Trajectory Generation and Controller Design for a Quadrotor-Slung Load System

Sean Fielding

The Department of Mechanical Engineering
McGill University, Montreal

November 2019

A thesis submitted to McGill University in partial fulfilment of the requirements for the degree of Master's of Engineering.

Abstract

Unmanned aerial vehicles (UAVs), in particular quadrotors, have received increased interest recently for a variety of applications including aerial photography and payload transportation. Unlike fixed wing aircraft, quadrotors are well suited to flying in crowded urban environments due to their ability to hover as well as takeoff and land vertically. These drones are inherently difficult to fly though and typically require skilled pilots or assistive controllers. This presents significant barriers for companies looking to use quadrotors for novel applications. In response to this limitation, there has been a significant focus on developing flight controllers and flight trajectory generators to enable autonomous operation of quadrotor drones.

While aerial photography applications are already well established for quadrotors, autonomous payload delivery presents an ongoing challenge. Currently, parcel delivery with quadrotors typically requires mounting a rigid container to the underside of the vehicle and filling this container with items. This configuration limits the size and shape of objects that can be transported, adds weight to the vehicle and adversely affects its attitude dynamics. A compelling alternative, inspired from military helicopters, is to suspend the payload from the underside of the quadrotor via one or more cables. Doing so saves weight and significantly reduces the impact that the payload has on the vehicle's attitude dynamics. The resulting quadrotor-slung load system also offers versatility in terms of the size and shape of payload that can be transported

The deployment of autonomous quadrotor-slung load systems presents numerous challenges though. A slung load will inherently tend to swing underneath the drone when acted upon by external disturbances like wind gusts. This can lead to instability in the system and potentially cause the drone to crash. It is thus critical to develop robust flight controllers for quadrotor-slung load systems that can autonomously track prescribed delivery routes while suppressing the swinging motion of the payload. Significant swinging motion can also be induced in the slung load as a result of the flight trajectory of the quadrotor drone. We should thus specifically design flight trajectories that help to prevent swinging motion from being induced.

At its core, this thesis seeks to develop techniques to both prevent and actively suppress swinging motion for an autonomous slung load system. In the following chapters we begin by exploring how quadrotor flight trajectories affect the swinging motion of a slung payload. In particular we focus on the technique of input shaping desired flight trajectories to help prevent induced payload swinging. We demonstrate how this concept can be extended to non-rest to rest flight trajectories and develop a computationally simple algorithm for generating these flight trajectories onboard a quadrotor as it flies. We next develop a flight controller to enable an autonomous quadrotor-slung load system to track these prescribed input-shaped flight trajectories. We explore existing designs for such controllers and develop a novel approach that distinguishes between swinging motion caused by tracking a prescribed flight trajectory and that caused by external disturbances. We demonstrate in simulation how this proposed controller is able to manage

swinging disturbances while tracking an input-shaped flight trajectory better than two baseline controller designs. We later demonstrate how the underlying concepts behind this controller can be used to develop a model predictive controller for tracking input-shaped trajectories with a quadrotor-slung load system. We also discuss the implementation of the proposed trajectory generator and controller onto an actual quadrotor drone and some of the challenges faced during this process. We conclude by making recommendations for future work for this project.

Résumé

Les véhicules aériens autonomes, en particulier les quadricoptères, sont de plus en plus utilisés pour une grande variété de tâches tels que la photographie aérienne et le transport de charges utiles. Comparativement aux drones à voilure fixe, les quadricoptères peuvent atterrir et décoller verticalement ainsi que maintenir un vol stationnaire. Ceci leur permet d'opérer dans des environnements encombrés. La commande de ces drones peut présenter de nombreux défis et donc nécessite souvent un pilote expérimenté ou l'aide d'un contrôleur. Ceci rends plus difficile l'intégration de ces machines au sein d'entreprises qui aimeraient les utiliser pour de nouvelles applications. Il est donc important de développer des algorithmes pour permettre plus d'autonomie dans ces systèmes.

Le transport de charges utiles avec des quadricoptères autonomes présente un défi important que plusieurs compagnies aimeraient surmonter. À présent, plusieurs prototypes commerciaux utilisent un contenant rigide attaché au drone pour tenir la charge utile. Ceci ajoute un poids supplémentaire à notre système et limite la grandeur d'objets qu'on peut transporter. De plus, il rend plus difficile la tâche de réorienter le quadricoptère en vol. Il serait donc important d'explorer si d'autres options pour transporter des charges utiles pourraient être utilisées. Dans le domaine militaire par exemple les hélicoptères transportent souvent des charges suspendues par des câbles. Ceci permet de diminuer le poids du système et réduit l'impact de la charge utile sur la capacité du véhicule de se réorienter. Nous allons donc étudier le transport d'une charge suspendue par un quadricoptère autonome.

Le transport de charges suspendues présente de nombreux défis. Par exemple, la charge aura tendance à osciller sous l'effet du vent. Ces mouvements peuvent déstabiliser le quadricoptère et entrainer un écrasement dangereux. Il est donc très important d'atténuer les oscillations de notre charge suspendue avec le contrôleur de notre drone. En même temps, ce contrôleur doit bien suivre une trajectoire prescrite pour compléter une livraison. Nous devons aussi tenir compte du fait que des mouvements oscillants peuvent être causés par le trajet de vol du quadricoptère. Il est donc important de choisir des trajectoires qui n'entraineront pas de grands mouvements oscillants.

Dans cette thèse, nous étudions des techniques pour éviter et atténuer les mouvements oscillants d'une charge suspendue transportée par un quadricoptère autonome. Tout d'abord, nous examinons comment calculer des trajectoires de vols qui nous permettront de livrer notre charge sans causer des mouvements oscillants dangereux. En particulier, nous utilisons une technique de modifications de commandes. Nous démontrons comment appliquer cette technique à une grande variété de trajectoires et nous présentons un algorithme pour calculer nos trajectoires de vol. Par la suite, nous développons un nouveau contrôleur pour permettre à notre système de suivre ces trajectoires. En particulier, notre contrôleur est capable de différencier entre un mouvement oscillant causé par le trajet de vol ou par des sources externes comme le vent. Nous démontrons avec des simulations que ce nouveau contrôleur est capable de mieux suivre une trajectoire de vol en atténuant des mouvements oscillants néfastes. Nous nous inspirons aussi de ce

nouveau contrôleur pour développer un contrôleur prédictif pour ce système. Par la suite, nous discutons les difficultés rencontrées pour faire des vols d'essai. Nous concluons avec des recommandations pour des travaux futurs.

Acknowledgements

Firstly, I'd like to thank my supervisor, Professor Meyer Nahon for all of his guidance and support throughout my graduate studies. Pursuing this degree has been a tremendous growth experience for me and I am immensely grateful for having had the opportunity to work in his research group. Our meetings together and his feedback throughout this project were instrumental in guiding me from the early conceptual phase through to this finished thesis. I so appreciated being able to learn about robotics and research from Professor Nahon.

I would also like to thank my friends and colleagues in the Aerospace Mechatronics Lab for all of their support over these last two years. I was very fortunate to be able to share a lab with so many talented researchers who were always willing to lend a hand. I particularly want to thank Eitan Bulka, Juan Carlos Hernández Ramírez, Fares El Tin, Walter Jothiraj, Corey Miles and Romain Chiappinelli for their assistance with implementing my work. Your experience and guidance was invaluable throughout this project and I really appreciated how you all took time to help me troubleshoot my system and brainstorm solutions to technical challenges. I would also like to thank Talha Tariq and Christian Patience for their tireless work setting up the motion capture test environment for me. I would not have been able to get my quadrotor off the ground without your assistance. A special thanks also to Siddharth Kumanduri, Shatil Rahman and Jad Wehbeh for your assistance with conducting flight tests of my system. It was also a pleasure getting to share a work space with Sahand Rezaei-Shoshtari, Kieran Ratcliffe, Hunter Song, Mikkel Jorgensen, Sam El Toufaili, Tim Thompson, Catherine Massé, Yunpeng Hu, Denis Kartachov and Professor Inna Sharf.

I would also like to extend a special thanks to my parents, Allan Fielding and Marilyn Alfano for all of their love and support throughout my graduate studies. You have always encouraged me to pursue my goals and I would not have been able to complete this degree without you by my side.

The work presented in this thesis was made possible with the financial support of the Canada Graduate Scholarships-Master's Award from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Bourse de maîtrise en recherche from the Fonds de recherche du Québec - Nature et technologies (FRQNT), the McGill Engineering Undergraduate Student Masters Award (MEUSMA) and the McGill Masters Top-Up award.

Dedicated to my Nanna and Grampy, I love you always

Contents

Abstract iii

Résumé v

Acknowledgements vii

Table of Contents ix

List of Figures xi

List of Tables xiii

1 Introduction 1
- 1.1 Objectives . . . 4
- 1.2 Thesis Organization . . . 5

2 Input-Shaped Trajectory Generation 7
- 2.1 Flight Trajectory Generation for Quadrotor-Slung Load Systems . . . 7
- 2.2 Input Shaping Theory . . . 8
  - 2.2.1 Types of Input Shapers . . . 10
  - 2.2.2 Input Shaping for Quadrotor-Slung Load Systems . . . 17
  - 2.2.3 Implementation . . . 18
- 2.3 Non-Rest to Rest Input-Shaped Trajectory Generation . . . 21
  - 2.3.1 Generating an Unshaped Flight Trajectory . . . 24
  - 2.3.2 Summary of Trajectory Generator . . . 26
  - 2.3.3 Simulation Results . . . 28
- 2.4 Natural Frequency of a Quadrotor-Slung Load System . . . 34

3 Controller Design 41
- 3.1 Survey of Quadrotor-Slung Load Flight Controller Designs . . . 41
  - 3.1.1 Payload Trajectory Following Controllers . . . 42
  - 3.1.2 Quadrotor Trajectory Following Controllers . . . 43
  - 3.1.3 Quadrotor Trajectory Following + Payload Swing Feedback Controllers . . . 45
- 3.2 Quadrotor-Slung Load System Simulation Model . . . 47
  - 3.2.1 Simulator setup . . . 51

3.3 Geometric Controller . . . 52
3.3.1 Oscillation Controller . . . 54
3.3.1.1 Swing Prediction . . . 55
3.3.2 Position Controller . . . 57
3.3.3 Attitude Controller . . . 58
3.3.4 Thrust Allocation and Saturation . . . 60
3.4 Controller Evaluation . . . 62
3.4.1 Controller Tuning Process . . . 62
3.4.2 Baseline Flight Controllers . . . 66
3.4.3 Simulation Results . . . 67
3.4.4 Cable Length and Payload Mass Changes . . . 75

4 Model Predictive Controller Design **81**
4.1 Model Predictive Control for Quadrotor-Slung Load Systems . . . 81
4.2 Implementation . . . 83
4.2.1 Prediction Model . . . 84
4.2.2 Constraints and Cost Function . . . 86
4.2.3 Simulator Setup . . . 88
4.2.4 Controller Tuning . . . 89
4.3 Simulation Results . . . 90

5 Experimental Setup **101**
5.1 Trajectory Generator and Controller Implementation . . . 101
5.2 Test Setup . . . 104
5.3 Payload Detection . . . 105
5.4 Drone Characterization . . . 108
5.4.1 Propeller Characterization . . . 109
5.4.2 Motor Characterization . . . 110
5.5 Real-Time Implementation and Preliminary Flight Testing . . . 113

6 Conclusion **117**
6.1 Future Work . . . 118

A Differential Flatness of a Quadrotor **121**

Bibliography **127**

List of Figures

1.1 Quadrotor-Slung Load System . . . 2
1.2 Hierarchical Quadrotor Controller Design . . . 4

2.1 Input-Shaped Step Response Example . . . 9
2.2 Destructive Interference Example . . . 10
2.3 PRV Versus Angular Velocity Error for Common Input Shapers . . . 15
2.4 PRV Versus Damping Ratio Error for Common Input Shapers . . . 16
2.5 Improper Input Shaping of Non-Rest to Rest Trajectory . . . 21
2.6 Proper Input Shaping of Non-Rest to Rest Trajectory . . . 22
2.7 Rest to Rest Motion Trajectory Comparison . . . 31
2.8 Rest to Rest Motion Swing Comparison . . . 31
2.9 Non-Rest to Rest Motion Trajectory Comparison . . . 32
2.10 Non-Rest to Rest Motion Swing Comparison . . . 33
2.11 Non-Rest to Rest Motion with Design Error Swing Comparison . . . 34
2.12 Rigid Body Pendulum Model . . . 36
2.13 (Top) Comparison of Predicted T_d Values, (Bottom) Percent Error Between Point Mass and Rigid Body ω_2 . . . 38

3.1 Model Frame Setup and Nomenclature . . . 48
3.2 Proposed Flight Controller Overview . . . 54
3.3 Predicted Swinging Motion (blue) due to Prescribed Input-Shaped Flight Trajectory (red) . . . 56
3.4 Quadrotor Motor Naming Convention . . . 61
3.5 Simulation 1: Time History of Swing Angle . . . 68
3.6 Simulation 1: Time History of Quadrotor Trajectory Tracking Error Magnitude . . . 68
3.7 Simulation 2: Top View of Quadrotor Motion . . . 69
3.8 Simulation 2: Time History of Swing Angle . . . 70
3.9 Simulation 2: Time History of Quadrotor Trajectory Tracking Error Magnitude . . . 70
3.10 Simulation 2: Oscillation Feedback Force . . . 71
3.11 Simulation 3: Top View of Quadrotor Motion . . . 72
3.12 Simulation 3: Time History of Swing Angle . . . 73
3.13 Simulation 3: Time History of Quadrotor Trajectory Tracking Error Magnitude . . . 73
3.14 Simulation 3: Oscillation Feedback Force . . . 74

3.15 Simulation 3: Position Feedback and Feedforward Force 74
3.16 0.6m Cable: Time History of Swing Angle 77
3.17 1.35kg Payload: Time History of Swing Angle 79

4.1 Simulation 4: Time History of Swing Angle 91
4.2 Simulation 4: Time History of Quadrotor Trajectory Tracking Error Magnitude 91
4.3 Simulation 4: Time History of Motor Thrust Forces 92
4.4 Simulation 5: Top View of Quadrotor Motion 93
4.5 Simulation 5: Time History of Swing Angle 94
4.6 Simulation 5: Time History of Quadrotor Trajectory Tracking Error Magnitude 94
4.7 Simulation 5: Time History of Motor Thrust Forces 95
4.8 Simulation 6: Top View of Quadrotor Motion 96
4.9 Simulation 6: Time History of Swing Angle 96
4.10 Simulation 6: Time History of Quadrotor Trajectory Tracking Error Magnitude 97
4.11 Simulation 6: Time History of Motor Thrust Forces 97

5.1 Modified AscTec Pelican used for Flight Testing 105
5.2 VICON Motion Capture Flight Test Setup 105
5.3 Payload Tracking with Camera Nomenclature 106
5.4 Thrust Application Process 109
5.5 Thrust Versus RPM^2 for APC 10×4.7 Propeller 110
5.6 Moment Versus RPM^2 for APC 10×4.7 Propeller 111
5.7 PWM to RPM Experimental Setup 111
5.8 PWM to RPM Experimental Setup 113
5.9 VICON Communication Setup 115

A.1 Decomposition of Transformation from Inertial Frame to the Quadrotor Frame 122

List of Tables

3.1 Properties of the Pelican Quadrotor and Slung Load. 63
3.2 Controller Gains for Simulated Pelican Quadrotor and Slung Load. . . . 65
3.3 Cable Length Analysis 76
3.4 Mass Robustness Analysis 78

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs), in particular quadrotors have received increased interest recently for a variety of applications including aerial photography, inspection and payload transportation. A major advantage of quadrotors compared to fixed wing UAVs is that, like helicopters, they can hover as well as takeoff and land vertically. Compared to helicopters though, quadrotors are significantly less complex mechanically with fewer moving parts in their rotor assemblies [1]. Overall, quadrotor drones are particularly well suited to operation in dense urban environments or areas otherwise lacking unobstructed flat surfaces to use as a runway. A quadrotor is an inherently unstable underactuated system and as such can be difficult to fly. As a result, skilled operators are typically required to operate these vehicles and many consumer-grade systems provide an assistive control structure to simplify the flying process. Increasingly, there is a push towards automating the flight operations of quadrotors so that they can complete missions with minimal human oversight. Doing so will enable quadrotor drones to be better used in disaster relief and other emergency situations where manpower is a vital resource and timely information is key to decision making processes. Autonomous quadrotors also have the potential to be used for commercial payload transportation applications and indeed numerous companies are exploring their use for short-distance deliveries. A major requirement for achieving autonomous operation of quadrotors is the development of robust flight controllers and path planning algorithms. Both these topics have received extensive focus for many years within academic research communities.

In this thesis, we narrow our focus to payload delivery applications for quadrotor drones. This represents a newer application that has seen limited commercialization thus far but has the potential to dramatically alter how goods are delivered. Current prototypes for achieving parcel delivery with quadrotors typically store payload in a container rigidly

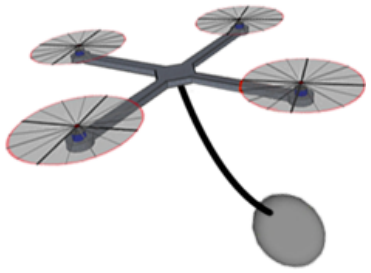

FIGURE 1.1: Quadrotor-Slung Load System

mounted to the underside of the vehicle. A major drawback of this approach is that it inherently limits the size and shape of objects that can be transported. The rigid container also adds weight to the drone thereby reducing its already limited payload carrying capacity. This mounting setup also increases the inertia of the quadrotor making it more difficult for the vehicle to change its attitude. The ability to quickly change attitudes is particularly critical for quadrotors since, like helicopters, they need to pitch and roll in order to accomplish horizontal maneuvers. An alternate mounting strategy is to use one or more cables to suspend the payload underneath the vehicle. The resulting quadrotor-slung load system, shown in Figure 1.1 reprinted from [2], is lightweight, simple to set up, and provides significant versatility in terms of the size and shape of object that can be transported. Slung load transportation has also been studied and implemented with manned helicopters, especially for military applications, for decades [3] [4] [5]. A key benefit of using a cable to mount a load onto a quadrotor is that this reduces the payload's impact on the vehicle's attitude dynamics compared to a rigid mounting setup [6] [7]. Another key benefit of slung load systems is that multiple quadrotors can be tethered to a common load. This means that payloads that would normally be too heavy for a single quadrotor to lift can still be transported without having to design a new quadrotor with increased carrying capacity. Collaborative load transportation is beyond the scope of this thesis but is increasingly being studied in academia such as in [8], [9] and [10].

Transporting a slung load, even in manned helicopter operations, presents numerous challenges. The slung load will inherently tend to swing underneath the vehicle due to external disturbances such as the wind, rotor downwash, collisions with the environment or even due to the motion of the vehicle. In helicopter-slung load systems, this swinging motion is particularly difficult to manage even for experienced pilots and can lead to crashes [5]. Swing management can also become critical during landing operations where there might be an increased risk of colliding with the environment leading to damage

or even cable entanglement. In [5], the author attempts to address these issues by developing a feedback controller to assist pilots by damping the swinging of the payload. For unmanned systems, we would need to extend this concept further by having robust flight controllers that can simultaneously manage flight trajectory tracking and swing suppression requirements.

Due to the potential dangers of swinging motion in slung load systems, it makes sense to look at potential strategies for preventing payload swinging. A good starting point is to consider how the motions of our quadrotor along its delivery route affect load oscillations. Quadrotor-slung load systems are inherently underactuated and thus lateral accelerations of the quadrotor will inevitably cause some form of swinging motion in the payload. In a realistic flight scenario, a quadrotor may have an assigned geometric flight path but could fly along that path using any number of trajectories involving different accelerations and velocities. Ultimately, we would like to ensure that whenever the quadrotor stops accelerating there is no relative swinging motion between it and its payload. To gain some intuition into this problem, we can conduct a simple experiment by moving a pendulum by hand. The goal is to move our hand a set distance and have no residual swinging in the pendulum at the end of the motion. With a bit of practice, one can quickly establish a successful motion strategy of selectively speeding up and slowing down the movement in time with the swinging of the pendulum. Ideally, we'd now like implement this same strategy in the design of flight trajectories for a quadrotor carrying a slung load. It turns out that we can do this by using input shaping. Input shaping is a process whereby a signal is modified such that it no longer will excite oscillations in a system at a prescribed target frequency. Here, we want to input shape flight trajectories for quadrotors transporting slung loads such that once the vehicle stops accelerating, the payload will not swing. Input shaping is based on exploiting destructive interference and has been demonstrated in numerous papers dealing with helicopter and quadrotor-slung load systems. Swing prevention strategies such as input shaping can have limited effectiveness for realistic systems due to measurement uncertainty though. Unpredictable external disturbances will also act on the payload and will need to be managed. For this reason, we typically will need to augment our system by incorporating active swing suppression as part of the flight controller design.

Flight controllers for quadrotors and quadrotor-slung load systems have been developed and demonstrated in numerous previous works. An excellent summary of the current state of the art is presented in [11]. These controllers typically take on the hierarchical form shown in [11], reprinted here as Figure 1.2. Figure 1.2, illustrates how the desired position and yaw motion of the quadrotor is converted into a desired attitude and thrust force

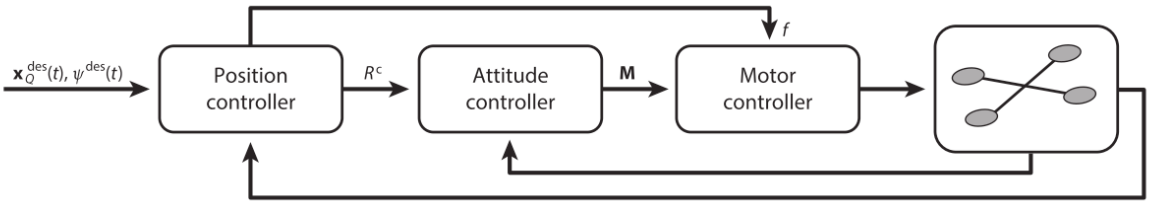

FIGURE 1.2: Hierarchical Quadrotor Controller Design

by a Position controller. The desired attitude is used to compute a desired moment by the Attitude controller and this resulting thrust and moment are then allocated to forces for each motor on the vehicle. This control strategy is motivated by the underactuated nature of the quadrotor where attitude changes are required to make positional changes. A conceptually similar design will also hold for quadrotor-slung load systems. In many cases, these controllers make use of the property of differential flatness for quadrotors and quadrotor-slung load system models. In essence, a differentially flat system is one in which the states and inputs can be defined as a function of a limited set of outputs and their higher derivatives. For a quadrotor for instance we find that the thrust, attitude and angular velocity can all be expressed as functions of the desired position, yaw and their higher derivatives. This idea will be discussed in more detail in Appendix A. In this thesis, we exploit differential flatness in our development of a flight controller for a quadrotor-slung load system.

A key limitation with quadrotor-slung load systems is that the only way to suppress payload swinging is through motions of the quadrotor. Thus during a flight trajectory tracking operation, suppressing swinging motion will almost inevitably result in larger tracking errors. These errors become more critical when tracking input-shaped flight trajectories though since we carefully design these trajectories in order to achieve swing prevention. We thus need to carefully balance when our controller makes swing suppression motions in order to achieve the full benefit of our input-shaped trajectory. This can be challenging though since input-shaped motion trajectories will induce some swinging motion in the payload during accelerating portions but will ultimately help eliminate residual swinging motion.

1.1 Objectives

The goal of this thesis is to improve the autonomy of quadrotor-slung load systems by developing techniques to prevent and actively suppress swinging in a suspended payload.

For swing prevention, we focus on developing quadrotor flight trajectories that inherently avoid inducing swinging motion in the slung load through the use of input shaping. In this work we develop a method to extend the concept of input shaping to generate non-rest to rest flight trajectories while maintaining a computationally simple framework that can readily be implemented on a quadrotor drone with limited computational resources. We next develop a flight controller for a quadrotor-slung load system that can effectively track this input-shaped flight trajectory while also actively suppressing swinging disturbances. Unlike previous controllers presented in the literature, in this thesis we develop a novel formulation that attempts to distinguish between swinging motion caused by the accelerations inherent to a prescribed flight trajectory and those caused by external disturbances. Once this distinction is made, our controller only attempts to suppress swinging motion due to external disturbances. We compare this proposed controller in simulation to two simpler designs to assess their ability to track input-shaped flight trajectories both with and without external disturbances acting on the payload.

1.2 Thesis Organization

This thesis has the following structure: Chapter 2 presents a literature review of trajectory generation and input shaping for quadrotor-slung load systems as well as the development of a computationally simple input-shaped trajectory generator for non-rest to rest motions. Chapter 3 presents a literature review of control strategies for quadrotor-slung load systems and subsequently discusses the development of our novel controller. Simulation results are also presented comparing our controller to baseline designs. Chapters 2 and 3 are based on work presented in [2]. Chapter 4 presents the development of a model predictive controller for tracking input-shaped flight trajectories inspired by the controller developed in Chapter 3. These two controllers are then compared in simulation. Chapter 5 discusses the implementation process of taking our trajectory generator and controller from Chapters 2 and 3 and implementing them on an actual quadrotor drone. Chapter 6 provides a conclusion and discusses future work.

Chapter 2

Input-Shaped Trajectory Generation

This chapter briefly introduces the problem of generating flight trajectories for quadrotor-slung load systems. We then introduce the technique of input shaping and subsequently present a novel formulation for how to extend this concept to non-rest to rest flight trajectories.

2.1 Flight Trajectory Generation for Quadrotor-Slung Load Systems

A key requirement for autonomous payload delivery systems with quadrotor drones is developing a robust method to autonomously plan flight routes for the drone. As discussed in [11], numerous papers have demonstrated the capability for quadrotor drones to autonomously navigate through previously unknown environments at high speeds. However, extending these capabilities to quadrotor-slung load systems remains an ongoing challenge.

A key distinction that appears in the literature is whether papers focus on generating desired motion trajectories for the payload or for the quadrotor drone itself. In some sense, the decision between which strategy to employ comes down to a matter of perspective; is our payload a carriage being drawn by a quadrotor or is our drone towing a suspended load. In [12], the authors formulate the quadrotor flight trajectory problem as a mathematical program with complementarity constraints subject to the full system dynamics of the quadrotor-slung load system. The resulting optimization problem can accommodate constraints on the payload's motion, control inputs, as well as obstacle

avoidance. The authors also demonstrate how their algorithm can be applied to waypoint navigation and payload throwing tasks and inherently generates flight trajectories where the cable remains taut. An alternate approach focusing on payload trajectories is developed in [13]. The problem is formulated as a mixed integer quadratic program with a goal of navigating a prescribed set of waypoints and avoiding collisions between the environment and any part of the slung load system. The algorithm is meant to allow the cable to go slack during portions of the motion trajectory. Both these methods are computationally intensive though and require global information about the environment.

There have also been numerous papers that attempt to generate swing-free maneuvers for slung load systems such as in [14] and [15]. In [14], dynamic programming is used to generate a swing-free quadrotor flight trajectory. Dynamic programming essentially involves recursively solving for an optimal motion that will minimize residual swinging. Their method showed promising swing mitigation results in simulation but the experimental results suffered a bit due to trajectory tracking difficulties. In [15], approximate value iteration reinforcement learning is used to develop a system that can generate flight trajectories between waypoints that minimize residual swinging motion of the payload. The resulting learned trajectory outperformed cubic polynomial flight trajectories and achieved similar results to a dynamic programming approach for swing mitigation.

Ultimately, though the trajectory generation methods in [12], [13] and [14] rely on solving an optimization problem to generate flight trajectories. The trajectory generation method in [15] on the other hand requires training the system for its payload first. These requirements are problematic since ultimately our goal is to generate flight trajectories that will mitigate swinging in our slung load onboard our quadrotor drone in real time. Input shaping offers a compelling alternative strategy that is computationally simple to implement and helps to manage swinging motion in a slung load system.

2.2 Input Shaping Theory

By itself, a quadrotor drone is inherently underactuated with four control inputs and six degrees of freedom. The addition of a slung load to this system introduces at least two more degrees of freedom, further solidifying the underactuated nature of the resulting quadrotor-slung load system. From a practical standpoint, this means that lateral accelerating maneuvers of the quadrotor such as speeding up, slowing down or performing a turn at speed will inevitably induce some level of relative swinging motion between the payload and the drone. Even when the drone stops accelerating, this induced swinging

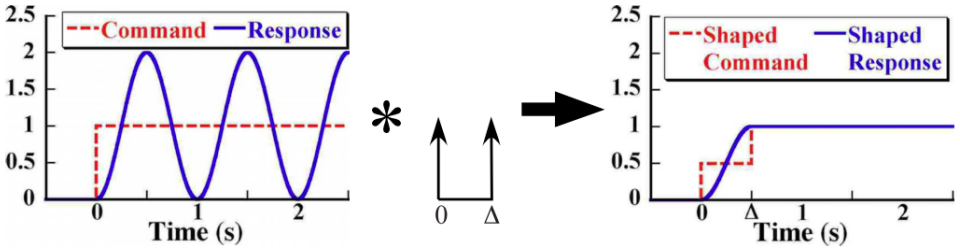

FIGURE 2.1: Input-Shaped Step Response Example

motion can persist, leading to instability in the system. The best case scenario would be that this swinging motion no longer occurs once the quadrotor stops accelerating and either comes to rest or reaches a new steady state velocity. Input shaping is specifically meant to achieve this best case scenario.

Input shaping is a process whereby a reference signal given to a flexible system is convolved with a sequence of impulses, known as an input shaper, in order to cancel the system's vibration modes [16]. Early derivations of the concept such as that presented in [17] are based on first assessing the oscillations of a linear second order system due to a sequence of impulses. The amplitudes and timing of these impulses could then be optimized to find the shortest set of impulses that would result in zero residual swinging motion of the system. A critical observation discussed in [17] is that this same vibration reduction can be achieved by convolving any input with this sequence of impulses to obtain a shaped input. Furthermore, by normalizing the impulses to sum to a value of one, we can guarantee that the convolution process will not cause the shaped input to exceed the value of the original input. An early example of this concept called the Posicast input was developed in 1958 [17].

An example of the input shaping process for a simple step input signal is given in [18] reprinted here as Figure 2.1. As demonstrated in Figure 2.1, the original step input command induces an undesirable oscillating response in the system. After convolving the original step command with an input shaper and applying the resulting shaped command to the system, the shaped response now shows no residual oscillation. As will be shown, the amplitudes and timing of the impulses in the input shaper are based on the properties of the oscillating system. A typical application of input shaping is to modify the desired motion trajectory of an actuated body connected via cable to a passive body. For example, input-shaped motion trajectories have been employed to mitigate swinging in gantry cranes [19] as well as helicopter [20] [21] [22] [23] and quadrotor-slung load systems [10] [24] [25] [26]. In [10], input shaping is even applied to collaborative load transport using multiple aerial vehicles tethered to a common payload.

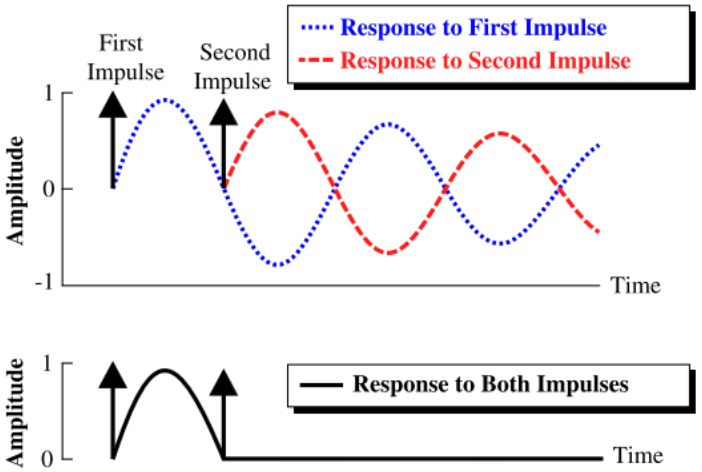

FIGURE 2.2: Destructive Interference Example

At its most basic level, input shaping works by exploiting destructive interference between impulse responses for an oscillatory system. This is demonstrated visually in an example presented in [23], reprinted here as Figure 2.2. The example shown in Figure 2.2 involves only two impulses however similar results can be obtained with three or four impulses. Like a notch filter, input shaping is specifically designed to target and suppress swinging at a chosen frequency. An alternative form that behaves more like a low pass filter for swing mitigation using an embedded prefilter is presented in [27]. In the following sections we will provide an overview of some of the various types of input shapers presented in the literature as well as discussing how the convolution process shown in Figure 2.1 is implemented mathematically. We will then examine the current state of the art for input-shaped flight trajectories in quadrotor-slung load systems in order to motive our subsequent work.

2.2.1 Types of Input Shapers

Numerous different types of input shapers with varying numbers of impulses have been presented. At first glance this might seem odd since from Figures 2.1 and 2.2 two impulses appear to be sufficient to obtain the desired oscillation mitigation. The motivation behind creating more complicated input shapers with additional amplitudes is to improve robustness. Figures 2.1 and 2.2 present an idealized scenario where the natural frequency and damping of the oscillatory system are known perfectly and an input shaper is designed based on this knowledge. In reality, these properties will be estimated based on models or experimental data and this can in turn introduce error, degrading the effectiveness of the input-shaped trajectory for mitigating oscillations in the shaped response. Consider

for instance the extreme case where instead of applying a second impulse after one half period of oscillation, as done in Figure 2.2, we applied the second impulse after a full period of oscillation. In this situation, the input-shaped trajectory would amplify the oscillations of the system. As will be shown, adding additional impulses to the input shaper allows for greater errors between the true system natural frequency and damping and the values used to design the shaper while maintaining oscillation mitigation performance. The main drawback of using additional impulses is that this lengthens the duration of the input shaper and slows down the reference trajectory. In Figure 2.1 for instance, the unshaped step command reaches its maximum value instantaneously at time 0. After shaping with an input shaper of duration Δ s though, the resulting shaped command now only reaches the same maximum value after Δ s. As will be shown, the value of Δ is proportional to the system's period of oscillation. From a practical standpoint, this means that oscillating systems with large periods such as slung loads attached to long cables will require input shapers with longer durations resulting in slower shaped commands. It is thus important to balance robustness to uncertainty with shaper duration in selecting an input shaper to use for a given problem.

The derivation of the amplitudes and timing of an input shaper's impulses is based on the impulse response of a basic second order system with undamped natural frequency ω and damping ratio ζ. From [16] and [24], the amplitude of oscillation A_Σ of an underdamped second-order system due to a sequence of n impulses $(A_1, A_2, ..., A_n)$ at times $(0, t_2, ..., t_n)$ is given by

$$A_\Sigma = \frac{\omega}{\sqrt{1-\zeta^2}} \mathrm{e}^{-\zeta\omega t_n} \sqrt{\left(\sum_{i=1}^{n} A_i \mathrm{e}^{\zeta\omega t_i} \cos\left(\omega t_i \sqrt{1-\zeta^2}\right)\right)^2 + \left(\sum_{i=1}^{n} A_i \mathrm{e}^{\zeta\omega t_i} \sin\left(\omega t_i \sqrt{1-\zeta^2}\right)\right)^2} \tag{2.1}$$

From equation (2.1) we also have that for a single unit-magnitude impulse at time zero the resulting amplitude of residual oscillation $A_\uparrow$ would be

$$A_\uparrow = \frac{\omega}{\sqrt{1-\zeta^2}} \tag{2.2}$$

The value of A_Σ can be divided by $A_\uparrow$ in order to obtain the Percentage Residual Vibration (PRV) as done in [16] and [18].

$$\mathrm{PRV} \doteq \mathrm{e}^{-\zeta\omega t_n} \sqrt{\left(\sum_{i=1}^{n} A_i \mathrm{e}^{\zeta\omega t_i} \cos\left(\omega t_i \sqrt{1-\zeta^2}\right)\right)^2 + \left(\sum_{i=1}^{n} A_i \mathrm{e}^{\zeta\omega t_i} \sin\left(\omega t_i \sqrt{1-\zeta^2}\right)\right)^2} \tag{2.3}$$

The PRV essentially tells us how effective the input shaper is at eliminating residual oscillations in our second order system. When PRV = 0, this corresponds to the ideal scenario where our sequence of impulses completely avoid exciting residual oscillations in the system after the final impulse. We can derive the value of our input shaper's impulses by imposing constraints on equation (2.3). For example, consider a two impulse input shaper with impulses (A_1, A_2) at times $(0, t_2)$. We would like to impose the constraint that PRV = 0 and solve equation (2.3) for A_1, A_2, t_2. From Figure 2.2 though we expect a solution to exist if we impose that the second impulse A_2 occur after one half period. That is $t_2 = T_d/2$, where T_d is the damped period of oscillation of our second order system.

$$T_d = \frac{2\pi}{\omega_{IS}\sqrt{1-\zeta_{IS}^2}} \tag{2.4}$$

In equation (2.4), ω_{IS} and ζ_{IS} are the estimated values of the true undamped natural frequency and damping ratio of our second order system used to compute the parameters of our input shaper. Under ideal circumstances, these values would match perfectly with the system's true values of ω and ζ that appear in equations (2.1), (2.2) and (2.3). If we assume a perfect scenario where $\omega_{IS} = \omega$ and $\zeta_{IS} = \zeta$ and substitute $t_1 = 0$ and $t_2 = T_d/2$ into equation (2.3), with $n = 2$ we obtain the following result.

$$\text{PRV} = 0 = \mathrm{e}^{\frac{-\zeta_{IS}\pi}{\sqrt{1-\zeta_{IS}^2}}}\sqrt{\left(A_1 - A_2\mathrm{e}^{\frac{\zeta_{IS}\pi}{\sqrt{1-\zeta_{IS}^2}}}\right)^2} \tag{2.5}$$

We can further impose the constraint that $A_1 + A_2 = 1$ and $A_1 > 0$, $A_2 > 0$. This first constraint ensures that the input shaping process will not alter the overall amplitude of our reference signal. We thus find that

$$A_1 = \frac{1}{1+\mathrm{e}^{\frac{-\zeta_{IS}\pi}{\sqrt{1-\zeta_{IS}^2}}}} \quad , \quad A_2 = \frac{\mathrm{e}^{\frac{-\zeta_{IS}\pi}{\sqrt{1-\zeta_{IS}^2}}}}{1+\mathrm{e}^{\frac{-\zeta_{IS}\pi}{\sqrt{1-\zeta_{IS}^2}}}} \tag{2.6}$$

Notice for example if we have an undamped system with $\zeta = 0$ that we obtain $A_1 = A_2 = 0.5$ and $t_1 = 0$, $t_2 = T_d/2$ corresponding to a situation with two impulses of equal amplitude spaced out by one half period. In the above process we have essentially derived the values for the simplest type of input shaper, the Zero Vibration (ZV) input shaper [16] [24]. This derivation can be traced back to earlier works such as [17]. Throughout

this paper we will denote this input shaper as $h_{ZV}[t]$ where

$$h_{ZV}[t] = \begin{cases} \frac{1}{1+K}, & \text{if } t = 0 \\ \frac{K}{1+K}, & \text{if } t = \frac{T_d}{2} \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

$$K \doteq \mathrm{e}^{\frac{-\zeta_{IS}\pi}{\sqrt{1-\zeta_{IS}^2}}} \tag{2.8}$$

where the value of T_d is obtained from equation (2.4). A basic ZV input shaper can produce favourable swing mitigation if ω_{IS} and ζ_{IS} in equations (2.7) and (2.8) perfectly match the true values ω and ζ for our system.

We can apply a similar technique to derive input shapers with more than two impulses. Suppose that we imposed as constraints that PRV $= 0$ and $\partial\text{PRV}/\partial\omega = 0$ for a three impulse system. Solving the resulting system would yield the values for the Zero Vibration and Derivative (ZVD) input shaper $h_{ZVD}[t]$ as done in [16].

$$h_{ZVD}[t] = \begin{cases} \frac{1}{1+2K+K^2}, & \text{if } t = 0 \\ \frac{2K}{1+2K+K^2}, & \text{if } t = \frac{T_d}{2} \\ \frac{K^2}{1+2K+K^2}, & \text{if } t = T_d \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

where the value of K comes from equation (2.8). Notice though that compared to $h_{ZV}[t]$, $h_{ZVD}[t]$ lasts twice as long with its final impulse occurring at $t = T_d$. We can continue in a similar manner to derive the Zero Vibration and Double Derivative (ZVDD) and Zero Vibration and Triple Derivative (ZVDDD) input shapers. The resulting amplitude values are presented in [10] and [16].

A separate category of input shapers can be derived by forcing PRV $= 0$ at frequencies above and below the natural frequency of the system [28]. The resulting Extra Insensitive (EI) input shapers are designed with a chosen tolerable PRV level V_{tol} and attempt to maintain the PRV below this level for as wide a range of natural frequencies as possible. This concept can also be extended to multi-hump EI shapers as done in [28]. One such example is the the two-hump EI input shaper. The main benefit of these types of input shapers is that they provide added robustness to model uncertainty compared to ZV shapers while using the same number of impulses. From [16], for an undamped system

the EI and two-hump EI shapers $h_{EI}[t]$ and $h_{2EI}[t]$ have the following forms.

$$h_{EI}[t] = \begin{cases} \frac{1+V_{tol}}{4}, & \text{if } t = 0 \\ \frac{1-V_{tol}}{2}, & \text{if } t = \frac{T_d}{2} \\ \frac{1+V_{tol}}{4}, & \text{if } t = T_d \\ 0, & \text{otherwise} \end{cases} \tag{2.10}$$

$$h_{2EI}[t] = \begin{cases} A_{12H}, & \text{if } t = 0 \\ \frac{1}{2} - A_{12H}, & \text{if } t = \frac{T_d}{2} \\ \frac{1}{2} - A_{12H}, & \text{if } t = T_d \\ A_{12H}, & \text{if } t = \frac{3T_d}{2} \\ 0, & \text{otherwise} \end{cases} \tag{2.11}$$

$$A_{12H} \doteq \frac{3X^2 + 2X + 3V_{tol}^2}{16X} \quad , \quad X \doteq \sqrt[3]{V_{tol}^2\left(\sqrt{1 - V_{tol}^2} + 1\right)} \tag{2.12}$$

Throughout this work, unless otherwise indicated we will chose the value $V_{tol} = 0.05$ corresponding to a five percent allowable PRV for our EI and two-hump EI input shapers. Our choice of V_{tol} was based on the value used in [16].

