MODELING AND CONTROL OF HELICOPTERS CARRYING SUSPENDED LOADS

A Thesis
Presented to
The Academic Faculty

by

Christopher James Adams

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

Georgia Institute of Technology
August 2012

MODELING AND CONTROL OF HELICOPTERS CARRYING SUSPENDED LOADS

Approved by:

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

Dr. Mark Costello
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Eric Johnson
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 12 June 2012

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. William Singhose, for his guidance and patience. His constant support and boundless enthusiasm have helped and inspired me greatly. I would also like to thank my committee members, Dr. Mark Costello and Dr. Eric Johnson, for their support of this work. Also, special thanks go to Dr. Mark Costello for providing use of the Indoor Flight Facility for conducting some of the radio-controlled helicopter experiments and for providing me with resources in my initial research of model-following control. I would also like to thank James Potter for his help, tutelage, and encouragement and Dr. Joshua Vaughan for his help and wisdom.

I would also like to thank my parents for their love and unquestioning support, without which I would not be where I am today. They have made many sacrifices to provide me with opportunities in life and to ensure that I have received a good education.

Lastly, I would like to thank God for the many blessings he has given me. *Ad majorem Dei gloriam.*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Helicopters are often used to transport supplies and equipment. When a heavy load is carried via suspension cables below a helicopter, the load oscillates in response to helicopter motion and disturbance forces, such as wind. This oscillation is dangerous and adversely affects control of the helicopter, especially when carrying large or heavy loads.

By adding a command-shaping method called input shaping to the helicopter's flight controller, the suspended load oscillation caused by helicopter motion is greatly reduced. A significant benefit of this approach is that it does not require measurement of the load position.

This thesis contains derivations and analysis of simple planar helicopter-load dynamic models, and these models are verified using experimental data from model-scale, radio-controlled helicopters. The effectiveness of input shaping at eliminating suspended load oscillation is then demonstrated on this experimental hardware. In addition, the design of an attitude command, near-hover flight controller that combines input shaping and a common flight control architecture is illustrated using dynamic models of a Sikorsky S-61 helicopter, and simulation results are shown for example lateral and longitudinal repositioning movements. Results show that applying input shaping to simulated pilot commands greatly improves performance when carrying a suspended load.

CHAPTER I

INTRODUCTION

1.1 Motivation

A helicopter can be used as a "flying crane" by hanging a load (most often called a *suspended load* or *sling load*) from cables attached to the helicopter. A flying crane is extremely versatile. It can be used to transport logs during remote logging operations, deliver power transmission towers to their installation locations, rescue people stranded in remote areas, and even deliver food and supplies to a disabled cruise ship, as shown in Figure 1. Helicopters transporting logs and power transmission towers are shown in Figures 2 and 3, respectively. These are just a few examples of tasks that are too expensive, too slow, or physically impossible to perform with other types of vehicles.

One example of such a heavy lift helicopter is the Kaman K-MAX® Aerial Truck[4]. It is available in both manned and unmanned versions. A photo of the unmanned version during a demonstration to the U.S. Marine Corps is shown in Figure 4. As an example of the utility of flying crane helicopters, two of the unmanned K-MAX helicopters have transported over one million pounds of cargo over a four-month period for the U.S. Marine Corps in Afghanistan[6].

Unfortunately, the suspended load behaves as a pendulum, making efficient and accurate transfer of the load difficult. Load swing also adversely affects control of the helicopter. In fact, accidents can be caused by violent suspended load swing [58]. The helicopter becomes particularly difficult to control when carrying heavy loads [18].

---

[1]Photo Source: Gregory Bull. "Navy helicopter drops supplies onto the Carnival Splendor off Mexico s Baja Peninsula," Associated Press, 11 November 2010. Appears in: Gene Sloan, "On disabled cruise ship, a 'nightmare'," USA Today, page FA.

[2]Photo Source: http://tealjones.com/Forestry_Heli.htm

[3]Photo Source: http://dorukhava.en.ec21.com/Slung_Cargo_Transportation–4186984.html

[4]http://www.kaman.com/aerospace/helicopters/products-services/k-max/

[5]Photo Source: http://www.armybase.us/2010/02/k-max-demonstrates-successful-unmanned-helicopter-cargo-resupply-to-u-s-marine-corps/

[6]http://www.kaman.com/news/deployment-extended-for-k-max-unmanned-aerial-cargo-hauler-as-it-exceeds-million-pound-milestone/

Figure 1: Helicopter delivering supplies to a stranded cruise ship[1].

Figure 2: Helicopter transporting logs during a remote logging operation[2].

Figure 3: Helicopter carrying a power transmission tower[3].

Figure 4: Unmanned K-MAX® helicopter transporting cargo during a demonstration to the U.S. Marine Corps[5].

Figure 5 shows the lateral load oscillation during and following a simulated near-hover lateral move performed by a helicopter carrying a heavy suspended load. The load oscillates with a large amplitude and slow period. The load suspension point is below the helicopter's center of gravity, so the tension in the suspension cable produces an oscillating torque about the helicopter's center of gravity as the load swings. This effect is known as load-attitude coupling.

A plot of the pilot's attitude command and the resulting helicopter roll attitude during and following the lateral move is shown in Figure 6. Due to load-attitude coupling, the load swing causes residual roll attitude oscillations that have an amplitude of nearly 2 degrees. Residual attitude oscillations larger than 0.5 degrees are considered excessive for any type of maneuver [59]. These attitude oscillations make the helicopter difficult to control, and the load swing slows down load transfer operations.

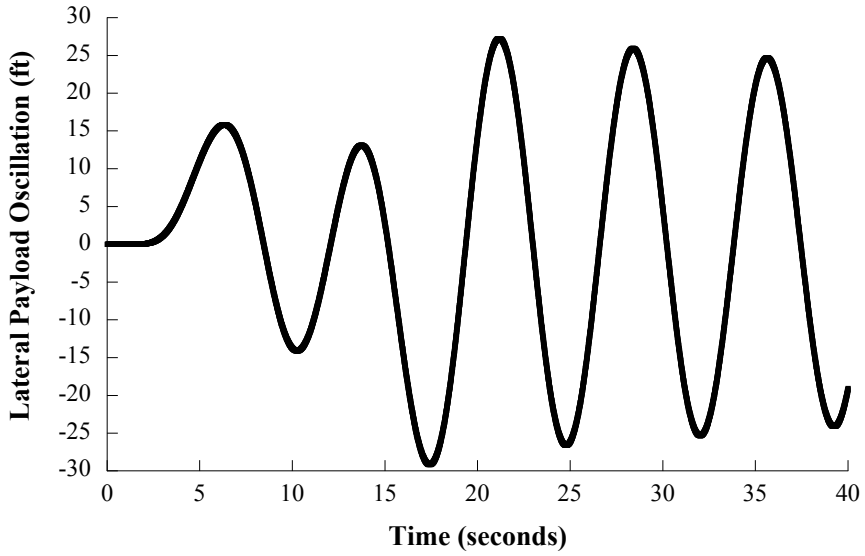


Figure 5: Lateral load oscillation caused by a lateral helicopter move.

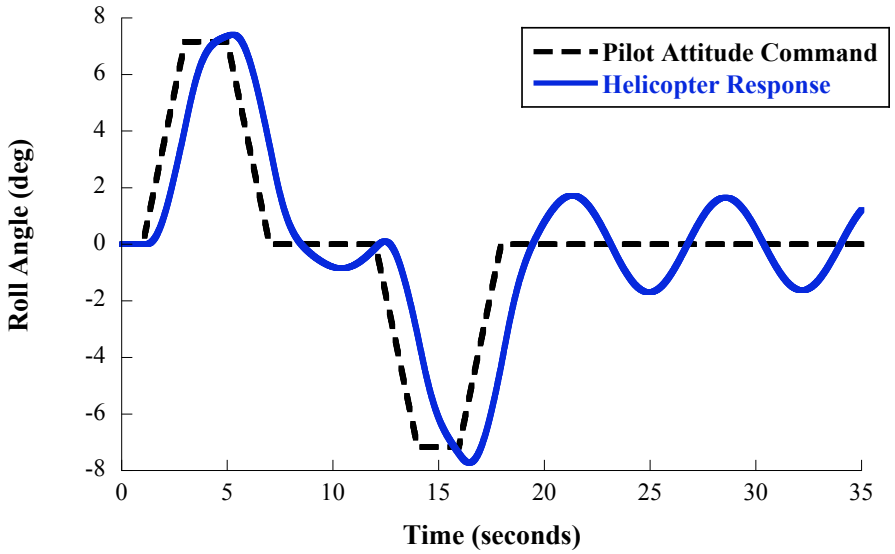


Figure 6: Helicopter roll attitude response to a pilot's attitude command when carrying a heavy suspended load.

Guidelines for suspended load operations suggest that the best way for a pilot to regain control when load swing becomes too large is to slow down the helicopter [8, 58]. By trying to actively cancel load swing, the pilot may actually amplify the problem if his or her control inputs are not in the correct phase relative to the swing [8]. In cases where the swing amplitude becomes extreme and the pilot has difficulty stabilizing the load, the pilot

may elect to jettison the load to regain control of the helicopter [58]. The safety of sling load operations would be improved significantly by reducing load swing and its effects on helicopters.

Reducing load swing would also increase productivity during load transfer operations. Once a load is positioned above its desired location, it cannot be safely deposited until the swing amplitude settles below an acceptable level. Keeping the swing at a low amplitude would allow the pilot to transfer and deposit loads more quickly and safely.

1.2 Solution Approach

1.2.1 Input Shaping

This thesis investigates a method to improve performance during suspended load operation by adding a technique called *input shaping* to the helicopter's flight controller. Input shaping strategically modifies a reference command by convolving it with a series of impulses, called an *input shaper*. The resulting command induces little or no residual vibration [39, 50]. Designing input shapers only requires estimates of the natural frequency and damping ratio of the undesired vibratory mode.

By applying input shaping control to helicopters carrying suspended loads, residual load oscillations caused by helicopter motion can be significantly reduced. A significant advantage of this approach is that it does not require real-time measurement or estimation of the load states.

Input shaping has proven effective on many kinds of machines, including cranes [7, 51], robotic arms [10, 14, 29], coordinate measuring machines [24, 48], and satellites [2, 45, 56, 65]. Input shapers can be designed to suppress multiple flexible modes [22, 40, 42, 46]. In addition, many studies of crane operators have shown that input shaping can greatly improve performance [25, 26, 30]. The primary disadvantages of input shaping are that it cannot reduce vibration caused by external disturbances and it introduces a small response lag due to the method used to form the shaped commands.

The application of input shaping to helicopters carrying suspended loads has been considered by a few previous researchers [6, 35, 37]. Bisgaard et al. [6] used a state estimator

to measure the natural frequency of the suspended load oscillation. They then used this measurement to adaptively change an input shaper to suppress oscillation at the measured natural frequency. Ottander and Johnson [35] combined input shaping with delayed feedback control of the payload swing angle. This controller allowed the the payload to more accurately track a desired trajectory. Input shaping reduced load swing caused by helicopter motion, and external payload disturbances were canceled using feedback of the load swing angle. Both approaches required using a vision system to measure the load swing. A vision system that yields accurate measurements of the load swing would be costly and potentially difficult to implement in most real-world situations.

Input shapers are designed to suppress motion-induced residual oscillation, so they will not decrease oscillation induced by external disturbance forces, such as wind gusts. Suppressing that swing does require feedback of the load states or an estimation technique that can accurately predict the load swing caused by a measured wind gust. Although it can be effective to combine input shaping with feedback control [35], this thesis focuses on eliminating load swing caused by helicopter motion through use of input shaping control.

1.2.2 Integrating Input Shaping with Helicopter Flight Controllers

Effective design of a helicopter's flight control system is critical to the overall performance of the aircraft. Stability augmentation systems and high authority control augmentation systems assist the pilot in maneuvering the aircraft by stabilizing its response, adding damping to oscillatory aircraft modes, and/or introducing different flight characteristics such as altitude or heading holds [23, 27, 38]. When carrying a heavy load, the pilot's control task is challenging despite the assistance of these control systems [8, 18, 58].

To be effective at suppressing suspended load oscillations, the input-shaping technique must be integrated with the rest of the helicopter's flight control system. Done properly, this would combine the features of helicopter flight control systems listed above with the suspended load oscillation suppression characteristics of input shaping. The most logical location for the input shaper in the controller is immediately after the pilot inputs have been received by the digital flight control system. The integration of input shaping into a

common flight control architecture will be investigated in Chapter 4.

1.2.3 Helicopter Modeling

Helicopter dynamic models assist in the design of flight control systems. Models that incorporate the suspended load dynamics can be used to design input shapers by providing estimates of the natural frequency and damping ratio of the load oscillation. These estimates should be good enough for the purposes of designing input shapers and would not rely on a vision system to obtain measurements of the suspended load natural frequency. This thesis will investigate using such models to design input shapers.

Potter et al. [37] experimentally verified a simple model of the suspended load oscillation, and used the model to test via simulation the effectiveness and robustness of various types of input shapers. However, as was explained above, suspended load oscillation adversely affects control of the helicopter due to coupling between the load swing and the helicopter motion. The model they investigated did not take into account this coupling effect. The models investigated in this thesis will incorporate load back-driving of the helicopter and load-attitude coupling effects.

More sophisticated models that include more of the helicopter dynamics can be used to design and test helicopter flight control systems for suspended load operations. This thesis will use such a model to study the combination input shaping with a flight control system for use during suspended load operations.

1.3 Helicopter Handling Qualities

The characteristics of an aircraft and its flight control system affect the aircraft's aptitude at performing flight tasks. It is also important to consider the pilot workload required to accomplish these tasks. Military and civil ratings requirements governing task performance and pilot workload for various missions, maneuvers, visual cues, external environments, and a range of influencing factors have been developed [59]. Under these requirements, the flight characteristics of an aircraft along with its control system and accompanying visual displays are rated based on how effective they make the aircraft and pilot at performing tasks and maneuvers in various environments. These ratings are known as the handling–or

flying–qualities of the aircraft. Padfield states that handling qualities are considered good when "safe flight is guaranteed throughout the operational flight envelope" [36]. Cooper and Harper defined handling qualities as "those qualities or characteristics of an aircraft that govern the ease and precision with which a pilot is able to perform the tasks required in support of an aircraft role" [9].

1.3.1 Cooper-Harper Rating Scale

Cooper and Harper's handling qualities rating scale, shown in Figure 7, is based on pilot evaluation of an aircraft's characteristics for a variety of maneuvers and tasks [9]. For a given maneuver, flight test pilots use the decision tree shown in Figure 7 to assign a given aircraft configuration a rating from 1 to 10. A rating of 1 indicates a controllable aircraft with satisfactory performance and 10 indicates an uncontrollable aircraft. This rating is then used to assign a handling qualities level from 1 to 3. Most aircraft are required to be Level 1, but Level 2 is acceptable in failed and emergency situations; Level 3 is never acceptable [36]. According to Padfield [36], the handling qualities rating scale developed by Cooper and Harper is "the most developed and widely recognized quality scale" for helicopters.

1.3.2 Aeronautical Design Standard 33E

In addition to Cooper and Harper's handling qualities ratings scale, performance specifications such as the U.S. Army's Aeronautical Design Standard 33E (ADS-33E) [59] quantify the requirements necessary to achieve each handling-quality level for many mission-related tasks. Such numerical requirements enable helicopter design and control engineers to use flight test data and simulation to assign a handling-quality level to aircraft characteristics for various tasks. Aircraft are designed to achieve Level 1 handling qualities for all mission tasks and visual environments to ensure that they will be safe to fly throughout their operational flight envelope.

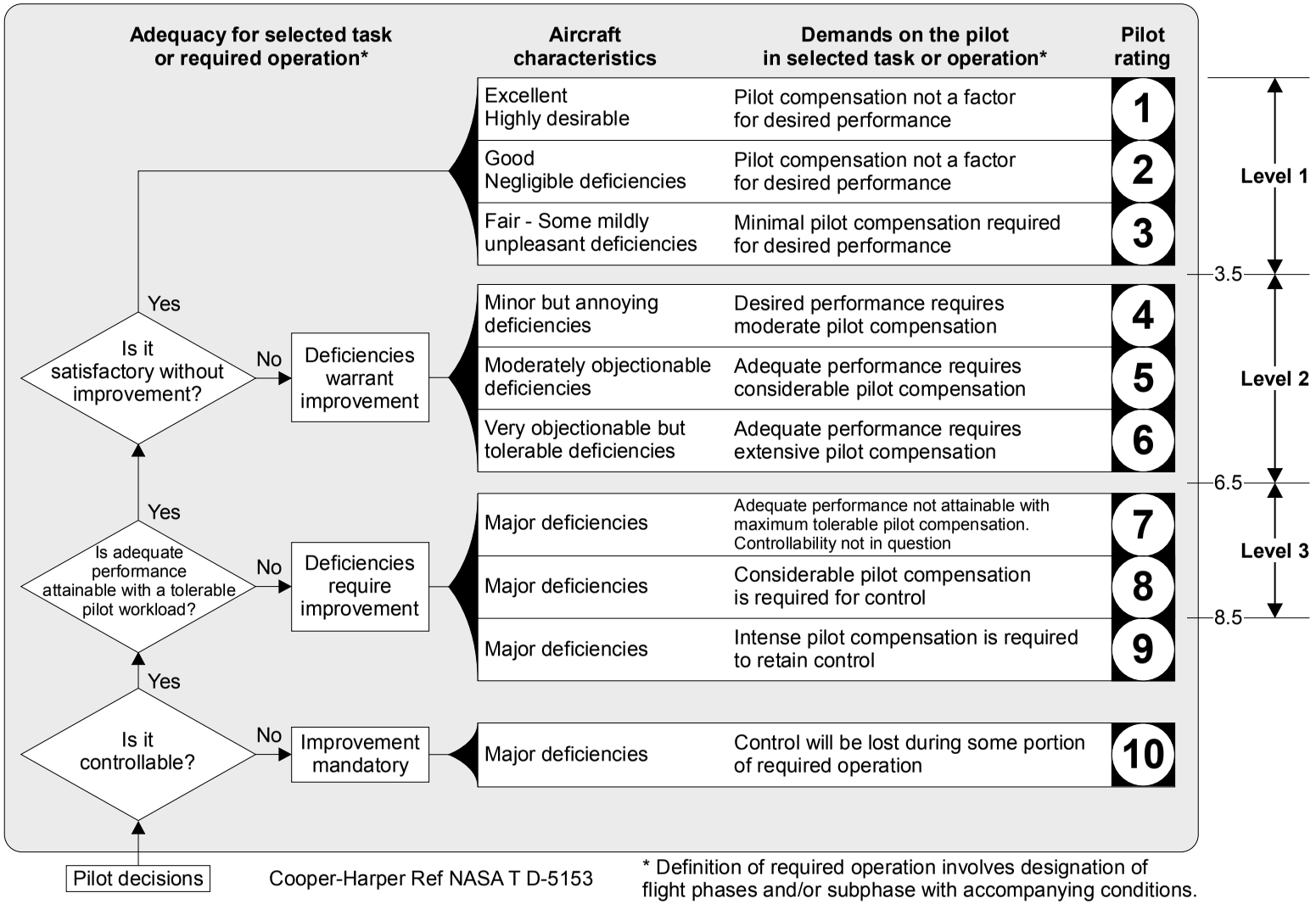


Figure 7: Cooper-Harper handling qualities rating scale [9].

1.3.3 Handling Qualities for Flight with Suspended Loads

Hoh et al. [18] studied the influence of the suspended load on pilots' perceptions of the aircraft handling qualities and the overall controllability of the aircraft for a variety of helicopter and suspended load configurations. Their conclusions indicate that the translational response of the aircraft when carrying a load has a much greater impact on the perceived handling qualities than the attitude response, particularly for small- and medium- amplitude maneuvers. This is somewhat contrary to the guidelines provided by ADS-33E, where more emphasis is placed on the attitude response. Their results also show that the handling qualities degrade significantly when the load mass approaches or is greater than half of the helicopter mass.

Other agencies regulate suspended load operations and ensure proper safety procedures are in place during suspended load flight. One such group in the United Kingdom is the Civil Aviation Authority Safety Regulation Group [58]. They publish the CAP 426 standard [58] that provides rules and guidelines that address many factors associated with suspended load operations. Some of the rules and guidelines include the types of loads that can be carried, how to properly connect the load to the helicopter, how to ensure the safety of personnel on the ground, take-off and set-down procedures, and how to prevent accidents caused by load oscillation.

