CONTROL OF A FOLDING QUADROTOR WITH A SLUNG LOAD USING INPUT SHAPING

A Thesis
Presented to
The Academic Faculty

By

Nicholas Andrew Johnson

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Mechanical Engineering

Georgia Institute of Technology

May 2017

CONTROL OF A FOLDING QUADROTOR WITH A SLUNG LOAD USING INPUT SHAPING

Approved by:

Dr. William E. Singhose, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Nader Sadegh
School of Mechanical Engineering
Georgia Institute of Technology

Dr. I. Charles Ume
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: April 3, 2017

ACKNOWLEDGEMENTS

I would like to acknowledge the support I have received in my endeavors from pre-school to graduate school. I would like to thank Dr. Singhose for guiding me through this Masters project, my family who have gotten me to this point, my friends who made the ride enjoyable, and God for blessing me every day. AMDG.

I'd also like to thank the past groups who have worked on this quadrotor project, whose work I am building from.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Quadrotors are being used in an increasing number of applications. One such application is in carrying suspended payloads. However, as the payload swings — due to quadrotor motion or due to a disturbance — it also pulls on the quadrotor. The resulting force and torque from the payload can be significant and destabilize the quadrotor. Additionally, the swinging may be undesirable for fragile payloads, or the payload may collide with an obstacle. A typical approach is to allow the swinging to damp out, but this takes away from the limited battery life of the quadrotor.

Instead, input shaping techniques can be implemented in the software to reduce the payload oscillations. Input shaping has been shown in similar systems (cranes, double pendula, and helicopters) to significantly reduce the amount of residual oscillation after a maneuver. Because it is a passive technique, the position of the payload does not need to be obtained in a motion capture facility or by using computationally expensive observer algorithms. This allows the quadrotor to be used in a variety of applications, including in disaster relief and cargo transport where cost is a factor.

In this thesis, the dynamics of the quadrotor and payload are derived and explained in depth, including physicalparameters not usually considered. Then PID feedback is explored in controlling the system. In addition to the simulation results, hardware limitations must also be considered. Input shaping is then integrated into the system. The simulations show that input shaping greatly reduces unwanted residual oscillations. Finally, a folding quadrotor design is presented. Quadrotors must have large frames in order to carry large payloads, and a folding design saves space when transporting the quadrotor between flights. The quadrotor can then be placed in the trunk of a car or deployed from a rocket to reach an inaccessible area to provide aid.

CHAPTER 1
INTRODUCTION

1.1 Motivation

Quadrotors are increasingly popular rotorcraft that are seeing many applications due to their size and maneuverability. They have been used in surveillance, defense, photography, and videography. With four rotors and six degrees of freedom, quadrotors are underactuated; therefore, control techniques are critical to prevent unstable or uncontrollable behavior.

Another use of quadrotors is in carrying slung loads — payloads suspended from a single suspension point. Helicopters often carry large payloads, such as logs in remote-logging operations and even people in rescue missions. A quadrotor can be seen as scaled-down, more maneuverable version of a helicopter. In effect, they can both be seen as flying cranes with eight degrees of freedom, able to perform maneuvers that other vehicles cannot as easily while carrying a payload to a destination. For instance, food and medicine can be brought to an otherwise inaccessible region in an emergency. In humanitarian efforts, perishable foods can be unloaded off a ship more quickly, preventing spoilage and increasing the efficiency of the unloading process.

A problem with slung loads is that they pull on the quadrotor as they swing, as explored in helicopters by Adams [1]. With a large enough payload mass and with a too-large amplitude of swing, the payload can destabilize the quadrotor, causing it to fall and potentially crash. Feng et al. [2] showed that increasing the mass of the payload increases its effect on the quadrotor.

When the payload is not attached at the center of gravity of the quadrotor, load-attitude coupling takes place, causing the system to act similarly to a double pendulum. Swinging may also be undesirable in general for fragile payloads or when there are obstacles that

the payload could collide into. Often, the payload swing is allowed to damp out due to drag, but this delay reduces efficiency and consumes battery life in an unproductive way. Battery capacity is limited in a single flight of a quadrotor since the weight of each battery is an additional load that the quadrotor must carry. Additionally, certain operator commands may unintentionally antagonize the payload oscillations if they are not in the correct phase. These commands can cause more swinging, causing even greater delays or undesirable behavior.

In order to prevent collisions, maximize productive battery usage, and ensure the safety of payloads and the environment, the amplitude of the payload oscillations must be reduced.

Since the slung load adds two additional degrees of freedom, control techniques must be investigated for this system. However, much research has involved a system within a motion-capture facility [3]–[7], where the eight degrees of freedom were known with high accuracy and could be used as reliable feedback to the system. This is an expensive solution that may not be in place for real-world applications, such as delivering medicine after a natural disaster. For these types of situations, control techniques must be limited to the hardware of the quadrotor.

1.2 Quadrotor model

In order to understand the dynamics of a quadrotor and how the payload affects the system, both the quadrotor and quadrotor-with-payload systems are modeled for simulation in MATLAB. The quadrotor model has been derived previously, but further depth is provided in these derivations.

Fusato et al. [8] presented the model of a helicopter with a slung load. The model of the quadrotor with a slung load is very similar. Others have presented more complex models, but several simplifying assumptions are made to reduce the complexity of the quadrotor model. Salih et al. [9] used the small angle assumption to state that the local angular velocity was approximately identical to the global angular velocity. This assumption removed three

state variables, simplifying the model. Kaya et al. [10] went into depth on blade element theory and aerodynamic effects. They used parameter estimation to obtain the relationships between thrust and angular velocity as well as torque and angular velocity. Mian et al. [11] implemented a gearbox model to determine how motor speeds evolved.

These relationships are simplified before being used in the presented model, as others have done. These simplifications hold because the quadrotor carrying a payload that is presented is not performing aggressive maneuvers. The more complex models are linearized about hover conditions to yield the presented equations. When aggressive maneuvers are considered, the validity of simulations would benefit from a more complex model.

In addition to the dynamic model, the method of controlling the model in simulation is PID feedback and its derivatives. This is selected due to its relative simplicity to implement. Li et al. [12] condensed the dynamics of an unloaded quadrotor into linear time-invariant state space form and derived transfer functions. However, the dynamics of the loaded quadrotor are highly nonlinear and cannot be expressed in linear time-invariant form without oversimplifying the system. One such simplification was presented by Hehn et al. [4], in which the mass of the payload was small enough that the payload hardly affected the quadrotor. However, this assumption is not robust enough to be used in this thesis, as quadrotors may be required to lift heavy payloads. Erginer et al. [13] also used PID control on an unloaded quadrotor but implemented computer vision techniques in the controller. Hancer et al. [14] used PID control on an unloaded quadrotor but used an observer to account for disturbances, such as wind gusts.

Nonlinear and optimal control techniques have also been used, but these often acted on both the quadrotor and payload state variables rather than just those of the quadrotor.

Al-Younes et al. [15] compared PID control to linear quadratic regulator (LQR) and adaptive integral backstepping (AIB) controllers on an unloaded quadrotor. The AIB controller showed reduced settling times compared to the PID and LQR controllers, as designed. They stated that the AIB controller estimated modeling errors and increased the

robustness of the system against disturbances.

de Crousaz et al. [16] presented a quadrotor-with-payload model that used stochastic linear quadratic (SLQ) control, which is a form of nonlinear trajectory optimization. They described the "window task," where a loaded quadrotor navigated through a narrow opening. This required the payload to intentionally swing at a large angle in order for both the quadrotor and payload to make it through the window. They also investigated the case in which the payload cable lost tension. A similar test was shown by Mellinger et al. [5], where an unloaded quadrotor was subject to the window test. This test was experimentally performed using a trajectory generated by sequential composition.

Feng et al. [17] presented adaptive control on the attitude and altitude of the quadrotor in order to handle the force and torque that the payload induced on the quadrotor. Palunko et al. [6] took adaptive control a step further for when the payload displaces the center of gravity of the quadrotor from its mass center. Palunko et al. [7] generated an optimal trajectory using dynamic programming.

Faust et al. [3] presented a neural network solution, in which the system was trained to generate swing-free trajectories. By rewarding the equilibrium state and penalizing the distance from the goal state in addition to the payload swing, a robust policy was formed after several hundred iterations. However, as presented, the payload swing grew large if the distance from the goal state was large, as the policy prioritized achieving the goal state over maintaining a low payload swing throughout.

However, a goal of this thesis is to present a controller that does not need to know the position of the payload at any time. Therefore, optimal control techniques that take the payload position into account are out of scope, and a PID controller receiving quadrotor states as feedback is presented.

1.3 Input shaping

Input shaping is proposed as a way to reduce residual payload oscillations. Input shaping is a passive technique where the input command is modified to produce a more desirable response. The position of the payload is not needed in order to modify the command, so this method of control is advantageous in a wide range of situations outside of a motion capture facility. The only two pieces of information needed are the natural frequency and the damping ratio. Input shapers can be designed to be more robust to modeling errors. This can be useful for a quadrotor that must be able to carry a variety of payloads, as the input shaper would not necessarily have to be changed based on the mass of the payload.

Vaughan et al. [18] used input shaping to reduce the residual oscillations of a double pendulum system. By implementing a Specified Insensitivity (SI) shaper, they were able to target the frequencies of both pendula. By adding a tolerable amount of residual oscillation, the SI shaper became more robust to modeling errors, such as different payload lengths. This input shaper improved the usability of a tower crane system. A human trial was set up in which operators navigated a payload through an obstacle course. Both the average completion time and average number of collisions were reduced with the SI shaper than in the unshaped trials.

Adams [1] showed that input shaping on a constrained RC helicopter was an effective way to reduce payload oscillations. For manned helicopters, a residual attitude oscillation of 0.5 degrees is considered excessive. Larger oscillations threaten the stability of the craft, and the payload may have to be jettisoned for the craft to regain stability.

Because of the effectiveness of input shaping in similar systems, in this thesis, the effectiveness of input shaping on reducing the residual oscillations of payloads is investigated on quadrotors carrying slung loads. Kozak et al. [19] presented multiple methods of judging input shaping performance. However, only the amplitude of the payload oscillation is investigated here.

Sadr et al. [20] previously implemented input shaping on a loaded quadrotor following a path. However, due to sensor limitations, the position estimation of the quadrotor is not reliable enough to use over time. The quadrotor may use GPS and integrations of accelerometer readings to estimate position. However, GPS is not precise, and integrating the noise of an accelerometer causes errors to build over time. Achtelik et al. [21] stated that several challenges in micro aerial vehicles (MAVs) include integrating acceleration noise and computational demand. If position is not a reliable quantity to use, control techniques must take advantage of only the more reliable parameters, such as attitude, which can be obtained through an extended Kalman filter and several sensors. Additionally, input shaping is less computationally expensive than other techniques, including observers. Therefore, this thesis focuses on applying input shaping to attitude commands, although input shaping on velocity commands is also discussed.

1.4 Folding mechanism

In order to carry payloads of an appreciable size, the motors and propellers must be of adequate rating and size. To fit the larger propellers, the quadrotor itself must be made bigger to increase controllability and so that the propellers do not collide with each other. The additional weight results in additional thrust required to hover, requiring larger and heavier batteries to maintain flight time, which also increase the weight. The large wingspan of the resulting quadrotor is impractical to transport, such as in the trunk of a car.

It is then beneficial to reduce the size of the quadrotor during transport with the ability to expand it to full size for flight. A folding quadrotor is presented, where the wingspan of the quadrotor is reduced in its folded state. In this design, folding the arms inward stretches springs,which contract when the arms are released. This was designed for the quadrotor being transported to a location in a confined space. Upon being ejected, the unfolding would happen quickly, and the quadrotor could then begin operation. A 3D model of the quadrotor, including the folding mechanism, is developed in SolidWorks.

Mintchev et al. [22] presented a different type of folding quadrotor, where the folded arms wrapped around the frame. As the rotors spun in the folded configuration, the torque on the arms caused them to unfold. Using thrust or torque to unfold the arms is not done in the presented design because the arms fold down. Any thrust imbalance would cause the arms to unfold unevenly, causing the motors to be at an angle with respect to their intended orientation. Additionally, if the arms wrapped around the frame, then the rotors would collide with the electronic components that lie on top of the frame.

1.5 Thesis contributions

This thesis adds to the field of micro aerial vehicles and their applications, specifically when carrying slung loads outside of a motion capture facility. The main contributions are:

1. Nonlinear dynamics derivation and simulations
2. Evaluation of input shaping on attitude profiles
3. Folding quadrotor test platform

1.6 Thesis outline

Chapter 2 describes the dynamics of the quadrotor and quadrotor-with-payload systems. Chapter 3 presents the advantages and disadvantages of several feedback controllers. Chapter 4 investigates the usefulness of input shaping in reducing residual payload oscillations in the presented model. Chapter 5 presents the design and design decisions of the folding quadrotor. Chapter 6 summarizes the presented work and suggests topics of future work. Finally, Appendix A shows the MATLAB code used for simulations.

CHAPTER 2

QUADROTOR AND PAYLOAD DYNAMICS

The dynamics of the quadrotor-with-payload platform have been derived previously, but many interesting dynamical effects have often been left out.

The 3D model and sign conventions of the quadrotor with a slung load are illustrated in Figure 2.1.

2.1 Quadrotor dynamics

The four inputs to the system are the four motor speeds. However, these can be treated as thrust, T, and three torques, $\tau = \begin{bmatrix} \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$ by the following relationship:

$$\begin{bmatrix} T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 \\ lk_1 & 0 & -lk_3 & 0 \\ 0 & lk_2 & 0 & -lk_4 \\ b_1 & -b_2 & b_3 & -b_4 \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{2.1}$$

where ω_i is the angular speed of each of the four motors, k_i is the thrust coefficient of each motor, b_i is the torque coefficient of each motor, and l is the horizontal distance from the center of gravity to the center of the motor. k_i and b_i can be found experimentally. In simulation, it is convenient to abstract away ω_i and deal only with thrust and torque. However, ω_i should be monitored in some way to ensure that unrealistic values are not used. For convenience, let $T_r = \begin{bmatrix} 0 & 0 & T \end{bmatrix}^T$ be the thrust vector in quadrotor-local coordinates, which always points locally upward.

The parameter r represents the $\begin{bmatrix} x & y & z \end{bmatrix}^T$ position; ξ is the $\begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ attitude (pitch, roll, and yaw); g is gravity; and R is a 3×3 ZXY rotation matrix going from

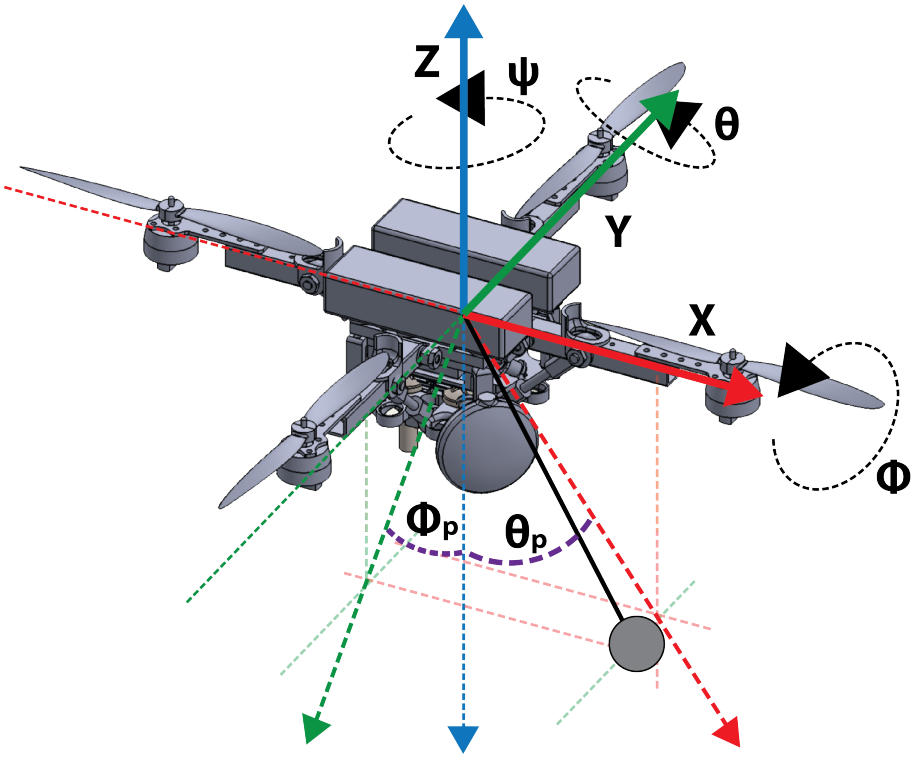

Figure 2.1: The quadrotor model and sign conventions.

quadrotor-local space to inertial-global space, given by:

$$R(\xi) = \begin{bmatrix} c_\theta c_\psi - s_\phi s_\theta s_\psi & -c_\phi s_\psi & c_\psi s_\theta + c_\theta s_\phi s_\psi \\ c_\theta s_\psi + c_\psi s_\phi s_\theta & c_\phi c_\psi & s_\psi s_\theta - c_\psi c_\theta s_\phi \\ -c_\phi s_\theta & s_\phi & c_\phi c_\theta \end{bmatrix} \tag{2.2}$$

where $c_\phi = \cos(\phi)$, $s_\phi = \sin(\phi)$, $t_\phi = \tan(\phi)$, etc.

Near-hover conditions are assumed in order to avoid the additional rotation matrix between quadrotor-local angular velocity (often $\begin{bmatrix} p & q & r \end{bmatrix}^T$) and global angular velocity ($\begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$). In other words, the quadrotor-local coordinate axes are always approximately aligned with the global coordinate axes (small-angle approximation). This assumption falters when performing aggressive maneuvers, but such maneuvers are not considered here.

2.1.1 Linear dynamics

The equation of motion for the linear dynamics is:

$$\ddot{r} = \frac{1}{m_{\mathrm{q}}}(F_g + F_{\mathrm{d}} + RT_r + RF_{\mathrm{p}}) \tag{2.3}$$

where m_{q} is the mass of the quadrotor, F_g is the gravitational force, F_{d} is the drag force, and F_{p} is the force of the payload on the quadrotor. Vector quantities multiplied by R are quadrotor-local and must be brought into the global frame.

The gravitational force is:

$$F_g = \begin{bmatrix} 0 \\ 0 \\ -m_{\mathrm{q}}g \end{bmatrix} \tag{2.4}$$

and the drag force is:

$$F_{\mathrm{d}} = -\frac{1}{2}C_{\mathrm{d}}A\rho_{\mathrm{air}} \left\|\dot{r}\right\| \dot{r} \tag{2.5}$$

where C_{d} is the drag coefficient, A is the exposed area for drag consideration, and ρ_{air} is the density of air, as a constant, although it can be a function of altitude or pressure if necessary. It should be noted that thrust would also change with changing air density since the mass flow rate of air through the propellers would change. In other words, assuming air density is just a function of altitude, the thrust coefficient k is also a function of altitude,

$$k(z) = k_0\frac{\rho(z)}{\rho_0} \tag{2.6}$$

where k_0 is the value of the thrust coefficient k at a reference pressure ρ_0. However, air density will be treated as a constant here.

Without drag, an artificial maximum speed would have to be imposed on the system to

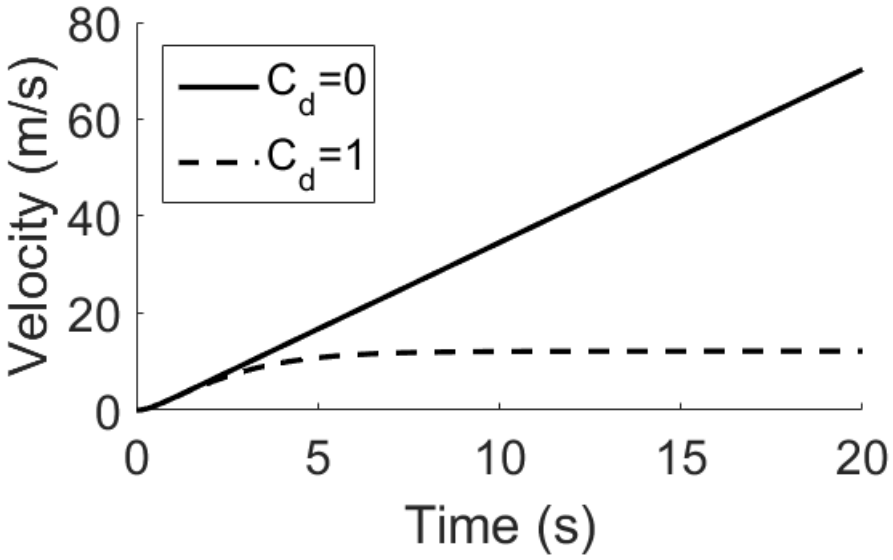

Figure 2.2: The effect of drag on creating a maximum velocity.

prevent it from growing to an unrealistic value. The effect of drag on limiting the maximum speed is shown in Figure 2.2, comparing drag coefficients $C_d = 0$ and $C_d = 1$, and the corresponding quadratic drag force is plotted in Figure 2.3 for the case where drag is considered.

When a cruising attitude is achieved, part of the thrust is required to maintain the altitude of the quadrotor, and the rest of the thrust propels the quadrotor forward. The quadrotor accelerates to a velocity, limited by when the drag force exactly opposes the forward component of thrust. If there is no drag in the model, then there is no bound on the velocity. Therefore, the maximum speed of the quadrotor, given drag, can be calculated as a function of ξ:

$$\|\dot{r}_{\text{max}}(\xi)\| = \sqrt{\frac{2R(\xi)T}{\rho_{\text{air}} C_{\text{d}} A}} \tag{2.7}$$

The force of the payload on the quadrotor, F_{p}, is discussed later.

With a constant thrust and nonzero ϕ or θ, the dynamic equations indicate that the quadrotor will descend as it moves forward due to gravity being greater than the upward component of thrust. Therefore, the thrust must increase — meaning the motors must spin

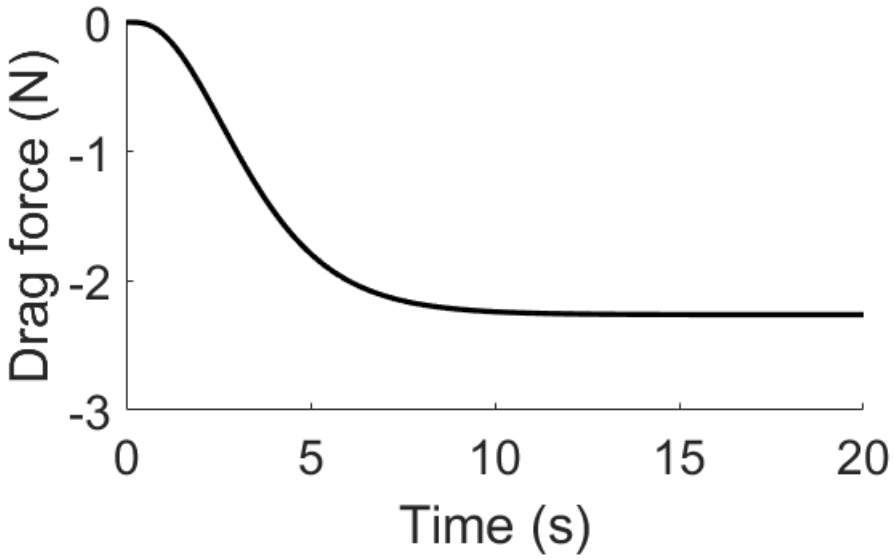

Figure 2.3: The drag force that creates a cruising velocity.

faster — to avoid falling too much. Limiting the maximum angle or maximizing excess thrust ($T_{\text{max}} - mg$) improves the ability of the quadrotor to maintain or recover altitude. Figure 2.4 highlights the altitude response when the excess thrust is not enough to also control the altitude of the quadrotor. When $T = 1.68g$ N, there are 680 grams of excess thrust, 40 percent of the maximum thrust, which is sufficient for control. For $T = 1.06g$ N, there are 60 grams of thrust, about 6 percent of the maximum thrust, which is shown to be too little to both maintain altitude and move forward at ξ_{des}. More excess thrust also allows for the quadrotor to pick up larger payloads and remain stable.

To fix this in practice, motors with a higher kv rating (RPM per voltage) should be chosen to increase T_{max}, or the overall weight of the quadrotor should be reduced. For moving from one point to another, if neither fix is feasible, then the controller may need to alternate between moving closer to the target and recovering altitude so that the quadrotor eventually reaches its destination without falling too far and crashing.

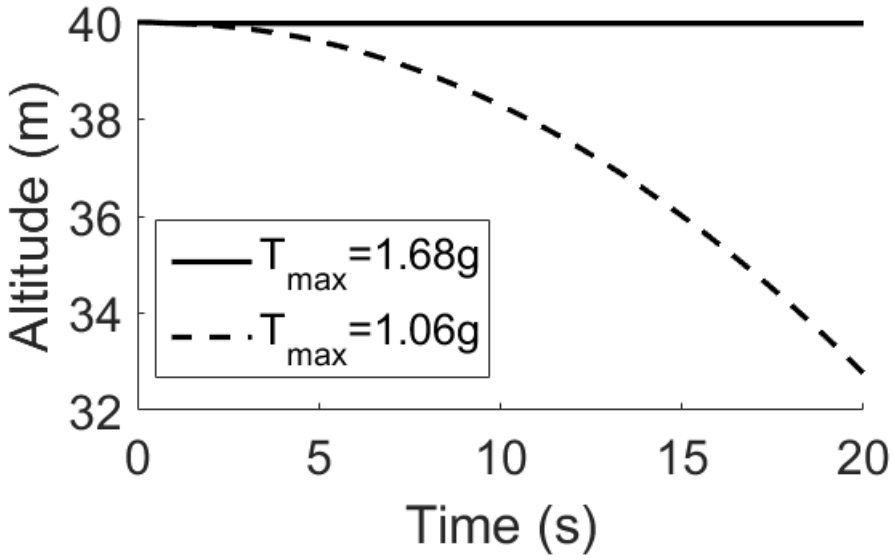

Figure 2.4: The effect of not having enough excess thrust to both maintain altitude and move forward.