A useful way to understand the benefit of one shaper compared to another is to assess their robustness to errors between ω_{IS} and the true ω value for the second order system under consideration. A more robust input shaper will have a lower PRV value for larger errors. From a practical standpoint this means that our input-shaped motion trajectory will still mitigate swinging even if we design it for the wrong undamped natural frequency. A standard graph that is typically presented in the literature is a plot of PRV versus ω/ω_{IS}. In Figure 2.3 the PRV is computed using equation (2.3) for a second order system with $\omega = 0.5\text{rad}\,\text{s}^{-1}$ and $\zeta = \zeta_{IS} = 0$ for a ZV, ZVD, ZVDD, EI and two-hump EI input shaper. Both the EI and two-hump EI shapers were designed for $V_{tol} = 0.05$ corresponding to a five percent allowable PRV value shown in the plot. Figure 2.3 illustrates the tradeoff that is present when selecting an input shaper with additional impulses. The ZV input shaper for instance will have good vibration mitigation performance when ω_{IS} closely matches the true ω value for our system but this performance quickly degrades as this error increases. The sharp "v" shape of the ZV shapers curve is due to the fact that we only derived it with a PRV $= 0$ constraint. Adding the $\partial\text{PRV}/\partial\omega = 0$ constraint for the ZVD shaper causes the its curve to take on a more gradual "u" shape in figure 2.3. We in turn can observe how adding this constraint causes the PRV value for the ZVD shaper

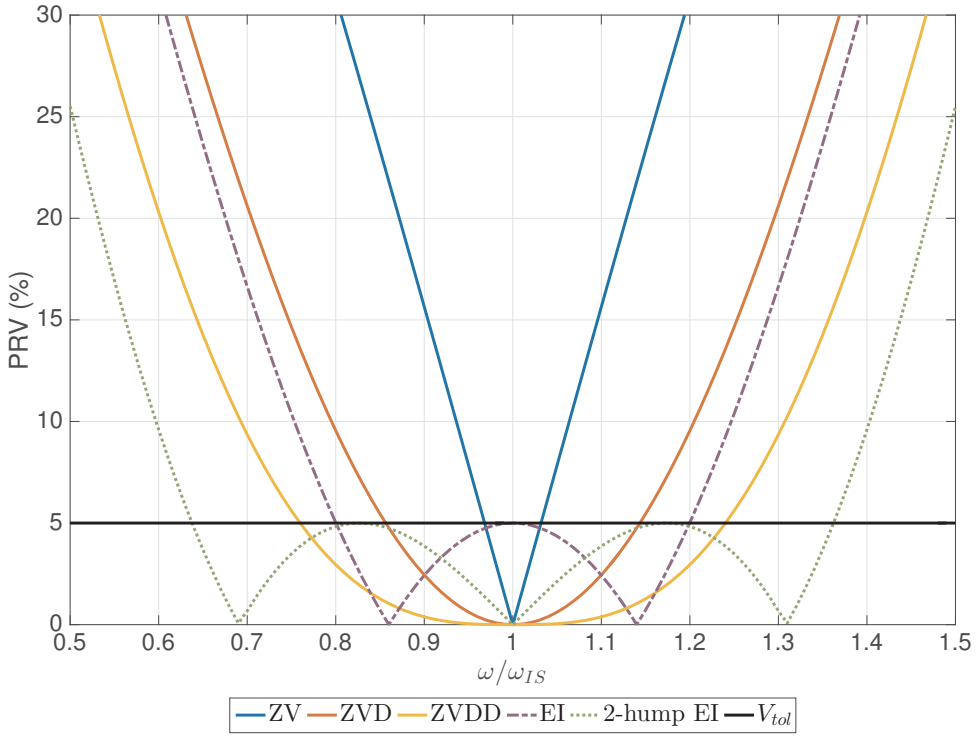

FIGURE 2.3: PRV Versus Angular Velocity Error for Common Input Shapers

to increase much more gradually than the ZV shaper and the ZVDD shaper increases even more gradually. This improved robustness comes at a cost of a longer input shaper with more impulses which will extend the duration of our shaped trajectory as previously discussed. In Figure 2.3 notice also how the performance of the EI and two-hump EI shapers differ from the ZV, ZVD and ZVDD shapers. The EI shaper in particular does not achieve a 0 PRV value when its design angular velocity ω_{IS} perfectly matches the true system value ω. Instead, its PRV value starts at the prescribed tolerance level and remains below this level for a wide range of errors. Notice that from equations (2.9) and (2.10) both the ZVD and EI input shapers have the same number of impulses with the same overall duration. In Figure 2.3 though we can see that the EI shaper stays below our tolerance PRV level over a wider range of error in the design angular velocity than the ZVD shaper. Similarly, the ZVDD shaper and the two-hump EI shaper both have the same number of impulses and duration but the two-hump EI shaper demonstrates improved robustness to error compared to the ZVDD shaper. The two-hump EI shaper gets its name from the distinctive behaviour that it presents in this graph with humps reaching the prescribed tolerance level at error values around 0.8 and 1.2. An advantage of the two-hump EI shaper compared to the EI shaper is that it achieves a PRV value of 0 when ω_{IS} perfectly matches the true ω.

We can also create a similar plot looking at errors in ζ_{IS} versus the true ζ value for our

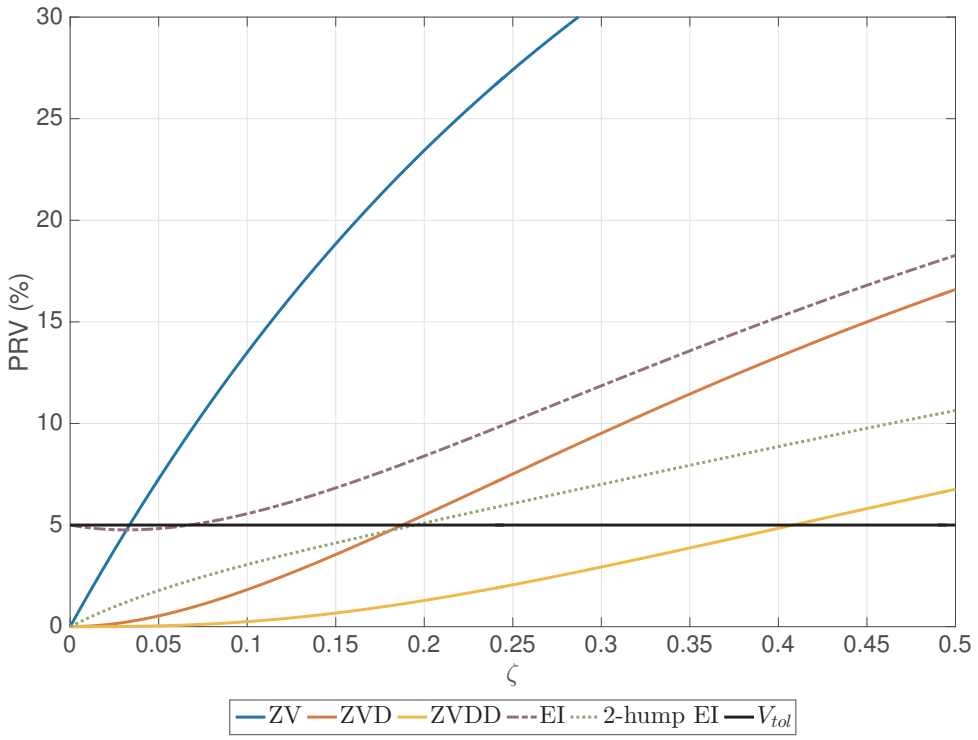

FIGURE 2.4: PRV Versus Damping Ratio Error for Common Input Shapers

system. In Figure 2.4 we again compute the PRV value for our input shapers but this time we set $\omega_{IS} = \omega = 0.5\text{rad}\,\text{s}^{-1}$, $\zeta_{IS} = 0$ and we vary the true value of ζ for our second order system. In Figure 2.4 we can see that the ZVDD shaper maintains a PRV value below the chosen tolerance of five percent for a wide range of true damping ratios despite being designed for $\zeta_{IS} = 0$. The two-hump EI shaper manages to maintain a PRV below the prescribed tolerance for up to around $\zeta = 0.2$ and overall appears to have slightly better performance than the ZVD shaper throughout the range considered. Once again, the ZV input shaper demonstrates poor robustness to error. Overall, by using Figures 2.3 and 2.4 we can make an informed design decision in terms of which shaper to apply to a given design problem based on how accurately we know our system's design parameters.

Additional types of input shapers are also found in the literature. Specified Insensitivity (SI) shapers for instance can be created with PRV versus ω/ω_{IS} curves that are not symmetric about 1 like the curves shown in Figure 2.3. This form of input shaper offers the greatest possible robustness but at the cost of requiring the user to solve an optimization problem to obtain the input shaper's amplitudes and timing as opposed to the closed forms for other shapers [16]. Another class of input shapers can be developed by allowing certain impulses to have negative magnitudes as discussed in [18]. In this thesis however, we will focus primarily on the input shapers presented in Figures 2.3 and 2.4 due to their

simple closed form equations and the fact that they appear to be the most commonly used shapers.

2.2.2 Input Shaping for Quadrotor-Slung Load Systems

A potential concern with implementing input shaping for a quadrotor-slung load system is the fact that our derivation of input shapers is based on impulse response characteristics for linear second order systems. In reality, a quadrotor-slung load system will exhibit nonlinearity, which brings into question how effective the technique will be at swing mitigation. In [29], the authors assert that robust input shapers such as the ZVD and EI shapers will work well even for systems with "moderate nonlinearities". The authors also discuss how nonlinearities can be dealt with by adaptive input shapers which employ sensor feedback in order to tune the shaper parameters online [29]. In [29] the authors further explored how well ZV and ZVD shapers would perform for systems with changing natural frequencies due to nonlinearities. They found that, as with a linear system, the ZVD shaper generally had lower PVR than the ZV shaper. Based on this result, they argue that a shaper that is designed to be robust to errors in the system natural frequency of a linear system, would naturally be robust in changes of the system behaviour due to nonlinearity, i.e., that "the robustness of a shaper in the linear domain transfers into robustness for nonlinear systems" [29]. These results help to alleviate the concern with applying input shaping to swing management in quadrotor-slung load systems.

In surveying the existing literature on the use of input shaping in quadrotor-slung load systems there are a few key limitations that appear. A major limitation is that input shaping is typically only applied to rest to rest maneuvers such as in [10] [20] [21] [22] [24] [25] [30]. The concept of an input-shaped trajectory for going from rest to a forward velocity was discussed in [20], but ultimately the paper only explored rest to rest motions. These papers also did not place significant emphasis on performing the input shaping process onboard the drone in real time. Generating non-rest to rest input-shaped flight trajectories onboard a quadrotor drone as it flies could offer significant advantages. In a realistic flight scenario for instance, a quadrotor carrying a slung load may be following a predefined delivery route and encounter an unexpected obstruction. In this scenario, it would be advantageous to be able to generate a non-rest to rest input-shaped flight trajectory that would link the quadrotor's current state to a desired end state. The resulting motion could be used to turn the drone around, have the drone come to rest or even have it deviate laterally while maintaining the same overall heading and flight speed.

In any case, the drone would need to accelerate and input shaping could help mitigate residual swinging caused by these accelerations.

In order to extend the concept of input shaping to non-rest to rest maneuvers and implement the method in real time, we must first examine more closely how the convolution step is evaluated mathematically.

2.2.3 Implementation

This section focuses on the details of how to properly implement the convolution shown in Figure 2.1 for any input signal in discrete time. The motivation behind examining this process in more detail is that we would like to be able to generate input-shaped flight trajectories onboard our drone and in real time in response to new target waypoints being generated by high level navigation algorithms.

Implementing input shaping in discrete time is summarized in [24] and [31]. Along the x axis, suppose that we are given an unshaped flight trajectory $x[t]$ defining the desired motion of our quadrotor over a closed time interval $t \in [0, T_U]$. In this example $x[0]$, $\dot{x}[0]$ ... correspond to the current state of the quadrotor and $x[T_U]$, $\dot{x}[T_U]$ ... would be the desired state for the drone to reach along the x axis after T_U seconds. We will now input shape $x[t]$ using a ZV input shaper. For simplicity, we will recast the equation of our ZV input shaper $h_{ZV}[t]$ in equation (2.7) into the more general form shown in equation (2.14). The discrete time convolution between $x[t]$ and $h_{ZV}[t]$ can be evaluated as follows.

$$x_{ZV}^{IS}[t] \doteq x[t] \star h_{ZV}[t] = \sum_{k=-\infty}^{\infty} x[k] h_{ZV}[t-k] \tag{2.13}$$

$$h_{ZV}[t] = \begin{cases} A_1, & \text{if } t = 0 \\ A_2, & \text{if } t = t_2 \\ 0, & \text{otherwise} \end{cases} \quad , \quad A_1 + A_2 = 1 \tag{2.14}$$

Given that $t_2 > 0$ we find that $T_S = t_2$ where T_S is the duration of our input shaper. We can combine equations (2.13) and (2.14) to get

$$x_{ZV}^{IS}[t] = A_1 x[t] + A_2 x[t - t_2] \tag{2.15}$$

We can also extend the results in equation (2.15) for higher derivatives of $x[t]$.

$$\frac{d}{dt}(x_{ZV}^{IS}[t]) = \frac{d}{dt}(x[t] \star h_{ZV}[t]) = \frac{d}{dt}(x[t]) \star h_{ZV}[t] \tag{2.16}$$

$$\dot{x}_{ZV}^{IS}[t] = A_1\dot{x}[t] + A_2\dot{x}[t - t_2] \tag{2.17}$$

Similar results also hold for input shapers with more impulses such as the two-hump EI shaper $h_{2EI}[t]$ with $0 = t_1 < t_2 < t_3 < t_4$, $T_S = t_4$. Again, we recast the equations for the amplitudes and timing of the shaper's impulses in equation (2.11) in the more general form given in equation (2.19).

$$x_{2EI}^{IS}[t] \doteq x[t] \star h_{2EI}[t] = \sum_{k=-\infty}^{\infty} x[k]h_{2EI}[t - k] \tag{2.18}$$

$$h_{2EI}[t] = \begin{cases} A_1, & \text{if } t = 0 \\ A_2, & \text{if } t = t_2 \\ A_3, & \text{if } t = t_3 \\ A_4, & \text{if } t = t_4 \\ 0, & \text{otherwise} \end{cases} \quad , \quad A_1 + A_2 + A_3 + A_4 = 1 \tag{2.19}$$

$$x_{2EI}^{IS}[t] = A_1x[t] + A_2x[t - t_2] + A_3x[t - t_3] + A_4x[t - t_4] \tag{2.20}$$

An important consequence of the convolution process in equations (2.13) and (2.18) is that the duration of the resulting shaped trajectory ($x_{ZV}^{IS}[t]$ or $x_{2EI}^{IS}[t]$) will equal the sum duration of the original trajectory $x[t]$ and the duration of the input shaper. That is, if $x[t]$ defines a motion trajectory on the closed interval $t \in [0, T_U]$ then in equation (2.13), $x_{ZV}^{IS}[t]$ will now be defined for $t \in [0, T_U + t_2]$. For the two-hump EI shaper $h_{2EI}[t]$ in equation (2.18) we find that $x_{2EI}^{IS}[t]$ is defined for $t \in [0, T_U + t_4]$. This result was shown visually in Figure 2.1.

Equipped with this information we can now evaluate equation (2.15) at its boundary conditions.

$$x_{ZV}^{IS}[0] = A_1x[0] + A_2x[-t_2] \tag{2.21}$$

$$x_{ZV}^{IS}[T_U + t_2] = A_1x[T_U + t_2] + A_2x[T_U] \tag{2.22}$$

Notice how in the above two equations, we need to make evaluations of $x[t]$ that fall outside of its original range of definition $[0, T_U]$. We thus need to develop an extended version of our function $x[t]$, which we will call $x_{ex}[t]$, defined for any t. For rest to rest motion trajectories this extension process is straight forward and we can simply propagate

the initial and final positions of $x[t]$.

$$x_{ex}[t] = \begin{cases} x[0], & \text{if } t < 0 \\ x[t], & \text{if } t \in [0, T_U] \\ x[T_U], & \text{if } t > T_U \end{cases} \quad , \quad \dot{x}_{ex}[t] = \begin{cases} 0, & \text{if } t < 0 \\ \dot{x}[t], & \text{if } t \in [0, T_U] \\ 0, & \text{if } t > T_U \end{cases} \tag{2.23}$$

Using this new form we can recast equations (2.13) and (2.18) as convolutions between $x_{ex}[t]$ and our input shaper to obtain the following results.

$$x_{ZV}^{IS}[t] = A_1 x_{ex}[t] + A_2 x_{ex}[t - t_2] \tag{2.24}$$

$$\dot{x}_{ZV}^{IS}[t] = A_1 \dot{x}_{ex}[t] + A_2 \dot{x}_{ex}[t - t_2] \tag{2.25}$$

$$x_{2EI}^{IS}[t] = A_1 x_{ex}[t] + A_2 x_{ex}[t - t_2] + A_3 x_{ex}[t - t_3] + A_4 x_{ex}[t - t_4] \tag{2.26}$$

$$\dot{x}_{2EI}^{IS}[t] = A_1 \dot{x}_{ex}[t] + A_2 \dot{x}_{ex}[t - t_2] + A_3 \dot{x}_{ex}[t - t_3] + A_4 \dot{x}_{ex}[t - t_4] \tag{2.27}$$

The above equations are very useful from an onboard implementation standpoint. We can use these equations to solve for the required position of the quadrotor along our input-shaped motion trajectory at any time t as long as we have enough information to evaluate $x_{ex}[t]$ at any time t. Furthermore, because of the constraints on the A_i values presented in (2.14) we get the following results for the boundary conditions of our shaped flight trajectory.

$$x_{ZV}^{IS}[0] = x[0] \quad , \quad \dot{x}_{ZV}^{IS}[0] = \dot{x}[0] = 0 \tag{2.28}$$

$$x_{ZV}^{IS}[T_U + t_2] = x[T_U] \quad , \quad \dot{x}_{ZV}^{IS}[T_U + t_2] = \dot{x}[T_U] = 0 \tag{2.29}$$

Similar results will also hold for $x_{2EI}^{IS}[t]$ and its derivatives at $t = 0$ and $t = T_U + t_4$. The results in equations (2.28) and (2.29) are helpful for implementation in that they provide guarantees for how the boundary conditions of our chosen $x[t]$ will affect the resulting input-shaped trajectory that we will ultimately have our drone fly. As we will discuss in the subsequent section though, the extension technique in equation (2.23) does not produce good results for non-rest to rest maneuvers. We will also demonstrate that the favourable boundary condition properties in equations (2.28) and (2.29) no longer hold. These difficulties motivate creating a new approach for generating input-shaped trajectories for non-rest to rest maneuvers.

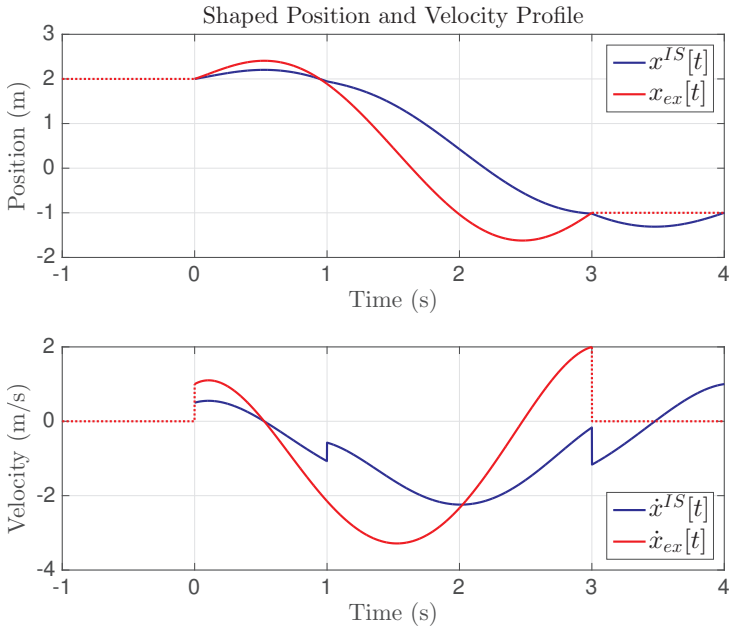

FIGURE 2.5: Improper Input Shaping of Non-Rest to Rest Trajectory

2.3 Non-Rest to Rest Input-Shaped Trajectory Generation

The goal of this section is to investigate how to apply input shaping to non-rest to rest flight trajectories in real time. In the previous section we established that our starting motion trajectory $x[t]$ needed to be extended into the new form $x_{ex}[t]$ in order to properly evaluate the resulting convolution. We proposed a basic extension technique in equation (2.23). If we apply this same technique to a non-rest to rest $x[t]$ trajectory though we will get unfavorable results as demonstrated in Figure 2.5 where we use a ZV input shaper with $t_2 = 1$. In Figure 2.5, $x_{ex}[t]$ is continuous but $\dot{x}_{ex}[t]$ has jump discontinuities which are multiplied by the convolution process illustrated here. This results in an input-shaped velocity trajectory $\dot{x}^{IS}[t]$ that is not feasible to track. We thus proposed propagating $x[t]$ by assuming a constant acceleration motion. Assuming that $x[t]$ has boundary conditions x_0, $\dot{x}_0$, $\ddot{x}_0$, x_U, $\dot{x}_U$, $\ddot{x}_U$ at $t = 0$ and $t = T_U$ respectively we thus get

$$x_{ex}[t] = \begin{cases} x_0 + \dot{x}_0 t + \frac{1}{2}\ddot{x}_0 t^2 & \text{if } t < 0 \\ x[t] & \text{if } t \in [0, T_U] \\ x_U + \dot{x}_U(t - T_U) + \frac{1}{2}\ddot{x}_U(t - T_U)^2 & \text{if } t > T_U \end{cases}$$

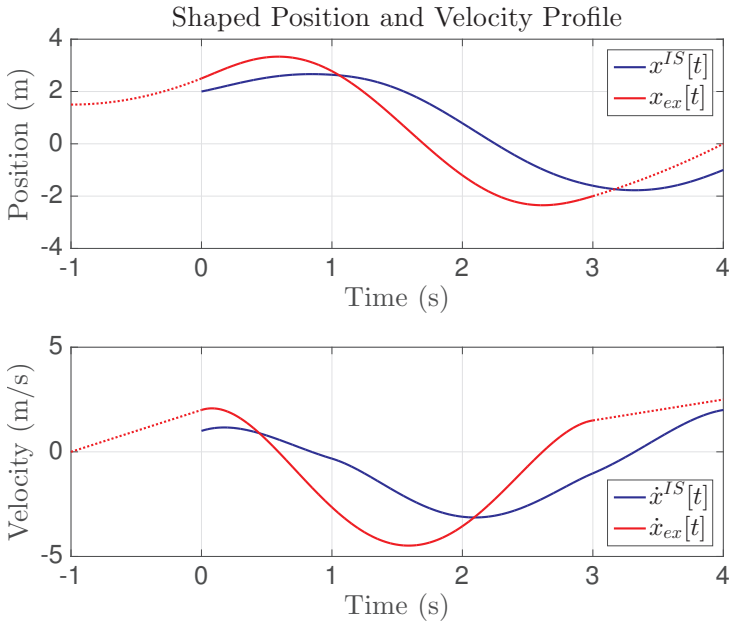

FIGURE 2.6: Proper Input Shaping of Non-Rest to Rest Trajectory

$$\dot{x}_{ex}[t] = \begin{cases} \dot{x}_0 + \ddot{x}_0 t & \text{if } t < 0 \\ \dot{x}[t] & \text{if } t \in [0, T_U] \\ \dot{x}_U(t - T_U) + \ddot{x}_U(t - T_U) & \text{if } t > T_U \end{cases} \quad , \quad \ddot{x}_{ex}[t] = \begin{cases} \ddot{x}_0, & \text{if } t < 0 \\ \ddot{x}[t] & \text{if } t \in [0, T_U] \\ \ddot{x}_U, & \text{if } t > T_U \end{cases} \tag{2.30}$$

For rest to rest motions, equation (2.30) collapses back to (2.23). Applying the extension technique from equation (2.30) to the example in Figure 2.5, we now obtain the input-shaped trajectory in Figure 2.6. Notice how in Figure 2.6 our input-shaped trajectories for position and velocity are continuous. At the same time though, the boundary conditions for our input-shaped trajectory and unshaped $x[t]$ function no longer match as they did previously with rest to rest motions in equations (2.28) and (2.29). This is problematic since we ultimately want to ensure that our input-shaped trajectory starts where the quadrotor currently is and ends where we want it to be.

It turns out though that we can predict how much these boundary conditions will change as a result of the extension and input shaping process. Specifically, we can generate a closed form mapping between the desired boundary conditions of our resulting input-shaped trajectory and those required for our starting trajectory. For a two-impulse shaper

such as the ZV shaper for instance, we find that

$$\begin{aligned}
T_F =& T_U + T_S = T_U + t_2 && (2.31)\\
x^{IS}[0] = x_0^{IS} =& A_1 x_{ex}[0] + A_2 x_{ex}[-t_2] \\
=& A_1 x_0 + A_2(x_0 - \dot{x}_0 t_2 + \frac{1}{2}\ddot{x}_0 t_2^2) \\
=& x_0 + A_2(-\dot{x}_0 t_2 + \frac{1}{2}\ddot{x}_0 t_2^2) && (2.32)\\
\dot{x}^{IS}[0] = \dot{x}_0^{IS} =& \dot{x}_0 - A_2 \ddot{x}_0 t_2 && (2.33)\\
\ddot{x}^{IS}[0] = \ddot{x}_0^{IS} =& A_1 \ddot{x}_0 + A_2 \ddot{x}_0 = \ddot{x}_0 && (2.34)\\
x^{IS}[T_F] = x_F^{IS} =& x_U + A_1(\dot{x}_U t_2 + \frac{1}{2}\ddot{x}_U t_2^2) && (2.35)\\
\dot{x}^{IS}[T_F] = \dot{x}_F^{IS} =& \dot{x}_U + A_1 \ddot{x}_U t_2 && (2.36)\\
\ddot{x}^{IS}[T_F] = \ddot{x}_F^{IS} =& \ddot{x}_U && (2.37)
\end{aligned}$$

Similarly, for a four-impulse shaper we have:

$$\begin{aligned}
T_F =& T_U + T_S = T_U + t_4 && (2.38)\\
x_0^{IS} =& x_0 + A_2\Big(- \dot{x}_0 t_2 + \frac{1}{2}\ddot{x}_0 t_2^2\Big) + A_3\Big(- \dot{x}_0 t_3 + \frac{1}{2}\ddot{x}_0 t_3^2\Big) \\
&+ A_4\Big(- \dot{x}_0 t_4 + \frac{1}{2}\ddot{x}_0 t_4^2\Big) && (2.39)\\
\dot{x}_0^{IS} =& \dot{x}_0 - \ddot{x}_0\Big(A_2 t_2 + A_3 t_3 + A_4 t_4\Big) && (2.40)\\
\ddot{x}_0^{IS} =& \ddot{x}_0 && (2.41)\\
x_F^{IS} =& A_1\Big(\dot{x}_U t_4 + \frac{1}{2}\ddot{x}_U t_4^2\Big) + A_2\Big(\dot{x}_U(t_4 - t_2) + \frac{1}{2}\ddot{x}_U(t_4 - t_2)^2\Big) \\
&+ A_3\Big(\dot{x}_U(t_4 - t_3) + \frac{1}{2}\ddot{x}_U(t_4 - t_3)^2\Big) + x_U && (2.42)\\
\dot{x}_F^{IS} =& \dot{x}_U + A_1 \ddot{x}_U t_4 + A_2 \ddot{x}_U(t_4 - t_2) + A_3 \ddot{x}_U(t_4 - t_3) && (2.43)\\
\ddot{x}_F^{IS} =& \ddot{x}_U && (2.44)
\end{aligned}$$

These equations present a closed form solution for how the process of input shaping will change the boundary conditions and duration of our flight trajectories as seen in Figure 2.6. We can in turn use these relations to map a desired duration T_F and boundary conditions x_0^{IS} , $\dot{x}_0^{IS}$, $\ddot{x}_0^{IS}$, x_F^{IS}, $\dot{x}_F^{IS}$, $\ddot{x}_F^{IS}$ for our input-shaped trajectory to those required for our unshaped trajectory $x[t]$. This forms the basis for our novel trajectory generation algorithm. Notice how for a rest to rest input-shaped trajectory we would again recover the relations we found in equations (2.28) and (2.29).

From testing, this trajectory generation method works best when the duration of the

unshaped trajectory T_U is longer than the duration of the input shaper T_S. For a ZV shaper for instance this imposes the requirement $T_U > t_2$ and for the two-hump EI shaper we get that $T_U > t_4$. The extension technique proposed in (2.30) means that we can still evaluate our input-shaped trajectory if this condition is not met however this tends to produce more aggressive motion trajectories.

It is important to highlight here how this mapping technique differs from conventional approaches to input shaping. As was previously shown, for rest to rest motions we can employ the basic extension technique in equation (2.23) and the resulting shaped trajectory's boundary conditions will perfectly match those of the unshaped motion trajectory $x[t]$. For this reason, papers exploring input-shaped rest to rest maneuvers for swing mitigation in slung load systems will typically compare simulations and flight tests of tracking $x[t]$ to tracking the input-shaped version of $x[t]$. Both are viable flight trajectories that will bring the system from its current position to the desired end position. In our approach, we no longer treat $x[t]$ as a viable flight trajectory for the quadrotor. Its boundary conditions are computed based on a mapping process and no longer correspond to where the drone starts or should end up. In our system, $x[t]$ is purely a tool for creating a non-rest to rest input-shaped flight trajectory that meets our design needs and that we will ultimately want to fly.

2.3.1 Generating an Unshaped Flight Trajectory

Numerous options are available for generating an unshaped quadrotor flight trajectory $x[t]$ defining the motion along the x axis over time interval $[0, T_U]$ with boundary conditions x_0, $\dot{x}_0$, $\ddot{x}_0$, x_U, $\dot{x}_U$, $\ddot{x}_U$. Our goal here is to identify a suitable approach that will be computationally simple enough to implement onboard a quadrotor drone. In [32] for instance, B-splines are used to generate dynamically feasible smooth flight trajectories for quadrotor drones. Another popular trajectory generation method is to use polynomials of varying degree [11]. The use of polynomials is typically tied to the differentially flat nature of quadrotor drones. A shown in Appendix A, for a simple quadrotor drone the applied moment is a function of the fourth derivative of the motion trajectory and the second derivative of the desired yaw trajectory [11] [33]. Solving for the trajectory that minimizes the norm of these two values yields constraints on the derivatives of the position and yaw trajectories of the quadrotor drone and informs the selected polynomial order [11]. We thus typically see 5th order polynomials used as quadrotor flight trajectories. In [34] for instance, multiple fifth order polynomials are stitched together to form a position trajectory. Doing so enables the system to generate motion trajectories

that can dwell at the system's maximum velocity for long durations. Generating these trajectories though requires assessing whether the quadrotor will actually be able to reach its maximum velocity in the allotted time. As such, this technique would likely be better suited to generating longer flight trajectories. A similar approach using a single fifth order polynomial to define the motion trajectory is presented in [35]. In [35] though, the proposed trajectory must be validated using a series of pass fail tests to ensure that it does not exceed the physical limits of the quadrotor. If this does occur, the time scale of the trajectory can be modified such that the same path is followed but over a longer motion time [11].

Given the boundary conditions specified in this section, it makes sense to use a single fifth order polynomial for the motion trajectory as done in [35]. A benefit of this approach is that it is computationally simple to implement. As demonstrated in [35], hundreds of fifth order polynomials can be generated and evaluated using onboard computational power in order to select a viable motion path. This approach differs from an optimization strategy which would seek to find one viable trajectory that minimizes a given cost function.

From [35] we express our fifth order polynomial for $x[t]$ and its higher derivatives as

$$x[t] = \frac{\alpha}{120}t^5 + \frac{\beta}{24}t^4 + \frac{\gamma}{6}t^3 + \frac{\ddot{x}_0}{2}t^2 + \dot{x}_0 t + x_0 \tag{2.45}$$

$$\dot{x}[t] = \frac{\alpha}{24}t^4 + \frac{\beta}{6}t^3 + \frac{\gamma}{2}t^2 + \ddot{x}_0 t + \dot{x}_0 \tag{2.46}$$

$$\ddot{x}[t] = \frac{\alpha}{6}t^3 + \frac{\beta}{2}t^2 + \gamma t + \ddot{x}_0 \tag{2.47}$$

To solve for the unknown coefficients α, β, and γ we first solve for the following terms.

$$\Delta p = x_U - x_0 - \dot{x}_0 T_U - \frac{1}{2}\ddot{x}_0 {T_U}^2 \tag{2.48}$$

$$\Delta v = \dot{x}_U - \dot{x}_0 - \ddot{x}_0 T_U \tag{2.49}$$

$$\Delta a = \ddot{x}_U - \ddot{x}_0 \tag{2.50}$$

When can then evaluate equations (2.45) through (2.47) at time $t = T_U$ and substitute Δp, Δv and Δa to obtain the following system of equations.

$$\begin{bmatrix} \frac{1}{120}{T_U}^5 & \frac{1}{24}{T_U}^4 & \frac{1}{6}{T_U}^3 \\ \frac{1}{24}{T_U}^4 & \frac{1}{6}{T_U}^3 & \frac{1}{2}{T_U}^2 \\ \frac{1}{6}{T_U}^3 & \frac{1}{2}{T_U}^2 & T_U \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \Delta p \\ \Delta v \\ \Delta a \end{bmatrix} \tag{2.51}$$

Notice that the matrix in equation (2.51) has determinant $-{T_U}^9/8640$ and is thus invertible as long as $T_U \neq 0$. Solving for this inversion yields

$$\frac{1}{{T_U}^5}\begin{bmatrix} 720 & -360T_U & 60{T_U}^2 \\ -360T_U & 168{T_U}^2 & -24{T_U}^3 \\ 60{T_U}^2 & -24{T_U}^3 & 3{T_U}^4 \end{bmatrix}\begin{bmatrix} \Delta p \\ \Delta v \\ \Delta a \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{2.52}$$

Evaluating equation (2.52) allows us to fully solve for all the coefficients required to construct our unshaped motion trajectory $x[t]$ in (2.45) with a 5th order polynomial. Combined with the proposed constant acceleration propagation technique in equation (2.30) we can now essentially evaluate $x_{ex}[t]$ and all of its higher derivatives at any time t. This is significantly less memory intensive than storing $x_{ex}[t]$ as a lookup table of desired positions, velocities, accelerations etc. for select times.

2.3.2 Summary of Trajectory Generator

This section summarizes how our approach generates flight trajectories as a precomputation step and subsequently solves for the desired position, velocity and acceleration of the drone at any time step along the flight trajectory. For simplicity we will present the algorithm only for generating the motion along the x axis. The entire process can be repeated along the y and z axes to generate the input-shaped flight trajectories for those motions.

Algorithm 1 provides a pseudocode implementation of the precomputation step. This algorithm gets run one time when the quadrotor receives a new target state to reach. All the computations are performed onboard the quadrotor using information about the quadrotor's current state and the desired state. The desired states for the quadrotor would normally be chosen by a high level system responsible for obstacle detection and avoidance. The development of this high level system is beyond the scope of this thesis.

The outputs from Algorithm 1 provide all the information needed to evaluate the $x_{ex}[t]$ function specific to the required maneuver at any time t_{eval}. Any time we want to fly a new maneuver to a new target end state, we must rerun Algorithm 1 to generate a new $x_{ex}[t]$ function. By convention we define $t = 0$ to be the start of a given flight trajectory and measure t_{eval} as the time elapsed since then. Algorithm 2 uses the information provided by Algorithm 1 to solve for the desired state of the drone at $t = t_{eval}$.

Algorithm 2 gets evaluated with every loop of the flight controller onboard the system to supply the desired position, velocity, acceleration, jerk and snap that the drone should

Algorithm 1: Input-Shaped Trajectory Precomputation

input : Current drone state (x_0^{IS}, $\dot{x}_0^{IS}$, $\ddot{x}_0^{IS}$), Desired drone state (x_F^{IS}, $\dot{x}_F^{IS}$, $\ddot{x}_F^{IS}$), Desired motion duration (T_F), Period of oscillation for natural frequency of swinging (T_d), $\zeta_{IS} = 0$, $V_{tol} = 0.05$, Input shaper type (ZV, two-hump EI)

if *Input shaper type == ZV* **then**
  Solve for A_1, A_2, t_2 from equation (2.14) using equations (2.7) and (2.8)
  Solve for x_0, $\dot{x}_0$, $\ddot{x}_0$, x_U, $\dot{x}_U$, $\ddot{x}_U$ and T_U using equations (2.31) through (2.37)
else
  if *Input shaper type == two-hump EI* **then**
    Solve for A_1, A_2, A_3, A_4, t_2, t_3, t_4 from equation (2.19) using equations (2.11) and (2.12)
    Solve for x_0, $\dot{x}_0$, $\ddot{x}_0$, x_U, $\dot{x}_U$, $\ddot{x}_U$ and T_U using equations (2.38) through (2.44)
  else
    Error: Invalid Shaper Type
  end
end

Solve for Δp, Δv, Δa using equations (2.48) through (2.50)
Solve for α, β, γ using equation (2.52)

output: α, β, γ, x_0, $\dot{x}_0$, $\ddot{x}_0$, x_U, $\dot{x}_U$, $\ddot{x}_U$, T_U, Input shaper impulse amplitudes (A_i) and timing (t_i)

Algorithm 2: Input-Shaped Trajectory Evaluation

input : α, β, γ, x_0, $\dot{x}_0$, $\ddot{x}_0$, x_U, $\dot{x}_U$, $\ddot{x}_U$, T_U, Input shaper type (ZV, two hump EI), Input shaper impulse amplitudes (A_i) and timing (t_i), t_{eval}

if *Input shaper type == ZV* **then**
  Evaluate $x_{ZV}^{IS}[t_{eval}]$ and its higher derivatives using equation (2.24), where $x_{ex}[t]$ is given by equation (2.30) and $x[t]$ is given by equation (2.45).
else
  if *Input shaper type == two-hump EI* **then**
    Evaluate $x_{2EI}^{IS}[t_{eval}]$ and its higher derivatives using equation (2.26), where $x_{ex}[t]$ is given by equation (2.30) and $x[t]$ is given by equation (2.45).
  else
    Error: Invalid Shaper Type
  end
end

output: $x^{IS}[t_{eval}]$, $\dot{x}^{IS}[t_{eval}]$, $\ddot{x}^{IS}[t_{eval}]$, $\dddot{x}^{IS}[t_{eval}]$, $\ddddot{x}^{IS}[t_{eval}]$

have along the x axis at time t_{eval}. This entire process is also run for the y and z axis motion in order to compute the full desired state of the drone. The desired position $\mathbf{r}^{dq}$, velocity $\mathbf{v}^{dq}$ and acceleration $\mathbf{a}^{dq}$ will all be used as part of the flight controller for feedback and feedforward control elements in Section 3.3. The desired jerk and snap values will be used as part of the Attitude Controller for a set of differential flatness computations. This will be discussed in Appendix A.

One of the benefits of this trajectory generation method is that it is computationally simple to implement. The extended polynomial motion trajectories for each axis $x_{ex}[t]$, $y_{ex}[t]$, $z_{ex}[t]$ and their higher derivatives are are easy to evaluate due to the use of simple polynomial elements. Furthermore, the input shaping process has been simplified to a weighted sum. This facilitates implementing these algorithms onboard a quadrotor drone with limited memory. The requirements for implementing this system are negligible compared to the memory requirements to store the entire time history of the desired flight trajectory and its higher derivatives in a discretized lookup table format.

Although it was not included in our original formulation of the trajectory generator, an additional verification step can be added between Algorithm 1 and Algorithm 2 to validate that the proposed motion trajectory will not violate physical constraints on our system. A series of conservative computationally efficient pass/fail tests for doing so are presented in [35]. The extension and input shaping techniques presented in this work does complicate this process. However, from [17], since we impose that the amplitudes of all the impulses in our shaper sum to one, we know that the input shaping process will not alter the feasibility of our motion trajectory. Thus as long as we can verify that $x_{ex}[t]$ is feasible using the methods in [35], we can guarantee that the resulting input-shaped version is also feasible.

2.3.3 Simulation Results

This section studies the effectiveness of the proposed trajectory generation method for swing management in a slung load system. To do so, we create a simulated gantry crane transporting a slung load. The upper portion of the crane will perfectly track the prescribed input-shaped motion trajectory from our algorithm and we will assess the effect that this has on the resulting motion of the payload. In essence, this simulation attempts to replicate the behavior that a quadrotor-slung load system would have if it achieved perfect flight trajectory tracking. For our gantry crane model, we will use an inelastic, constant length cable model presented in [19].

Within this model, the position of the upper gantry resolved in the inertial frame is specified using $[\xi\,\zeta\,\eta]^\intercal$ while the motion of the point mass payload is given by $[x\,y\,z]^\intercal$. An inelastic cable of length l connects the gantry to the point mass. We will compute the angular velocity corresponding to the natural frequency of the system as $\omega = \omega_{IS} = \sqrt{g/l}$. Neglecting damping we get from [19] that the equations of motion of the payload are given as

$$\begin{aligned}\ddot{x} + \omega^2 x =& \omega^2\xi - \frac{x-\xi}{l}\ddot{\eta} - \omega^2\frac{x-\xi}{2l^2}\Big((x-\xi)^2 + (y-\zeta)^2\Big) \\ &- \frac{x-\xi}{l^2}\Big((\dot{x}-\dot{\xi})^2 + (\dot{y}-\dot{\zeta})^2 + (x-\xi)(\ddot{x}-\ddot{\xi}) + (y-\zeta)(\ddot{y}-\ddot{\zeta})\Big) \qquad (2.53)\end{aligned}$$

$$\begin{aligned}\ddot{y} + \omega^2 y =& \omega^2\zeta - \frac{y-\zeta}{l}\ddot{\eta} - \omega^2\frac{y-\zeta}{2l^2}\Big((x-\xi)^2 + (y-\zeta)^2\Big) \\ &- \frac{y-\zeta}{l^2}\Big((\dot{x}-\dot{\xi})^2 + (\dot{y}-\dot{\zeta})^2 + (x-\xi)(\ddot{x}-\ddot{\xi}) + (y-\zeta)(\ddot{y}-\ddot{\zeta})\Big) \qquad (2.54)\end{aligned}$$

Equations (2.53) and (2.54) can be rearranged and combined together to isolate for $\ddot{x}$ and $\ddot{y}$ in order to create a simulation model in MATLAB. At each time step, the position, velocity and acceleration of the gantry (ξ, $\dot{\xi}$, $\ddot{\xi}$, ζ, $\dot{\zeta}$, $\ddot{\zeta}$, $\ddot{\eta}$) are computed using our trajectory generation algorithm and used to numerically solve for the resulting motion history of the payload.

In the following examples we consider a slung load system with chosen cable length $l = 0.5$m. This yields $\omega = 4.427\text{s}^{-1}$ and $T_d = 1.4192$s. Thus for a ZV input shaper we have $T_s = 0.7096$s and for a two-hump EI shaper we have $T_s = 2.1288$s. If we employ the latter shaper, we can safely use our trajectory generation technique for shaped motion trajectories of duration longer than 4.26s.

A consequence of the convolution process required for input shaping is that it necessarily extends the duration of the flight trajectory. Most papers that study the benefits of input shaping however do not take this into account when comparing the swing mitigation performance of shaped and unshaped flight trajectories. For example, if an input shaper has duration 1s and it is used to shape a motion trajectory that normally lasts 5s, most papers would compare the performance of tracking the original 5s trajectory to the performance for the input-shaped 6s trajectory. This raises the question of whether an unshaped trajectory might achieve favourable swing mitigation if it lasted as long as the shaped trajectory did. Consider a scenario where we have a fixed start and end state as well as a fixed time to reach that end state. An interesting question to ask would be whether it is better to generate an unshaped smooth trajectory that uses the full allotted

time to reach the end state or whether it would be better to generate a more aggressive unshaped trajectory that reaches its end state faster but is then input-shaped to completely use the allotted time. We believe that this scenario constitutes a fairer way to evaluate the performance of an input-shaped motion trajectory and this will form the basis of our approach in this section.

To begin with, we will consider a rest to rest motion of our simulated gantry crane system covering a lateral distance of 20m meters in a time of 5s. Our baseline motion trajectory will be a simple fifth order polynomial with coefficients chosen such that the trajectory reaches the end state in 5s. We will compare this motion trajectory against input-shaped motion trajectories created using ZV and two-hump EI input shapers using Algorithms 1 and 2.

Figure 2.7 shows the x and y motion trajectory generated for this rest to rest motion for all three options. Notice how both input-shaped motion trajectories require higher velocities than the unshaped motion trajectory. This is a direct consequence of the fact that we are constraining all three motion trajectories to take 5s. The convolution process also causes the desired acceleration profile for both shaped trajectories to have a number of sharp corners especially for the two hump EI input shaper. Figure 2.8 shows the magnitude of the swing angle of the payload for all three motion trajectories. From this figure, it is apparent how effective the input shaping process has been at mitigating swinging in the payload after the motion is completed. It is important to highlight that input shaping does not eliminate all swinging motion in the payload. Indeed there is significant swinging in the payload between times 0s to 5s for all three trajectories. This swinging motion is inevitable though because the upper portion of our gantry system is accelerating. Once the accelerating maneuver finishes though, we can see that the use of input shaping has helped to mitigate swinging. Notice how there is a roughly fourfold decrease in swing amplitude for both input-shaped trajectories compared to the unshaped trajectory from 5s onwards. This illustrates how even though the input-shaped trajectories required higher velocities and had more aggressive acceleration profiles they still managed to mitigate swinging. While these kinds of results for rest to rest motion trajectories have been presented in numerous papers, it is important to note that the time normalization step we present here has not previously been considered. We believe that this further strengthens the argument for using input shaping motion trajectories.