1.4 Helicopter Flight Control Systems

Control of helicopters is primarily accomplished by producing moments and forces on the helicopter via one or more rotors that change the helicopter velocity and orientation, or attitude, relative to a trim or equilibrium state [23]. The forces and moments are applied by changing the pitch of the rotor blades as they rotate around the rotor hub and/or by changing the engine power. The changes in pitch as a function of the angle of a rotor blade around the rotor changes the angle of attack of the rotor blades. This produces aerodynamic forces that vary over the area of the rotor disk and effectively cause the rotor to tilt relative to the helicopter. These aerodynamic forces resolve into a thrust vector that changes with the rotor tilt, producing a moment about the helicopter center of gravity via the rotor hub

and shaft [23].

A helicopter is not stable without some type of stability or control augmentation. Therefore, without augmentation the pilot must perform the duties of a feedback controller to stabilize the aircraft. Automatic flight control systems are designed to modify the helicopter's control characteristics and performance and to reduce the pilot's workload [23].

Helicopters and their flight controllers must be designed to satisfy handling qualities specifications, such as ADS-33E [59], imposed by regulatory agencies [23]. Typically, control design for hover and forward flight require separate analysis because the helicopter has different responses and performance requirements in these two flight regimes [23].

Longitudinal and lateral stability analysis and control design are usually performed separately by uncoupling the helicopter equations of motion. This approach relies on assuming that the longitudinal and lateral dynamics of the helicopter are separable [23]. Also, only the steady-state response of the rotor is included in the model [23], and the rotor is used as a source of control forces and moments applied to the helicopter in response to control inputs or changes in the flight condition [38]. Johnson [23] states that the basic characteristics of the helicopter flight dynamics are well-represented by a model with these two approximations.

Control and stability can also be augmented through the use of control gyros and an entirely mechanical control system, such as stabilizer bars for two-bladed teetering rotors and gyro stabilizers for three- and four-bladed hingeless rotors [23].

In modern flight control systems, digital fly-by-wire makes higher-authority control possible. One type of modern flight control system is model-following control. Model-following control is used in helicopter flight control systems to force specified helicopter states, such as the attitude, to respond like a prescribed ideal model [52]. Model-following control has become an attractive control technique for helicopter flight control systems. For example, Boeing Helicopters used a control law architecture consisting of model-following control on several demonstrator programs in the 1980's and 1990's, including the V-22 and RAH-66 [27].

1.5 Helicopter Control When Carrying a Suspended Load

Several strategies have been proposed for improving control of helicopters carrying suspended loads. Strategies include active and passive cable suspension point systems that minimize the effect of the load on the helicopter or reduce the load swing itself. Other approaches incorporate feedback of the load states to modify the pilot's reference commands in such a way that minimizes load swing. These approaches rely on measurement and estimation of the load states.

One focus is on the control of helicopters carrying suspended loads in near-hover operations [15, 28]. The near-hover case is often studied because, according to Szustak and Jenney [53], "the ultimate hovering task for a [flying] crane pilot is to acquire and maintain a stable hover over a specific point on the ground while a slung load is attached or dropped."

The Kaman K-MAX® Aerial Truck is equipped with a cable suspension point mechanism that passively reduces load-attitude coupling effects. Figure 8 shows a photo of the underside of the K-MAX® helicopter where the suspension point mechanism can be seen. The suspension cable hook is mounted on a trolley that slides along a curved rail. The curved rail is oriented along the lateral axis of the vehicle.

Instead of allowing the load swing in the lateral direction to be directly transmitted to the helicopter, this sliding suspension point absorbs some of the load back-driving in the motion of the trolley along the rail. The rail also helps keep the suspension point centered vertically below the helicopter center of gravity during rolling maneuvers.

The suspension cable hook itself can pivot relative to the trolley. The pivot is aligned parallel to the rail to absorb some load motion in the longitudinal direction. Longitudinal load-attitude coupling is not as much of a problem as lateral coupling because the longitudinal moment of inertia of a helicopter is typically much larger than the lateral moment of inertia. The larger moment of inertia provides better resistance to load-attitude coupling effects, so a longitudinal sliding rail system is not necessary.

Control strategies for damping the suspended load swing include an actuated suspension

[7] Photo Source: http://www.kaman.com/aerospace/helicopters/products-services/k-max/

Figure 8: Underside of the K-MAX® helicopter showing the suspension point sliding-rail mechanism[7].

point or some other form of active load stabilization [49]. In fact, Dukes [11, 12] states that an actively controlled, moving suspension point is an effective way of damping the load oscillation. However, retrofitting existing heavy-lift helicopters with moving suspension points would be very costly compared to small modifications to the digital fight control system.

Some proposed control strategies rely on feedback of the suspended load states. Such control algorithms require real-time measurements of the load states and may be difficult to implement in practice due to the difficulty and cost of obtaining accurate and reliable measurements. For example, one approach for suspended load state measurement relies on using a vision system to measure the load swing [6, 35].

Instead of using direct measurement of the load states, other control strategies rely on estimation techniques to predict the load swing [4, 5, 15, 31]. To accurately predict the load

states, these estimation processes require complicated models accounting for aerodynamic effects and estimation of external disturbances caused by wind [4]. Such models tend to be computationally expensive due to their complexity and therefore may not be feasible for estimating the load state information in real-time. Another load state estimation technique requires accurate measurements of the helicopter position relative to a reference point on the ground [15]. Many of these control strategies may not be robust to slight changes in parameters such as suspension cable length or payload geometry.

1.6 Thesis Outline

Chapter II discusses dynamic modeling of a helicopter carrying a suspended load. The first model is a simple planar crane model. The model is experimentally verified on a radio-controlled helicopter carrying a suspended load. The next model attempts to correct some shortcomings of the planar crane model by including the helicopter attitude, allowing load-attitude coupling effects in the model. The last two models investigated are models of a Sikorsky S-61 helicopter with and without a suspended load. These models will be used in Chapter IV to design and test the combination of input shaping and a common helicopter flight controller.

Chapter III investigates the application of input shaping to helicopters carrying suspended loads. The simple dynamic models from Chapter II are used to obtain estimates of the natural frequency and damping ratio of the suspended load oscillation for a radio-controlled helicopter carrying a suspended load. It is shown that an input shaper designed using the estimated natural frequency and damping ratio is effective at reducing the suspended load oscillation. Also, an input shaping MATLAB program written for and delivered to two helicopter companies is briefly presented. These companies will use the program to test the effectiveness of input shaping at reducing suspended load operations.

Chapter IV studies the integration of input shaping with a common helicopter flight control system. The flight control architecture investigated is known as model-following control. A model-following controller is designed and implemented in simulation for a Sikorsky S-61 helicopter with and without a suspended load. The control of the unloaded

helicopter is analyzed as a baseline case to compare to the performance of the controller for the loaded helicopter. Lastly, input shaping is added to the controller for the loaded helicopter. With input shaping is added to the controller, the helicopter better tracks the prescribed model and the suspended load oscillation is greatly reduced. This chapter shows that input shaping can be effectively integrated with a helicopter flight control system.

Lastly, Chapter V summarizes the contributions of this thesis and provides suggestions for future work.

CHAPTER II

DYNAMIC MODELING OF HELICOPTERS CARRYING SUSPENDED LOADS[1]

In this chapter, the dynamic modeling of a helicopter carrying a suspended load will be addressed. One goal is to develop a model that yields estimates of the natural frequency and damping ratio of the suspended load swing for a range of helicopter and load configurations. A primary goal of this modeling effort is to produce natural frequency and damping ratio estimates that are sufficiently accurate for the purposes of designing input shapers. Another goal is to identify models that incorporate most of the dynamic effects important to helicopter-suspended load operations. These models can then be used for designing and simulation testing of flight controllers proposed for suspended load operation.

Many researchers have studied dynamic modeling of helicopters carrying suspended loads. Dukes [11, 12] studied helicopters carrying suspended loads using simple models consisting of only a few degrees of freedom. He used those models to study methods of damping the pendulum mode of the sling load for various helicopter maneuvers. Potter et al. [37] used a simple translational model motivated by the similarities between helicopters carrying suspended loads and cranes to approximate the suspended load dynamics. Lucassen and Sterk [28] showed that a helicopter carrying a suspended load near hover is mechanically similar to a double pendulum. Bisgaard et al. [3] modeled different slung load suspension types including multi-lift configurations and experimentally verified the results on a small-scale helicopter.

The first goal is addressed by considering that the dynamics of a load suspended from a helicopter are similar to those of crane payloads. In fact, both systems can be most simply modeled as pendulums. This justifies modeling the complex helicopter-load system

[1]Parts of this chapter have been accepted for publication in the 2012 12th International Conference on Control, Automation and Systems

using some of the same techniques used to model cranes, while also including some of the more complicated dynamics such as load-vehicle coupling. With the relaxation of a few assumptions used to derive models of cranes, simple models of the helicopter-load system can be derived. A similar approach used by Potter et al. [37] involved combining a simple second-order underdamped model of the payload swing with a first order model that approximates the helicopter attitude and translational dynamics.

The second goal is addressed by considering a model of a Sikorsky S-61 helicopter [15, 16]. Two versions of the model are presented. The first is an unloaded model linearized about hover obtained directly from flight test data [16]. The second is a loaded model [15], also linearized about hover, that combined the model from [16] with a model of the suspended load states. In Chapter IV, the Sikorsky S-61 models will be used to develop a near-hover flight controller that combines input shaping and a standard helicopter flight control architecture.

In the first section of this chapter, a planar model is presented where the load is allowed to back-drive the suspension point, an effect commonly seen with heavy sling loads. In the second section, load-attitude coupling is included in addition to the back drive effect. In the last section, a linearized model of a Sikorsky S-61 helicopter in hover from the literature [15, 16] is presented and analyzed.

2.1 Planar Crane Model

In this section, a simple model is proposed for designing input shapers for suppressing suspended load oscillations. A micro coaxial radio-controlled helicopter was used to experimentally verify this model. The model can be used to design input shapers for helicopter-load configurations that cannot be safely or effectively tested using the available experimental setup. By experimentally verifying a model that better captures the dynamics of a helicopter carrying a suspended load, this section extends the work performed by Potter et al. [37].

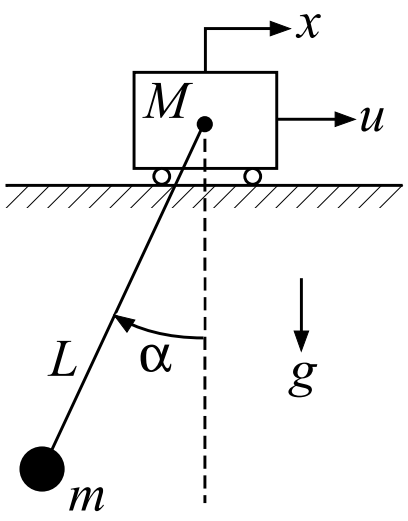


Figure 9: Schematic diagram of a helicopter-suspended load system modeled as a planar crane.

2.1.1 Model Description

A simple crane model that captures the lateral motion of a helicopter carrying a suspended load is a two-degree of freedom planar crane model. Figure 9 shows a schematic diagram of a planar crane.

Heavy payloads have a significant impact on the translational degrees of freedom of the helicopter [18]. To incorporate this effect, the payload is assumed to back-drive the helicopter position. The suspension cable is assumed to be rigid and massless and to be attached at the centers of gravity of the helicopter and load. The load is modeled as a point-mass. Due to the low speeds attainable by the model helicopter, the effects of aerodynamic drag are modeled as viscous damping terms on the helicopter position and cable angle. Rotor downwash effects are neglected. There are several other assumptions implicit in the application of a planar crane model to the lateral motion of a helicopter; it is assumed that the helicopter maintains a constant heading and altitude, and the helicopter's attitude and position in the longitudinal direction remain constant.

As derived in [13], the equations of motion of a planar crane are

$$\ddot{x} = \frac{-m\sin\alpha(L\dot{\alpha}^2 + g\cos\alpha)}{M + m\sin^2\alpha} + \frac{1}{M + m\sin^2\alpha}u - \frac{D_H}{M}\dot{x}, \tag{1}$$

$$\ddot{\alpha} = \frac{-(m+M)g\sin\alpha - mL\dot{\alpha}^2\sin\alpha\cos\alpha}{ML + mL\sin^2\alpha} + \frac{\cos\alpha}{ML + mL\sin^2\alpha}u - \frac{D_P}{m}\dot{\alpha}, \tag{2}$$

where x is the helicopter displacement, α is the deflection angle of payload cable, and D_H and D_P are viscous damping coefficients on the helicopter position and payload angle, respectively. The helicopter and payload masses are M and m, respectively, and the suspension cable length is given by L. The input to the system is a force applied to the helicopter in the horizontal direction, given by u. Note that the horizontal displacement of the payload relative to the helicopter can be calculated as

$$x_L = L\sin\alpha \tag{3}$$

This expression is useful when converting the payload deflection angle to position for comparison with experimental payload swing results.

2.1.2 Linearized Planar Crane Equations and Load Swing Natural Frequency

(1) and (2) can be linearized by applying a small angle approximation to α ($\sin\alpha \approx \alpha, \cos\alpha \approx 1$) and neglecting higher order terms such as $\sin^2\alpha$ and $\dot{\alpha}^2$. This yields the following linearized equations of motion

$$\ddot{x} = \frac{-m\alpha g}{M} + \frac{1}{M}u - \frac{D_H}{M}\dot{x} \tag{4}$$

$$\ddot{\alpha} = \frac{-(m+M)g\alpha}{ML} + \frac{1}{ML}u - \frac{D_P}{m}\dot{\alpha} \tag{5}$$

The linearized equations of motion given by (4) and (5) suggest that the planar crane has two modes: one rigid body mode corresponding to translation of the helicopter and load as a whole and an underdamped mode corresponding to oscillation of the load relative to the helicopter. By inspection of (5), the natural frequency ω_n of the load oscillation mode can be expressed as

$$\omega_n = \sqrt{\frac{g}{L}\left(\frac{m+M}{M}\right)} \tag{6}$$

The ratio of the payload mass to the total mass of the helicopter and the load is commonly referred to as the Load-Mass Ratio. In equation form, the Load-Mass Ratio R is defined as

$$R \equiv \frac{m}{M+m} \tag{7}$$

Using this definition of the Load-Mass Ratio, (6) can be rewritten as

$$\omega_n = \sqrt{\frac{g}{L}\left(\frac{1}{1-R}\right)} \tag{8}$$

(8) agrees with an approximation for the load natural frequency used by flying qualities engineers at Boeing Helicopters[2].

The natural frequency predicted by (8) is plotted for a range of the suspension cable lengths and Load-Mass Ratios in Figure 10. The configuration represented by the CX3 experimental helicopter that will be analyzed in this section is labeled by the circle. The natural frequencies predicted by (8) for three helicopters carrying a suspended load are also labeled on the plot. These helicopters include E-flite Blade CX3 and Blade 400 radio-controlled helicopters and a Sikorsky S-61 helicopter. The CX3 helicopter will be discussed in the next section and the Blade 400 is part of an experimental setup that will be discussed in Chapter III. The Sikorsky S-61 helicopter will be discussed in Section 2.3.

2.1.3 Model Verification

2.1.3.1 Experimental Hardware: Blade CX3 Radio-Controlled Helicopter and Indoor Flight Facility

Figure 11 shows an experimental setup in the Indoor Flight Facility at the Georgia Institute of Technology. The helicopter shown is a E-Flite Blade CX3 micro coaxial radio-controlled (RC) helicopter. The mass of the RC helicopter and its payload are 0.534 lb and 0.033 lb, respectively. The length of the load suspension cable is 3.38 ft. The load was designed to approximate a point mass and have a low aerodynamic profile to minimize the unmodeled effects of aerodynamic drag and rotor downwash. A Vicon MX motion capture system is used to measure the position and orientation of the helicopter in real-time.

[2]Source: Email correspondence with Pamela Montanye of Boeing Helicopters on 4 November 2011.

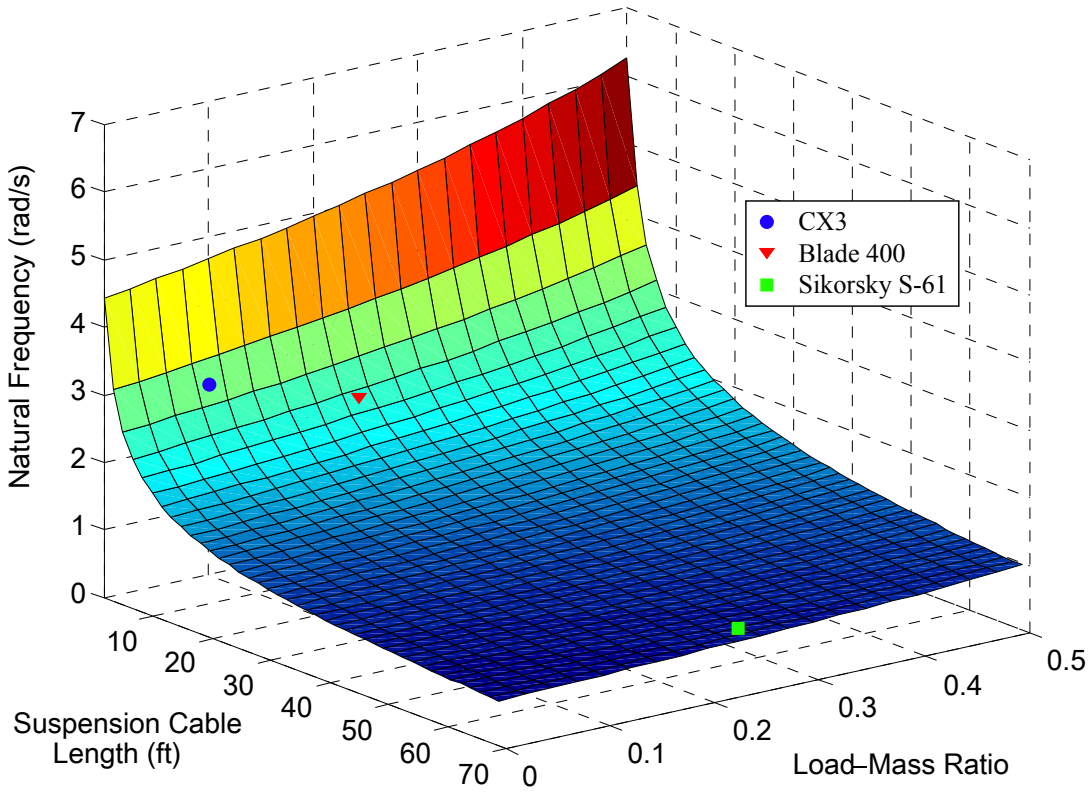


Figure 10: Plot of suspended load linearized natural frequency for a range of helicopter-load configurations.

The data signal flow in the motion capture system is shown in Figure 12. The system consists of 12 MX-3+ cameras connected via 2 Vicon MX Ultranet HD units that stream camera data to the computer at 120 Hz. Vicon iQ version 2.5 software running on the computer processes the camera data. The resulting position and orientation measurements are exported to MATLAB using the Vicon Tarsus Realtime data streaming application. Each MX-3+ camera can record 659x493 grayscale pixels, and position measurements made using this system have a resolution of approximately 1 mm [62, 63].

