c_m \\ \Omega(s)\tau_m s + \Omega(s) &= E_a(s) c_m \end{aligned}$$

Then, doing an inverse Laplace transform using the properties:

$$s * Y(s) \rightarrow \frac{dy(t)}{dt} \quad \text{and} \quad X(s) \rightarrow x(t)$$

grants the differential equation

$$\tau_m \frac{d\omega(t)}{d(t)} + \omega(t) = c_m e(t) \tag{2}$$

Rearranged to

$$\dot{\omega} = \frac{-1}{\tau_m}\omega(t) + \frac{c_m}{\tau_m}e(t)$$

where $\boldsymbol{\omega(t)}$ is the angular rate of a rotor as a function of time and $\boldsymbol{e(t)}$ is the voltage supplied to said rotor as a function of time.

Using Matlab's **ode45**, the differential equation can be evaluated with a step input. In my test, an initial rotor rate of $\omega = 0\ rad/sec$ and step magnitude of 1 volt was given to the differential equation. **Figure 2** below illustrates the step response under these conditions.

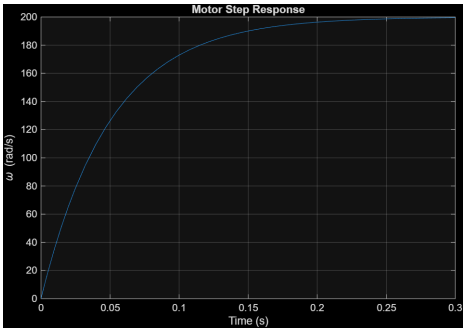


Figure 2: Differential Equation of Angular Rate Step Response

Figure 1 and **Figure 2** depict near identical step response plots (minor difference in format). This was to be expected, as the differential equation was directly derived from the transfer function, and expresses the same relationship between voltage and rotor angular rate.

2.5 PD Control

PD Control is a subset of PID Control, where the I (integral) term is strategically omitted to disregard steady state error, as in systems such as this one, accumulated error can be presumed negligible.

In reference to **Figure 3** above [2], values for k and k_d can be found under the assumption $k_y = 1$ such that the following performance metrics are met: 90% rise time $T_r < 0.25$ seconds; percent overshoot P_o ¡ 30%, settling time to within 2% of reference value T_s ¡ 2 seconds.

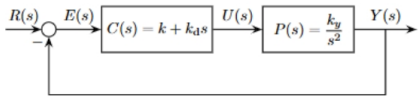


Figure 3: Basic PD Control of a Plant with Double-Integrator Dynamics

The closed loop transfer function $T(s)$ is

$$\begin{aligned} T(s) &= \frac{C(s)P(s)}{1 + C(s)P(s)} \\ &= \frac{k_y(k + k_d s)}{s^2 + k_y(k + k_d s)} \end{aligned}$$

as stated in [3, Page 42]. This transfer function takes the form

$$T(s) = \frac{\omega_n^2 + 2\zeta\omega_n s}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

by substitution of k, k_y, and k_d.

Also note, Page 39 Section 4.3.1 of [3] describes Settling Time, Percent Overshoot, and Rise Time as functions of ω_n and ζ.

Solving for ω_n and ζ in reference to the desired performance metrics, values for $k = \omega^2$ and $k_d = 2\zeta\omega$ can be found.

Plugging in the performance metrics, it is found that $\zeta \geq 0.36$, and $\omega_n > 7.2$. Generously rounding up, I chose parameters $\omega_n = 8$ and $\zeta = 0.5$. Computing $k = \omega_n^2 = 64$ and $k_d = 2\zeta\omega = 8$. After testing with Matlab's "step()" function, k_d was adjusted to 9 to reach performance.