2.1.2 Rotational dynamics

The equation of motion for the rotational dynamics is:

$$\ddot{\xi} = I^{-1}(\tau_{\mathrm{d}} + \tau + \tau_{\mathrm{p}} - \dot{\xi} \times (I\dot{\xi})) \tag{2.8}$$

where I is the diagonal inertia matrix, τ_{d} is the torque due to drag, and τ_{p} is the torque due to the payload. The drag torque is:

$$\tau_d = (Rr_{\mathrm{cop}}) \times F_{\mathrm{d}} \tag{2.9}$$

where r_{cop} is the center of pressure, i.e. where the drag force is applied relative to the center of gravity, as illustrated in Figure 2.5. If drag is applied at the COG, it does not induce a torque. If applied above the COG in the quadrotor-local space, the drag torque would oppose the motion and act as damping. This yields a more stable but less maneuverable system, and the equilibrium angle is less than the expected value in magnitude. If applied below the COG, the drag torque would be in the same direction as the motion, and the

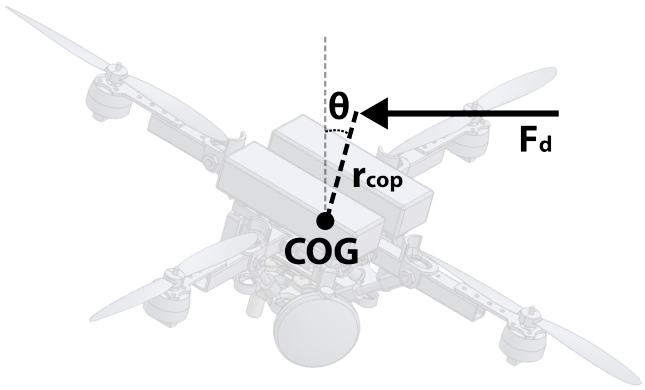

Figure 2.5: An illustration of the center of pressure, r_{cop}.

equilibrium angle is greater than the desired angle. This yields a more maneuverable but less stable system, as the quadrotor can more easily tip over.

The effect of the center of pressure on ϕ and $\dot{r}_{\text{max}}$ is shown in Figures 2.6 and 2.7, respectively. Let $r_{\text{cop}} = \begin{bmatrix} 0 & 0 & z_{\text{cop}} \end{bmatrix}^T$. The equilibrium angle for $z_{\text{cop}} = 0$ is the desired angle. For $z_{\text{cop}} > 0$ cm, the equilibrium angle is less than the desired angle in magnitude, causing the quadrotor to move slower but not be at risk for tipping over. For $z_{\text{cop}} < 0$ cm, the quadrotor exceeds the desired angle in magnitude and is at risk of tipping over. The steeper angle permits a higher maximum velocity, however.

The center of pressure is a function of the geometry of the quadrotor. The value used in simulation is assumed to be above the COG, providing a damping torque, but this was not experimentally determined.

The cruise attitude can be found with:

$$\xi = \xi_{\text{des}} + \frac{r_{\text{cop}} \times F_{\text{d}}}{K_{\text{p},\xi}} \tag{2.10}$$

where ξ_{des} is the desired attitude and $K_{\text{p},\xi}$ is the proportional gain of a PD controller, discussed in the next section.

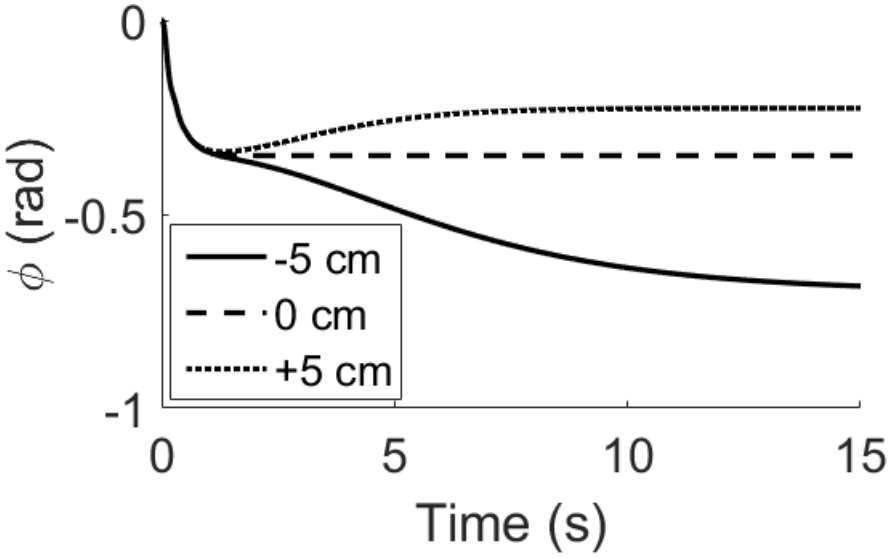

Figure 2.6: The effect of r_{cop} on cruise attitude.

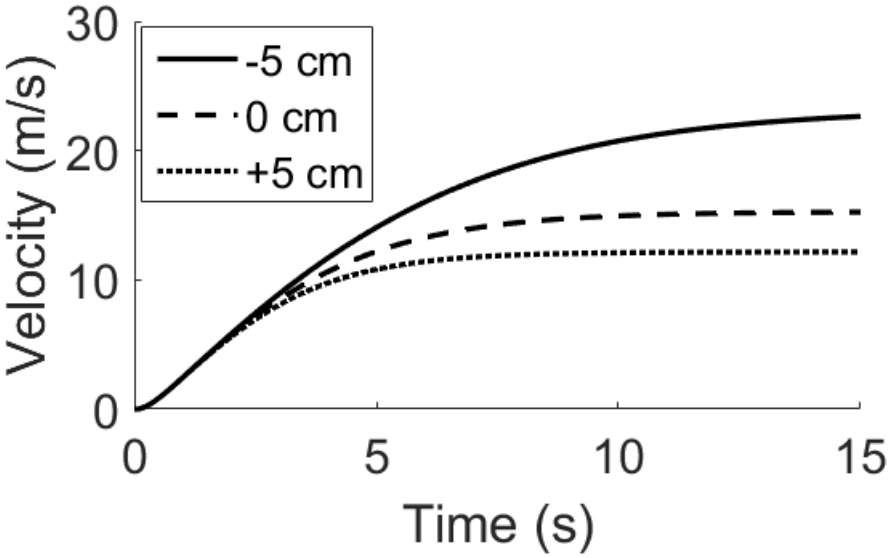

Figure 2.7: The effect of r_{cop} on cruise velocity, due to the effect on ϕ.

The torque due to the payload is discussed later.

2.1.3 Dynamics in simulation

The nonlinear dynamics are simulated in MATLAB's *ode45*. The simulation code is presented in Appendix A.

The control of a quadrotor as presented here utilizes two or three control loops. The first is attitude control, which attempts to follow an attitude profile or hold a desired attitude. The three τ inputs come from this controller. The second is altitude control for controlling z and, thereby, the T input. A third, optional controller is one to control position or velocity. This can be xy-only or include z as well, in which case the altitude controller is simply one aspect of the position/velocity controller. Position control either follows a path or attempts to reach a single destination. Velocity control follows a profile or maintains a cruising velocity. Either of these is performed by transforming x and y or $\dot{x}$ and $\dot{y}$ inputs into ξ inputs and controlling attitude.

In order to perform simulations by integrating the dynamics, the thrust and torque evolve following:

$$\dot{T} = c_T(T_{\text{des}} - T) \tag{2.11}$$

and

$$\dot{\tau} = c_\tau(\tau_{\text{des}} - \tau) \tag{2.12}$$

where c_T and c_τ are rate-limiting constants, needed because real motor cannot change speeds instantaneously. Additionally, T_{des} is the desired thrust, and τ_{des} is the desired torque.

The desired values come from:

$$\begin{gathered} T_{\text{des}} = (m_{\text{q}} + m_{\text{p}})(g + c_\phi c_\theta \ddot{z}_{\text{des}}) \\ T_{\text{min}} \leq T_{\text{des}} \leq T_{\text{max}} \end{gathered} \tag{2.13}$$

where m_{p} is the mass of the payload, T_{min} and T_{max} bound the desired thrust, $c_\phi c_\theta$ is the projection onto the horizontal plane, and $\ddot{z}_{\text{des}}$ is the desired acceleration in the z direction. Additionally,

$$\tau_{\text{des}} = \text{PD}(\xi_{\text{des}} - \xi) \tag{2.14}$$

where ξ_{des} is the desired attitude, and $\text{PD}(\cdot)$ is a proportional-derivative feedback controller. Other controllers can be used in this step, but this thesis will utilize PID control, or subsets thereof.

In open-loop position or velocity control, ξ_{des} would be the ultimate input to the system. For closed-loop control, its value can be automatically set by any desired outer controller, following:

$$\begin{gathered} \xi_{\text{des}} = \begin{bmatrix} \sin^{-1}(-\frac{\ddot{y}_{\text{des}}}{\|\ddot{r}_{\text{des}}\| c_\theta}) \\ \sin^{-1}(\frac{\ddot{x}_{\text{des}}}{\|\ddot{r}_{\text{des}}\|}) \\ 0 \end{bmatrix} \\ \|\xi_{\text{des}}\| \leq \xi_{\text{max}} \end{gathered} \tag{2.15}$$

where the desired attitude magnitude is bounded. Yaw control is often used for planning optimal trajectories, where performing an off-axis maneuver is less stable and less preferable than adjusting ψ and moving along only one axis. However, yaw control is ignored here. Coupling between ϕ and θ arises due to the direction the motors spin in conjunction with the conservation of angular momentum, as can be seen by the $\frac{1}{c_\theta}$ in the expression for ϕ_{des}, i.e. the first term of ξ_{des}.

Additionally,

$$\ddot{r}_{\text{des}} = \begin{cases} \text{PID}(r_{\text{path}} - r) \\ \text{PD}(r_{\text{final}} - r) \\ \text{PI}(\dot{r}(t) - \dot{r}) \\ \text{etc} \end{cases} \tag{2.16}$$

where PID granular path following, PD long-range navigation, PI velocity profile tracking, or other controllers are used to convert the primary input of the system into a desired acceleration. Only PID feedback control is considered here, but others have used alternative control methods. Control is the topic of Chapter 3 and will be explained more thoroughly there.

If desired, the charge of the battery supply can be modeled as well. Individual motor thrust, T_{i}, and current draw, I_{i}, exhibit a linear relationship. The experimental motor data, shown in Table 2.1, is linearly fit and plotted in Figure 2.8. Zeroing the y-intercept, the slope, s_{i}, directly transforms thrust into current draw and is assumed to be the same value for each motor. The additional current draw from the microcontroller and sensors, I_{mc}, can be included to find the total current draw, I:

$$I = -\sum I_{\text{i}} - I_{\text{mc}} = -\sum s_{\text{i}} T_{\text{i}} - I_{\text{mc}} \tag{2.17}$$

Current can be integrated to find the change in battery charge as a function of time, with the initial battery charge being the number of coulombs the battery supply starts with. This could be relevant if one wanted to optimize energy usage or end a simulation early if the charge reaches zero. The simulation could also continue beyond the battery being drained, but T_{min}, T_{des}, and τ_{des} must all be set to zero to simulate an unpowered quadrotor.

The quadrotor and payload parameters used in simulation are presented in Table 2.2. The values are taken from the physical quadrotor describes in Chapter 5, the Solidworks 3D model, literature, and appropriate estimations.

Table 2.1: Experimental data for motor current draw versus generated thrust.

Thrust (N)	Current (A)
0.0000	0.04
0.3924	0.366
0.7848	0.685
1.0791	1.056
1.4225	1.433
1.7658	1.83
2.0601	2.23
2.4525	2.73
2.8449	3.52
3.5316	4.38
3.5806	4.57
3.6297	4.64
3.6788	4.73

2.2 Payload dynamics

The dynamics of the payload are highly nonlinear, with the majority of terms being coupled or higher order.

In previous literature, the payload dynamics are not derived in the quadrotor-local system. Instead, a new H frame of reference, x_H, y_H, z_H, is the quadrotor-local space multiplied by a rotation matrix. This frame of reference is illustrated in Figure 2.9. A rotation that could be used with the presented model could be $R(\begin{bmatrix} 0 & \pi/2 & \pi \end{bmatrix}^T)$, yielding $x_H = -x$, $y_H = -y$, and $z_H = -z$. However, such a rotation is not performed in these derivations and is only mentioned to explain the difference in form between the presented work and previous literature. The derivations in either case are mathematically identical and yield the same equations of motion.

2.2.1 Payload dynamics derivation

A suspension offset, $r_{\text{susp}} = \begin{bmatrix} 0 & 0 & -d \end{bmatrix}$, is the vector distance from the center of gravity of the quadrotor to where the payload is suspended from, where d is the magnitude of the

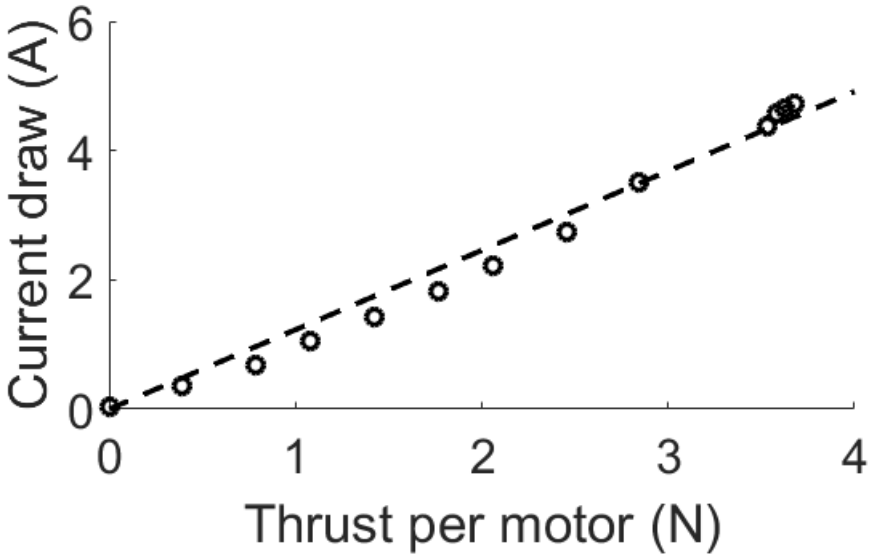

Figure 2.8: The linear relationship between motor current draw and generated thrust.

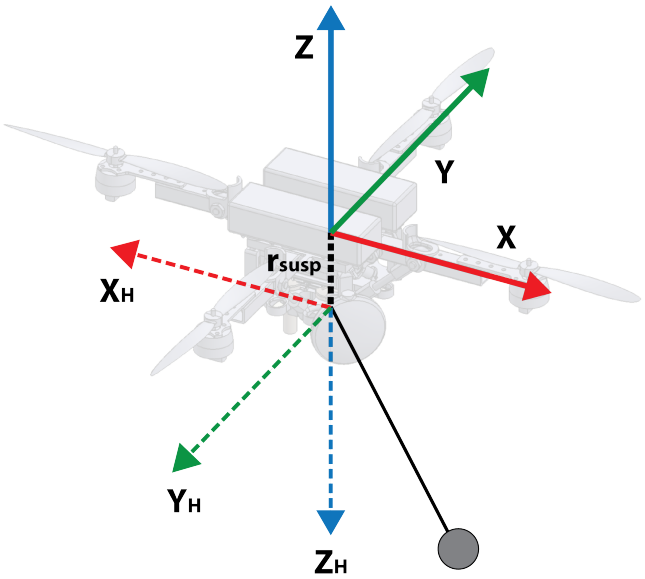

Figure 2.9: An illustration of the suspension offset, r_{susp}, and the H frame.

Table 2.2: Parameters used in MATLAB simulations.

Parameter	Value	Unit
m_{q}	1.0	kg
g	9.81	m/s^2
I_{xx},I_{yy}	6.8e-9	kgm^2
I_{zz}	4.5e-9	kgm^2
L	0.155	m
r	8.89	cm
T_{min}	1.00	N
T_{max}	16.48	N
ρ_{air}	1.22	kg/m^3
k_{i}	1.28e-7	N/RPM^2
b_{i}	1e-9	Nm/RPM^2
A	0.025	m^2
C_{d}	1	-
r_{cop}	[0, 0, 5.0]$^{\text{T}}$	cm
m_{p}	0.073	kg
L_{p}	1.0	m
d	0	cm
$C_{\text{d,p}}$	0.5	-
A_{p}	0.0081	m^2
ξ_{max}	0.35	rad
c_T	16	s^{-1}
c_τ	20	s^{-1}

distance. An offset can be added in general to all three axes, but only a vertical offset is considered in these derivations, as shown in Figure 2.9.

The position of the payload relative to this suspension point, r_l, is given by:

$$r_{\text{l}} = -L_{\text{p}} \begin{bmatrix} -c_{\phi_{\text{p}}} s_{\theta_{\text{p}}} \\ s_{\phi_{\text{p}}} \\ c_{\theta_{\text{p}}} c_{\phi_{\text{p}}} \end{bmatrix} \tag{2.18}$$

where L_{p} is the length of the massless and inelastic cable, ϕ_{p} and θ_{p} are the tangential and

orthogonal angles of the payload. The transformation matrix that is multiplied by L_p is:

$$XY^{-1}\begin{bmatrix}0\\0\\1\end{bmatrix} = Y^{-1}X^{-1}\begin{bmatrix}0\\0\\1\end{bmatrix} \tag{2.19}$$

Including the suspension offset, the payload position relative to the quadrotor, r_p, is:

$$r_\mathrm{p} = r_l + r_\mathrm{susp} \tag{2.20}$$

Furthermore, if L_p is significantly larger than d, then:

$$r_\mathrm{p} \approx r_\mathrm{l} \tag{2.21}$$

Differentiating the global position of the payload, $r + r_\mathrm{p}$, the absolute velocity can be found to be:

$$\dot{r}_\mathrm{p} = \dot{r} + \dot{r}_\mathrm{l} + \dot{\xi} \times r_\mathrm{l} \tag{2.22}$$

and absolute acceleration is:

$$\ddot{r}_\mathrm{p} = \ddot{r} + \ddot{r}_\mathrm{l} + \ddot{\xi} \times r_\mathrm{l} + 2\dot{\xi} \times \dot{r}_\mathrm{l} + \dot{\xi} \times (\dot{\xi} \times r_\mathrm{l}) \tag{2.23}$$

2.2.2 Torque equilibrium

By finding $\ddot{r}_\mathrm{l}$, one can obtain both $\ddot{\phi}_\mathrm{p}$ and $\ddot{\theta}_\mathrm{p}$ from the chain rule multiplications. These are the angular accelerations that define the dynamics of the payload. To determine these accelerations, a moment balance is performed about the suspension point. Assuming $0 < \phi_\mathrm{p}, \theta_\mathrm{p} < \pi/2$, and $0 < \dot{\phi}_\mathrm{p}, \dot{\theta}_\mathrm{p}$, both the gravitational and drag forces cause a positive moment about both the x and y axes. Following this sign convention, the moment balance in

the quadrotor-local frame can be written as:

$$m_{\mathrm{p}} r_{\mathrm{l}} \times (\ddot{r} - \ddot{r}_{\mathrm{l}} - \ddot{\xi} \times r_{\mathrm{l}} - 2\dot{\xi} \times \dot{r}_{\mathrm{l}} - \dot{\xi} \times (\dot{\xi} \times r_{\mathrm{l}}) - g_{\xi} + f_{\xi}) = 0 \tag{2.24}$$

where g_{ξ} is the relative acceleration due to gravity, and f_{ξ} is the relative acceleration due to the drag force acting on the payload, both relative to the orientation of the quadrotor. The linear acceleration, $\ddot{r}$ holds the opposite sign of the terms containing r_{l} because of where each acceleration/force is located with respect to the suspension point, i.e. the lever arms with respect to the pivot point. This is in contrast to the absolute payload acceleration, (2.23), where all of the terms are added together.

The relative gravity is:

$$g_{\xi} = g \begin{bmatrix} -c_{\phi} s_{\theta} \\ s_{\phi} \\ c_{\theta} c_{\phi} \end{bmatrix} \tag{2.25}$$

where the transformation matrix multiplied by g is the XY^{-1} rotation matrix. When this vector is multiplied by the ZXY rotation matrix R, as in (2.3), the resultant gravity vector points downward in the inertial reference frame, as expected. Although the force of gravity causes a positive moment, the sign multiplied by g_{ξ} in (2.24) is negative because the force and the acceleration hold opposing signs as defined: $F_{g_{\xi}} = -m_{\mathrm{p}} g_{\xi}$.

As with gravity, drag is a global force that must be first be transformed into quadrotor-local space. The relative drag acceleration is:

$$f_{\xi} = R^{T} \frac{F_{\mathrm{d,p}}}{m_{\mathrm{p}}} \tag{2.26}$$

where R^{T} is the transpose of the rotation matrix R, and $F_{\mathrm{d,p}}$ is the drag force acting on the payload. Payload drag is calculated the same as for drag on the the quadrotor, (2.5), except with a different drag coefficient, area, and velocity: $C_{\mathrm{d,p}}$, A_{p}, and $\dot{r}_{\mathrm{p}}$.

2.2.3 Payload EOM

Expanding the torque equilibrium yields three equations — one of which is redundant since derivatives of L_p are zero — where $\ddot{\phi}_\mathrm{p}$ and $\ddot{\theta}_\mathrm{p}$ are the two unknowns. Let f_x represent the x component of the relative drag force on the payload, f_ξ, for ease of writing (and correspondingly so for f_y and f_z). Using Mathematica, the two angular accelerations are solved to be:

$$\begin{aligned}\ddot{\theta}_\mathrm{p} = \\ &-c_{\theta_\mathrm{p}} s_{\theta_\mathrm{p}} \dot{\phi}^2 + 2t_{\phi_\mathrm{p}} \dot{\theta}_\mathrm{p} \dot{\phi}_\mathrm{p} - 2c_{\theta_\mathrm{p}} \dot{\phi}_\mathrm{p} \dot{\psi} + c_{\theta_\mathrm{p}} s_{\theta_\mathrm{p}} \dot{\psi}^2 + \dot{\phi}(2s_{\theta_\mathrm{p}} \dot{\phi}_\mathrm{p} + (c^2_{\theta_\mathrm{p}} - s^2_{\theta_\mathrm{p}})\dot{\psi}) \\ &+t_{\phi_\mathrm{p}} \dot{\theta}(c_{\theta_\mathrm{p}} \dot{\phi} - 2\dot{\phi}_\mathrm{p} + s_{\theta_\mathrm{p}} \dot{\psi}) + (c_{\theta_\mathrm{p}}(f_x + g c_\phi s_\theta) + (f_z - g c_\theta c_\phi) s_{\theta_\mathrm{p}} + c_{\theta_\mathrm{p}} \ddot{x} \\ &+c^2_{\phi_\mathrm{p}} s_{\theta_\mathrm{p}} \ddot{z})/L_\mathrm{p}/c_{\phi_\mathrm{p}} + \ddot{\theta} + s_{\theta_\mathrm{p}} t_{\phi_\mathrm{p}} \ddot{\phi} - c_{\theta_\mathrm{p}} t_{\phi_\mathrm{p}} \ddot{\psi}\end{aligned} \tag{2.27}$$

and

$$\begin{aligned}\ddot{\phi}_\mathrm{p} = \\ &-s_{\phi_\mathrm{p}} c_{\phi_\mathrm{p}} \dot{\theta}^2 - s_{\phi_\mathrm{p}} c_{\phi_\mathrm{p}} \dot{\theta}^2_\mathrm{p} + s_{\phi_\mathrm{p}} c_{\phi_\mathrm{p}} (s_{\theta_\mathrm{p}} \dot{\phi} - c_{\theta_\mathrm{p}} \dot{\psi})^2 + 2c^2_{\phi_\mathrm{p}} \dot{\theta}_\mathrm{p}(-s_{\theta_\mathrm{p}} \dot{\phi} + c_{\theta_\mathrm{p}} \dot{\psi}) \\ &+\dot{\theta}(2s_{\phi_\mathrm{p}} c_{\phi_\mathrm{p}} \dot{\theta}_\mathrm{p} + (c^2_{\phi_\mathrm{p}} - s^2_{\phi_\mathrm{p}})(s_{\theta_\mathrm{p}} \dot{\phi} - c_{\theta_\mathrm{p}} \dot{\psi})) - (c_{\phi_\mathrm{p}}(f_y - g s_\phi) + (-f_z c_{\theta_\mathrm{p}} \\ &+g(s_\theta s_{\theta_\mathrm{p}} + c_\theta c_{\theta_\mathrm{p}}) c_\phi + f_x s_{\theta_\mathrm{p}}) s_{\phi_\mathrm{p}} + c_{\phi_\mathrm{p}} \ddot{y} - s_{\phi_\mathrm{p}}(-s_{\theta_\mathrm{p}} \ddot{x} + c_{\theta_\mathrm{p}} \ddot{z}))/L_\mathrm{p} \\ &+c_{\theta_\mathrm{p}} \ddot{\phi} + s_{\theta_\mathrm{p}} \ddot{\psi}\end{aligned} \tag{2.28}$$

NB: these equations rely on components and derivatives of both ξ and ξ_p. Generalized, they take the form of $\ddot{\xi}_\mathrm{p} = h(\dot{r}, \ddot{r}, \xi, \dot{\xi}, \ddot{\xi}, \xi_\mathrm{p}, \dot{\xi}_\mathrm{p})$.

A caveat is that these equations implicitly rely on $\ddot{r}$ and $\ddot{\xi}$. This means that the EOM of the quadrotor depend on the accelerations of the payload, as expected, but also the payload

EOM also depend on the quadrotor accelerations. In order to use the quadrotor accelerations in the payload EOM, best estimates or previous values of $\ddot{r}$ and $\ddot{\xi}$ should be used, depending on the implementation.

Finally, with everything else known, the absolute acceleration of the payload, $\ddot{r}_{\mathrm{p}}$, can be found in general using (2.23). The force and torque of the payload on the quadrotor are:

$$F_{\mathrm{p}} = m_{\mathrm{p}}(\ddot{r}_{\mathrm{p}} - g_{\xi} + f_{\xi}) \tag{2.29}$$

and

$$\tau_{\mathrm{p}} = (Rr_{\mathrm{susp}}) \times F_{\mathrm{p}} \tag{2.30}$$

respectively.

2.3 Model verification

The presented model was verified by comparing it to models in previous literature, including both helicopters and quadrotors with slung loads. Although the presented model has increased complexity, the equations for $\ddot{\phi}_{\mathrm{p}}$ and $\ddot{\theta}_{\mathrm{p}}$ aligned with those in previous literature. These equations have not usually been presented in previous works, but their respective models were input into Mathematica to solve for their payload EOM.

For the more complex dynamics, including relative drag, it was necessary to fix certain parameters to observe the effects of others. For instance, if the position of the quadrotor was fixed, then the quadrotor model could be rotated without inducing a velocity. In one example, the payload was attached at the COG of the quadrotor. The quadrotor was given a fixed attitude, $\xi \equiv \xi_0$, and the payload was given an initial angle, $\xi_{\mathrm{p}} = \xi_{\mathrm{p}0}$. No matter the ξ_0 chosen, the gravitational and drag forces acting on the payload should be the same for the same initial payload angle. Without the transformations in (2.25) and (2.26), gravity and drag would have affected the payload as functions of ξ, which is not physically accurate.