The results in Figure 2.8 are also interesting in that the input-shaped trajectories did not perfectly eliminate residual swinging motion in the payload from 5s onwards. From Figures 2.3 and 2.4 we might expect our input-shaped trajectory to result in no residual swinging since we matched the natural frequency and damping ratio for our shaper and

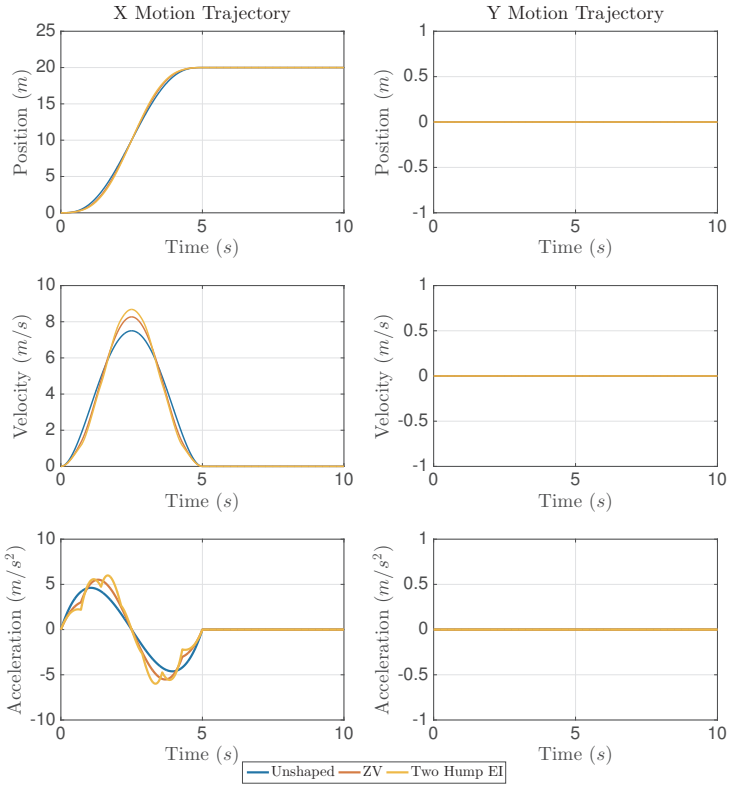

FIGURE 2.7: Rest to Rest Motion Trajectory Comparison

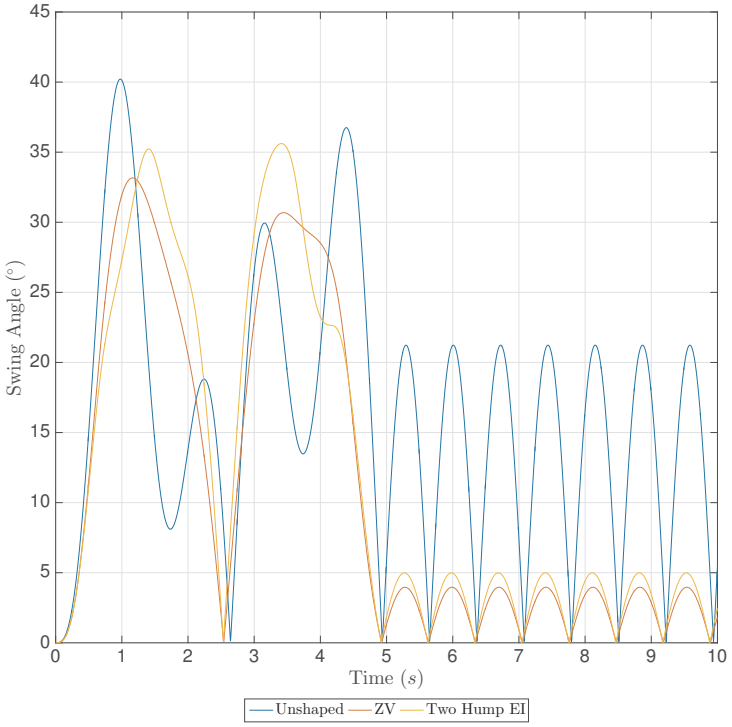

FIGURE 2.8: Rest to Rest Motion Swing Comparison

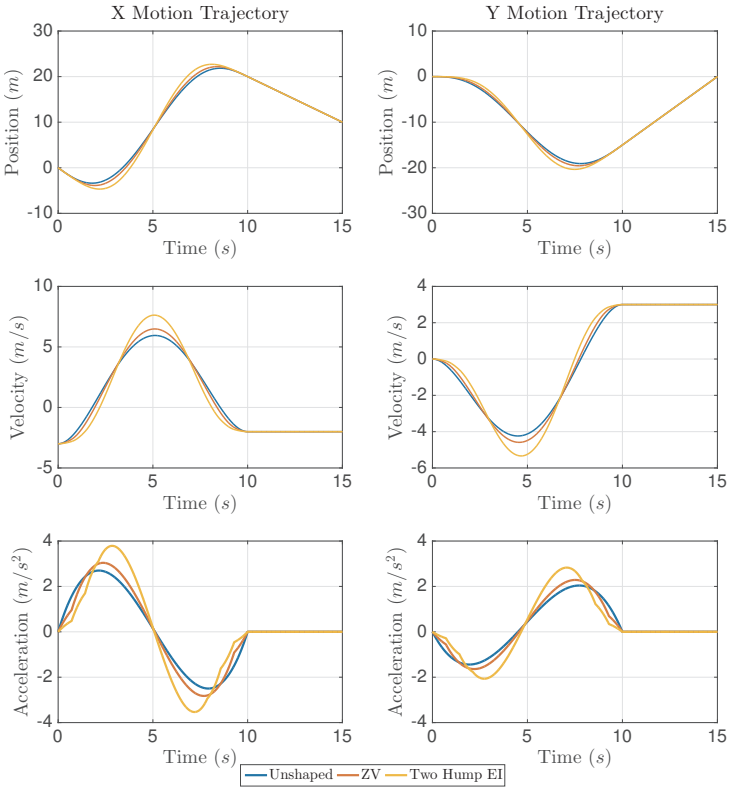

FIGURE 2.9: Non-Rest to Rest Motion Trajectory Comparison

the gantry model. It is important to keep in mind though that input shaping is derived based on a second order linear model while our gantry system, given in equations (2.53) and (2.54), is a second order nonlinear model. Thus while input shaping still achieves swing mitigation, it will not necessarily guarantee a perfect swing-free behaviour once the gantry stops accelerating. In testing with this gantry model, we found that input-shaped trajectories with higher accelerations tended to still have some residual swinging. This swinging was always worse for the corresponding unshaped trajectory.

We will now consider a more complicated non-rest to rest motion trajectory for the simulated gantry crane system. In this situation, the gantry system will start at the origin with an initial velocity of $[-3 \quad 0 \quad 0]^\intercal \mathrm{m\,s^{-1}}$ and reach a target velocity of $[-2 \quad 3 \quad 0]^\intercal \mathrm{m\,s^{-1}}$ at position $[20 \quad -15 \quad 0]^\intercal \mathrm{m}$ in 10s. In this simulation there is no initial relative velocity between the payload and the upper gantry. The motion trajectories using both input shapers and without any input shaping are presented in Figure 2.9.

As with Figure 2.7, in Figure 2.9 we see that the input-shaped motion trajectories reach higher velocities and accelerations than the unshaped trajectory. In Figure 2.10 though, we can see that the input shapers have effectively eliminated all residual swinging motion of the payload once the gantry system enters steady state motion after 10s. In contrast,

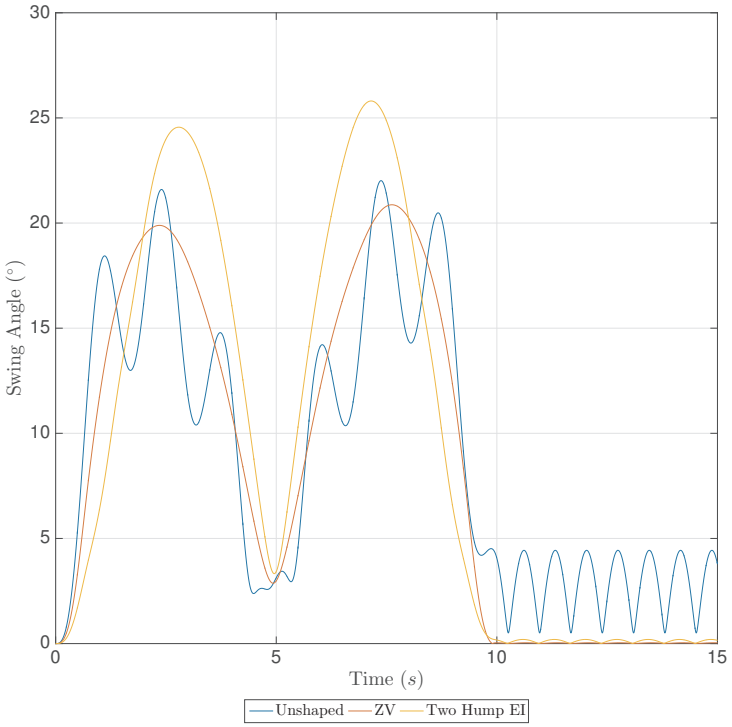

FIGURE 2.10: Non-Rest to Rest Motion Swing Comparison

the unshaped trajectory has a consistently non-zero swing magnitude suggesting that the payload is spinning about the z axis in addition to oscillating. These results highlight the effectiveness of our proposed technique for generating non-rest to rest input-shaped motion trajectories to mitigate swinging motion.

The results presented in Figures 2.8 and 2.10 raise an important question of which shaper would be best to select. For the rest to rest motion for instance, the ZV input shaper produced slightly smaller residual oscillations and for both simulations presented, the ZV input shaper has a consistently lower swinging magnitude during the motion trajectory. The ZV input-shaped trajectories also require lower velocities and accelerations than the two hump EI shaped trajectories. As discussed previously in Section 2.2.1 though, the two-hump EI shaper is more robust to uncertainty in the natural frequency of our system. In the previous simulations we used a cable length of $l = 0.5$m with $T_d = 1.4192$s. We now repeat the non-rest to rest simulation with the same cable length but we will introduce error into our system by designing our input shapers for $T_d = 1.1354$s. The resulting swing angle magnitudes are presented in Figure 2.11. Notice how when we compare Figures 2.10 and 2.11 that the swing mitigation performance of the ZV shaper is degraded by this error while the performance of the two-hump EI shaper remains essentially unchanged. This makes sense since as we previously saw with Figure 2.3, the PRV for a ZV input shaper increases significantly faster than the PRV for a two-hump EI shaper as we introduce

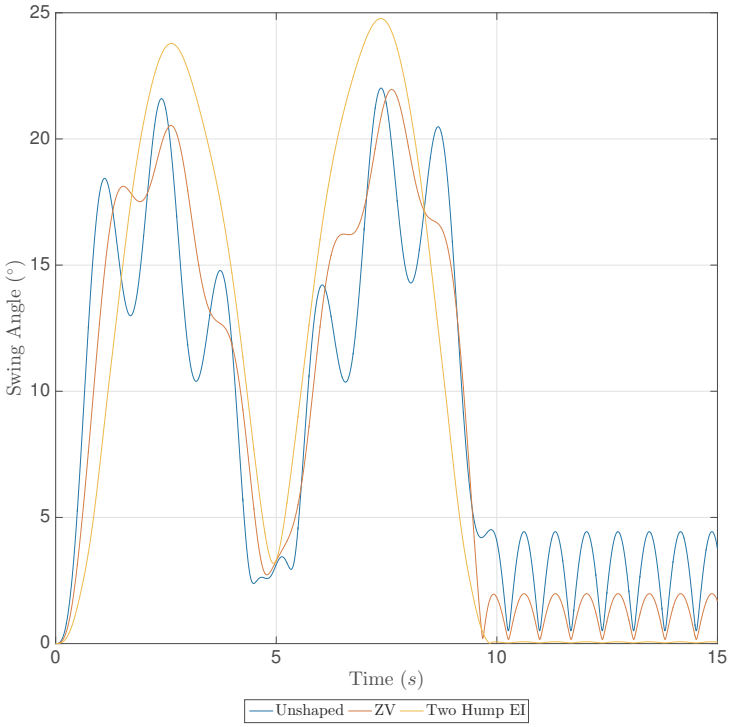

FIGURE 2.11: Non-Rest to Rest Motion with Design Error Swing Comparison

error between the true natural frequency of our system and the value used to design our shaper. These results highlight the potential benefits of using more robust input shapers even for non-rest to rest motions.

In summary, this section has demonstrated how our input-shaped trajectory generation technique can effectively mitigate swinging in a simulated gantry system with both rest to rest and non-rest to rest motions. The results presented herein also highlights how uncertainty in the natural frequency of the system can affect the performance of certain input shapers. We can manage this uncertainty by using more robust input shapers. In the next section we will explore the issue of identifying the natural frequency of oscillation of a slung load in more detail.

2.4 Natural Frequency of a Quadrotor-Slung Load System

Implementing input shaping on a quadrotor-slung load system inherently requires knowledge of the natural frequency and damping ratio for the system. These values affect the

magnitudes and timing of the impulses that form the input shaper as previously discussed in Section 2.2.1. In this section, we discuss how to estimate the natural frequency of oscillation of the slung load in a quadrotor-slung load system.

A basic starting point for this process is to model our cable as a rigid link of length l and model our slung load as a point mass. For simplicity we can also assume no damping is present in our system. This approach yields a period T_d corresponding to the natural frequency of oscillation given in equation (2.56).

$$\omega = \sqrt{\frac{g}{l}} \tag{2.55}$$

$$T_d = 2\pi\sqrt{\frac{l}{g}} \tag{2.56}$$

Equations (2.55) and (2.56) only capture one oscillation mode for the payload. Experiments with manned helicopters transporting slung loads often explore a wider set of swinging behaviour. In [3] for instance, the author discusses yawing motion of slung loads as well as secondary oscillations of the load about its center of gravity. We can capture the latter oscillation mode by treating our payload as a rigid body where the cable mounting point is not collocated with the payload's center of mass. This new configuration could in turn affect the natural frequency of our system and introduce error into our input shaper. It is thus worth exploring this effect to see if a more advanced prediction model is warranted here. Specifically we will consider a two dimensional pendulum consisting of a rigid rod of length l_1 with a mounting point offset from its center of mass by distance l_2 as shown in Figure 2.12.

Using the nomenclature defined in Figure 2.12 we can derive the equations of motion of our system and solve for the natural frequency of its two modes of oscillation. Using the Lagrangian approach for solving the dynamics of our system we obtain the following results.

$$T = \frac{m_p {v_p}^2}{2} + \frac{I_p {\dot{\alpha_2}}^2}{2} \tag{2.57}$$

$$V = -m_p g(l_1 \cos(\alpha_1) + l_2 \cos(\alpha_2)) \tag{2.58}$$

$$L = T - V = \frac{m_p {v_p}^2}{2} + \frac{I_p \dot{\alpha}_2^2}{2} + m_p g(l_1 \cos(\alpha_1) + l_2 \cos(\alpha_2)) \tag{2.59}$$

Where T and V are the kinetic and potential energy of our system. Assuming that the values of lengths l_1 and l_2 remain unchanged we can decompose the payload velocity v_p

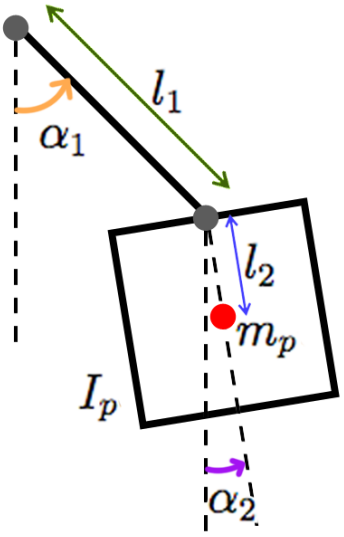

FIGURE 2.12: Rigid Body Pendulum Model

along the x and y axes and obtain an expression for ${v_p}^2$.

$$v_{px} = l_1 \cos(\alpha_1)\dot{\alpha}_1 + l_2 \cos(\alpha_2)\dot{\alpha}_2 \tag{2.60}$$

$$v_{py} = l_1 \sin(\alpha_1)\dot{\alpha}_1 + l_2 \sin(\alpha_2)\dot{\alpha}_2 \tag{2.61}$$

$$\begin{aligned} {v_p}^2 &= {v_{px}}^2 + {v_{py}}^2 && (2.62)\\ &= {l_1}^2\dot{\alpha}_1^2 + {l_2}^2\dot{\alpha}_2^2 + 2l_1l_2\dot{\alpha}_1\dot{\alpha}_2(\sin(\alpha_1)\sin(\alpha_2) + \cos(\alpha_1)\cos(\alpha_2)) && (2.63)\\ &= {l_1}^2\dot{\alpha}_1^2 + {l_2}^2\dot{\alpha}_2^2 + 2l_1l_2\dot{\alpha}_1\dot{\alpha}_2\cos(\alpha_1 - \alpha_2) && (2.64) \end{aligned}$$

We can now substitute this result into equation (2.59) to express L solely as a function of the lengths and angles in Figure 2.12.

$$L = \frac{m_p}{2}({l_1}^2\dot{\alpha}_1^2 + {l_2}^2\dot{\alpha}_2^2 + 2l_1l_2\dot{\alpha}_1\dot{\alpha}_2\cos(\alpha_1 - \alpha_2)) + \frac{I_p\dot{\alpha}_2^2}{2} + m_pg(l_1\cos(\alpha_1) + l_2\cos(\alpha_2)) \tag{2.65}$$

Assuming small angles α_1 and α_2 and using the Taylor series expansion of the cosine function we can make the following simplifications.

$$\cos(\alpha_1) \approx 1 - \frac{{\alpha_1}^2}{2} \quad , \quad \cos(\alpha_2) \approx 1 - \frac{{\alpha_2}^2}{2} \quad , \quad \cos(\alpha_1 - \alpha_2) \approx 1 - \frac{{\alpha_1}^2 - {\alpha_2}^2}{2} \approx 1 \tag{2.66}$$

We then substitute these simplifications into equation (2.65).

$$L = \frac{m_p}{2}({l_1}^2\dot{\alpha}_1^2 + {l_2}^2\dot{\alpha}_2^2 + 2l_1l_2\dot{\alpha}_1\dot{\alpha}_2) + \frac{I_p\dot{\alpha}_2^2}{2} + m_pg(l_1(1 - \frac{\alpha_1^2}{2}) + l_2(1 - \frac{\alpha_1^2}{2})) \tag{2.67}$$

Using this result we can obtain the equations of motion of our system.

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{\alpha}_1}) - \frac{\partial L}{\partial \alpha_1} = 0 \quad , \quad \frac{d}{dt}(\frac{\partial L}{\partial \dot{\alpha}_2}) - \frac{\partial L}{\partial \alpha_2} = 0 \tag{2.68}$$

$$m_p {l_1}^2 \ddot{\alpha}_1 + m_p l_1 l_2 \ddot{\alpha}_2 + m_p g l_1 \alpha_1 = 0 \tag{2.69}$$

$$(I_p + m_p {l_2}^2)\ddot{\alpha}_2 + m_p l_1 l_2 \ddot{\alpha}_1 + m_p g l_2 \alpha_2 = 0 \tag{2.70}$$

Equations (2.69) and (2.70) can be recast into matrix form $\mathbf{M}\ddot{\boldsymbol{\alpha}} + \mathbf{K}\boldsymbol{\alpha} = \mathbf{0}$ in order to identify the oscillation modes and their natural frequency.

$$\begin{bmatrix} m_p {l_1}^2 & m_p l_1 l_2 \\ m_p l_1 l_2 & I_p + m_p {l_2}^2 \end{bmatrix} \begin{bmatrix} \ddot{\alpha}_1 \\ \ddot{\alpha}_2 \end{bmatrix} + \begin{bmatrix} m_p g l_1 & 0 \\ 0 & m_p g l_2 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.71}$$

Solving for $det(\mathbf{K} - \omega^2 \mathbf{M}) = 0$ allows us to identify the natural frequencies of oscillation of the system.

$$\omega^4 (m_p {l_1}^2 I_p) - \omega^2 (m_p {}^2 g l_1 l_2 (l_1 + l_2) + I_p m_p g l_1) + m_p {}^2 g^2 l_1 l_2 = 0 \tag{2.72}$$

$$\omega^2 = \frac{(m_p {}^2 g l_1 l_2 (l_1 + l_2) + I_p m_p g l_1) \pm \sqrt{(m_p {}^2 g l_1 l_2 (l_1 + l_2) + I_p m_p g l_1)^2 - 4(m_p {}^3 {l_1}^3 l_2 g^2 I_p)}}{2 m_p {l_1}^2 I_p} \tag{2.73}$$

For our system, we are primarily concerned with eliminating the pendulous oscillation mode with the lower natural frequency. Compared to equation (2.55), equation (2.73) requires additional information to predict the natural frequency of oscillation of the system. Having a rough knowledge of the mass of the payload m_p for a parcel delivery application would be reasonable. The exact moment of inertia of the parcel about its center of mass I_p would however be more difficult to obtain. If we assume a uniform mass distribution within the parcel and a square cross section of side length $2l_2$ we find that $I_p = \frac{2 m_p l_2^2}{3}$. Substituting this result into (2.73) yields

$$\omega^2 = \frac{(m_p {}^2 g l_1 l_2 (l_1 + l_2) + \frac{2}{3} m_p {}^2 g l_1 {l_2}^2) \pm \sqrt{\left(m_p {}^2 g l_1 l_2 (l_1 + l_2) + \frac{2}{3} m_p {}^2 g l_1 {l_2}^2\right)^2 - \frac{8}{3}(m_p {}^4 {l_1}^3 {l_2}^3 g^2)}}{\frac{4}{3} m_p {}^2 {l_1}^2 {l_2}^2} \tag{2.74}$$

$${\omega_1}^2 = \frac{(g l_1 l_2 (l_1 + l_2) + \frac{2}{3} g l_1 {l_2}^2) + \sqrt{\left(g l_1 l_2 (l_1 + l_2) + \frac{2}{3} g l_1 {l_2}^2\right)^2 - \frac{8}{3}({l_1}^3 {l_2}^3 g^2)}}{\frac{4}{3} {l_1}^2 {l_2}^2} \tag{2.75}$$

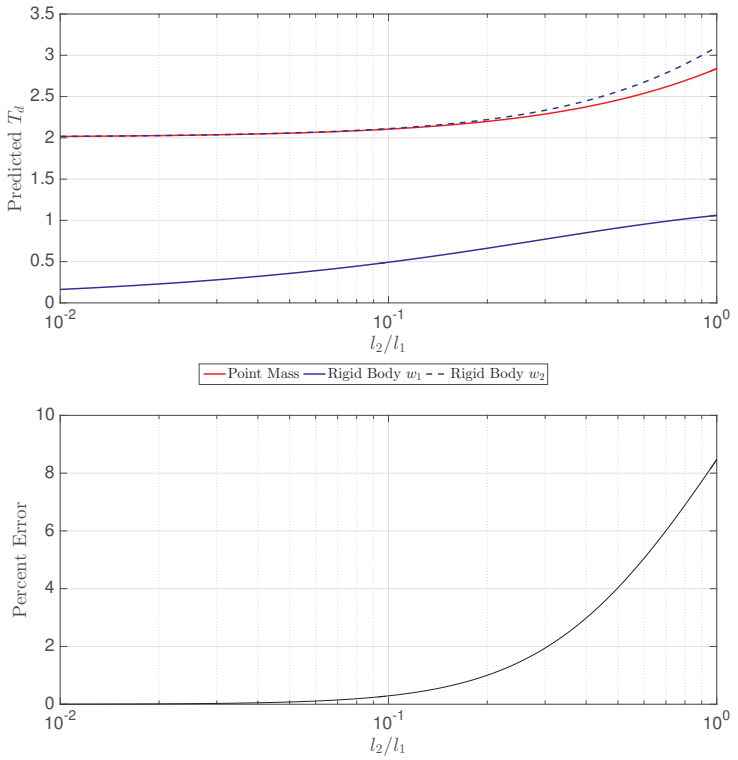

FIGURE 2.13: (Top) Comparison of Predicted T_d Values, (Bottom) Percent Error Between Point Mass and Rigid Body ω_2

$$\omega_2{}^2 = \frac{(gl_1l_2(l_1+l_2)+\frac{2}{3}gl_1l_2{}^2) - \sqrt{\left(gl_1l_2(l_1+l_2)+\frac{2}{3}gl_1l_2{}^2\right)^2 - \frac{8}{3}(l_1{}^3l_2{}^3g^2)}}{\frac{4}{3}l_1{}^2l_2{}^2} \tag{2.76}$$

where ω_1 corresponds to the oscillation mode of the payload about its mounting point to the cable and ω_2 corresponds to the main pendulous oscillation mode. As illustrated, with the proposed simplification, the mass of the payload no longer affects the computed value of ω. A similar result would hold even if we allow the cross section of the rigid body to be rectangular instead of square. In order to compare this result against equation (2.55) we will chose $l = l_1 + l_2$. We can thus assess the difference in the predicted T_d values for both models under consideration here. In Figure 2.13 we chose $l = 1$m and compute the value of T_d for both models for a range of values for $\frac{l_2}{l_1}$. Figure 2.13 also shows the percent error between the predicted T_d value from equation (2.56) and that obtained from ω in equation (2.76). As expected, both models predict similar values for T_d as l_2 approaches zero. The difference between the predicted pendulous period only becomes significant when l_2 approaches the same order of magnitude as l_1. Similar results hold for other values of l_1 that were tested but not shown here. In general, we expect $l_1 > l_2$ which suggests that there is little benefit to using the more complicated rigid body

pendulum. We thus propose using the simpler prediction model given in equation (2.56) to predict the natural frequency of our slung load system for all subsequent input shaper computations. If l_2 were to approach the same order of magnitude as l_1, we could employ a more robust input shaper such as a ZVDD shaper and continue to use equation (2.56) to compute T_d. Alternatively we could try using equation (2.76) with a less robust input shaper such as the ZV shaper.

Chapter 3

Controller Design

This chapter presents the development of a flight controller specifically designed to track input-shaped flight trajectories. We begin with a literature review focusing on controller design for a quadrotor-slung load systems to identify key trends and motivate the development of a new controller specifically for this problem. We next present a model for a quadrotor-slung load system that we can use to create a simulation environment for assessing the performance of various controllers. Afterwards we present a detailed breakdown of our proposed controller and conclude by comparing this novel controller to a series of baseline designs using our simulation environment. These simulations highlight how our proposed controller outperforms baseline controller designs for tracking input-shaped flight trajectories while rejecting external disturbances acting on the slung load.

3.1 Survey of Quadrotor-Slung Load Flight Controller Designs

There have been numerous papers which have proposed controller designs for quadrotor-slung load systems. In this work, we group these controllers into the following categories based on the overarching control strategy as done in [2]. In the subsequent subsections we will discuss these categories in greater detail.

1. Payload Trajectory Following [36] [37] [38] [39]
2. Quadrotor Trajectory Following [7] [10] [20] [22] [30] [40] [41]
3. Quadrotor Trajectory Following + Payload Swing Feedback [21] [24] [25] [26] [42] [43] [44] [45] [46]

3.1.1 Payload Trajectory Following Controllers

The first category encompasses flight controller designs that are based around tracking a prescribed payload motion trajectory as opposed to a quadrotor flight trajectory. In [37] for instance, the desired trajectory of the payload is transformed into a required cable orientation time history using differential flatness. The cable is treated as a massless rod mounted on one end to the quadrotor center of mass and to a point mass payload at the other end. The error between the measured cable state and this desired state is used to compute a desired thrust vector whose orientation is tracked by an attitude controller. The proposed geometric controller demonstrates robustness to different cable lengths and is able to maintain control during flight testing with aggressive maneuvers requiring swing angles of up to 53°. Both [36] and [38] are precursors to [37] and present the development of this controller design. In particular [38] shows the robustness of the proposed controller in simulated scenarios with even higher swing angles. A finite horizon linear quadratic regulator controller for payload trajectory tracking is presented in [39]. A major difference that separates this work from the others in this section is that it models the cable as a chain of rigid links. As with [37], the resulting system is also shown to be differentially flat with respect to the position of the point mass payload and the yaw angle of the quadrotor. The controller in [39] is shown in simulation to be able to track prescribed payload trajectories in spite of large differences between the true and desired starting states.

One limitation of these works is that the differentially flat model being used assumes that the cable is mounted at the center of mass of the quadrotor. This assumption means that the payload will not affect the attitude dynamics of the system and that changing the attitude of the quadrotor will not affect the motion of the payload. Another potential concern with this proposed controller design is how it would function in a scenario where the payload might temporarily be difficult to detect. In [37] for instance, the authors employ a downwards facing camera to feed position readings into a custom Extended Kalman Filter for payload state estimation. The authors note though that payload detection becomes more challenging with more aggressive maneuvers and with a longer cable. While [37] was able to achieve low estimation errors in all experimental setups it is unclear how their system would perform in a scenario where the payload was no longer detectable by the camera. This scenario could conceivably occur if the drone were to make aggressive changes in attitude or due to external disturbances such as wind gusts. While this issue would be inherent to almost any quadrotor-slung load system, the effect of payload detection loss would arguably be more critical in a system where the controller is based around tracking a prescribed payload flight trajectory instead of

a quadrotor flight trajectory. A further potential complexity with implementing these control strategies for parcel delivery is that the controller would need to be changed mid flight to allow the drone to operate without a payload such as for return flights.

Tracking a payload motion trajectory is not without merits though and indeed for certain applications might be preferable. If the drone's slung payload is a sensor meant to take readings while in flight, it might make sense to develop a flight controller that attempts to track a payload motion trajectory. Such a setup has notably been proposed for landmine clearing operations where a quadrotor drone would carry a metal detector suspended below it and sweep over areas to detect possible mines [21]. Alternatively, if the drone is transporting a slung load in an environment with obstacles at payload height during flight it might make more sense from a trajectory generation standpoint to create feasible payload motion trajectories to track. In [13] for instance, the authors employ the geometric controller proposed in [38] and develop flight trajectories for payload throwing and having the slung load pass through narrow openings. In this work though, we primarily envision a commercial delivery situation where our payload is simply a load to be transported from one location to another through a relatively unobstructed environment. The use of input shaping also inherently generates quadrotor flight trajectories to be tracked. The form of the geometric controller used for swing management in [37] and [38] though remains a compelling option due to its coordinate-free design.

3.1.2 Quadrotor Trajectory Following Controllers

The second category of controllers encompasses designs that attempt to track a prescribed quadrotor flight trajectory without employing feedback control to dampen the swinging motion of the payload. Within this category, a first set of papers employ basic quadrotor flight controllers and rely entirely on input-shaped flight trajectories to manage payload swinging [10] [20] [22]. In [10], the authors rely on a ZVDD input shaper for a simulated 2D quadrotor-slung load system. In [20], the authors use a ZVD shaper for a helicopter-slung load system. These papers demonstrate that input-shaped motion trajectories are effective at preventing swinging motion due to accelerations of the drone along a prescribed flight trajectory.

A key limitation of the proposed approach is that it is not meant to actively cancel out swinging disturbances. This issue was noted in [22] where they achieved poorer swing mitigation results during test flights as compared to simulations. Conventional input shapers are designed to avoid inducing swinging motion in systems that are not already swinging. More recent works have proposed input shaper designs that are meant to

cancel out an initial swinging disturbance such as [47] and [48]. These works focused on crane applications although the technique should likely be applicable to quadotor-slung load systems. While this new form of input shaper was demonstrated to be effective in experimental setups, it remains an open loop trajectory generation technique for swing management. It will thus not extend well to cancelling a swinging disturbance while attempting to hold a given position such as during a landing operation. In this scenario, an active feedback controller would likely be preferable due to the risk of a payload collision or cable entanglement with the surrounding environment.

Another set of papers within this category attempt to achieve trajectory tracking in spite of the swinging motion of the payload [7] [30] [40] [41]. For these controllers, the tension from the cable attached to the payload is treated as a disturbance force to be estimated and cancelled. This differs from controllers in the third category which will deliberately deviate from their prescribed flight trajectory in order to try to actively cancel out swinging motions of their payload. In [41], the quadrotor-slung load system is treated as an unbalanced quadrotor problem. An adaptive controller is developed to track aggressive maneuvers while managing the changing center of gravity of the system. The proposed controller was not able to dampen the swinging motion of the payload though and the authors used dynamic programming to generate swing free flight trajectories in order to suppress swinging. In [7], the presence of the slung load is treated as an external disturbance to be rejected during the course of a trajectory tracking operation. The proposed controller is later applied to a delivery scenario in a simulated forest environment [1]. In [40], the proposed controller estimates the force and moment acting on the quadrotor due to the tensile force in the cable and attempts to cancel their effect with an adaptive controller. The resulting simulated behaviour of the quadrotor-slung load system shows good stability even in the presence of payload swinging. In [30], the authors propose a controller that demonstrates robust trajectory tracking in spite of the swinging of the payload. The authors go on to demonstrate the effect of tracking an input-shaped trajectory for swing management and highlight how active swing management could be achieved by including feedback into the proposed controller design.

A major question regarding the controllers presented in this category is how well they will be able to react to an external swinging disturbance on the payload compared to a controller that is actively trying to regulate the payload to not swing. In [7] the authors demonstrated the ability to dampen an initial swinging disturbance acting on the load. In [41] however, experimental results demonstrate that the controller by itself was ineffective at damping swinging motion. Similarly, in [45], a quadrotor-slung load controller without payload stabilization is simulated and is shown to be unable to effectively dampen

the swinging motion. The mass of the payload relative to the quadrotor likely plays an important role in this swing damping behaviour. In [41] for instance, the payload had a mass of 47g while the AscTec Hummingbird used for testing has a minimum takeoff mass of 510g [49]. A swinging slung load of sufficient mass will be able to impart some of its kinetic energy onto the connected quadrotor by pulling it away from its desired position. The quadrotor's controller may dissipate this energy by counteracting the tension force that pulls it aside in a process that could eventually cause the payload to stop oscillating. This effect is not guaranteed to occur though and likely depends on the interplay between the controller's gain as well as the cable length and mass of the payload. In [45] their simulation involved a 408g quadrotor transporting a 100g load and yet there was no effective swing damping. From a practical standpoint, the main difference between controllers in this category and the third category is that controllers in this category will tend to move opposite the direction of the swinging payload while controllers in the later will try to catch up to the swinging payload. As noted in [50] though, having the quadrotor apply a force in the opposite direction of the swinging of the payload can increase the swinging motion leading to a chain reaction that creates instability in the overall system. Given that we would ultimately like our controller to reliably dampen swinging disturbances acting on the payload regardless of the parameters of our slung load system, it appears that controllers in this category would not necessarily be a good starting point for our controller design.

3.1.3 Quadrotor Trajectory Following + Payload Swing Feedback Controllers

The third category of controllers incorporate feedback control over the swinging motion of the payload to dampen its motion while tracking a quadrotor flight trajectory. Neither [42], [43], [44], [45] or [46] employ input-shaped flight trajectories to help manage payload swinging and instead relied solely on feedback control. In [44], a H_∞ controller is developed for trajectory tracking and is then augmented with an additional control law to dampen the swinging motion of the payload. The augmented controller is shown in simulation to significantly improve the swing damping of a lightweight payload subject to wind effects. In [42], the authors use a nonlinear control law for managing the swinging motion of the payload in an autonomous helicopter-slung load system. The payload position relative to the helicopter is projected onto the XY plane of the inertial frame and the resulting vector is decomposed into components along each axis to determine control accelerations using a PD control law. The proposed controller estimates the steady state deflection due to wind drag and treats this as a reference swing angle. A similar

approach is presented in [45] for a quadrotor-slung load system. In [45], the proposed controller is compared against a simpler design without swing stabilization and is shown to outperform the latter in simulation. Disturbance rejection was also demonstrated in with hovering flight tests. A more complex controller is presented in [43] and [46]. In these papers, the cable is discretized into a series of interconnected links as done in [39] and the orientation and angular velocity of each link is used to create a PD control law for damping the swinging motion of the slung load. The authors prove in [46] that their proposed control law is exponentially stable for the hanging equilibrium and demonstrate its effectiveness in both simulation and with an actual drone. Obtaining measurements of the cable at multiple locations along its length can be tricky to implement in an actual slung load system though as discussed in [39]. Indeed in [46], the authors simplified their model for implementation on an actual drone such that the cable was treated as a single rigid link.

Within this third category, a popular approach found in the literature is to track input-shaped trajectories while employing delayed feedback to eliminate swinging motion in the slung load as done in [21], [24], [25] and [26]. Delayed feedback, such as that employed in [24], computes a quadrotor motion to cancel the payload swinging as defined by the angle between the cable and the z axis of the inertial frame. Additional details regarding this controller and its tuning can be found in [21]. This motion is then appended to the desired state obtained from the input-shaped trajectory to obtain a new desired state for the quadrotor position controller to track. An advantage of the proposed controller is that its tuning parameters are independent of the cable length. The authors in [24] though noted that the delayed feedback and input shaping strategies appeared to conflict with each other. The proposed solution was to delay the activation of the delayed feedback control element until after the input shaper has induced the initial swinging motion in the payload. A similar issue was noted in [25]. In this paper, the authors used a nonlinear trajectory tracking controller for a quadrotor with the desired motion computed using a combination of input shaping and delayed feedback. The proposed solution in [25] to the conflict between the input-shaped trajectory and the feedback controller was to gain schedule the delayed feedback element based on the desired acceleration of the input-shaped trajectory. Functionally this resulted in a system where the delayed feedback system was only active during constant velocity portions of the desired motion trajectory.

The conflict between a swing suppression strategy like delayed feedback and input shaping noted in [24] and [25] makes sense because input shaping is based on the concept of inducing and then eliminating swinging via destructive interference as discussed in Section 2.2. As was shown in Section 2.3.3, tracking an input-shaped trajectory can

induce significant swinging motion during accelerating phases of the trajectory but results in minimal swinging motion during non-accelerating phases. Conventional swing suppression control strategies however treat any swinging motion as an error and seek to correct it by modifying the motion of the drone. This can degrade the effectiveness of using input-shaped trajectories since they are only effective for swing management when accurately tracked. We thus need to effectively instruct the swing suppression element of the controller to ignore the swinging motions of the payload that are caused by tracking our input-shaped trajectory. Both [24] and [25] accomplish this by effectively turning off all swing feedback control during accelerating portions of the input-shaped trajectory. This strategy has some significant drawback though as the flight controller effectively degrades its ability to react to external disturbances acting on the payload for portions of its flight. If the payload were to collide with its environment just as the drone began an accelerating phase of its prescribed trajectory, the swing feedback element of the controller would only properly react to this once that accelerating phase was complete. This delay could prove critical especially if longer input-shaped accelerating maneuvers are employed. Accelerating maneuvers such as transitioning from hover to forward flight or performing turns could also pose an increased risk for collisions between the payload and its environment. An alternate strategy would be to supply the controller with both an input-shaped quadrotor flight trajectory and the corresponding swing trajectory. Thus instead of instructing the drone to ignore swinging motion over certain time intervals, the controller could naturally ignore swinging that closely matches the given reference while reacting to swinging motion that deviates from it for any reason. The controller would thus maintain the ability to react to an external disturbance at any point during its input-shaped flight trajectory instead of needing to wait to complete an accelerating portion of the motion before reacting to the swinging disturbance. This idea forms the core concept behind the proposed controller in this chapter.

3.2 Quadrotor-Slung Load System Simulation Model

In order to assess the performance of our proposed flight controller, we first develop a model for a quadrotor-slung load system. This model will then be used to create a simulator for our system in MATLAB to enable simulation testing of our novel controller as well as baseline controllers. Modelling of slung load systems is discussed in depth in [21] as well as in [51]. One main area in which models diverge is how they handle the cable(s) connecting the payload and the quadrotor. A conventional approach for single cable systems is to treat the cable as an inelastic link connecting the center of mass of

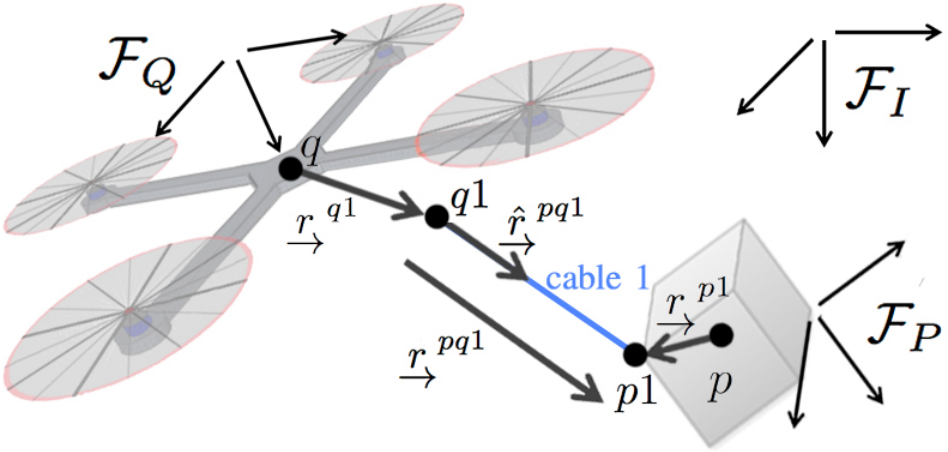

FIGURE 3.1: Model Frame Setup and Nomenclature

the quadrotor to a point mass payload such as in [25] [36] [37] [38] [44]. This approach neglects any effect that the payload may have on the vehicle's attitude dynamics. This effect is captured in [26], [40], [45], [52] where the authors use an inelastic model but allow the cable mount to be offset from the drone's center of mass. Overall though, the inelastic cable assumption is potentially problematic in that it could artificially generate compressive loads acting on the payload during maneuvers. Using inelastic cables can also become overly restrictive if we consider a scenario with multiple cable connecting a quadrotor drone to its payload. A more advanced approach involves discretizing the cable into a series of elastic or inelastic links to form a chain-like structure as done in [39] and [43]. This approach captures the potential slack cable behaviour in the system but adds computational complexity to the model.

In this work we model each cable as having extensible elasticity and damping while constraining the tension force to always be positive. The payload is treated as a rigid body and cable mounting points can be assigned to any point on the drone or payload. The proposed model is based on [51] and can accommodate multiple cables linking the quadrotor to its payload to allow for studying multi-cable systems. We will present the equations of motion for a multi-cable system however in this work we only focus on a single-cable configuration. Figure 3.1, reprinted from [2], presents an overview of the frames of reference and nomenclature used for the model.

In Figure 3.1, $\mathcal{F}_I$ is the inertial frame while $\mathcal{F}_P$ and $\mathcal{F}_Q$ are body-fixed frames connected to the payload and quadrotor. We define the direction cosine matrices $\mathbf{R}_{QI}$ and $\mathbf{R}_{PI}$ to denote the attitude of the quadrotor and payload frames relative to the inertial frame. Throughout this chapter, subscripts on boldface lowercase letters will be used to clarify which frame vectors are being resolved in. The attitude of the quadrotor and payload

frames can also be parameterized with quaternions $\mathbf{q}^Q$ and $\mathbf{q}^P$. The following equations summarize how to convert from a quaternion to a direction cosine matrix.

$$\mathbf{q}^Q = \begin{bmatrix} \eta^Q \\ \boldsymbol{\epsilon}^Q \end{bmatrix} = \begin{bmatrix} \cos(\theta^Q/2) \\ \sin(\theta^Q/2)\hat{\mathbf{a}}^Q \end{bmatrix} \tag{3.1}$$

$$\mathbf{R}_{IQ} = (1 - 2\boldsymbol{\epsilon}^{Q\mathsf{T}}\boldsymbol{\epsilon}^Q)\mathbf{1} + 2\boldsymbol{\epsilon}^Q\boldsymbol{\epsilon}^{Q\mathsf{T}} + 2\eta^Q\boldsymbol{\epsilon}^{Q\times} \tag{3.2}$$

$$\mathbf{R}_{QI} = \mathbf{R}_{IQ}{}^{\mathsf{T}} \tag{3.3}$$

where $\hat{\mathbf{a}}^Q$ and θ^Q define the axis of rotation and angle of rotation defining the orientation of $\mathcal{F}_Q$ relative to $\mathcal{F}_I$. $\mathbf{1}$ denotes the three by three identity matrix and the $\times$ superscript denotes the conversion of a column vector to a cross product matrix for example

$$\boldsymbol{\epsilon}^{Q\times} = \begin{bmatrix} \epsilon_1^Q \\ \epsilon_2^Q \\ \epsilon_3^Q \end{bmatrix}^\times = \begin{bmatrix} 0 & -\epsilon_3^Q & \epsilon_2^Q \\ \epsilon_3^Q & 0 & -\epsilon_1^Q \\ -\epsilon_2^Q & \epsilon_1^Q & 0 \end{bmatrix} \tag{3.4}$$

We denote the position and velocity of the centers of mass q and p of the quadrotor and payload resolved in the inertial frame $\mathcal{F}_I$ as $\mathbf{r}^q$, $\mathbf{v}^q$, $\mathbf{r}^p$ and $\mathbf{v}^p$ respectively. $\mathbf{r}_Q^{q1}$ and $\mathbf{r}_P^{p1}$ denote the position of attachment points $q1$ and $p1$ for cable 1 on the quadrotor and payload relative to the center of masses for each body. Each relative position vector is resolved in the appropriate body fixed frame. $\mathbf{r}_I^{pq1}$ is the vector linking mounting points $q1$ and $p1$ resolved in frame $\mathcal{F}_I$. Its associated unit vector resolved in $\mathcal{F}_I$ is denoted $\hat{\mathbf{r}}_I^{pq1}$.