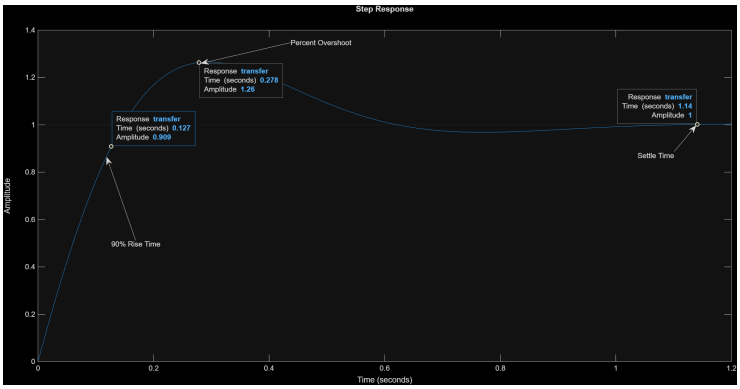


Figure 4: Step Response of Calculated PD Transfer Function

Figure 4 depicts the step response from the PD control loop transfer function utilizing the calculated values $k = 64$ and $k_d = 8$. The annotations of the plot describe the performance metrics. First, the 90% rise time is seen to be

0.127 seconds ($T_r < 0.25$), the percent overshoot is 26% ($P_o < 30\%$), and the settle time is 1.14 seconds ($T_s < 2$ seconds).

2.6 Deriving the Eigenvector of Rotation

From [1], the Rodrigues Formula defines a rotation matrix $R(\hat{a}, \phi)$ as a function of a unit vector $\hat{a}$ and an angle ϕ.

$$R(\hat{a}, \phi) = \cos\phi\, I_3 + (1 - \cos\phi)\, \hat{a}\hat{a}^T - \sin\phi\, [\hat{a}\times]$$

Let

$$\hat{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Then

$$\hat{a}\hat{a}^T = \begin{bmatrix} a_1^2 & a_1a_2 & a_1a_3 \\ a_2a_1 & a_2^2 & a_2a_3 \\ a_3a_1 & a_3a_2 & a_3^2 \end{bmatrix}$$

and

$$[\hat{a}\times] = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}.$$

Then, simplifying respective expressions (please note the notation $c\phi = cos\phi$ and $s\phi = sin\phi$):

$$c\phi I_3 = \begin{bmatrix} c\phi & 0 & 0 \\ 0 & c\phi & 0 \\ 0 & 0 & c\phi \end{bmatrix}$$

and

$$(1 - c\phi)\hat{a}\hat{a}^T = \begin{bmatrix} (1 - c\phi)a_1^2 & (1 - c\phi)a_1a_2 & (1 - c\phi)a_1a_3 \\ (1 - c\phi)a_2a_1 & (1 - c\phi)a_2^2 & (1 - c\phi)a_2a_3 \\ (1 - c\phi)a_3a_1 & (1 - c\phi)a_3a_2 & (1 - c\phi)a_3^2 \end{bmatrix}$$

and

$$-s\phi[\hat{a}\times] = \begin{bmatrix} 0 & s\phi a_3 & -s\phi a_2 \\ -s\phi a_3 & 0 & s\phi a_1 \\ s\phi a_2 & -s\phi a_1 & 0 \end{bmatrix}.$$

Substituting into Rodrigues' formula gives the full expansion:

$$R = \begin{bmatrix} \cos\phi + (1 - \cos\phi)a_1^2 & (1 - \cos\phi)a_1a_2 - \sin\phi\, a_3 & (1 - \cos\phi)a_1a_3 + \sin\phi\, a_2 \\ (1 - \cos\phi)a_2a_1 + \sin\phi\, a_3 & \cos\phi + (1 - \cos\phi)a_2^2 & (1 - \cos\phi)a_2a_3 - \sin\phi\, a_1 \\ (1 - \cos\phi)a_3a_1 - \sin\phi\, a_2 & (1 - \cos\phi)a_3a_2 + \sin\phi\, a_1 & \cos\phi + (1 - \cos\phi)a_3^2 \end{bmatrix}.$$

Subtracting asymmetric terms yields simple expressions. For example:

$$\begin{aligned} R_{23} - R_{32} &= ((1 - c\phi)a_2a_3 - s\phi a_1) - ((1 - c\phi)a_3a_2 + sin\phi a_1 \\ &= 2s\phi a_1 \end{aligned}$$