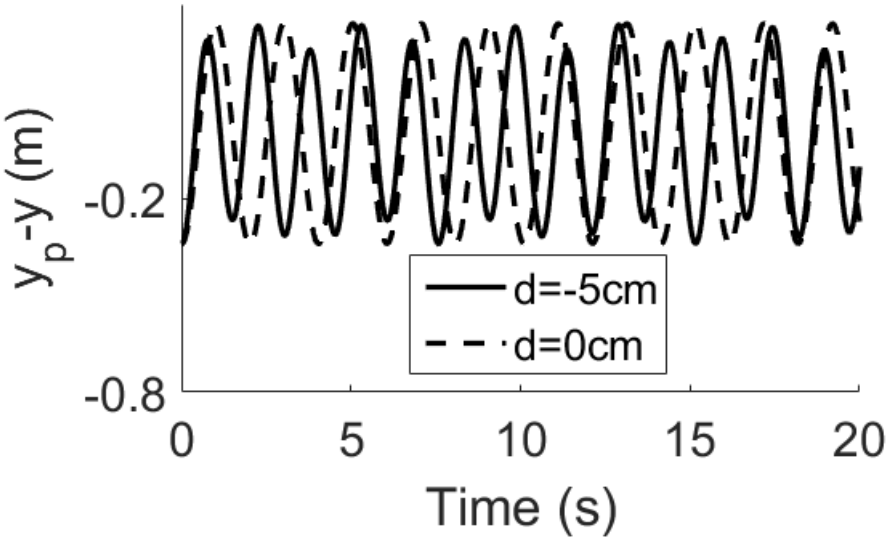

Figure 2.10: The effect of d on a quadrotor with a fixed position and free attitude.

2.4 Suspension offset

Assuming the payload is suspended from the center of gravity of the quadrotor, the swinging of the payload does not induce a torque. However, even a small d in r_{susp} causes the system to essentially act as a double pendulum. More precisely, the system exhibits load-attitude coupling, which has similar characteristics to a double pendulum.

The effect of d is shown in Figure 2.10, where the initial $\phi_{\text{p0}} = \frac{\pi}{9}$, $T_0 = mg$. The drag coefficients of both the quadrotor and payload are zero here, r is fixed, and the quadrotor is free to rotate about its COG. The $d = 0$ cm case is a normal sinusoidal pendulum response. However, the $d = -5$ cm case is a double pendulum, where the response is a combination of sinusoids and where the payload induces an angular acceleration in the quadrotor. The response is well-behaved compared to that of a chaotic double pendulum because the torque induced by the payload is small in this example.

In a typical system, where r is not fixed, the swinging behaves differently. Let $\xi_{\text{des}} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. For the specific controllers described in Chapter 3, the following effects occur. Figure 2.11 highlights that the amplitude of swinging of the double pendulum decays much faster than that when $d = 0$, where both drag coefficients are still zero. What is happening

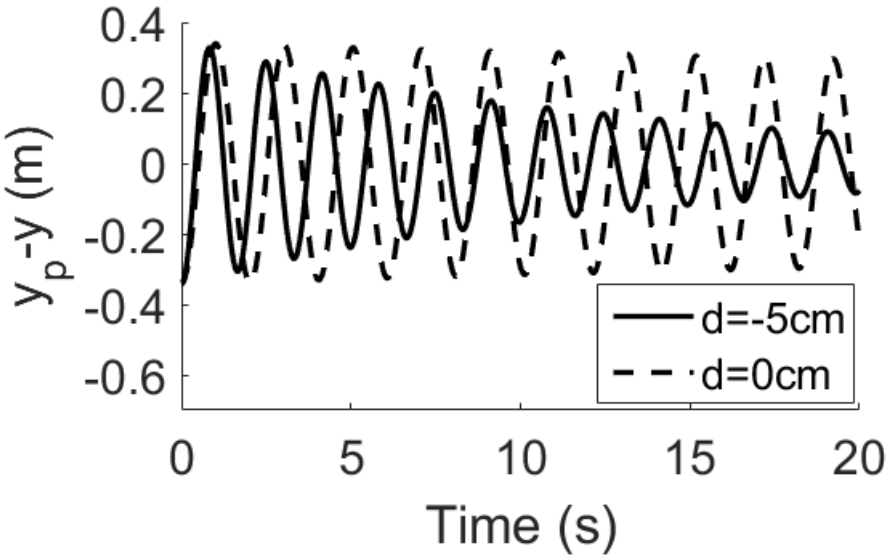

Figure 2.11: The effect of d on relative payload position while the quadrotor holds attitude.

is that as the payload induces a torque on the quadrotor, the quadrotor pitches or rolls to counteract the torque and maintain a zero attitude. This eventually damps out the payload response. However, because the quadrotor is rotating (when $d \neq 0$) and because the payload pulls on the quadrotor, the linear position of the quadrotor also changes, as depicted in Figure 2.12. Although the double pendulum has clear advantages in reducing payload oscillation when holding a zero attitude, it does require slightly more thrust, current, and energy. As is shown later, these advantages disappear when performing other types of maneuvers.

2.5 Stability and validity of the model

Based on (2.28), specifically the $\cos(\cdot)^{-1}$ terms, $\ddot{\theta}_\text{p}$ divergence will occur when ϕ_p approaches $\pm\pi/2$. For uniaxial x movements, where ϕ_p is undisturbed, $\ddot{\theta}_\text{p}$ cannot diverge if the other parameters are bounded.

Given all of the presented dynamics, the quadrotor-payload model can be seen to be a bounded-input bounded-output system as long as $-\pi/2 < \theta, \phi_\text{p} < \pi/2$. Although there are no mathematical restrictions on ϕ and θ_p, naturally they should also be kept within the aforementioned range.

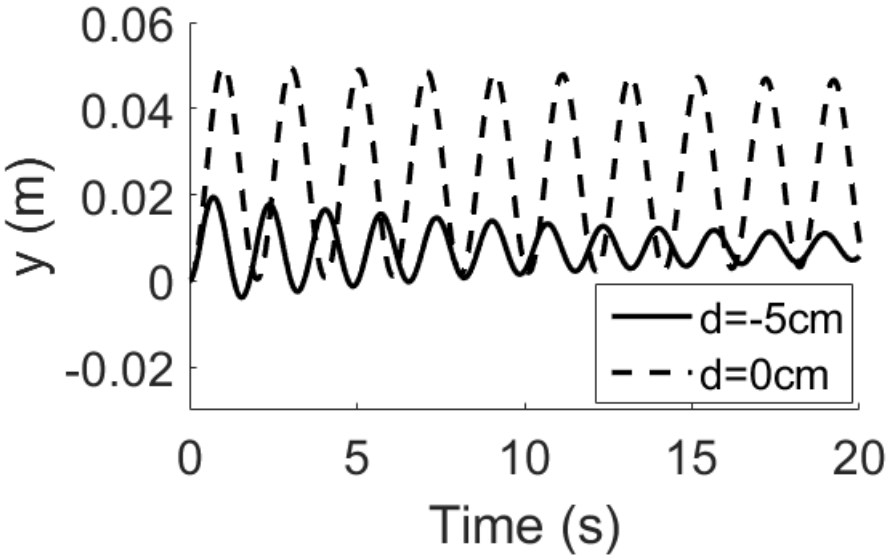

Figure 2.12: The effect of payload swing on quadrotor position while holding attitude.

In addition to boundedness, the definition of stability in this context will be refined. As highlighted in Figure 2.4, too little excess thrust will cause the system to go unstable. Based on the defined ξ_{max}, the minimum excess thrust can be calculated, under which the system should be treated as unstable. Moreover, the presented model is invalid at a large enough payload angle. A real cable would likely have slack in it at a large angle, and the inelastic cable assumption that this model relies on would no longer be valid.

2.6 Simplified models

The equations for the payload accelerations are complex, with many trigonometric calculations that must be performed. Even if the values of c_ϕ, c_θ, etc are calculated once and reused as needed, there is still much algebra to perform. The equations can be linearized about hover conditions, where all coupled and higher-order terms are assumed to be zero. Additionally, small angles can be assumed, where $s_\phi = \phi$, $c_\phi = 1$ etc, and where the drag force is assumed to be zero. After linearizing (2.27) and (2.28) using the aforementioned

assumptions, only terms containing gravity remain:

$$\ddot{\theta}_{\mathrm{p}} = \frac{g(\theta - \theta_{\mathrm{p}})}{L_{\mathrm{p}}} \tag{2.31}$$

and

$$\ddot{\phi}_{\mathrm{p}} = \frac{g(\phi - \phi_{\mathrm{p}})}{L_{\mathrm{p}}} \tag{2.32}$$

The force of the payload on the quadrotor would then simply be:

$$F_{\mathrm{p}} = -m_{\mathrm{p}} g_{\xi} \tag{2.33}$$

This degree of linearization is valid for small movements but loses accuracy as the quadrotor or payload velocity increases. Assuming the quadrotor is a pendulum with fixed position, Figure 2.13 shows that a small initial angle causes the linear and nonlinear forms of $\ddot{\phi}_{\mathrm{p}}$ to be virtually identical, even in the double pendulum case presented. However, Figure 2.14 shows the payload response to an attitude command for both the linearized and nonlinear accelerations. The linearized model does not capture the rich dynamics of the nonlinear form, and the two responses differ greatly.

A further simplification has been made in previous literature where $F_{\mathrm{p}} = 0$. This extreme linearization is not robust because of the simple fact that the weight of the payload pulls down on the quadrotor, even if swinging is not considered. It would only be valid for payload masses quite small in comparison to the quadrotor mass.

The intent of these simplifications is to reduce the complexity of the system and to reduce the time it takes to simulate. However, the simplified models presented are not used in the remainder of this thesis. A reduced model, such as a state-space representation, is not necessary for further analysis in this thesis. The presented simulations still run quickly using the more nonlinear forms of the equations, and their rich dynamic behavior is preserved.

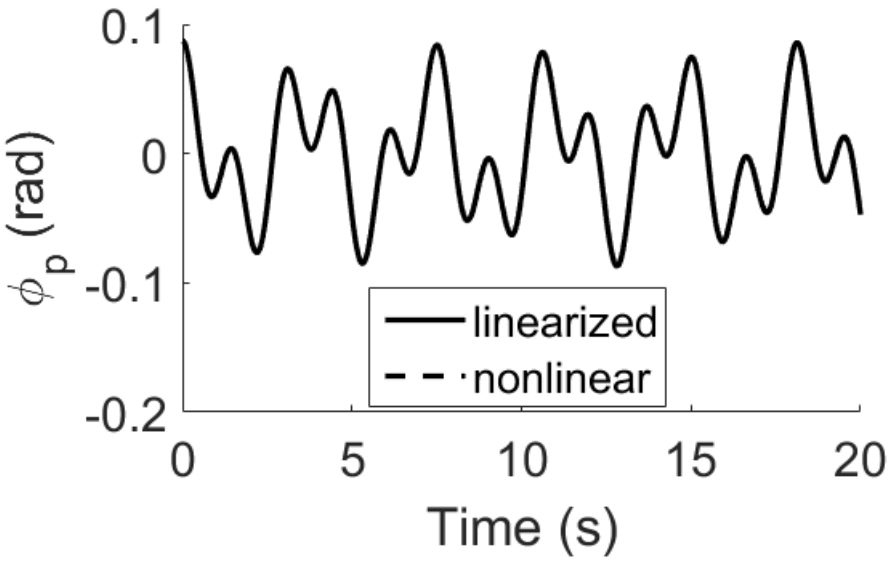

Figure 2.13: The linearized $\ddot{\phi}_p$ compared to the nonlinear form for the quadrotor as a fixed pendulum.

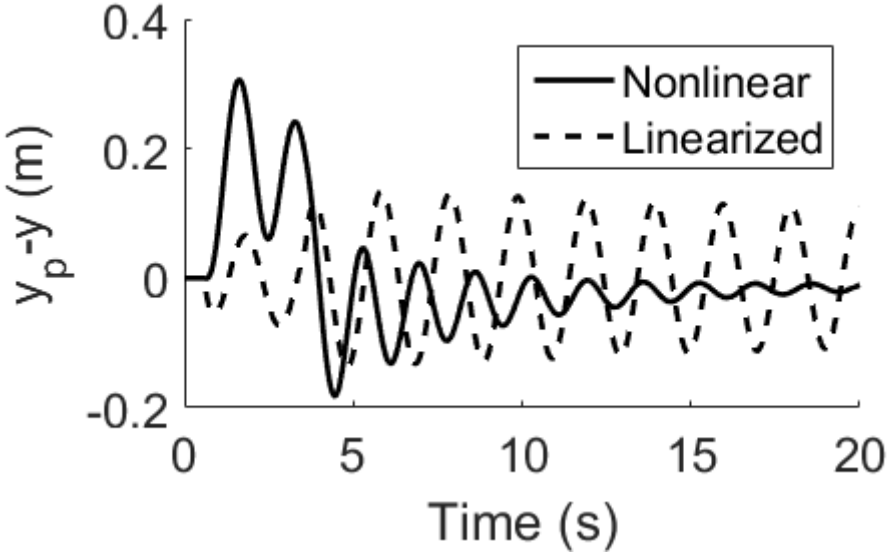

Figure 2.14: A comparison between the linearized and nonlinear payload accelerations for an example maneuver.

CHAPTER 3

FEEDBACK CONTROL OF QUADROTORS WITH SUSPENDED PAYLOADS

3.1 Feedback control

Effective feedback controllers are required for controlling the now-eight-DOF system: the six of the quadrotor and two of the payload. In this chapter, several PID controllers are investigated to determine their applicability and compatibility with input shaping. The focus of this thesis is on feedback control for the Pixracer microcontroller, which is described in Chapter 5.

3.1.1 Position control

Quadrotors using feedback control generally have two or three control loops: an inner attitude controller, an outer position or velocity controller, and an optionally separate altitude controller. The inner feedback loop is usually PD or PID control with a critically damped response and short rise time, providing a fast, but realistic, response. The outer feedback loop is most often a PID position controller following a desired path. In the case of long travel distances, PD control is preferred because integral error would accumulate and become irrelevantly large — proportional control alone would cause actuator saturation for a long distance, and integral control would have little effect. For PID position control, a path of "waypoints" is the input to the outer loop, and the inner loop adjusts the attitude to follow the profile. For PD control, the final position would normally be the input.

Position control is achieved in several built-in control modes of the Pixracer microcontroller, including pre-programmed "missions" (waypoint navigation), "return-to-home" mode, and "loiter-at-current-position" mode. However, these require the use of a GPS unit.

Position control simulations

The gains used in simulation for position control are listed in Table 3.1. These values were selected by first using the Ziegler-Nichols gain tuning method, where the gains are calculated using characteristics of the proportional-only response. Tuning methods such as Ziegler-Nichols are intended for simple systems, and so the resulting gains were further adjusted. The adjustments were made by changing individual gain values until a subjective specification was met, including the rise time, settling time, overshoot, and residual oscillation of the position response. The x and y directional properties of the quadrotor are assumed to be symmetric and, thus, share the same gains.

The selected gains were not designed to satisfy some type of "optimal" cost function, but there exist methods for how to find optimal parameters. One class of methods is the set of randomized optimization algorithms, where the algorithm maximizes a fitness function by randomizing the gain parameters and prioritizing certain aspects of the resulting system response. These algorithms include hill-climbing, simulated annealing, genetic algorithms, and others. They can be used in this context if one wanted to find some type of "optimal" gains. Relevant fitness functions can minimize the sum of squared position error, energy usage by minimizing thrust, residual oscillation, settling time, or others.

Table 3.1: The gains used for horizontal position and altitude control.

Type	P	I	D
PID (altitude)	30	1	6
PID (horizontal)	12	1	6
PD (horizontal)	12	0	6

The distance between PID waypoints affects the speed. The quadrotor slows down as it approaches the next waypoint due to the proportional error decreasing, so a finer resolution — more waypoints — would cause an overall slower response. The responses using several different waypoint resolutions are compared in Figure 3.1.

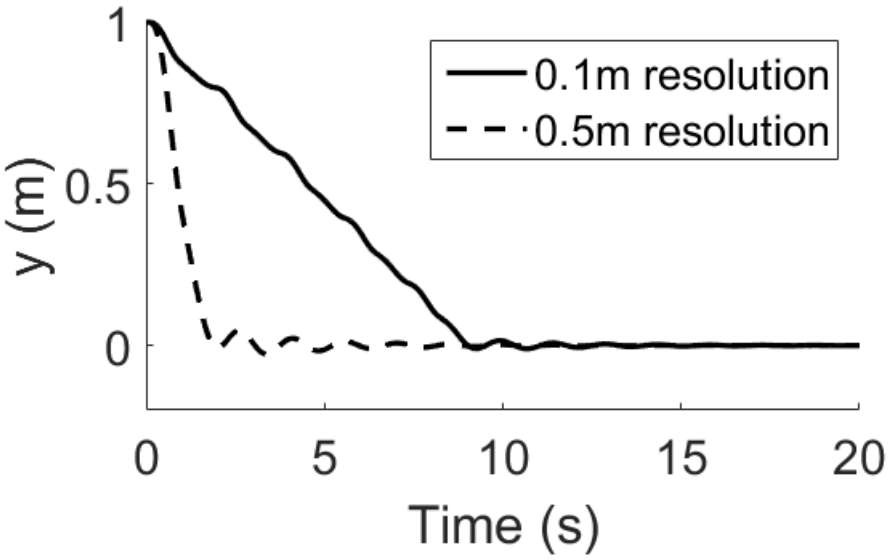

Figure 3.1: The effect of waypoint resolution on the PID position response of an unloaded quadrotor.

Because the quadrotor generally moves faster when there are fewer waypoints, it also overshoots its destination by a larger percent. The gains of the PD controller are the same values that the PID controller uses, excluding integral gain, so they are not optimized to the specific controller. However, increasing derivative gain or reducing proportional gain to reduce overshoot would cause the quadrotor to move slower. If constraints and specifications are imposed on the system, one could choose gains in a more rigorous way. However, this was deemed out of scope for the position controllers in the presented work.

With a large enough resolution, though, the PID case would become more similar to the PD case, eventually becoming the PD case when the resolution is exactly the distance from the starting point to ending point. Example responses of a PID (0.5 meter resolution) and a PD move are shown together in Figure 3.2. In this figure, the PD response quickly reaches the desired position but then overshoots by a large amount. The PID controller more slowly approaches the destination.

Note: for the simulations shown in Figures 3.1 and 3.2, the drag coefficients are no longer zero. For the rest of this chapter, they are $C_{\mathrm{d}} = 1.0$ and $C_{\mathrm{d,p}} = 0.5$, unless otherwise specified.

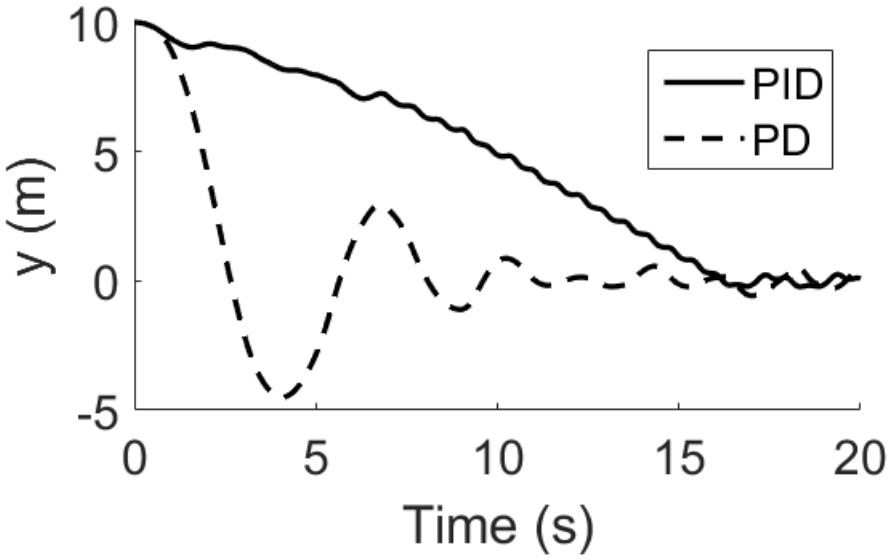

Figure 3.2: A PID response compared to a PD response.

Position control with payload

When the payload is considered, the waypoint resolution becomes a more important consideration. The higher the acceleration of the quadrotor, the more oscillation occurs. This is evident in Figure 3.3, which shows the payload swing of the PD controller used to generate Figure 3.2. The swing angle throughout the single pendulum response ($d = 0$ cm) is enough to make the model less valid, and the double pendulum response ($d = -5$ cm) has even higher amplitudes. This is in contrast to Figure 3.4, where the PID controller used with 0.5 m resolution — in which the quadrotor moves slower and accelerates less drastically than in the PD case — yields a response with a smaller peak-to-peak amplitude. The amplitude of the double pendulum response, however, is still large. Finally, a 0.25 m resolution payload response is shown in Figure 3.5, where the quadrotor is moving slowly enough that both the single and double pendulum responses are within the bounds of validity for the presented model.

A comparison of the velocities in the PD, PID (0.5), and PID (0.25) cases is presented in Figure 3.6. It is shown that by increasing the waypoint resolution from 0.25 m to the full path length, the speed increases. From these figures, it is clear that the PD gains should be

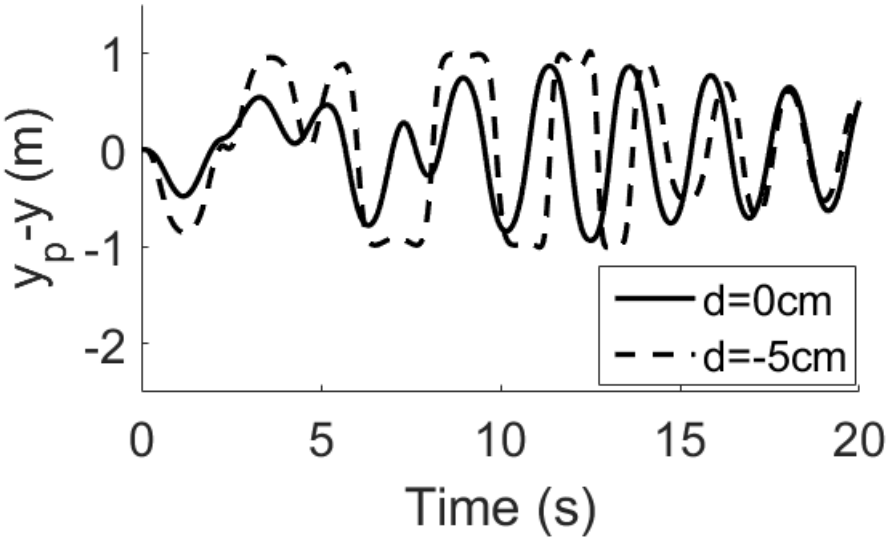

Figure 3.3: The payload swing of a PD path maneuver.

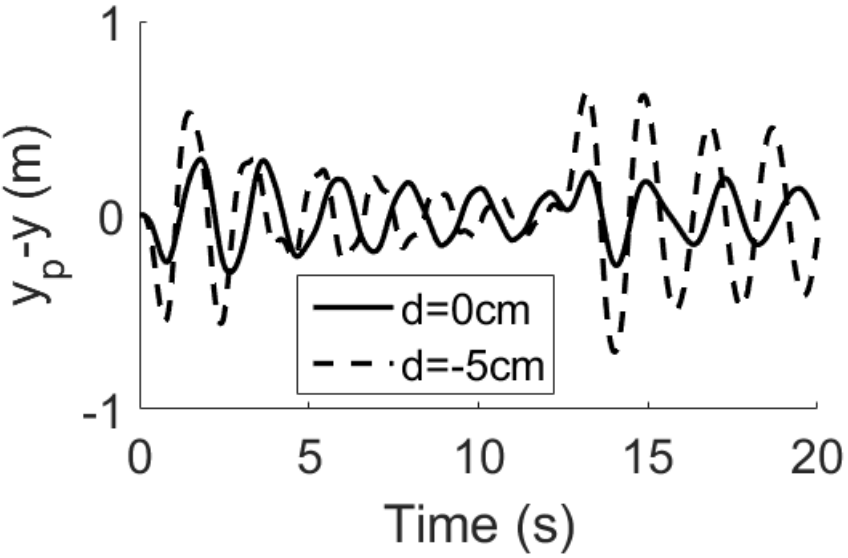

Figure 3.4: The payload swing of a PID path maneuver with 0.5 m waypoint resolution.

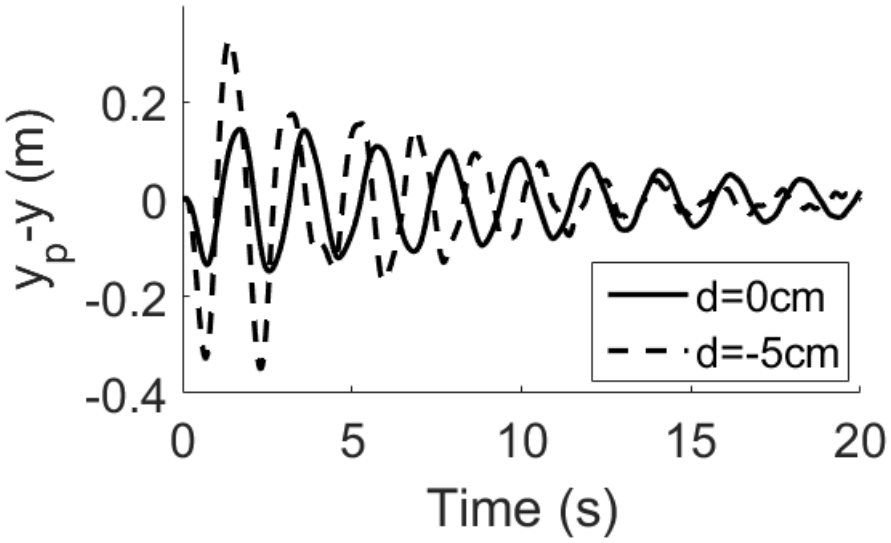

Figure 3.5: The payload swing of a PID path maneuver with 0.25 m waypoint resolution.

modified if payload swing is a concern. Alternatively, advanced control techniques can be implemented, which are discussed in Chapter 4.

3.1.2 Limitations to position control

Quadrotors may have an accelerometer, gyro, magnetometer, and temperature sensor (or an all-in-one inertial measurement unit, IMU), but these provide unreliable position information, which path-following requires. Using sensor fusion, the sensors can provide a fairly accurate acceleration estimation. However, integrating this twice to estimate position is often problematic, especially if the acceleration noise is large. Integrating acceleration to find displacement is useful, but subject to accumulating errors. GPS units are usually used for position and can reinforce the position calculated by integrating acceleration. However, the sampling rate and error — once per second to within a few meters based on the weather — cannot be used for reliable feedback. For imprecise control, such as moving toward a faraway target, PD control with GPS may suffice. However, for more precise control, especially that needed for advanced control techniques to be of any use, position control is not reliable enough.