Using this nomenclature we can define the Newton–Euler equations of motion for both the quadrotor and the payload.

$$m^q\dot{\mathbf{v}}^q = \mathbf{f}^q, \quad m^p\dot{\mathbf{v}}^p = \mathbf{f}^p \tag{3.5}$$

$$\mathbf{J}^q\dot{\boldsymbol{\omega}}^Q + \boldsymbol{\omega}^{Q\times}\mathbf{J}^q\boldsymbol{\omega}^Q = \mathbf{m}^q, \quad \mathbf{J}^p\dot{\boldsymbol{\omega}}^P + \boldsymbol{\omega}^{P\times}\mathbf{J}^p\boldsymbol{\omega}^P = \mathbf{m}^p \tag{3.6}$$

where m^q and m^p are the mass of the quadrotor and payload. $\mathbf{J}^q$ and $\mathbf{J}^p$ are the inertia matrices for the second moment of mass of the quadrotor and payload relative to their center of mass resolved in the body frame of each body. $\boldsymbol{\omega}^Q$ and $\boldsymbol{\omega}^P$ are the angular velocity of $\mathcal{F}_Q$ and $\mathcal{F}_P$ relative to $\mathcal{F}_I$ resolved in their respective body fixed frames. $\mathbf{f}^q$ and $\mathbf{f}^p$ denote the overall force vector acting on the quadrotor and payload resolved in the inertial frame. $\mathbf{m}^q$ and $\mathbf{m}^p$ denote the overall moment acting on the quadrotor and payload resolved in their respective body frames $\mathcal{F}_Q$ and $\mathcal{F}_P$.

For this model, we consider forces and moments acting on the quadrotor and payload due to gravity, the tension force in each cable and the thrusters. We neglect aerodynamic effects for simplicity. We can thus express the overall force vectors acting on the quadrotor and payload as follows.

$$\mathbf{f}^q = m^q\mathbf{g} + \mathbf{R}_{IQ}\mathbf{f}_Q^T + \mathbf{f}^C \quad , \quad \mathbf{f}_Q^T = F^{thrust}\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \quad , \quad \mathbf{f}^p = m^p\mathbf{g} - \mathbf{f}^C \tag{3.7}$$

where $\mathbf{g}$ is the gravitational acceleration vector resolved in $\mathcal{F}_I$. $\mathbf{f}_Q^T$ is the thrust force acting on the quadrotor resolved in the quadrotor frame $\mathcal{F}_Q$. F^{thrust} is a positive scalar denoting the total thrust force acting on the quadrotor. $\mathbf{f}^C$ is the sum of the tension force in each cable within the system. For this thesis, we chose to model the cables as a single massless link with extensible elasticity and damping while imposing a constraint to avoid compressive loads. While this approach does allow the modelled cables to lose tension, it does not capture the resulting behaviour of the slack cable. This approach also assumes negligible inertia for the cable. We deem this assumption to be reasonable in this situation since our envisioned test setup will involve moderately sized quadrotor drones using lightweight cables likely made from fishing line. Using these assumptions we obtain the following results.

$$\mathbf{f}^C = \sum_{N=1}^{Ntot} T_N\hat{\mathbf{r}}_I^{pqN} \tag{3.8}$$

$$T_N = max(0, \quad k_N(l_N - l_N^0) + b_N\dot{l}_N) \tag{3.9}$$

where k_N, b_N and l_N^0 are the spring constant, damping constant and unstretched length of the Nth cable.

$$l_N = \sqrt{\mathbf{r}_I^{pqN\intercal}\mathbf{r}_I^{pqN}} \tag{3.10}$$

$$\mathbf{r}_I^{pqN} = \mathbf{r}^p + \mathbf{R}_{IP}\mathbf{r}_P^{pN} - \mathbf{R}_{IQ}\mathbf{r}_Q^{qN} - \mathbf{r}^q \tag{3.11}$$

$$\hat{\mathbf{r}}_I^{pqN} = \frac{\mathbf{r}_I^{pqN}}{l_N} \tag{3.12}$$

$$\dot{l}_N = (\mathbf{v}^{pN} - \mathbf{v}^{qN})^\intercal\hat{\mathbf{r}}_I^{pqN} \tag{3.13}$$

$$\mathbf{v}^{qN} = \mathbf{v}^q + \mathbf{R}_{IQ}(\boldsymbol{\omega}^{Q\times}\mathbf{r}_Q^{qN}) \quad , \quad \mathbf{v}^{pN} = \mathbf{v}^p + \mathbf{R}_{IP}(\boldsymbol{\omega}^{P\times}\mathbf{r}_P^{pN}) \tag{3.14}$$

Similarly, we can compute the moments acting on the quadrotor and payload as follows.

$$\mathbf{m}^q = \mathbf{m}_Q^T + \mathbf{m}^{Cq}, \quad \mathbf{m}^p = \mathbf{m}^{Cp} \tag{3.15}$$

where $\mathbf{m}_Q^T$ is the moment acting on the quadrotor resolved in $\mathcal{F}_Q$ generated by the the input thrust. $\mathbf{m}^{Cq}$ and $\mathbf{m}^{Cp}$ are the moment exerted on the quadrotor and payload due to the sum of all tension forces in the system. This term in essence allows the payload to affect the attitude dynamics of the quadrotor. This effect is often neglected in simulation models but it could have a significant effect on the drone's flight performance if the payload is not perfectly attached at the center of gravity of the drone. Using the values from equations (3.9) and (3.12) we can compute these moments as follows.

$$\mathbf{m}^{Cq} = \mathbf{R}_{QI}\left(\sum_{N=1}^{Ntot}((\mathbf{R}_{IQ}\mathbf{r}_Q^{qN})^{\times}(T_N\hat{\mathbf{r}}_I^{pqN}))\right) \tag{3.16}$$

$$\mathbf{m}^{Cp} = \mathbf{R}_{PI}\left(\sum_{N=1}^{Ntot}((\mathbf{R}_{IP}\mathbf{r}_P^{pN})^{\times}(-T_N\hat{\mathbf{r}}_I^{pqN}))\right) \tag{3.17}$$

We complete the required equations for our simulation with the time derivative of the quaternions

$$\dot{\mathbf{q}}^Q = \frac{1}{2}\mathbf{q}^Q \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}^Q \end{bmatrix}, \quad \dot{\mathbf{q}}^P = \frac{1}{2}\mathbf{q}^P \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}^P \end{bmatrix} \tag{3.18}$$

where $\otimes$ denotes quaternion multiplication. For example

$$\mathbf{q}^A \otimes \mathbf{q}^B = \begin{bmatrix} \eta^A \\ \epsilon_1^A \\ \epsilon_2^A \\ \epsilon_3^A \end{bmatrix} \otimes \begin{bmatrix} \eta^B \\ \epsilon_1^B \\ \epsilon_2^B \\ \epsilon_3^B \end{bmatrix} = \begin{bmatrix} \eta^A & -\epsilon_1^A & -\epsilon_2^A & -\epsilon_3^A \\ \epsilon_1^A & \eta^A & -\epsilon_3^A & \epsilon_2^A \\ \epsilon_2^A & \epsilon_3^A & \eta^A & -\epsilon_1^A \\ \epsilon_3^A & -\epsilon_2^A & \epsilon_1^A & \eta^A \end{bmatrix} \begin{bmatrix} \eta^B \\ \epsilon_1^B \\ \epsilon_2^B \\ \epsilon_3^B \end{bmatrix} \tag{3.19}$$

As previously discussed in Section 2.4, we employ a basic point mass pendulum model to predict the natural frequency of oscillation of our slung payload. For this work, we will primarily focus on a single cable system and we thus choose l to be the distance between the cable mounting point on the quadrotor and the payload center of mass.

$$l = l_1^0 + \| \underrightarrow{r}^{p1} \|_2 \quad , \quad T_d = 2\pi\sqrt{l/g} \tag{3.20}$$

3.2.1 Simulator setup

Equations (3.5) through (3.18) were implemented in MATLAB in order to generate a simulation environment for testing flight controllers. Any time we simulated a flight, we first performed an initial hovering simulation to establish the stretching required in

the cable(s) that support the payload in a hovering equilibrium. This becomes especially useful when multi-cable systems are being simulated. We started this hovering simulation with the quadrotor center of mass at the origin and our payload located some distance above the expected hanging equilibrium. We then set $F^{thrust} = (m^q + m^p)g$ and $\mathbf{m}_Q^T = -\mathbf{m}^{Cq}$ and simulated the resulting motion of the quadrotor-slung load system. During this simulation, the payload naturally settles into an equilibrium state where the cable(s) that hold it to the quadrotor become taut. The damping in the cable(s) helps to ensure that this initial simulation converges quickly. From this hovering equilibrium solution, we extract the relative position between the payload and the quadrotor. We then simulate the actual flight by prescribing a motion trajectory for the quadrotor and allowing our flight controller to compute values for F^{thrust} and $\mathbf{m}_Q^T$. For our simulated flight, the quadrotor initial state is dictated by our desired trajectory. The initial position of the payload is computed using the relative position value we found from the hovering equilibrium. We can simulate flights with a swinging disturbance by adjusting the initial velocity of the payload.

All code elements were written in MATLAB scripts and the differential equations were solved using the ode15s solver with parameters $RelTol = 1e-6$ and $AbsTol = 1e-8$. The simulator was set up to generate 200 data points per second of simulated flight time. The core state of our ode solver being integrated consisted of $\mathbf{r}^q$, $\mathbf{v}^q$, $\mathbf{r}^p$, $\mathbf{v}^p$, $\boldsymbol{\omega}^Q$, $\boldsymbol{\omega}^P$, $\mathbf{q}^Q$ and $\mathbf{q}^P$. Additional terms were added to accommodate integral control elements.

As part of our simulations, we also incorporated measurement noise into the payload position and velocity signals being used for our controllers. To implement this, a set of Gaussian noise data is pre generated for each time step in the simulation based on prescribed mean and variance values. This noise data is saved and reused for subsequent simulations to allow for comparison of multiple controllers with the same noise effects. Since ode15s is a variable step solver, we had MATLAB interpolate the provided noise data in order to establish the required values. We also lowpass filtered the generated noise using a discretized first order filter with cutoff frequency 31.8Hz to ensure that the values would not create issues within our solver. An overview of this simulator is provided in Algorithm 3.

3.3 Geometric Controller

The goal of our flight controller is to accurately track a prescribed input-shaped flight trajectory while also being able to react to unexpected swinging disturbances acting on

Algorithm 3: Quadrotor-Slung Load Simulator Overview

input : Quadrotor-Slung Load System parameters (see Table 3.1), *ControlLaw*(system state, desired state) to solve for F^{thrust} and $\mathbf{m}_Q^T$, Quadrotor initial state ($\mathbf{r}_0^q$, $\mathbf{v}_0^q$, $\boldsymbol{\omega}_0^Q$, $\mathbf{q}_0^Q$), Payload initial velocity ($\mathbf{v}_0^p$), Initial guess for settled position of payload relative to quadrotor ($\mathbf{r}_{guess}^p$), Precomputed payload position and velocity signal noise (*noise*)

Function `ode15s()`:

system state: $(\mathbf{r}^q, \mathbf{v}^q, \mathbf{r}^p, \mathbf{v}^p, \boldsymbol{\omega}^Q, \boldsymbol{\omega}^P, \mathbf{q}^Q, \mathbf{q}^P)$
initial conditions: $(\mathbf{0}, \mathbf{0}, \mathbf{r}_{guess}^p, \mathbf{0}, \mathbf{0}, \mathbf{0}, [1,0,0,0], [1,0,0,0])$
differential equations: (3.5) through (3.18)
properties: $RelTol = 1e-6$, $AbsTol = 1e-8$
system inputs: $F^{thrust} = (m^q + m^p)g$, $\mathbf{m}_Q^T = -\mathbf{m}^{Cq}$
return *Equilibrium State*

Using *Equilibrium State* measure distance $\mathbf{d} = \mathbf{r}^p - \mathbf{r}^q$

Solve for the starting position of the payload $\mathbf{r}_0^p = \mathbf{r}_0^q + \mathbf{d}$

Function `ode15s()`:

system state: $(\mathbf{r}^q, \mathbf{v}^q, \mathbf{r}^p, \mathbf{v}^p, \boldsymbol{\omega}^Q, \boldsymbol{\omega}^P, \mathbf{q}^Q, \mathbf{q}^P)$
initial conditions: $(\mathbf{r}_0^q, \mathbf{v}_0^q, \mathbf{r}_0^p, \mathbf{v}_0^p, \boldsymbol{\omega}_0^Q, \mathbf{0}, \mathbf{q}_0^Q, [1,0,0,0])$
differential equations: (3.5) through (3.18)
properties: $RelTol = 1e-6$, $AbsTol = 1e-8$
system inputs: F^{thrust}, $\mathbf{m}_Q^T = ControlLaw$(system state + *noise*, desired state)
return *MotionTimeHistory*

Post process *MotionTimeHistory* to capture control effort, error, forces etc.

output: *MotionTimeHistory*, post processed data, plots of data

the payload. As discussed in Section 3.1.3, the key element that separates our proposed controller from other such designs is that our controller uses both the desired input-shaped flight trajectory and the corresponding swinging trajectory in its computations. This corresponding swinging trajectory is computed within the controller during each loop. Thus by comparing the actual swinging motion to this swing trajectory our controller can differentiate between swinging caused by accurate trajectory tracking versus that caused by disturbances. Our controller only tries to eliminate the latter. This differs from the more conventional approach discussed in Section 3.1.3 where the controller effectively tries to eliminate all swinging motion observed in the payload while tracking a prescribed trajectory. A preliminary version of our controller is presented and analyzed in [2].

Figure 3.2, reprinted from [2], presents an overview of the core elements of the proposed controller that we developed. At every time step the controller receives the desired state of the quadrotor from our trajectory generator as discussed in Section 2.3.2. This information gets passed to the Oscillation Controller and Position Controller elements

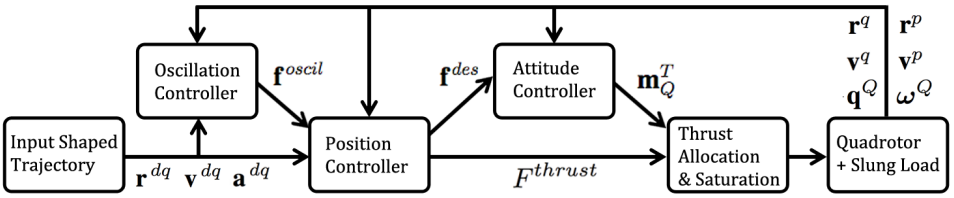

FIGURE 3.2: Proposed Flight Controller Overview

which ultimately determine a required force vector resolved in the inertial frame $\mathbf{f}^{des}$ that the drone must exert to 1) eliminate unwanted swinging motion and 2) track the desired flight trajectory. The Oscillation Controller element is also responsible for predicting the swinging motion induced by the prescribed input-shaped trajectory. The desired thrust vector is then decomposed into a heading and an overall thrust magnitude. The Attitude Controller in turn computes a required moment $\mathbf{m}_Q^T$ that the drone should exert to reach the desired heading. The overall thrust force and moment are then handed over to a Thrust Allocation and Saturation module which determines the required thrust that each motor should produce. The controller receives feedback information in the form of the measured position and velocity of the quadrotor and its payload as well as the quadrotor's attitude and angular velocity.

The main limitation of the proposed controller is that it is only meant to track input-shaped quadrotor flight trajectories. One could replace the swing prediction element of this controller with a desired swing trajectory generated by alternate means in order to apply the proposed controller to different cases. For example, a reference swinging trajectory could be generated in order to achieve a payload throwing motion as described in [12] and [13]. We feel that this is a reasonable limitation however considering the advantages that input-shaped trajectories present. Due to the underactuated nature of quadrotor-slung load system, anytime the quadrotor performs an accelerating maneuver, this will inevitably induce some level of swinging motion in the slung load. The best case scenario from a swing mitigation standpoint would be that once the quadrotor drone stops accelerating and either returns to rest or reaches a steady state velocity, no more residual swinging motion exists in the payload. As demonstrated in the Chapter 2, input-shaped motion trajectories achieve this exact behaviour.

3.3.1 Oscillation Controller

The Oscillation Controller is a geometric PD control law based on [38]. In [38], the authors use this geometric control law as part of a flight controller which receives as an input the

desired payload flight trajectory. Their controller converts the payload trajectory into a desired swing angle and uses this control law to have the quadrotor cause this desired swinging motion in the payload. As shown in Figure 3.2, this control element takes in information about the state of the system and the desired motion trajectory and computes a desired force vector $\mathbf{f}^{oscil}$. We use the following relations to compute $\mathbf{f}^{oscil}$.

$$\mathbf{f}^{oscil} = \mathbf{K}_p^{oscil}\boldsymbol{\epsilon}_p^{oscil} + \mathbf{K}_d^{oscil}\boldsymbol{\epsilon}_d^{oscil} \tag{3.21}$$

$$\boldsymbol{\epsilon}_p^{oscil} = (\hat{\mathbf{c}}^{\times})^2\hat{\mathbf{c}}^d \tag{3.22}$$

$$\boldsymbol{\epsilon}_d^{oscil} = \dot{\hat{\mathbf{c}}} - (\hat{\mathbf{c}}^{d\times}\dot{\hat{\mathbf{c}}}^d)^{\times}\hat{\mathbf{c}} \tag{3.23}$$

$$\hat{\mathbf{c}} = \frac{\mathbf{c}}{\sqrt{\mathbf{c}^\intercal\mathbf{c}}} \quad , \quad \hat{\mathbf{c}}^d = \frac{\mathbf{c}^d}{\sqrt{\mathbf{c}^{d\intercal}\mathbf{c}^d}} \tag{3.24}$$

$$\dot{\hat{\mathbf{c}}} = \frac{\dot{\mathbf{c}}}{\sqrt{\mathbf{c}^\intercal\mathbf{c}}} - \frac{\mathbf{c}\mathbf{c}^\intercal\dot{\mathbf{c}}}{\left(\sqrt{\mathbf{c}^\intercal\mathbf{c}}\right)^3} \tag{3.25}$$

$$\mathbf{c} = \mathbf{r}^p - \mathbf{r}^q \quad , \quad \mathbf{c}^d = \mathbf{r}^{dp} - \mathbf{r}^{dq} \tag{3.26}$$

$$\dot{\mathbf{c}} = \mathbf{v}^p - \mathbf{v}^q \quad , \quad \dot{\mathbf{c}}^d = \mathbf{v}^{dp} - \mathbf{v}^{dq} \tag{3.27}$$

$\mathbf{f}^{oscil}$ is the thrust force vector resolved in the inertial frame that the quadrotor should apply to correct the swinging motion of the payload. $\mathbf{K}_p^{oscil}$ and $\mathbf{K}_d^{oscil}$ are diagonal control gain matrices. $\hat{\mathbf{c}}$ is a unit vector resolved in the inertial frame pointing from the measured quadrotor position $\mathbf{r}^q$ to that of the payload $\mathbf{r}^p$. $\hat{\mathbf{c}}$ thus approximates the orientation of the assumed taut cable. The unit vector $\hat{\mathbf{c}}^d$ represents the desired orientation of this cable. The desired position and velocity of the quadrotor ($\mathbf{r}^{dq}$, $\mathbf{v}^{dq}$) are computed from the prescribed input-shaped trajectory as discussed in Section 2.3.2. The controller also requires the expected position and velocity of the payload ($\mathbf{r}^{dp}$, $\mathbf{v}^{dp}$). These elements are computed by a swing prediction element in our controller detailed in the subsequent section.

3.3.1.1 Swing Prediction

Our controller has a prediction element within it to compute the expected swinging motion that a pendulum would have if its upper mounting point perfectly tracked the prescribed input-shaped flight trajectory. This processes is illustrated visually in Figure 3.3.

Mathematically, this prediction step is accomplished using a set of differential equations. For our MATLAB simulation, these equations are numerically integrated as part of the main set of states in the simulator. For an actual drone, the numerical integration step is performed onboard the drone with each loop of the flight controller.

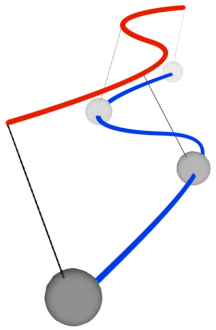

FIGURE 3.3: Predicted Swinging Motion (blue) due to Prescribed Input-Shaped Flight Trajectory (red)

For this work we use a second order system to predict the swinging motion of the pendulum along the x and y axes. We then impose two constraint equations to solve for the motion along the z axis. The following differential equations are used for this prediction.

$$\mathbf{r}^{dq} = \begin{bmatrix} x^{dq} \\ y^{dq} \\ z^{dq} \end{bmatrix}, \quad \mathbf{r}^{dp} = \begin{bmatrix} x^{dp} \\ y^{dp} \\ z^{dp} \end{bmatrix}, \quad \mathbf{v}^{dp} = \begin{bmatrix} \dot{x}^{dp} \\ \dot{y}^{dp} \\ \dot{z}^{dp} \end{bmatrix} \tag{3.28}$$

$$\frac{d}{dt}(x^{dp}) = \dot{x}^{dp}, \quad \frac{d}{dt}(y^{dp}) = \dot{y}^{dp} \tag{3.29}$$

$$\frac{d}{dt}(\dot{x}^{dp}) = \left(\frac{2\pi}{T_d}\right)^2 (x^{dq} - x^{dp}), \quad \frac{d}{dt}(\dot{y}^{dp}) = \left(\frac{2\pi}{T_d}\right)^2 (y^{dq} - y^{dp}) \tag{3.30}$$

where $\mathbf{r}^{dq}$ is the desired quadrotor position resolved in the inertial frame defined by evaluating the prescribed input-shaped flight trajectory at the given point in time. $\mathbf{r}^{dp}$ and $\mathbf{v}^{dp}$ are the predicted payload position and velocity resolved in the inertial frame to be computed. With each numerical integration step, new values for x^{dp}, y^{dp}, $\dot{x}^{dp}$ and $\dot{y}^{dp}$ are obtained. The initial conditions used for this integration process are $x^{dp}(t=0) = x^{dq}(t=0)$ and $\dot{x}^{dp}(t=0) = \dot{x}^{dq}(t=0)$ for both the x and y axes. We impose the following constraint equations to solve for z^{dp} and $\dot{z}^{dp}$.

$$(x^{dp} - x^{dq})^2 + (y^{dp} - y^{dq})^2 + (z^{dp} - z^{dq})^2 = l^2 \tag{3.31}$$

$$(\mathbf{r}^{dp} - \mathbf{r}^{dq})^\intercal (\mathbf{v}^{dp} - \mathbf{v}^{dq}) = 0 \tag{3.32}$$

The value of T_d and l in equations (3.30) and (3.31) are chosen to match the values used for creating the input shaper in equation (3.20).

For times when the drone is no longer flying an accelerating input-shaped trajectory we no longer need to use the above equations and can simply set

$$\mathbf{r}^{dp} = \begin{bmatrix} x^{dq} \\ y^{dq} \\ z^{dq} + l \end{bmatrix}, \quad \mathbf{v}^{dp} = \mathbf{v}^{dq} \tag{3.33}$$

This corresponds to the situation where the payload is directly underneath the drone as seen in the inertial frame and there is no relative velocity between the payload and quadrotor.

Initially while developing this prediction step more complicated models were used. When tested though, we found that these models had a tendency to predict a payload trajectory that had some residual payload swinging. In Section 2.3.3 for instance, we presented equations for an inelastic cable model in order to test the effectiveness of our input shaper. We could easily run this model onboard our quadrotor drone to predict the swinging motion of the payload due to our input-shaped motion trajectory. Notice though that in Figure 2.8 when we use this model with an input-shaped motion trajectory we still get some residual swinging amplitude. This is problematic since we would be using this swinging motion as a reference for our controller to track. In turn, that means that our controller may be trying to induce a slight residual oscillation in our payload which is not desirable. In essence, we would like a prediction model that is accurate but that also converges to zero residual swinging motion so that we can use its prediction values as a tracking reference for our controller. We found that using a second order model with an undamped natural frequency and damping ratio that matched those used to design our input shaper produced exactly these kinds of results.

As shown in Figures 2.3 and 2.4, both the ZV and two-hump EI input shapers achieve a PRV of 0 when the design angular velocity and damping ratio perfectly match those of the second order linear system subjected to the input shaper's impulses. This means that our prediction model will have no residual swinging motion if it were subjected to the impulses for either input shaper. Furthermore, because our input to this second order system is a motion trajectory that has been convolved with our input shaper we know from [17] that our output should also have no residual swinging motion.

3.3.2 Position Controller

The proposed Position Controller is based on [11] and [42] and computes a desired thrust vector $\mathbf{f}^{des}$ resolved in the inertial frame. In keeping with the approach taken in [42]

this thrust vector is a weighted sum of the feedback force computed from the Oscillation Controller computed with equation (3.21) and a feedback control law for position tracking. Unlike [42] though, we employ a PID control law instead of a PD control law in order to manage steady state altitude errors.

$$\mathbf{f}^{des} = k^{path}\mathbf{f}^{path} + k^{oscil}\mathbf{f}^{oscil} + \mathbf{f}^{weight} + \mathbf{f}^{acc} \tag{3.34}$$

$$\mathbf{f}^{path} = \mathbf{K}_p^{path}(\mathbf{r}^{dq} - \mathbf{r}^q) + \mathbf{K}_d^{path}(\mathbf{v}^{dq} - \mathbf{v}^q) + \mathbf{K}_i^{path}\int_0^t (\mathbf{r}^{dq}(\tau) - \mathbf{r}^q(\tau))d\tau \tag{3.35}$$

$$\mathbf{f}^{weight} = -(m^p + m^q)\mathbf{g} \tag{3.36}$$

$$\mathbf{f}^{acc} = (m^p + m^q)\mathbf{a}^{dq} \tag{3.37}$$

where k^{path} and k^{oscil} are scalars which can be used to emphasize position tracking or oscillation management. In [42], these scalars are set to different values during different portions of the flight envelope to alter the performance of the system. For our controller, we keep all gains fixed throughout a given flight. $\mathbf{K}_p^{path}$, $\mathbf{K}_d^{path}$ and $\mathbf{K}_i^{path}$ are diagonal control gain matrices for the PID control law. This marks a slight departure from controllers typically found in the literature which would assign a single scalar for each gain. We opted for a diagonal matrix form here to allow for assigning different gains for horizontal and vertical motions of the drone. $\mathbf{a}^{dq}$ is the desired quadrotor acceleration vector computed from the input-shaped flight trajectory for the drone.

Once computed, $\mathbf{f}^{des}$ can be projected onto the current heading of the quadrotor resolved in $\mathcal{F}_I$, $\hat{\mathbf{h}}$, to establish the thrust magnitude F^{thrust}.

$$\hat{\mathbf{h}} = -\mathbf{R}_{IQ}([0 \quad 0 \quad 1]^\mathsf{T}), \quad F^{thrust} = \mathbf{f}^{des^\mathsf{T}}\hat{\mathbf{h}} \tag{3.38}$$

During initial development we tried to set F^{thrust} equal to the magnitude of the $\overrightarrow{f^{des}}$ vector but this produced unstable results. The approach given in (3.38) is typically used for quadrotor drones such as in the controller presented in [11]. Throughout this chapter we employ the North East Down convention for our coordinate frames and thus the heading direction for the quadrotor is parallel to the negative z axis of the $\mathcal{F}_Q$ frame.

3.3.3 Attitude Controller

The Attitude Controller is based on the nonlinear geometric PID controller originally developed for a quadrotor drone in [53]. A similar controller is also presented in [11]. The

goal of this control element is to drive the quadrotor's heading to be parallel to the desired thrust vector $\underrightarrow{f^{des}}$ computed previously by the Position Controller using equation (3.34). A PID attitude control law was chosen for this system as a slung load could induce a steady state moment acting on the drone due to the positioning of the attachment point for the cable. To begin, we convert our desired thrust vector and yaw attitude ψ^{Qd} into an overall desired quadrotor attitude quaternion $\mathbf{q}^{Qd}$.

$$\theta^{Qd} = \arccos\left(-[0 \quad 0 \quad 1]\frac{\mathbf{f}^{des}}{\sqrt{\mathbf{f}^{des\intercal}\mathbf{f}^{des}}}\right) \tag{3.39}$$

$$\hat{\mathbf{a}}^{Qd} = \frac{\mathbf{a}^{Qd}}{\sqrt{\mathbf{a}^{Qd\intercal}\mathbf{a}^{Qd}}}, \quad \mathbf{a}^{Qd} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}^{\times} \mathbf{f}^{des} \tag{3.40}$$

$$\mathbf{q}^{Qd} = \begin{bmatrix} \cos(\theta^{Qd}/2) \\ \sin(\theta^{Qd}/2)\hat{\mathbf{a}}^{Qd} \end{bmatrix} \otimes \begin{bmatrix} \cos(\psi^{Qd}/2) \\ 0 \\ 0 \\ -\sin(\psi^{Qd}/2) \end{bmatrix} \tag{3.41}$$

where the $\times$ superscript was previously defined in equation (3.4) and the quaternion multiplication operator $\otimes$ was defined in equation (3.19). Once we have computed $\mathbf{q}^{Qd}$, we can convert it into a desired direction cosine matrix attitude $\mathbf{R}_{IQ}^{d}$ using equation (3.2). We then apply the following control law using the measured attitude $\mathbf{R}_{IQ}$ and angular velocity $\boldsymbol{\omega}^{Q}$ of the drone.

$$\begin{aligned} \mathbf{m}_Q^T = &-\mathbf{K}_R^{att}\mathbf{e}_R - \mathbf{K}_\Omega^{att}\mathbf{e}_\Omega - \mathbf{K}_I^{att}\mathbf{e}_I + \mathbf{J}^q\mathbf{R}_{IQ}{}^{\intercal}\mathbf{R}_{IQ}^{d}\dot{\boldsymbol{\omega}}^{Qdf} \\ &+ (\mathbf{R}_{IQ}{}^{\intercal}\mathbf{R}_{IQ}^{d}\boldsymbol{\omega}^{Qdf})^{\times}\mathbf{J}^q(\mathbf{R}_{IQ}{}^{\intercal}\mathbf{R}_{IQ}^{d}\boldsymbol{\omega}^{Qdf}) \end{aligned} \tag{3.42}$$

$$\mathbf{e}_R = \frac{1}{2}(\mathbf{R}_{IQ}^{d}{}^{\intercal}\mathbf{R}_{IQ} - \mathbf{R}_{IQ}{}^{\intercal}\mathbf{R}_{IQ}^{d})^{\vee} \tag{3.43}$$

$$\mathbf{e}_\Omega = \boldsymbol{\omega}^{Q} - \mathbf{R}_{IQ}{}^{\intercal}\mathbf{R}_{IQ}^{d}\boldsymbol{\omega}^{Qdf} \tag{3.44}$$

$$\mathbf{e}_I = \int_0^t (\mathbf{e}_\Omega(\tau) + \mathbf{C}_2\mathbf{e}_R(\tau))d\tau \tag{3.45}$$

where $\mathbf{K}_R^{att}$, $\mathbf{K}_\Omega^{att}$, $\mathbf{K}_I^{att}$ and $\mathbf{C}_2$ are diagonal control gain matrices. The use of diagonal matrices marks an important difference between the Attitude Controller presented in this work and that presented in [2] and [53]. In [53], scalars are used which means that the same gains are applied for controlling pitch, roll and yaw errors. The authors then develop a constraint on the $\mathbf{C}_2$ value that ensures asymptotic stability of the resulting controller. For most quadrotors though, it is far easier to generate large pitching and rolling moments

than it is to generate large yawing moments. While developing our proposed controller, we found that favourable gains for pitch and roll moments resulted in requested yaw moments that could not be generated. This led to significant motor saturation issues which hampered the drone's ability to fly properly. One potential solution to this issue is to rework the drone's mixer so as to prioritize pitching and rolling moment generation and deprioritize yaw moment generation as done in [54]. An alternate solution though was to introduce diagonal control gain matrices such that yaw gains could be decoupled from those required for pitching and rolling. This approach also helped solve an integral windup problem that became apparent during preliminary gain tuning. Specifically, we could eliminate the integral control element for yaw moments while leaving it in place for pitch and roll moments. This strategy makes sense physically since it is unlikely that a steady state yawing moment would be generated on the quadrotor.

A key element of the proposed Attitude Controller is the feedforward terms. These require the desired angular velocity $\boldsymbol{\omega}^{Qdf}$ and angular acceleration $\dot{\boldsymbol{\omega}}^{Qdf}$ of the drone resolved in $\mathcal{F}_Q$. These two values are computed based on the desired flight trajectory of the drone using differential flatness equations. In [11] differential flatness equations are also used to compute $\dot{\boldsymbol{\omega}}^{Qdf}$ for a feedforward attitude control term. The equations for computing these values are summarized in Appendix A.

3.3.4 Thrust Allocation and Saturation

At this stage the controller has computed a desired moment $\mathbf{m}_Q^T$ and overall thrust force F^{thrust} using equations (3.42) and (3.38). Ultimately though for a real quadrotor, the controller must supply a command to each of the four motors on the drone. The resulting four thrusts can generate an overall thrust and moment acting on the drone but due to actuator saturation these values may not match the desired values. For our quadrotor drone we will assume a square configuration of the thrusters where the distance of each propeller from the x and y axes is a as shown in Figure 3.4.

In general, we will model the the vertical thrust and moment about the axis of rotation generated by our propeller spinning at a given number of revolutions per minute (RPM), Ω, using the following relations.

$$\mathbf{f}_Q^i = \begin{bmatrix} 0 \\ 0 \\ -k_t {\Omega_i}^2 \end{bmatrix} \tag{3.46}$$

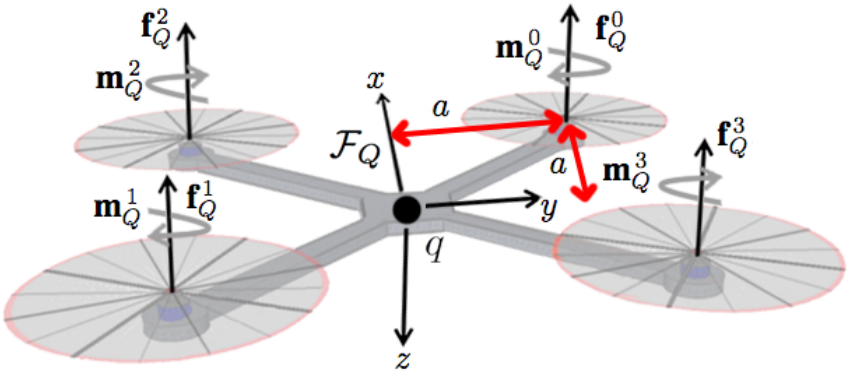

FIGURE 3.4: Quadrotor Motor Naming Convention

$$\mathbf{m}_Q^i = \begin{bmatrix} 0 \\ 0 \\ \pm k_q {\Omega_i}^2 \end{bmatrix} \tag{3.47}$$

We assign a negative sign in equation (3.46) to account for the fact that we are using the North East Down convention for our quadrotor. The sign of the moment generated in equation (3.47) depends on the direction that the propeller is spinning. k_t and k_q are assumed to be approximately constant and are specific to the propellers being used for our drone. For our drone we assume that propellers 0 and 1 spin counterclockwise while propellers 2 and 3 spin clockwise.

Using this naming convention, we can establish a mapping between the desired control moment $\mathbf{m}_Q^T$ and thrust magnitude F^{thrust} and the required RPM of each motor.

$$\begin{bmatrix} F^{thrust} \\ \mathbf{m}_Q^T \end{bmatrix} = \begin{bmatrix} k_t & k_t & k_t & k_t \\ -k_t a & k_t a & k_t a & -k_t a \\ k_t a & -k_t a & k_t a & -k_t a \\ k_q & k_q & -k_q & -k_q \end{bmatrix} \begin{bmatrix} {\Omega_0}^2 \\ {\Omega_1}^2 \\ {\Omega_2}^2 \\ {\Omega_3}^2 \end{bmatrix} \tag{3.48}$$

We can invert the four by four matrix in equation (3.48) in order to obtain a mapping between the desired control force and moment and the required RPM values for each motor. In our controller setup, we take $\mathbf{m}_Q^T$ and F^{thrust} and convert these into a preliminary set of motor RPMs Ω_0, Ω_1, Ω_2, Ω_3. We then clip these RPM values to ensure that they fall within the attainable range of RPM values for our system that is we constrain $\Omega_{min} \leq \Omega_i \leq \Omega_{max}$. After performing this clipping process, we pass the resulting RPM values back through equation (3.48) to obtain values of $\mathbf{m}_Q^T$ and F^{thrust}. This process ensures that our controller does not exceed the motor limits of our system.

The technique presented above is fairly rudimentary and indeed other papers such as [54] have explored more advanced methods for allocating thrust to the motors of a quadrotor that prioritize achieving pitching and rolling moments.

3.4 Controller Evaluation

In this section we present simulation results for our proposed controller and trajectory generator created using a MATLAB implementation of the quadrotor-slung load system presented in Section 3.2. We first discuss the tuning process for all the control gains required in the system and then present two baseline controller designs that will be compared against our proposed design. Afterwards we present a series of simulations with all three controllers tracking a variety of input-shaped maneuvers both with and without swinging disturbances. An analysis of these results will follow to assess the merits of our proposed controller design compared to the baseline versions presented. Table 3.1 presents the physical properties used for our quadrotor-slung load system. We will use a modified Pelican quadrotor for all testing in this project. The values for k_t, k_q, Ω_{max} and Ω_{min} in Table 3.1 were determined experimentally as will be discussed in Section 5.4.

3.4.1 Controller Tuning Process

The proposed controller in this paper requires tuning numerous gains for the Position, Oscillation and Attitude Controllers. This process can be further complicated by the fact that the Position and Attitude Controllers need to work in tandem in order to achieve horizontal motions of the drone. The Oscillation Controller requires selecting gains for the diagonal control gain matrices $\mathbf{K}_p^{oscil}$ and $\mathbf{K}_d^{oscil}$ in equation (3.21). For the Position Controller, scalars k^{path} and k^{oscil} and diagonal control gain matrices $\mathbf{K}_p^{path}$, $\mathbf{K}_d^{path}$, and $\mathbf{K}_i^{path}$ in equations (3.34) and (3.35) need to be tuned. For the Attitude Controller, diagonal control gain matrices $\mathbf{K}_R^{att}$, $\mathbf{K}_\Omega^{att}$, $\mathbf{K}_I^{att}$ and $\mathbf{C}_2$ in equations (3.42) and (3.45) need to be selected. Assuming that the same gains are used for x and y position control elements as well as for pitch and roll attitude elements we have a total of 20 gains to tune for this controller.

In this work, we divided the tuning process into four parts. The first tuning step focused on tuning the components of $\mathbf{K}_p^{path}$, $\mathbf{K}_d^{path}$, and $\mathbf{K}_i^{path}$ required for vertical motion of the quadrotor. For this tuning process, we set the quadrotor to hold its current position and

TABLE 3.1: Properties of the Pelican Quadrotor and Slung Load.

Parameter	**Symbol**	**Value**	**Unit**
Quadrotor mass	m^q	1.35	kg
Payload mass	m^p	0.4	kg
Quadrotor moment of inertia	$\mathbf{J}^q$	$\begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{bmatrix} \times 10^{-2}$	$\mathrm{kg\,m^2}$
Payload moment of inertia	$\mathbf{J}^p$	$\begin{bmatrix} 2.4 & 0 & 0 \\ 0 & 2.4 & 0 \\ 0 & 0 & 2.4 \end{bmatrix} \times 10^{-5}$	$\mathrm{kg\,m^2}$
Number of cables	$Ntot$	1	
Cable unstretched length	l_1^0	1	m
Cable elasticity	k_1	6.2832×10^3	$\mathrm{N\,m^{-1}}$
Cable damping	b_1	300	$\mathrm{N\,s\,m^{-1}}$
Quadrotor cable attachment point	$\mathbf{r}_Q^{q1}$	$\begin{bmatrix} 0 \\ 0 \\ 0.15 \end{bmatrix}$	m
Payload cable attachment point	$\mathbf{r}_P^{p1}$	$\begin{bmatrix} 0 \\ 0 \\ -0.05 \end{bmatrix}$	m
Motor xy arm length	a	0.1414	m
Propeller thrust coefficient	k_t	1.993×10^{-7}	$\mathrm{N}RPM^{-2}$
Propeller moment coefficient	k_q	3.3×10^{-9}	$\mathrm{N\,m}RPM^{-2}$
Max propeller RPM	Ω_{max}	7800	RPM
Min propeller RPM	Ω_{min}	840	RPM

set the mass of the drone and payload used for computing $\mathbf{f}^{weight}$ in equation (3.36) to be 10 percent higher than the values shown in Table 3.1. In simulation, this causes the drone to initially fly upwards along the negative z axis. We first set the z element of $\mathbf{K}_i^{path}$ to zero and manually adjusted the z values of $\mathbf{K}_p^{path}$ and $\mathbf{K}_d^{path}$. Our criteria for selecting these gains was to minimize the time it took the drone to stabilize to a new altitude while avoiding overshoots. The behaviour of the system closely followed that of a classic second order system with both underdamped and overdamped behavior. In the end we settled for values that yielded approximately critical damping. Once we achieved satisfactory results, we then added back in the z element of $\mathbf{K}_i^{path}$ and incrementally increased it. Our evaluation criteria were the maximum altitude error plus the time it took the drone's altitude to settle to within ± 1 centimeter of the original value.

The next step in the tuning process was to tune the x and y elements of $\mathbf{K}_p^{path}$ and $\mathbf{K}_d^{path}$ as well as the pitching and rolling moment components of $\mathbf{K}_R^{att}$, $\mathbf{K}_\Omega^{att}$, $\mathbf{K}_I^{att}$ and $\mathbf{C}_2$. We elected to set the x and y values of $\mathbf{K}_i^{path}$ to zero as we intended the integral position element to primarily compensate for uncertainties in the system's mass. We

had the quadrotor perform a lateral rest to rest maneuver, set $k^{path} = 1$, $k^{oscil} = 0$ and reduced the weight of the payload to 1 gram in order to eliminate its effect on the drone's flight. We started off with low gains for $\mathbf{K}_p^{path}$ and $\mathbf{K}_d^{path}$. A typical cycle would then involve adjusting the Attitude Controller gains to achieve good tracking of the desired attitude generated by the Position Controller. We would then make the position control gains more aggressive to improve the trajectory tracking performance. This sometimes required us to revisit the Attitude Controller gains to ensure that we were still achieving satisfactory performance. A key element of this process involves accurately diagnosing which control element is leading to undesirable performance of the system. One strategy that proved useful for this process was to convert the current and desired attitude of the drone into Euler angles and to plot these on top of each other. The resulting plots were much easier to interpret than plots of the attitude error components in equation (3.42) or plots of the components of the true and desired quaternion elements. The ultimate goal was to minimize the trajectory tracking error throughout a given flight trajectory for the drone. This typically requires achieving good tracking of the desired drone attitude. The tuning was then verified for a more aggressive non-rest to rest maneuver in simulation to validate that it achieved satisfactory trajectory tracking and attitude tracking.