Rearrange,

$$a_1 = \frac{R_{23} - R_{32}}{sin\phi}$$

and similarly,

$$a_2 = \frac{R_{31} - R_{13}}{sin\phi}$$

$$a_3 = \frac{R_{12} - R_{21}}{sin\phi}$$

Given that,

$$e_E = \begin{bmatrix} e_{23} - e_{32} \\ e_{31} - e_{13} \\ e_{21} - e_{12} \end{bmatrix} \tag{3}$$

it can be seen that e_E is the vector form of R_E, and is proportional to the rotation axis with a scaling $2sin\phi$.

3 Implementation

3.1 High Fidelity Quadrotor Dynamics Simulator

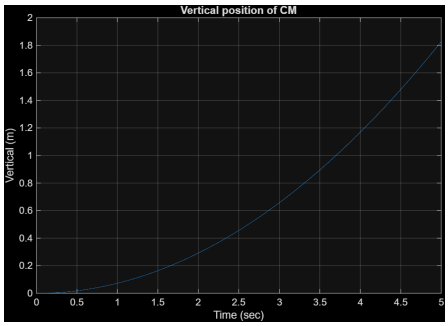
As mentioned previously in **Section 1 and 2**, this lab aims to implement a "high fidelity" simulator that accounts for aerodynamic drag and motor dynamics [2]. In **Lab 1**, a simple simulator was designed and is described in [1]. Updating this basic simulator to account for the factors listed above requires reconfiguring the velocity ODE and adding a differential model for the rotor angular rate as described in **Section 2.4**.

There are two steps to account for these new features. First, the parent function **simulateQuadrotorDynamicsHF** must be implemented such that the data flow supports both a new input of motor voltage over each time step and packages the quad rotor rates as a component of the quad's state. Second, the interface for the "high fidelity" ODE function must be developed in **quadODEFunctionHF**. To design this interface, the following governing equations must be deployed.

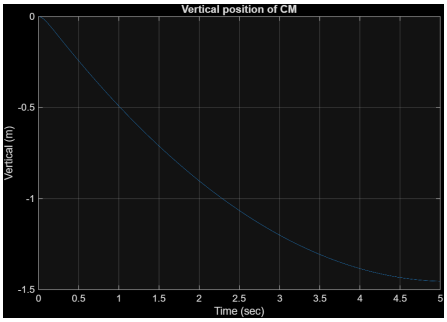
$$m\ddot{r}_I = -d_a v_I^u - mgZ_I + \sum_{i=1}^{4} F_{iI} + d_I \tag{4}$$

and differential equation **(2)** described above.

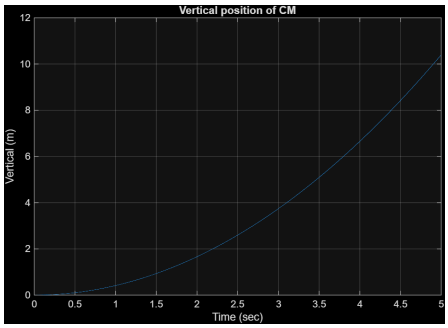
To compare these implemented changes to the simple simulator developed in [1], I configure the top scripts that call these functions such that the quad is at a near-zero velocity hover state. In this state, applying constant rotor rates $w_i = w_c$ for i = 1,2,3,4 where w_c takes the values 590, 610, 630 rad/sec. These values were chosen as they all are greater than the minimum rotor rate needed for hover of 585.65 rad/sec, and will all produce some positive nonzero upward velocity. Additionally, incrementing between each test by 20 rad/sec insured that change would be clearly expressed in output. Likewise, the respective voltages supplied to the motors $e_a = w_c/c_m$ are 2.95, 3.15, 3.35 V.