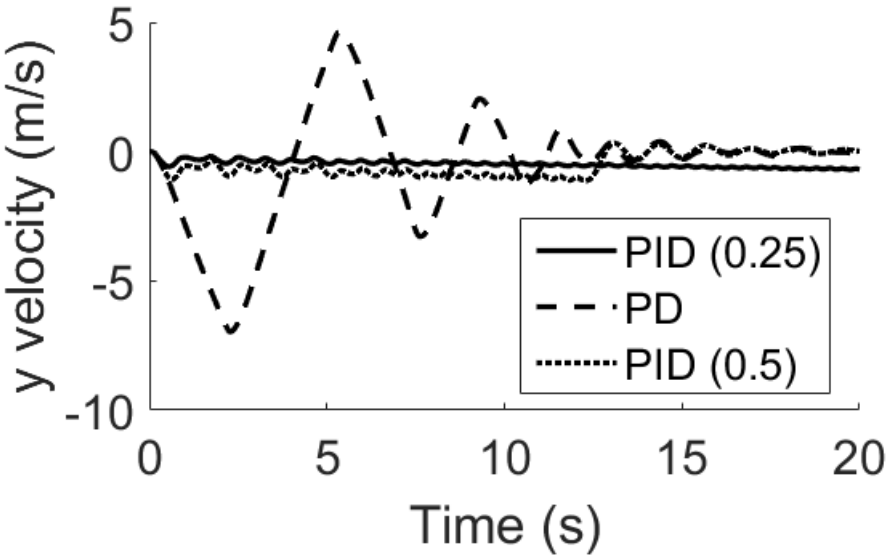

Figure 3.6: The velocities of the PD, PID with 0.5 m resolution, and PID with 0.25 m resolution controllers.

Previous studies have addressed this problem by performing most tests in a motion capture facility, with the capture cameras providing data to the feedback loop multiple times per second. However, this is not possible in real world uses, and so feedback must eventually be achieved with the hardware on board the quadrotor.

3.1.3 Velocity control

With unreliable position information, PID control to follow a path is unreliable. Instead, velocity control using a PI feedback loop can be implemented. A velocity profile is the input to the outer loop, and the inner loop attempts to achieve that profile. Although velocity is still an integrated estimation, it is a better estimation than position. Fixed-wing aircraft, such as planes, could make use of an airspeed sensor, such as a pitot-static probe, which uses Bernoulli's equation to find velocity. However, the Pixracer multirotor controller only has a barometer built-in for estimating altitude as a function of air pressure.

PI velocity control is one of the built-in control modes of the Pixracer controller ("position control" mode, where the user controls linear velocity), so implementing this controller is simply a matter of changing the gains in the code. There is a relatively small derivative

gain in the code, but its impact did not appear relevant in any simulation and was thus ignored. It would come into play if the Pixracer controller attempted a large linear acceleration.

With altimeters that measure air pressure, altitude does not need to be obtained by integration. The built-in PI velocity controller uses PID control on altitude rather than PI control on vertical velocity. Here, the altitude controller is a separate entity from the horizontal position controller.

Velocity control simulations

The PI gains for velocity control were obtained in the same manner as for position control and are listed in Table 3.2. When optimizing the position response, one must remember that P and I gains for velocity act as D and P gains for position, respectively. The x and y directional properties of the quadrotor are assumed to be symmetric and, thus, share the same gains.

Table 3.2: The gains used for horizontal velocity and altitude control.

Type	P	I	D
PID (altitude)	30	1	6
PI (x and y)	0.4	0.1	0

Although the velocity commands used in simulation are trapezoidal, the acceleration time used is 0.1 seconds in duration, which cannot easily be seen at larger timescales. Because the quadrotor can only directly control ξ and z, there is a delay between the commanded and actual velocity profiles, as illustrated in Figure 3.7. The corresponding position response is shown in Figure 3.8.

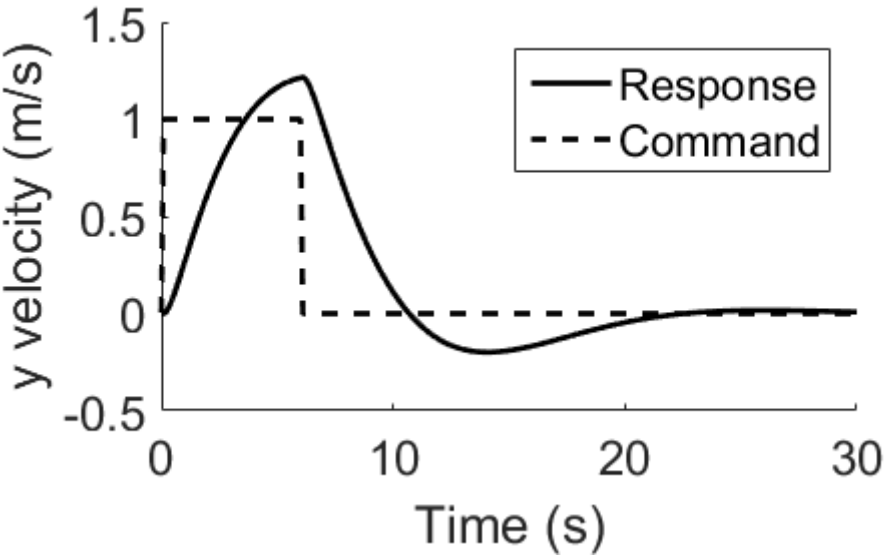

Figure 3.7: The velocity response of a trapezoidal velocity command on an unloaded quadrotor.

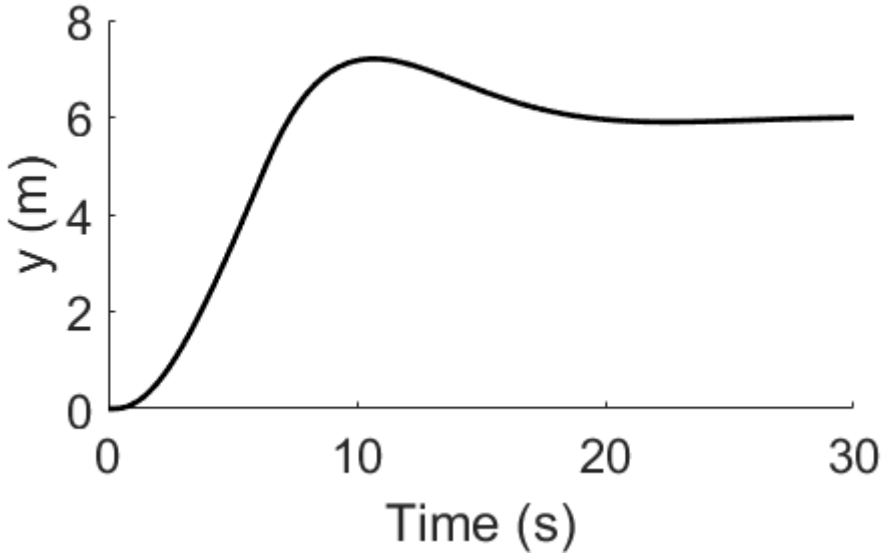

Figure 3.8: The position response of a PI velocity command on an unloaded quadrotor.

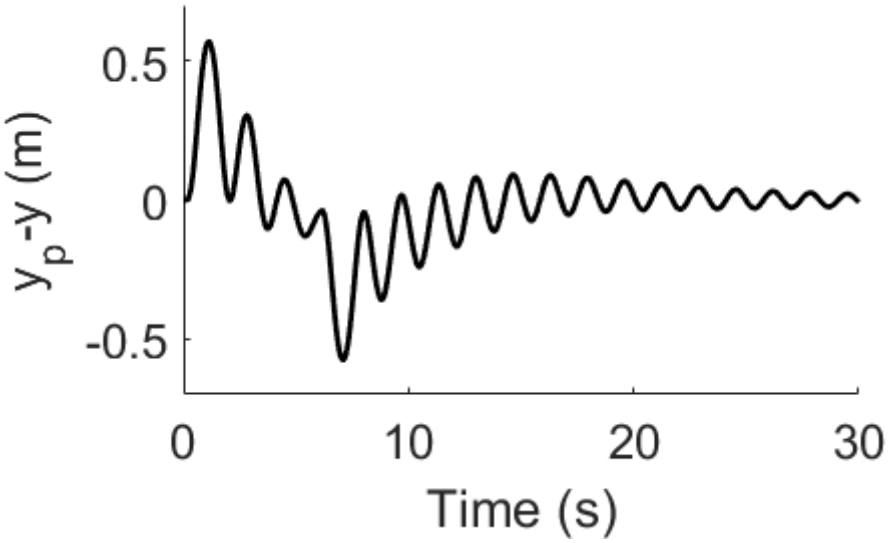

Figure 3.9: The swing response of a PI velocity command, with load-attitude coupling considered.

Velocity control with payload

The behavior of the payload swing is similar to that under position control, i.e. a faster velocity, more dramatic acceleration, and load-attitude coupling ($d < 0$ cm) both cause a greater swing amplitude. An example response is shown in Figure 3.9. The largest amplitude of the swing occurs during the transient deceleration, which occurs at $t = 7.3$ s. The relationship between this amplitude and the desired velocity is shown in Table 3.3.

Table 3.3: The desired velocity and maximum swing amplitude of PI control with load-attitude coupling.

$\dot{r}_{\text{des}}$ (m/s)	Max ϕ_{p} amplitude (rad)
1.0	0.1488
2.0	0.2772
3.0	0.3898
4.0	0.4886
5.0	0.5751
6.0	0.6522

3.1.4 Limitations to velocity control

For automated velocity control, a velocity profile must be created ahead of time. This applies to automated machinery where the pickup and delivery locations are known, and it could work under similar conditions for quadrotors. However, for applications where the environment or objective locations are not already known — or where manual control is desired — velocity profile tracking is not desirable to use.

Manual velocity control is the alternative, but it does not automatically control position. In this regard, the human operator who is manually controlling velocity acts as the position controller. However, for the topic of advanced control, a velocity profile must be generated at some point, as it is the specific velocity profile that, when followed, creates a desired response. Thus, velocity profile tracking must be accurate in order to use advanced control techniques.

The argument can be made that the integration of a noisy signal results in an unreliable measurement. The Pixracer microcontroller passes certain estimations through a low-pass filter to attenuate noise, but the reliability of these measurements was not investigated. Additionally, other small errors, including those associated with the sampling rate, would accumulate when integrated. Over time, this may lead to an estimation that is noticeably incorrect.

3.1.5 Manual control

A human performing manual control is a mode of operation common in hobbyists. The analog to human control is perhaps PD position control with a tolerance, where neither position error, nor steady-state velocity, are required to tend toward zero. There is an element of being "close enough" in the spatial states. If precise positioning is a requirement in a certain application, then the human operator can adapt to the circumstances.

When a payload is attached, r error, $\dot{r}$, ξ_{p}, and $\dot{\xi}_{\mathrm{p}}$ should be minimized in the final state. Advanced control techniques, such as linear quadratic regulation (LQR) and H_{∞}

control, can be used when multiple disparate states should be fed back into the controller. However, these require the software to estimate the location and the effects of the payload. Alternatively, a skilled human should be able to judge the parameters and drive them to be within a desired tolerance.

Without a motion capture facility, the position of the payload can be estimated using an observer. However, observer design is not considered in this thesis. There comes a point when the microcontroller is computationally overburdened, and this hardware limitation would have to also be considered in the observer design. From qualitative observation, though, the Pixracer microcontroller could likely implement one additional observer.

3.1.6 Attitude control

The attitude estimations that are produced by sensor fusion in IMUs are more reliable than the integrated acceleration estimations due to there being less noise that needs to be filtered away. Therefore, attitude control is considered. PD attitude control is a built-in control mode of the Pixracer controller found in "angle" and "altitude control" modes.

Attitude control simulations

The attitude PD gains were chosen by attempting to achieve a fast, but overdamped, attitude response. A critically damped response may be theoretically ideal, but the real quadrotor may not be able to respond that quickly. The center of pressure, r_{cop} was temporarily set to zero so that the equilibrium angle would not differ from the desired angle. The time required to achieve a certain attitude was estimated from various real-world sources, and the gains were selected to achieve such settling times. The selected gains are listed in Table 3.4, with the ϕ and θ directional properties sharing gain values due to symmetry. ψ gains were not tested and are not included.

Because attitude is a more reliable estimation than velocity, attitude profile tracking and attitude holding are both considered. An example trapezoidal attitude command and the

Table 3.4: The gains used for attitude control.

Type	P	I	D
PD (ϕ, θ)	0.90	0	0.28

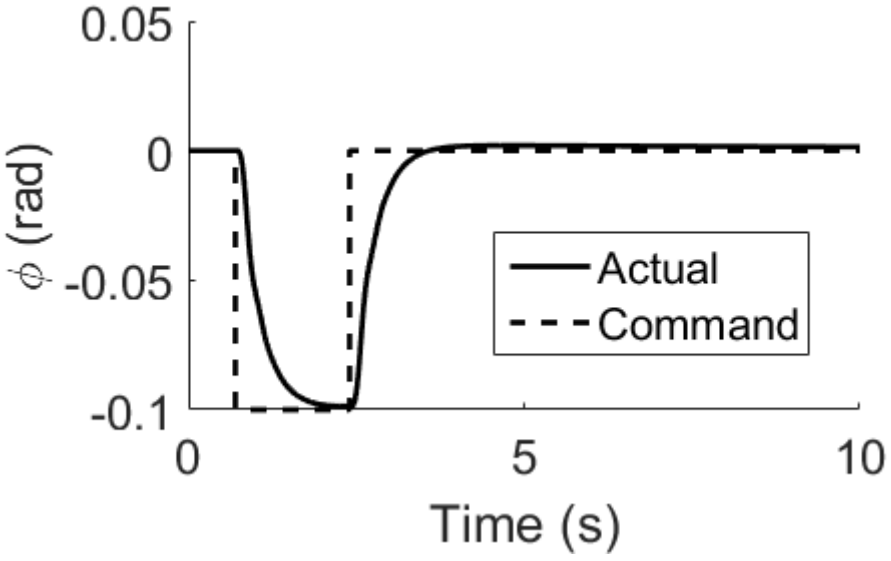

Figure 3.10: An attitude command and the resulting unloaded quadrotor response.

corresponding response are shown in Figure 3.10. The attitude response tracks the command well.

It was shown in Figure 2.6 that the center of pressure, r_{cop}, affects the attitude response in terms of a torque induced by the drag force. For $\phi_{\text{des}} = \frac{\pi}{6}$, the effects of z_{cop} is shown in Figure 3.11 for the ϕ response. From zero to five seconds, the equilibrium ϕ of the nonzero z_{cop} is less than that of when $z_{\text{cop}} = 0$ (although the former does not actually reach equilibrium in the presented amount of time). However, because the direction of the drag force is dependent on velocity, the quadrotor overshoots zero at $t = 6$ s for the nonzero z_{cop} case. The same damping torque causes overshoot as the quadrotor attempts to return to zero attitude.

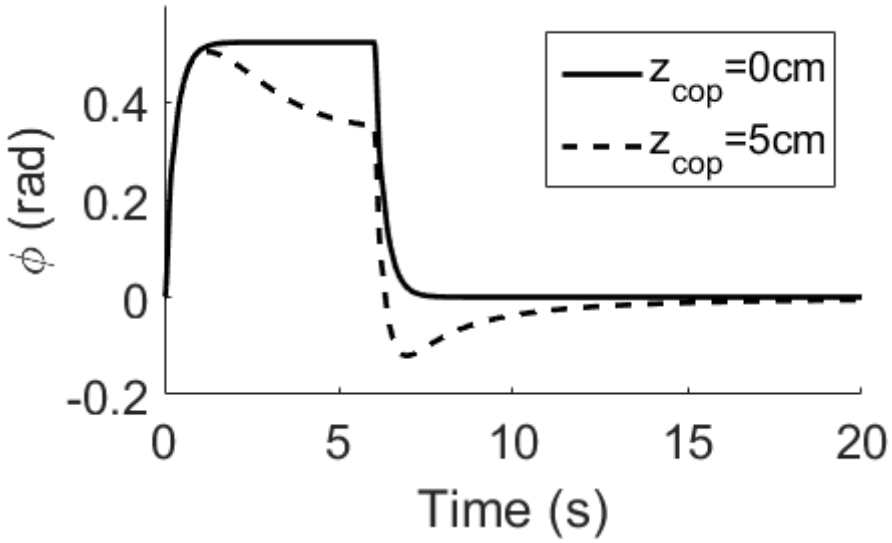

Figure 3.11: The effect of r_{cop} on ϕ of an attitude command.

Attitude control with payload

The amplitude of swing increases with increasing ϕ_{des}. However, in the double pendulum case, the payload affects the attitude of the quadrotor, in addition to its position. The attitude response and payload swing are shown in Figures 3.12 and 3.13, respectively. As before, the swinging amplitude of the double pendulum case is greater than that of the single pendulum case. In the attitude response, there is an additional slight oscillation in the case exhibiting load-attitude coupling.

3.1.7 Limitations to attitude control

The limitations of attitude control are that the position and velocity of the quadrotor are not addressed by the controller. For practical use-cases, position and velocity are among the most desirable parameters to control. For this reason, attitude control is not terribly useful alone, although it holds promise as a basis for advanced control techniques. It would then be ideal if a position or velocity controller generated profiles for the inner attitude controller to follow. These profiles could be modified by control techniques to produce a response with reduced oscillations.

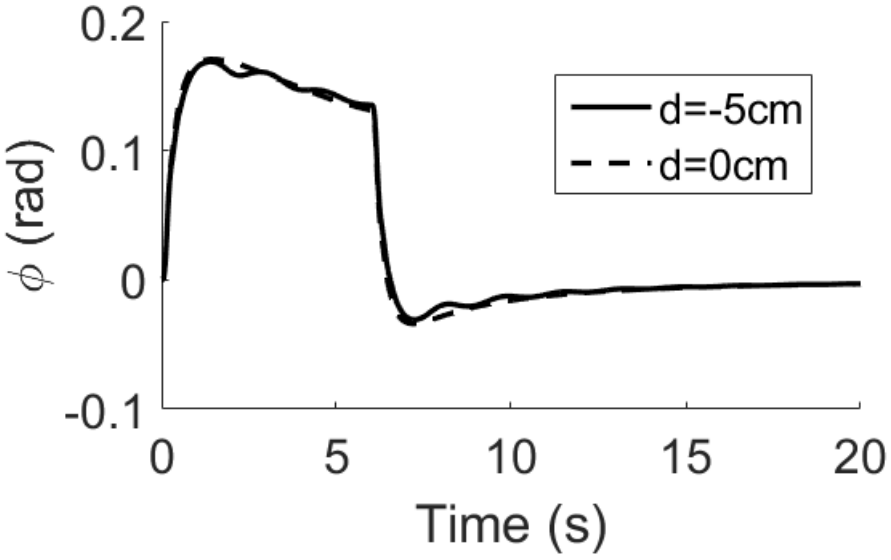

Figure 3.12: The effect of d on the ϕ response of an attitude command.

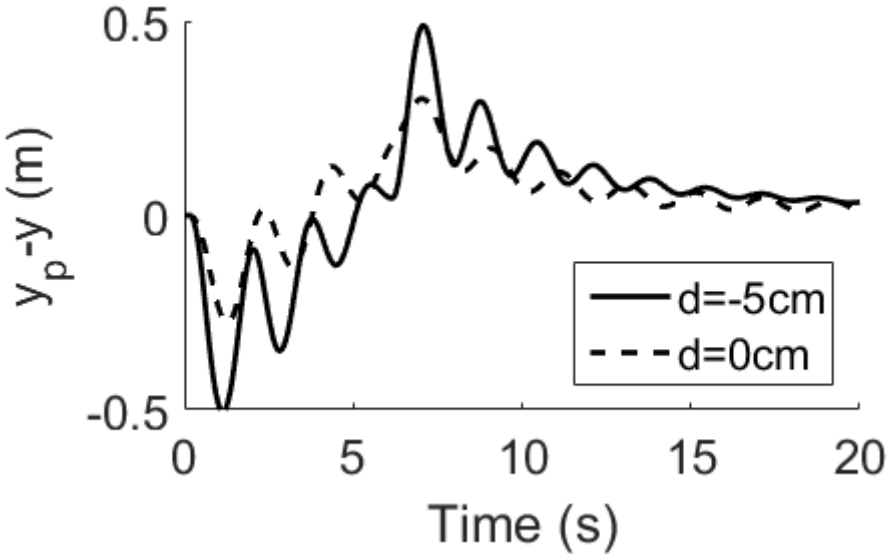

Figure 3.13: The effect of d on the swing response of an attitude command.

Although gyroscopes measure attitude rate, the attitude estimations do not solely come from integration. Instead, using sensor fusion, the outputs of the gyroscope, accelerometer, and magnetometer are passed through an extended Kalman filter in order to estimate attitude. The resulting attitude is relative to the user-defined zero-attitude orientation, which itself is relative to the direction of gravity.

3.2 Outer-loop control with attitude tracking

Chapter 2 described the desired attitude, ξ_{des}, as a value recomputed during each loop of the simulation based on $\ddot{r}_{\text{des}}$, if an outer-loop controller is also being used. As mentioned previously, in order to implement advanced control techniques to modify an attitude profile, the outer-loop controller must generate those attitude profiles instead of instantaneous values.

In simulation, however, it is difficult to generate and use a profile during *ode45* integration because the solver can go backwards in time. ODE solvers do this when the residual or absolute error of an integration step is not below a user-defined maximum, meaning that too large of a Δt was integrated. Adding to, or removing from, a global variable technically violates causality if the solver opts to move to a previous time in the next iteration. Moreover, it is impossible to perform real-time manual control during the *ode45* integration routine.

3.2.1 Manual control simulation algorithm

Instead of running the solver once, integrating from $t = 0$ to $t = t_{\text{final}}$, the dynamics are integrated from $t = t_i$ to $t = t_i + 0.1$ until the user terminates the program. There is a delay after each call to *ode45*, during which the user can input commands via the keyboard. The delay also serves to progress the simulation in real time, where one second of simulation occurs over one second of real time, approximately. Attitude profiles are generated during the delay based on keyboard input, and this buffer is accessed during subsequent integration steps. This buffer is passed into a modifying routine, changing it into a trapezoidal or other

command. Keyboard input does limit the amplitude of input commands to a single positive number, a single negative number, and zero as opposed to the range of values provided by the joystick of a physical RC controller. A wider range of values could be implemented by utilizing more keys on the keyboard, at the cost of usability. An RC controller or even a video game controller could provide input to MATLAB using its *joystick* objects, but this was not performed for this research.

The manual control simulation algorithm proceeds as follows. The model parameters and initial conditions are given values. Pitching and rolling are mapped to the arrow keys on a keyboard in order to add human interactivity. In an infinite loop, the inputs are checked, a command buffer is generated from those inputs or lack of inputs, the *ode45* simulation is advanced by a constant time step, states of interest are plotted versus time, the final state becomes the initial conditions of the next iteration, and a delay is inserted to update the plots and also approximate a real-time simulation. Any states can be plotted, but the most relevant ones to this research are position, velocity, and payload swing. Finally, the commands input by the user can be stored and replayed exactly under different initial conditions or model parameters. The commands can also be edited or generated separately and played in simulation. The algorithm is contained within *modelCont.m*, presented in Appendix A.

The user interface is shown in Figure 3.14, with position and velocity being updated in quasi-real-time. The command that causes the aforementioned position and velocity responses is shown in Figure 3.15, where a positive command corresponds to pitching forward, a negative command corresponds to pitching backward, and a zero command meaning there is currently no input. Only y- and ϕ-related variables are shown, but both x and y can be controlled at the same time by controlling θ and ϕ, respectively.

The advantages of this approach are that manual control is an option, a real-time visual representation of the quadrotor states is available, and the attitude profile buffer can be implemented on the Pixracer microcontroller in a custom control mode. These reasons make this simulation environment ideal for performing user feedback studies without the

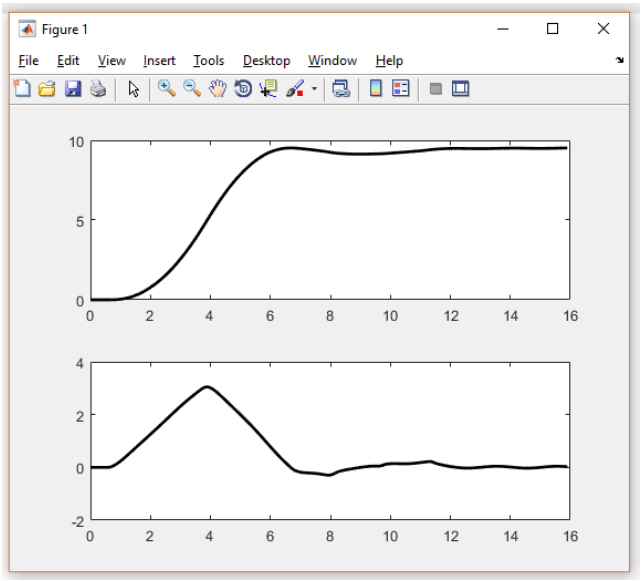

Figure 3.14: The user interface of the simulation algorithm, showing example position (above) and velocity (below) in real-time.

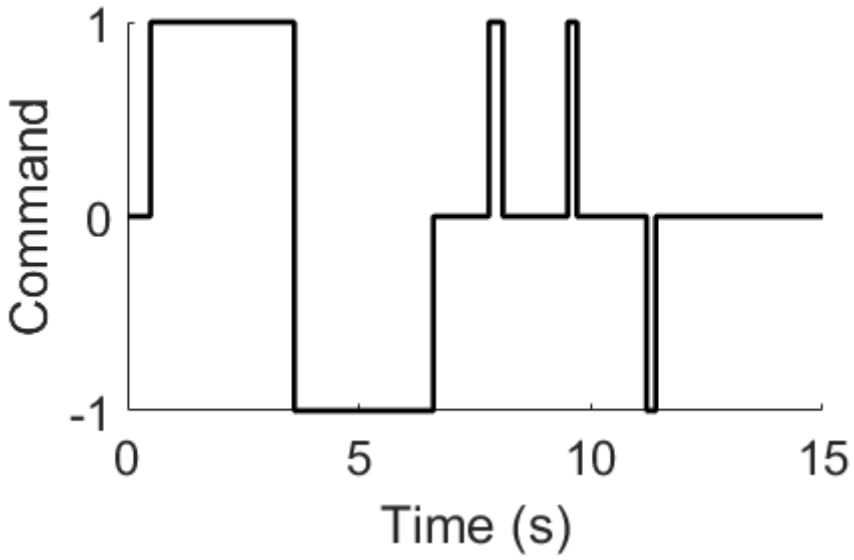

Figure 3.15: Example commands that the user inputs in real-time.

fear of a participant crashing and damaging a real quadrotor.

A human trial can be set up by plotting only position and velocity and having the tester try to achieve and maintain a desired position. After conditions are met and the simulation ends, the time response of the payload swing can be plotted. The payload angle could be plotted during the simulation, and experienced testers could time their inputs with the peaks and troughs of the time response to optimally control the system. However, in real life with a human operator, the period of the payload is relatively short, and the payload angle is not known with much precision. Controlling the simulation based on the precise payload angle may not be representative of a true experience for testers.

Because the attitude controller is assumed not to possess knowledge of the payload position or angles, any further control techniques should also possess no knowledge of the payload position or angles. One such technique is input shaping, which is the subject of the following chapter.

CHAPTER 4
INPUT SHAPING CONTROL OF PAYLOAD SWING

Input shaping is a technique where the command input to a system is modified with the purpose of suppressing one or multiple modes of vibration. The original input is convolved with a specific series of impulses that are timed to cancel out the vibrations of each other.