The third step in the tuning process focused on tuning the Oscillation Controller's gains. For this step we set the quadrotor to hold its starting position and introduced an initial velocity between the payload and the quadrotor. We first performed a baseline test setting $k^{oscil} = 0$ to assess how long it would take the drone to dampen the swinging disturbance to within $\pm$1deg using only its Position Controller. This effect was previously discussed in Section 3.1.2. We then set $k^{oscil} = 1$ and began the tuning process for the x and y components of $\mathbf{K}_p^{oscil}$ and $\mathbf{K}_d^{oscil}$. The metric for assessing the tuning was how much of an improvement in swing disturbance damping we were able to achieve in comparison to the baseline test. We found that the z component of $\mathbf{K}_p^{oscil}$ and $\mathbf{K}_d^{oscil}$ had a negligible effect on the swing damping performance for this test. We performed an additional test with a rest to rest maneuver with an initial swinging disturbance and saw no improved performance from adding a z component to $\mathbf{K}_p^{oscil}$ and $\mathbf{K}_d^{oscil}$.

The final tuning step involved the yawing moment components of $\mathbf{K}_R^{att}$, $\mathbf{K}_\Omega^{att}$, $\mathbf{K}_I^{att}$ and $\mathbf{C}_2$. When performing preliminary flight testing on an actual quadrotor drone, we found significant performance issues with the drone due to its yaw control elements. The propellers on the Pelican have a low k_q value which means that the drone has a limited ability to generate yawing moments. This meant that the drone could easily reach saturation when trying to correct for yaw errors if the gains were too high. We also encountered significant integrator windup issues for the yaw gains in our system. Similar results were

TABLE 3.2: Controller Gains for Simulated Pelican Quadrotor and Slung Load.

Control Gain	**Equation**	**Value**	**Unit**
$\mathbf{K}_p^{oscil}$	(3.21)	$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	N
$\mathbf{K}_d^{oscil}$	(3.21)	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	N s
k^{path}	(3.34)	1	
k^{oscil}	(3.34)	1	
$\mathbf{K}_p^{path}$	(3.35)	$\begin{bmatrix} 5.5 & 0 & 0 \\ 0 & 5.5 & 0 \\ 0 & 0 & 16 \end{bmatrix}$	$\mathrm{N\,m^{-1}}$
$\mathbf{K}_d^{path}$	(3.35)	$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 9 \end{bmatrix}$	$\mathrm{N\,s\,m^{-1}}$
$\mathbf{K}_i^{path}$	(3.35)	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 16 \end{bmatrix}$	$\mathrm{N\,s^{-1}\,m^{-1}}$
$\mathbf{K}_R^{att}$	(3.42)	$\begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 0.4 \end{bmatrix}$	N m
$\mathbf{K}_\Omega^{att}$	(3.42)	$\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}$	N m s
$\mathbf{K}_I^{att}$	(3.42)	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	N m
$\mathbf{C}_2$	(3.45)	$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\mathrm{s^{-1}}$

found in the simulator and the yaw gains needed to be set significantly lower than the attitude gains for pitching and rolling moments. Based on these observations, we opted to eliminate the yaw component of $\mathbf{K}_I^{att}$ and $\mathbf{C}_2$. For tuning the remaining $\mathbf{K}_R^{att}$, $\mathbf{K}_\Omega^{att}$ components, we prescribed a step input in the desired yaw value for the drone while having it maintain its current position. The main criteria for assessing the tuning was the time it took for the drone to reach the desired yaw value while also avoiding actuator saturation. We progressively increased the size of the yaw setpoint step up to 45 deg and selected a set of tuning values that offered good settling time without initial saturation. A summary of the controller gains resulting from this process is presented in Table 3.2.

3.4.2 Baseline Flight Controllers

In order to assess the performance of our proposed controller we will compare it against two baseline controller designs. As previously discussed in Section 3.1, there are numerous flight controller designs for quadrotor and quadrotor-slung load systems. For this work, our goal was to establish as fair a comparison as possible. In particular, we were concerned about how differences in the Attitude Controller might impact our results. The main novelty of our proposed controller is the use of a swing prediction model for tracking input-shaped flight trajectories. We thus want to make sure that differences in the flight performance are due to this change and not due to differences in the Attitude Controller used in this work compared to others. Another important point was the potential for bias during the tuning process for baseline controller designs. One might be able to argue for instance that differences between the performance of our controller versus a baseline design are due to the fact that the baseline controller was not adequately tuned for the properties of our specific drone.

With these points in mind, we opted to create baseline controllers by simplifying our proposed controller design. We created a first baseline controller design by simply setting $k^{oscil} = 0$ in equation (3.34). The resulting controller, henceforth referred to as Controller A, essentially ignores the swinging motion of the payload and solely attempts to track a prescribed flight trajectory. This baseline controller design is based on the fact that, in Section 3.1.2, numerous papers dealing with the use of input shaping for quadrotor-slung load systems had no active swing damping elements. The swinging motion of the payload is purely managed through the use of an input-shaped flight trajectory. Controller A also represents the simplest possible controller design for a quadrotor-slung load system as it is essentially a generic quadrotor flight controller.

Our second baseline controller, Controller B, eliminates the swing prediction element in our proposed controller and always uses equation (3.33) in order to compute the desired position and velocity of the payload. This simplification means that this controller constantly attempts to eliminate any swinging motion in the payload as seen in the inertial frame. This strategy is similar to numerous controller designs presented in Section 3.1.3, where the drone attempts to constantly eliminate any swinging motion of the payload as the drone flies along a prescribed flight trajectory. Our novel controller will be referred to as Controller C. Controller B is computationally simpler than Controller C and thus provides a good opportunity to assess the merits of our proposed swing prediction element.

Apart from these changes, all three controllers use the exact same equations and control gains. This eliminates the potential effect of tuning bias and the use of different attitude controllers. This same approach was used in [2] to generate baseline comparisons for our proposed controller. The following subsection presents simulation results for all three controllers for a series of tests to assess their performance.

3.4.3 Simulation Results

In order to verify the performance of the proposed controller, we performed simulations comparing our controller against the baseline designs proposed in Section 3.4.2. The properties of our quadrotor-slung load system are given in Table 3.1. In order to test the robustness of the proposed controller we scaled the values of m^q, m^p and $\mathbf{J}^q$ by a factor of 0.9 whenever these values were used within any control equations. We also introduced zero mean measurement noise for the position $\mathbf{r}^p$ and velocity $\mathbf{v}^p$ of the payload. The noise values had standard deviations of 0.005m and $0.02\text{m}\,\text{s}^{-1}$ respectively. In order to avoid complications with MATLAB's variable step size solvers we passed our noise values through a discrete time low pass filter using a critical frequency of 31.8Hz. A single set of noise values was generated and then applied to each trial considered in order to maintain consistency between trials.

For the purposes of comparison throughout this section, we define the swing angle of the payload based on the vector $\mathbf{r}_I^{cable}$ between the quadrotor's mounting point and the payload's center of mass resolved in the inertial frame.

$$\mathbf{r}_I^{cable} = \mathbf{r}^p - \mathbf{R}_{IQ}\mathbf{r}_Q^{q1} - \mathbf{r}^q \tag{3.49}$$

We then solve for the angle between $\mathbf{r}_I^{cable}$ and the z axis of the inertial frame.

For Simulation 1, we have the drone dampen an initial swinging disturbance on the payload while attempting to hold its starting position. We introduced a relative velocity of $[4 \quad 0 \quad 0]^\intercal \text{m}\,\text{s}^{-1}$ between the center of mass of the payload and the quadrotor. Figure 3.5 shows the resulting swing angle of the payload for this simulation. Figure 3.6 shows the magnitude of the error in the XY plane between the quadrotor's actual position and the desired position.

In both Figures 3.5 and 3.6, the performances of Controller B and C are identical since Controller C's reference swing angle is zero throughout this simulation. Controller A is still able to dampen the swinging motion of the pendulum over time, but this happens much slower than for Controllers B and C. Controller A achieves its swing damping

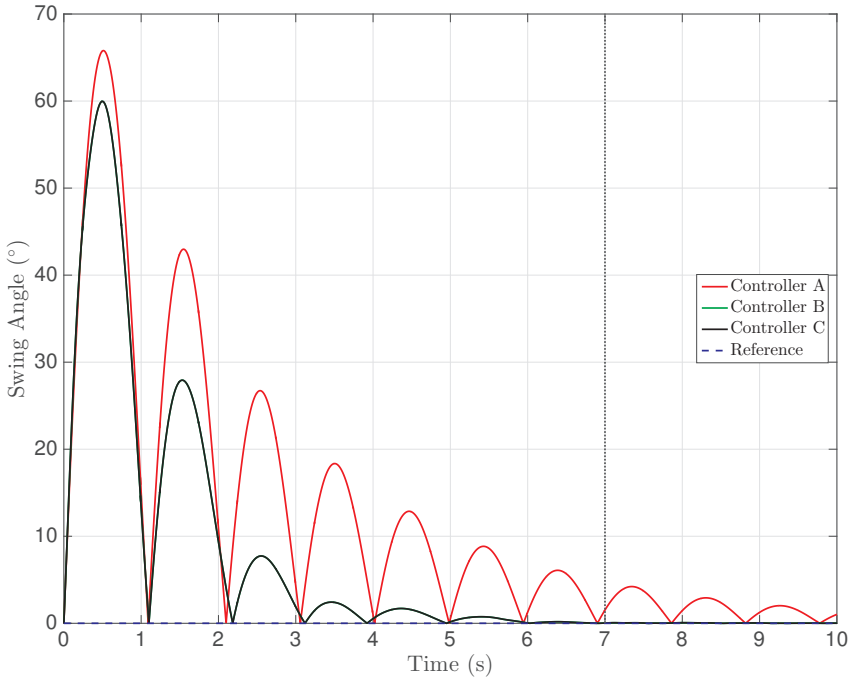

FIGURE 3.5: Simulation 1: Time History of Swing Angle

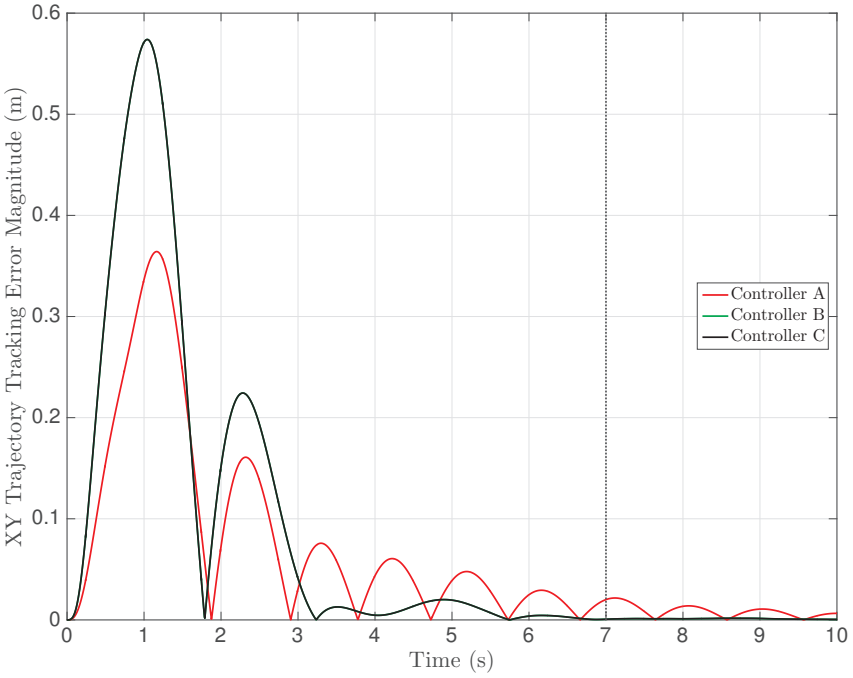

FIGURE 3.6: Simulation 1: Time History of Quadrotor Trajectory Tracking Error Magnitude

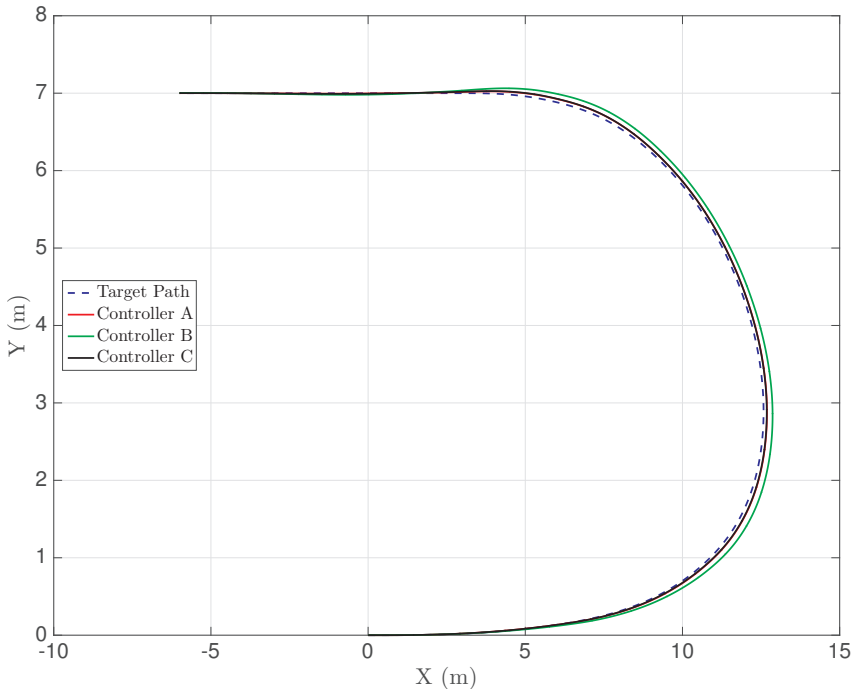

FIGURE 3.7: Simulation 2: Top View of Quadrotor Motion

due to the fact that, as the payload swings, it shares some of its kinetic energy with the quadrotor drone by pulling it away from its target position. The flight controller dissipates this kinetic energy by attempting to bring the quadrotor back to rest at its starting position. Controllers B and C instead make a large initial displacement in order to dissipate the swinging motion of the payload.

Simulation 2 involves an input-shaped non-rest to rest u-turn maneuver. In this simulation, the drone and its payload start off at the origin with an initial velocity of $[6 \quad 0 \quad 0]^\intercal \text{m s}^{-1}$ and we generate an input-shaped trajectory to reach final position $[3 \quad 7 \quad 0]^\intercal \text{m}$ with velocity $[-3 \quad 0 \quad 0]^\intercal \text{m s}^{-1}$ in a total time of 7s. We employ a two-hump EI shaper for this maneuver and the initial and final accelerations of the drone are set to zero. We simulate the resulting behaviour for a total of 10s. Figure 3.7 shows a top view of the motion of the drone for all three controllers. Figure 3.8 compares the swing angle for all three controllers throughout the maneuver. Figure 3.9 shows the position tracking error magnitude for the XY plane. Figure 3.10 shows the magnitude of $(k^{oscil}\mathbf{f}^{oscil})$ from equations (3.21) and (3.34) throughout the simulation, as an indication of the control effort being expended.

The results from Simulation 2 help to highlight some of the potential benefits of our proposed controller (C) as compared to the two baseline designs (A and B). Figure 3.8 for instance shows how both Controller B and C are able to achieve similar swing damping

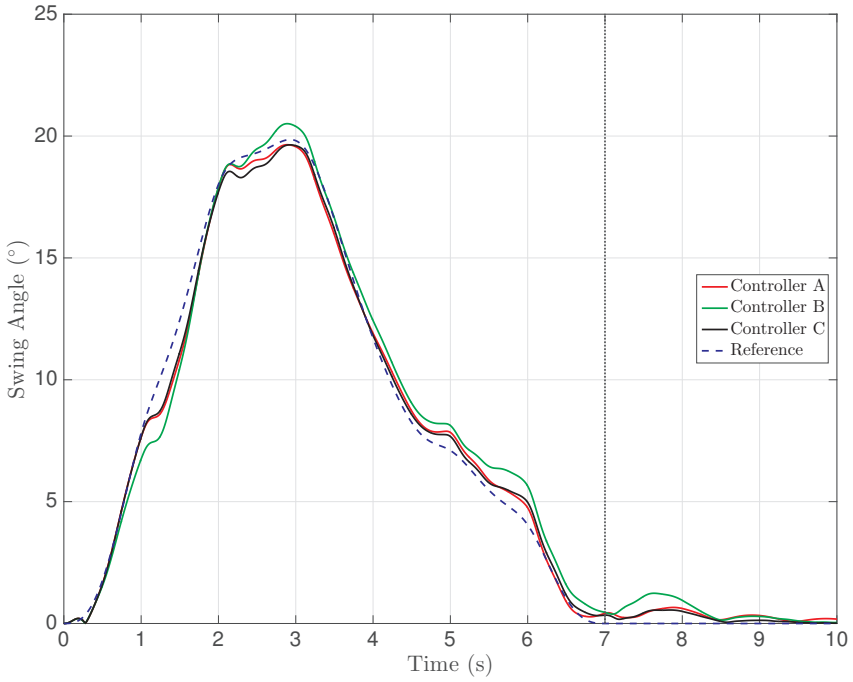

FIGURE 3.8: Simulation 2: Time History of Swing Angle

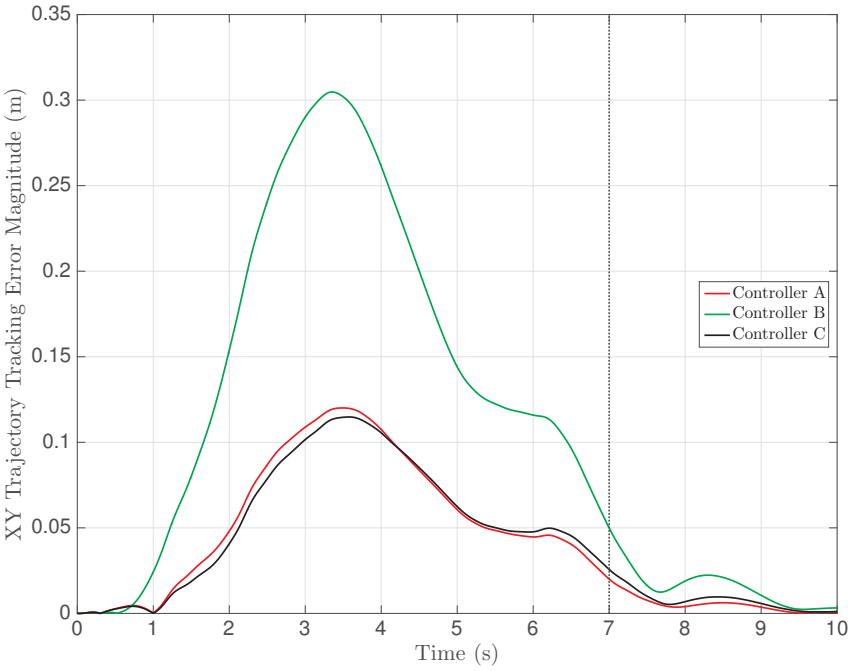

FIGURE 3.9: Simulation 2: Time History of Quadrotor Trajectory Tracking Error Magnitude

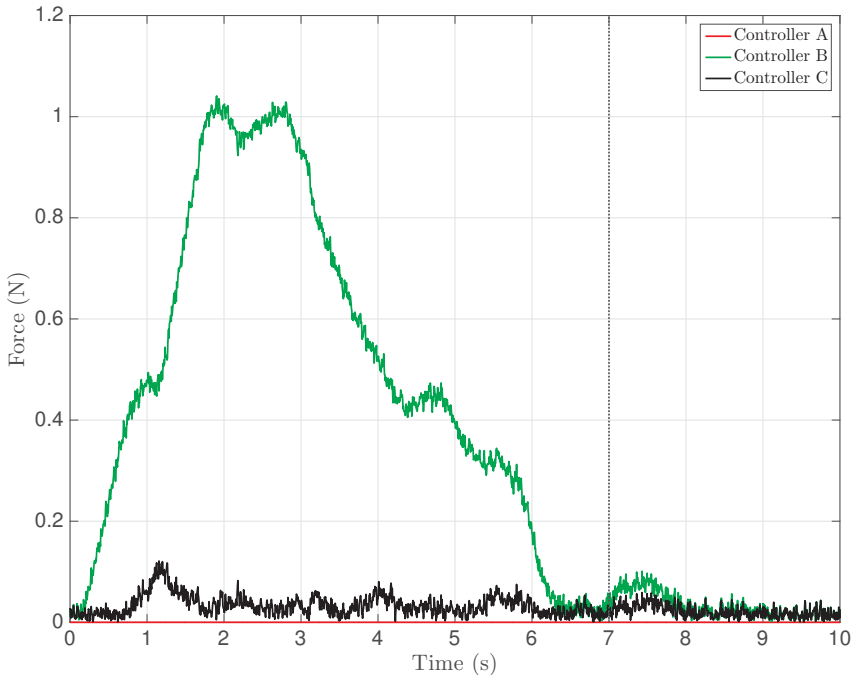

FIGURE 3.10: Simulation 2: Oscillation Feedback Force

performance. However, as shown in Figure 3.9, Controller B has more difficulty tracking the desired input-shaped flight trajectory and experiences higher tracking errors than A or C throughout. Controller B is trying to constantly eliminate the swinging motion of the payload while tracking the prescribed flight trajectory. Clearly from Figure 3.8, Controller B fails to eliminate all swinging in the payload during the accelerating portion of the prescribed trajectory over first seven seconds of the flight. This makes sense since horizontal accelerations will inevitably induce swinging motion in our underactuated system. As the payload swings outwards during the u-turn, a high control effort is computed by the Oscillation Controller to have the quadrotor chase after the payload. This results in Controller B making a wider u-turn than than Controller A or C as shown in Figure 3.7. Ultimately, the wider u-turn flown by Controller B induces the swinging motion seen in Figure 3.8. The higher control effort for Controller B is demonstrated in Figure 3.10. Notice for instance how the timing of the peak in control effort in Figure 3.10 matches the timing of the peak in the swing angle shown in Figure 3.8. The swinging motion of the pendulum for Controller C shown in Figure 3.8 closely matches that of the second order prediction model shown as the Reference. This leads to minimal control effort for Controller C in Figure 3.10 and allows its trajectory tracking error to approach the performance of Controller A. The results of this first simulation also highlight the effectiveness of input shaping for swing prevention in the absence of external swinging disturbances. In Figure 3.8 for instance, Controller A has almost no residual swing angle

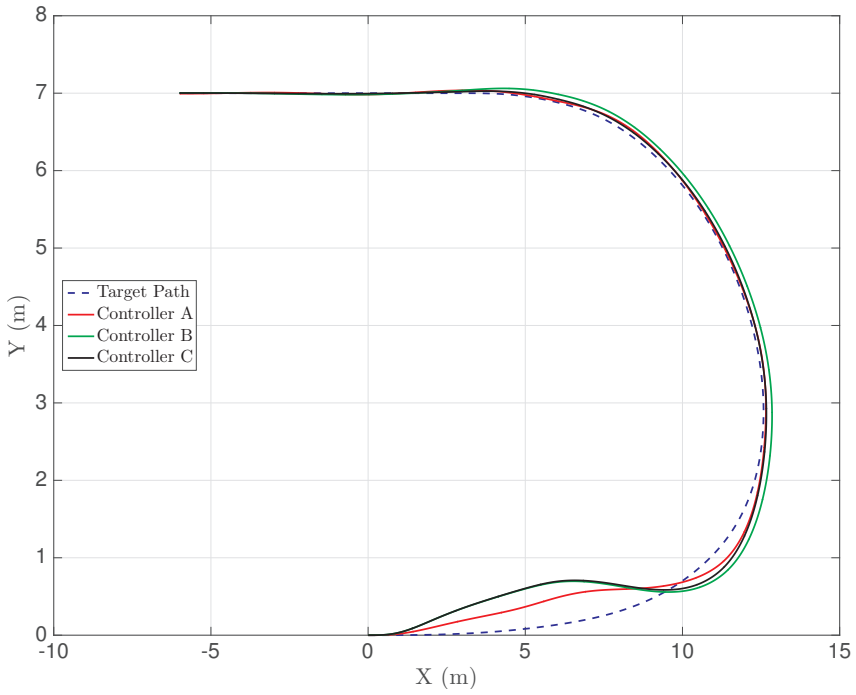

FIGURE 3.11: Simulation 3: Top View of Quadrotor Motion

after the 7s mark of the simulation once the drone has completed the accelerating phase of the motion.

Simulation 3 repeats the scenario in Simulation 2 but this time introduces an initial relative velocity between the payload and the quadrotor centers of mass of $[-2 \quad 4 \quad 0]^{\intercal}\mathrm{m\,s^{-1}}$. For this simulation, Figure 3.11 shows a top view of the resulting motion of the drone for all three controllers. Figure 3.12 compares the swing angle for all three controllers throughout the maneuver. Figure 3.13 shows the position tracking error magnitude for the XY plane. Figure 3.14 shows the magnitude of the feedback oscillation force.

Compared to Simultation 2, the addition of a swinging disturbance causes increased payload motion for Controller A throughout Figure 3.12. The swinging motion does appear to be gradually dampening out however it continues to oscillate much longer than with Controllers B or C. This result highlights one of the weaknesses of turning off the swing dampening control element during accelerating portions of an input-shaped maneuver as proposed in [24] and [25]. Had we employed that strategy here, the results would have mirrored those of Controller A over the first 7s of simulation. Figure 3.15 shows the magnitude of the XY component of $(k^{path}\mathbf{f}^{path})$ for all three controllers for Simulation 3. From Figure 3.15 it becomes apparent that Controller A is expending a large control effort with its Position Controller in order to dampen the swinging motion of the payload. It is also important to recognize that the damping performance of Controller A is due to

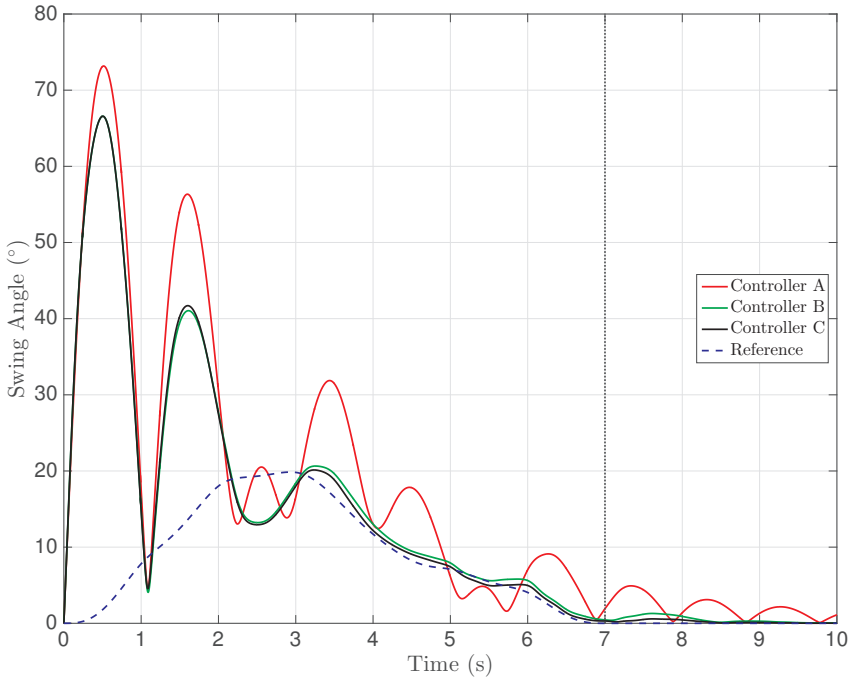

FIGURE 3.12: Simulation 3: Time History of Swing Angle

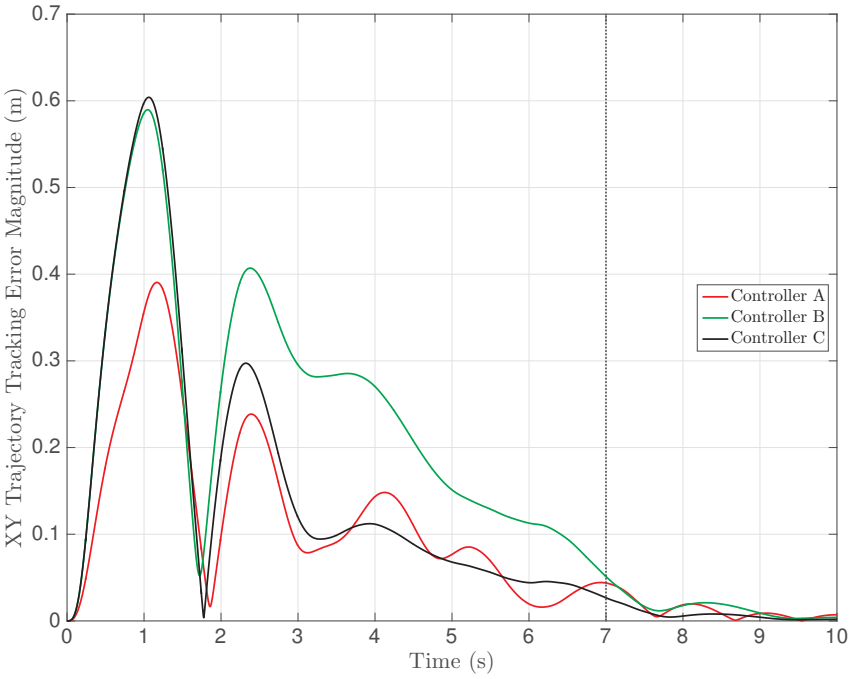

FIGURE 3.13: Simulation 3: Time History of Quadrotor Trajectory Tracking Error Magnitude

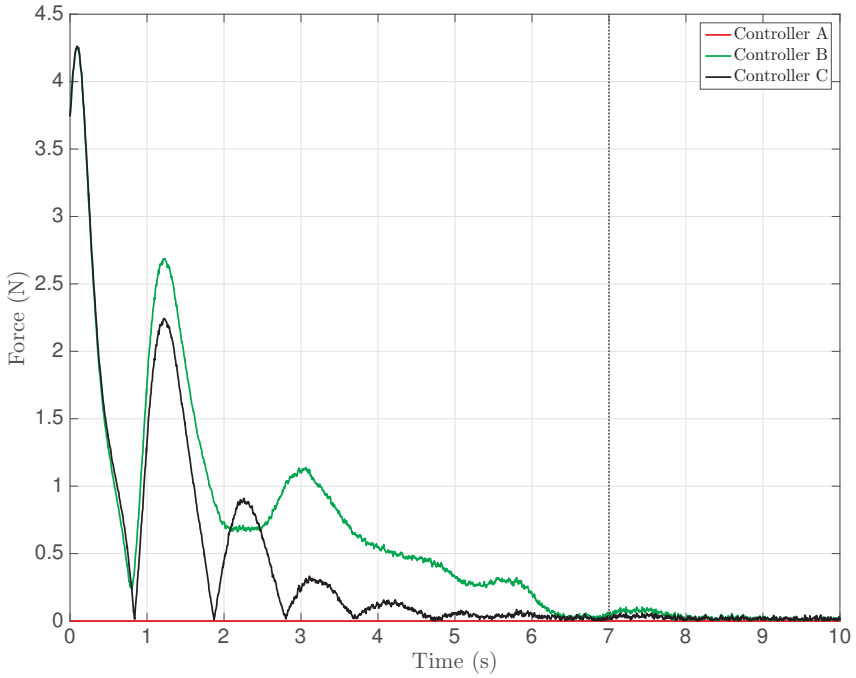

FIGURE 3.14: Simulation 3: Oscillation Feedback Force

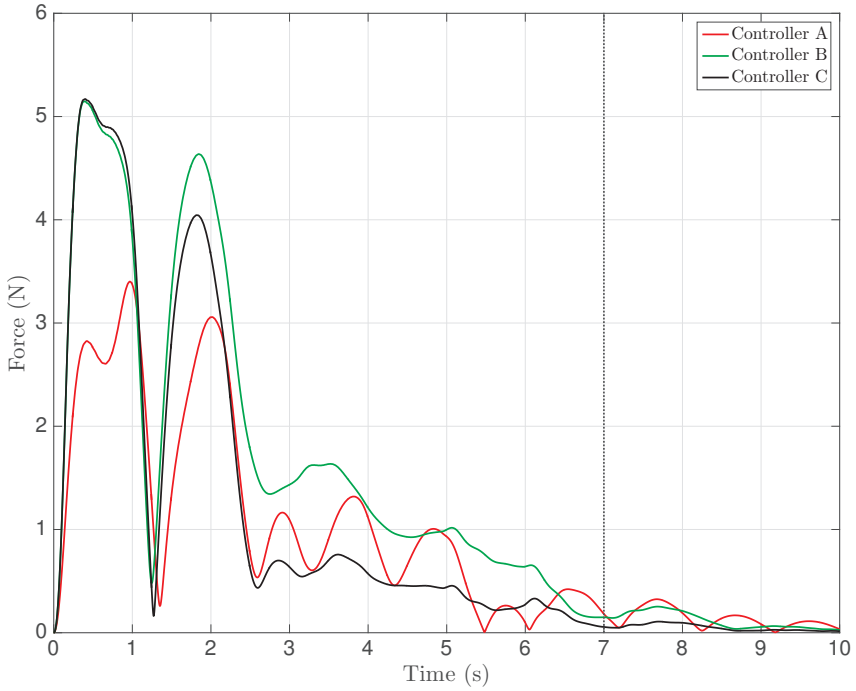

FIGURE 3.15: Simulation 3: Position Feedback and Feedforward Force

the payload being heavy enough to drag the quadrotor off course. Comparing Controllers B and C, we can see in Figure 3.12 that their swing damping performance is again almost identical for the maneuver being considered. However, in Figure 3.13 we see that the trajectory tracking error of Controller B is much higher than C between 2s through 7s. During this portion of the maneuver, Controller B at times reaches more than double the position error that Controller C had while requiring consistently higher oscillation feedback forces as shown in Figure 3.14. The trajectory tracking force requirements for Controller B mirror the position error magnitude terms as shown in Figure 3.15. This suggests that Controller C was able to achieve similar swing damping performance to Controller B but with reduced control effort. We believe that this is a direct result of Controller C making better use of the properties of the prescribed input-shaped flight trajectory. During the first three seconds of motion, Controller C exerts large control forces in order to compensate for the significant swinging disturbance acting on the payload. Afterwards though, once the swinging disturbance has been dissipated, the controller exerts minimal oscillation feedback control force and allows the input-shaped trajectory to prevent additional oscillations from occurring after the maneuver is completed. In essence, the controller effectively shifts its priorities from swing damping to trajectory tracking.

Overall, the simulations presented here suggest that our proposed controller is able to effectively dampen swinging disturbances in a quadrotor-slung load system while tracking an input-shaped flight trajectory. We have shown in simulation that our controller (C) outperforms simpler baseline control designs (A and B) by better managing the interplay between swing suppression control and tracking input-shaped flight trajectories. The results in this section are similar to those we obtained with a preliminary version of Controller C presented in [2].

3.4.4 Cable Length and Payload Mass Changes

The results presented Section 3.4.3 are a somewhat idealized scenario in that the cable length and payload mass used in our simulation are the same as those used during the controller tuning process. In a realistic scenario, the quadrotor will need to be able to transport a variety of slung loads with different masses and potentially different cable lengths. Our goal in this section is thus to investigate how well the control gains developed in Section 3.4.1 for a 1m cable with a 0.4kg payload will perform when the cable length and mass are changed in the simulation of the system.

To begin with, we will assess the effect of changing the unstretched cable length l_1^0 of our system on the swing damping performance of our controller. We will replicate the scenario from Simulation 1 in Section 3.4.3 where we begin our simulation with a relative velocity between the payload and quadrotor and assess how well the system dampens out this swinging motion. As shown previously in Figure 3.5, the performance of our proposed controller (C) and the simplified B controller are identical for this scenario. We thus compare the swing damping performance of our controller against the A controller which has no swing feedback terms. We will reuse the same tuning gains presented in Table 3.2 for these trials as well as the same payload measurement noise applied in Section 3.4.3. We once again scale the values of m^q, m^p and $\mathbf{J}^q$ by a factor of 0.9 anytime they appear in our controller. The relative velocity between the payload center of mass and the quadrotor takes the form $[v \quad 0 \quad 0]^\intercal \mathrm{m\,s^{-1}}$. For our trials, we normalize the initial velocity applied based on the cable length being investigated. Specifically, we impose the constraint that $0.5v^2 = gl_1^0$. This corresponds to the requirement that a point mass attached to a fixed pendulum of length l_1^0 with initial velocity v will have just enough kinetic energy to swing to 90°. We will then compare the performance of Controllers A and C based on three metrics. First we examine the maximum swing angle of the slung load relative to the z axis of our inertial frame. We also consider the maximum displacement of the quadrotor along the x axis. Finally, we assess how long it takes the quadrotor to dampen the pendulum's swinging to be within ±3°. The results from these trials are presented in Table 3.3

TABLE 3.3: Cable Length Analysis

l_1^0 (m)	v ($\mathrm{m\,s^{-1}}$)	Max Swing Angle (°)		Max Quad Displacement (m)		Time to dampen to ±3° (s)	
		A	C	A	C	A	C
0.2	1.98	65	Unstable	0.171	Unstable	4.94	Unstable
0.4	2.8	69	63	0.255	0.383	6.62	2.365
0.6	3.429	71	64	0.315	0.486	7.3	2.845
0.8	3.96	72	66	0.36	0.558	7.61	3.225
1	4.427	73	67	0.393	0.608	8.435	2.99
1.4	5.238	74	68	0.438	0.669	9.11	4.645
1.8	5.94	75	69	0.464	0.702	10.565	5.81
2	6.261	75	70	0.473	0.712	11.315	7.53

A key issue that we observed in these trials is that our controller (C) was unstable for cable lengths of 0.2m and lower. On closer inspection of these simulations we found that our Oscillation Controller was computing large values for $\mathbf{f}^{oscil}$ from equation (3.21) due to the high velocity of the payload. Furthermore, due to the short cable length, these

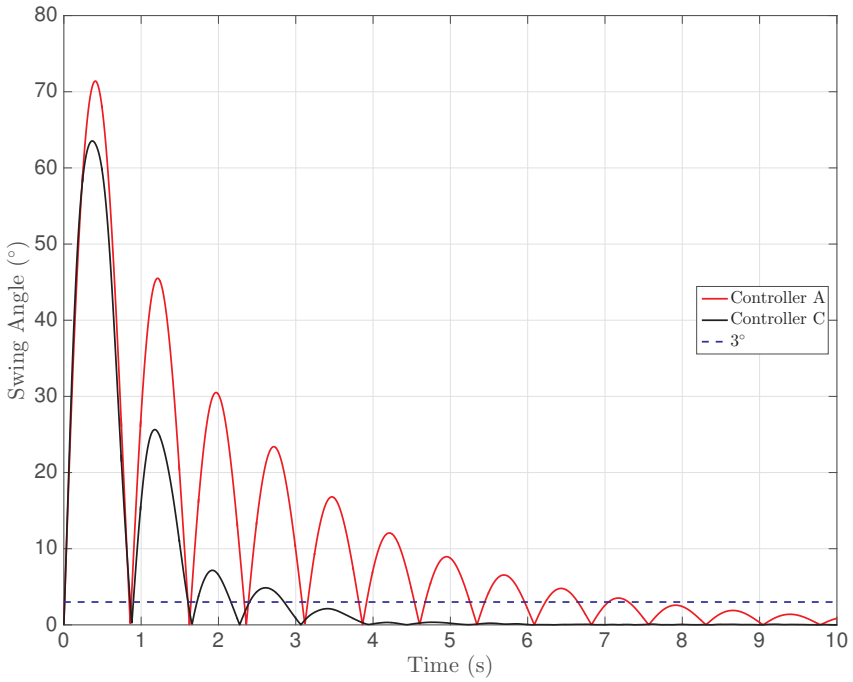

FIGURE 3.16: 0.6m Cable: Time History of Swing Angle

requested force values change sign at a high frequency. The resulting desired attitude changes for the drone were too aggressive for our Attitude Controller to track and our system experienced saturation on all four motors. We would likely need to reduce the controller gains in equation (3.21) in order to manage this kind of scenario. Realistically though, we doubt that cable lengths of 0.2m or lower would be implemented for quadrotor-slung load systems. Aside from these trials though, the results in Table 3.3 suggest that our tuning can maintain a good performance across a wide range of cable lengths. Our controller (C) consistently dampens the swinging motion of the payload much faster than the A controller but at the cost of larger displacements of the quadrotor. This makes sense since our controller essentially chases the swinging load while the A controller is simply pulled away from its starting position by the cable tension. There also appears to be a minimal reduction in the maximum swinging angle of the payload from using our controller. The performance of our controller for a wide range of cable lengths makes sense given that our Oscillation Controller relies on unit vectors instead of the position of the payload to determine control inputs. Figure 3.16 shows the swing angle time history for the 0.6m cable length trial. This figure clearly demonstrates the improved swing damping of Controller C compared to Controller A.

We will now assess the effect of changing the payload mass on our controllers performance. We replicate the same test scenario as above but this time alter the payload mass m^p as well as the initial relative velocity v along the x axis. For all trials presented here we

once again use a 1m long cable. For this set of trials, we normalize v to ensure that the payload always has the same starting kinetic energy of 5J regardless of how the payload mass is changed. That is we impose the constraint $0.5m^p v^2 = 5$. Once again we compare Controllers A and C based on the maximum cable swing angle, quadrotor displacement and time to dampen swinging to $\pm 3°$. Our original controller was tuned based on a 1m cable, 0.4kg payload and a 1.35kg quadrotor. The results from this analysis are summarized in Table 3.4.

TABLE 3.4: Mass Robustness Analysis

m^p (kg)	v ($\mathrm{m\,s^{-1}}$)	Max Swing Angle (°)		Max Quad Displacement (m)		Time to dampen to $\pm 3°$ (s)	
		A	C	A	C	A	C
0.3	5.774	102	92	0.342	0.458	11.81	4.805
0.4	5	83	77	0.425	0.635	8.55	3.685
0.8	3.536	52	47	0.599	0.756	4.27	3.17
1.1	3.015	42	38	0.664	0.781	3.25	3.16
1.35	2.722	37	33	0.7	0.793	3.135	3.135

In these trials low payload mass results in significant swinging motion of the payload. The trial with a payload mass of 0.3kg resulted in angles of 102° or 92° which would pose a serious risk to the drone platform and likely lead to crashes simply due to collisions between the cable or payload and the drone's propellers. For this reason we did not consider smaller payload masses in this set of trials. An interesting trend in these results is that as the payload mass increases, the swing damping performance of Controllers A and C appear to be converging as do the overall displacements induced in the quadrotor. Figure 3.17 shows the performance of the A and C controllers for the trial where $m^p = 1.35$kg. This represents a fairly extreme test case since the quadrotor itself has a mass of 1.35kg in these simulations. From Table 3.4, both controllers appear to have roughly identical swing dampening performance for this trial. Controller A reacts to being pulled away from its target position by the tension force in the cable as the payload moves. It thus makes sense that the A controller would take longer to dampen swinging disturbances caused by a lighter payload that cannot as easily displace it. Overall, it appears that the tuning we obtained using a 0.4kg payload stays robust for a wide range of payload masses.

The results presented in this section for Controller A are somewhat surprising and likely merit further investigation. Transporting slung payloads with manned helicopters is inherently difficult as discussed in [3], [4] and [5]. We would thus expect that a simple controller design such as Controller A might have more difficulty transporting slung loads

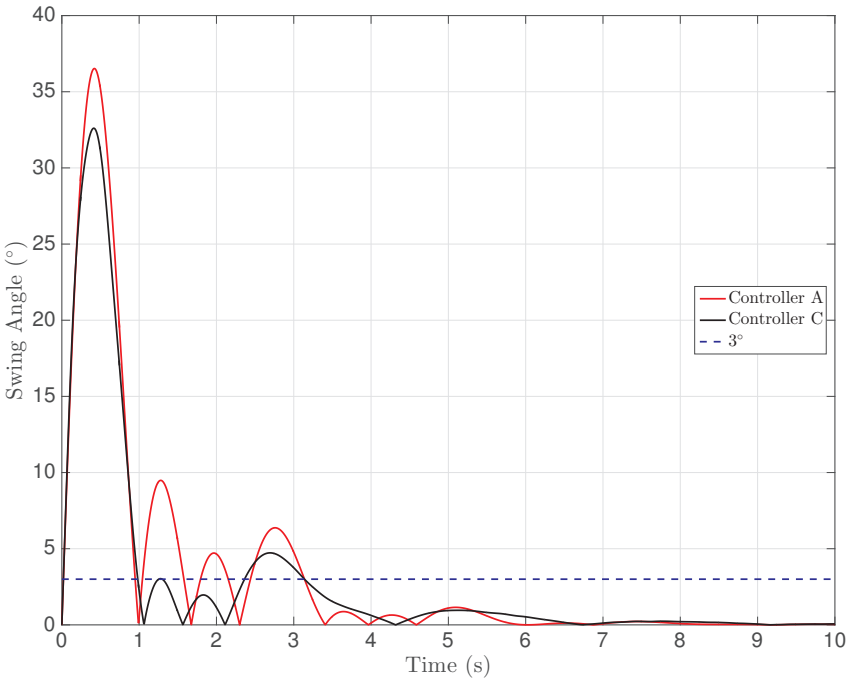

FIGURE 3.17: 1.35kg Payload: Time History of Swing Angle

and would be more prone to failure than Controller C. We also might expect Controller A to inadvertently add energy to the slung load while trying to correct for displacement errors. In our simulations though, we generally found that Controller A remained stable for a wide range of simulated scenarios. One possible explanation is that our simulation model does not limit how quickly thrust inputs can change. On a real quadrotor drone though, the propellers will take time to change rotation speeds and cannot instantly supply a desired thrust force. In general, we found that the performance of both the A and C controllers degraded quickly if motor saturation was consistently being reached. This makes sense since motor saturation indicates that our motors will no longer supply the desired thrust and moment requested by our controller. Controller A may be slightly less prone to saturation than Controller C since it does not have a $\mathbf{f}^{oscil}$ term contributing to the desired thrust vector $\mathbf{f}^{des}$ in equation (3.34). In addition to potentially increasing the magnitude of $\mathbf{f}^{des}$, the swing feedback $\mathbf{f}^{oscil}$ term can also change direction quickly if the payload is swinging violently. This could in turn lead to a more aggressive desired attitude trajectory that would require larger control moments to track. Ultimately though, these controllers will need to be compared on an actual quadrotor drone.