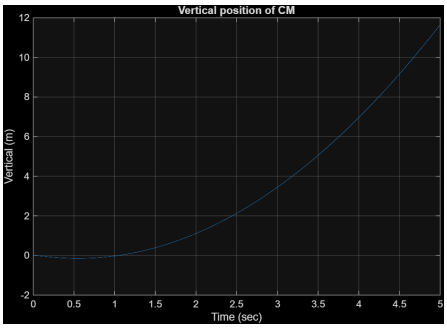
(a) Simple Simulator Vertical Distance: $w = 590$



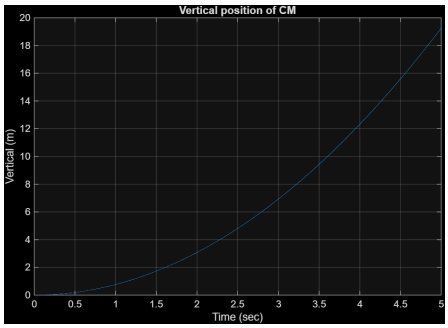
(b) Simple Simulator Vertical Distance: $e_a = 2.95$



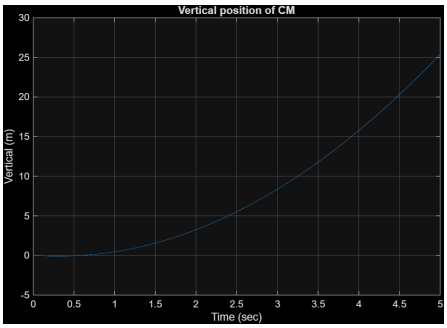
(c) Simple Simulator Vertical Distance: $w = 610$



(d) HF Simulator Vertical Distance: $e_a = 3.15$



(e) Simple Simulator Vertical Distance: $w = 630$



(f) HF Simulator Vertical Distance: $e_a = 3.35$

Figure 5: Simple and HF Vertical Position over 5 Second Interval

Figure 5 on the previous page depicts the vertical position of the simulated quad as a function of time in both the simple and high fidelity model given set constant angular rotor rates and voltages.

The major difference in the plots that share the same commanded rotor rate

is the speedy at which the quad responds to the command. In the simple model, the quad is directed to spin its motors at a selected rate and instantaneously achieves that rate, expressed by the graph with a increasing non-zero value for the span of 5 seconds. The HF simulator on the other hand takes into account motor dynamics, and can not instantly spin its rotors at the desired rate. The effect can be seen in two different ways among the graphs. First, given $e_a = 2.95$, the quad does not produce a rotor rate above the $w = 585.65$ hover threshold before the 5 second span completes, rendering a vertical trajectory decreasing at a decreasing rate. Second, the following HF plots for $e_a = 3.15$ and $e_a = 3.35$ depict a slight dip below zero whilst the motors are spinning up, and then display the expected rise in an upward direction.

3.2 Trajectory Controller

The Trajectory Controller aims to determine the necessary thrust (and direction) in order to approach the desired position. This relationship is modeled by the PD function

$$F_I^* = ke_r + k_d\dot{e} + mge_3 + m\ddot{r}_I^* \tag{5}$$

where F_I^* is the desired thrust, k, kd are the PD coefficients (descirbed further in **Section 4**), $e_r, \dot{e_r}$ are the positional error and velocity error respectively, m is the mass of the quad, g is acceleration due to gravity, $e_3 = [0, 0, 1]^T$, and $\ddot{r}_I^*$ is the desired acceleration.

The desired z axis z_I^* is expressed as a function of F_I^* where

$$z_I^* = \frac{F_I^*}{||F_I^*||}$$

Once z_I^* is found, the total thrust is the projection of F_I^* onto z_I:

$$F = F_I^* \cdot z_I = (F_I^*)^T R_{BI}^T e_3$$

The function returns total thrust $\boldsymbol{F}$ and the desired z axis $\boldsymbol{z_I^*}$.