4.1 Introduction to input shaping

Consider an undamped mass-spring system, which has a sinusoidal response to an applied impulse. Next, consider a second impulse applied to the system after the first input, which has a similar, but temporally offset, response. The two responses added together yield the full response of the system. If the second impulse is placed exactly at half the period of oscillation, as illustrated in Figure 4.1, then the superposition of the responses equals zero for every time after the half-period, as shown in Figure 4.2. The first half-period of the response is unaffected, and there is no residual oscillation afterward.

This two-impulse sequence is the most basic type of input shaper, a Zero Vibration (ZV) shaper. An impulse, step, or any generic input is divided into two parts, with the second part timed to cancel the vibration induced by the first part. Let P be the period of the oscillation to be suppressed. The timing and scaling convolution that operates on the input is shown in Table 4.1.

Table 4.1: The convolution components of a ZV shaper.

	First	Second
Amplitude scaling	0.5	0.5
Time offset (s)	0	$P/2$

When the damping ratio is greater than zero, the amplitude scaling generalizes to [23],

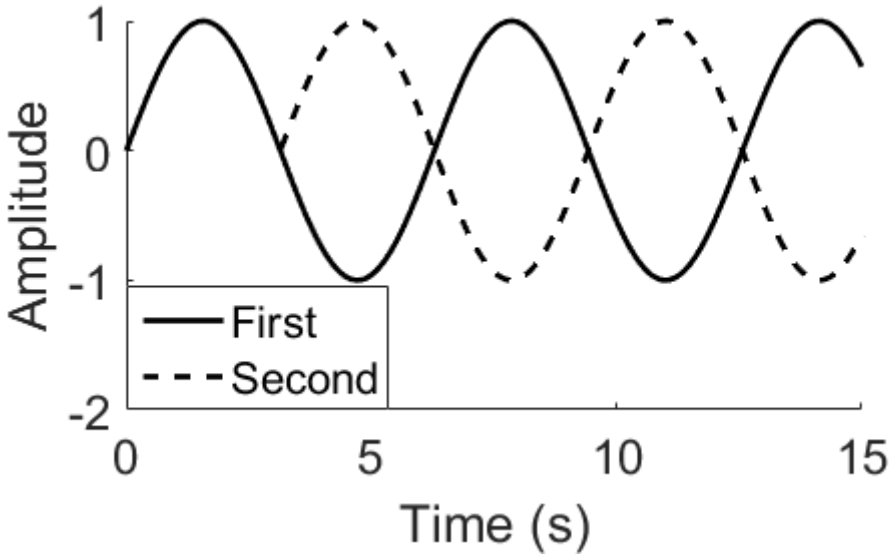

Figure 4.1: Two system responses offset by half the period.

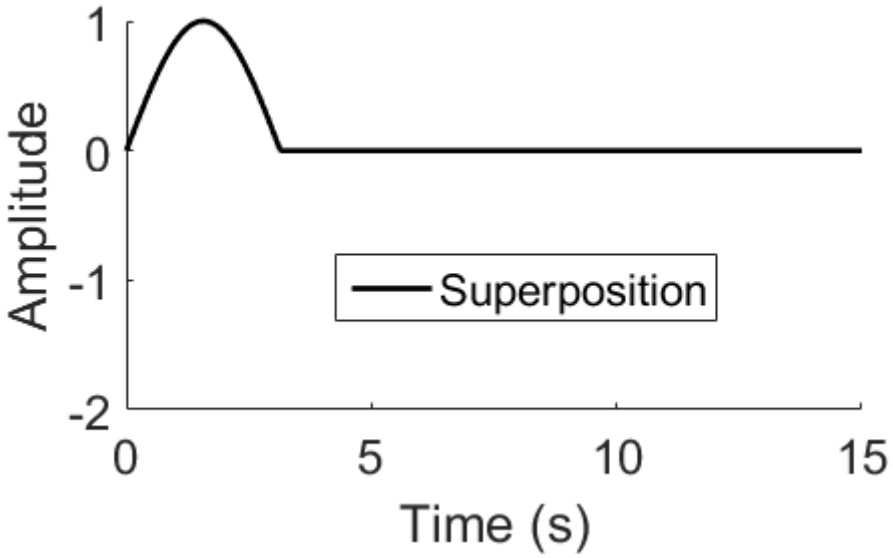

Figure 4.2: The superposition of sine waves, yielding a response with zero residual oscillation.

[24]:

$$A_1 = \frac{1}{1+K} \tag{4.1}$$

$$A_2 = \frac{K}{1+K} \tag{4.2}$$

where,

$$K = \exp(-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}) \tag{4.3}$$

where A_1 is the first amplitude scaling, A_2 is the second, and ζ is the damping ratio. Thus, the two pieces of information an input shaper requires are the period of oscillation and the damping ratio. These parameters are more reasonable to acquire than instantaneous payload angle at any time for the quadrotor, which may be required for other advanced control algorithms.

4.2 Input shaping applications

The input being shaped does not have to be position; velocity could be shaped, where the integral of the first half-period yields displacement, and the system becomes stationary afterwards. A real-world use-case for this is in cranes carrying payloads, where the trolley follows a velocity command and residual payload swing is undesirable. Input shaping could modify the velocity command so that the trolley still moves the desired distance, but the payload beneath swings into place with ideally zero residual oscillation.

For example, the trapezoidal velocity profile input to the PI velocity controller mentioned previously can be ZV shaped. The period used to design the shaper is the approximation for a single pendulum, $P = 2\pi\sqrt{\frac{L_\mathrm{p}}{g}}$, and the damping ratio, ζ, is assumed to be zero. The original commanded profile and the new ZV-shaped profile are compared in Figure 4.3. Note that the shaped command ends $P/2$ seconds later than the original command.

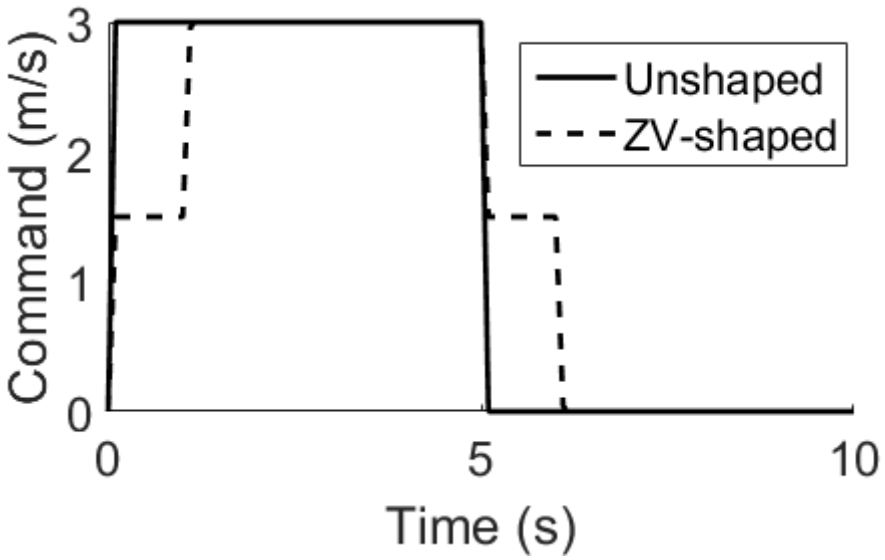

Figure 4.3: A comparison of a command pulse and its ZV-shaped counterpart.

This slight delay in the shaped case can also be seen in the position response, Figure 4.4.

The significant benefits of input shaping are shown in Figure 4.5, which shows that the transient swinging is reduced and the residual oscillation is greatly reduced. From the maximum transient peak to the minimum, the peak-to-peak amplitude is reduced from 0.4468 m to 0.2218 m, a 50 percent reduction. For the residual, the peak-to-peak amplitude near the beginning is reduced from 0.2157 m to 0.005949 m, a 97 percent reduction. Using the logarithmic decrement method, the damping ratio, ζ is found to be 0.0048, which can safely be assumed to be zero.

The response for $d = -5$ cm is shown in Figure 4.6. The transient peak-to-peak is reduced from 0.6764 m to 0.4190 m, a 38 percent reduction; the residual is reduced from 0.08586 m to 0.01865 m, a 78 percent reduction. However, these results are only for a move time of five seconds; different move times have different reductions in oscillation. Certain move times cause near-zero oscillation, even without input shaping. This relationship is presented later.

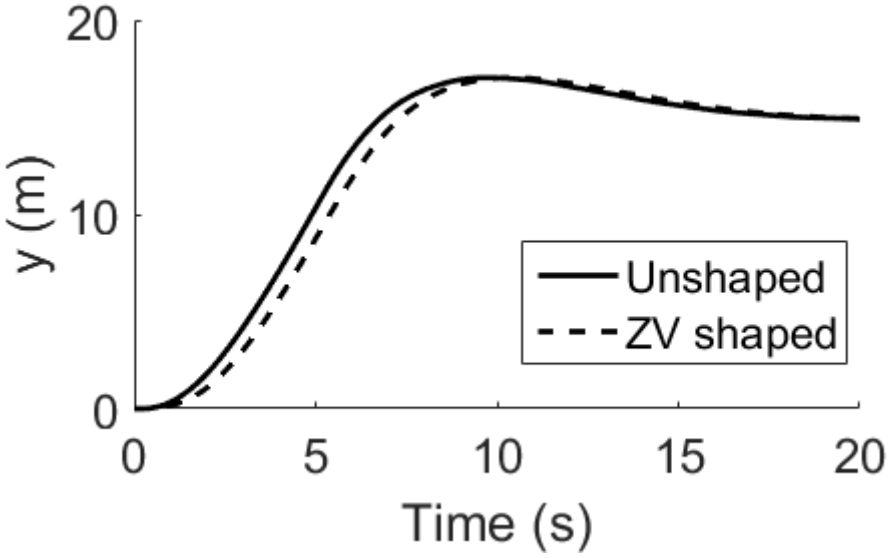

Figure 4.4: The position response resulting from a velocity command and its ZV-shaped counterpart.

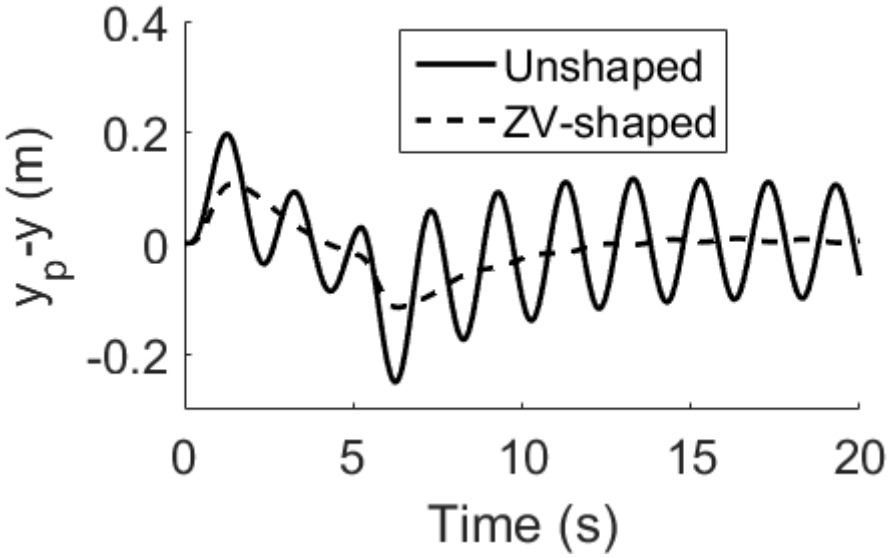

Figure 4.5: The payload swing resulting from a velocity command and its ZV-shaped counterpart, with no load-attitude coupling.

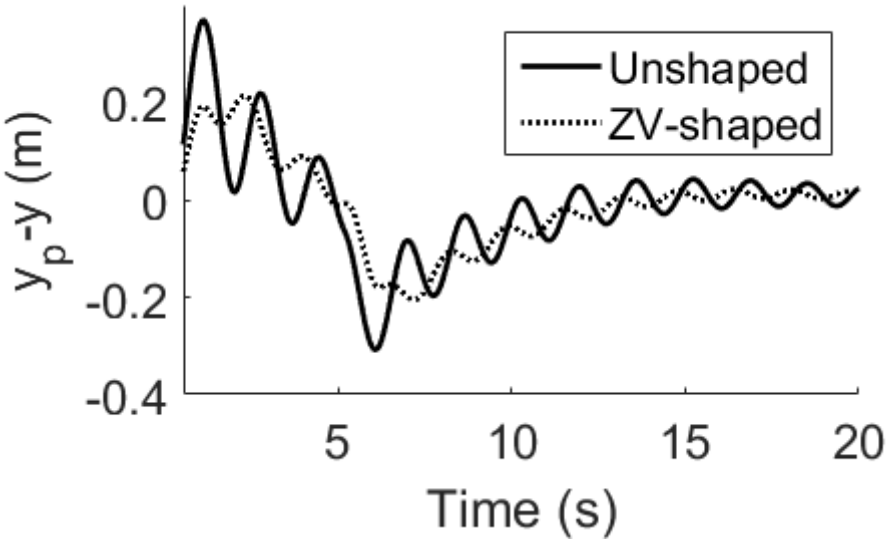

Figure 4.6: The payload swing resulting from a velocity command and its ZV-shaped counterpart, with load-attitude coupling considered.

4.3 Robust input shaping

When $d \neq 0$, the system has more than one frequency to be attenuated: one for the payload and another for the suspension point. More advanced input shapers can reduce the effects of multiple frequencies. One such is a Zero Vibration and Derivative (ZVD) shaper. The ZVD shaper is less sensitive to modeling errors around the associated frequency. Assume P_1 and P_2 are the periods of the two oscillatory modes, with $P_1 \leq P_2$. The amplitudes and times of a ZVD shaper designed with zero damping are listed in Table 4.2. If $P_1 = P_2$, the second and third impulses can be added together.

Table 4.2: The convolution components of a ZVD shaper.

	First	Second	Third	Fourth
Amplitude scaling	0.25	0.25	0.25	0.25
Time offset (s)	0	$P_1/2$	$P_2/2$	$P_1/2 + P_2/2$

There is a measure of robustness for each input shaping method: some shapers are more or less sensitive to modeling errors, including inaccurate periods and damping ratios. For

a crane, the hoist length can change, which affects the period of oscillation. Moreover, the payload mass may not be constant, affecting the damping ratio. One ideal scenario is that the selected input shaper attenuates the oscillations of all targeted frequencies. A consideration here is that each targeted frequency slows the response down, and targeting several frequencies may cause the system to respond undesirably slowly. Another is that the shaper is a function of the parameters, meaning the amplitudes and timings of the impulse sequence are functions of the system parameters. However, the amplitudes and times may need to be obtained numerically, and the physical controller may not be capable of obtaining those numbers in real-time, given a rapidly changing system.

There can be many modes of oscillation in a system — technically, there are infinite modes — but high-amplitude or low-frequency modes should be prioritized in attenuating as they have the most impact on the amplitude of residual oscillations. Another option is to design a shaper for minimizing oscillation rather than intending to reduce it to zero, as a ZV shaper does. If a tolerance is added to the desired vibration amplitude, then the resulting shaper becomes more robust. However, this effect is not evaluated in this thesis.

4.4 Disadvantages of input shaping

The downsides to using input shaping are twofold. Because the input command is modified, the corresponding system response may be unexpected to a human controller. Input shaping delays a portion of the subsequent commands by at least half the period. If the oscillatory mode has a large period, then the system would take a long time to achieve the desired maneuver. Inexperienced human controllers may overcompensate if the actual system response does not match their expectations, and overall performance may be degraded.

Secondly, input shaping cannot reduce pre-existing oscillations or reject disturbances, as there is no active component of input shaping that monitors the response. However, since the goal of this research is to reduce payload oscillations without knowledge of the location of the payload, this disadvantage is not applicable.

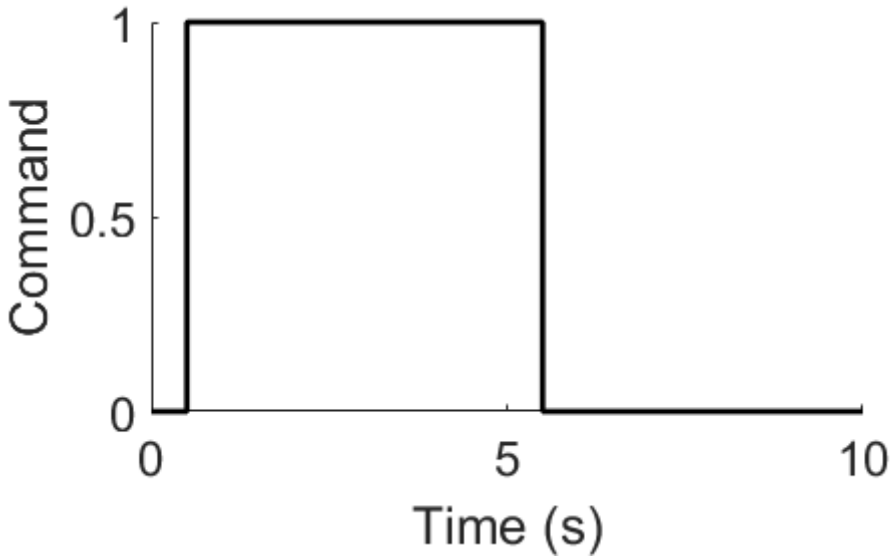

Figure 4.7: A single pulse of an attitude command with a five-second duration.

4.5 Oscillation analysis

As stated previously, the amplitude of payload oscillation depends on the move time of the maneuver. Some move times cause a response with a high amplitude, and others cause a near-zero-amplitude response. In order to show this effect, first a single pulse of a trapezoidal attitude command is considered. An example pulse is shown in Figure 4.7. The pulse amplitude is $\phi_{\text{des}} = 0.1$ rad. The duration of that pulse — the move time — is varied, and the residual amplitude of the payload response is recorded, as shown in Figure 4.8. Both drag coefficients are treated as zero. These results show that the residual peak-to-peak swing is periodic with move time, ranging from near-zero to a maximum of 0.3 m.

For a 1-meter payload length, L_{p}, the period of the payload oscillation, P, is approximately 2 seconds. Both the maxima and minima of Figure 4.8 are periodic with respect to that period. This is due to the fact that the payload motion is also periodic: the payload as an ideal pendulum has the same position and velocity at time t as it has at time $t + P$. It follows that the residual responses of those two pulse durations would also be the same. This analysis was repeated for a different payload length, $L_{\text{p}} = 0.5$ m, and the results are also periodic with respect to the new period, about 1.4 seconds, as shown in Figure 4.9.

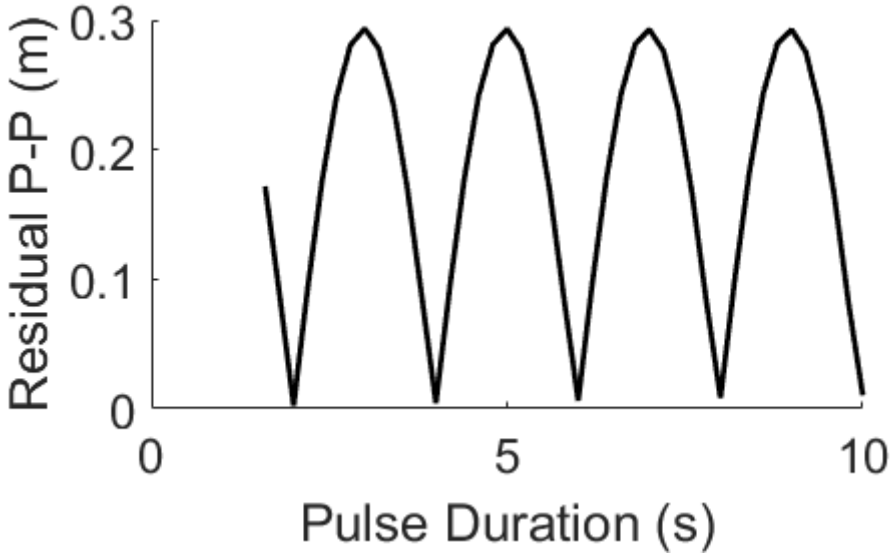

Figure 4.8: The peak-to-peak residual oscillation induced by an attitude command versus the duration of that command for a 1-meter payload length.

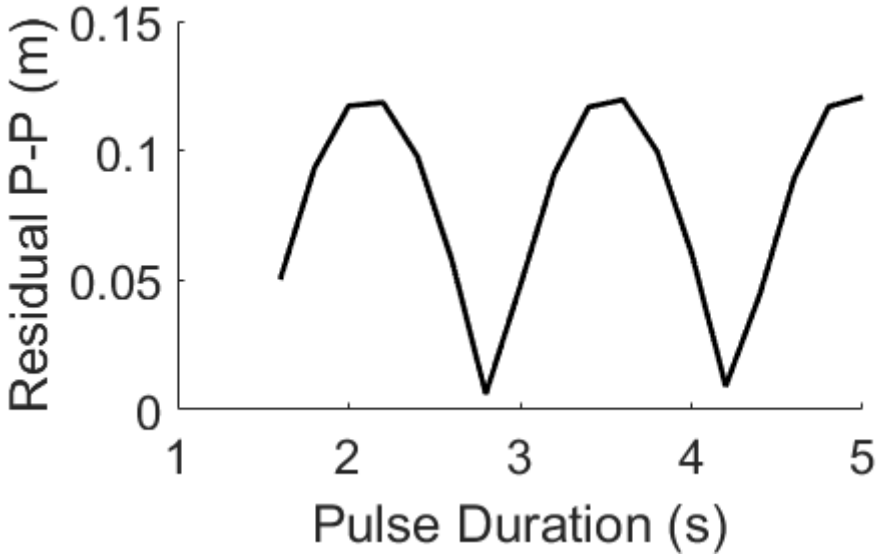

Figure 4.9: The peak-to-peak residual oscillation induced by an attitude command versus the duration of that command for a 0.5-meter payload length.

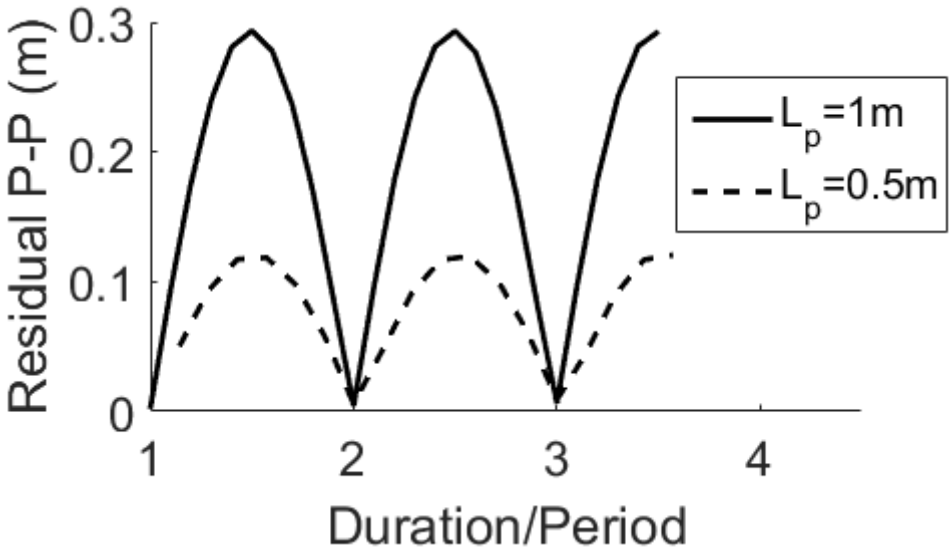

Figure 4.10: The normalized peak-to-peak residual oscillations.

If both responses are normalized with respect to their respective periods, the peaks and troughs line up with each other, as shown in Figure 4.10. Due to the nonlinearities present in the system, linear amplitude scaling does not occur.

When input shaping is applied to the system, there is a nearly 100 percent reduction in residual oscillation and significant percent reduction in the transient peak-to-peak amplitude. One such comparison is shown in Figure 4.11, where the reduction in peak-to-peak residual amplitude is 99 percent, and the reduction in transient peak-to-peak amplitude is 67 percent. In the transient regime, there is also little swing when input-shaped. The transient peak-to-peak amplitude gets reduced by 96 percent within the duration of the move. By simply delaying half of the command with input shaping, the payload hardly deviates from the mean within the transient and within the residual regimes. This is shown in Figure 4.12, which is the comparison between the trapezoidal residual amplitudes and the ZV-shaped trapezoidal residual amplitudes. The amplitudes of the ZV-shaped input do not exceed 1 mm in simulation.

For completeness, a response with both drag coefficients considered is shown in Figure 4.13. However, because the residual response is not centered at a constant number, the

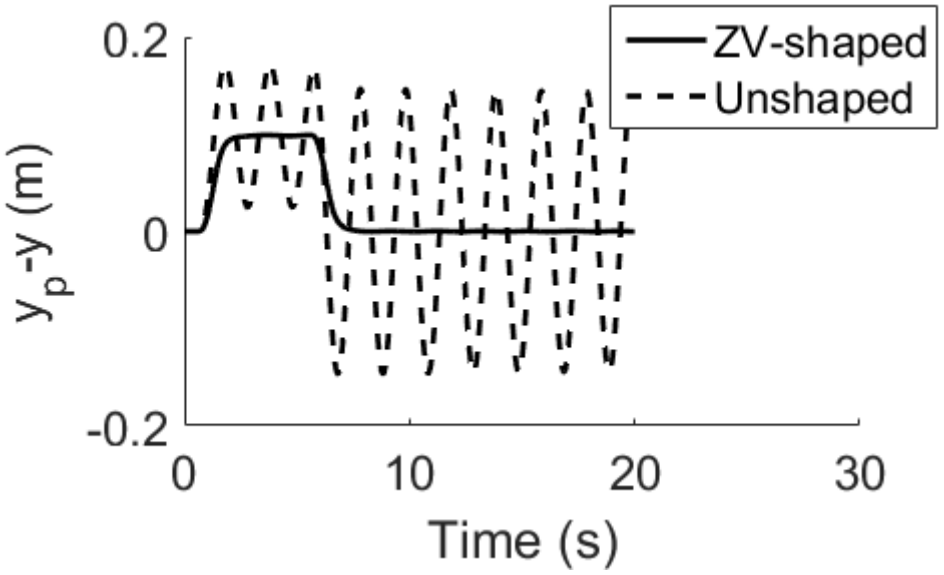

Figure 4.11: The payload responses of a trapezoidal and ZV-shaped trapezoidal input for a five-second command.