The spatial and orientation information sent to MATLAB is used in a feedback controller to automatically control the position and orientation of the helicopter. Vicon measures orientation angles with respect to a global reference frame. The feedback controller requires an Euler angle representation of the helicopter orientation, where the Euler angles are defined using a ZYX Tait-Bryan convention. Therefore, the measured orientation angles are converted to Euler angles in MATLAB. The control signals calculated by the feedback

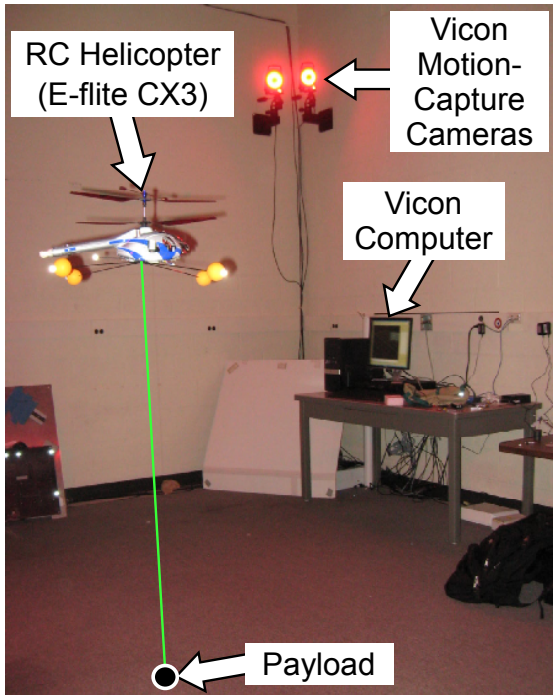


Figure 11: Photograph of the CX3 helicopter flying in the Indoor Fight Facility.

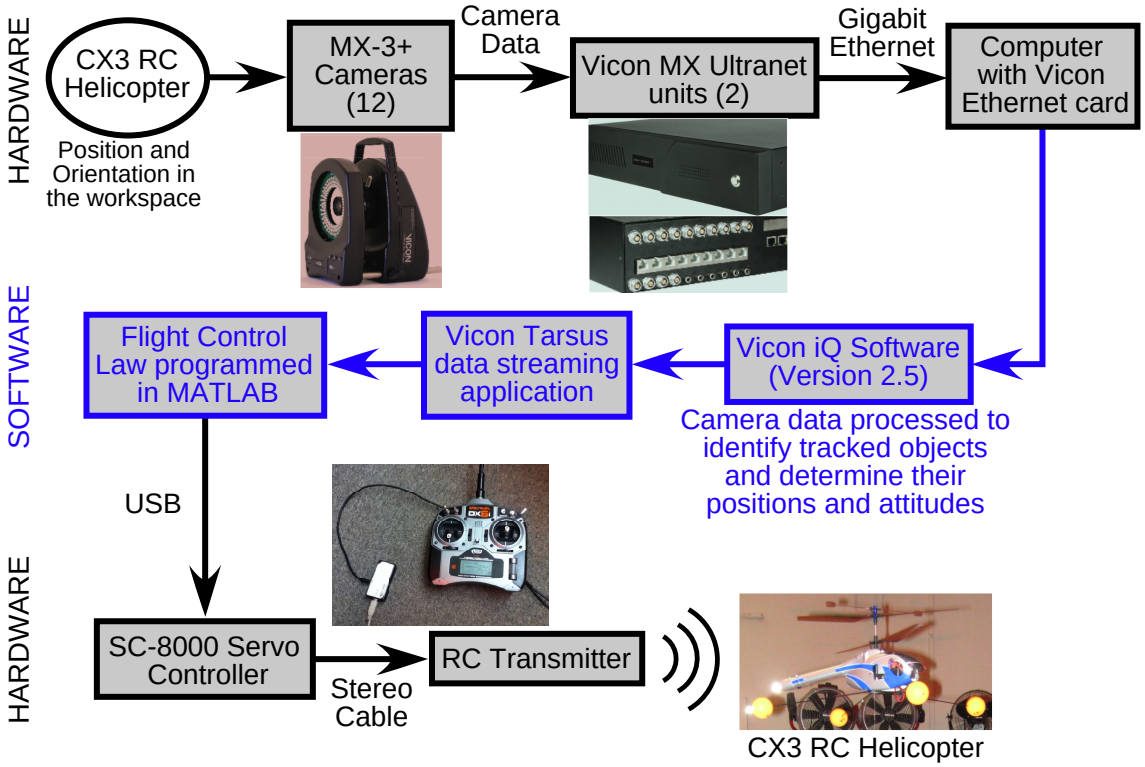


Figure 12: Motion capture and flight controller signal flow [63, 64].

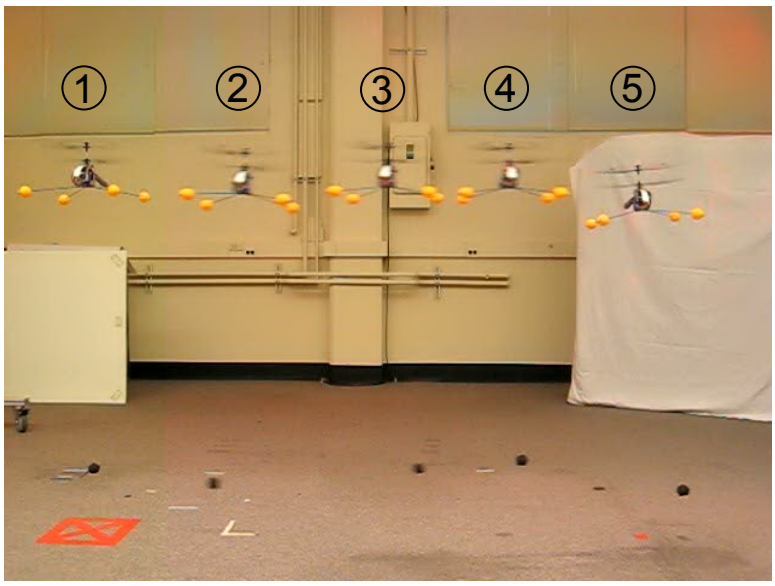


Figure 13: Composite photo of the CX3 helicopter performing a lateral move.

controller are then sent to the helicopter through a Spektrum DX6i transmitter connected to the computer using a SC-8000 Servo Controller that performs serial to Pulse Position Modulation (PPM) conversion.

2.1.3.2 Experimental Validation

The responses predicted by the nonlinear model given by (1) and (2) were compared to experimental results from the CX3 RC helicopter flying in the Indoor Flight Facility.

For the experiments in this section, lateral commands were sent to the CX3 helicopter. The flight controller used feedback of the helicopter's position and attitude to suppress motion in all of the helicopter's degrees of freedom except for the lateral position and roll angle. The sling load position was not recorded by the motion capture system but was later extracted from video recordings of the experimental trials.

The lateral move executed by the helicopter is illustrated by the photo composite shown in Figure 13. The helicopter begins in hover at the left of the image (step 1), moves to the right approximately 7 ft (steps 2-4), and resumes hovering at the target location (step 5). The command sent to the helicopter to execute this lateral move was preprogrammed in the flight controller to ensure consistent responses for each experimental trial.

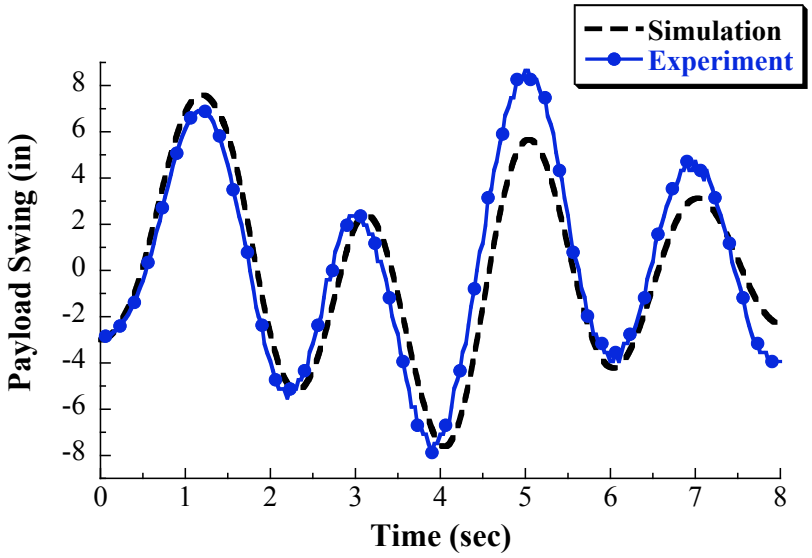
The model given by (1) and (2) was simulated with parameters matching the physical

parameters of the CX3 helicopter and its payload and with the same initial conditions seen in the experimental data. Because the experimental payload deflection was measured in terms of horizontal displacement relative to the helicopter, the simulated payload deflection angle was converted to horizontal displacement relative to the helicopter using (3).

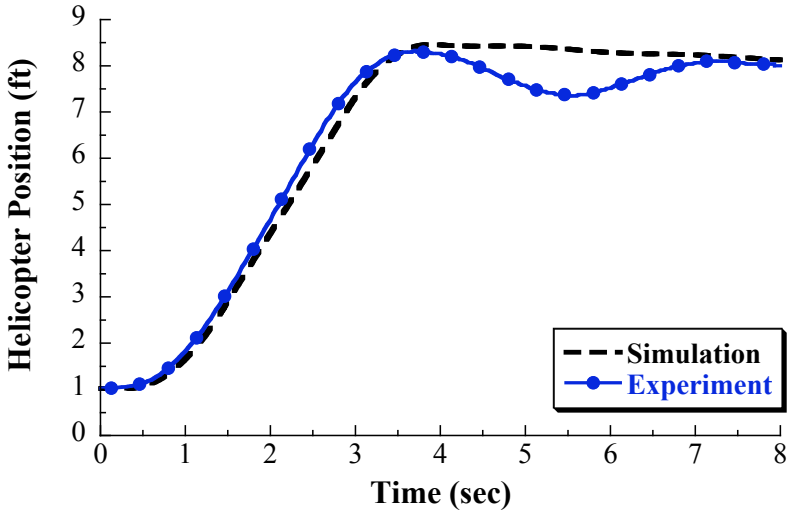
Figure 14a shows a plot comparing the experimental and simulated payload responses to the lateral move, which began at 0.35 s. The model accurately predicts the payload response for the first two periods.

Figure 14b shows a plot of the helicopter position as it executes the lateral move. The model does not predict as much translational oscillation of the helicopter following the command as seen in the experimental trial. This is likely because the model does not account for coupling between the roll attitude and the load. As a result of this coupling, the load swing causes changes in the roll attitude not predicted by this model. This load-attitude coupling effect causes changes in the thrust vector angle that result in changes in helicopter acceleration. The acceleration changes cause the small oscillations seen in the position trajectory of the CX3 helicopter. If more accurate predictions of the helicopter response are required, then the next step in modeling complexity is to include the helicopter attitude as a state with the suspension cable attachment point offset from the helicopter center of gravity. This would incorporate load-attitude coupling in the model. Another possible modeling solution could be to use a 5th-order model similar to that proposed in [28], where the helicopter-load system was shown to behave similarly to a double pendulum. The load-attitude coupling approach will be considered in the next section.

While the helicopter response is not as accurately predicted by the model as the payload response, an accurate prediction of the payload response is more important for designing input shapers that will suppress the payload swing. The natural frequency and damping ratio of the experimental payload swing were approximately 3.2 rad/s and 0.09, respectively, while the values predicted by the simulation were 3.15 rad/s and 0.1. This shows close agreement between the simulation and the experimental trial in terms of the two important input-shaper design parameters. Therefore, the simulation predicts the payload swing with enough accuracy to design an effective input shaper for the experimental CX3



(a) Payload response.



(b) Helicopter response.

Figure 14: Comparison of simulated and experimental (a) payload and (b) helicopter responses.

helicopter. The effectiveness of an input shaper shaper designed using these parameters will be investigated in Chapter III.

The estimate for the natural frequency given by (8), found by linearizing the equations of motion, also provides an accurate estimate of the natural frequency of the CX3 helicopter configuration. (8) predicts a natural frequency of 3.18 rad/s, which is only about 1% less than the actual natural frequency of 3.2 rad/s. This is also enough accuracy for designing

an effective input shaper.

2.2 Load-Attitude Coupling Model

The next step in modeling complexity is to incorporate the helicopter attitude and an offset between the helicopter center of gravity and the load cable suspension point into the model. This model is called the load-attitude coupling model because incorporating the helicopter attitude in the model will introduce a coupling effect between the load and the helicopter attitude. This coupling is an important factor because the load suspension cable is not attached to the helicopter at its center of gravity, so the tension in the cable produces an oscillatory torque about the helicopter center of gravity as the load swings.

2.2.1 Model Description

Figure 15 shows a schematic diagram representing this model. The helicopter rigid body is attached to a massless cart that translates horizontally and enables the helicopter to rotate about the pivot point G. Point G is also the helicopter center of gravity. It is assumed that the helicopter maintains a constant heading and altitude, and its attitude and position in the off-axis direction (perpendicular to the page) remain constant. As with the planar crane model, the suspension cable is assumed to be rigid and massless, the load is modeled as a point-mass, the effects of aerodynamic drag are modeled as viscous damping terms, and rotor downwash effects are neglected.

The horizontal displacement of the helicopter's center of gravity is given by x, the helicopter attitude relative to horizontal is given by θ, and the load deflection angle relative to vertical is given by β. The helicopter and payload masses are M and m, respectively, the helicopter moment of inertia about point G is I_G, and the suspension cable length is L. There is a linear damper attached to the helicopter at point G, and its viscous damping coefficient is given by c. Also, there is a rotational damper on the helicopter attitude, and its viscous damping coefficient is given by d.

Several frames will be used in the derivation. The inertial frame has unit vectors $\hat{I}$ and $\hat{J}$. Another frame attached to the helicopter has unit vectors $\hat{i}$ and $\hat{j}$. The helicopter attitude θ is measured clockwise relative to $\hat{I}$, and load deflection angle β is measured

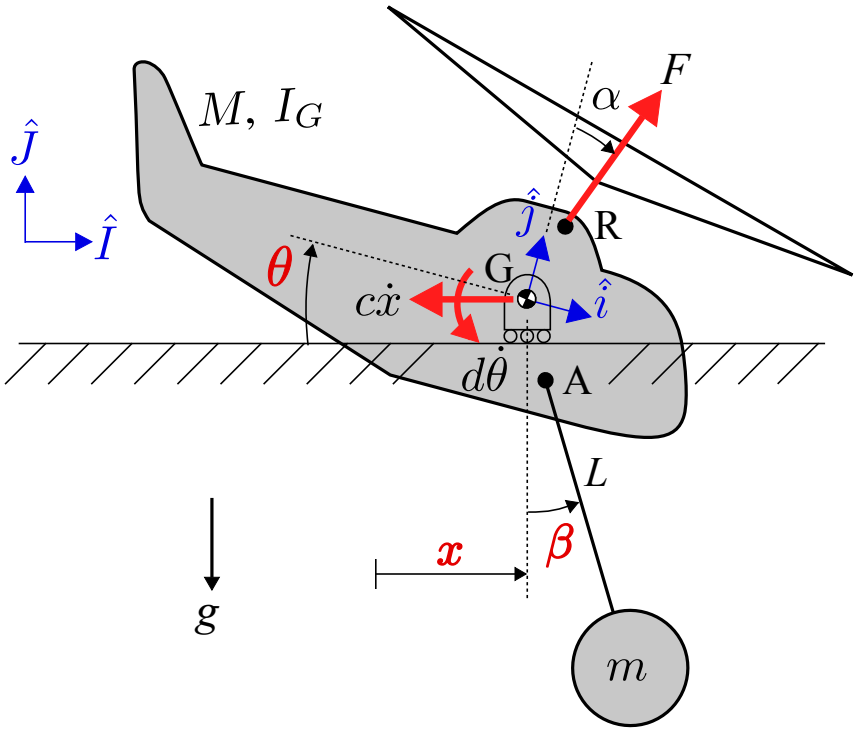


Figure 15: Schematic diagram of a helicopter carrying a suspended load with the helicopter attitude incorporated.

counterclockwise relative to $\hat{J}$.

This model uses a quasistatic representation, or approximation, of the rotor dynamics. The quasistatic rotor representation neglects the rotor dynamics by assuming that the rotor tilt relative to the helicopter fuselage can be changed instantaneously using the cyclic controls [15, 23]. This assumption is justified because the rotor blade flapping response is fast compared to a pilot's control inputs to the rotor [19, 23, 38]. In essence, only the steady-state response of the rotor is considered [23]. Use of the quasistatic rotor approximation is common practice when the models will be used for helicopter stability analysis or in the design of flight control systems [16, 19, 23].

Due to the use of the quasistatic rotor approximation, the control inputs to the model are the longitudinal rotor tilt angle α and the thrust force F applied to the helicopter by the rotor at point R. The force F models the thrust generated by the rotor, and α models the rotor tilt angle relative to the helicopter body. In this sense, the line between points R and G represents the rotor shaft. Note that α is measured clockwise relative to $\hat{j}$.

The distance between the helicopter center of gravity, point G, and the suspension point, point A, is defined as

$$\vec{r}_{A/G} = x_A\hat{i} + y_A\hat{j} \tag{9}$$

The distance between the helicopter center of gravity and point R is defined as

$$\vec{r}_{R/G} = x_R\hat{i} + y_R\hat{j} \tag{10}$$

Note that x_A, y_A, x_R, and y_R are defined relative to the helicopter body. The position vectors $\vec{r}_{A/G}$ and $\vec{r}_{R/G}$ are expressed using those values in the frame attached to the helicopter.

2.2.2 Derivation of Equations of Motion

Lagrange's equations will be used to derive the equations of motion of the load-attitude coupling model. The generalized coordinates are x, θ, and β. Lagrange's equations are given by

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} + \frac{\partial V}{\partial q_j} = Q_j, \qquad j = 1, 2, \ldots, N \tag{11}$$

where q_j are the generalized coordinates, T is the total kinetic energy of the system, V is the total potential energy of the system, Q_j are the generalized forces, and N is the number of generalized coordinates. In this case, $N = 3$ because their are three generalized coordinates.

The total kinetic energy of the helicopter and suspended load is given by

$$T = \frac{1}{2}M\left(\vec{v}_G \cdot \vec{v}_G\right) + \frac{1}{2}I_G\dot{\theta}^2 + \frac{1}{2}m\left(\vec{v}_m \cdot \vec{v}_m\right) + \frac{1}{2}I_m\dot{\beta}^2 \tag{12}$$

where v_G is the velocity of the point G, v_m is the velocity of the payload, and I_m is the moment of inertia of the payload. For a rigid, massless suspension cable with a point-mass payload, I_m is given by

$$I_m = mL^2 \tag{13}$$

The velocity of point G is given by

$$\vec{v}_G = \dot{x}\hat{I} \tag{14}$$

and the velocity of the payload is given by

$$\vec{v}_m = \Big(\dot{x}\cos\theta - x_A\dot{\theta} + \dot{\beta}L\cos\beta\cos\theta - \dot{\beta}L\sin\beta\sin\theta\Big)\,\hat{i} \tag{15}$$

$$+ \Big(\dot{x}\sin\theta + y_A\dot{\theta} + \dot{\beta}L\cos\beta\sin\theta + \dot{\beta}L\sin\beta\cos\theta\Big)\,\hat{j} \tag{16}$$

Substituting (13), (14), and (15) into (12) and simplifying gives the following expression for the kinetic energy

$$\begin{aligned} T = {} & \frac{1}{2}M\dot{x}^2 + \frac{1}{2}\left[I_G + m\left({x_A}^2 + {y_A}^2\right)\right]\dot{\theta}^2 + \frac{1}{2}m\Big[\dot{x}^2 + 2\left(y_A\sin\theta - x_A\cos\theta\right)\dot{x}\dot{\theta} \\ & + 2L\dot{x}\dot{\beta}\cos\beta + 2L\Big(y_A\sin\left(\beta+\theta\right) - x_A\cos\left(\beta+\theta\right)\Big)\dot{\theta}\dot{\beta} + \left(L^2 - L^2\sin 2\beta\sin 2\theta\right)\dot{\beta}^2\Big] \end{aligned} \tag{17}$$

(17) expresses the kinetic energy as a function of the generalized coordinates and velocities. It can be differentiated with respect to the generalized coordinates and velocities for use in Lagrange's equations.

The total potential energy of the helicopter and suspended load is given by

$$V = mg\left(-L\cos\beta - x_A\sin\theta + y_A\cos\theta\right) \tag{18}$$

where the potential energy datum is a horizontal line passing through point G. (18) expresses the potential energy as a function of the generalized coordinates. It can be differentiated with respect to the generalized coordinates for use in Lagrange's equations.