3.3 Attitude Controller

The attitude controller aims to determine the desired torque in order to orient the quad in a desirable manner. Similar to the trajectory controller, a PD function is used:

$$N_B = Ke_E - K_d\omega_B + [\omega_B\times]J\omega_B$$

where K, K_d are the matrix PD coefficients (discussed later in **Section 4**, e_E is defined with (3), ω_B is the angular rate, and J is the intertial matrix of the quad.

This function returns a 3x1 vector of torques $\boldsymbol{N_B}$

3.4 Converting Trajectory and Attitude into Voltage

Utilizing the values of total thrust $\boldsymbol{F}$ and necessary torque $\boldsymbol{N_B}$ found in the trajectory and attitude controllers described above, the individual force scalars for each rotor can be found with the conversion matrix on Page 6 of [2].

The indivudal thrust scalars can be related to to angular rotor rates $F_i = k_F\omega_i^2$ and in turn voltage $\omega = c_m e_a$ where e_a is the voltage.

This conversion from total thrust and torque to isolated rotor thrust can return voltage values that exceed the maximum voltage the motors are rated for. To prevent this, the thrust is recalculated with a decreasing α value as described in [2, Page 6], stepping down the voltage whilst maintaining the proper ratio of torque.

This function returns the commanded voltage as a 4x1 vector $\boldsymbol{e_a}$

3.5 Integrated Quadrotor Controller

To integrate the trajectory controller, attitude controller, voltage coverter, and high fidelity ODE, I followed the following dataflow.

First, the desired position, velocity, and acceleration and the current state of the quad is sent to the trajectory controller. The trajectory controller then returns the total thrust needed to approach the desired state, and the desired z axis of the quad expressed in the I frame.

Second, the desired z and x axis along with the quad state is sent to the attitude controller, which returns the necessary torques to achieve the desired orientation.

Third, the total thrust and necessary torques are sent to the voltage converter, where the isolated voltage commands for each motor are calculated through the conversion matrix described in [2].

Lastly, the high fidelity ODE uses the voltage values and the current state of the quad to model the dynamics of the system.

4 Experimentation

To test my integration of the high fidelity simulator and the trajectory and attitude controller, I created a set of inputs that would result in the quad to travel in a circular path at a given altitude. The circular path was constrained to 4 meters in diameter and to have a time period of 10 seconds. In turn, the frequncy $w = 2\pi/10 = \pi/5$.

To do this, I first needed to establish my desired position over time, as well as my desired velocity and acceleration. Starting with position, r^*_{xI}, r^*_{yI}, and r^*_{zI} were defined as follows:

$$r^*_{xI} = 2 * cos(\frac{\pi}{5}t)$$

$$r^*_{yI} = 2 * sin(\frac{\pi}{5}t)$$

$$r^*_{zI} = 1$$

where t is the vector of all time steps.

These assignments were used in reference to the common parametric equation for a circle. To determine the desired velocity in I $\boldsymbol{v^*_I}$ and the desired acceleration in I $\boldsymbol{a^*_I}$ I took the derivative and the second derivative respectfully of the desired position equations. Producing:

$$\dot{r}^*_{xI} = -\frac{2\pi}{5} sin(\frac{\pi}{5}t)$$

$$\dot{r}^*_{yI} = \frac{2\pi}{5} cos(\frac{\pi}{5}t)$$

$$\dot{r}^*_{zI} = 0$$

and

$$\ddot{r}^*_{xI} = -\frac{2\pi^2}{25} cos(\frac{\pi}{5}t)$$

$$\ddot{r}^*_{yI} = -\frac{2\pi^2}{25} sin(\frac{\pi}{5}t)$$

$$\ddot{r}^*_{zI} = 0$$

Additionally, my desired body x axis $\boldsymbol{x^*_I}$ needed to always point towards the center of the circle. In other words, the x axis needed to point towards the opposite position in the circle. Normalizing and negating x^*_I and y^*_I allows simple assignment of the desired x axis to be

$$x^*_I = \begin{bmatrix} -\frac{r^*_{xI}}{||r^*_I||} \\ -\frac{r^*_{yI}}{||r^*_I||} \\ 0 \end{bmatrix}$$

For my initial conditions of my quad, I constrained the velocity, angular velocity, and acceleration to zero. For my initial position and orientation, I chose values to simplify my quad's necessary movements.