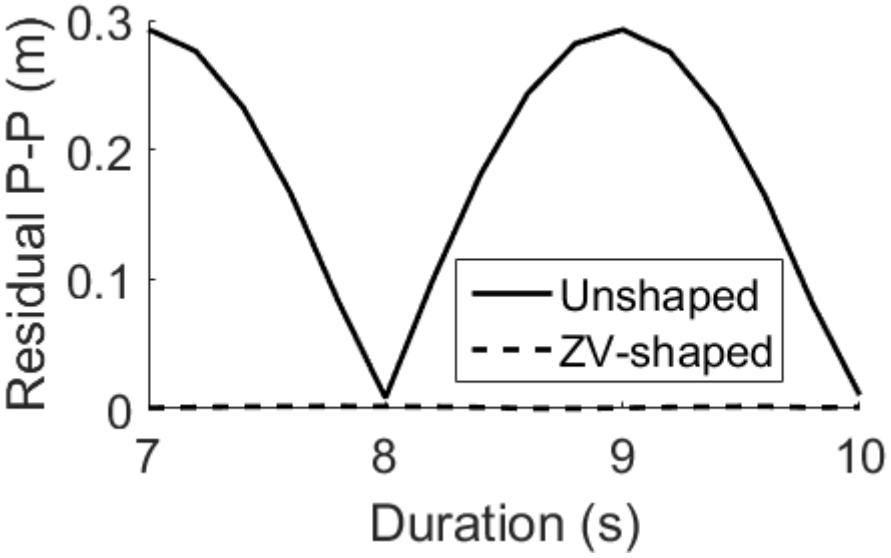

Figure 4.12: A comparison between the residual oscillations of a trapezoidal input and a ZV-shaped input.

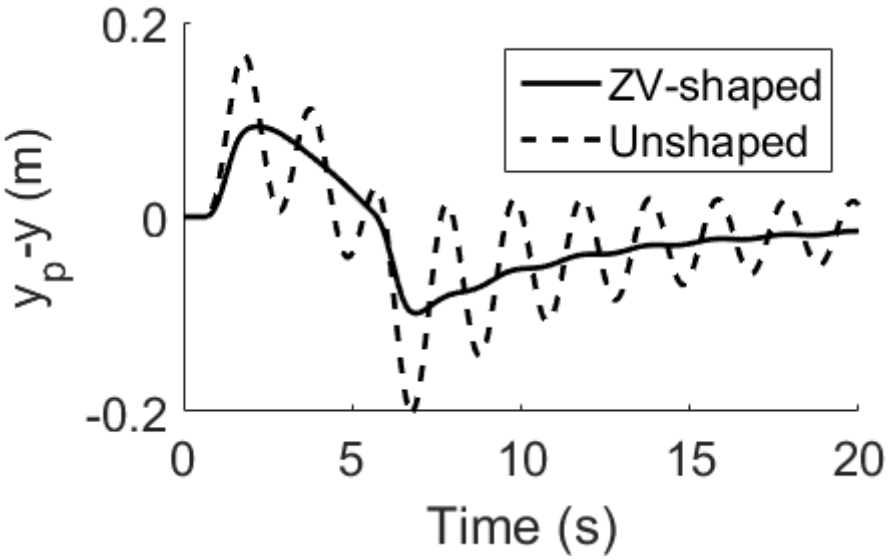

Figure 4.13: The payload responses of a trapezoidal and ZV-shaped input with nonzero drag forces.

peak-to-peak amplitude is somewhat ambiguous, although a significant reduction in amplitude does occur.

Finally, the comparison where $d = -5$ cm is shown in Figure 4.14. Because the response has different frequencies than the single pendulum case, the five-second pulse duration no longer corresponds to a maximum. An input shaper targeting a different frequency or set of frequencies would produce better results.

In a real-world situation, however, more than one pulse command will likely be used. Consider the case when two pulses of the same duration, one positive and one negative, are used as the input. The time between the pulses is the quantity being varied. The amplitude of the residual oscillation after the second pulse is shown in Figure 4.15. The maximum first transient and minimum second transient peak-to-peak amplitudes are shown in Figure 4.16. These are both periodic, as before, with a period equal to the period of the payload oscillation.

When a ZV shaper is applied, the transient peak-to-peak is constant at 0.2 m, up to a 58 percent reduction from the unshaped input. In the residual part, the results are similar to those of the single-pulse trial, where oscillations are reduced by up to 99 percent. Fig-

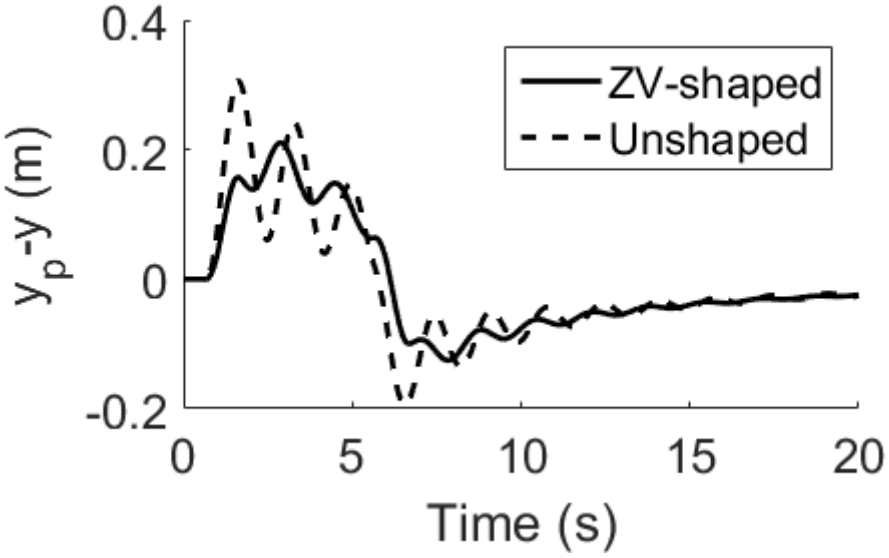

Figure 4.14: The payload responses of a trapezoidal and ZV-shaped input for the double pendulum case.

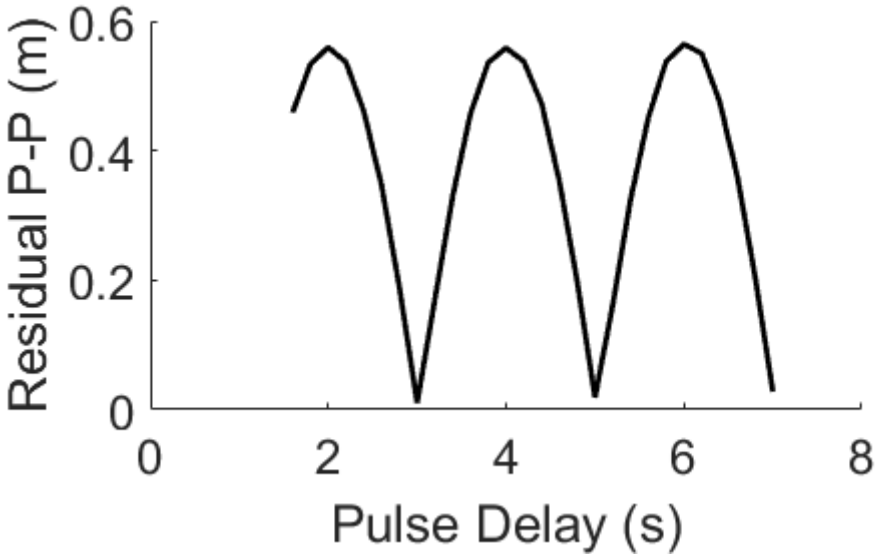

Figure 4.15: The residual payload response of two trapezoidal pulses as a function of the time between the pulses.

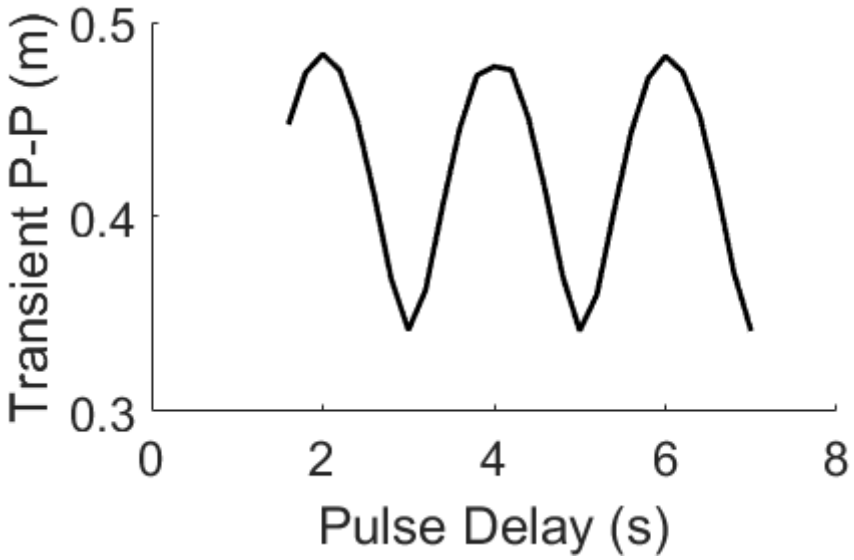

Figure 4.16: The transient payload response of two trapezoidal pulses as a function of the time between the pulses.

ure 4.17 shows an example response when the delay between pulses is four seconds long, corresponding to a maximum in the residual amplitude.

4.6 ZV robustness

The ZV shaper has only a small amount of robustness to modeling errors. If the payload length is different than that used to design the shaper, then the oscillation reduction suffers. For a five-second single pulse with no drag and $d = 0$ cm, where the ZV shaper is targeted at $L_{\text{p}} = 1$ m (or $P = 2$ s), the residual amplitude as a function of the period is shown in Figure 4.18.

If one defines a tolerable amount of oscillation, e.g. up to 0.2 cm, then the range of usable periods (and, thus, payload lengths) widens. A misidentified payload length within this range would still yield an acceptable amount of oscillation, by definition. More complex input shapers would have a "flatter" minimum or would have multiple minima, both corresponding to additional usable payload lengths.

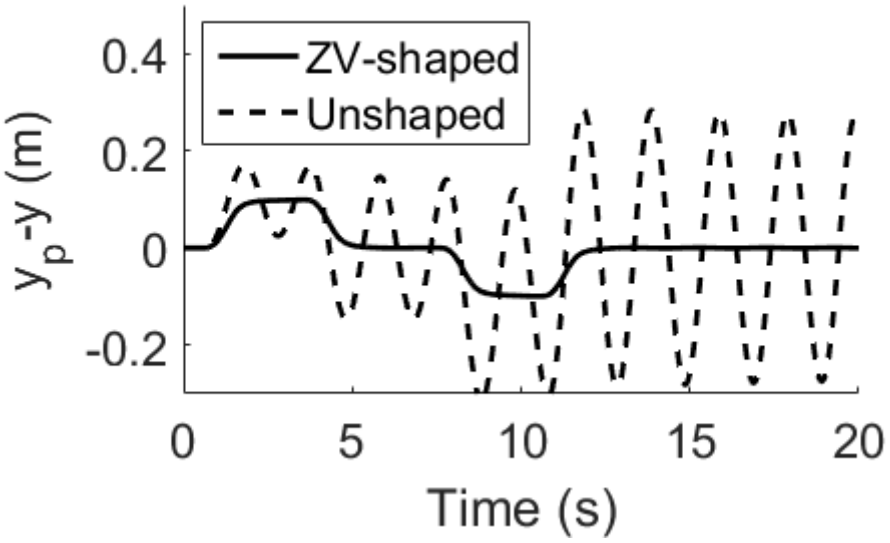

Figure 4.17: The payload responses of an example two-pulse input command.

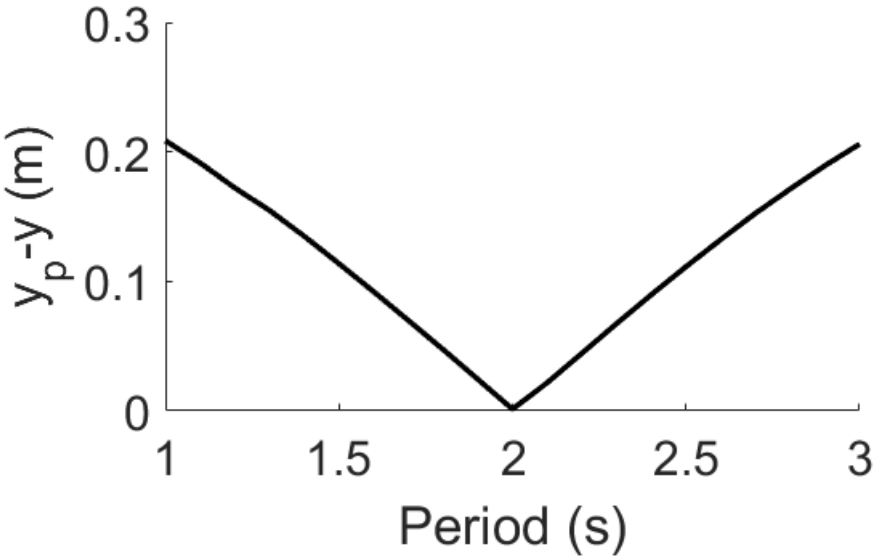

Figure 4.18: The robustness of a ZV shaper targeted at $P = 2$ s.

CHAPTER 5
QUADROTOR DESIGN

The 3D model of the fully unfolded quadrotor is shown in Figure 5.1, and the physical version is presented in the photograph shown in Figure 5.2. The main components of the quadrotor are the frame, the arms and folding mechanism, motors and mounts, propellers, batteries, a GPS unit, and the Pixracer microcontroller.

5.1 Mechanical design

5.1.1 Frame

The frame, pictured in Figure 5.3, was designed to hold and protect the microcontroller and provide structure to the folding mechanism. The frame was 3D-printed for rapid prototyping. The aluminum arm holder was screwed into the frame, and the other electronics sat at the top of the frame, shown in Figure 5.4.

5.1.2 Folding mechanism

The initial goal of the physical quadrotor was to have it participate in the CanSat competition, an annual competition organized by the American Astronautical Society (AAS) and the American Institute of Aeronautics and Astronautics (AIAA). In this event, a rocket carrying the system is launched several thousand meters up. Near apogee, the system is ejected from the rocket and must both land on the ground and autonomously reach a target. Without a folding design, only a small quadrotor would have been able to fit within the diameter of the rocket. A small quadrotor could not generate enough thrust or carry enough batteries for it to reach the target before running out of charge. Although the components of the folding design add weight, it allows a larger quadrotor to participate. Using springs

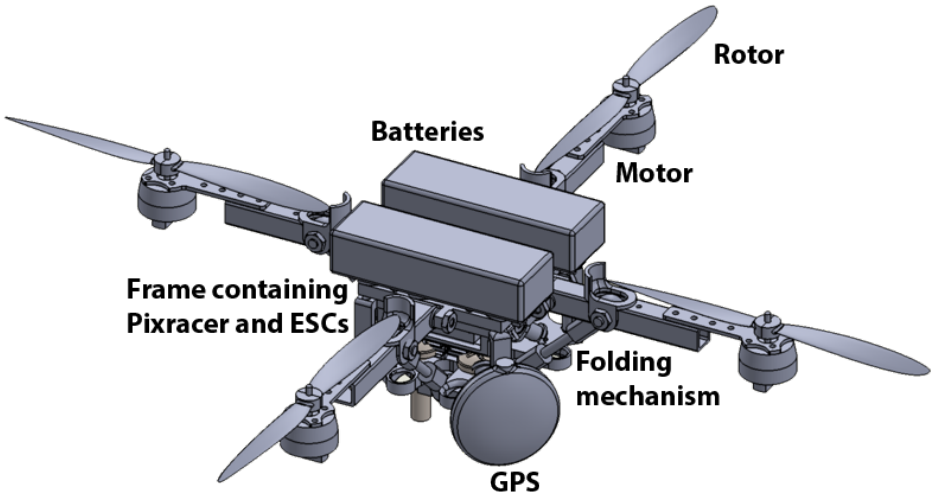

Figure 5.1: The 3D model of the quadrotor in its unfolded state.

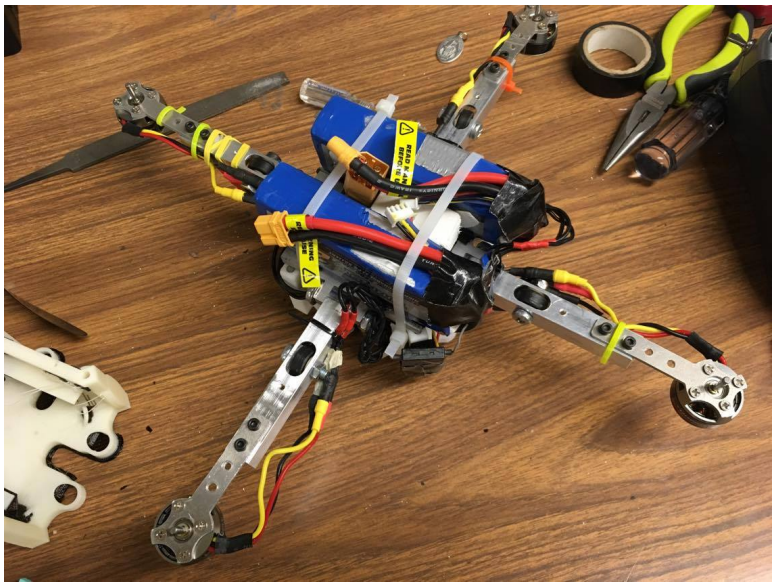

Figure 5.2: The physical quadrotor in its unfolded state, without rotors.

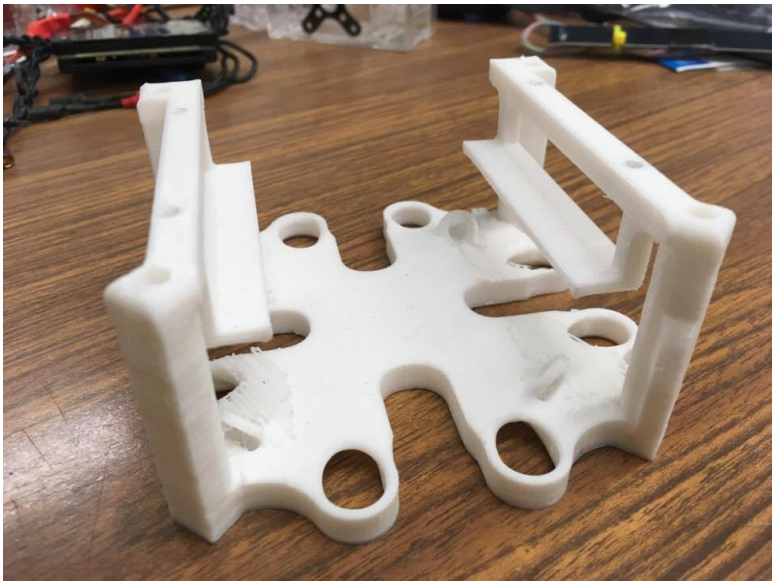

Figure 5.3: The 3D-printed frame that holds the electronics.

Figure 5.4: A side view of the frame showing how the springs line up.

that are in tension when folded, the arms unfold when they are no longer contained by the rocket body. Upon detecting freefall, the quadrotor can then begin autonomous operation.

The folding design can be useful in other applications. A quadrotor must have powerful motors if it is to carry heavy payloads. Large rotors are needed to provide more thrust, and a larger frame is required to improve controllability and so that the rotors do not collide with each other during operation. This large frame may become unwieldy to transport from one location to another, e.g., in the trunk of a car. Using a folding design, a quadrotor with a larger wingspan can be transported in a smaller space. Moreover, the quadrotor in a rocket may be adapted to a real-world scenario. The rocket can travel several miles away, possibly after a natural disaster where terrain is temporarily inaccessible. The rocket can then deploy the quadrotor, which unfolds and begins operation.

The folding mechanism is shown in Figures 5.5 and 5.6, the 3D model and physical quadrotor, respectively. The folding action was achieved by using springs, rods attached to the arms, ball joints, and a central base where the rods and springs are connected. The mechanism is shown in Figure 5.7. The four hollow aluminum arms, shown in Figure 5.8, are held in place by four brass rods attached to a thick aluminum base. The springs, always in tension, keep the base in place, which keeps the rods in place, which keep the arms in place. When the arms are folded down, the rods push the base down, which stretches the springs more. If the arms are released, the springs contract, raising the arms back into place. Ball joints connect the rods to the arms and to the base as there is a slight pivot required when folding or unfolding the arms due to where they are attached on the base.

The springs in the design do a good job of guiding the base into the proper position. With uneven unfolding, the arms are not orthogonal with respect to the vertical axis, and the quadrotor drifts. This drift requires additional thrust and torque to overcome. Additionally, the arms appear to be rigid during flight, not inducing additional modes of vibration.

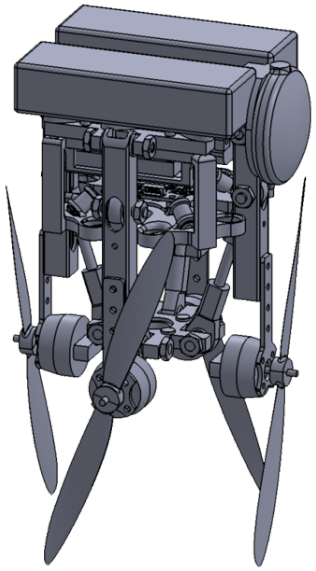

Figure 5.5: The 3D model of the quadrotor in its folded state.

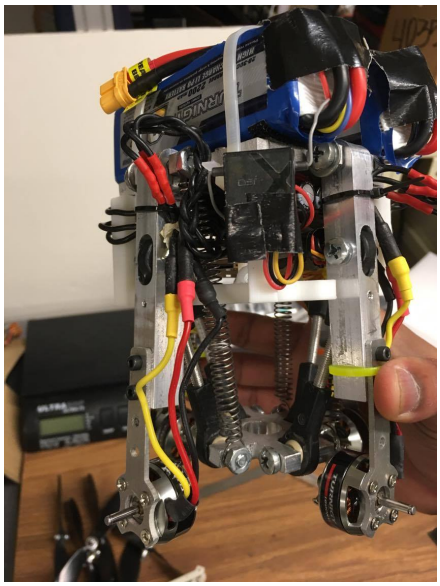

Figure 5.6: The physical quadrotor in its folded state.

Figure 5.7: The base, springs, and rods that make up the folding mechanism.

Figure 5.8: One of the four arms of the quadrotor with the motor mount attached.

Figure 5.9: One of the Turnigy Park 300 1600kv motors used.

5.1.3 Motors

The motors selected were four Turnigy Park 300 1600kv brushless outrunners, pictured in Figure 5.9. The kv rating (PRM per voltage) was chosen through trial and error: both the 1080kv and 1380kv versions did not provide enough thrust. Each 1600kv motor provided a maximum of about 420 grams of thrust, 1.68 kg in total, which is enough to lift the 1.0 kg quadrotor and have enough excess thrust for control and payloads. Thrust was measured on a stand with a scale underneath, as shown in Figure 5.10.

The thrust depends on the size of the rotors being used, with larger rotors providing more thrust per rotation but also being harder to rotate due to the increased inertia. The geometry of the quadrotor also restricts the rotor size, where rotors cannot ever collide with each other or with anything else on the quadrotor. Based on geometry, 7-inch propellers were chosen with a pitch of 4 (a “7x4” propeller). Two of the propellers were oriented clockwise, and the other two are oriented counter-clockwise. On the quadrotor, opposing motors should have similarly oriented propellers, and adjacent motors should be opposite.

These motors drew a significant current, upwards of 5.0 A each for maximum thrust, so high charge-density batteries are a must if the intent is to fly for a reasonable amount

Figure 5.10: The quadrotor on the test rig for thrust measurements.

of time. The batteries must also be able to discharge that much current over a long time. Lithium-polymer (LiPo) batteries were chosen to deal with both issues.

5.1.4 Batteries

There are a wide variety of LiPo batteries to choose from, but voltage, size, discharge rate, and capacity are the parameters by which a battery should be determined. Because the motor speed is determined by the supplied voltage, the battery of choice had to be 11.1 V. The size constraint was important so that the battery does not interfere with the folding mechanism. The discharge rate and capacity together determined how much current the batteries can supply, with current equal to the discharge rate times the capacity. Although some batteries have a "burst" discharge rate, the current should be sustainable for longer periods of time.

The battery chosen was the Turnigy 2200 mAh 3S 35C Lipo Pack, which can provide $2.2 \text{ Ah} * 35 \text{ C} = 77 \text{ A}$ of current, where "C" is not coulombs but rather the discharge rate, which has units of inverse hours. Two of these batteries were placed in parallel to double the amount of charge the quadrotor can operate with. This did not double the flight time,

Figure 5.11: The QBrain ESC hub that regulates the voltage to the motors.

however, as the extra weight of the additional battery required more thrust and, thus, more current to keep aloft. Using the linear relationship between current draw and thrust, one could estimate the new flight time and compare it to the old time. In this case, the second battery added approximately 60 percent more flight time.

5.1.5 Electronic speed controllers

In order to vary the voltage sent to the motors and control their speed, electronic speed controllers (ESCs) must be used. These convert the DC battery current into three inputs that are used to control each motor. The control board communicates with the ESCs through pulse-width modulation, and there is one ESC per motor. Each ESC has a current rating, so the maximum current each motor can draw should be less than the current rating of its respective ESC.

A QBrain 4x25A ESC hub for quadrotors, pictured in Figure 5.11, was used to simplify the hookups.

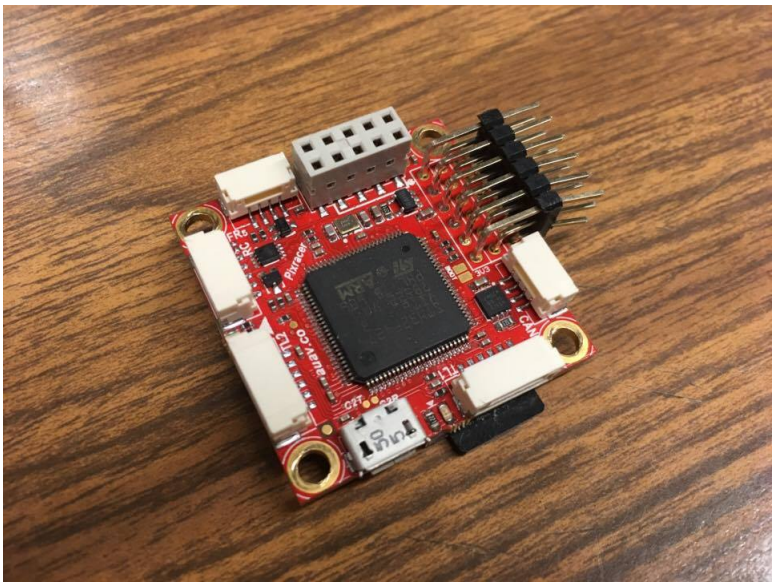

Figure 5.12: A top view of the Pixracer microcontroller.

5.1.6 Microcontroller

The microcontroller of a quadrotor is responsible for taking sensor inputs and adjusting the RPM of each motor correspondingly. This is complex for a user to have to code, so quadrotor libraries were utilized. The complexity arises due to accelerometer and gyroscope noise and drift, RC input, GPS measurements, estimating position or velocity or attitude, and PID feedback control. Additional complexity arises for non-essential functionality, including data logging, telemetry, autonomous control, etc. Initially, an Arduino Uno was used, but it was quickly determined that its CPU could not handle what was required for the quadrotor to be controllable and also to estimate its states. Furthermore, a large portion of the codebase would have to be self-developed. Therefore, the open-source Pixracer microcontroller was selected, depicted in Figure 5.12.