The generalized forces were found using the method of virtual displacements. The generalized force for the helicopter position generalized coordinate x is

$$Q_x = F\sin\left(\alpha+\theta\right) - c\dot{x} \tag{19}$$

The generalized force for the helicopter attitude generalized coordinate θ is

$$Q_\theta = F\left(y_R\sin\alpha - x_R\cos\alpha\right) - d\dot{\theta} \tag{20}$$

The generalized force for the payload deflection angle generalized coordinate β is

$$Q_\beta = 0 \tag{21}$$

The equations of motion can be found by taking the necessary derivatives of (17) and (18), and substituting them and the expressions for the generalized forces given in (19),

(20), and (21) into Lagrange's equations for the three generalized coordinates. This process yields the following equations of motion for x, θ, and β:

$$\begin{aligned}&(M+m)\,\ddot{x} + mL\ddot{\beta}\cos\beta - m\left(x_A\sin\theta - y_A\cos\theta\right)\ddot{\theta} + c\dot{x}\\&- mL\dot{\beta}^2\sin\beta + m\left(x_A\cos\theta + y_A\sin\theta\right)\dot{\theta}^2 = F\sin\left(\alpha+\theta\right)\end{aligned} \tag{22}$$

$$\begin{aligned}&\left[I_G + m\left(x_A^2 + y_A^2\right)\right]\ddot{\theta} + d\dot{\theta} - m\left(x_A\sin\theta - y_A\cos\theta\right)\ddot{x} - gm\left(x_A\cos\theta + y_A\sin\theta\right)\\&- mL\left[x_A\sin\left(\beta+\theta\right) - y_A\cos\left(\beta+\theta\right)\right]\ddot{\beta} - mL\left[x_A\cos\left(\beta+\theta\right) + y_A\sin\left(\beta+\theta\right)\right]\dot{\beta}^2\\&= F\left(y_R\sin\alpha - x_R\cos\alpha\right)\end{aligned} \tag{23}$$

$$\begin{aligned}&L\ddot{\beta} + \ddot{x}\cos\beta - \left[x_A\sin\left(\beta+\theta\right) - y_A\cos\left(\beta+\theta\right)\right]\ddot{\theta}\\&- \left[x_A\cos\left(\beta+\theta\right) + y_A\sin\left(\beta+\theta\right)\right]\dot{\theta}^2 + g\sin\beta = 0\end{aligned} \tag{24}$$

These equations describe the dynamics of the helicopter translation, helicopter orientation, and the payload swing for this model. The correctness of the equations was verified by also using MotionGenesis, a commercial dynamic modeling software package, to independently derive the equations of motion.

2.3 Sikorsky S-61 Models

Designers of helicopter flight control systems often use helicopter models linearized about a given flight condition. These models are often obtained from flight-test data, or flight-test data is used to improve existing models [17, 20, 54]. One such model of a 13,200 lb Sikorsky S-61 helicopter in near-hover operation was investigated by Hall and Bryson [16]. Gupta and Bryson [15] combined Hall and Bryson's linearized, near-hover model of the Sikorsky S-61 with a linear model of a helicopter carrying a suspended load. This gave them a linearized, near-hover model of a Sikorsky S-61 carrying a suspended load. The mass of the suspended load is 4,400 lb and the suspension cable length is 65.6 ft.

Both the unloaded and loaded models only incorporate the longitudinal and lateral motion of the helicopter. The heave and yawing modes of the aircraft are not included. This is equivalent to assuming a fixed heading and altitude for the aircraft. The models also neglect the effects of wind and changes in atmospheric properties as a function of vehicle position. These models will be used in the flight control system design presented in

Chapter IV. The unloaded model will give a baseline performance measure for evaluating the performance of the loaded helicopter model. This will make it easier to identify the effects of carrying a load on the helicopter response.

2.3.1 Unloaded Sikorsky S-61 Model

The S-61 models are state-space models of the form

$$\dot{\vec{x}} = \mathbf{A}\vec{x} + \mathbf{B}\vec{u} \tag{25}$$

Both models use a quasistatic representation of the rotor dynamics [15, 16]. Due to the use of the quasistatic rotor approximation, the control inputs to the models are the longitudinal and lateral rotor disk angles. The rotor disk angles are also sometime referred to as the rotor tilt angles or the angles of the tip-path-plane. The tip-path-plane is the plane formed by the tips of the rotor blades.

The helicopter state and input vectors for the unloaded model are defined as

$$\vec{x} = \begin{bmatrix} \theta_H & \dot{\theta}_H & \dot{x}_H & \phi_H & \dot{\phi}_H & \dot{y}_H \end{bmatrix}^T \tag{26}$$

$$\vec{u} = \begin{bmatrix} \theta_s & \theta_c \end{bmatrix}^T \tag{27}$$

where θ_H is the helicopter fuselage pitch attitude, $\dot{\theta}_H$ is the fuselage pitch rate, $\dot{x}_H$ is the fuselage longitudinal velocity, ϕ_H is the fuselage roll attitude, $\dot{\phi}_H$ is the fuselage roll rate, $\dot{y}_H$ is the fuselage lateral velocity, θ_s is the longitudinal cyclic blade pitch angle, and θ_c is the lateral cyclic blade pitch angle. All units are in ft, sec, and rad.

By neglecting rotor dynamics and through proper scaling of the input matrix $\mathbf{B}$, the input vector given by (27) becomes identical to the rotor tilt angles [16]

$$\vec{u} = \begin{bmatrix} \theta_R & \phi_R \end{bmatrix}^T \tag{28}$$

where θ_R is the longitudinal rotor tilt angle and ϕ_R is the lateral rotor tilt angle, with both angles having units of rad.

The state matrix **A** for the unloaded S-61 model is [16]

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -0.415 & 0.00338 & 0 & 0.318 & 0.00116 \\ -32.2 & 4.70 & -0.0198 & 0 & -1.02 & -0.0059 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1.23 & 0.00415 & 0 & -1.58 & -0.0124 \\ 0 & -1.02 & 0.0059 & 32.2 & -4.70 & -0.0198 \end{bmatrix} \tag{29}$$

and the input matrix **B** is [16]

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 6.27 & 0.295 \\ -32.2 & -0.977 \\ 0 & 0 \\ -1.08 & 23.1 \\ -0.977 & 32.2 \end{bmatrix} \tag{30}$$

The input matrix given by (30) is scaled such that the inputs are the rotor tilt angles and the input vector is given by (28).

2.3.2 Loaded Sikorsky S-61 Model

Gupta and Bryson [15] modified the unloaded Sikorsky S-61 model above to include suspended load dynamics. The helicopter state and input vectors for the loaded model are defined as

$$\vec{x} = \begin{bmatrix} \theta_H & \dot{\theta}_H & x_H & \dot{x}_H & x_L & \dot{x}_L & \phi_H & \dot{\phi}_H & y_H & \dot{y}_H & y_L & \dot{y}_L \end{bmatrix}^T \tag{31}$$

$$\vec{u} = \begin{bmatrix} \theta_R & \phi_R \end{bmatrix}^T \tag{32}$$

where θ_H is the helicopter pitch attitude, $\dot{\theta}_H$ is the helicopter pitch rate, x_H is the helicopter longitudinal position, $\dot{x}_H$ is the helicopter longitudinal velocity, x_L is the load longitudinal position, $\dot{x}_L$ is the load longitudinal velocity, ϕ_H is the helicopter roll attitude, $\dot{\phi}_H$ is the helicopter roll rate, y_H is the helicopter lateral position, $\dot{y}_H$ is the helicopter lateral velocity,

y_L is the load lateral position, $\dot{y}_L$ is the load lateral velocity, θ_R is the longitudinal rotor tilt angle, and ϕ_R is the lateral rotor tilt angle. All units are in ft, sec, and rad.

The state matrix $\mathbf{A}$ for the loaded S-61 is [15]

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \tag{33}$$

where

$$\mathbf{A}_{11} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -2.25 & -0.415 & -0.0317 & 0.0034 & 0.0317 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -43.6 & 4.69 & -0.164 & -0.0198 & 0.164 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 2.41 & 0 & 0.491 & 0 & -0.491 & -0.0026 \end{bmatrix}, \tag{34}$$

$$\mathbf{A}_{12} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.318 & 0 & 0.0012 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.02 & 0 & -0.0059 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{35}$$

$$\mathbf{A}_{21} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.23 & 0 & 0.0041 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.02 & 0 & 0.0059 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{36}$$

and

$$\mathbf{A}_{22} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -8.28 & -1.58 & 0.117 & -0.0124 & -0.117 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 43.6 & -4.69 & -0.164 & -0.0198 & 0.164 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -2.41 & 0 & 0.491 & 0 & -0.491 & -0.0026 \end{bmatrix} \tag{37}$$

and the input matrix $\mathbf{B}$ is [15]

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 8.38 & 0.393 \\ 0 & 0 \\ -3.99 & -0.121 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -1.43 & 30.7 \\ 0 & 0 \\ -0.121 & 3.99 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{38}$$

2.3.3 Analysis of Sikorsky S-61 Models

The unloaded and loaded Sikorsky S-61 models are open-loop unstable. This makes it difficult to simulate their time responses to evaluate the open-loop performance of the aircraft in unloaded and loaded configurations. Instead, it can be useful to plot the pole locations for each model. This is typically done to enable identification of various aircraft flight modes since each flight mode is associated with an eigenvalue or pole [36]. This analysis can be performed with uncoupled longitudinal and lateral helicopter models, and it is often simpler to analyze the uncoupled models [36]. However, in this thesis the coupled

models will be used.

The poles can be found by calculating the eigenvalues of the models' **A** matrices. Each real eigenvalue or pair of complex conjugate eigenvalues corresponds to a flight mode of the helicopter [36]. The eigenvectors corresponding to each mode can be used to determine what model states are most affected by the mode. Each row of an eigenvector corresponds to one state from the model. States related to each flight mode have a larger magnitude in their row of the eigenvector. By determining the states that have the largest contribution in a given eigenvector, the helicopter's flight modes can be identified.

Table 1 shows the eigenvalues and eigenvectors of the unloaded Sikorsky S-61 **A** matrix given in (29). The flight mode that corresponds to each real eigenvalue or complex conjugate pair is labeled at the top of the table. As an example of how the flight modes were determined, note that the value of the third row of the Long Term Longitudinal Oscillation Mode eigenvectors is 0.8717. This value is larger than the other elements of those eigenvectors. It is also larger than any of the elements in the third row of the other eigenvectors. The third row corresponds to the longitudinal velocity state, $\dot{x}_H$. This suggests that the mode represented by these two eigenvectors is related to the longitudinal motion of the helicopter. Because the corresponding eigenvalues are $0.1092 \pm 0.3635i$, this mode is oscillatory. Its oscillatory nature and its relation to the longitudinal motion of the helicopter give the Long Term Longitudinal Oscillation Mode its name. This mode is also referred to as the long phugoid or long-period phugoid mode [36]. Note that this mode is also unstable.

A plot of the real and imaginary parts of the eigenvalues of the unloaded model is shown in Figure 16. The mode names are labeled for each real eigenvalue or complex conjugate pair. The source of the real, negative-valued Short Term Pitch Mode is the rotor pitch damping [23]. The Long Term Longitudinal Oscillation Mode corresponds to the unstable oscillation caused by the coupling of the helicopter pitch and longitudinal velocity [23]. The instability results from coupling between the pitch moments caused by longitudinal velocity, or speed stability, and the component of the helicopter weight force acting on the helicopter in the longitudinal direction due to the pitch angle [23].

The Roll Mode is a result of rotor roll damping [23]. The Roll Mode is faster than the

Table 1: Unloaded Sikorsky S-61 model [16] eigenvectors, with the associated eigenvalues and flight modes labeled.

Modes		Roll Mode	Short Term Pitch Mode	Lateral Oscillation Mode		Long Term Longitudinal Oscillation Mode	
Eigenvalues		-1.2700	-1.0680	$0.0426 + 0.4962i$	$0.0426 - 0.4962i$	$0.1092 + 0.3635i$	$0.1092 - 0.3635i$
States	θ_H	0.0435	0.0507	$0.0186 - 0.0019i$	$0.0186 + 0.0019i$	$-0.0097 - 0.0335i$	$-0.0097 + 0.0335i$
	$\dot{\theta}_H$	-0.0552	-0.0541	$0.0017 + 0.0092i$	$0.0017 - 0.0092i$	$0.0111 - 0.0072i$	$0.0111 + 0.0072i$
	$\dot{x}_H$	0.4390	0.5772	$0.0167 + 0.3618i$	$0.0167 - 0.3618i$	0.8717	0.8717
	ϕ_H	-0.0965	-0.0762	$0.0025 + 0.0476i$	$0.0025 - 0.0476i$	$-0.0163 + 0.0107i$	$-0.0163 - 0.0107i$
	$\dot{\phi}_H$	0.1225	0.0814	$-0.0235 + 0.0033i$	$-0.0235 - 0.0033i$	$-0.0057 - 0.0047i$	$-0.0057 + 0.0047i$
	$\dot{y}_H$	0.8820	0.8055	0.9303	0.9303	$0.1493 + 0.4649i$	$0.1493 - 0.4649i$

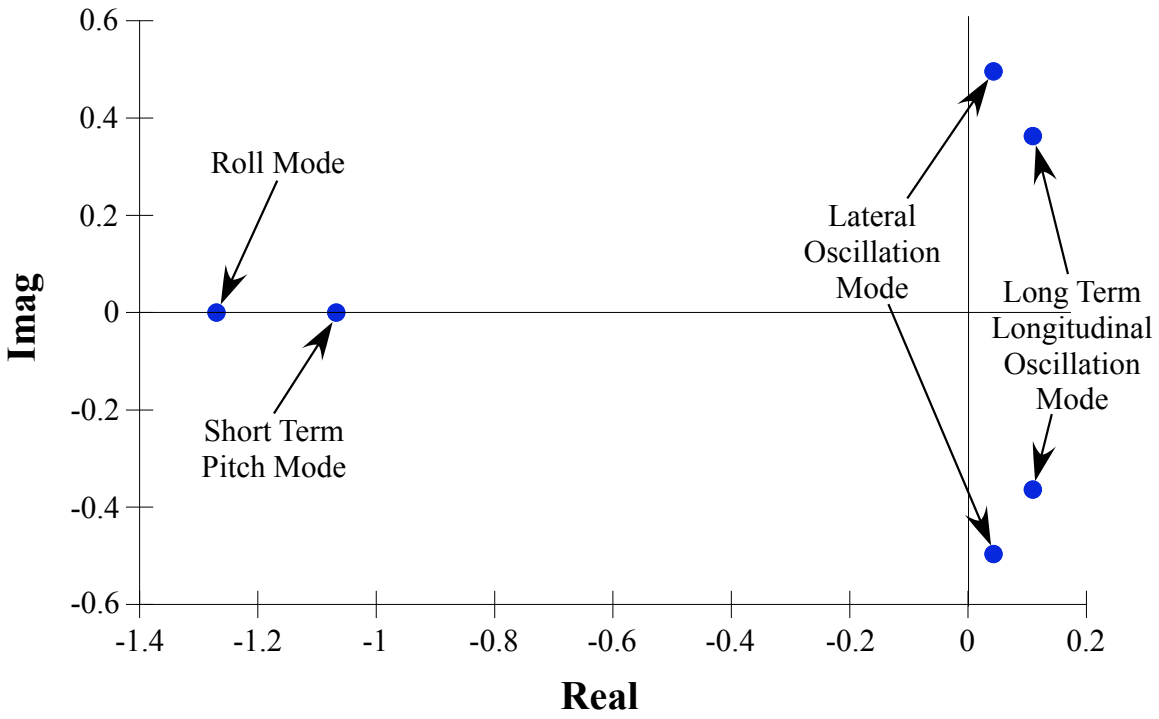


Figure 16: Eigenvalues of the unloaded Sikorsky S-61 model from [16].

Short Term Pitch Mode because the helicopter, like most traditional single-rotor helicopters, has a smaller moment of inertia about the roll axis than the pitch axis. The Lateral Oscillation Mode is unstable as a result of the rotor dihedral effect [23]. The instability of the Lateral Oscillation Mode is more objectionable to pilots than the instability of the Long Term Longitudinal Mode because it has a higher frequency [23]. This means that it is important for the flight controller to stabilize this mode.

While resources such as Johnson [23] and Padfield [36] present detailed discussions of flight modes for unloaded helicopters with a variety of rotor configurations, not much work has been done on identifying or labeling flight modes of a helicopter with a suspended load. The following analysis will attempt to identify flight modes of the loaded Sikorsky S-61. This analysis will show that poles, and therefore the behavior of the flight modes, change significantly when the helicopter is carrying a suspended load.

Table 2 shows the eigenvalues and eigenvectors of the loaded Sikorsky S-61 **A** matrix given in (33). The flight mode that corresponds to each real eigenvalue or complex conjugate pair is labeled above each eigenvalue and eigenvector. Also, a plot of the real and imaginary

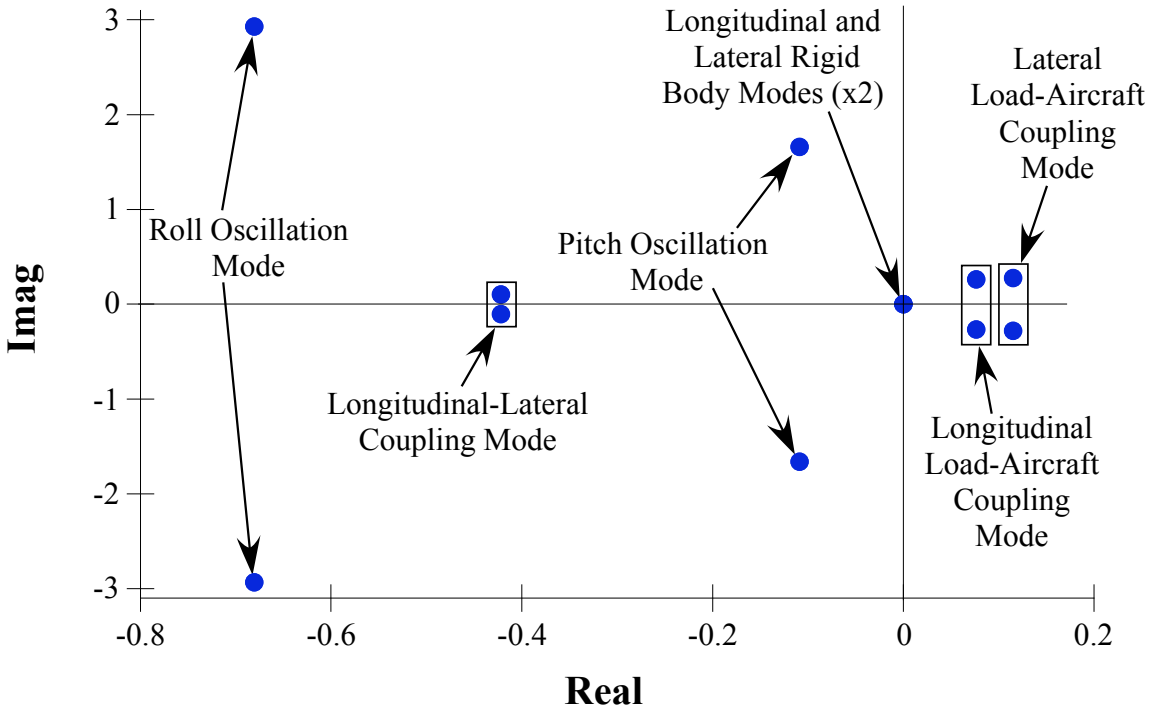


Figure 17: Eigenvalues of the loaded Sikorsky S-61 model from [15].

parts of the eigenvalues is shown in Figure 17.

The first two modes listed in Table 2 result from the relationships between the helicopter attitudes, attitude rates, and translational velocities. The eigenvectors of the first mode have strong contributions from the roll rate and lateral velocity, and the mode is oscillatory. Similarly, the eigenvectors of the longitudinal mode have strong contributions from the pitch rate and longitudinal velocity, and the mode is also oscillatory. As a result, the first two modes are labeled the Roll and Pitch Oscillation Modes, respectively.

The eigenvectors of the third mode have the largest contributions from the helicopter longitudinal and lateral translational motion. The entries in the eigenvectors in the rows related to the helicopter longitudinal and lateral positions are approximately 90 degrees out of phase. This suggests an exchange of energy between the longitudinal and lateral degrees of freedom indicative of coupling between the two directions. For this reason, this mode is labeled the Longitudinal-Lateral Coupling Mode.