Chapter 4

Model Predictive Controller Design

The controller developed in Chapter 3 is based on the idea that when tracking an input-shaped flight trajectory for a quadrotor-slung load system one should simultaneously track a swinging motion reference trajectory. In our proposed controller, Controller C, flight trajectory and swing trajectory tracking are treated as separate objectives. We compute required forces to accomplish each one and then sum the results together as shown in equation (3.34) to obtain an overall desired thrust force vector. This value is then passed on to an Attitude Controller and subsequently a Thrust Allocation and Saturation element. As we developed this controller, we conjectured that it might be possible to formulate this entire problem in a single step. At the same time, we wanted to see if we could incorporate saturation directly into our controller's design. This led us to explore the use of model predictive control (MPC) for a quadrotor-slung load system. In this chapter, we begin by briefly introducing the topic of model predictive control and its applications to quadrotor-slung load systems. We then discuss the development of a preliminary model predictive controller using MATLAB and conclude by comparing it to Controller C developed in Chapter 3.

4.1 Model Predictive Control for Quadrotor-Slung Load Systems

As explained in [55], model predictive control is based on using a model to predict a system's future states based on the current state and a selected sequence of future inputs. In this work, we restrict ourselves to a linearized and discretized system model and perform predictions over a finite number of steps called the prediction horizon [56] [57].

For each step forward in the prediction horizon our controller must select a corresponding step input. To simplify this problem, we typically limit the degrees of freedom of our control inputs using a control horizon.

We can illustrate this setup mathematically as done in [58]. We begin by linearizing and discretizing a system model with state $\mathbf{x}$ and input $\mathbf{u}$ to obtain the form

$$\mathbf{x}_{j+1} = \mathbf{A}^*\mathbf{x}_j + \mathbf{B}^*\mathbf{u}_j \tag{4.1}$$

Using equation (4.1), we can determine a future sequence of states $\mathbf{x}$, denoted as $\bar{\mathbf{x}}$ given a sequence of future inputs $\bar{\mathbf{u}}$. We define N to be the number of steps in our prediction horizon and M to be the number of steps in our control horizon. We thus have $\bar{\mathbf{x}} = [\bar{\mathbf{x}}_0, \bar{\mathbf{x}}_1, ..., \bar{\mathbf{x}}_N]$ and $\bar{\mathbf{u}} = [\bar{\mathbf{u}}_0, \bar{\mathbf{u}}_1, ..., \bar{\mathbf{u}}_{N-1}]$. Ultimately, we will want our controller to track a prescribed set of states and perhaps even a set of nominal input values. To do so, we can supply our system with desired future state and input trajectories $\bar{\mathbf{x}}^{des}$ and $\bar{\mathbf{u}}^{des}$. We now need to select values for $\bar{\mathbf{u}}$ that will, based on our discretized model, help our system track these desired values. More formally, our goal is to determine $\bar{\mathbf{u}}$ such that we minimize a chosen cost function J.

$$\min_{\bar{\mathbf{u}}} J(\bar{\mathbf{x}}, \bar{\mathbf{u}}) \tag{4.2}$$

As an example, we can use the form in [58]:

$$J(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = (\bar{\mathbf{x}}_N - \bar{\mathbf{x}}_N^{des})^\intercal \mathbf{M}_1 (\bar{\mathbf{x}}_N - \bar{\mathbf{x}}_N^{des}) + \sum_{j=0}^{N-1} \Big((\bar{\mathbf{x}}_j - \bar{\mathbf{x}}_j^{des})^\intercal \mathbf{M}_2 (\bar{\mathbf{x}}_j - \bar{\mathbf{x}}_j^{des}) + (\bar{\mathbf{u}}_j - \bar{\mathbf{u}}_j^{des})^\intercal \mathbf{M}_3 (\bar{\mathbf{u}}_j - \bar{\mathbf{u}}_j^{des}) \Big) \tag{4.3}$$

We can now subject the optimization problem in equation (4.2) to constraints:

$$\bar{\mathbf{x}}_{j+1} = \mathbf{A}^*\bar{\mathbf{x}}_j + \mathbf{B}^*\bar{\mathbf{u}}_j \quad , \quad \bar{\mathbf{x}}_0 = \mathbf{x}_{curr} \tag{4.4}$$

$$\bar{\mathbf{u}}_j = \begin{cases} \in U, & \text{if } j \leq M \\ \bar{\mathbf{u}}_M, & \text{if } j > M \end{cases} \tag{4.5}$$

$$U := \{\mathbf{u} \in \mathbb{R}^m \,|\, \mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max}\} \tag{4.6}$$

$$\bar{\mathbf{x}}_j \in X := \{\mathbf{x} \in \mathbb{R}^n \,|\, \mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max}\} \tag{4.7}$$

Constraint (4.4) imposes the linearized and discretized dynamics of our prediction model onto the optimization problem. Values n and m correspond to the number of states and inputs in our system. Weighting matrices $\mathbf{M}_1$ through $\mathbf{M}_3$ can be used to prioritize tracking certain state or input trajectories over others in the system. Constraint (4.5) functionally restricts the number of independent control inputs in $\bar{\mathbf{u}}$ to be equal to the

number of steps in our control horizon M that is $\bar{\mathbf{u}} = [\bar{\mathbf{u}}_0, \bar{\mathbf{u}}_1, ..., \bar{\mathbf{u}}_M, ..., \bar{\mathbf{u}}_M]$. This significantly simplifies the optimization problem by decreasing the number of values to solve for. Constraints (4.6) and (4.7) allow us to build input and state constraints directly into our controller as opposed to the technique used previously in Section 3.3.4. With every loop of our flight controller, we take in the current state of our system $\mathbf{x}_{curr}$ and solve problem (4.2) subject to constraints (4.4) through (4.7). We then apply the first control input $\bar{\mathbf{u}}_0$ from sequence $\bar{\mathbf{u}}$ to our actual system and repeat the whole process at the next loop of the controller. Numerous different forms of cost function J can be used and we can also apply additional constraints on our system's inputs and states. A more detailed summary of the equations for setting up a model predictive controller for a nonlinear system that has been linearized and discretized is presented in [56].

MPC has previously been applied to quadrotor-slung load systems. In [58], the authors linearized the equations of motion of the system about the hovering condition. Their controller implements constraints on the thrust from each motor as well as the quadrotor's height, pitch, roll and angular velocity. As with baseline Controller B presented in Section 3.4.2, the proposed controller in [58] uses a desired payload motion history whereby the payload is always directly below the quadrotor. The proposed controller is demonstrated in simulation and with flight tests. It is important to highlight though that for the flight tests the controller was implemented off-board and commands were streamed to the vehicle. The authors do note though that MPC has been demonstrated using only onboard computations for a quadrotor drone. Another implementation of MPC for a quadrotor-slung load system is presented in [59]. Here the authors also used off-board computations and incorporated a downward facing camera into their controller design to detect the motion of the payload. In a more recent publication [60], a controller was developed for a quadrotor-slung load system with a focus on obstacle avoidance and navigation through cluttered environments. The authors performed flight tests with their proposed system but encountered significant payload swinging throughout as their controller was not trying to dampen payload swinging at all.

4.2 Implementation

In order to implement our model predictive controller we relied heavily on MATLAB's built in model predictive control toolbox. We first generated a model that could be linearized and discretized to predict the behavior of our system. We then set constraints and chose an appropriate cost function to complete the optimization problem. We then

tuned the weights within our cost function to achieve a desirable overall performance of the quadrotor-slung load system in simulation, as will be discussed in Section 4.2.4.

4.2.1 Prediction Model

A key requirement for a model predictive controller is developing a linearized model for a quadrotor-slung load system. The model proposed in Section 3.2 is not well suited to this process since it requires computing tension forces within each cable. This greatly complicates the process of linearizing the model. We thus implemented a simpler model of a quadrotor-slung load system that could more readily be linearized. The proposed model is based on [38]. The main differences between this model and that presented in Section 3.2 is that this model incorporates only a single cable and treats it as an inelastic rod. The rod is also assumed to be mounted on one end at the quadrotor's center of mass and at the other end to a point mass payload. Thus the swinging motion of the payload will not directly affect the attitude dynamics of the quadrotor. The proposed model also employs a East North Up convention for the reference frames. We denote the unit vector pointing from the quadrotor center of mass to the payload along the cable resolved in the inertial frame as $\hat{\mathbf{c}}$. We also parameterize the attitude of the quadrotor frame $\mathcal{F}_Q$ relative to the inertial frame $\mathcal{F}_I$ using Euler angles (ϕ, θ, ψ) where ϕ is the roll angle, θ is the pitch angle and ψ is the yaw angle. We follow the same convention used in the model predictive controller presented in [58] to evaluate the direction cosine matrix $\mathbf{R}_{IQ}$ as a function of our Euler angles.

$$\mathbf{R}_{IQ} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \right)^{\mathsf{T}} \tag{4.8}$$

$$\mathbf{R}_{IQ} = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{4.9}$$

Based on [38] and [58], the equations of motion of the quadrotor-slung load system can be summarized as follows.

$$(m^q + m^p)(\ddot{\mathbf{r}}^q + g\mathbf{z}^I) = f\mathbf{R}_{IQ}\mathbf{z}_Q^Q + \frac{fm^p}{m^q}(\hat{\mathbf{c}}^\times \mathbf{R}_{IQ}\mathbf{z}_Q^Q)^\times \hat{\mathbf{c}} + m^p l \boldsymbol{\Omega}^\mathsf{T}\boldsymbol{\Omega}\hat{\mathbf{c}} \tag{4.10}$$

$$\mathbf{z}^I = \mathbf{z}_Q^Q = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.11}$$

$$\dot{\hat{\mathbf{c}}} = \boldsymbol{\Omega}^{\times}\hat{\mathbf{c}} \tag{4.12}$$

$$m^q l\dot{\boldsymbol{\Omega}} = -f\hat{\mathbf{c}}^{\times}\mathbf{R}_{IQ}\mathbf{z}_Q^Q \tag{4.13}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \frac{1}{\cos\theta}\begin{bmatrix} \cos\theta & \sin\phi\sin\theta & \cos\phi\sin\theta \\ 0 & \cos\phi\cos\theta & -\sin\phi\cos\theta \\ 0 & \sin\phi & \cos\phi \end{bmatrix}\boldsymbol{\omega}^Q \tag{4.14}$$

$$\mathbf{J}^q\dot{\boldsymbol{\omega}}^Q + \boldsymbol{\omega}^{Q\times}\mathbf{J}^q\boldsymbol{\omega}^Q = \mathbf{m}^q \tag{4.15}$$

where $\mathbf{z}^I$ and $\mathbf{z}_Q^Q$ are unit vectors parallel to the z axes of frames $\mathcal{F}_I$ and $\mathcal{F}_Q$ resolved in their respective frames. The angular velocity of the cable resolved in the inertial frame is given as $\boldsymbol{\Omega}$. The angular velocity of the quadrotor resolved in $\mathcal{F}_Q$ is $\boldsymbol{\omega}^Q$. f is the overall thrust force of the quadrotor's motors. In this setup since the cable is mounted at the center of mass of the quadrotor, $\mathbf{m}^q$ is simply the applied moment from the quadrotor's motors resolved in $\mathcal{F}_Q$. We thus get the following mapping between the thrust input from each of the four motors (f_0, f_1, f_2, f_3) and the resulting thrust and moment.

$$\begin{bmatrix} f \\ \mathbf{m}^q \end{bmatrix} = \begin{bmatrix} f \\ \mathrm{m}_x^q \\ \mathrm{m}_y^q \\ \mathrm{m}_z^q \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -a & a & a & -a \\ -a & a & -a & a \\ \frac{k^q}{k^t} & \frac{k^q}{k^t} & -\frac{k^q}{k^t} & -\frac{k^q}{k^t} \end{bmatrix}\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{4.16}$$

where a is the distance between the center of mass of the quadrotor and the motor along the x or y axis. The final state $\mathbf{x}$ and inputs $\mathbf{u}$ for our model predictive controller are

$$\mathbf{x} = \begin{bmatrix} \mathbf{r}^q \\ \dot{\mathbf{r}}^q \\ \phi \\ \theta \\ \psi \\ \boldsymbol{\omega}^Q \\ \hat{\mathbf{c}} \\ \boldsymbol{\Omega} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{4.17}$$

Using equations (4.9) and (4.16) as well as the definition of the $\times$ superscript given in equation (3.4) we can recast and expand equations (4.10) through (4.15) such that they

are only a function of elements of state $\mathbf{x}$ and inputs $\mathbf{u}$. Using MATLAB, the resulting model can be linearized about the hovering condition with

$$\mathbf{x}^{equilb} = \begin{bmatrix} \mathbf{0}^{12\times 1} \\ \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \\ \mathbf{0}^{3\times 1} \end{bmatrix} \quad , \quad \mathbf{u}^{equilib} = \frac{(m^q + m^p)g}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{4.18}$$

The only nonzero state in $\mathbf{x}^{equilib}$ corresponds to the cable unit vector $\hat{\mathbf{c}}$ pointing directly downwards based on an East North Up convention. We can thus express a linearized model of quadrotor-slung load dynamics in the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{4.19}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} = \begin{bmatrix} \mathbf{r}^q \\ \dot{\mathbf{r}}^q \\ \phi \\ \theta \\ \psi \\ \hat{\mathbf{c}} \end{bmatrix} \tag{4.20}$$

where for output $\mathbf{y}$ we select matrix $\mathbf{C}$ such that we extract elements $\mathbf{r}^q$, $\dot{\mathbf{r}}^q$, ϕ, θ, ψ, $\hat{\mathbf{c}}$ from state $\mathbf{x}$. We then discretize our system using the zero order hold method to obtain the form needed for equation (4.4).

4.2.2 Constraints and Cost Function

The other two main requirements for establishing our model predictive controller are to impose constraints on the inputs and states of our system and to establish a cost function for our optimizer to solve. In the example shown in Section 4.1 for instance we solve the optimization problem (4.2) with cost function (4.3) subject to input constraint (4.6) and state constraint (4.7).

A key constraint to include in this system is the maximum and minimum thrust values each motor can supply. Thus for for inputs f_0 through f_3 we impose

$$k_t\Omega_{min}^2 \leq f_i \leq k_t\Omega_{max}^2 \tag{4.21}$$

where the values of k_t, Ω_{min} and Ω_{max} are taken from Table 3.1. The MATLAB implementation environment that we are using to create and simulate our model predictive controller also allows us to impose limits on the maximum change in magnitude that an input can have from one step to the next (i.e. input rate constraints). We initially did not impose this type of constraint on our model predictive controller but during preliminary simulations we found that this produced unrealistic results. The model predictive controller's inputs would jump between the maximum and minimum allowable thrust values at a high frequency that would be unrealistic for a quadrotor to replicate. For this preliminary investigation we chose to limit the control inputs such that it would take at least one second for the thrust to go from the minimum to the maximum value. This was a conservative estimate as we did not have experimental data to rely on for the true rate limit. With both of these constraints in place, the requested thrust inputs for our model predictive controller no longer oscillate between the maximum and minimum values at high frequencies. During preliminary testing of our controller we also considered applying constraints to the pitch (θ) and roll (ϕ) Euler angles within our state $\mathbf{x}$ shown in equation (4.17). From MATLAB's online resources though it was recommended to avoid state constraints unless absolutely necessary and to instead restrict the states through controller tuning [61]. We thus did not include any state constraints in our implementation.

The general form of the cost function used in MATLAB's model predictive toolbox is given in [62]. Unlike the form presented in equation (4.3) MATLAB's toolbox considers errors between the predicted sequence of system outputs $\bar{\mathbf{y}}$ and their corresponding desired values $\bar{\mathbf{y}}^{des}$. As mentioned previously, we have chosen output $\mathbf{y}$ such that it contains states $\mathbf{r}^q$, $\dot{\mathbf{r}}^q$, ϕ, θ, ψ, $\hat{\mathbf{c}}$ for our system. The MATLAB cost function thus takes on the following form.

$$J(\bar{\mathbf{y}}) = \sum_{j=0}^{N} \left((\bar{\mathbf{y}}_j - \bar{\mathbf{y}}_j^{des})^\top \mathbf{M}\mathbf{M} (\bar{\mathbf{y}}_j - \bar{\mathbf{y}}_j^{des}) \right) \tag{4.22}$$

where N is the number of steps in our control horizon and $\mathbf{M}$ is a diagonal weighting matrix. Adjusting the values of each term in matrix $\mathbf{M}$ allows us to tune the model predictive controller by emphasizing tracking certain outputs over others.

In order to obtain values for $\bar{\mathbf{y}}^{des}$ we used the following approach. Reference values for $\mathbf{r}^q$ and $\dot{\mathbf{r}}^q$ are obtained directly from the trajectory generator by evaluating at future time steps. The references trajectories for Euler angles ϕ, θ and ψ are obtained by implementing the differential flatness computations in Appendix A to solve for $\mathbf{R}_{IQ}^{df}$ at every future time step and then using the parameterization given in equation (4.9). The reference values for the cable unit vector $\hat{\mathbf{c}}$ are obtained using the same second order model approach described in Section 3.3.1.1. Once we have solved for $\mathbf{r}^{dp}$, we find that

at any time step

$$\hat{\mathbf{c}} = \frac{\mathbf{r}^{dp} - \mathbf{r}^{dq}}{\sqrt{(\mathbf{r}^{dp} - \mathbf{r}^{dq})^{\intercal}(\mathbf{r}^{dp} - \mathbf{r}^{dq})}} \tag{4.23}$$

For our MATLAB implementation of the model predictive controller we precompute all desired values and save these as a large list. As the simulation proceeds, the controller will access a subset of this list in order to obtain the $\bar{\mathbf{y}}^{des}$ values based on the current time and the number of steps into the future that it needs. This differs from our previous implementation in Chapter 3 where we solved for the desired swinging motion of the payload onboard the drone at each time step. From a practical standpoint this means that our model predictive controller will require extra pre-computations compared to Controller C. It would be important to ensure that these extra pre-computations can be run fast enough so that the system can still safely recompute flight trajectories while flying at speed. This requirement may also mean that we would need a flight computer with more memory to be able to store this information for our model predictive controller.

4.2.3 Simulator Setup

Our model predictive controller is built around the model presented in Section 4.2.1 and in theory we could use this same model to create our simulation environment for testing purposes. This would represent an idealized situation though and would not necessarily capture the fact that our real world dynamics will never perfectly match those of our controller's model. Thus while our controller uses the quadrotor-slung load system model presented in Section 4.2.1, we will use the model developed in Section 3.2 for our simulation environment. As mentioned before, the main difference between these models is that our simulation environment allows for offsets between the mounting point of the cable on both the quadrotor and the payload and it allows the cable to effectively go slack. We reuse the same properties for our simulated quadrotor-slung load system given in Table 3.1. Whenever inertial properties are required in our predictive model such as in equations (4.10), (4.13) and (4.15) we scale these by a factor of 0.9 to introduce additional error between our controller's model and the simulated system dynamics. Based on our experience with test flying actual quadrotors, we assumed that our model predictive controller would run at 100Hz and thus discretized our model based on a sampling time of 0.01s. We similarly break down the simulated motion into discretized steps of 0.01s. Thus during pre computation steps, we solve for the reference output $\mathbf{y}$ for every time step in the simulation and store these values for use within our controller.

4.2.4 Controller Tuning

Our next step was to tune the proposed model predictive controller. In our cost function (4.22) we can select the number of steps in the prediction horizon, N, as well as the diagonal entries of weighting matrix $\mathbf{M}$. Furthermore, in constraint (4.5) we can select the number of steps in our control horizon, M. From preliminary testing we found that values of $N = 20$ and $M = 3$ appeared to produce good results for our system. We thus decided to lock in these values and concentrate the majority of our tuning efforts on adjusting the entries of diagonal control matrix $\mathbf{M}$. Matrix $\mathbf{M}$ has dimensions 12×12 with the diagonal entries corresponding to weights for errors in x, y, z position, x, y, z velocity, Euler angles ϕ, θ, ψ and the x, y, z components of unit vector $\hat{\mathbf{c}}$. In order to tune our controller we first considered a simple scenario where the quadrotor must maintain its position while the payload is given an initial relative velocity of $[3 \quad 0 \quad 0]^\intercal \mathrm{m\,s^{-1}}$. We initially tested this setup with the velocity and Euler angle gains set to zero and found that the controller quickly became unstable. In particular, we found that the quadrotor was making large pitch and roll motions. Setting non-zero Euler angle gains significantly improved the stability of our system in this setup by effectively penalizing deviations of the pitch and roll angles away from zero. We also found that introducing non zero weights for velocity errors significantly improved the speed at which the quadrotor recovered to its original position. We further adjusted the relative magnitude of all elements to achieve a good balance of swing suppression coupled with a quick recovery time to the starting position. One observation that me made during this process is that our model predictive controller tends to drop slightly in altitude. This makes sense since our controller does not incorporate any integral control element for the z position and has an imperfect estimate of its true weight. This was not a major issue though as we are primarily concerned with accurate tracking of trajectories in the XY plane. During the tuning process, we found that our model predictive controller was very sensitive to swinging disturbances and at times went unstable when large initial relative velocities were imposed on the payload. We found for instance that with our initial tuning, our model predictive controller was unable to track the maneuver used in Simulation 3 in Section 3.4.3. We attempted to manually adjust our gains for this simulated scenario but could not obtain a satisfactory tuning. We ultimately opted to test our tuning on with a different maneuver with slightly smaller swinging disturbances. This allowed us to make further adjustments to our tuning before settling on a satisfactory solution. Our main goal was to achieve good trajectory tracking for the quadrotor while also suppressing swinging motion in the payload. The

final control gains along with their units are presented below:

$$\mathbf{M} = \text{diag}\left[6\,(\tfrac{1}{\sqrt{\text{m}}}),\ 6\,(\tfrac{1}{\sqrt{\text{m}}}),\ 6\,(\tfrac{1}{\sqrt{\text{m}}}),\ 2\,(\sqrt{\tfrac{\text{s}}{\text{m}}}),\ 2\,(\sqrt{\tfrac{\text{s}}{\text{m}}}),\ 2\,(\sqrt{\tfrac{\text{s}}{\text{m}}}),\ 2\,(\tfrac{1}{\sqrt{\text{rad}}}),\ 2\,(\tfrac{1}{\sqrt{\text{rad}}}),\ 1\,(\tfrac{1}{\sqrt{\text{rad}}}),\ 9,\ 9,\ 9\right] \tag{4.24}$$

4.3 Simulation Results

Our goal in this section is to compare Controller C from Chapter 3 to our new model predictive controller. Building on the approach presented in Sections 3.4.2 and 3.4.3, we create two model predictive controllers to compare against Controller C. Controller D is a model predictive controller that uses the method outlined in Section 4.2.2 to compute a reference swinging motion. Controller E is identical to Controller D except that when computing unit vector $\hat{\mathbf{c}}$ for our reference trajectory we will impose $\hat{\mathbf{c}} = [0 \quad 0 \quad -1]^\intercal$ instead of using equation (4.23). Functionally this means that Controller E will view any swinging motion in the payload as an error to be corrected while Controller D will try to track a reference swinging motion paired with the desired flight trajectory. This is analogous to the setup we established in Sections 3.4.2 and 3.4.3 with Controllers B and C.

Once our tuning was established, we ran Controllers C, D and E through a series of simulations. Unlike the simulations in Section 3.4.3 though, we will not impose measurement noise on any of the controllers. We opted for this modification since the implementation of Controllers C, D and E in MATLAB are significantly different and this made it more complicated to ensure that each was being affected by noise in the same way. The parameters of the quadrotor and its slung load are taken from Table 3.1. Controller C uses the same gains outlined in Table 3.2 while Controllers D and E use the values from Section 4.2.4. For our first simulation, we will compare the capabilities of Controllers C, D and E for damping out an initial swinging disturbance on the payload while maintaining their current position. This disturbance is generated by imposing an initial relative velocity between the center of mass of the quadrotor and payload of $[-2 \quad -1 \quad 0]^\intercal \text{m}\,\text{s}^{-1}$. Figure 4.1 shows the resulting swing angle of the payload for this simulation. Figure 4.2 shows the magnitude of the error in the XY plane between the quadrotor's actual position and the desired position.

In Figures 4.1 and 4.2, the performance of Controllers D and E is identical as in this simulation there is no requested quadrotor motion and thus no reference swinging motion for Controller D. From Figure 4.1, it is apparent that our original controller (C) is able

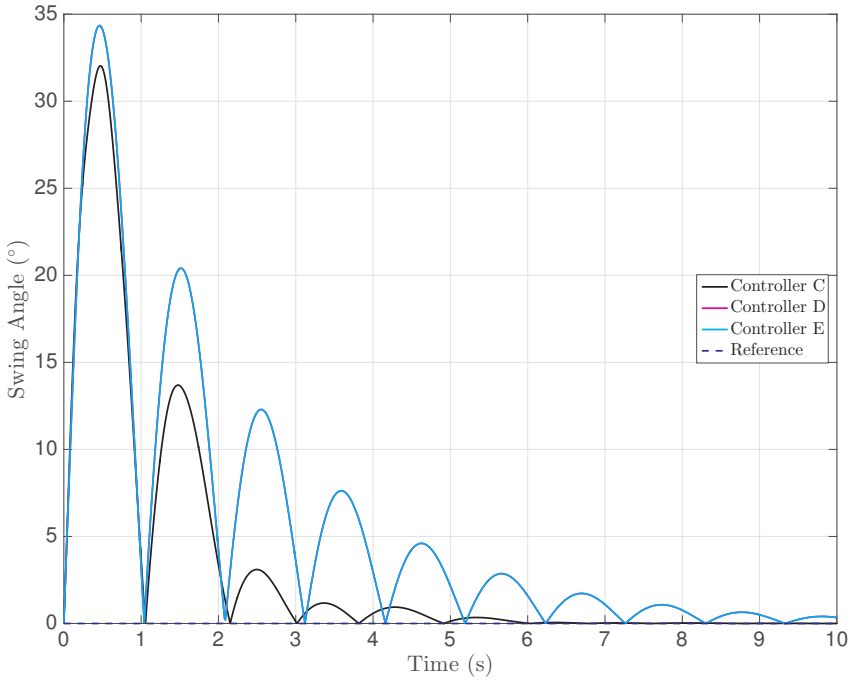

FIGURE 4.1: Simulation 4: Time History of Swing Angle

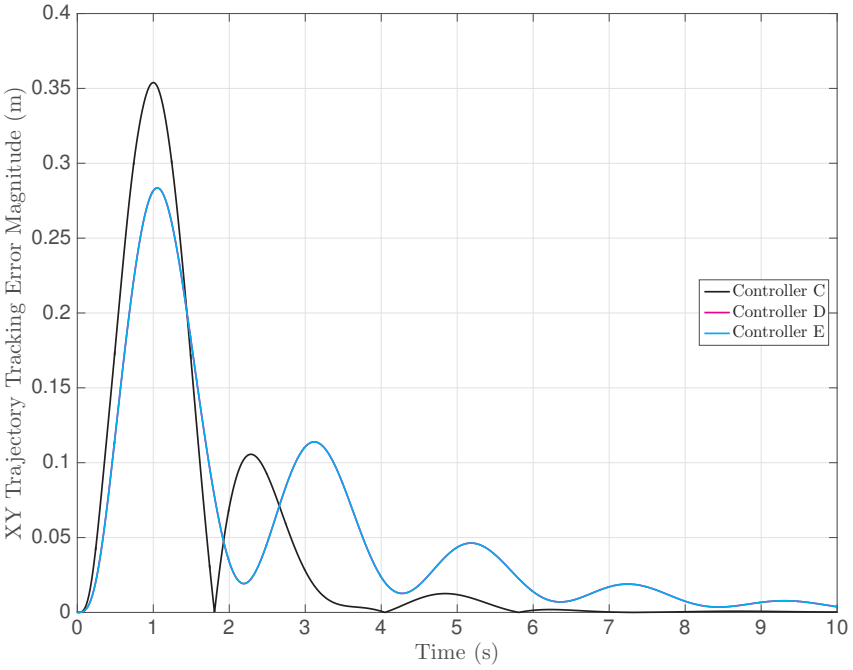

FIGURE 4.2: Simulation 4: Time History of Quadrotor Trajectory Tracking Error Magnitude

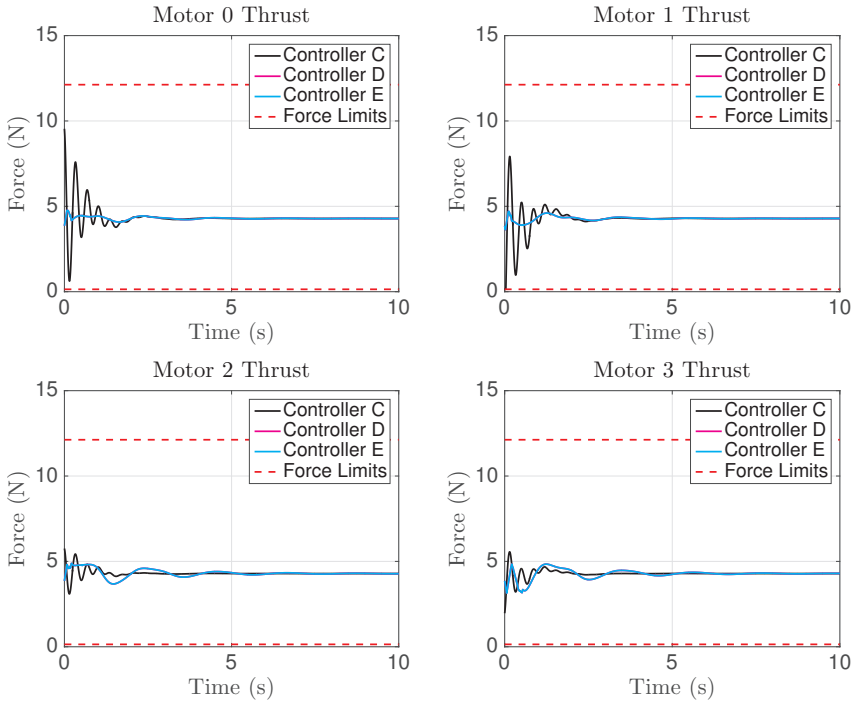

FIGURE 4.3: Simulation 4: Time History of Motor Thrust Forces

to dampen the swinging disturbance faster than either model predictive controller. We can also see from Figure 4.2 that the model predictive controllers (D and E) have slightly more difficulty returning to the desired starting position than Controller C. Figure 4.3 shows the thrust force from each motor throughout this simulated flight. Once again, in Figure 4.3 the performance of Controllers D and E are identical. We notice though that the requested force values from the model predictive controllers change more gradually than those from Controller C. This behaviour is due to the input rate constraint that we imposed while designing Controllers D and E. Only Controller C appears to reach thrust saturation on one of its motors at time zero. Upon further analysis we also found that both Controllers D and E temporarily reach the imposed input rate limit for all four motors within the first second of simulation but not afterwards.

We now compare the three controllers for tracking an input-shaped flight trajectory. In this simulation we will have the quadrotor start at the origin with an initial velocity of $[6 \quad 0 \quad 0]^\intercal \mathrm{m\,s^{-1}}$. Our goal will be to perform an aggressive turn by reaching position $[3 \quad -4 \quad 0]^\intercal \mathrm{m}$ with a final velocity of $[-2 \quad 4 \quad 0]^\intercal \mathrm{m\,s^{-1}}$ within a total time of 7s. The initial and final accelerations of the quadrotor are set to zero and we do not impose any relative velocity between the quadrotor and the payload at the start. We employ a two-hump EI shaper for this maneuver and we simulate the system behaviour for a total of 10s. Figure 4.4 shows a top view of the resulting motion of the quadrotor for all three

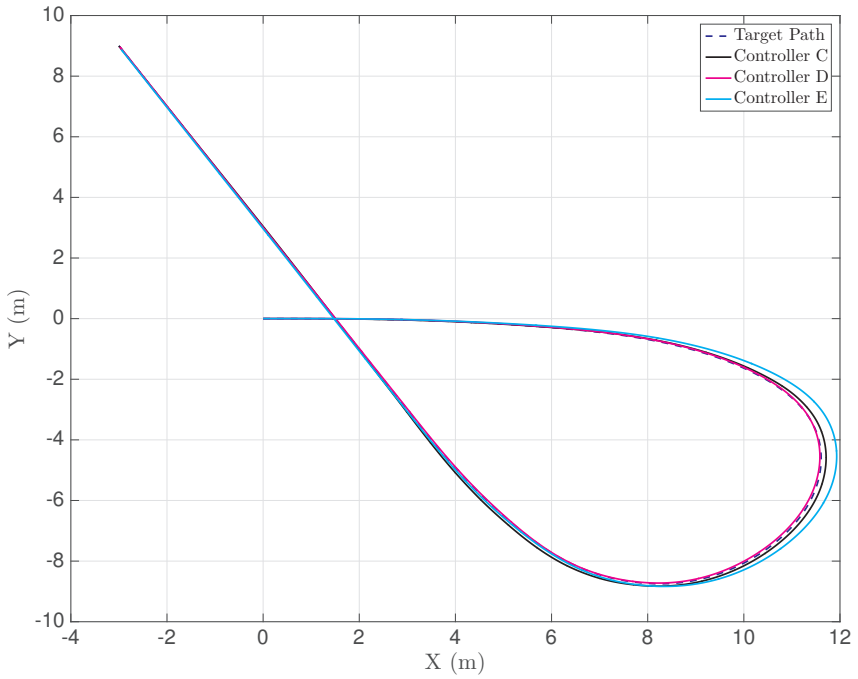

FIGURE 4.4: Simulation 5: Top View of Quadrotor Motion

controllers. Figure 4.5 compares the swing angle for all three controllers throughout the maneuver. Figure 4.6 shows the position tracking error magnitude for the XY plane. Figure 4.7 compares the thrust force requested for each motor by each controller.

From Figure 4.5 we can see that all three controllers experience nearly the same swinging motion of their payload and that this motion closely resembles the computed reference. In Figure 4.6 though we can see that all three controllers experienced significantly different tracking errors. The large tracking error for Controller E compared to Controller D can be seen in both Figures 4.4 and 4.6. We propose that this difference is due to the fact that Controller E is constantly trying to eliminate any swinging motion that it sees in the payload while Controller D has a reference swinging trajectory that it attempts to track. More specifically, Controller E attempts to chase after the swinging payload causing it to make a wider turning motion than Controller D as seen in Figure 4.4. This performance mirrors the results from Simulation 2 in Section 3.4.3 where Controller B had difficulty tracking an input-shaped u-turn trajectory because it too tried to eliminate any swinging motion that it saw in the payload. From Figure 4.6 we can also see that Controller D has lower tracking error than Controller C throughout the majority of the simulation. The input force magnitude plots in Figure 4.7 show that the model predictive controllers can produce requested inputs that do not change as aggressively or reach as high magnitudes as Controller C. This is especially apparent for Motors 0 and 1.

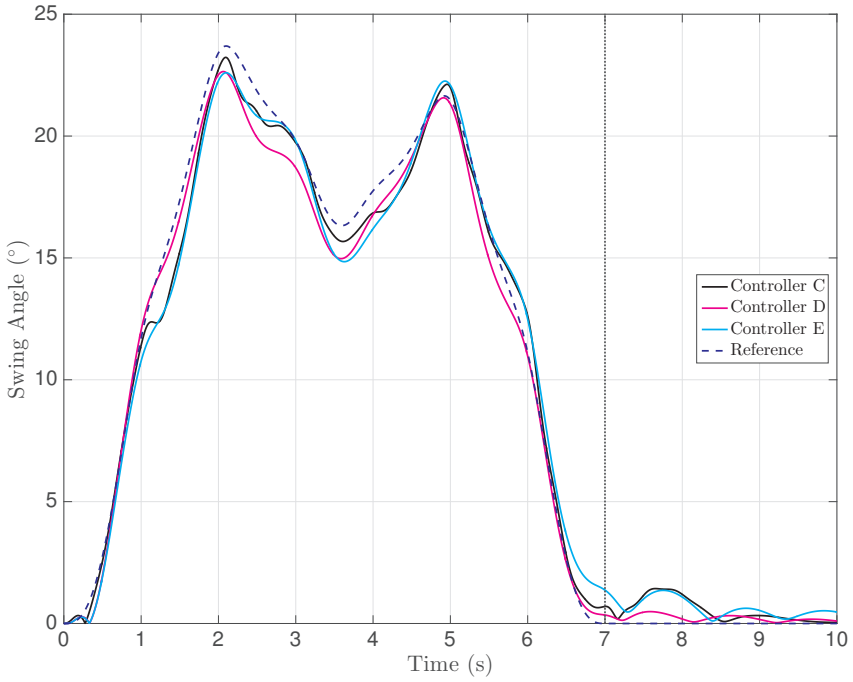

FIGURE 4.5: Simulation 5: Time History of Swing Angle

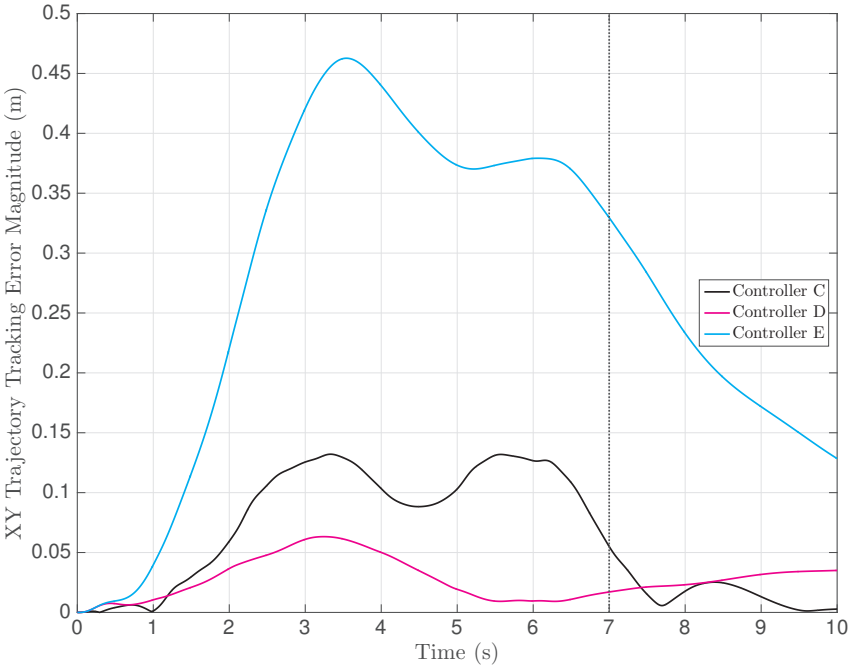

FIGURE 4.6: Simulation 5: Time History of Quadrotor Trajectory Tracking Error Magnitude

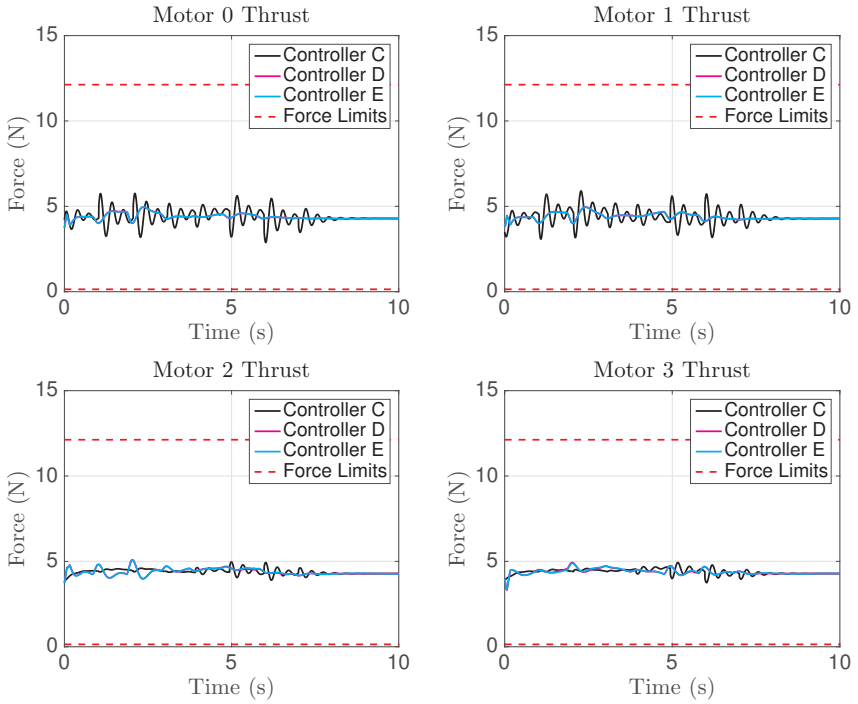

FIGURE 4.7: Simulation 5: Time History of Motor Thrust Forces

We now repeat this previous simulation but introduce an initial relative velocity between the center of mass of the quadrotor and payload of $[-2 \quad -1 \quad 0]^{\intercal}\,\mathrm{m\,s^{-1}}$. For the resulting simulation, Figure 4.8 shows a top view of the motion of the quadrotor for all three controllers. Figure 4.9 compares the swing angle for all three controllers. Figure 4.10 shows the position tracking error magnitude for the XY plane. Figure 4.11 compares the thrust force requested for each motor by each controller.

From Figure 4.9 we can see that all three controllers appear to perform equally well at damping the initial swinging disturbance. Controllers D and E experience larger swing amplitudes than C but they appear to be able to effectively recover. However, in Figure 4.10 we see that Controller E has significantly higher flight trajectory tracking error than Controller D. This mirrors the performance of Controllers B and C from Simulation 3 in Section 3.4.3. The trajectory tracking performance of Controller D appears to be better than Controller C throughout most of the simulation and reaches a lower peak error value. Once again, we can see in Figure 4.11 that the force requirements for Controllers D and E tend to change more gradually than those of Controller C especially for Motors 0 and 1 for this simulation. The high initial forces in Controller C are due to the fact that it uses a measurement of the high initial payload velocity in order to compute a control force to apply.