The Pixracer software performs the complex operations to fly and estimate states. Moreover, the software is extensible and allows for users to edit existing programs or create new ones, such as one that stores a buffer for input shaping. The board has an internal IMU and has hookups for other sensors. With the QGroundControl flight control software, one can easily change system parameters, plot waypoints on a map, and calibrate the software.

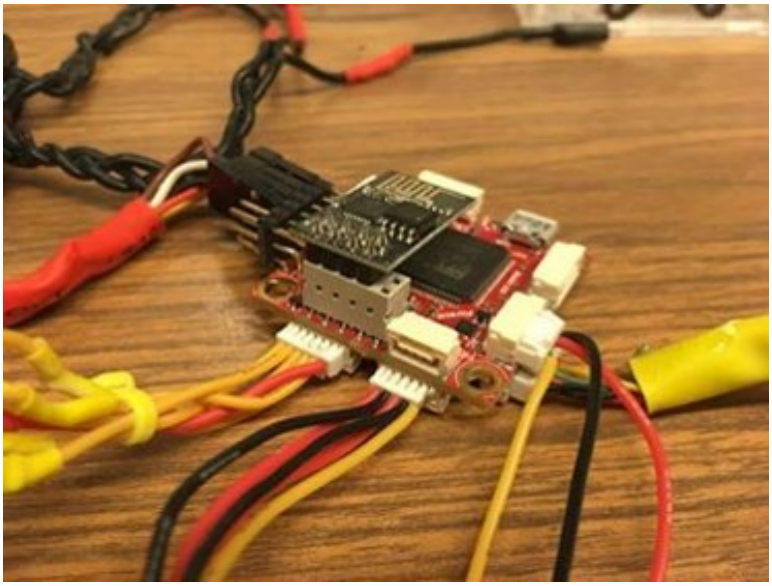

Figure 5.13: The Pixracer with all sensors attached.

5.1.7 Sensors

The Pixracer has several sensors built-in and allows for additional ones to be added. Internally, it contains an accelerometer, gyroscope, magnetometer with temperature compensation, and barometer. Externally, one can add a GPS unit, WiFi board, arming switch, and RC receiver, some of which are shown in Figure 5.13. GPS is required for several of the position-related control modes. WiFi allows the quadrotor to communicate with software like QGroundControl for telemetry or updating parameters. The arming switch is a safety feature that enables and disables the ESCs. The RC receiver is required for manual control, paired with a transmitter.

5.1.8 Design challenges

The main design challenges were to reduce the weight, optimize space for the electronics, have it unfold without human interaction, and respond to RC pitch and roll input. These were all accomplished in the current design, but more challenges remain.

High frequency vibrations caused by the four motors can vibrate screws and wires out. If a motor loses signal due to a wire coming out, the Pixracer microcontroller senses that

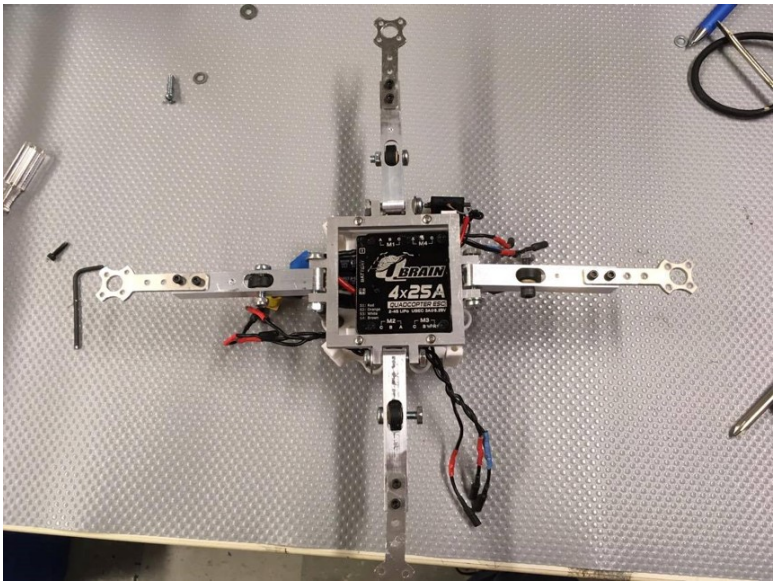

Figure 5.14: An overhead view of the quadrotor, highlighting its symmetry.

and attempts to land the quadrotor in a stable way. The code uses a low-pass filter, which attenuates noise in the IMU. Although the quadrotor seemed to behave no differently with and without dampers under the IMU, this was not quantitatively observed. To do so, the onboard position, velocity, and attitude would have to be compared to that measured in a motion capture facility. Furthermore, the system parameters used in simulation that can be obtained during flight would have to be acquired in the same way.

Because the base of the quadrotor is relatively small, landing without crashing is another design challenge. A wide foam pad is affixed to the base to both act as a shock absorber on impact and provide a wider base to land on. This foam does not interfere with the folding mechanism, although it may interfere with where payloads are suspended from.

Another concern is that due to the symmetric design, it is not immediately clear which direction is "forward," i.e. which way a pitch or roll command will tilt the quadrotor, highlighted by Figure 5.14. A single colored rotor in the direction of $+y$ is used to denote direction.

The amount of wires is a concern. If the wires are long, they must be tied down so they do not interfere with the rotors. If they are short, they weigh less but cannot be lengthened

if the design changes and requires it. In a more final design, the wires of the motors, ESCs, and external sensors could be shortened and connectors re-spliced to reduce tangle and weight.

Finally, if the quadrotor does crash, the ability to remake or reacquire parts is a concern. Although 3D-printed parts seem to break more easily, they are easy to recreate. If an aluminum part bends, it may be able to be bent back into shape or used in its bent state. Depending on the part, it may be easy to remake or tedious if it requires machining. In any case, the rotors are likely the most fragile part of the quadrotor, as they are thin, protruding, and spinning at a high frequency.

CHAPTER 6
FUTURE WORK AND SUMMARY

6.1 Future work

Input shaping was shown in simulation to be successful, but the techniques must now be tested on a real system. The Pixracer microcontroller has an extensible codebase, allowing custom modules. An input shaping buffer within the attitude control module would enable an operator to switch between attitude control and input-shaped attitude control modes. Further testing could be performed in a motion capture facility to quantify the payload oscillations, which can then be compared to the results obtained from simulation.

Additionally, modeling wind and sensor noise may prove useful in determining how position and velocity estimates drift over time. These phenomena can be treated as random Gaussian noise or a function with noise added, although the values should be deterministic for simulations.

Finally, an articulated cable model can replace the inelastic cable assumption presented. An articulated cable can more accurately model the slack in a cable. For large payload angles or in a rapid change of direction, these effects may become important to more fully understand the dynamics of the system.

6.2 Summary

In this thesis, the equations of motion of a quadrotor with a slung load were derived in Chapter 2. The model includes aerodynamic effects such as drag and the center of pressure as well as load-attitude coupling that causes double-pendulum-like behavior.

This allowed the application of PID feedback to control the system in Chapter 3. This type of control is easy to implement and does not require knowledge of the payload. Sensor

limitations were investigated, and it was determined that attitude control was qualitatively more reliable than position or velocity control.

Furthermore, input shaping techniques were implemented in the simulations in Chapter 4. The results showed that input shaping greatly reduces the oscillation of the payload. Because input shaping has been a successful approach in similar real-world systems, including cranes and helicopters, the presented work can be implemented on a real quadrotor with the expectation that the payload swing will be reduced.

Reducing the payload oscillations with input shaping has many benefits. With a reduced maximum transient amplitude, the payload is less likely to cause the quadrotor to destabilize, decreasing the chance that the quadrotor or payload fall and crash. Additionally, less residual payload swing is beneficial when the quadrotor must drop the payload onto a target position. Waiting for the payload swing to damp out is a common but battery-wasting method of reducing the residual swing. By using input shaping, there is little residual swing, and the quadrotor can reserve battery life for additional maneuvers.

Moreover, the code for these simulations was included in Appendix A. This provides a starting point for others to perform additional testing, including changing model parameters and applying other phenomena to the system.

Finally, a folding quadrotor design was shown in Chapter 5. The folding aspect reduces the wingspan of the quadrotor during transportation, allowing a wider and more powerful quadrotor to be brought to a location more easily. In experimentation, the folding mechanism did not appear to induce additional modes of vibration, suggesting that the presented design is fairly rigid in its unfolded state during operation.