The eigenvectors of the fourth mode suggest coupling between the lateral helicopter and load translational motion. This mode has a complex conjugate pair of eigenvalues that have

Table 2: Loaded Sikorsky S-61 model [15] eigenvectors, with the associated eigenvalues and flight modes labeled.

Modes		Roll Oscillation Mode		Pitch Oscillation Mode		Longitudinal-Lateral Coupling Mode	
Eigenvalues		$-0.6804 + 2.9306i$	$-0.6804 - 2.9306i$	$-0.1087 + 1.6593i$	$-0.1087 - 1.6593i$	$-0.4219 + 0.1048i$	$-0.4219 - 0.1048i$
States	θ_H	$0.0212 + 0.0069i$	$0.0212 - 0.0069i$	$0.0207 - 0.0854i$	$0.0207 + 0.0854i$	$-0.0052 - 0.0063i$	$-0.0052 + 0.0063i$
	$\dot{\theta}_H$	$-0.0348 + 0.0575i$	$-0.0348 - 0.0575i$	$0.1394 + 0.0436i$	$0.1394 - 0.0436i$	$0.0028 + 0.0021i$	$0.0028 - 0.0021i$
	x_H	$0.0183 - 0.0206i$	$0.0183 + 0.0206i$	$-0.0309 - 0.4720i$	$-0.0309 + 0.4720i$	$0.1245 + 0.4794i$	$0.1245 - 0.4794i$
	$\dot{x}_H$	$0.0480 + 0.0676i$	$0.0480 - 0.0676i$	0.7866	0.7866	$-0.1028 - 0.1892i$	$-0.1028 + 0.1892i$
	x_L	$-0.0023 + 0.0018i$	$-0.0023 - 0.0018i$	$0.0202 + 0.1277i$	$0.0202 - 0.1277i$	$0.0394 + 0.3566i$	$0.0394 - 0.3566i$
	$\dot{x}_L$	$-0.0038 - 0.0079i$	$-0.0038 + 0.0079i$	$-0.2141 + 0.0197i$	$-0.2141 - 0.0197i$	$-0.0540 - 0.1463i$	$-0.0540 + 0.1463i$
	ϕ_H	$-0.0799 + 0.1398i$	$-0.0799 - 0.1398i$	$-0.0203 - 0.0041i$	$-0.0203 + 0.0041i$	$0.0083 - 0.0039i$	$0.0083 + 0.0039i$
	$\dot{\phi}_H$	$-0.3552 - 0.3292i$	$-0.3552 + 0.3292i$	$0.0089 - 0.0332i$	$0.0089 + 0.0332i$	$-0.0031 + 0.0025i$	$-0.0031 - 0.0025i$
	y_H	$-0.0605 - 0.2606i$	$-0.0605 + 0.2606i$	$0.1256 - 0.0086i$	$0.1256 + 0.0086i$	0.5534	0.5534
	$\dot{y}_H$	0.8049	0.8049	$0.0006 + 0.2093i$	$0.0006 - 0.2093i$	$-0.2335 + 0.0580i$	$-0.2335 - 0.0580i$
	y_L	$0.0094 + 0.0253i$	$0.0094 - 0.0253i$	$-0.0331 + 0.0058i$	$-0.0331 - 0.0058i$	$0.3966 + 0.0576i$	$0.3966 - 0.0576i$
	$\dot{y}_L$	$-0.0806 + 0.0103i$	$-0.0806 - 0.0103i$	$-0.0060 - 0.0556i$	$-0.0060 + 0.0556i$	$-0.1734 + 0.0173i$	$-0.1734 - 0.0173i$

Modes		Lateral Load-Aircraft Coupling Mode		Longitudinal Load-Aircraft Coupling Mode		Longitudinal Rigid Body Mode	Lateral Rigid Body Mode
Eigenvalues		$0.1149 + 0.2796i$	$0.1149 - 0.2796i$	$0.0763 + 0.2625i$	$0.0763 - 0.2625i$	0	0
States	θ_H	$0.0032 + 0.0025i$	$0.0032 - 0.0025i$	$-0.0033 + 0.0018i$	$-0.0033 - 0.0018i$	0	0
	$\dot{\theta}_H$	$-0.0003 + 0.0012i$	$-0.0003 - 0.0012i$	$-0.0007 - 0.0007i$	$-0.0007 + 0.0007i$	0	0
	x_H	$0.0244 + 0.4090i$	$0.0244 - 0.4090i$	$-0.4529 - 0.0463i$	$-0.4529 + 0.0463i$	-0.7004	-0.2774
	$\dot{x}_H$	$-0.1116 + 0.0538i$	$-0.1116 - 0.0538i$	$-0.0224 - 0.1224i$	$-0.0224 + 0.1224i$	0	0
	x_L	$0.1037 + 0.4596i$	$0.1037 - 0.4596i$	-0.5250	-0.5250	-0.7004	-0.2774
	$\dot{x}_L$	$-0.1166 + 0.0818i$	$-0.1166 - 0.0818i$	$-0.0401 - 0.1378i$	$-0.0401 + 0.1378i$	0	0
	ϕ_H	$-0.0037 + 0.0030i$	$-0.0037 - 0.0030i$	$0.0023 + 0.0028i$	$0.0023 - 0.0028i$	0	0
	$\dot{\phi}_H$	$-0.0013 - 0.0007i$	$-0.0013 + 0.0007i$	$-0.0006 + 0.0008i$	$-0.0006 - 0.0008i$	0	0
	y_H	$0.4698 + 0.0769i$	$0.4698 - 0.0769i$	$-0.0322 - 0.4371i$	$-0.0322 + 0.4371i$	-0.0970	0.6504
	$\dot{y}_H$	$0.0325 + 0.1402i$	$0.0325 - 0.1402i$	$0.1123 - 0.0418i$	$0.1123 + 0.0418i$	0	0
	y_L	0.5475	0.5475	$-0.0882 - 0.4977i$	$-0.0882 + 0.4977i$	-0.0970	0.6504
	$\dot{y}_L$	$0.0629 + 0.1531i$	$0.0629 - 0.1531i$	$0.1239 - 0.0611i$	$0.1239 + 0.0611i$	0	0

a positive real part, suggesting that the mode is unstable. Similarly, the eigenvectors of the fifth mode suggest coupling between the longitudinal helicopter and load translational motion. This mode is also unstable because it has a complex conjugate pair of eigenvalues that have a positive real part. These modes were labeled Lateral and Longitudinal Load-Aircraft Coupling Modes. Note that the lateral mode is further from the origin, indicating that the lateral motion occurs at a higher frequency and will diverge more quickly than the longitudinal mode. The presence of these modes agree with the observation of Hoh et al. [18] that there is strong coupling between the load oscillation and the helicopter translational degrees of freedom. Hoh et al. also show that the strength of this coupling effect has a strong influence on the pilot's controllability of the helicopter, and therefore, on the handling qualities of the helicopter.

The final two modes correspond to the helicopter and load moving together in the longitudinal and lateral directions as if they are one rigid body. They appear in the loaded model due to the integration of the helicopter and load longitudinal and lateral velocities to calculate the helicopter and load positions.

CHAPTER III

INPUT SHAPING CONTROL OF SUSPENDED LOADS[1]

3.1 Input Shaping Background

Input shaping is a command-shaping control technique where the reference command is strategically modified by convolving it with a sequence of impulses. The impulse sequence is called an *input shaper*. When this modified, or input-shaped, command is applied to a vibratory system, the system will respond with little or no residual vibration [39, 50]. The input-shaping technique does not require real-time measurement of the system states to generate the impulse sequence.

Figure 18 demonstrates how two impulses can create a self-canceling vibratory response if they are correctly timed. In the top of Figure 18, an impulse is applied to a flexible dynamic system and induces a lightly-damped response. A similar response (shown by the dashed line) would result if a second impulse was applied a short time later. The bottom of Figure 18 shows the response that results from both impulses. Because the dynamic system is assumed to be linear and time-invariant, the two responses combine and eliminate vibration. Furthermore, the two specially-timed impulses can be convolved with any arbitrary function, and the resulting function will maintain the vibration-canceling properties of the original impulses.

The transfer function of a generic input shaper with n impulses is:

$$\mathbb{G}_{is} = A_1 e^{-t_1 s} + A_2 e^{-t_2 s} + \cdots + A_n e^{-t_n s}, \tag{39}$$

where A_i are the impulse amplitudes, and t_i are the time locations of each impulse. Without loss of generality, the first impulse time is $t_1 \equiv 0$. The impulse amplitudes and time locations are calculated using the estimated natural frequencies and damping ratios of flexible modes to be suppressed. These parameters are never known exactly in real situations. To account

[1]Parts of this chapter have been accepted for publication in the 2012 12th International Conference on Control, Automation and Systems

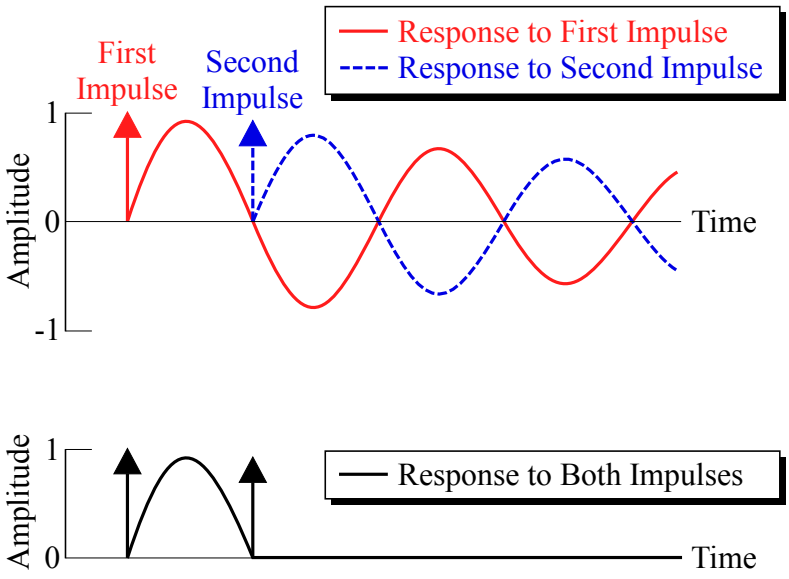


Figure 18: Two self-canceling impulses.

for this uncertainty, researchers have developed robust input shapers [39, 42, 41, 44, 47, 60] that are effective even when there are large errors in the estimated parameter values.

Various input shapers can be designed using different combinations of performance requirements. By constraining the impulses to be all positive and the residual vibration to be zero when parameter estimates are perfect, a *Zero Vibration* (ZV) shaper [50] is obtained. Its transfer function is:

$$\mathbb{G}_{zv} = A_1 + A_2 e^{-t_2 s}, \tag{40}$$

where A_1, A_2, and t_2 depend on the natural frequency and damping ratio of the flexible mode. Only the ZV shaper will be used in this thesis. For a description of many other kinds of shapers, see [60, 61].

Input shapers can be designed to suppress multiple flexible modes [22, 40, 42, 46], such as those caused by double pendulum and distributed-mass payloads. In addition, many studies of crane operators have shown that input shaping can greatly improve performance [25, 26, 30]. The primary disadvantages of input shaping are that it cannot reduce vibration caused by external disturbances and it introduces a small time lag due to the method used to form the shaped commands.

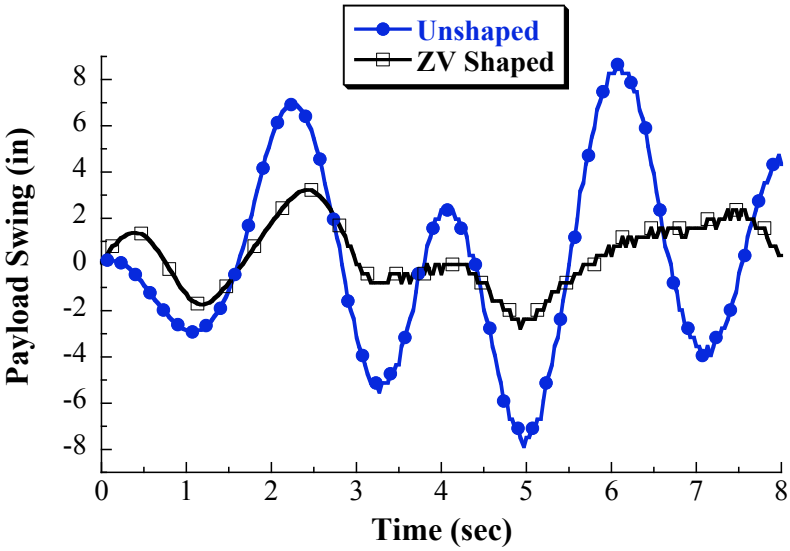
3.2 Input Shaping on the Blade CX3 Radio-Controlled Helicopter

The goal of this section is to demonstrate the performance improvement obtained by using input-shaped commands on the CX3 helicopter when it is carrying a suspended load. A ZV shaper was selected because the natural frequency and damping ratio of the payload swing were found fairly accurately from the experimental trial shown in Figure 14. A natural frequency of 3.1 rad/s and a damping ratio of 0.1 were used to design the ZV shaper.

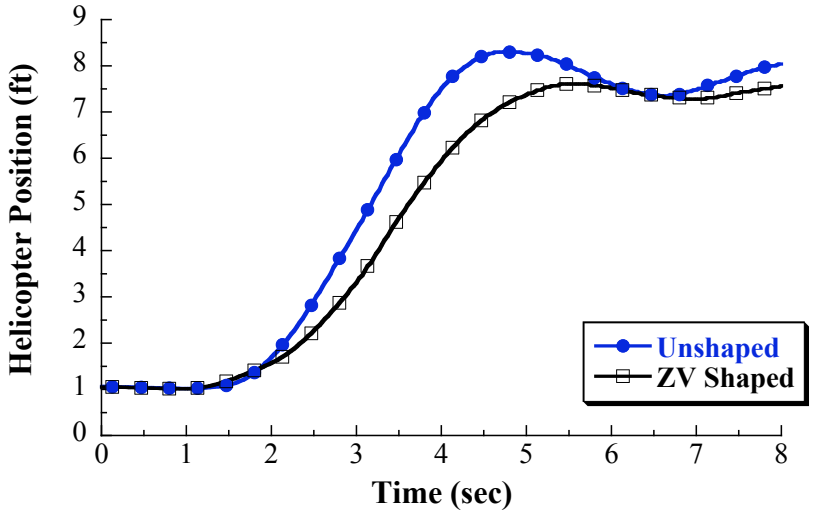
Figure 19 shows the payload and helicopter responses from the same experimental unshaped trial from Figure 14 compared to the payload and helicopter responses from a trial where a ZV shaper was used to shape the lateral command. The lateral command began at 1.35 seconds. As shown in Figure 19a, the ZV shaper significantly reduces the amount of payload swing caused by the helicopter motion. The residual swing peak-to-peak amplitude was reduced from 17 inches to 4 inches by using input shaping. The swing is not completely eliminated by the input shaping because the input shaper does not specifically target the small amount of initial swing present as the command began at 1.35 seconds.

The ZV-shaped helicopter response shown in Figure 19b is only slightly slower due to the time lag introduced by the shaper. Also, the helicopter arrives at the final hover position with much less overshoot and back-drive than in the unshaped case, even though the shaper was not designed to suppress the frequency of the helicopter translational oscillation. This is a result of the coupling between the load oscillation and the helicopter translation.

These results show that input shaping can be used to suppress the payload swing on the experimental helicopter. Also, the natural frequency predicted by the nonlinear planar crane model given by (1) and (2) (3.15 rad/s) was close to the natural frequency used to suppress the payload swing in the experiment (3.1 rad/s), and the damping ratio was nearly the same. This suggests that the model can predict the natural frequency and damping ratio of the experimental helicopter's payload swing with sufficient accuracy for designing effective ZV shapers. The expression for the natural frequency obtained from the linearized planar crane model, given by (8), predicts a natural frequency of 3.18 rad/s, which is only about 1% less than the actual natural frequency of 3.2 rad/s. This is also enough accuracy for designing an effective input shaper. As a result, the model can justifiably be used to



(a) Payload response.



(b) Helicopter response.

Figure 19: Comparison of unshaped and ZV-shaped experimental (a) payload and (b) helicopter responses for the CX3 helicopter.

estimate the natural frequency and damping ratio of other helicopter-load configurations and use them as first approximations for designing input shapers. Another helicopter-load configuration will be investigated in the next section to further verify if the natural frequency estimates provided by the model are effective for other helicopter-load configurations.

3.3 Input Shaping on the Planar Experimental Radio-Controlled Helicopter

The helicopter experimental setup that will be used in this section is the Planar Experimental Radio-Controlled Helicopter (PERCH). This helicopter will be used to further demonstrate the effectiveness of input shaping at suppressing suspended load oscillations.

3.3.1 Planar Experimental Radio-Controlled Helicopter

A photograph of the PERCH experimental setup is shown in Figure 20. The setup consists of an E-flite Blade 400 RC helicopter attached to a frame. The frame pivots about two carts that roll on the two guide rails. The helicopter is only able to pitch and translate forwards and backwards. The guide rails restrict motion in the lateral, yaw, and heave directions. A hard stop is positioned at the ends of each guide rail to stop the helicopter. A suspended load is attached to the helicopter. The round objects seen below the helicopter are colored circles attached to the helicopter frame that are used to extract the helicopter attitude from videos of experimental trials using a MATLAB image processing program.

A schematic diagram of the PERCH expeirmental setup is shown in Figure 21. Rather than directly transmitting operator commands to the helicopter using the manually-controlling transmitter, the commands are sent to a microcontroller that can modifiy the operator's commands. The microcontroller receives input from the manually-controlling transmitter using a 10-bit analog-to-digital converter. The commands are sent from the microcontroller to the helicopter via the signal-sending transmitter. The microcontroller sends the commands to the signal-sending transmitter using 8-bit Pulse-Width Modulation (PWM). The signal-sending transmitter then transmits the commands to the helicopter. This setup allows an operator to fly the RC helicopter using the manually-controlling transmitter, and input shaping may be applied to the operator's commands using the microcontroller. The microcontroller used in the PERCH setup is an Arduino UNO microcontroller.

The mass of the helicopter and its frame is 2.82 lb. The mass of the cart on each guide rail is 0.9 lb. The mass of the suspended load is 1.2 lb and the suspension cable length is 72 inches. Due to the hard stops and the length of the guide rails, the usable workspace is

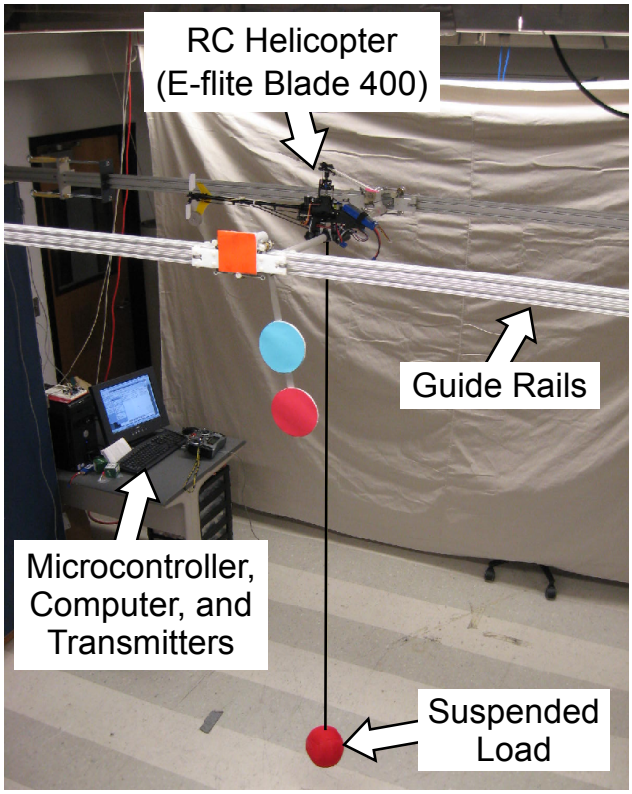


Figure 20: Photograph of the Planar Experimental Radio-Controlled Helicopter experimental setup.