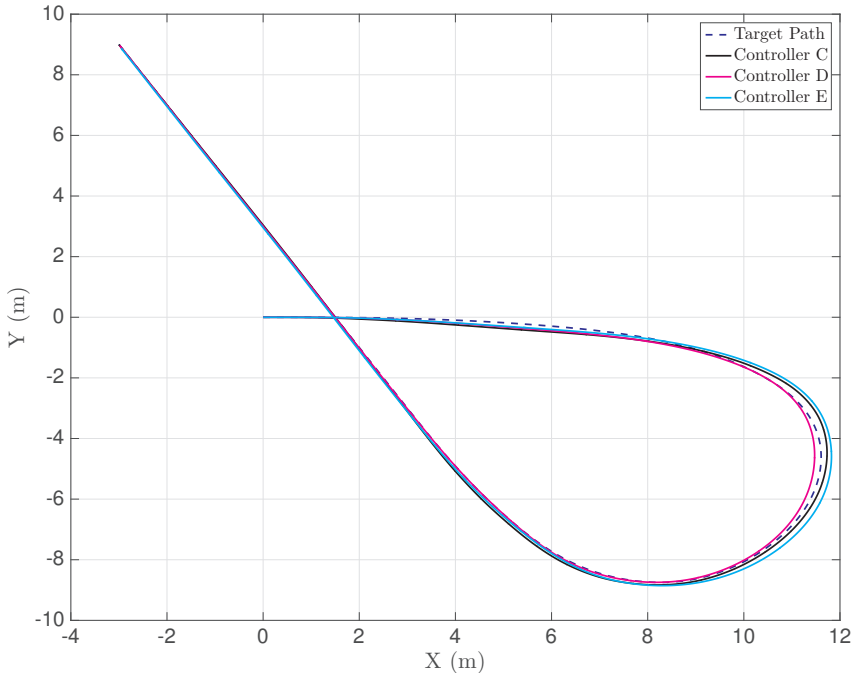

FIGURE 4.8: Simulation 6: Top View of Quadrotor Motion

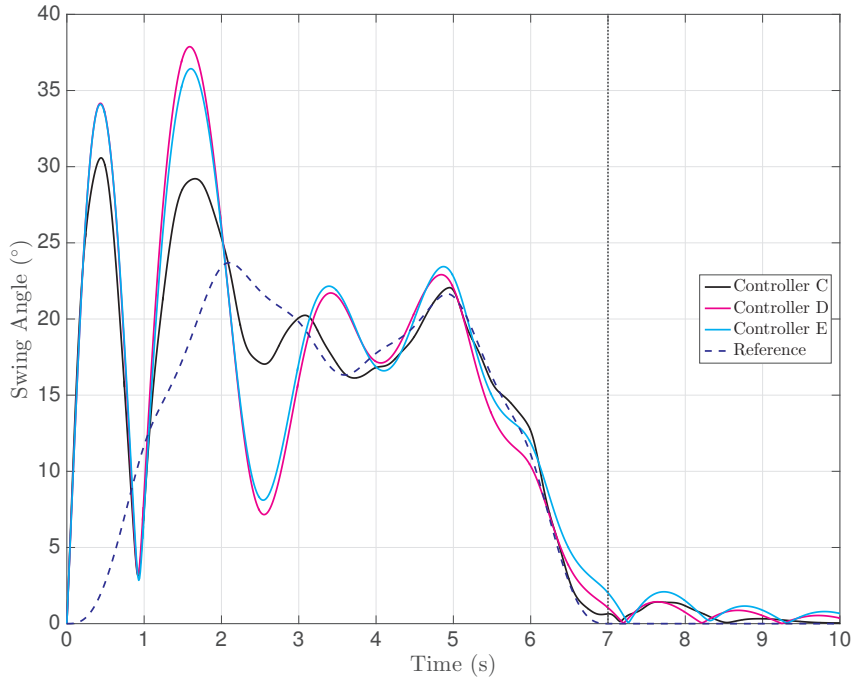

FIGURE 4.9: Simulation 6: Time History of Swing Angle

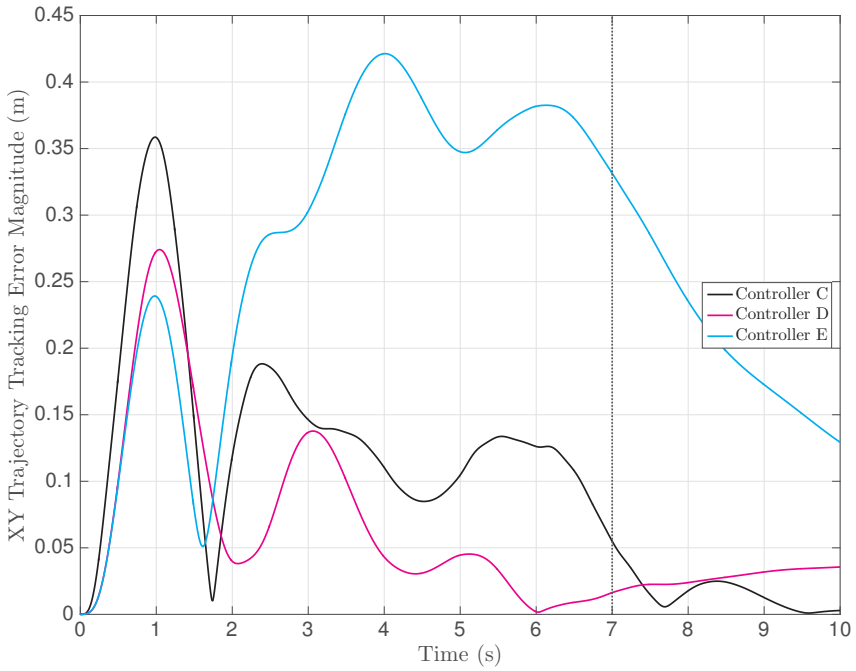

FIGURE 4.10: Simulation 6: Time History of Quadrotor Trajectory Tracking Error Magnitude

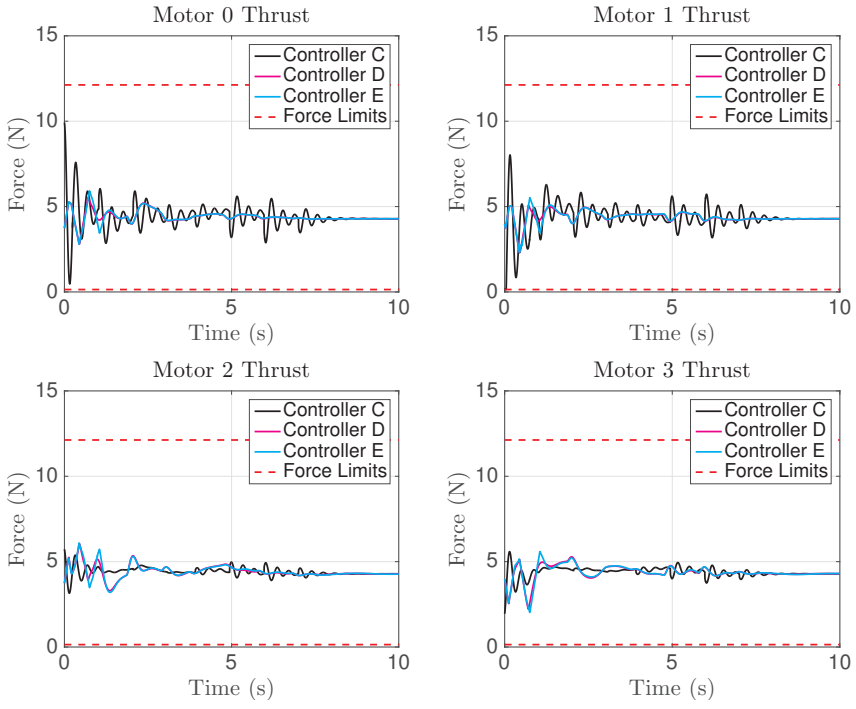

FIGURE 4.11: Simulation 6: Time History of Motor Thrust Forces

Overall, this section demonstrates that we can improve the input-shaped trajectory tracking performance of a model predictive controller by supplying it with a computed reference swinging motion. In both Figures 4.6 and 4.10, we see that Controller D is able to track the prescribed flight trajectory better than Controller E. This is due to the fact that Controller D is given access to both a desired flight trajectory as well as the corresponding desired swinging motion of the payload as computed by our second order model. The fact that the quadrotor and payload swing trajectories are coupled together means that Controller D will be more likely to find a feasible solution that will allow it to track both when it tries to minimize the cost function in equation (4.22). In contrast, Controller E's desired swinging trajectory does not correspond to its desired quadrotor motion and it will thus converge to solutions that balance trying to meet these competing demands. This overall results in poorer trajectory tracking performance for Controller E. It is encouraging that these results mirror the performance improvement seen previously between controllers C and B in Section 3.4.3.

We need to be cautious when comparing the performance of Controllers C and D though. Controller C outperformed Controller D for the simulated position holding task subject to an initial swinging disturbance. Specifically, Controller C suppressed swinging faster and was able to recover to its starting position sooner than Controller D. For trajectory tracking simulations though, Controller D had lower tracking error in the XY plane while having similar swing suppression to Controller C. The main benefit of Controller D seen throughout all simulations is that its inputs do not change as aggressively as those for Controller C. Controller D is built with constraints on the rate of change of its inputs which may overall lead to a more reasonable input trajectory for an actual quadrotor to replicate. At the same time, the imposed constraint is conservative and may be hindering the capabilities of the quadrotor to track prescribed paths. For example, during tuning we found that we were unable to achieve good trajectory tracking and swing dampening when we replicated the scenario in Simulation 3 from Section 3.4.3 with Controller D. We also found upon further analysis that both Controllers D and E reached rate saturation on each motor at least once in Simulations 4, 5 and 6. The thrust input rates never dwell at this maximum rate for more than a few tenths of a second and usually decline as the simulation progresses. At the same time, because these rate constraints are being used within our optimization problem at every loop of the flight controller they may still be effecting the chosen control inputs throughout the entire simulation. It would thus be worth experimentally measuring the true limit for the rate of change of the quadrotor's thrust and using these values within our model predictive controller. This may improve the performance of Controller D relative to Controller C. This preliminary analysis also

did not explore how the performance of our model predictive controllers could potentially be improved by increasing the number of steps in our prediction and control horizons.

There are also some key implementation differences between Controllers C and D to consider. Controller D is inherently more computationally demanding to implement as it requires solving an optimization problem with each loop of the flight controller. This may mean that Controller D would need to run at a lower frequency than Controller C on an actual quadrotor drone which could alter its performance. As previously discussed, an implementation of Controller D would also involve additional pre-compution steps and may require additional flight computer memory to store reference data. Ultimately, we believe that additional development work should be done in simulation for the proposed model predictive controllers and that Controllers C and D should be compared with actual flight tests.

Chapter 5

Experimental Setup

This chapter discusses the process of taking the trajectory generation algorithm developed in Chapter 2 and the subsequent controller developed in Chapter 3 (Controller C) and implementing both on an actual quadrotor drone for validation. We begin by discussing the process of integrating both elements into the PX4 open source flight stack. We then provide an overview of the flight hardware used in our setup including a discussion of setting up a downward facing camera for payload detection. This chapter concludes with a discussion of our preliminary flight testing.

5.1 Trajectory Generator and Controller Implementation

In order to apply our proposed trajectory generator and controller to an actual drone we need to integrate these elements into a full flight stack and run them on a flight computer. Our lab's research drones all run the open source PX4 flight stack which is compatible with the Pixhawk family of flight controllers. PX4 is an open source full stack software package that enables remote control or autonomous operation of fixed wing and quadrotor drones. The flight stack has a built-in state estimator to fuse sensor data from an onboard inertial measurement unit (IMU), gyroscope and magnetometer while also being able to accommodate global positioning system (GPS) and motion capture data. The software stack is divided into modules that ultimately publish and read data from a set of topics. The PX4 flight stack provides a solid foundation for implementing our trajectory generator and flight controller. For this work we modified the baseline flight controller to accommodate the controller from Chapter 3 and incorporated a new

module to implement the trajectory generator from Chapter 2. The baseline controller is similar in structure to the controller developed in Chapter 3 which facilitates this task. In contrast, the model predictive controller developed in Chapter 4 (Controller D) would be significantly more challenging to implement within PX4 as it requires additional code to solve an optimization problem with each loop of the flight controller. For this reason we decided to focus primarily on implementing the simpler controller from Chapter 3.

The baseline quadrotor controller provided as part of the PX4 stack is divided into a position controller and an attitude controller which each have their own module. For our implementation though, we constructed our full controller within one module. During the implementation phase, we created three operating modes for our drone. The first operating mode provides manual control over the quadrotor's flight to facilitate takeoff and landing. This mode also serves as an important safety feature during autonomous flights. Autonomous takeoff with a slung payload presents its own set of challenges and is beyond the scope of this thesis. An example of a controller and trajectory generator dedicated to this problem is presented in [63]. Initially, we used the baseline PX4 flight controller for this manual flight mode. We found that this created issues with the overall code development process and later decided to implement this manual flight mode by modifying our proposed flight controller. In our final implementation, the attitude joystick inputs are scaled to obtain desired pitch, roll and yaw angles which are converted into the desired attitude quaternion $\mathbf{q}^{Qd}$ and in turn the desired direction cosine matrix $\mathbf{R}_{IQ}^{d}$ for subsequent use in our Attitude Controller from Section 3.3.3. For simplicity, during this operating mode we also skip the differential flatness computations and set $\boldsymbol{\omega}^{Qdf}$ and $\dot{\boldsymbol{\omega}}^{Qdf}$ to zero. The throttle joystick input is scaled to obtain F^{thrust}. Our second operating mode is an autonomous hovering mode. The position of the drone is sampled when the hovering mode is activated and the measured value is used as a setpoint for the drone's position, bypassing the trajectory generator element of our system. In a typical flight test, we takeoff and achieve hover under manual control then set the drone to the autonomous hovering mode. We found that a hovering mode was useful for tuning the flight controller's gains. The final operating mode autonomously tracks a prescribed motion generated using the techniques discussed in Chapter 2.

The process of implementing our flight controller and trajectory generator within the PX4 flight stack led us to make a series of changes to both elements. In our original presentation of Controller C [2], the Position Controller had no integral control element in equation (3.35). During preliminary flight trials, we found that the drone had difficulty maintaining its altitude after transitioning from manual control to autonomous hovering. This issue was largely resolved by adding an integral control element for the z motion of the drone

as shown in equation (3.35). The performance of our controller could likely be further improved by introducing a battery voltage compensation system as done in [64]. The drop in voltage on the battery affects the thrust generated by the motors meaning that over time larger control inputs will need to be requested to achieve hover. Our original Attitude Controller in [2] also used scalar control gains instead of diagonal control gain matrices in equations (3.42) and (3.45). This meant that the same gains were applied for pitch, roll and yaw errors. Our quadrotor's propellers though have a low moment constant k_q resulting in a limited ability to generate yawing moments. Combined with the high control gains, this meant that our Attitude Controller would quickly reach saturation for small yaw errors. The saturation problem was further compounded by integral windup for yaw errors. Our solution was to decouple the yaw gains in our Attitude Controller from the pitch and roll gains as well as eliminating the integral control element for our yaw attitude. These changes dramatically improved the flight performance of our system and are reflected in the Attitude Controller presented in Section 3.3.3. In our final controller tuning, shown in Table 3.2, for instance notice how the proportional control gains for pitch and roll errors in $\mathbf{K}_R^{att}$ are significantly higher than the gain for yaw error.

A further modification we made during this stage involved the trajectory generator. The implementation done in MATLAB in order to perform simulations in Chapters 2 and 3 as well as in [2] only allows for a single maneuver to be studied. In a realistic flight scenario the algorithms developed in Chapter 2 would be used as part of a larger algorithm incorporating obstacle detection as well as information about the required task and allowable flight paths. The development of such a system is beyond the scope of this thesis but we still wanted to be able to have the drone perform flights involving multiple input-shaped motion trajectories that need to be computed mid flight. We thus developed Algorithm 4 to schedule and run multiple maneuvers sequentially. Algorithm 4 incorporates the previously developed Algorithms 1 and 2 while allowing one maneuver to interrupt another to simulate a scenario where a sudden course change is required. Algorithm 4 is run as a separate module in the PX4 flight stack for achieving autonomous flight when activated by the pilot. To simulate an interrupted maneuver we simply need to set $T^2_{start} - T^1_{start} < T^1_F$ as an example. Maneuvers are always generated using the most recently measured state of the drone. This ensures that even if the quadrotor cannot perfectly track a prescribed motion its next flight trajectory will always begin at its current position. Furthermore, the extension technique proposed in Section 2.3 means that our drone will still have a prescribed flight trajectory even if there are gaps in the prescribed maneuver schedule. In our implementation, the desired position, velocity, acceleration, jerk and snap of the quadrotor drone are published to a topic within the flight stack which can then be read by the controller module.

Algorithm 4: Maneuver Scheduling Algorithm

input : Maneuver start times (T^1_{start}, T^2_{start}, ...), Maneuver durations (T^1_F, T^2_F, ...),
Maneuver end states: $\begin{bmatrix} x_F^{IS\,1} \\ y_F^{IS\,1} \\ z_F^{IS\,1} \end{bmatrix} \begin{bmatrix} \dot{x}_F^{IS\,1} \\ \dot{y}_F^{IS\,1} \\ \dot{z}_F^{IS\,1} \end{bmatrix} \begin{bmatrix} \ddot{x}_F^{IS\,1} \\ \ddot{y}_F^{IS\,1} \\ \ddot{z}_F^{IS\,1} \end{bmatrix}, \begin{bmatrix} x_F^{IS\,2} \\ y_F^{IS\,2} \\ z_F^{IS\,2} \end{bmatrix} \begin{bmatrix} \dot{x}_F^{IS\,2} \\ \dot{y}_F^{IS\,2} \\ \dot{z}_F^{IS\,2} \end{bmatrix} \begin{bmatrix} \ddot{x}_F^{IS\,2} \\ \ddot{y}_F^{IS\,2} \\ \ddot{z}_F^{IS\,2} \end{bmatrix}$, ... , Period of oscillation for natural frequency of swinging (T_d),
$\zeta_{IS} = 0$, $V_{tol} = 0.05$, Input shaper type (ZV, two-hump EI)

Initialization : $T_{trigger} = T^1_{start}$, $maneuverID = 1$

Measure starting position of the drone $\mathbf{r}^q_{start} = \mathbf{r}^q$

while *True* **do**
  Record current state of the drone: $\mathbf{r}^q$, $\mathbf{v}^q$, $\mathbf{a}^q$
  Obtain current time t_{global}, measured from when started autonomous flight mode
  if $t_{global} < T^1_{start}$ **then**
    Hold starting position: $\mathbf{r}^{dq} = \mathbf{r}^q_{start}$, $\mathbf{v}^{dq} = \mathbf{a}^{dq} = \dddot{\mathbf{r}}^{dq} = \ddddot{\mathbf{r}}^{dq} = \mathbf{0}$
  else
    if $t_{global} > T_{trigger}$ **then**
      Run : Algorithm 1 for x, y and z axes using desired end state from maneuver $maneuverID$, store all outputs
      $T_{maneuver\,start} \leftarrow T_{trigger}$
      $maneuverID \leftarrow maneuverID + 1$
      $T_{trigger} \leftarrow T^{maneuverID}_{start}$
    end
    $t_{eval} \leftarrow t_{global} - T_{maneuver\,start}$
    Run : Algorithm 2 for x, y and z axes to obtain $\mathbf{r}^{dq}$, $\mathbf{v}^{dq}$, $\mathbf{a}^{dq}$, $\dddot{\mathbf{r}}^{dq}$, $\ddddot{\mathbf{r}}^{dq}$
  end
  output: $\mathbf{r}^{dq}$, $\mathbf{v}^{dq}$, $\mathbf{a}^{dq}$, $\dddot{\mathbf{r}}^{dq}$, $\ddddot{\mathbf{r}}^{dq}$
end

5.2 Test Setup

Our flight test area uses a modified AscTec Pelican quadrotor drone [65] as our main experimental platform. The quadrotor, shown in Figure 5.1, is equipped with four MS2216-11 KV 900 motors [66] each supplied by its own T Motor F20A 2-4S electronic speed controller (ESC) [67]. The quadrotor uses APC 10 × 4.7 propellers [68] and typically carries a single three-cell lithium polymer (LiPo) battery with a 2250mAh rating. We updated the Pelican drone to run on a mRo PixRacer [69] flight computer that is compatible with the PX4 flight stack. In order to obtain position feedback for indoor flights we outfit the drone with VICON markers as shown in Figure 5.2. VICON cameras in the flight test area shown in Figure 5.2 capture the position of each marker on the quadrotor

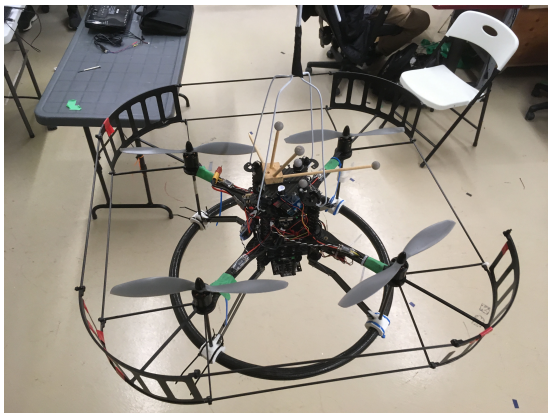

FIGURE 5.1: Modified AscTec Pelican used for Flight Testing

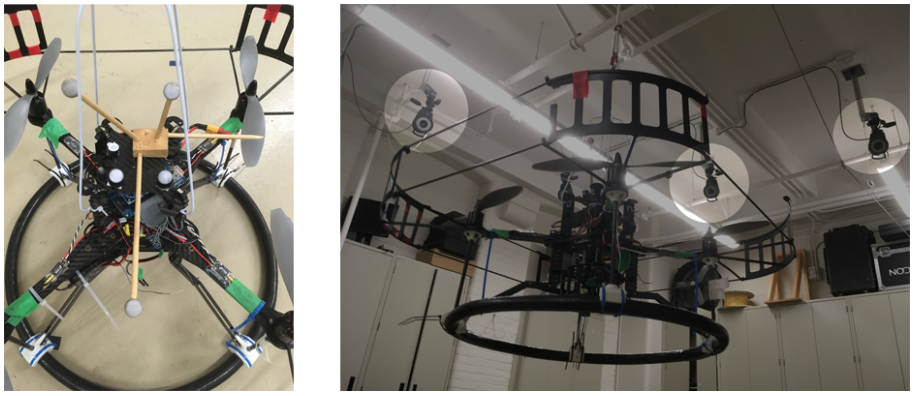

FIGURE 5.2: VICON Motion Capture Flight Test Setup

and transmit the data to a desktop computer. The computer matches the markers to a configuration that has been calibrated for our quadrotor to solve for the vehicle's position and attitude. This information is then sent via wifi to an Odroid XU 4 mounted on the Pelican drone. The Odroid passes the data to the PixRacer flight computer where it is fed into the system's state estimation algorithm. The drone is equipped with a metal harness so that it can be tethered during indoor flights for added security.

5.3 Payload Detection

A critical hurdle in conducting flight tests of our proposed flight controller is to measure the position of the payload relative to the quadrotor drone. For indoor flight testing, a common approach is to use a motion capture environment such as a VICON system to measure the position of the payload. This approach provides an unrealistic level of measurement accuracy and would not properly simulate a typical slung load delivery

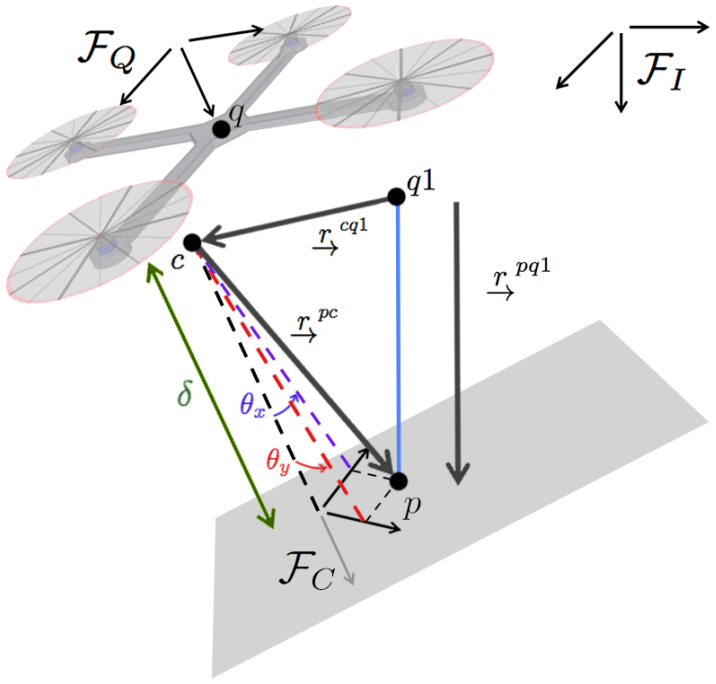

FIGURE 5.3: Payload Tracking with Camera Nomenclature

application. A more realistic option is to use a downward facing camera mounted on the drone to detect the motion of the payload as done in [37]. Another alternative is presented in [50] where a load cell is mounted on the cable and the resulting force measurements are fused with measurements from the drone's accelerometer to estimate the angle of the slung load. For this work we opted to use a Pixy Camera [70] mounted on our quadrotor to detect the position of the slung load. The Pixy Camera is purpose built for basic object detection and tracking in robotics applications. Furthermore, there is already existing code within the PX4 flight stack for using this camera to detect infrared beacons for autonomous landing operations [71]. For our project we removed a filter from the camera to enable it to detect a payload marked with coloured tape and modified the existing code in the PX4 flight stack for payload tracking.

Our Oscillation Controller in Section 3.3.1 requires measurements of the position $\mathbf{r}^p$ and velocity $\mathbf{v}^p$ of the payload resolved in the inertial frame $\mathcal{F}_I$. Our camera system will make measurements in its own coordinate frame $\mathcal{F}_C$ though and these will need to be converted to $\mathcal{F}_I$ first. Figure 5.3 provides an overview of the nomenclature used throughout this section. The Pixy Camera directly outputs the values for $(\tan\theta_x)$ and $(\tan\theta_y)$ in Figure 5.3 corresponding to the x and y position of the payload measured in frame $\mathcal{F}_C$. The camera can provide an estimate of the object depth δ based on the size of the marker but we felt that this would not be reliable given that the marker's size will appear to change as

its orientation shifts during a swinging motion. We thus developed an alternate method for determining δ based on the known cable length in our system. To simplify this analysis we will assume that the payload is treated as a point mass p. We will also assume that the vector between the payload mounting point $q1$ and the camera lens c resolved in the quadrotor's frame, $\mathbf{r}_Q^{cq1}$, is known. This value depends entirely on the geometry of the quadrotor and would be hard coded into the flight computer. Furthermore, we assume that the transformation between $\mathcal{F}_C$ and $\mathcal{F}_Q$ only involves a rotation about the z axis by a known angle ϕ_c. This is reasonable assumption here since we will be mounting our camera facing downwards along the positive z axis of the quadrotor but we may want to rotate the camera's frame relative to the quadrotor's frame to optimize the available field of view. We first parameterize the position of the payload as seen in the camera frame using depth δ and angles θ_x and θ_y shown in Figure 5.3. The position of the payload relative to the camera lens resolved in the camera frame, $\mathbf{r}_C^{pc}$, thus takes the form $\mathbf{r}_C^{pc} = [\delta \tan\theta_x \quad \delta \tan\theta_y \quad \delta]^\intercal$. Within this setup we have:

$$\underrightarrow{r}^{pq1} = \underrightarrow{r}^{pc} + \underrightarrow{r}^{cq1} \tag{5.1}$$

$$\mathbf{r}_Q^{pq1} = \mathbf{r}_Q^{cq1} + \mathbf{R}_{QC}\mathbf{r}_C^{pc} = \mathbf{r}_Q^{cq1} + \mathbf{R}_{QC}\begin{bmatrix} \delta\tan\theta_x \\ \delta\tan\theta_y \\ \delta \end{bmatrix} \tag{5.2}$$

$$\mathbf{R}_{QC} = \begin{bmatrix} \cos\phi_c & -\sin\phi_c & 0 \\ \sin\phi_c & \cos\phi_c & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

Defining $\mathbf{r}_Q^{cq1} = [\Delta_x \quad \Delta_y \quad \Delta_z]^\intercal$, where all Δ values are known, we find that

$$\mathbf{r}_Q^{pq1} = \begin{bmatrix} \Delta_x + (\cos\phi_c\tan\theta_x - \sin\phi_c\tan\theta_y)\delta \\ \Delta_y + (\sin\phi_c\tan\theta_x + \cos\phi_c\tan\theta_y)\delta \\ \Delta_z + \delta \end{bmatrix} \tag{5.4}$$

For a taut cable though, we expect that $\left\| \underrightarrow{r}^{pq1} \right\|_2 = l$ where l is computed using equation (3.20). Defining $a = (\cos\phi_c\tan\theta_x - \sin\phi_c\tan\theta_y)$, $b = (\sin\phi_c\tan\theta_x + \cos\phi_c\tan\theta_y)$, we find that

$$l^2 = (\Delta_x + a\delta)^2 + (\Delta_y + b\delta)^2 + (\Delta_z + \delta)^2 \tag{5.5}$$

$$0 = (1 + a^2 + b^2)\delta^2 + (2a\Delta_x + 2b\Delta_y + 2\Delta_z)\delta + (\Delta_x^2 + \Delta_y^2 + \Delta_z^2 - l^2) \tag{5.6}$$

$$\delta = \frac{-(2a\Delta_x + 2b\Delta_y + 2\Delta_z) \pm \sqrt{(2a\Delta_x + 2b\Delta_y + 2\Delta_z)^2 - 4(1 + a^2 + b^2)(\Delta_x^2 + \Delta_y^2 + \Delta_z^2 - l^2)}}{2(1 + a^2 + b^2)} \tag{5.7}$$

Equation (5.7) will always produce a single positive solution for δ. Having solved for δ, we can now fully solve for $\mathbf{r}_Q^{pq1}$ in equation (5.4). We then obtain the required $\mathbf{r}^p$ value based on having access to measurements of the quadrotor's position $\mathbf{r}^q$ and orientation $\mathbf{R}_{IQ}$ as follows

$$\mathbf{r}^p = \mathbf{r}^q + \mathbf{R}_{IQ}(\mathbf{r}_Q^{q1} + \mathbf{r}_Q^{pq1}) \tag{5.8}$$

where from Figure 3.1, $\mathbf{r}_Q^{q1}$ is the position of the cable mounting point on the quadrotor relative to its center of mass resolved in the quadrotor's frame.

In order to estimate the velocity of the payload $\mathbf{v}^p$ as measured in the inertial frame, we perform a finite difference computation based on the current and previously obtained measurement for the payload's position resolved in the camera frame.

$$\mathbf{v}^p = \mathbf{v}^q + \mathbf{R}_{IQ}\dot{\mathbf{r}}_Q^{pq1\,filtered} + \mathbf{R}_{IQ}\Big(\boldsymbol{\omega}^{Q\times}(\mathbf{r}_Q^{pq1} + \mathbf{r}_Q^{q1})\Big) \tag{5.9}$$

$$\dot{\mathbf{r}}_Q^{pq1} = \mathbf{R}_{QC}\Big(\frac{\mathbf{r}_{C\,current}^{pc} - \mathbf{r}_{C\,previous}^{pc}}{\Delta_t}\Big) \tag{5.10}$$

where Δ_t is the time elapsed since the previous camera measurement was made. This method relies on the fact that the quadrotor already has an estimate of its own velocity $\mathbf{v}^q$ and angular velocity $\boldsymbol{\omega}^Q$. In our implementation we obtain $\dot{\mathbf{r}}_Q^{pq1\,filtered}$ by applying a discrete first order lowpass filter to the computed values of $\dot{\mathbf{r}}_Q^{pq1}$. Our filter is designed with a time constant of 0.125s. This process helps to offset the noise induced the finite difference computation in equation (5.10). Rudimentary testing of the proposed method was conducted but we were unable to compare the accuracy of the computed payload position and velocity against a baseline such as VICON measurements. A more advanced approach to estimate the state of the payload would be to develop a state estimator that can fuse data from the drone's sensors with camera information. In [37] for instance, the authors develop an Extended Kalman Filter to perform payload state estimation for a quadrotor-slung load system equipped with a downward facing camera.

5.4 Drone Characterization

This section discusses some of the characterization work that was done in order to prepare the Pelican quadrotor drone for flight testing. The primary focus of this section is to bridge the gap between the signals from the onboard controller and the resulting thrust

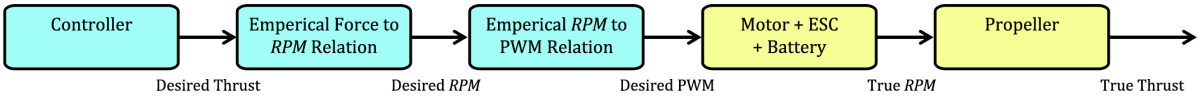

FIGURE 5.4: Thrust Application Process

applied to the drone. While performing initial flight testing with the Pelican drone, we found that there was a mismatch between the force values being requested by the flight controller and the actual thrust being applied by the quadrotor's motors. This meant that during autonomous hovering for example the steady state force requested by the controller did not match the actual weight of the drone. These observations motivated a closer study of the process by which the commanded thrust force for a given motor gets converted to an actual applied thrust force on the drone. A high level overview of this process is presented in Figure 5.4. The blue blocks in Figure 5.4 are software elements while the yellow blocks denote hardware elements. On the software side, we need to convert the desired thrust force prescribed by our controller for a given motor into a pulse width modulation (PWM) signal to send to the electronic speed controller (ESC). We model this conversion process using empirical relations. Under ideal circumstances, these empirical relations will perfectly mirror the behaviour of the hardware elements resulting in a true thrust that matches our desired thrust. Section 5.4.1 discusses the process of determining the empirical relation mapping a desired thrust to a desired motor RPM. Subsequently in Section 5.4.2 we will develop an empirical relation linking the desired RPM and the desired PWM signal.

5.4.1 Propeller Characterization

Typically we assume that for a propeller mounted with its axis of rotation pointing along the z direction that the resulting thrust force F_z and reaction moment M_z are

$$F_z = k_t RPM^2 \tag{5.11}$$

$$M_z = \pm k_q RPM^2 \tag{5.12}$$

The k_t and k_q constants depend solely on the propeller geometry and are independent of the motor and all other electric hardware being used. In order to obtain values for k_t and k_q we removed one of the APC 10×4.7 propellers from the quadrotor and mounted it on a test stand with a motor and load cell. The test stand is set up to cycle through a range of PWM signals while recording the resulting RPM values as well as the generated

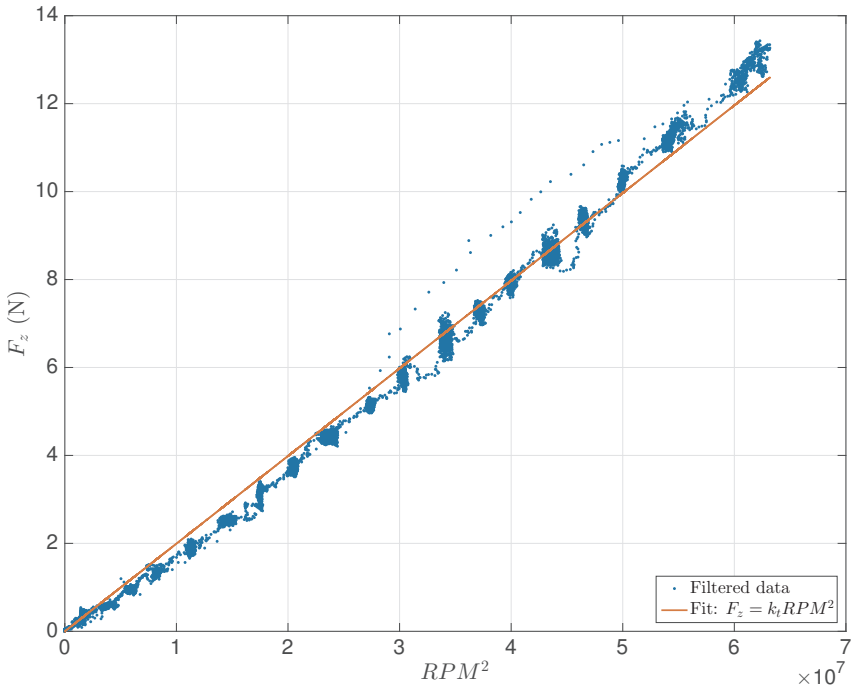

FIGURE 5.5: Thrust Versus RPM^2 for APC 10×4.7 Propeller

forces and moments. The resulting RPM, force and moment data was filtered using a Savitzky-Golay filter in MATLAB with polynomial order one. The frame size used for the filter was 21 for the the RPM and force data and 101 for the moment data. We passed the RPM data through the same filter two times due to the high amount of noise. We then plotted the resulting filtered data against the square of the RPM data and solved for the zero intercept line of best fit. These results are shown in Figures 5.5 and 5.6. From Figures 5.5 and 5.6 we obtain the values $k_t = 1.993 \times 10^{-7}\text{N}RPM^{-2}$ and $k_q = 3.3 \times 10^{-9}\text{N m}RPM^{-2}$ used in Table 3.1.

5.4.2 Motor Characterization

The relationship between the PWM signal and the resulting motor RPM is highly dependent on the motor and propeller being used. The test stand used in the previous section does not use the same motor or ESC as our quadrotor drone so we needed to develop a new testing setup for determining the relationship. Our experimental setup is shown in Figure 5.7. In Figure 5.7, the quadrotor is tied to a table in the lab so that it cannot move when the propellers spin. We then sent the same PWM signal to each motor on the drone with a modified flight controller. We put a piece of tape on each propeller and used a strobe light to measure the RPM of each propeller by manually matching the strobe

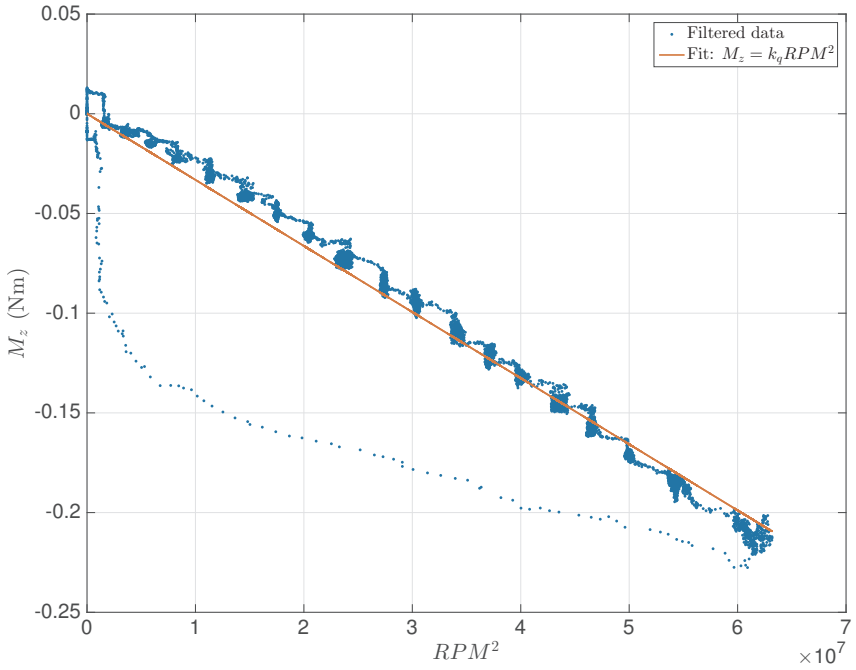

FIGURE 5.6: Moment Versus RPM^2 for APC 10×4.7 Propeller

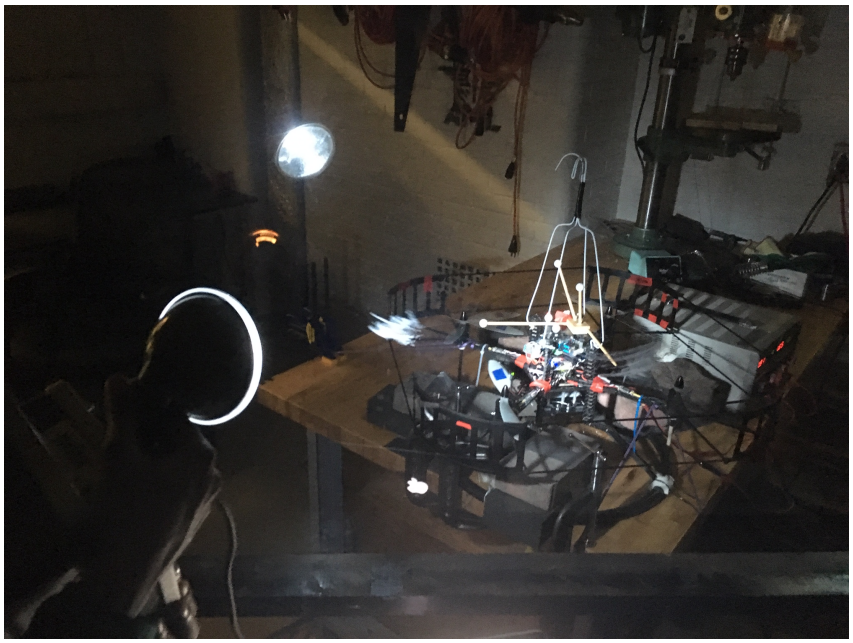

FIGURE 5.7: PWM to RPM Experimental Setup

frequency with the propeller RPM such that we had a stationary image. Care needs to be taken during this stage as false positive readings can be achieved if the strobe frequency is half that of the propeller's true RPM. For each PWM signal tested, we recorded the RPM of each propeller on the drone and then averaged these values. We found that two of the four motors typically had very similar RPM values but that the remaining two were consistently higher. Typically, the difference between the lowest and highest RPM values recorded was on the order of $400RPM$. Based on the k_t value of our propeller though, this corresponds to a relatively minor thrust difference on the order of 0.03N. During preliminary tests we found that the process of manually identifying the RPM of each motor tended to be time consuming and could easily deplete a fully charged battery after a one or two data points. This raised concerns about how the drop in battery voltage would affect our experimental results. To get around this issue, we modified our setup so that the drone was powered using an external power supply set to $12.2V$. Under normal operating conditions, the drone would have a three-cell LiPo battery with a fully charged voltage of 12.5V. The power supply was limited to a maximum current draw of 5A which restricted our ability to test the full range of PWM values for our system. In order to more fully test the available range of PWM values we conducted a limited number of tests using LiPo batteries for higher PWM values. The batteries were charged up to 12.5V and replaced when their voltage dropped below 11.8V. Another limitation we encountered during the battery trials was that our strobe light's measurement resolution decreases as we get to higher frequencies. This made it more challenging to match the motor spinning frequency with that of the strobe light. The results from this experiment are presented in Figure 5.8. In Figure 5.8 we applied a quadratic best fit line to our data and obtained an R^2 value of 0.985. We verified higher order polynomial lines of best fit but felt that the quadratic line was reasonable for the experimental behavior observed. The experimental relationship obtained was

$$\text{PWM} = (1.661 \times 10^{-6})RPM^2 + (1.49847 \times 10^{-1})RPM + 931.336 \tag{5.13}$$

The lowest RPM recorded from any propeller at minimum throttle input was $840RPM$ when a PWM signal of 1107 was used. We will thus use this value for Ω_{min} in Table 3.1. The maximum PWM signal that our controller can supply is 2000, substituting this value into equation (5.13) yields an expected maximum RPM value of $7800RPM$. We thus used this value for Ω_{max} in Table 3.1. In [64], a similar characterization is performed for a quadrotor drone in order to obtain a mapping between PWM signals and motor RPM. It is important to note that equation (5.13) is only valid for a supply voltage of 12.2V. As discussed in [64], the relationship between PWM and RPM can change significantly as battery voltage changes.

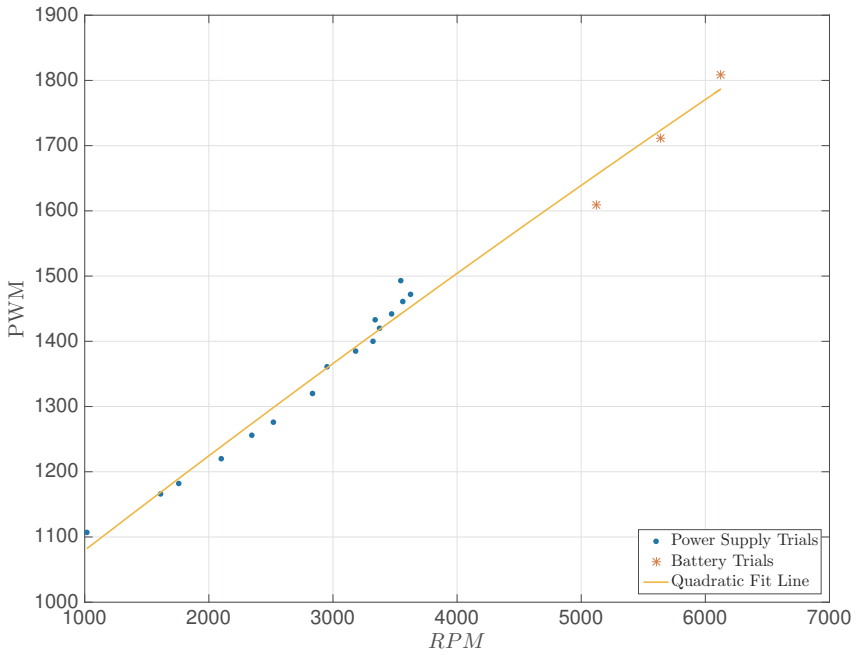

FIGURE 5.8: PWM to RPM Experimental Setup

For implementing our controller in the PX4 flight stack we use equation (5.11) to convert the desired motor force into a desired RPM value for the motor. We then use equation (5.13) to compute the required PWM signal to output to the ESC. We found that using this approach produced better results than a previous model that we had been using to do this step and improved the performance of the drone's controller. Ideally, a more advanced model could be used to account for the effect of battery voltage drop during flight. It would also be advisable to use a different RPM measurement setup as the strobe light is not well suited to taking large numbers of measurements.