Appendices

APPENDIX A

MATLAB CODE FOR SIMULATIONS

The code used for simulations was written in MATLAB and is included below. The entry points are *model.m* for automatic control and *modelCont.m* for manual control. The four included controllers are *attitude.m* (PD attitude control), *far.m* (PD position control), *path.m* (PID position control), and *vel.m* (PI velocity control). These controllers are the inputs to *ode45*. The initial conditions, controller gains, and model parameters are defined in *modelParams.m*. *modelResults.m* contains the values that can be plotted, e.g. *plot(T, thrust)*.

```
%% addGlobalForce.m
function [F_sum, T_sum] = addGlobalForce(force, r_off, R, F_sum, T_sum)
% sum of external forces/torques =
    % function( force to add, location of force, rotation matrix, current values of
force/torque )
F_sum = F_sum + force;
T_sum = T_sum + cross(R * r_off, force);
end


%% addLocalForce.m
function [F_sum, T_sum] = addLocalForce(force, r_off, R, F_sum, T_sum)
% sum of external forces/torques =
    % function( force to add, location of force, rotation matrix, current values of
force/torque )
F_sum = F_sum + R * force;
T_sum = T_sum + cross(r_off, force);
end


%% attitude.m
function [ x_d ] = attitude( t, x )
% attitude control

global state;

F_sum = [0; 0; 0]; % sum of external forces
T_sum = [0; 0; 0]; % sum of external torques

[r, xi, r_d, xi_d, r_i, r_d_i, thrust, tau, xi_p, xi_p_d, charge] = getStates(x);

r_e = state.r_start - r; % maintain a height with error

R = getRot(xi); % rotation matrix based on the current angles

ctcp = R(3, 3); % cos(theta) * cos(phi), the projection onto the horizontal plane

% add thrust to the sum of forces F_ext
[F_sum, T_sum] = addLocalForce([0; 0; thrust], [0; 0; 0], R, F_sum, T_sum);

% add drag at the center of pressure
[F_sum, T_sum] = addGlobalForce(getDrag(r, r_d, state.Cd * state.area), state.cop, R,
F_sum, T_sum);

% add gravity at the center of gravity
% adjust distance if COG is off-center
[F_sum, T_sum] = addGlobalForce([0; 0; -state.m * state.g], [0; 0; 0], R, F_sum,
T_sum);

r_dd = 1 / state.m * F_sum; % acceleration

T_sum = T_sum + tau - cross(xi_d, state.I * xi_d);
xi_dd = state.iI * T_sum; % angular acceleration

% add payload effects if the mass exists
if state.mp ~= 0

    % implicit accelerations to feed into payload dynamics
    Fp_est = R * [0; 0; -state.mp * state.g];
```

```
    r_dd_est = (F_sum + Fp_est) / state.m;
    xi_dd_est = state.iI * (T_sum + cross([0; 0; state.r_susp], Fp_est));

    [xi_p_dd, localForceOnQuad] = getPayloadParams(R, r, r_d, r_dd_est, xi, xi_d,
xi_dd_est, xi_p, xi_p_d);
    [F_sum, T_sum] = addLocalForce(localForceOnQuad, [0; 0; state.r_susp], R, F_sum,
T_sum);

    % recalc for payload
    r_dd = 1 / state.m * F_sum; % acceleration
    xi_dd = state.iI * T_sum; % angular acceleration
else
    xi_p_dd = [0; 0];
end

gains = state.gains;

% PID position = desired acceleration
r_dd_des = gains.Kp_r * r_e + gains.Kd_r * -r_d + gains.Ki_r * r_i;
% modify the Z gain to include hover thrust, project onto horiz plane
r_dd_des(3) = state.g + ctcp * r_dd_des(3);
thrust_des = (state.m + state.mp) * r_dd_des(3); % desired thrust
% only the z direction matters here

xi_des = state.profile(t);
xi_e = xi_des - xi; % angle error

% PD angles = desired torque
tau_des = gains.Kp_xi * xi_e + gains.Kd_xi * -xi_d;
% torque limiter
tau_d = 20 * (tau_des - tau);

if thrust_des < state.thrust_min % limit to minimum thrust
    thrust_des = state.thrust_min;
end

if thrust_des > state.thrust_max
    thrust_des = state.thrust_max; % limit to maximum thrust
end

% thrust limiter
thrust_d = 16 * (thrust_des - thrust);

charge_d = -thrust / state.thrust_I_ratio; % current draw from motors/ESCs
% possibly also add current draw from sensors and controller

% ode45 output
x_d = [r_d; xi_d; r_dd; xi_dd; r_e; zeros(3, 1); thrust_d; tau_d; xi_p_d; xi_p_dd;
charge_d];
end


%% far.m
function [ x_d ] = far( t, x )
%vel Goes full velocity, ideal for heading to a far-away locations

global state;


F_sum = [0; 0; 0]; % sum of external forces
```

```
T_sum = [0; 0; 0]; % sum of external torques

[r, xi, r_d, xi_d, r_i, r_d_i, thrust, tau, xi_p, xi_p_d, charge] = getStates(x);

r_e = state.r_final - r; % drives toward the final position

R = getRot(xi); % rotation matrix based on the current angles

ctcp = R(3, 3); % cos(theta) * cos(phi), the projection onto the horizontal plane

% add thrust to the sum of forces F_ext
[F_sum, T_sum] = addLocalForce([0; 0; thrust], [0; 0; 0], R, F_sum, T_sum);

% add drag at the center of pressure
[F_sum, T_sum] = addGlobalForce(getDrag(r, r_d, state.Cd * state.area), state.cop, R,
F_sum, T_sum);

% add gravity at the center of gravity
% adjust distance if COG is off-center
[F_sum, T_sum] = addGlobalForce([0; 0; -state.m * state.g], [0; 0; 0], R, F_sum,
T_sum);

r_dd = 1 / state.m * F_sum; % acceleration

T_sum = T_sum + tau - cross(xi_d, state.I * xi_d);
xi_dd = state.iI * T_sum; % angular acceleration

% add payload effects if the mass exists
if state.mp ~= 0

    % implicit accelerations to feed into payload dynamics
    Fp_est = R * [0; 0; -state.mp * state.g];
    r_dd_est = (F_sum + Fp_est) / state.m;
    xi_dd_est = state.iI * (T_sum + cross([0; 0; state.r_susp], Fp_est));

    [xi_p_dd, localForceOnQuad] = getPayloadParams(R, r, r_d, r_dd_est, xi, xi_d,
xi_dd_est, xi_p, xi_p_d);
    [F_sum, T_sum] = addLocalForce(localForceOnQuad, [0; 0; state.r_susp], R, F_sum,
T_sum);

    % recalc for payload
    r_dd = 1 / state.m * F_sum; % acceleration
    xi_dd = state.iI * T_sum; % angular acceleration
else
    xi_p_dd = [0; 0];
end

gains = state.gains;

% PD position = desired acceleration
r_dd_des = gains.Kp_r * r_e + gains.Kd_r * -r_d;
% modify the Z gain to include hover thrust, project onto horiz plane
r_dd_des(3) = state.g + ctcp * r_dd_des(3);
thrust_des = (state.m + state.mp) * r_dd_des(3); % desired thrust

mag = norm(r_dd_des);
if mag == 0
    mag = 1;
end
```

```
% use desired acceleration to find desired angles
% since the quad can only move via changing angles
xi_des = [
    asin(-r_dd_des(2) / mag / cos(xi(2)));
    asin(r_dd_des(1) / mag);
    0
];

mag = norm(xi_des);
if mag > state.maxAngle
    xi_des = xi_des / mag * state.maxAngle;
end

xi_e = xi_des - xi; % angle error

% PD angles = desired torque
tau_des = gains.Kp_xi * xi_e + gains.Kd_xi * -xi_d;
% torque limiter
tau_d = 20 * (tau_des - tau);

if thrust_des < state.thrust_min % limit to minimum thrust
    thrust_des = state.thrust_min;
end

if thrust_des > state.thrust_max
    thrust_des = state.thrust_max; % limit to maximum thrust
end

% thrust limiter
thrust_d = 16 * (thrust_des - thrust);

charge_d = -thrust / state.thrust_I_ratio; % current draw from motors/ESCs
% possibly also add current draw from sensors and controller

% ode45 output
x_d = [r_d; xi_d; r_dd; xi_dd; [zeros(6, 1)]; thrust_d; tau_d; xi_p_d; xi_p_dd;
charge_d];
end


%% getAirDensity.m
function rho = getAirDensity( z )
%getAirDensity Air density as a function of altitude

global state;
rho = 1.22; % kg/m3
% only needed as a function if air density is a function of altitude
end


%% getBufferAt.m
function profile = getBufferAt( t )

global state;

if isempty(state.buffer)
    profile = [0; 0; 0];
    return;
end
```

```
% state.buffer has to be t,x,y,z
i = 1;
tvec = state.buffer(:, 1);

while i <= length(tvec) && tvec(i) <= t
    i = i + 1;
end

if i > 1
    i = i - 1;
end

profile = state.buffer(i, 2:4)';
end


%% getDrag.m
function drag = getDrag(r, r_d, CdA)
drag = .5 * getAirDensity(r(3)) * -r_d .* abs(r_d) * CdA; % N
end


%% getPathAt.m
function next = getPathAt(current)

global state;

next = zeros(3, 1);

for i=1:length(current)

    s = state.r_start(i);
    f = state.r_final(i);
    c = current(i);

    if abs(f - c) < state.distScale
        next(i) = f;
    elseif f > c
        next(i) = s + state.distScale * floor((c - s) / state.distScale + 1);
    else
        next(i) = s - state.distScale * floor((s - c) / state.distScale + 1);
    end
end
end


%% getPayloadParams.m
function [xi_p_dd, localForceOnQuad] = getPayloadParams(R, r, r_d, r_dd, xi, xi_d,
xi_dd, xi_p, xi_p_d)

global state;

invR = inv(R);

x_dd = r_dd(1);
y_dd = r_dd(2);
z_dd = r_dd(3);

phi = xi(1);
```

```
theta = xi(2);
psi = xi(3);

phi_d = xi_d(1);
theta_d = xi_d(2);
psi_d = xi_d(3);

phi_dd = xi_dd(1);
theta_dd = xi_dd(2);
psi_dd = xi_dd(3);

phi_p = xi_p(1);
theta_p = xi_p(2);

phi_p_d = xi_p_d(1);
theta_p_d = xi_p_d(2);

cp = cos(phi);
ct = cos(theta);
sp = sin(phi);
st = sin(theta);

cpp = cos(phi_p);
ctp = cos(theta_p);
spp = sin(phi_p);
stp = sin(theta_p);
tpp = tan(phi_p);
ttp = tan(theta_p);

g_xi = state.g * [
    -cp * st;
    sp;
    cp * ct
];

rl = state.Lp * [
    cpp * stp;
    -spp;
    -cpp * ctp
];

rp = rl + [0; 0; state.r_susp];

rl_d = state.Lp * [
    cpp * ctp * theta_p_d - spp * stp * phi_p_d;
    -cpp * phi_p_d;
    ctp * spp * phi_p_d + cpp * stp * theta_p_d
];

rp_d = r_d + rl_d + cross(xi_d, rp);

drag_acc = invR * getDrag(r + rp, rp_d, state.Cdp * state.Ap) / state.mp;
fx = drag_acc(1);
fy = drag_acc(2);
fz = drag_acc(3);

% linearized: state.g * (xi(2) - xi_p(2))
theta_p_dd = -ctp*stp*phi_d^2+2*tpp*theta_p_d*phi_p_d-
2*ctp*phi_p_d*psi_d+ctp*stp*psi_d^2+phi_d*(2*stp*phi_p_d+(ctp^2-
stp^2)*psi_d)+tpp*theta_d*(ctp*phi_d-2*phi_p_d+stp*psi_d)+theta_dd+stp*tpp*phi_dd-
```

```
ctp*tpp*psi_dd+(ctp*(fx+state.g*cp*st)+(fz-
state.g*ct*cp)*stp+ctp*x_dd+cpp^2*stp*z_dd)/state.Lp/cpp;

% linearized: state.g * (xi(1) - xi_p(1))
phi_p_dd = (-spp*cpp*theta_d^2-spp*cpp*theta_p_d^2+spp*cpp*(stp*phi_d-
ctp*psi_d)^2+2*cpp^2*theta_p_d*(-
stp*phi_d+ctp*psi_d)+theta_d*(2*spp*cpp*theta_p_d+(cpp^2-spp^2)*(stp*phi_d-
ctp*psi_d))-(cpp*(fy-state.g*sp)+(-
fz*ctp+state.g*(st*stp+ct*ctp)*cp+fx*stp)*spp+cpp*y_dd)/state.Lp+spp*(-
stp*x_dd+ctp*z_dd)/state.Lp+ctp*phi_dd+stp*psi_dd);

rl_dd = state.Lp * [
    cpp * ctp * theta_p_dd - cpp * stp * (theta_p_d^2 + phi_p_d^2) - spp * stp *
phi_p_dd - 2 * ctp * spp * theta_p_d * phi_p_d;
    -cpp * phi_p_dd + spp * phi_p_d^2;
    cpp * ctp * (phi_p_d^2 + theta_p_d^2) + ctp * spp * phi_p_dd + cpp * stp *
theta_p_dd - 2 * spp * stp * theta_p_d * phi_p_d
];

rp_dd = r_dd + rl_dd + cross(xi_dd, rl) + 2 * cross(xi_d, rl_d) + cross(xi_d,
cross(xi_d, rl));

xi_p_dd = [phi_p_dd; theta_p_dd];

localForceOnQuad = state.mp * (rp_dd - g_xi + drag_acc);
end


%% getProfileAt.m
function [ profile ] = getProfileAt( t )

global state;

amp = 1;
accTime = .1;
moveTime = 20;

period = 2 * pi * sqrt(state.Lp / state.g);

% only one direction is currently being used
xx = 0;
yy = 0;
zz = 0;

% zv
if t < accTime
    yy = t / accTime * amp/2;
elseif t < period/2
    yy = amp/2;
elseif t < period/2 + accTime
    yy = amp/2 + (t - period/2) / accTime * amp/2;
elseif t < moveTime
    yy = amp;
elseif t < moveTime + accTime
    yy = amp - (t - moveTime) / accTime * amp/2;
elseif t < moveTime + period/2
    yy = amp/2;
elseif t < moveTime + period/2 + accTime
    yy = amp/2 - (t - moveTime - period/2) / accTime * amp/2;
else
```

```
    yy = 0;
end

% trapezoidal
% if t < accTime
%     yy = t / accTime * amp;
% elseif t < moveTime
%     yy = amp;
% elseif t < moveTime + accTime
%     yy = amp - (t - moveTime) / accTime * amp;
% else
%     yy = 0;
% end

% unshaped
% if t < moveTime
%     yy = amp;
% else
%     yy = 0;
% end

profile = [xx; yy; zz];
end


%% getRot.m
function [R] = getRot( angle )
%getRot Calculates the ZXY rotation matrix based on the angle

a_phi = angle(1);
a_theta = angle(2);
a_psi = angle(3);

c1 = cos(a_phi);
s1 = sin(a_phi);
c2 = cos(a_theta);
s2 = sin(a_theta);
c3 = cos(a_psi);
s3 = sin(a_psi);

% ZXY rotation matrix
R = [
    c2 * c3 - s1 * s2 * s3, -c1 * s3, c3 * s2 + c2 * s1 * s3;
    c2 * s3 + c3 * s1 * s2, c1 * c3, s3 * s2 - c3 * c2 * s1;
    -c1 * s2, s1, c1 * c2
];
end


%% getShaper.m
function buffer = getShaper( current, desired, baseTime )

global state;

% current is row vec: 0,0,0
% desired is col vec: 0;0;0

desiredT = desired';

% only when there's a new desired, otherwise keep following the old one
```

```
if all(current == desiredT)
    buffer = [baseTime, current];
else

    period = 2 * pi * sqrt(state.Lp / state.g);
    accTime = .1;
    dt = 0.01;

    % unshaped bang
    if strcmp(state.shaper, 'bang')
        buffer = [baseTime, desired'];

    % unshaped trap
    elseif strcmp(state.shaper, 'trap')
        tt = 0;
        i = 1;
        buffer = zeros(11, 4);
        while abs(tt - accTime) > dt
            buffer(i, :) = [baseTime + tt, current + (desiredT - current) * tt /
accTime];
            tt = tt + dt;
            i = i + 1;
        end

        buffer(i, :) = [baseTime + accTime, desiredT];

    % zv bang
    elseif strcmp(state.shaper, 'zv')
        buffer = [
            baseTime, current + (desiredT - current) / 2;
            baseTime + period/2, desiredT
        ];

    % zv trap
    elseif strcmp(state.shaper, 'zv_t')

        tt = 0;
        i = 1;

        buffer = zeros(21, 4);
        while abs(tt - accTime) > dt
            buffer(i, :) = [baseTime + tt, current + (desiredT - current) * tt /
accTime / 2];
            tt = tt + dt;
            i = i + 1;
        end

        buffer(i, :) = [baseTime + accTime, current + (desiredT - current) / 2];

        tt = 0;
        i = i + 1;

        while abs(tt - accTime) > dt
            buffer(i, :) = [baseTime + tt + period/2, (current + desiredT)/2 * (1 - tt
/ accTime) + desiredT * tt / accTime];
            tt = tt + dt;
            i = i + 1;
        end

        buffer(i, :) = [baseTime + period/2 + accTime, desiredT];
```

```
    end
end
end


%% getStates.m
function [ r, angle, r_d, angle_d, r_i, r_d_i, thrust, tau, angle_p, angle_p_d, charge
] = getStates( x )

r = x(1:3); % xyz
angle = x(4:6); % phi theta psi
r_d = x(7:9); % xyz dots
angle_d = x(10:12); % phi theta psi dots
r_i = x(13:15); % xyz integral
r_d_i = x(16:18); % xyz_d integral
thrust = x(19); % thrust
tau = x(20:22); % torque
angle_p = x(23:24); % payload angles
angle_p_d = x(25:26); % payload angle dots
charge = x(27); % charge, used in outOfCharge
end


%% handleBuffer.m
function handleBuffer( ti, X )

global state;

bufferL = [];
bufferR = [];
bufferU = [];
bufferD = [];

amp = .1;

% allow multiple orthogonal directions at the same time
if state.prevKeys.L

    if state.keys.L
        bufferL = getShaper(X(end, 4:6), [0;-amp;0], ti);
    else
        bufferL = getShaper(X(end, 4:6), [0;0;0], ti);
    end

    state.prevKeys.L = false;

elseif state.prevKeys.R

    if state.keys.R
        bufferR = getShaper(X(end, 4:6), [0;amp;0], ti);
    else
        bufferR = getShaper(X(end, 4:6), [0;0;0], ti);
    end

    state.prevKeys.R = false;

end

if state.prevKeys.U
```

```
    if state.keys.U
        bufferU = getShaper(X(end, 4:6), [-amp;0;0], ti);
    else
        bufferU = getShaper(X(end, 4:6), [0;0;0], ti);
    end

    state.prevKeys.U = false;

elseif state.prevKeys.D

    if state.keys.D
        bufferD = getShaper(X(end, 4:6), [amp;0;0], ti);
    else
        bufferD = getShaper(X(end, 4:6), [0;0;0], ti);
    end

    state.prevKeys.D = false;
end

buffer = [];

% this isn't perfect but it works when the times match
% should discretize time more
if ~isempty(bufferL)
    buffer = bufferL;
end

if ~isempty(bufferR)
    if isempty(buffer)
        buffer = bufferR;
    else
        buffer(:, 2:4) = buffer(:, 2:4) + bufferR(:, 2:4);
    end
end

if ~isempty(bufferU)
    if isempty(buffer)
        buffer = bufferU;
    else
        buffer(:, 2:4) = buffer(:, 2:4) + bufferU(:, 2:4);
    end
end

if ~isempty(bufferD)
    if isempty(buffer)
        buffer = bufferD;
    else
        buffer(:, 2:4) = buffer(:, 2:4) + bufferD(:, 2:4);
    end
end

% insert buffer into state.buffer
if ~isempty(buffer)
    mergeBuffers(buffer);
end
end

function mergeBuffers( newBuffer )
```

```
global state;

[rs, ~] = size(state.buffer);
[rn, ~] = size(newBuffer);

if isempty(state.buffer) || state.buffer(end, 1) < newBuffer(1, 1)
    state.buffer(rs+1:rs+rn, :) = newBuffer;
else
    for i = 1:rn

        newT = newBuffer(i, 1);

        for j = 1:rs

            stateT = state.buffer(j, 1);

            if newT > stateT
                if j == rs
                    state.buffer(end+1, :) = newBuffer(i, :);
                else
                    continue;
                end
            elseif newT < stateT
                state.buffer(j+1:end+1, :) = state.buffer(j:end, :);
                state.buffer(j, :) = newBuffer(i, :);
                rs = rs + 1;
                break;
            else
                state.buffer(j, 2:end) = state.buffer(j, 2:end) + newBuffer(i, 2:end);
                break;
            end
        end
    end
end
end


%% kpDown.m
function kpDown(~, event)

global state;

i = double(event.Character);

if i == 27

    state.keys.ESC = true;

elseif i == 28

    if ~state.keys.L
        state.keys.L = true;
        state.prevKeys.L = true;
    end

elseif i == 29

    if ~state.keys.R
        state.keys.R = true;
        state.prevKeys.R = true;
```

```
    end

elseif i == 30

    if ~state.keys.U
        state.keys.U = true;
        state.prevKeys.U = true;
    end

elseif i == 31

    if ~state.keys.D
        state.keys.D = true;
        state.prevKeys.D = true;
    end

end


%% kpUp.m
function kpUp(~, event)

global state;

i = double(event.Character);

if i == 28

    if state.keys.L
        state.keys.L = false;
        state.prevKeys.L = true;
    end

elseif i == 29

    if state.keys.R
        state.keys.R = false;
        state.prevKeys.R = true;
    end

elseif i == 30

    if state.keys.U
        state.keys.U = false;
        state.prevKeys.U = true;
    end

elseif i == 31

    if state.keys.D
        state.keys.D = false;
        state.prevKeys.D = true;
    end

end


%% model.m
clear global;
```

```
clear;

global state;

% define parameters
modelParams;

% define time bounds
t0 = 0;
tf = 20;

% define input
state.profile = @getProfileAt;

% integrate
[T, X, Te, Xe] = ode45(@far, [t0, tf], [r_0; xi_0; r_d_0; xi_d_0; r_i_0; r_d_i_0;
thrust_0; tau_0; xi_p_0; xi_p_d_0; charge_0]);

% name state variables for easier access
modelResults;


%% modelCont.m
clear global;
clear;

global state;

% define parameters
modelParams;

% so Xi can be reused for each step
Xi = [r_0; xi_0; r_d_0; xi_d_0; r_i_0; r_d_i_0; thrust_0; tau_0; xi_p_0; xi_p_d_0;
charge_0];

% time info
t0 = 0;
tf = -1; % if -1, simulation runs until ESC is pressed
ti = 0;
dt = 0.1;

% final state vector
X = Xi';
T = t0;

% command buffer
state.profile = @getBufferAt;
state.buffer = [0, 0, 0, 0];

% stored input commands to optionally replay
replaying = false;

if exist('commands.mat', 'file') == 2

    load('commands.mat');
    % delete('commands.mat');

    replaying = true;
```

```
    commands_clone = commands;
else
    commands = [0, 0];
end

% keyboard input
fig = figure('KeyPressFcn', @kpDown, 'KeyReleaseFcn', @kpUp);

while ti <= tf || tf < 0

    % if ESC is pressed, end simulation
    if state.keys.ESC
        break;
    end

    % update inputs if replaying from a file
    % otherwise just stores the input
    if replaying

        if ~isempty(commands)

            overrideInput(commands(1, 2));

            commands(1, 1) = commands(1, 1) - dt;

            % issues with eps, should be good enough
            if commands(1, 1) < dt * .1

                commands = commands(2:end, :);

                if isempty(commands)

                    replaying = false;

                    % bc the original commands were culled
                    commands = commands_clone;

                    disp('Done replaying.');
                end
            end
        end

    else
        commands = saveInput(commands, dt);
    end

    % generate command based on keyboard inputs
    handleBuffer(ti, X);

    % integrate
    [Ti, Xi] = ode45(@attitude, [ti, ti + dt], X(end, :));

    % update current values
    X = [X; Xi(2:end, :)]; % initial condits are already in X from prev step
    T = [T; Ti(2:end)];

    ti = ti + dt;

    % cull times < ti, except for the most recent one
```

```
    while state.buffer(1, 1) < ti && size(state.buffer, 1) > 1
        state.buffer = state.buffer(2:end, :);
    end

    % update the plot
    subplot(2, 1, 1);
    plot(T, X(:, 2), 'k-', 'LineWidth', 2); % plotting y
    subplot(2, 1, 2);
    plot(T, X(:, 8), 'k-', 'LineWidth', 2); % plotting y_d
    pause(0.075);
end

% generate data to plot input command
numCommands = size(commands, 1);

inputT = zeros(1, numCommands * 2);
inputC = zeros(1, numCommands * 2);
inTime = 0;
input = 0;
for i = 1:numCommands

    in = commands(i, 2);

    if in == 4
        input = 1;
    elseif in == 8
        input = -1;
    else
        input = 0;
    end

    inputT(i * 2) = inTime;
    inputC(i * 2) = input;

    inTime = inTime + commands(i, 1);

    inputT(i * 2 + 1) = inTime;
    inputC(i * 2 + 1) = input;
end

% name state variables for easier access
modelResults;


%% modelParams.m
global state;

% physical parameters
m = 1; % kg, mass
g = 9.81; % m/s^2, gravity
Ixx = 6815763.88e-9; % kgm^2, xx moment of inertia
Iyy = 6801641.50e-9; % kgm^2, yy moment of inertia
Izz = 4548279.56e-9; % kgm^2, zz moment of inertia
I = [Ixx, 0, 0; 0, Iyy, 0; 0, 0, Izz]; % inertia matrix
iI = inv(I); % inverse of inertia matrix, to simplify calculations
L = .15546; % m, rotor axis to center of grav
r = .0889; % m, radius of props
kv = 1600; % RPM/V, motor ratio
V = 11.1; % V, battery voltage
rpm = kv * V; % RPM, estimate of max motor RPM
thrust_min = 1; % N, minimum total thrust of all four motors combined
```

```
thrust_max = 1.680 * g; % N, total thrust of all four motors combined
k = thrust_max / 4 * g / rpm / rpm; % N/(rpm)^2 (per motor)
b = 1e-9; % Nm/(rad/s)^2 -> Nm/(rpm)^2 (guess)
area = .025; % m^2, estimate of cross-sectional area
Cd = 1; % drag coefficient, .5-1.5 are pretty decent values
cop = [0; 0; 0.05]; % m, center of pressure (where aerodynamic forces are applied
relative to COM)
thrust_I_ratio = .0171 * 4 * 9.81; % N/A, constant ratio of thrust to current draw

% payload parameters
mp = .073; % kg, mass
Lp = 1; % m, length of cable
r_susp = -.05; % m, suspension point of the cable from COM (assuming z-only offset)
Cdp = .5; % drag coefficient of the payload, 0.5 for sphere
Ap = pi * .0508^2; % m^2, estimate of cross-sectional area of the payload

maxAngle = pi / 9; % radians, limit desired angle
distScale = .5; % m, resolution of the path generation
% should prolly separate xy and z in distScale and path gen

% PID gains (position control)
Kp_r = [12, 12, 30]; % proportional position
Kd_r = [6, 6, 6]; % derivative position
Ki_r = [1, 1, 1]; % integral position

% PI gains (velocity control)
Kp_v = [.4, .4, 0];
Ki_v = [.1, .1, 0];

% PD gains (angle control)
Kp_xi = [.90, .90, .90]; % proportional angle
Kd_xi = [.28, .28, .28]; % derivative angle

shaper = 'trap'; % input type ('bang', 'trap', 'zv', 'zv_t')

% global state vector
state = struct('m', m, 'g', g, 'thrust_max', thrust_max, 'thrust_min', thrust_min,
'k', k, 'b', b, 'Ixx', Ixx, 'Iyy', Iyy, 'Izz', Izz, 'I', I, 'iI', iI, 'L', L, 'r', r,
...
    'area', area, 'Cd', Cd, 'cop', cop, ...
    'mp', mp, 'Lp', Lp, 'r_susp', r_susp, 'Cdp', Cdp, 'Ap', Ap, ...
    'maxAngle', maxAngle, 'distScale', distScale, 'thrust_I_ratio', thrust_I_ratio,
...
    'gains', struct('Kp_r', diag(Kp_r), 'Kd_r', diag(Kd_r), 'Ki_r', diag(Ki_r), ...
        'Kp_v', diag(Kp_v), 'Ki_v', diag(Ki_v), ...
        'Kp_xi', diag(Kp_xi), 'Kd_xi', diag(Kd_xi)), ...
    'shaper', shaper ...
);

% arrow key presses
state.keys = struct('U', false, 'D', false, 'L', false, 'R', false, 'ESC', false);
state.prevKeys = struct('U', false, 'D', false, 'L', false, 'R', false);


% Initial/final parameters
r_final = [0; 0; 5]; % final position

% +phi = -y
% +theta = +x
```

```
r_0 = [0; 0; 5]; % initial position
r_d_0 = [0; 0; 0]; % initial velocity
r_i_0 = [0; 0; 0]; % initial position error
r_d_i_0 = [0; 0; 0]; % initial position error
xi_0 = [0; 0; 0]; % initial angle
xi_d_0 = [0; 0; 0]; % initial angular velocity
thrust_0 = (m + mp) * g; % initial thrust
tau_0 = [0; 0; 0]; % initial thrust
xi_p_0 = [0; 0]; % initial payload angle
xi_p_d_0 = [0; 0]; % initial payload angular velocity
charge_0 = 15840; % C, initial charge on the battery: 1 mAh = 3.6 C

state.r_start = r_0;
state.r_final = r_final;


%% modelResults.m
% variables of interest
n = length(T);

x = X(:, 1);
y = X(:, 2);
z = X(:, 3);

r = [x, y, z];

phi = X(:, 4);
theta = X(:, 5);
psi = X(:, 6);

xi = [phi, theta, psi];

x_d = X(:, 7);
y_d = X(:, 8);
z_d = X(:, 9);

r_d = [x_d, y_d, z_d];

phi_d = X(:, 10);
theta_d = X(:, 11);
psi_d = X(:, 12);

xi_d = [phi_d, theta_d, psi_d];

thrust = X(:, 19);

tau_phi = X(:, 20);
tau_theta = X(:, 21);
tau_psi = X(:, 22);

tau = [tau_phi, tau_theta, tau_psi];

phi_p = X(:, 23);
theta_p = X(:, 24);

xi_p = [phi_p, theta_p];

phi_p_d = X(:, 25);
theta_p_d = X(:, 26);
```

```
xi_p_d = [phi_p_d, theta_p_d];

charge = X(:, 27);

xp = zeros(n, 1);
yp = zeros(n, 1);
zp = zeros(n, 1);

xh = zeros(n, 1);
yh = zeros(n, 1);
zh = zeros(n, 1);

susp = [0; 0; r_susp];

for i=1:length(T)

    R = getRot([phi(i); theta(i); psi(i)]);

    r_h = R * susp;
    xh(i) = x(i) + r_h(1);
    yh(i) = y(i) + r_h(2);
    zh(i) = z(i) + r_h(3);

    r_l = R * (Lp * [cos(phi_p(i)) * sin(theta_p(i)); -sin(phi_p(i)); -cos(phi_p(i)) *
cos(theta_p(i))]);

    xp(i) = xh(i) + r_l(1);
    yp(i) = yh(i) + r_l(2);
    zp(i) = zh(i) + r_l(3);
end

r_h = [xh, yh, zh];
r_p = [xp, yp, zp];

% rp_d


%% overrideInput.m
function overrideInput(command)

global state;

L = bitand(command, 1);
R = bitand(command, 2);
U = bitand(command, 4);
D = bitand(command, 8);

if state.keys.L ~= L
    state.keys.L = L;
    state.prevKeys.L = true;
end

if state.keys.R ~= R
    state.keys.R = R;
    state.prevKeys.R = true;
end

if state.keys.U ~= U
```

```
    state.keys.U = U;
    state.prevKeys.U = true;
end

if state.keys.D ~= D
    state.keys.D = D;
    state.prevKeys.D = true;
end
end


%% path.m
function [ x_d ] = path( t, x )

global state;

F_sum = [0; 0; 0]; % sum of external forces
T_sum = [0; 0; 0]; % sum of external torques

[r, xi, r_d, xi_d, r_i, r_d_i, thrust, tau, xi_p, xi_p_d, charge] = getStates(x);

r_e = getPathAt(r) - r; % drives toward the next waypoint

R = getRot(xi); % rotation matrix based on the current angles

ctcp = R(3, 3); % cos(theta) * cos(phi), the projection onto the horizontal plane

% add thrust to the sum of forces F_ext
[F_sum, T_sum] = addLocalForce([0; 0; thrust], [0; 0; 0], R, F_sum, T_sum);

% add drag at the center of pressure
[F_sum, T_sum] = addGlobalForce(getDrag(r, r_d, state.Cd * state.area), state.cop, R,
F_sum, T_sum);

% add gravity at the center of gravity
% adjust distance if COG is off-center
[F_sum, T_sum] = addGlobalForce([0; 0; -state.m * state.g], [0; 0; 0], R, F_sum,
T_sum);

r_dd = 1 / state.m * F_sum; % acceleration

T_sum = T_sum + tau - cross(xi_d, state.I * xi_d);
xi_dd = state.iI * T_sum; % angular acceleration

% add payload effects if the mass exists
if state.mp ~= 0

    % implicit accelerations to feed into payload dynamics
    Fp_est = R * [0; 0; -state.mp * state.g];
    r_dd_est = (F_sum + Fp_est) / state.m;
    xi_dd_est = state.iI * (T_sum + cross([0; 0; state.r_susp], Fp_est));

    [xi_p_dd, localForceOnQuad] = getPayloadParams(R, r, r_d, r_dd_est, xi, xi_d,
xi_dd_est, xi_p, xi_p_d);
    [F_sum, T_sum] = addLocalForce(localForceOnQuad, [0; 0; state.r_susp], R, F_sum,
T_sum);

    % recalc for payload
    r_dd = 1 / state.m * F_sum; % acceleration
```

```
    xi_dd = state.iI * T_sum; % angular acceleration
else
    xi_p_dd = [0; 0];
end

gains = state.gains;

% PID position = desired acceleration
r_dd_des = gains.Kp_r * r_e + gains.Kd_r * -r_d + gains.Ki_r * r_i;
% modify the Z gain to include hover thrust, project onto horiz plane
r_dd_des(3) = state.g + ctcp * r_dd_des(3);
thrust_des = (state.m + state.mp) * r_dd_des(3); % desired thrust

mag = norm(r_dd_des);
if mag == 0
    mag = 1;
end

% use desired acceleration to find desired angles
% since the quad can only move via changing angles
xi_des = [
    asin(-r_dd_des(2) / mag / cos(xi(2)));
    asin(r_dd_des(1) / mag);
    0
];

mag = norm(xi_des);
if mag > state.maxAngle
    xi_des = xi_des / mag * state.maxAngle;
end

xi_e = xi_des - xi; % angle error

% PD angles = desired torque
tau_des = gains.Kp_xi * xi_e + gains.Kd_xi * -xi_d;
% torque limiter
tau_d = 20 * (tau_des - tau);

if thrust_des < state.thrust_min % limit to minimum thrust
    thrust_des = state.thrust_min;
end

if thrust_des > state.thrust_max
    thrust_des = state.thrust_max; % limit to maximum thrust
end

% thrust limiter
thrust_d = 16 * (thrust_des - thrust);

charge_d = -thrust / state.thrust_I_ratio; % current draw from motors/ESCs
% possibly also add current draw from sensors and controller

% ode45 output
x_d = [r_d; xi_d; r_dd; xi_dd; r_e; [zeros(3, 1)]; thrust_d; tau_d; xi_p_d; xi_p_dd;
charge_d];
end


%% saveCommands.m
save('commands.mat', 'commands');
```

```
%% saveInput.m
function commands = saveInput( commands, dt )

global state;

% L = 0x1, R = 0x2, U = 0x4, D = 0x8

% if inputs haven't changed, nothing to check
if ~state.prevKeys.L && ~state.prevKeys.R && ~state.prevKeys.U && ~state.prevKeys.D
    commands(end, 1) = commands(end, 1) + dt;
    return;
end

command = 0;

if state.keys.L
    command = bitor(command, 1);
end

if state.keys.R
    command = bitor(command, 2);
end

if state.keys.U
    command = bitor(command, 4);
end

if state.keys.D
    command = bitor(command, 8);
end

commands(end+1, :) = [dt, command];
end


%% vel.m
function [ x_d ] = vel( t, x )
%vel Follows a velocity profile

global state;

F_sum = [0; 0; 0]; % sum of external forces
T_sum = [0; 0; 0]; % sum of external torques

[r, xi, r_d, xi_d, r_i, r_d_i, thrust, tau, xi_p, xi_p_d, charge] = getStates(x);

r_d_e = state.profile(t) - r_d; % follow a velocity profile
r_e = state.r_start - r;

R = getRot(xi); % rotation matrix based on the current angles

ctcp = R(3, 3); % cos(theta) * cos(phi), the projection onto the horizontal plane

% add thrust to the sum of forces F_ext
[F_sum, T_sum] = addLocalForce([0; 0; thrust], [0; 0; 0], R, F_sum, T_sum);

% add drag at the center of pressure
```

```
[F_sum, T_sum] = addGlobalForce(getDrag(r, r_d, state.Cd * state.area), state.cop, R,
F_sum, T_sum);

% add gravity at the center of gravity
% adjust distance if COG is off-center
[F_sum, T_sum] = addGlobalForce([0; 0; -state.m * state.g], [0; 0; 0], R, F_sum,
T_sum);

r_dd = 1 / state.m * F_sum; % acceleration

T_sum = T_sum + tau - cross(xi_d, state.I * xi_d);
angle_dd = state.iI * T_sum; % angular acceleration

% add payload effects if the mass exists
if state.mp ~= 0

    % implicit accelerations to feed into payload dynamics
    Fp_est = R * [0; 0; -state.mp * state.g];
    r_dd_est = (F_sum + Fp_est) / state.m;
    xi_dd_est = state.iI * (T_sum + cross([0; 0; state.r_susp], Fp_est));

    [xi_p_dd, localForceOnQuad] = getPayloadParams(R, r, r_d, r_dd_est, xi, xi_d,
xi_dd_est, xi_p, xi_p_d);
    [F_sum, T_sum] = addLocalForce(localForceOnQuad, [0; 0; state.r_susp], R, F_sum,
T_sum);

    % recalc for payload
    r_dd = 1 / state.m * F_sum; % acceleration
    angle_dd = state.iI * T_sum; % angular acceleration
else
    xi_p_dd = [0; 0];
end

gains = state.gains;

% so PI(r_d) for xy and PID(r) for z
pi = gains.Kp_v * r_d_e + gains.Ki_v * r_d_i;
pid = gains.Kp_r * r_e + gains.Kd_r * -r_d + gains.Ki_r * r_i;

r_dd_des = [pi(1); pi(2); pid(3)];

r_e(1:2) = 0; % so xy integral doesn't accumulate

% modify the Z gain to include hover thrust, project onto horiz plane
r_dd_des(3) = state.g + ctcp * r_dd_des(3);
thrust_des = (state.m + state.mp) * r_dd_des(3); % desired thrust

mag = norm(r_dd_des);
if mag == 0
    mag = 1;
end

% use desired acceleration to find desired angles
% since the quad can only move via changing angles
xi_des = [
    asin(-r_dd_des(2) / mag / cos(xi(2)));
    asin(r_dd_des(1) / mag);
    0
];
```

```
mag = norm(xi_des);
if mag > state.maxAngle
    xi_des = xi_des / mag * state.maxAngle;
end

angle_e = xi_des - xi; % angle error

% PD angles = desired torque
tau_des = gains.Kp_xi * angle_e + gains.Kd_xi * -xi_d;
% torque limiter
tau_d = 20 * (tau_des - tau);

if thrust_des < state.thrust_min % limit to minimum thrust
    thrust_des = state.thrust_min;
end

if thrust_des > state.thrust_max
    thrust_des = state.thrust_max; % limit to maximum thrust
end

% thrust limiter
thrust_d = 16 * (thrust_des - thrust);

charge_d = -thrust / state.thrust_I_ratio; % current draw from motors/ESCs
% possibly also add current draw from sensors and controller

% ode45 output
x_d = [r_d; xi_d; r_dd; angle_dd; r_e; r_d_e; thrust_d; tau_d; xi_p_d; xi_p_dd;
charge_d];
end
```

REFERENCES

[1] C. J. Adams, "Modeling and control of helicopters carrying suspended loads", Thesis, 2012.

[2] Y. Feng, C. A. Rabbath, and C.-Y. Su, "Modeling of a micro uav with slung payload", in Handbook of Unmanned Aerial Vehicles, K. P. Valavanis and G. J. Vachtsevanos, Eds. Dordrecht: Springer Netherlands, 2015, pp. 1257–1272, ISBN: 978-90-481-9707-1.

[3] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, "Learning swing-free trajectories for uavs with a suspended load", 2013 Ieee International Conference on Robotics and Automation (Icra), pp. 4902–4909, 2013.

[4] M. Hehn and R. D'Andrea, "A flying inverted pendulum", in IEEE International Conference on Robotics and Automation, pp. 763–770.

[5] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors", The International Journal of Robotics Research, vol. 31, no. 5, pp. 664–674, 2012.

[6] I. Palunko, P. Cruz, and R. Fierro, "Agile load transportation : Safe and efficient load manipulation with aerial robots", IEEE Robotics and Automation Magazine, vol. 19, no. 3, pp. 69–79, 2012.

[7] I. Palunko, R. Fierro, and P. Cruz, "Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach", in IEEE International Conference on Robotics and Automation, pp. 2691–2697.

[8] D. Fusato, G. Guglieri, and R. Celi, "Flight dynamics of an articulated rotor helicopter with an external slung load", Journal of the American Helicopter Society, vol. 46, no. 1, pp. 3–13, 2001.

[9] A. L. Salih, M. Moghavvemi, H. A. F. Mohamed, and K. S. Gaeid, "Flight pid controller design for a uav quadrotor", Scientific research and essays, vol. 5, no. 23, pp. 3660–3667, 2010.

[10] D. Kaya and A. T. Kutay, "Aerodynamic modeling and parameter estimation of a quadrotor helicopter".

[11] A. A. Mian and W. Daobo, "Modeling and backstepping-based nonlinear control strategy for a 6 dof quadrotor helicopter", Chinese Journal of Aeronautics, vol. 21, no. 3, pp. 261–268, 2008.

[12] J. Li and Y. Li, "Dynamic analysis and pid control for a quadrotor", in IEEE International Conference on Mechatronics and Automation, pp. 573–578.

[13] B. Erginer and E. Altug, "Modeling and pd control of a quadrotor vtol vehicle", 2007 Ieee Intelligent Vehicles Symposium, Vols 1-3, pp. 1177–1182, 2007.

[14] C. Hancer, K. T. Oner, E. Sirimoglu, E. Cetinsoy, and M. Unel, "Robust hovering control of a quad tilt-wing uav", Iecon 2010 - 36th Annual Conference on Ieee Industrial Electronics Society, 2010.

[15] Y. M. Al-Younes, M. A. Al-Jarrah, and A. A. Jhemi, "Linear vs. nonlinear control techniques for a quadrotor vehicle", 7th International Symposium on Mechatronics and its Applications, 2010.

[16] C. de Crousaz, F. Farshidian, M. Neunert, and J. Buchli, "Unified motion control for dynamic quadrotor maneuvers demonstrated on slung load and rotor failure tasks", 2015 Ieee International Conference on Robotics and Automation (Icra), pp. 2223–2229, 2015.

[17] Y. Feng, C. A. Rabbath, S. Rakheja, and C.-Y. Su, "Adaptive controller design for generic quadrotor aircraft platform subject to slung load", pp. 1135–1139.

[18] J. Vaughan, D. Kim, and W. Singhose, "Control of tower cranes with double-pendulum payload dynamics", IEEE Transactions on Control Systems Technology, 2010.

[19] K. Kozak, W. Singhose, and I. Ebert-Uphoff, "Performance measures for input shaping and command generation", Journal of Dynamic Systems, Measurement, and Control, vol. 128, no. 3, p. 731, 2006.

[20] S. Sadr, S. A. A. Moosavian, and P. Zarafshan, "Dynamics modeling and control of a quadrotor with swing load", Journal of Robotics, vol. 2014, pp. 1–12, 2014.

[21] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Autonomous navigation and exploration of a quadrotor helicopter in gps-denied indoor environments", 2009.

[22] S. Mintchev, L. Daler, G. L'Eplattenier, L. Saint-Raymond, and D. Floreano, "Foldable and self-deployable pocket sized quadrotor", in 2015 IEEE International Conference on Robotics and Automation (ICRA 2015).

[23] O. Smith, Feedback control systems. McGraw-Hill Book Co., Inc., 1958.

[24] N. C. Singer and W. P. Seering, "Preshaping command inputs to reduce system vibration", Journal of Dynamic Systems, Measurement, and Control, vol. 112, no. 1, pp. 76–82, 1990.