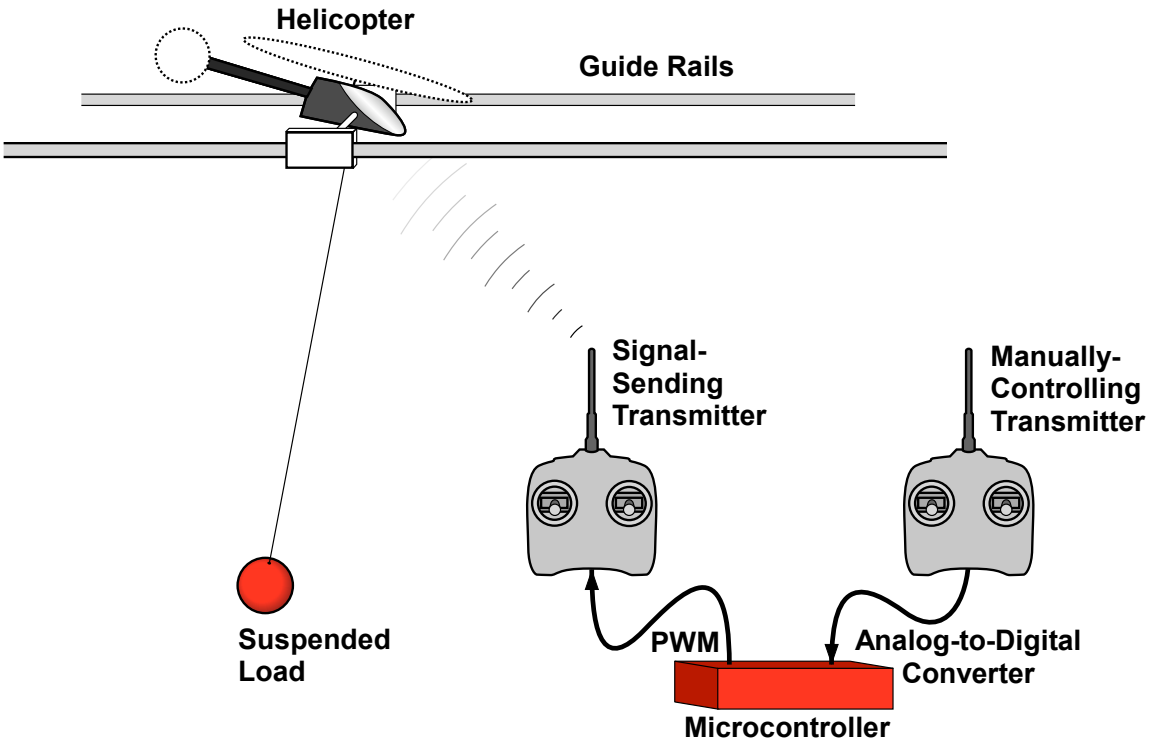


Figure 21: Schematic diagram of the Planar Experimental Radio-Controlled Helicopter.

80 inches long.

By constraining the motion of the helicopter to the pitch direction only, this experimental setup slightly changes the dynamics of the helicopter. In an unconstrained helicopter, pitching forward is accompanied by downward motion. This occurs because the increase in pitch angle leads to a decrease in the vertical component of the thrust vector. The smaller vertical force does not fully support the weight of the helicopter, so pitching forward leads to a small downwards acceleration. When carrying a suspended load, this downward acceleration of the helicopter causes effects similar to a pendulum with a vertically accelerating suspension point. The accelerating suspension point affects the natural frequency load swing. This coupling between the pitch and vertical motions is small compared to the helicopter pitch and suspended load swing dynamics for small to moderate pitch angles. By allowing the helicopter to pitch and translate in the longitudinal direction, the experimental setup captures many of the important dynamics, such as coupling between the load and helicopter, and allows these effects to be studied.

3.3.2 Input Shaper Design

To design an input shaper for suppressing the oscillation of PERCH's suspended load, an estimate of the natural frequency and damping ratio of the oscillation is needed. An estimate of the natural frequency can be obtained using the expression for the natural frequency obtained from the linearized planar crane model (8). Including the mass of both carts, the total effective helicopter mass is 4.62 lb. Using this value for the helicopter mass, the suspended load mass of 1.2 lb, and the suspension cable length of 72 inches, (8) yields a natural frequency of 2.6 rad/s. This frequency is labeled in the 3D plot of natural frequencies shown in Figure 10.

An impulse response was performed to verify this estimate of the natural frequency and measure the damping ratio of the suspended load oscillation. The impulse response was performed by quickly pushing the load while the helicopter hovered at the center of the guide rail workspace. The suspended load oscillation relative to the helicopter position was extracted from a video recording of this experimental trial.

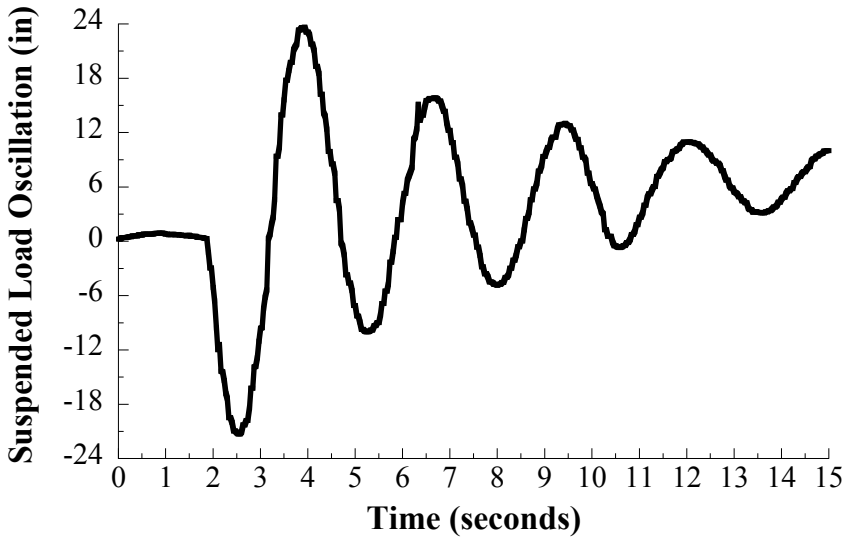


Figure 22: Planar Experimental Radio-Controlled Helicopter suspended load impulse response.

The impulse response of the suspended load is shown in Figure 22. The impulse was applied at approximately two seconds. The oscillation amplitude damps to less than 50% of the peak amplitude within four periods. The natural frequency and damping ratio of the oscillation can be calculated from this impulse response using the log decrement method. The log decrement method yielded a natural frequency of 2.27 rad/s and a damping ratio of 0.075. These values were used to design a ZV input shaper. The effectiveness of this input shaper at suppressing suspended load oscillation will be shown in the next section.

The natural frequency of 2.6 rad/s predicted by (8) has an error of 12.7% relative to the actual frequency of 2.27 rad/s. If the natural frequency approximation given by (8) is used to design a ZV input shaper, then this ZV input shaper will not fully suppress the suspended load oscillation due to the 12.7% error between the estimated and actual values. This is because the ZV input shaper is not very robust to error between the natural frequency used to design the input shaper and the actual natural frequency [39].

However, other types of input shapers have been derived that are more robust to error in the natural frequency, such as the Zero Vibration and Derivative [39] and Extra-Insensitive [44] input shapers. These robust input shapers would still be effective at suppressing the suspended load oscillation even with 12.7% error in the natural frequency used to design the

input shaper. Therefore, (8) yields estimates of the natural frequency that are acceptable for use in the design of effective input shapers if more robust types of input shapers are used.

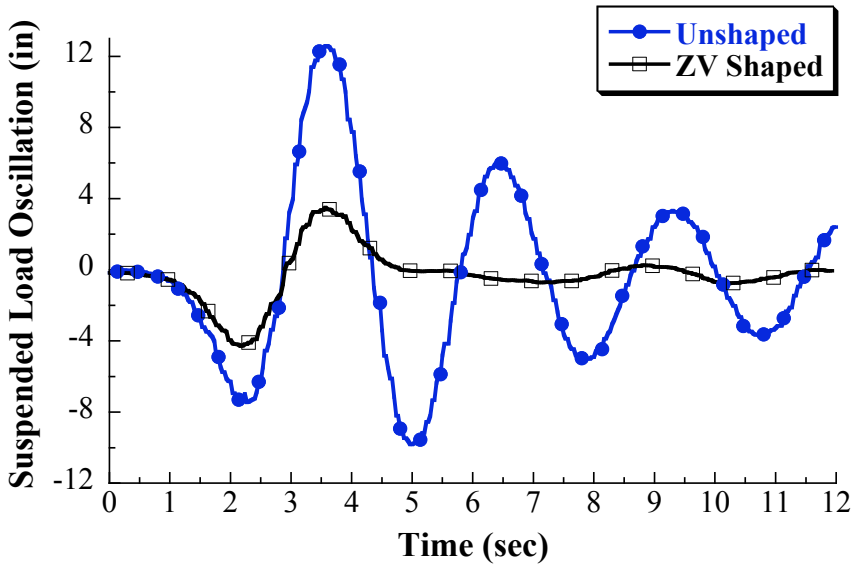
3.3.3 Input Shaping Experimental Results

A ZV input shaper was designed using the natural frequency and damping ratio estimates obtained from the suspended load impulse response

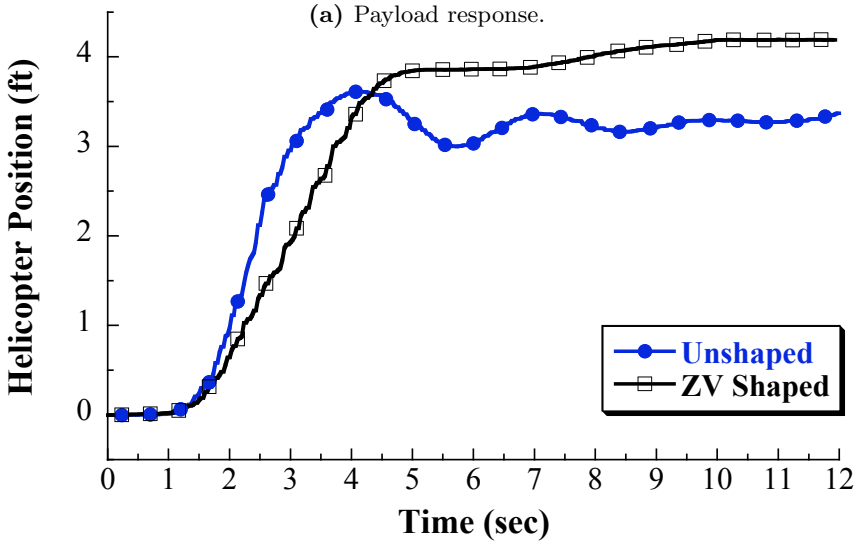
To perform the unshaped and input-shaped trials, a human operator flew the helicopter from one position to another on the guide rails using the manually-controlling transmitter to issue commands. The helicopter's collective pitch input was set at the same level for both trials in the microcontroller. The collective pitch sets the thrust of the helicopter and was set at a level that balances the weight of the helicopter and the load.

Figure 23 shows the payload and helicopter responses for unshaped and ZV input-shaped commands. Figure 23a shows a plot of the suspended load oscillation relative to the helicopter. The use of the ZV input shaper significantly reduces the amount of payload swing caused by the helicopter motion. Input shaping reduces the residual oscillation peak-to-peak amplitude from 22 inches to 1 inch.

The helicopter responses are shown in Figure 23b. In the unshaped response, the load oscillation significantly back-drove the helicopter following the move. This back-driving is not seen in the ZV-shaped response because the oscillation of the load was suppressed by the input shaping. The ZV-shaped helicopter response was slightly slower. This is due to the scaling of the command by the impulses resulting in less acceleration. The move distances in both responses were slightly different because the human operator's commands were not exactly the same. The helicopter drifts slightly in the shaped response because the human operator was not sending a completely neutral command. It is only noticeable because there is not significant back-driving of the helicopter by the load in the shaped response.



(a) Payload response.



(b) Helicopter response.

Figure 23: Comparison of unshaped and ZV input-shaped experimental (a) payload and (b) helicopter responses for the Planar Experimental Radio-Controlled Helicopter.

3.4 Input Shaping On Full-Scale Helicopters

A full-scale helicopter with a suspended load is not available for testing the effectiveness of input shaping at suppressing suspended load oscillations. A project with the Vertical Lift Consortium and the National Rotorcraft Technology Center is investigating flight control concepts and handling qualities requirements for suspended load operation. Through this project, an input-shaping MATLAB program was prepared for, and delivered to, Boeing

Helicopters, Kaman Aerospace, and Sikorsky. They will use this MATLAB program for testing and evaluation of the input-shaping technique as part of full-scale helicopter flight control systems. A thorough discussion of the input shaping MATLAB program is available in Appendix A. Appendix A explains the MATLAB syntax for using the program and includes the complete program code.

CHAPTER IV

FLIGHT CONTROLLER DESIGN COMBINING INPUT SHAPING WITH MODEL-FOLLOWING CONTROL

This chapter investigates combining input shaping with model-following control to improve helicopter performance when carrying a suspended load. First, the model-following control architecture is presented. Model-following control is commonly used in helicopter flight controllers to force specified helicopter states, such as the attitude, to respond like a desirable model [52]. There are obvious benefits in terms of designing to satisfy handling qualities requirements if the helicopter's response can be prescribed by an ideal model of the designer's choosing [52]. However, tracking of the prescribed model is significantly degraded when the helicopter is carrying a suspended load. Also, this approach cannot eliminate suspended load oscillation without using feedback of the suspended load states.

A near-hover, attitude-command model-following controller is designed for a Sikorsky S-61 helicopter with and without a suspended load using the dynamic models of this helicopter obtained from previous researchers [15, 16]. The design of the attitude-command model-following controller is demonstrated for the unloaded helicopter case, and simulation results demonstrate the controller's effectiveness. Next, this controller is applied to the loaded helicopter case, and it is shown that the model tracking performance is degraded by large load oscillations. Then, input shaping is added to the model-following controller. The results show that input-shaping reduces swing of the suspended load and greatly improves helicopter tracking of the prescribed model.

4.1 Model-Following Control

Model-following control has become an attractive control technique for helicopter flight controllers. Boeing Helicopters used a control law architecture consisting of model-following control on several demonstrator programs in the 1980's and 1990's, including the V-22 and RAH-66 [27]. It is an effective and viable control architecture because the aircraft response

can be specified via the prescribed model to achieve favorable handling qualities and is well-suited for full authority fly-by-wire control systems [27].

Model-following control for aircraft first appeared in the literature in the 1960's and 1970's. Murphy and Narendra [34] adjoined a model that approximates the pilot's inputs to the helicopter equations of motion, and then used optimal control to design a regulator that would track the model response. Winsor and Roy [66] showed used a combination of partial state feedback and feedforward compensation to achieve good tracking of the prescribed model. More recently, Trentini and Pieper [55] designed a model-following controller for a helicopter in hover to meet handling qualities requirements using an optimal control design approach. Model-following has also been used for other types of aircraft, such as NASA Ames' and the U.S. Navy's oblique wing research aircraft [1]. Furthermore, model-following control has been synthesized with other types of controllers to achieve better performance in difficult flight conditions or for specific missions, such as gust rejection during shipboard operations [21].

There are two types of model-following control. Implicit model-following uses optimal control to design a feedback controller that yields a closed-loop control system whose dynamics match the dynamics of the prescribed model [66]. The prescribed model itself does not appear in the control law directly; its output is only used in an optimal control performance index used to calculate feedback controller gains [1, 55, 57]. Explicit, or real, model-following control uses the prescribed model directly in the control system, typically as a feedforward compensator [1, 32, 52, 55]. The control law in explicit model-following is typically constructed using the feedforward of the prescribed model states and a feedback controller that uses plant-state feedback [57, 66].

Explicit model-following control is preferred over implicit model-following control for use in aircraft control systems [55]. Explicit model-following controllers incorporate real-time measurement of the model tracking error, allowing the controller to reject disturbances and to correct the aircraft trajectory in the presence of errors in the model used in the design process [66]. In addition, explicit model-following can be implemented with partial state feedback and effective model tracking can still be achieved [66]. Most importantly, the

feedback stabilization can be designed independently from the feedforward compensator and prescribed model [27]. For these reasons, explicit model-following control was selected for the baseline helicopter flight controller implemented in this chapter.

The implemented controller combines feedforward and feedback control. A state feedback controller is used to stabilize the helicopter system. Also, the state feedback control enables the error dynamics of the model tracking to be specified. Based on the recommendation of Tyler [57], the feedback controller is designed as a regulator independent from the rest of the controller using only the helicopter plant model. The regulator is designed using the pole placement state feedback technique. The feedforward controller uses model inversion techniques to cancel the undesired helicopter dynamics.

However, tracking of the prescribed model is significantly degraded when the helicopter is carrying a suspended load. One way to improve helicopter performance when carrying a suspended load would be to replace the original feedforward model with one that accounts for the sling-load dynamics. A drawback of this method is that the feedback portion of such a controller would require real-time measurement of the sling load position or angle, which is almost never available in practice. This chapter investigates an alternative way to improve system performance without measurements of the load states by adding input shaping to the controller.

4.1.1 Controller Description

Figure 24 shows the block diagram of an explicit model-following controller. The structure of this controller is similar to that presented in Landis et al. [27]. The pilot command is used as the input to a prescribed model $\mathbb{G}_M$. The rest of the controller is designed to force the output $\vec{x}$ of the helicopter plant $\mathbb{G}_P$ to track the prescribed model output $\vec{x}_M$. Generally, helicopters have one or more unstable modes, so a feedback controller is used to stabilize $\mathbb{G}_P$. Also, the feedback $\mathbb{G}_{FB}$ determines the error dynamics of the model tracking. The feedforward control $\mathbb{G}_{FF}$ is then selected to cancel undesired helicopter dynamics using model inversion techniques.

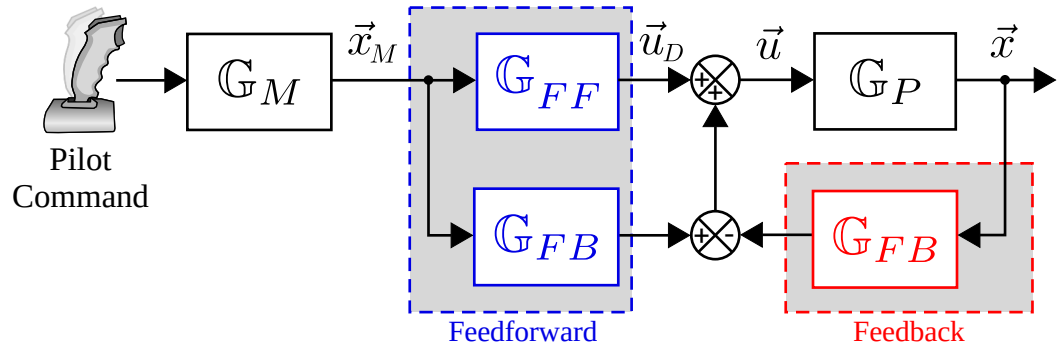


Figure 24: Block diagram of an explicit model-following control structure.