Initial position began at the edge of the circle referenced at time step zero, with height as the value I expected to hover at. In this case: $r_I = [2, 0, 1]^T$. For orientation, I set the yaw angle to π radians to initially point the x-axis of the quad towards the center of the circle, and set the other Euler angles to 0 such that $e = [0, 0, \pi]^T$.

To generate the circle, my trajectory controller and my attitude controller were iteratively tuned to optimize the circular path. The bounds of the tuning followed $k, k_d \epsilon [2, 5]$ and $K, K_d \epsilon [0.05, 1.2]$. Where k and k_d are the proportional and derivative coefficient respectively for the trajectory controller, and K and K_d are the proportional and derivative coefficient respectively for the attitude controller. The values that I found to result in the most efficient circle were $k = 5, k_d = 5, K = 1.2, K_d = 0.45$.

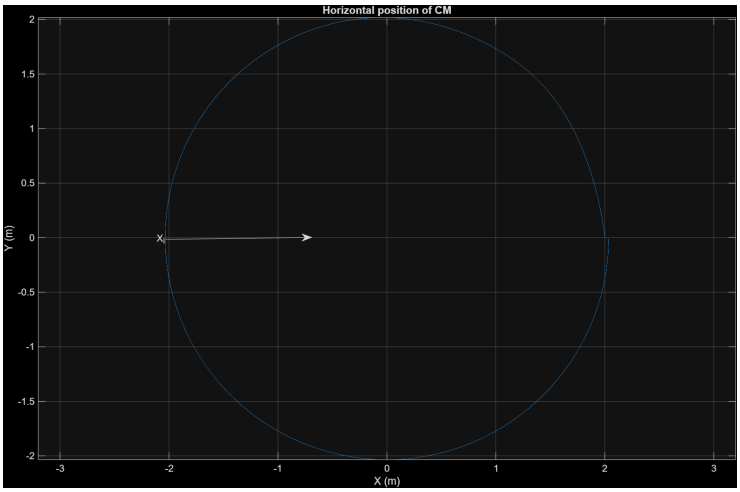


Figure 6: XY Path Plot

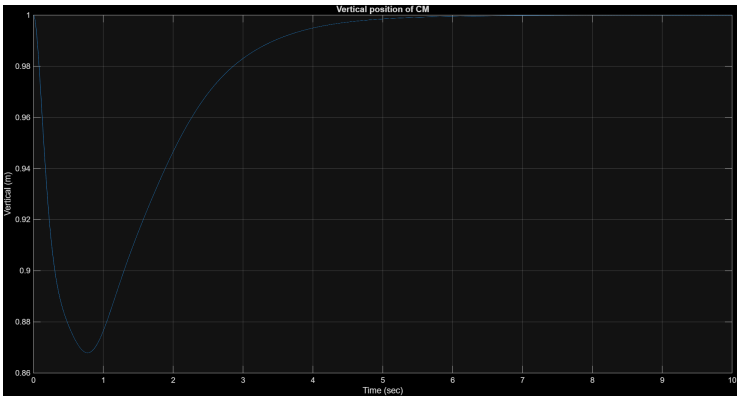


Figure 7: Vertical Path Plot

Figure 6 and 7 depict the XY and Z path of the quad traveling in a circular path. The vector in Figure 6 represents the quad body x-axis as it travels around the circular path where it faces the center. Figure 7 displays the trajectory controller response as well as the non-instantaneous motor response, as it is initially is at vertical position of $z_I = 1$ and falls slightly below that value for the first few iterations. It is then seen to correct itself and settle back at altitude 1.

References

1 Vanbrocklin, Jett. "Simulating Quadrotor Dynamics." Feb. 2026.

2 Humphreys, Todd. "lab2." Feb. 2026.

3 Humphreys, Todd. "Main." Jan. 2026.