5.5 Real-Time Implementation and Preliminary Flight Testing

We encountered significant challenges throughout the implementation phase of this project which limited our ability to conduct experimental test flights to validate the results from Section 3.4.3. The process of integrating the flight controller into PX4 required several iterations due to the complexity of the flight stack and limited resources for effecting significant changes to the flight controller. A key challenge during this process was managing the limited computational resources available on the PixRacer flight computer.

The PixRacer only has 256KB of RAM [69] and must simultaneously perform computations for state estimation, trajectory generation and control as well as running any other background processes. After several months, we were eventually able to fully implement Controller C and our trajectory generator in the flight stack. We then generated sample computation results in MATLAB and compared these against our PX4 version to validate that no errors were made during this integration process.

We then ran our controller in a hardware in the loop (HITL) testing environment to perform some preliminary validation. The core concept behind HITL testing is to replicate the actual conditions of a flight from the Pixracer's perspective. We load our modified PX4 flight stack onto the Pixracer and connect it to a laptop running a simulation environment populated with a single simulated quadrotor. Sensor information is obtained from this simulation and fed to the Pixracer in the same way it would be during an actual flight. The Pixracer then performs all the required computations for state estimation, trajectory generation and control in real time at 100 Hz and ultimately outputs a set of motor commands. These commands are then fed to the simulated quadrotor creating a closed loop. From the Pixracer's perspective, there is no difference between HITL testing and a real flight. We can thus accurately test whether the computational resources of our flight computer are sufficient to run our proposed flight controller and trajectory generator.

We started off by performing HITL testing of our flight stack with a simulated quadrotor without any slung load. This testing allowed us to confirm that our controller, swing prediction computation and the main **while** loop in Algorithm 4 could easily run at a prescribed frequency of 100Hz on a PixRacer flight computer. Furthermore, we found that running Algorithm 1 for all three axes took on the order of 1.5ms. These results validate that our controller and trajectory generator are viable to run on a real quadrotor drone with limited computational resources. These HITL tests also provided a safe testing environment for debugging and tuning during the process of implementing our controller and trajectory generator on the PX4 flight stack.

We next tried to modify the simulation environment to incorporate a quadrotor with a slung payload. While we were able to implement the required simulation model, we were unable to provide state feedback for the payload to the flight computer. We were thus unable to compute $\mathbf{f}^{oscil}$ in equation (3.21) of our controller. Another challenge that we faced during this process was making sure that the properties of the simulated quadrotor drone matched those of our Pelican quadrotor. In particular, we had difficulty establishing how to properly set the k_t and k_q values for simulated quadrotor's propellers.

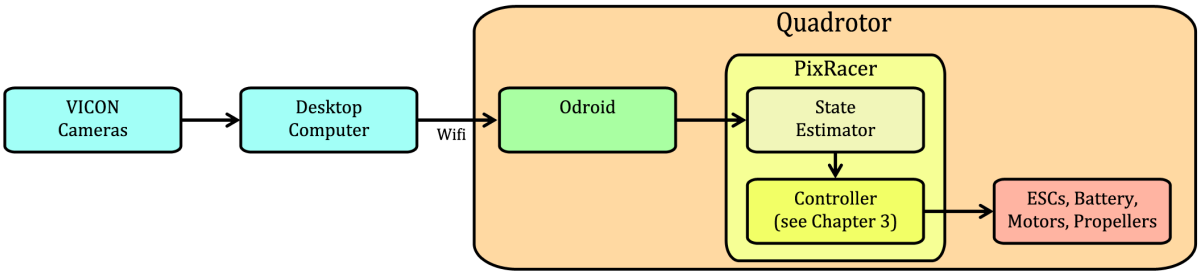

FIGURE 5.9: VICON Communication Setup

As a result, we found that controller gains tuned during HITL testing did not work well for our actual drone during later flight tests.

After performing HITL testing, we next tested our controller using an actual quadrotor drone. For this testing, we chose to fly indoors using our lab's motion capture environment. Logistically, outdoor flights are more challenging to set up limiting the number of trials we could perform. This would have been particularly problematic since a large number of flight tests were needed in order to refine our controller's tuning. Indoor flight testing also provides an added level of security in that we can tether the drone and thus safely cut power to the motors if the system becomes unstable. This was particularly important given that we completely removed the existing controller within the PX4 flight stack, including some of its safety features. Flying indoors also provides a more controlled testing environment where we can better manage external disturbances such as the wind on our system. Setting up our indoor flight tests presented several challenges though. For indoor flights, our drone cannot locate itself using GPS and must instead rely on data from our VICON motion capture setup. The communication pipeline required for this process is shown in Figure 5.9. Establishing this link between the desktop computer running the VICON software to the onboard Odroid to the PixRacer flight computer took several iterations to debug.

Once established, we encountered issues with the onboard state estimator when it tried to fuse the VICON data with accelerometer data. As soon as the quadrotor drone began to move more aggressively within the VICON field, the state estimator would throw away the position measurements coming from the VICON system and stop estimating the local x and y position of the drone. We ultimately were able to resolve this issue by switching the onboard state estimator on our drone to a legacy version that is included with the PX4 flight stack but is no longer supported by the developers. Overall this was a lengthy debugging process due to the number of elements communicating with each other and the complexity of the state estimation algorithms within the PX4 flight stack.

After resolving these issues we were able to conduct some preliminary flight tests with the quadrotor drone. We started off with a tuning that produced favourable results in HITL testing but quickly found that this did not produce satisfactory results during actual flights. The drone had a tendency to drift laterally during autonomous hovering and achieved poor trajectory tracking. We initially theorized that ground effects may be adversely affecting our flight performance. To test this theory we had a skilled pilot fly our drone in our test environment. The pilot was able to achieve significantly more stable hovering than our autonomous system leading us to conclude that the drone's tuning was the problem. Tuning our controller on the actual quadrotor drone was significantly more challenging than in previous simulations. Testing a given set of gains required lengthy setup and post processing steps. We did develop procedures to streamline this process where possible. Ultimately though, due to time limits we were unable to gather meaningful flight test data for our proposed controller. Section 6.1 will suggest future steps for flight testing our proposed controller.

Chapter 6

Conclusion

In this thesis we have explored how to prevent and actively suppress swinging motion in the payload of an autonomous quadrotor-slung load system. For swing prevention, we have exploited input shaping to generate flight trajectories that inherently avoid inducing residual swinging motion in the payload. Specifically, we have demonstrated how to extend the concept of input shaping to generate non-rest to rest motion trajectories. Our algorithm for generating these non-rest to rest input-shaped trajectories is a closed form solution that is easy to implement onboard a quadrotor drone. We have paired this algorithm with fifth order polynomial motion trajectories in order to create a computationally simple flight trajectory generator. We have also demonstrated in simulation how our proposed algorithm can effectively prevent swinging motion for a basic gantry system. After implementing our algorithm on an actual quadrotor drone's flight computer, we validated that our algorithm could quickly compute new motion trajectories as well as supply our controller with desired states at a rate of 100Hz.

In order to track our prescribed input-shaped trajectories we developed a novel controller for a quadrotor-slung load system. We surveyed the literature to identify trends in controller design for this system and motivated the need for a new controller specifically tailored to track input-shaped motion trajectories. A key novelty that our controller introduces is the use of a 2nd order prediction model to compute a desired swinging motion trajectory based on the desired flight trajectory of the quadrotor. Our controller in turn seeks to correct for differences between the actual swinging motion of the slung load and this prediction model's value. We demonstrated in simulation how our controller effectively ignores swinging motion induced by accurate trajectory tracking while still reacting to swinging motion induced by external disturbances. We also showed how, compared to two simpler baseline designs, our controller was better able to simultaneously suppress

swinging disturbances while accurately tracking a prescribed input-shaped flight trajectory. We implemented this controller on an actual drone's flight computer and validated that it could run at 100Hz. We also provided a framework for using a downward-facing camera mounted on our quadrotor to detect the state of our payload for use within our controller. Unfortunately, we were unable to conduct flight tests to validate our proposed controller due to a series of setbacks discussed in Chapter 5. Finally, we provided a preliminary look at how the core concept from our proposed controller could be implemented into a more complicated model predictive controller design for a quadrotor-slung load system.

6.1 Future Work

As discussed in Section 4.3, additional development work is needed to study the proposed model predictive controller. Firstly, we recommend setting up a test stand with the same motor, propeller and ESC as our Pelican Quadrotor in order to assess the maximum rate of change of the generated thrust force. This test stand would also allow us to repeat the motor characterization work done in Section 5.4.2. Specifically we could cover a wider range of RPM and PWM values as well as investigating the effect of battery voltage on our emperical relation presented in equation (5.13). Once a suitable thrust rate limit has been experimentally determined, we can revisit the performance of the model predictive controller developed in Chapter 4. It would also be worth exploring a wider range of values for the number of steps in the control horizon and prediction horizon to see if better performance can be achieved. An integral control element could also be added in to eliminate steady state altitude errors in our model predictive controller. Moving forward, Controller D from Chapter 4 would also need to be implemented on an actual flight computer to validate its feasibility.

A main limitation of the work presented in this thesis is that we were unable to obtain experimental data to validate the simulation results obtained for Controller C in Section 3.4.3. As discussed in Chapter 5, we encountered numerous difficulties during the implementation phase for this system which limited our ability to perform proper flight tests. We were able to fully implement Controller C and our trajectory generator on a flight computer and validate that they run properly. The main remaining challenge is to properly tune the controller gains for an actual quadrotor system and to finish integrating the downward facing camera system. The tuning methodology developed in Section 3.4.1 should likely be used as a template for proceeding with this work. A potentially useful way to tune the controller would be to have a skilled pilot fly the drone in manual mode

and look at the resulting magnitude of control inputs required to correct for disturbances. These could help to establish the order of magnitude required for certain control gains. We recommend first focusing on tuning the position and attitude control gains to establish hover in a quadrotor without any slung load. The gains could then be refined based on tracking basic motion trajectories before introducing the downward facing camera and slung load controller elements. The proposed payload detection method in Section 5.3 may also need to be revisited as part of this process.

In parallel, it would be useful to develop a better understanding of the hardware in the loop (HITL) simulation environment for development of novel controllers. As part of this project we investigated how to modify this environment so that the simulated drone matched the properties of our actual vehicle but we ultimately found that control gains tuned from HITL did not achieve good results on an actual quadrotor drone. We believe that these difficulties are due to a lack of proper understanding of how the HITL flight environment works and how experimentally measured motor and propeller properties appear within this setup. Unfortunately, there is a lack of documentation for how these processes work within the PX4 HITL setup and this will likely require additional development work moving forward. Similarly, for further development of quadrotor-slung load controllers it would be beneficial to create a proper HITL simulation setup. Currently it is possible to attach a suspended load to a simulated quadrotor carrying a downward facing camera in the HITL environment. The main limitation though is having this simulated camera capture the motion of the slung load and feed this information to the controller in simulation. Bridging this gap would allow for easier troubleshooting and controller tuning for quadrotor-slung load systems and reduce the number of flight tests required.

Moving forward, it would also be beneficial to extend this project to collaborative systems where multiple quadrotors are attached to the same payload. This problem presents numerous technical challenges in terms of coordinating movement and passing information from one system to another. At the same time, this system offers the potential to transport larger, heavier payloads and could offer new avenues for commercial use of quadrotors for payload transportation.

Appendix A

Differential Flatness of a Quadrotor

As discussed in [39], a system in the form $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ is differentially flat if there are flat outputs $\mathbf{y}$ such that all sates $\mathbf{x}$ and inputs $\mathbf{u}$ can be written as functions of $\mathbf{y}$ and its higher derivatives. Numerous proofs have been summarized in the literature for the differential flatness of a quadrotor [33], as well as quadrotor-slung load system where the cable is modelled as a single [37] or multiple links [39]. Additional differential flatness proofs for collaborative load transportation with quadrotor drones are given in [39]. Typically, the derivation of the differential flatness for these increasingly complex models are all built upon the original proof for a simple quadrotor system. In this section we provide a detailed walkthrough of the relations summarized in [33]. To this end, we show that a basic quadrotor model is differentially flat with respect to the position of the drone $\mathbf{r}^q$ and its yaw ψ^Q. This means that we will obtain relations showing $\boldsymbol{\omega}^Q = g_1(\mathbf{r}^q, \dot{\mathbf{r}}^q, ..., \psi^Q, \dot{\psi}^Q ...)$ and $\dot{\boldsymbol{\omega}}^Q = g_2(\mathbf{r}^q, \dot{\mathbf{r}}^q, ..., \psi^Q, \dot{\psi}^Q ...)$. Since we will ultimately be applying these equations to computing feedforward terms for our controller based on the desired position $\mathbf{r}^{dq}$ and yaw ψ^{Qd} of the drone we will substitute $\mathbf{r}^q = \mathbf{r}^{dq}$, $\psi^Q = \psi^{Qd}$ and solve for the desired angular velocity $\boldsymbol{\omega}^{Qdf}$ and angular acceleration $\dot{\boldsymbol{\omega}}^{Qdf}$ in equation (3.42).

For a quadrotor drone without any slung payload we have

$$m^q(\ddot{\mathbf{r}}^{dq} - \mathbf{g}) = \mathbf{R}_{IQ}^{df} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} U^{df} \quad , \quad \mathbf{g} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{A.1}$$

where $\mathbf{R}_{IQ}^{df}$ is the attitude of the quadrotor that is required to achieve the desired quadrotor acceleration $\ddot{\mathbf{r}}^{dq}$. U^{df} is a negative scalar representing the thrust required to achieve $\ddot{\mathbf{r}}^{dq}$. Note that U^{df} is a negative value because we employ the NED convention in this section. For this same reason $g = 9.8\mathrm{m\,s^{-2}}$. To solve for $\mathbf{R}_{IQ}^{df}$ we perform the following

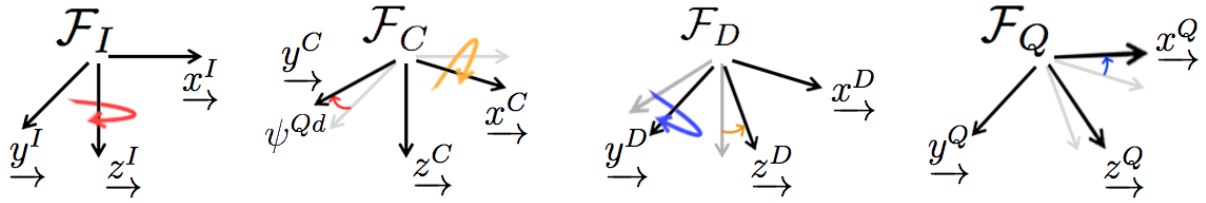

FIGURE A.1: Decomposition of Transformation from Inertial Frame to the Quadrotor Frame

decomposition.

$$\mathbf{R}_{IQ}^{df} := [\mathbf{x}_I^Q \quad \mathbf{y}_I^Q \quad \mathbf{z}_I^Q] \tag{A.2}$$

where $\mathbf{x}_I^Q$ is a column matrix representing the x axis of frame $\mathcal{F}_Q$ resolved in frame $\mathcal{F}_I$. The same convention holds for $\mathbf{y}_I^Q$ and $\mathbf{z}_I^Q$. Combining equations (A.1) and (A.2) yields the following results.

$$\ddot{\underrightarrow{r}}^{dq} - \underrightarrow{g} = \frac{U^{df}}{m^q}\underrightarrow{z}^Q \tag{A.3}$$

$$\ddot{\mathbf{r}}^{dq} - \mathbf{g} = \frac{U^{df}}{m^q}\mathbf{z}_I^Q = \left(\frac{-U^{df}}{m^q}\right)(-\mathbf{z}_I^Q) \tag{A.4}$$

$$U^{df} = -m^q\sqrt{(\ddot{\mathbf{r}}^{dq} - \mathbf{g})^\intercal(\ddot{\mathbf{r}}^{dq} - \mathbf{g})} \tag{A.5}$$

$$\mathbf{z}_I^Q = \frac{-(\ddot{\mathbf{r}}^{dq} - \mathbf{g})}{\sqrt{(\ddot{\mathbf{r}}^{dq} - \mathbf{g})^\intercal(\ddot{\mathbf{r}}^{dq} - \mathbf{g})}} \tag{A.6}$$

To solve for $\mathbf{x}_I^Q$ and $\mathbf{y}_I^Q$ we decompose the transformation from $\mathcal{F}_I$ to $\mathcal{F}_Q$ into three steps as shown in Figure A.1. We first define an intermediate frame $\mathcal{F}_C$ created by rotating the inertial frame $\mathcal{F}_I$ about the z axis by the desired quadrotor yaw angle ψ^{Qd}. We then rotate frame $\mathcal{F}_C$ about its x axis to define a new frame $\mathcal{F}_D$. We finally rotate $\mathcal{F}_D$ about its y axis to obtain $\mathcal{F}_Q$. This yields the following relationships between the unit magnitude coordinate axis vectors for the various frames.

$$\underrightarrow{z}^I = \underrightarrow{z}^C \quad , \quad \underrightarrow{x}^C = \underrightarrow{x}^D \quad , \quad \underrightarrow{y}^D = \underrightarrow{y}^Q \tag{A.7}$$

$$\underrightarrow{x}^D \perp \underrightarrow{y}^D \implies \underrightarrow{x}^C \perp \underrightarrow{y}^Q \tag{A.8}$$

$$\underrightarrow{z}^Q \perp \underrightarrow{y}^Q \tag{A.9}$$

$$\underrightarrow{y}^Q = \underrightarrow{z}^Q \times \underrightarrow{x}^C \quad , \quad \mathbf{y}_I^Q = \mathbf{z}_I^{Q\times}\mathbf{x}_I^C \tag{A.10}$$

$$\mathbf{x}_I^C = \begin{bmatrix} \cos\psi^{Qd} \\ \sin\psi^{Qd} \\ 0 \end{bmatrix} \tag{A.11}$$

$$\mathbf{x}_I^Q = \mathbf{y}_I^{Q\times}\mathbf{z}_I^Q \tag{A.12}$$

This parameterized approach requires that $\underrightarrow{z}^Q$ and $\underrightarrow{x}^C$ not be parallel. From a practical standpoint this means that the drone cannot have a pitch of 90 degrees. This is a reasonable assumption for a quadrotor drone doing parcel delivery. Taking the time derivative of equation (A.4) with respect to the inertial frame yields the following results.

$$m^q \dddot{\underrightarrow{r}}^{dq} = \dot{U}^{df}\underrightarrow{z}^Q + U^{df}\underrightarrow{\omega}^{Qdf} \times \underrightarrow{z}^Q \tag{A.13}$$

$$m^q \dddot{\mathbf{r}}^{dq} = \dot{U}^{df}\mathbf{z}_I^Q + U^{df}\boldsymbol{\omega}_I^{Qdf\times}\mathbf{z}_I^Q \tag{A.14}$$

If we left multiply equation (A.14) by $\mathbf{z}_I^{Q\mathsf{T}}$ we get the following result.

$$\dot{U}^{df} = m^q \mathbf{z}_I^{Q\mathsf{T}} \dddot{\mathbf{r}}^{dq} = m^q \dddot{\mathbf{r}}^{dq\mathsf{T}} \mathbf{z}_I^Q \tag{A.15}$$

Combining together equations (A.15) and (A.14) yields

$$\boldsymbol{\omega}_I^{Qdf\times}\mathbf{z}_I^Q = \frac{m^q}{U^{df}}\left(\dddot{\mathbf{r}}^{dq} - (\dddot{\mathbf{r}}^{dq\mathsf{T}}\mathbf{z}_I^Q)\mathbf{z}_I^Q\right) \tag{A.16}$$

Note that while the preceding equation contains $\boldsymbol{\omega}_I^{Qdf}$, which is vector $\underrightarrow{\omega}^{Qdf}$ resolved in the inertial frame, we ultimately need to resolve $\underrightarrow{\omega}^{Qdf}$ in frame $\mathcal{F}_Q$ to obtain the desired $\boldsymbol{\omega}^{Qdf}$. To do so we define vector $\underrightarrow{h}^1 := \underrightarrow{\omega}^{Qdf} \times \underrightarrow{z}^Q$. We can then resolve this cross product in frame $\mathcal{F}_Q$ or $\mathcal{F}_I$ to obtain

$$\mathbf{h}_Q^1 = \boldsymbol{\omega}^{Qdf\times}\mathbf{z}_Q^Q \quad , \quad \mathbf{h}_I^1 = \boldsymbol{\omega}_I^{Qdf\times}\mathbf{z}_I^Q \tag{A.17}$$

Defining $\boldsymbol{\omega}^{Qdf} := [p \quad q \quad r]^\mathsf{T}$ and using $\mathbf{z}_Q^Q := [0 \quad 0 \quad 1]^\mathsf{T}$ we find

$$\mathbf{h}_Q^1 = \begin{bmatrix} q \\ -p \\ 0 \end{bmatrix} \tag{A.18}$$

$$\mathbf{R}_{QI}^{df} = \mathbf{R}_{IQ}^{df\ \mathsf{T}} \tag{A.19}$$

$$\mathbf{h}_Q^1 = \mathbf{R}_{QI}^{df}\mathbf{h}_I^1 = \mathbf{R}_{QI}^{df}(\boldsymbol{\omega}_I^{Qdf\times}\mathbf{z}_I^Q) \tag{A.20}$$

Combining equations (A.20), (A.18) and (A.16) yields a solution for components q and p of $\boldsymbol{\omega}^{Qdf}$.

$$\begin{bmatrix} q \\ -p \\ 0 \end{bmatrix} = \mathbf{R}_{QI}^{df} \left(\frac{m^q}{U^{df}} \left(\dddot{\mathbf{r}}^{dq} - (\dddot{\mathbf{r}}^{dq\top} \mathbf{z}_I^Q) \mathbf{z}_I^Q \right) \right) \tag{A.21}$$

where $\mathbf{z}_I^Q$ is solved using equation (A.6) and $\dddot{\mathbf{r}}^{dq}$ comes from our trajectory generator. The remaining r component of $\boldsymbol{\omega}^{Qdf}$ can be found by again considering the relationship between the angular velocities of reference frames $\mathcal{F}_I$, $\mathcal{F}_C$ and $\mathcal{F}_Q$.

$$\underrightarrow{\omega}^{Qdf} = \underrightarrow{\omega}^{QC} + \underrightarrow{\omega}^{CI} \tag{A.22}$$

where $\underrightarrow{\omega}^{CI}$ is the angular velocity of $\mathcal{F}_C$ relative to $\mathcal{F}_I$ and $\underrightarrow{\omega}^{QC}$ is the angular velocity of $\mathcal{F}_Q$ relative to $\mathcal{F}_C$. Given that the transformation from $\mathcal{F}_C$ relative to $\mathcal{F}_I$ is simply a rotation about the z axis, we can resolve $\underrightarrow{\omega}^{CI}$ in $\mathcal{F}_C$ or $\mathcal{F}_I$ to obtain

$$\boldsymbol{\omega}_I^{CI} = \boldsymbol{\omega}_C^{CI} = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi}^{Qd} \end{bmatrix} \quad , \quad \underrightarrow{\omega}^{CI} = \dot{\psi}^{Qd} \underrightarrow{z}^I \tag{A.23}$$

where $\dot{\psi}^{Qd}$ is the desired yaw rate of the drone. Furthermore, we expect that the angular velocity $\underrightarrow{\omega}^{QC}$ will have no z component when resolved in frame $\mathcal{F}_Q$ [33]. We can thus perform a dot product on all terms of equation (A.22) with $\underrightarrow{z}^Q$, the z vector of frame $\mathcal{F}_Q$.

$$\underrightarrow{\omega}^{Qdf} \cdot \underrightarrow{z}^Q = \underrightarrow{\omega}^{QC} \cdot \underrightarrow{z}^Q + \underrightarrow{\omega}^{CI} \cdot \underrightarrow{z}^Q \tag{A.24}$$

$$= \underrightarrow{\omega}^{CI} \cdot \underrightarrow{z}^Q \tag{A.25}$$

$$= \dot{\psi}^{Qd} \underrightarrow{z}^I \cdot \underrightarrow{z}^Q \tag{A.26}$$

In order to evaluate the dot product in equation (A.26), we resolve the computation in frame $\mathcal{F}_Q$.

$$\boldsymbol{\omega}^{Qdf\top} \mathbf{z}_Q^Q = \dot{\psi}^{Qd} \mathbf{z}_Q^{I\top} \mathbf{z}_Q^Q \tag{A.27}$$

$$\begin{bmatrix} p & q & r \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \dot{\psi}^{Qd} \mathbf{z}_Q^{I\top} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{A.28}$$

$$r = \dot{\psi}^{Qd} \mathbf{z}_Q^{I\top} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{A.29}$$

$$\mathbf{z}_Q^I = \mathbf{R}_{QI}^{df}\mathbf{z}_I^I = \mathbf{R}_{QI}^{df}\begin{bmatrix}0\\0\\1\end{bmatrix} \tag{A.30}$$

At this point we have fully solved for the components of $\boldsymbol{\omega}^{Qdf}$. In order to solve for $\dot{\boldsymbol{\omega}}^{Qdf}$ we take the time derivative of equation (A.14) with respect to the inertial frame $\mathcal{F}_I$.

$$m^q \underrightarrow{\ddot{r}}^{dq} = \ddot{U}^{df}\underrightarrow{z}^Q + U^{df}\underrightarrow{\dot{\omega}}^{Qdf} \times \underrightarrow{z}^Q + 2\dot{U}^{df}\underrightarrow{\omega}^{Qdf} \times \underrightarrow{z}^Q + U^{df}\underrightarrow{\omega}^{Qdf} \times \left(\underrightarrow{\omega}^{Qdf} \times \underrightarrow{z}^Q\right) \tag{A.31}$$

$$m^q \ddot{\mathbf{r}}^{dq} = \ddot{U}^{df}\mathbf{z}_I^Q + U^{df}\dot{\boldsymbol{\omega}}_I^{Qdf\times}\mathbf{z}_I^Q + 2\dot{U}^{df}\boldsymbol{\omega}_I^{Qdf\times}\mathbf{z}_I^Q + U^{df}\boldsymbol{\omega}_I^{Qdf\times}\left(\boldsymbol{\omega}_I^{Qdf\times}\mathbf{z}_I^Q\right) \tag{A.32}$$

If we left multiply equation (A.32) by $\mathbf{z}_I^{Q\mathsf{T}}$ we obtain an equation to solve for $\ddot{U}^{df}$.

$$\ddot{U}^{df} = m^q\mathbf{z}_I^{Q\mathsf{T}}\ddot{\mathbf{r}}^{dq} - U^{df}\mathbf{z}_I^{Q\mathsf{T}}\left(\boldsymbol{\omega}_I^{Qdf\times}\left(\boldsymbol{\omega}_I^{Qdf\times}\mathbf{z}_I^Q\right)\right) \tag{A.33}$$

We now define $\underrightarrow{h}^2 := \underrightarrow{\dot{\omega}}^{Qdf} \times \underrightarrow{z}^Q$. This cross product can be resolved in frame $\mathcal{F}_Q$ or $\mathcal{F}_I$ to obtain

$$\mathbf{h}_Q^2 = \dot{\boldsymbol{\omega}}^{Qdf\times}\mathbf{z}_Q^Q \quad , \quad \mathbf{h}_I^2 = \dot{\boldsymbol{\omega}}_I^{Qdf\times}\mathbf{z}_I^Q \tag{A.34}$$

Defining $\dot{\boldsymbol{\omega}}^{Qdf} := [\alpha_1 \quad \alpha_2 \quad \alpha_3]^\mathsf{T}$ and using $\mathbf{z}_Q^Q := [0 \quad 0 \quad 1]^\mathsf{T}$ we find

$$\mathbf{h}_Q^2 = \begin{bmatrix}\alpha_2\\-\alpha_1\\0\end{bmatrix} \tag{A.35}$$

$$\mathbf{h}_Q^2 = \mathbf{R}_{QI}^{df}\mathbf{h}_I^2 \tag{A.36}$$

Rearranging equation (A.32) we can isolate and solve for $\mathbf{h}_I^2$ and use equations (A.36) and (A.35) to solve for components α_1 and α_2 of $\dot{\boldsymbol{\omega}}^{Qdf}$. The final component α_3 can be solved by considering the relationship between angular accelerations of frames $\mathcal{F}_I$, $\mathcal{F}_C$ and $\mathcal{F}_Q$.

$$\underrightarrow{\dot{\omega}}^{Qdf} = \underrightarrow{\dot{\omega}}^{QC} + \underrightarrow{\dot{\omega}}^{CI} + \underrightarrow{\omega}^{CI} \times \underrightarrow{\omega}^{QC} \tag{A.37}$$

Exploiting the fact that $\underrightarrow{z}^Q \cdot \underrightarrow{\dot{\omega}}^{QC} = 0$ and $\underrightarrow{z}^Q \cdot (\underrightarrow{\omega}^{CI} \times \underrightarrow{\omega}^{QC}) = 0$ [33] we can dot product equation (A.37) with $\underrightarrow{z}^Q$ and resolve the result in frame $\mathcal{F}_Q$.

$$\underrightarrow{\dot{\omega}}^{Qdf} \cdot \underrightarrow{z}^Q = \underrightarrow{\dot{\omega}}^{CI} \cdot \underrightarrow{z}^Q \tag{A.38}$$

$$\underrightarrow{\dot{\omega}}^{CI} = \ddot{\psi}^{Qd} \underrightarrow{z}^I \tag{A.39}$$

$$[\alpha_1 \quad \alpha_2 \quad \alpha_3] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \alpha_3 = \ddot{\psi}^{Qd} \mathbf{z}_Q^{I\,\top} \mathbf{z}_Q^Q \tag{A.40}$$

where $\ddot{\psi}^{Qd}$ is the desired yaw angular acceleration for the quadrotor. Although not necessary for our controller, we can also solve for the applied quadrotor moment $\mathbf{m}^{df}$ using differential flatness.

$$\mathbf{J}^q \dot{\boldsymbol{\omega}}^{Qdf} + \boldsymbol{\omega}^{Qdf\times} \mathbf{J}^q \boldsymbol{\omega}^{Qdf} = \mathbf{m}^{df} \tag{A.41}$$

In summary, this section has demonstrated how, given derivatives of the desired quadrotor motion trajectory $\ddot{\mathbf{r}}^{dq}$, $\dddot{\mathbf{r}}^{dq}$ and $\ddddot{\mathbf{r}}^{dq}$ resolved in the inertial frame as well as the desired yaw ψ^{Qd} and its higher derivatives $\dot{\psi}^{Qd}$ and $\ddot{\psi}^{Qd}$ we can algebraically solve for all remaining states and inputs of the system.

Bibliography

[1] Igor Henrique Beloti Pizetta, Alexandre Santos Brandão, and Mário Sarcinelli-Filho. Control and obstacle avoidance for an uav carrying a load in forestal environments. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 62–67. IEEE, 2018.

[2] Sean Fielding and Meyer Nahon. Input shaped trajectory generation and controller design for a quadrotor-slung load system. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 153–161. IEEE, 2019.

[3] N Matheson. Stability of helicopter slung loads. 1976.

[4] David T Liu. In-flight stabilization of externally slung helicopter loads. Technical report, Northrop Corp Hawthorne CA Electronics Div, 1973.

[5] Christina Ivler. Design and flight test of a cable angle feedback control system for improving helicopter slung load operations at low speed. Technical report, Army Aviation and Missile Research Development and ENG CTR Moffett Field CA, 2014.

[6] Daniel KD Villa, Alexandre S Brandão, and Mário Sarcinelli-Filho. Load transportation using quadrotors: A survey of experimental results. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 84–93. IEEE, 2018.

[7] Igor Henrique Beloti Pizetta, Alexandre Santos Brandão, and Mário Sarcinelli-Filho. Modelling and control of a quadrotor carrying a suspended load. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 249–257. IEEE, 2015.

[8] Koushil Sreenath and Vijay Kumar. Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots. *Robotics: Science and Systems*, 2013.

[9] Farhad A Goodarzi and Taeyoung Lee. Stabilization of a rigid body payload with multiple cooperative quadrotors. *Journal of Dynamic Systems, Measurement, and Control*, 138(12):121001, 2016.

[10] Nadia S Zúñiga, Filiberto Muñoz, Marco A Márquez, Eduardo S Espinoza, and Luis R García Carrillo. Load transportation using single and multiple quadrotor aerial vehicles with swing load attenuation. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 269–278. IEEE, 2018.

[11] Sarah Tang and Vijay Kumar. Autonomous flight. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:29–52, 2018.

[12] Philipp Foehn, Davide Falanga, Naveen Kuppuswamy, Russ Tedrake, and Davide Scaramuzza. Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload. In *Robotics: Science and Systems*, pages 1–10, 2017.

[13] Sarah Tang and Vijay Kumar. Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2216–2222. IEEE, 2015.

[14] Ivana Palunko, Rafael Fierro, and Patricio Cruz. Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach. In *2012 IEEE International Conference on Robotics and Automation*, pages 2691–2697. IEEE, 2012.

[15] Aleksandra Faust, Ivana Palunko, Patricio Cruz, Rafael Fierro, and Lydia Tapia. Learning swing-free trajectories for uavs with a suspended load. In *2013 IEEE International Conference on Robotics and Automation*, pages 4902–4909. IEEE, 2013.

[16] Joshua Vaughan, Aika Yano, and William Singhose. Comparison of robust input shapers. *Journal of Sound and Vibration*, 315(4-5):797–815, 2008.

[17] Neil C Singer and Warren P Seering. Preshaping command inputs to reduce system vibration. *Journal of dynamic systems, measurement, and control*, 112(1):76–82, 1990.

[18] Joshua Vaughan, Aika Yano, and William Singhose. Robust negative input shapers for vibration suppression. *Journal of Dynamic Systems, Measurement, and Control*, 131(3):031014, 2009.

[19] Eihab M Abdel-Rahman, Ali H Nayfeh, and Ziyad N Masoud. Dynamics and control of cranes: A review. *Modal Analysis*, 9(7):863–908, 2003.

[20] Morten Bisgaard, Anders la Cour-Harbo, and Jan Bendtsen. Input shaping for helicopter slung load swing reduction. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 6964, 2008.

[21] Morten Bisgaard. *Modeling, estimation, and control of helicopter slung load system.* Department of Control Engineering, Aalborg University, 2008.

[22] James Potter, William Singhose, and Mark Costelloy. Reducing swing of model helicopter sling load using input shaping. In *Control and Automation (ICCA), 2011 9th IEEE International Conference on*, pages 348–353. IEEE, 2011.

[23] Christopher Adams, James Potter, and William Singhose. Input-shaping and model-following control of a helicopter carrying a suspended load. *Journal of Guidance, Control, and Dynamics*, 38(1):94–105, 2014.

[24] Recep Cetin. Indoor navigation system and suspended load control for multirotors. Master's thesis, NTNU, 2015.

[25] Kristian Klausen, Thor I Fossen, and Tor Arne Johansen. Nonlinear control with swing damping of a multirotor uav with suspended load. *Journal of Intelligent & Robotic Systems*, 88(2-4):379–394, 2017.

[26] Morten Bisgaard, Anders la Cour-Harbo, and Jan Dimon Bendtsen. Adaptive control system for autonomous helicopter slung load operations. *Control Engineering Practice*, 18(7):800–811, 2010.

[27] Chang-Wan Ha and Dongwook Lee. Analysis of embedded prefilters in motion profiles. *IEEE Transactions on Industrial Electronics*, 65(2):1481–1489, 2017.

[28] William E Singhose, Lisa J Porter, Timothy D Tuttle, and Neil C Singer. Vibration reduction using multi-hump input shapers. *Journal of dynamic systems, Measurement, and control*, 119(2):320–326, 1997.

[29] Jonathan Y Smith, Kris Kozak, and William E Singhose. Input shaping for a simple nonlinear system. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 1, pages 821–826. IEEE, 2002.

[30] Kristian Klausen, Thor I Fossen, and Tor Arne Johansen. Nonlinear control of a multirotor uav with suspended load. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 176–184. IEEE, 2015.

[31] Dan Kielsholm Thomsen, Rune Søe-Knudsen, David Brandt, Ole Balling, and Xuping Zhang. Generating vibration free rest-to-rest trajectories for configuration dependent dynamic systems via 3-segmented input shaping. In *IEEE International Conference on Robotics and Automation*, 2018.

[32] Shupeng Lai, Kangli Wang, and Ben M Chen. Dynamically feasible trajectory generation method for quadrotor unmanned vehicles with state constraints. In *2017 36th chinese control conference (CCC)*, pages 6252–6257. IEEE, 2017.

[33] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2520–2525. IEEE, 2011.

[34] Sonja Macfarlane and Elizabeth A Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics and Automation*, 19(1):42–52, 2003.

[35] Mark W Mueller, Markus Hehn, and Raffaello D'Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics*, 31(6):1294–1310, 2015.

[36] Koushil Sreenath, Nathan Michael, and Vijay Kumar. Trajectory generation and control of a quadrotor with a cable-suspended load-a differentially-flat hybrid system. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4888–4895. IEEE, 2013.

[37] Sarah Tang, Valentin Wüest, and Vijay Kumar. Aggressive flight with suspended payloads using vision-based control. *IEEE Robotics and Automation Letters*, 3(2): 1152–1159, 2018.

[38] Koushil Sreenath, Taeyoung Lee, and Vijay Kumar. Geometric control and differential flatness of a quadrotor uav with a cable-suspended load. In *CDC*, pages 2269–2274. Citeseer, 2013.

[39] Prasanth Kotaru, Guofan Wu, and Koushil Sreenath. Differential-flatness and control of quadrotor (s) with a payload suspended through flexible cable (s). In *2018 Indian Control Conference (ICC)*, pages 352–357. IEEE, 2018.

[40] Ying Feng, Camille Alain Rabbath, Subhash Rakheja, and Chun-Yi Su. Adaptive controller design for generic quadrotor aircraft platform subject to slung load. In *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on*, pages 1135–1139. IEEE, 2015.

[41] Ivana Palunko, Patricio Cruz, and Rafael Fierro. Agile load transportation: Safe and efficient load manipulation with aerial robots. *IEEE robotics & automation magazine*, 19(3):69–79, 2012.

[42] Keeryun Kang, JVR Prasad, and Eric Johnson. Active control of a uav helicopter with a slung load for precision airborne cargo delivery. *Unmanned Systems*, 4(03): 213–226, 2016.

[43] Farhad A Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric control of a quadrotor uav transporting a payload connected via flexible cable. *International Journal of Control, Automation and Systems*, 13(6):1486–1498, 2015.

[44] Guilherme V Raffo and Marcelino M de Almeida. Nonlinear robust control of a quadrotor uav for load transportation with swing improvement. In *2016 American Control Conference (ACC)*, pages 3156–3162. IEEE, 2016.

[45] Byung-Yoon Lee, Hae-In Lee, Dong-Wan Yoo, Gun-Hee Moon, Dong-Yeon Lee, Yun young Kim, and Min-Jea Tahk. Study on payload stabilization method with the slung-load transportation system using a quad-rotor. In *2015 European Control Conference (ECC)*, pages 2097–2102. IEEE, 2015.

[46] Farhad Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric stabilization of a quadrotor uav with a payload connected by flexible cable. In *American Control Conference (ACC), 2014*, pages 4925–4930. IEEE, 2014.

[47] Daniel Newman, Seong-Wook Hong, and Joshua E Vaughan. The design of input shapers which eliminate nonzero initial conditions. *Journal of Dynamic Systems, Measurement, and Control*, 140(10):101005, 2018.

[48] Daniel Newman and Joshua Vaughan. Command shaping of a boom crane subject to nonzero initial conditions. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1189–1194. IEEE, 2017.

[49] AscTec Hummingbird. http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-hummingbird/, . Online; accessed 12 July 2019.

[50] Seung Jae Lee and H Jin Kim. Autonomous swing-angle estimation for stable slung-load flight of multi-rotor uavs. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4576–4581. IEEE, 2017.

[51] Luigi S Cicolani and Gerd Kanning. Equations of motion of slung-load systems, including multilift systems. 1992.

[52] Ying Feng, Camille Alain Rabbath, and Chun-Yi Su. Modeling of the dynamics of a micro uav with a single slung load. *Handbook of Unmanned Aerial Vehicles*, pages 1–19, 2018.

[53] Farhad Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric nonlinear pid control of a quadrotor uav on se (3). In *Control Conference (ECC), 2013 European*, pages 3845–3850. IEEE, 2013.

[54] Matthias Faessler, Davide Falanga, and Davide Scaramuzza. Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight. *IEEE Robotics and Automation Letters*, 2(2):476–482, 2016.

[55] Basil Kouvaritakis and Mark Cannon. *Model predictive control: Classical, robust and stochastic*. Springer International Publishing, 2016.

[56] Pengkai Ru and Kamesh Subbarao. Nonlinear model predictive control for unmanned aerial vehicles. *Aerospace*, 4(2):31, 2017.

[57] Understanding Model Predictive Control Part 3: MPC Design Parameters. https://store.mrobotics.io/mRo-PixRacer-R14-Official-p/auav-pxrcr-r14-mr.htm, . Online; accessed 30 July 2019.

[58] Stefan Notter, Alexander Heckmann, Aaron Mcfadyen, and F Gonzalez. Modelling, simulation and flight test of a model predictive controlled multirotor with heavy slung load. *IFAC-PapersOnLine*, 49(17):182–187, 2016.

[59] Markus Zürn, Kye Morton, Alexander Heckmann, Aaron McFadyen, Stefan Notter, and Felipe Gonzalez. Mpc controlled multirotor with suspended slung load: System architecture and visual load detection. In *2016 IEEE Aerospace conference*, pages 1–11. IEEE, 2016.

[60] Clark Youngdong Son, Hoseong Seo, Taewan Kim, and H Jin Kim. Model predictive control of a multi-rotor with a suspended load for avoiding obstacles. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–6. IEEE, 2018.

[61] Specify Constraints. https://www.mathworks.com/help/mpc/ug/specifying-constraints.html, . Online; accessed 10 August 2019.

[62] Optimization Problem. https://www.mathworks.com/help/mpc/ug/optimization-problem.html, . Online; accessed 10 August 2019.

[63] Patricio J Cruz and Rafael Fierro. Cable-suspended load lifting by a quadrotor uav: hybrid model, trajectory generation, and control. *Autonomous Robots*, 41(8): 1629–1643, 2017.

[64] Chen Wang, Meyer Nahon, and Mike Trentini. Controller development and validation for a small quadrotor with compensation for model variation. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 902–909. IEEE, 2014.

[65] AscTec Pelican. http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-pelican/, . Online; accessed 22 July 2019.

[66] T Motor MS2216 KV900 for Quads. https://www.hobbypartz.com/02p-motor-376-ms2216-kv900.html. Online; accessed 22 July 2019.

[67] T Motor F20A 2-4S ESC. https://www.getfpv.com/tiger-motor-f-20a-2-4s-esc-blheli-s-w-dshot.html. Online; accessed 22 July 2019.

[68] 10x4.7SF. https://www.apcprop.com/product/10x4-7sf/. Online; accessed 22 July 2019.

[69] PixRacer R14. https://store.mrobotics.io/mRo-PixRacer-R14-Official-p/auav-pxrcr-r14-mr.htm, . Online; accessed 24 July 2019.

[70] Pixy Camera. https://pixycam.com/pixy-cmucam5/, . Online; accessed 23 July 2019.

[71] Precision Landing. https://docs.px4.io/v1.9.0/en/advanced_features/precland.html. Online; accessed 23 July 2019.