4.1.2 Theoretical Model-Following Controller Performance

4.1.2.1 Steady-State Performance

If the feedforward control $\mathbb{G}_{FF}$ equals the inverse of the helicopter plant, $\mathbb{G}_P^{-1}$, then the helicopter output $\vec{x}$ asymptotically tracks the model output $\vec{x}_M$. The feedback portion of the controller also enables the controller to reject disturbances that affect the model tracking. The asymptotic model tracking can easily be shown from the block diagram in Figure 24 using block diagram reduction. The helicopter output $\vec{x}$ is given by

$$\vec{x} = \mathbb{G}_P \vec{u} \tag{41}$$

where $\vec{u}$ is the input to the helicopter plant $\mathbb{G}_P$. From the block diagram, the control law, or the input to the plant, is given by

$$\vec{u} = \mathbb{G}_{FF}\vec{x}_M + \mathbb{G}_{FB}\vec{x}_M - \mathbb{G}_{FB}\vec{x} \tag{42}$$

Substituting (42) into (41), the output of the helicopter plant $\vec{x}$ can be expressed as

$$\vec{x} = \mathbb{G}_P \left(\mathbb{G}_{FF}\vec{x}_M + \mathbb{G}_{FB}\vec{x}_M - \mathbb{G}_{FB}\vec{x}\right) \tag{43}$$

If $\mathbb{G}_{FF}$ equals the inverse of the helicopter plant $\mathbb{G}_P^{-1}$, then (43) simplifies to

$$\vec{x} = \vec{x}_M + \mathbb{G}_P\mathbb{G}_{FB}\vec{x}_M - \mathbb{G}_P\mathbb{G}_{FB}\vec{x} \tag{44}$$

collecting terms and simplifying yields

$$\vec{x} = \vec{x}_M \tag{45}$$

Therefore, if the feedforward control cancels the helicopter dynamics, then the helicopter output tracks the prescribed model output. This can be accomplished by inverting a model of the helicopter plant. In this case, the control law is given by

$$\vec{u} = \mathbb{G}_P^{-1}\vec{x}_M + \mathbb{G}_{FB}\vec{x}_M - \mathbb{G}_{FB}\vec{x} \tag{46}$$

4.1.2.2 Model Tracking Error Dynamics

It can be shown that the design of the feedback controller determines the dynamic characteristics of the model tracking error. The helicopter plant can be expressed in state space form as

$$\dot{\vec{x}} = \mathbf{A}\vec{x} + \mathbf{B}\vec{u} \tag{47}$$

A state space representation of the prescribed model is given by

$$\dot{\vec{x}}_M = \mathbf{A}_M\vec{x}_M + \mathbf{B}_M\vec{r} \tag{48}$$

where $\vec{r}$ is the pilot command. The model tracking error $\vec{e}$ is given by

$$\vec{e} = \vec{x}_M - \vec{x} \tag{49}$$

Assuming that all of the helicopter states are measurable, full state feedback is selected for the feedback controller. Full state feedback calls for feedback control of the form $\vec{u} = -\mathbf{K}\vec{x}$, where $\mathbf{K}$ is a constant state feedback gain matrix. Therefore, $\mathbb{G}_{FB}$ from Figure 24 equals the gain $\mathbf{K}$. Using state feedback for the feedback controller with constant gain matrix $\mathbf{K}$, the control law in (46) can be written as

$$\vec{u} = \vec{u}_D + \mathbf{K}\vec{x}_M - \mathbf{K}\vec{x} \tag{50}$$

where $\vec{u}_D$ is the output of the feedforward block and its corresponding signal is labeled in Figure 24. Recall that the feedforward block is chosen to be an inverted model of the helicopter plant. A state space representation of the helicopter plant model to be inverted is given by

$$\dot{\vec{x}}_D = \mathbf{A}_D\vec{x}_D + \mathbf{B}_D\vec{u}_D \tag{51}$$

In practice, this model will not perfectly represent the helicopter. An expression for $\vec{u}_D$ is obtained by solving (51) for $\vec{u}_D$, or

$$\vec{u}_D = \mathbf{B}_D^{\dagger}\left(\dot{\vec{x}}_D - \mathbf{A}_D\vec{x}_D\right) \tag{52}$$

where $\mathbf{B}_D^{\dagger}$ is the pseudoinverse of $\mathbf{B}_D$ and $\vec{x}_D$ is the input to the inverse, which equals the prescribed model output $\vec{x}_M$ as can be seen in Figure 24. The pseudoinverse is required because $\mathbf{B}$ is usually not a square matrix.

Substituting the plant inverse given in (52) into (50), and setting $\dot{\vec{x}}_D = \dot{\vec{x}}_M$ and $\vec{x}_D = \vec{x}_M$, the control law can be written as

$$\vec{u} = \mathbf{B}_D^{\dagger}\left(\dot{\vec{x}}_M - \mathbf{A}_D\vec{x}_M\right) + \mathbf{K}\vec{x}_M - \mathbf{K}\vec{x} \tag{53}$$

To analyze the dynamics of the model tracking error, an expression for the derivative of $\vec{e}$ must be found. Taking the derivative of (49) gives

$$\dot{\vec{e}} = \dot{\vec{x}}_M - \dot{\vec{x}} \tag{54}$$

Substituting (48) and (47) for $\dot{\vec{x}}_M$ and $\dot{\vec{x}}$ in (54) gives

$$\dot{\vec{e}} = \mathbf{A}_M\vec{x}_M + \mathbf{B}_M\vec{r} - \mathbf{A}\vec{x} - \mathbf{B}\vec{u} \tag{55}$$

Substituting (53) into (55) yields

$$\dot{\vec{e}} = \mathbf{A}_M\vec{x}_M + \mathbf{B}_M\vec{r} - \mathbf{A}\vec{x} - \mathbf{B}\left(\mathbf{B}_D^{\dagger}\left(\dot{\vec{x}}_M - \mathbf{A}_D\vec{x}_M\right) + \mathbf{K}\vec{x}_M - \mathbf{K}\vec{x}\right) \tag{56}$$

If the model inversion is exact, or equivalently $\mathbf{A}_D = \mathbf{A}$ and $\mathbf{B}_D = \mathbf{B}$, then (56) simplifies to

$$\dot{\vec{e}} = \mathbf{A}_M\vec{x}_M + \mathbf{B}_M\vec{r} - \dot{\vec{x}}_M + (\mathbf{A} - \mathbf{B}\mathbf{K})(\vec{x}_M - \vec{x}) \tag{57}$$

Recognizing that $\vec{x}_M - \vec{x} = \vec{e}$ and $\dot{\vec{x}}_M = \mathbf{A}_M\vec{x}_M + \mathbf{B}_M\vec{r}$, (57) reduces to

$$\dot{\vec{e}} = (\mathbf{A} - \mathbf{B}\mathbf{K})\,\vec{e} \tag{58}$$

This result shows that the eigenvalues of $(\mathbf{A} - \mathbf{B}\mathbf{K})$ determine the dynamics of the model tracking error, $\vec{e} = \vec{x}_M - \vec{x}$, and designing a controller for the model tracking error reduces to the regulator design problem as discussed by Tyler [57]. Therefore, using techniques such as pole placement to calculate a state feedback gain $\mathbf{K}$ allows the designer to specify the model tracking error dynamics.

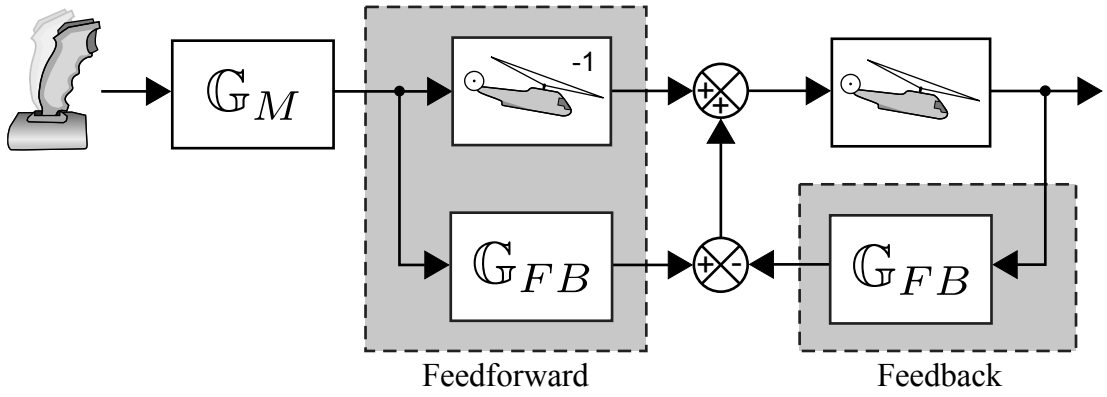


Figure 25: Model-following controller for the unloaded Sikorsky S-61 helicopter.

4.2 Near-Hover, Attitude-Command Model-Following Controller for the Sikorsky S-61 Helicopter

A near-hover, attitude-command model-following controller for the unloaded Sikorsky S-61 helicopter was designed using the Sikorsky S-61 model from Hall and Bryson [16]. The characteristics of this model were discussed in Chapter II. The model was used for designing the feedback controller and for the model inversion part of the feedforward controller. The block diagram of this controller is shown in Figure 25. This is a more specific version of the generic model-following block diagram shown in Figure 24, where the unloaded helicopter model is used for the model inversion and for simulating the helicopter response. Note that the inversion of the unloaded model leads to cancellation of the dynamics of the unloaded helicopter plant.

However, when the helicopter carries a suspended load, the controller will no longer completely cancel the dynamics of the plant, and error between the actual output and the prescribed model will develop. The block diagram for this scenario is shown in Figure 26. Note that the loaded helicopter is simulated using the model of the load-carrying Sikorsky S-61 from Gupta and Bryson [15] and discussed in Chapter II. With the plant helicopter carrying a suspended load, the model inversion will cancel only the baseline helicopter dynamics but not the additional dynamics introduced by the presence of the suspended load.

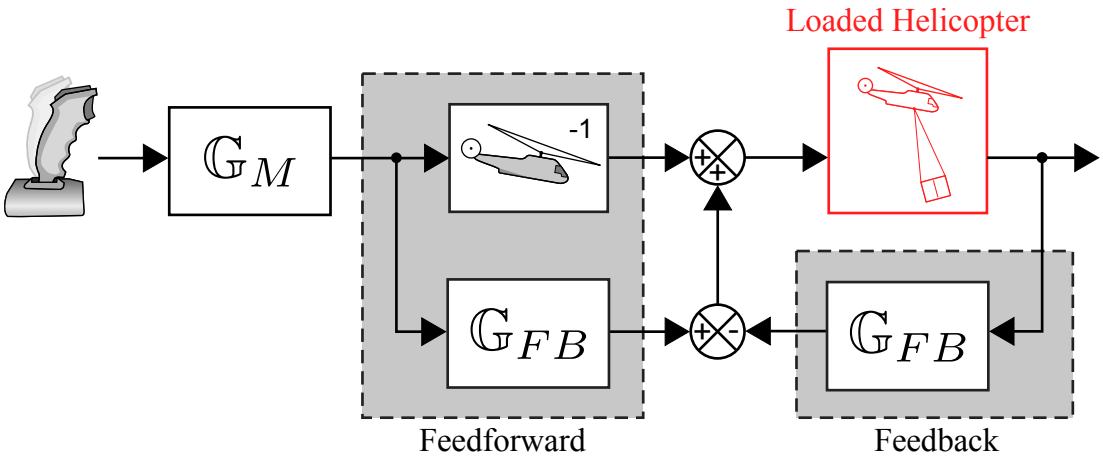


Figure 26: Model-following controller for the load-carrying Sikorsky S-61 helicopter.

4.2.1 Feedback Controller Design Using Unloaded S-61 Model

According to the analysis above and (58), the dynamics of the model tracking error are dependent on the eigenvalues of $\mathbf{A}-\mathbf{BK}$. The model tracking performance can be specified by selecting eigenvalues that give suitable model tracking error dynamics, and then the pole placement technique may be used to calculate $\mathbf{K}$. This feedback controller could be designed using other techniques, but the pole placement technique proves simple and effective.

The gain matrix $\mathbf{K}$ was calculated with the MATLAB *place* function using the helicopter state matrix $\mathbf{A}$ and input matrix $\mathbf{B}$ and the selected poles. Using the unloaded Sikorsky S-61 $\mathbf{A}$ and $\mathbf{B}$ matrices give a gain matrix of

$$\mathbf{K} = \begin{bmatrix} 22.60 & 2.735 & -1.523 & 1.464 & -0.1871 & 0.7717 \\ 1.339 & 0.1425 & -0.0536 & 2.083 & 0.2455 & 0.1808 \end{bmatrix} \tag{59}$$

4.2.2 Prescribed Models

The prescribed model $\mathbb{G}_M$ determines how the helicopter responds to pilot commands. A model that yields the desired helicopter performance must be selected. However, the model should not be too aggressive. An overly-aggressive model will lead to plant commands that the helicopter actuators cannot reproduce. When the actuators cannot faithfully reproduce the commanded inputs, the helicopter will not follow the trajectory specified by the prescribed model.

For the Sikorsky S-61 model-following controller, a third-order model was selected that prescribes the helicopter attitude and attitude rate. This third-order model is a combination of an underdamped second-order model and a first-order lag. The first-order lag is included to account for actuator dynamics, and effectively smooths the plant command, ensuring that the command is something that the helicopter rotor can reproduce. This was accomplished with a relatively fast time constant for the first-order model. The majority of the dynamics at the time scale of gross helicopter translation and attitude response is prescribed by the second-order underdamped model.

The state-space model of the third-order pitch and roll model is given by

$$\dot{\vec{x}}_M = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\omega_n^2\tau & -\left(\omega_n^2 + 2\zeta\omega_n\tau\right) & -\left(2\zeta\omega_n + \tau\right) \end{bmatrix} \vec{x}_M + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \vec{r} \tag{60}$$

$$\vec{y}_M = \begin{bmatrix} \omega_n^2\tau & 0 & 0 \\ 0 & \omega_n^2\tau & 0 \end{bmatrix} \vec{x}_M$$

where τ is the time constant of the first-order lag, ω_n is the natural frequency of the second-order model, and ζ is the damping ratio of the second-order model. The pilot's attitude command (a pitch or roll angle) is the input r to this model. Note that this model was designed to have a steady-state gain of one, though this can be changed depending on the application and the source of the pilot's attitude command. Specifying the prescribed model requires selecting values for the parameters τ, ω_n, and ζ.

The model in (60) is applied to the pitch and roll attitude commands. There are two of these third-order models in the control system, one each for the longitudinal and lateral channels. Each model is applied independently to the pilot's attitude command in that channel. The same first-order time constant, natural frequency, and damping ratio are used in the two models to give similar characteristics to the pitch and roll attitude responses.

The parameters of the two third-order models were selected to have a damping ratio of 0.707 because it yields a good balance between fast rise time and low overshoot. A settling time t_s of 2.5 seconds was selected. Using this damping ratio and settling time, the natural

frequency was found using the following approximation

$$t_s = \frac{4}{\omega_n \zeta} \tag{61}$$

Solving (61) with the selected damping ratio and settling time resulted in a corresponding natural frequency of 2.26 rad/s.

A first-order time constant of 0.125 seconds was selected. This time constant was slow enough to ensure that the helicopter rotor could respond to the commanded rotor tilt angles, while not significantly altering the performance characteristics prescribed by the second-order model.

The pitch and roll prescribed models are solved in real time by the controller, and the outputs of each model, $\vec{y}_M$, are composed of the desired attitude trajectory and the corresponding attitude rate trajectory. With models in both the longitudinal and lateral channels, the result is pitch attitude, pitch rate, roll attitude, and roll rate trajectories prescribed by the models. Because the helicopter velocities are also needed to form the full state vector for inverting the unloaded model, the measured velocities of the plant are used. The attitude trajectories are combined with the measured longitudinal and lateral velocities to form the complete helicopter state vector. This state vector can be interpreted as the desired state trajectory.

4.2.3 Controller Implementation and Simulation

This controller was tested in simulation using the unloaded and loaded Sikorsky S-61 models from [16] and [15]. The controller and models were programed in MATLAB and Simulink to perform the simulation. Simulated pilot pitch and roll attitude commands were used that moved the helicopter from one position to another, starting and stopping in hover. A fairly conservative maneuver was selected to ensure that the helicopter response does not exceed the near-hover regime of the unloaded and loaded models.

4.2.4 Simulation Results and Controller Performance

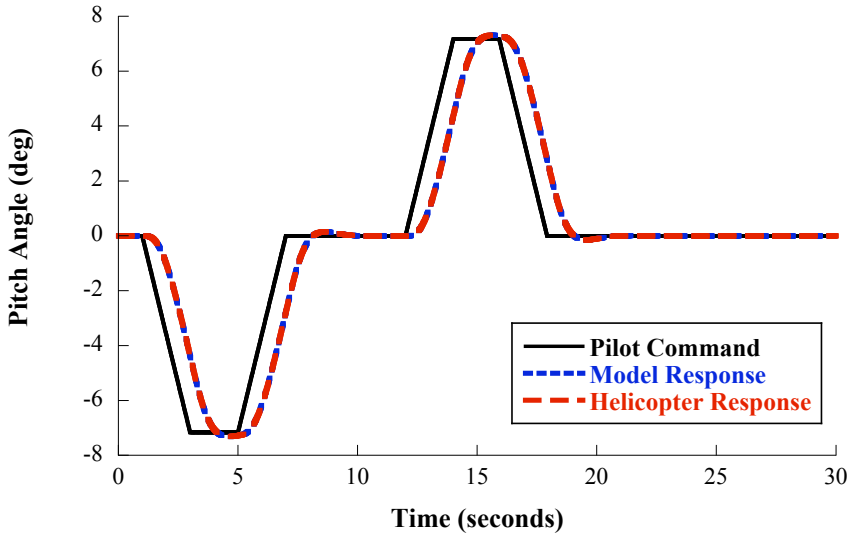
4.2.4.1 Unloaded S-61 Performance

To evaluate the performance of this controller for the unloaded Sikorsky S-61 helicopter, a pitch and roll maneuver was simulated. The pilot attitude command was selected to move the helicopter in both the longitudinal and lateral directions to a target location, starting and stopping in hover. The attitude command has a maximum amplitude of 7.16 degrees, and this can be considered a small-to-moderate amplitude command.

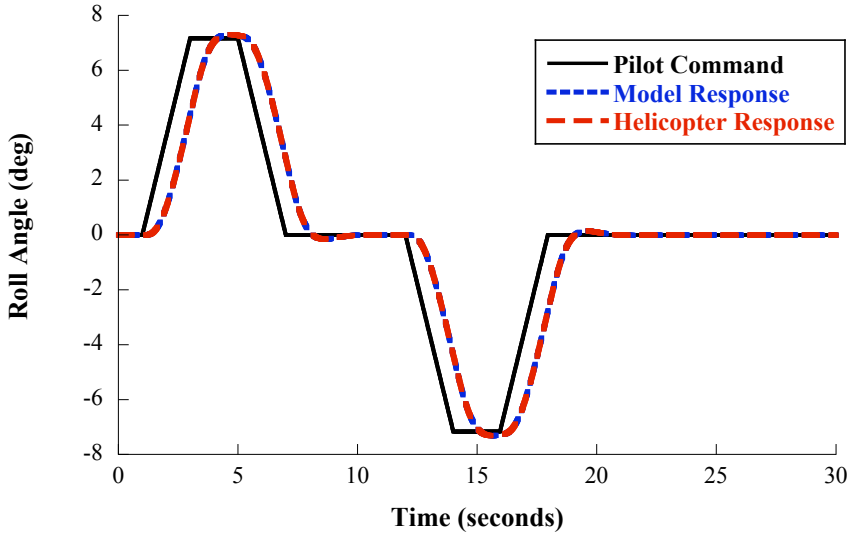
Figure 27a shows the pilot command, the resulting response of the prescribed model, and the helicopter attitude response for the pitch channel. The pitch attitude tracks the prescribed model almost perfectly. Figure 27b shows the pilot command, the resulting response of the prescribed model, and the helicopter attitude response for the roll channel. As in the pitch channel, the roll attitude tracks the prescribed model almost perfectly. The controller also minimizes the coupling between the longitudinal and lateral motion of the helicopter.

If the control effort required to execute this maneuver is too fast or has too large of an amplitude, then the quasistatic rotor assumption used in the modeling process may be inadequate for approximating the rotor dynamics. In other words, the commanded rotor disk angle may require the helicopter rotor to tilt faster than it can. Due to the larger inertia of the helicopter about the pitch axis, it is expected that the command in the longitudinal channel will need to be more aggressive and, therefore, more likely to exceed the limits of the rotor.

To investigate this, the command sent to the helicopter plant in the longitudinal channel is shown in Figure 28a. Recall that the commands for the Sikorsky S-61 models are the rotor tilt, or disk, angles. At this point, it is worth noting that the response of the rotor to control inputs occurs on the same time scale as the rotor rotation rate. The time to half amplitude of the rotor tilt is on the order of 0.05 seconds [23], or similarly, the time constant of the rotor response is typically a quarter to a half of the rotor rotation period [38]. Half of the rotor rotation period for the Sikorsky S-61 main rotor is approximately 0.15 seconds [16]. The longitudinal command generated by the controller requires a rotor



(a) Pitch pilot command.

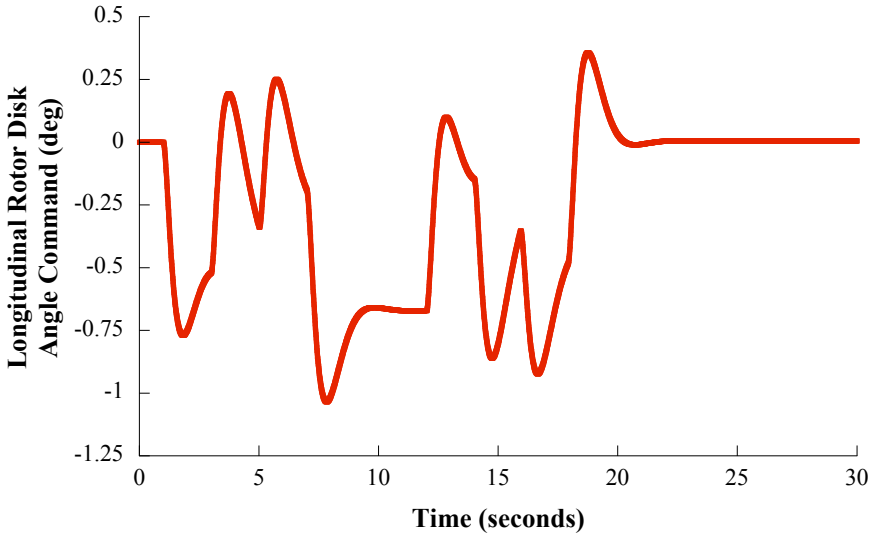


(b) Roll pilot command.

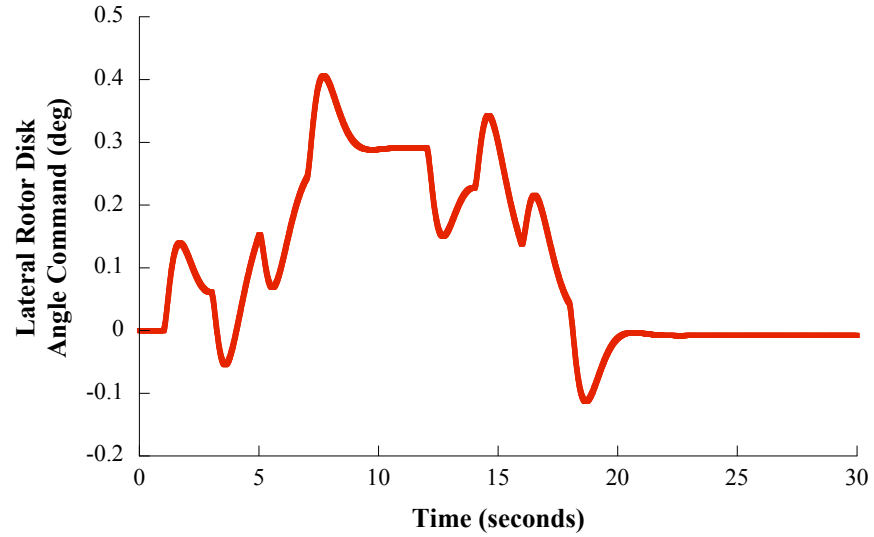
Figure 27: Pilot command, prescribed model response, and helicopter attitude response in the (a) pitch and (b) roll channels and for the unloaded Sikorsky S-61.

response with a time to half amplitude of approximately 0.4 seconds, which is almost an order of magnitude slower than the rotor capability. Also, the maximum amplitude reached by the command is only 1 degree, which is a small rotor tilt angle.

The lateral command is shown in Figure 28b. As can be seen by comparing Figure 28b with Figure 28a, the longitudinal command is more aggressive. As expected due to the



(a) Longitudinal command.



(b) Lateral command.

Figure 28: Control effort required by the model-following controller in the (a) longitudinal and (b) lateral channels for the unloaded Sikorsky S-61.

larger inertia of the helicopter about the pitch axis, the longitudinal command has a larger magnitude and changes more rapidly than the lateral command.

4.2.4.2 Loaded S-61 Performance

The performance of the load-carrying Sikorsky S-61 helicopter was investigated using the same controller and pilot commands as for the unloaded helicopter in the previous section.

For feedback, the controller still uses only the unloaded helicopter states (pitch and roll attitudes and attitude rates, and longitudinal and lateral translational velocities). Measurements of the load states are not required.

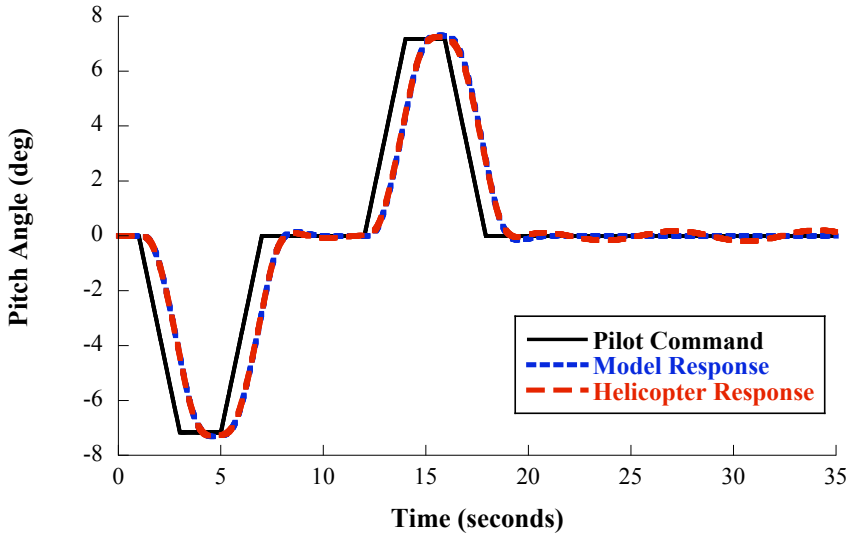
Figure 29 shows the pilot command, the resulting response of the prescribed model, and the helicopter attitude response for the pitch and roll channels. The response in the pitch channel is shown in Figure 29a. The pitch attitude tracking of the prescribed model is not significantly affected by the addition of a suspended load. This is due to the larger pitch inertia of the helicopter resisting the moment applied by the swinging suspended load.

The response in the roll channel is shown in Figure 29b. The roll attitude does not track the model response very well, particularly following the completion of the command. Following the command, there are residual roll attitude oscillations that have an amplitude of nearly 2 degrees. These oscillations are caused by the tension in the suspension cable supporting the load applying an oscillatory torque about the helicopter center of gravity as the load swings. In this sense, the load is acting like a disturbance applied to the helicopter.

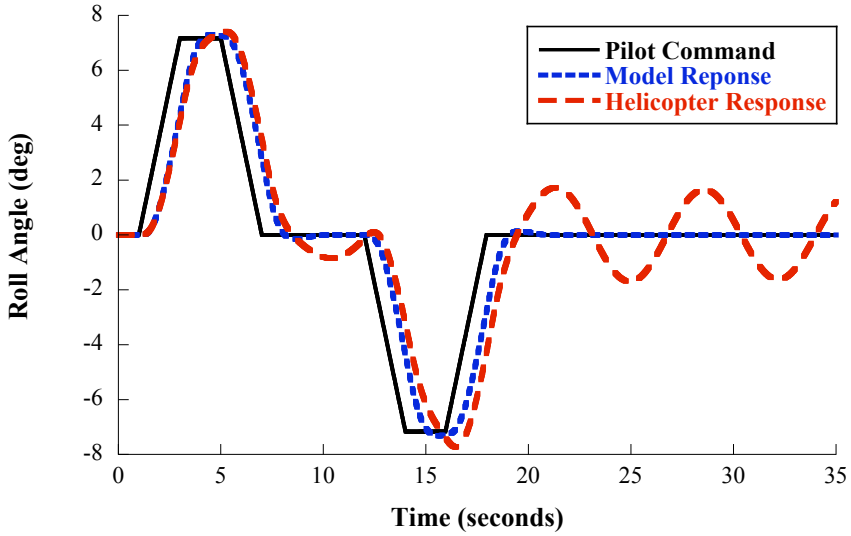
According to ADS-33E requirements [59], residual attitude oscillations larger than 0.5 degrees are considered excessive for any type of maneuver. As seen in Figure 29a, the pitch attitude residual oscillation amplitude is less than 0.5 degrees, and remains within ADS-33E requirements. However, the residual roll attitude oscillation amplitude is nearly 2 degrees, so it does not satisfy ADS-33E requirements.

Figure 30 shows the longitudinal and lateral load oscillation caused by the helicopter motion. The 65.6 ft suspension cable leads to a large-amplitude, low-frequency response of the load. The peak-to-peak amplitude of the load oscillation in both directions is nearly 55 ft. This amount of load swing would be dangerous and hard for the pilot to control following the maneuver. It would also increase the time it takes for the pilot to set down the load, as he or she would have to wait for the load oscillation to damp out or actively move the helicopter to cancel out the swing before setting the load down.

The amount of residual attitude and load oscillation is dependent on properties of the pilot's commands, such as move distance and velocity. This means the amount of residual oscillation will vary depending on the pilot's commands, making the oscillations hard for



(a) Pitch channel.

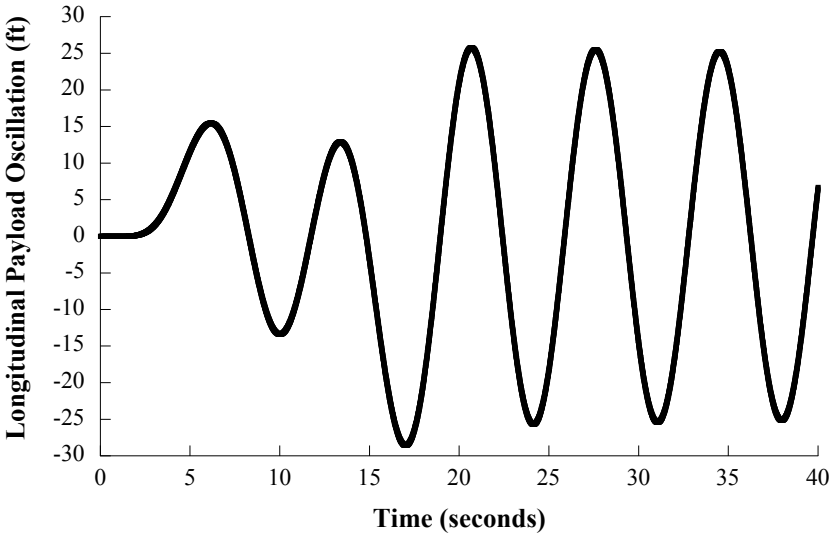


(b) Roll channel.

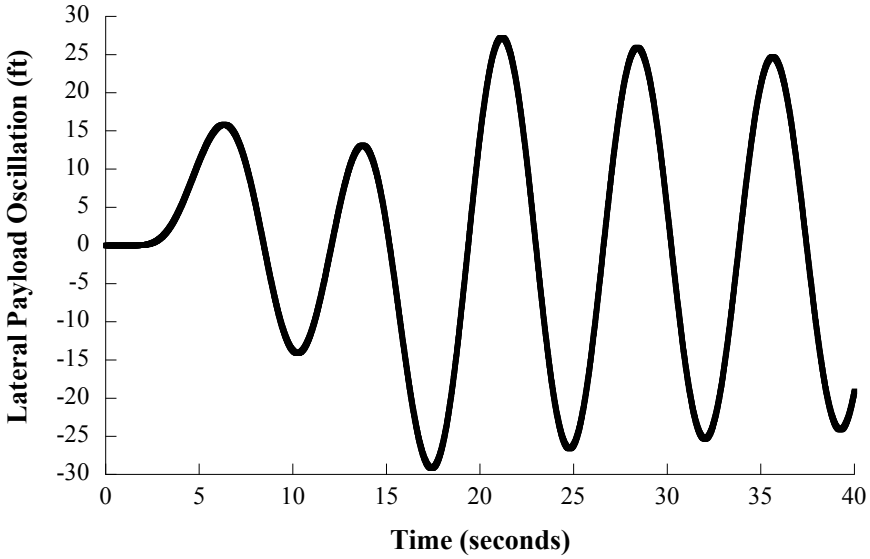
Figure 29: Pilot command, prescribed model response, and helicopter attitude response in the (a) pitch and (b) roll channels for the loaded Sikorsky S-61.

the pilot to predict and thus difficult to control.

The model-following controller should be modified to improve tracking of the prescribed model and to reduce the suspended load oscillation. These modifications should make the helicopter response more predictable to the pilot and improve the speed and safety of load transfer operations.



(a) Longitudinal load swing.



(b) Lateral load swing.

Figure 30: Suspended load swing in the (a) longitudinal and (b) lateral directions.

4.3 Combining Input Shaping and Model-Following Control

The model-following controller does not address the load oscillation nor its disturbance-like effects on the model tracking error. As was demonstrated in the previous section, oscillation of the load is dangerous and adversely affects control of the helicopter. To reduce the load oscillation, input shaping is added to the model-following controller. The model-following controller block diagram with input shaping is shown in Figure 31. Input shaping is added

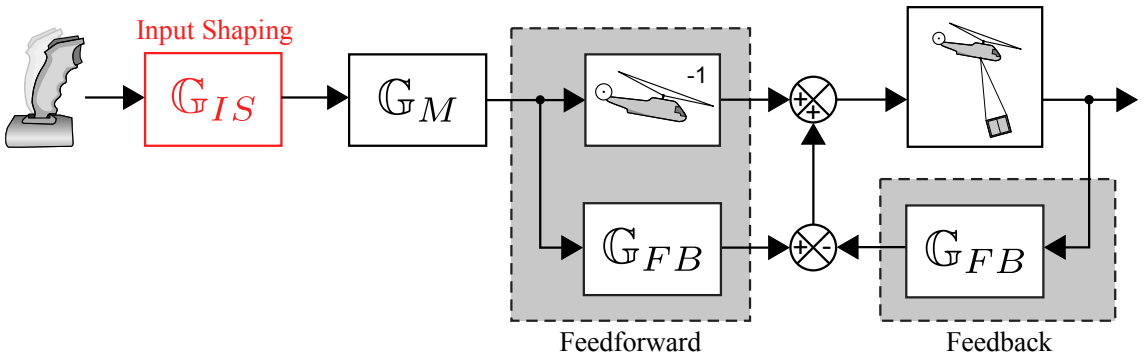


Figure 31: Model-following controller for the load-carrying Sikorsky S-61 helicopter with input shaping added.

to the controller before the prescribed model.

This implementation approach is equivalent to incorporating an input shaper in the prescribed model. Effectively, the input shaping is built into the prescribed model, and the model output is an input-shaped that does not excite suspended load swing. The order of the input shaper and the prescribed model in the controller does not matter because input shapers and the chosen prescribed model are linear.

4.3.1 Input Shaper Design

To design an input shaper that suppresses the load oscillation in the longitudinal and lateral directions, the natural frequencies and damping ratios of the oscillation in those directions are required. The frequencies and damping ratios of the longitudinal and lateral load oscillations shown in Figure 30 were determined using the log decrement method. The longitudinal load oscillation has a frequency of 0.907 rad/s and a damping ratio of 0.001. The lateral load oscillation has a frequency of 0.869 rad/s and a damping ratio of 0.007. The frequencies are slightly different, and the damping ratio of the lateral oscillation is slightly larger. The damping is larger because more energy is being transferred to or exchanged with the helicopter's roll attitude oscillation.

It is worth noting here that the natural frequency and damping ratio are also partially influenced by the feedback part of the controller. The feedback controller is trying to drive the model tracking error to zero. However, the load acts like a disturbance on the helicopter motion, as can be seen in Figure 29, because the load is coupled with the helicopter attitude

and translation. This disturbance introduces an error between the trajectory specified by the prescribed model and the actual helicopter states that the feedback controller attempts to eliminate. The feedback controller affects the load oscillation as the controller attempts to eliminate the model tracking error. It does so by adjusting the commands sent to the helicopter plant and therefore affects the motion of the helicopter and the load. This means the poles used to design the feedback controller and calculate the gain matrix **K** end up having a small effect on the motion of the load and its swing frequencies and damping ratio.

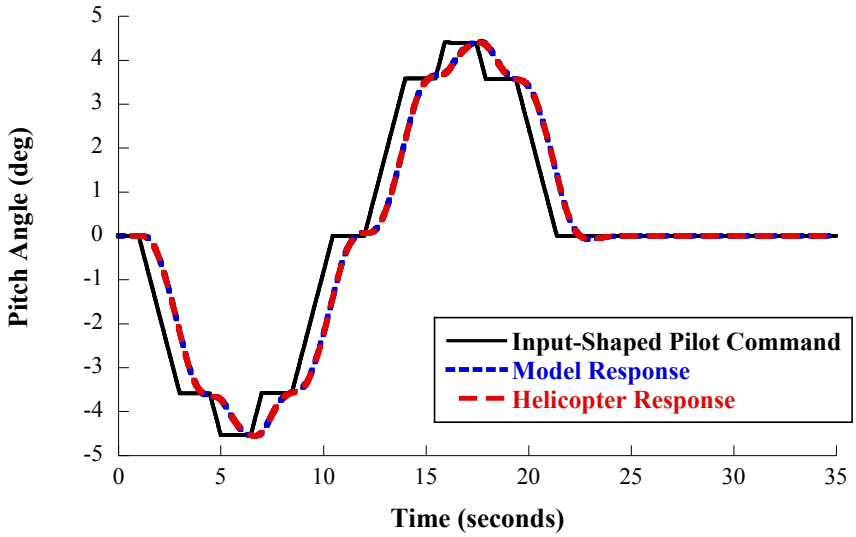
ZV shaping was selected for use in this model-following controller. The shaper amplitudes and times were calculated from the natural frequencies and damping ratios. A separate shaper was used for the longitudinal and lateral load oscillations because their frequencies were somewhat different.

4.3.2 Simulation Results and Controller Performance with Input Shaping

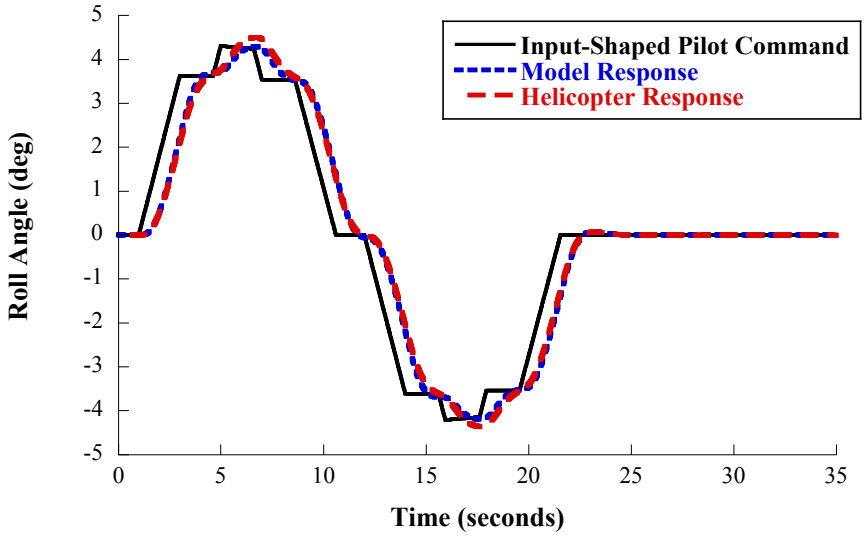
Figure 32 shows the input-shaped pilot command, the resulting response of the prescribed model, and the helicopter attitude response for the pitch and roll channels with input shaping added to the model-following controller. The response in the pitch channel is shown in Figure 32a. The pitch attitude tracking of the prescribed model is not as significantly affected as the roll channel by the addition of a suspended load as shown in Figure 29, but adding input shaping to the controller improves the tracking by reducing the small residual pitch attitude oscillations.

The real benefit, in terms of the attitude response, of adding input shaping to the controller is in the roll channel. The response in the roll channel is shown in Figure 32b. The roll attitude tracking of the model response is greatly improved with input shaping added to the controller. The amplitude of the roll attitude oscillations following the command has been greatly reduced by adding input shaping to the controller, and the ADS-33E requirements for residual attitude oscillation amplitude are satisfied. With the residual attitude oscillations reduced, the addition of input shaping to the controller should make the response of the helicopter more predictable to the pilot.

Figure 33 shows the longitudinal and lateral load oscillation caused by the helicopter



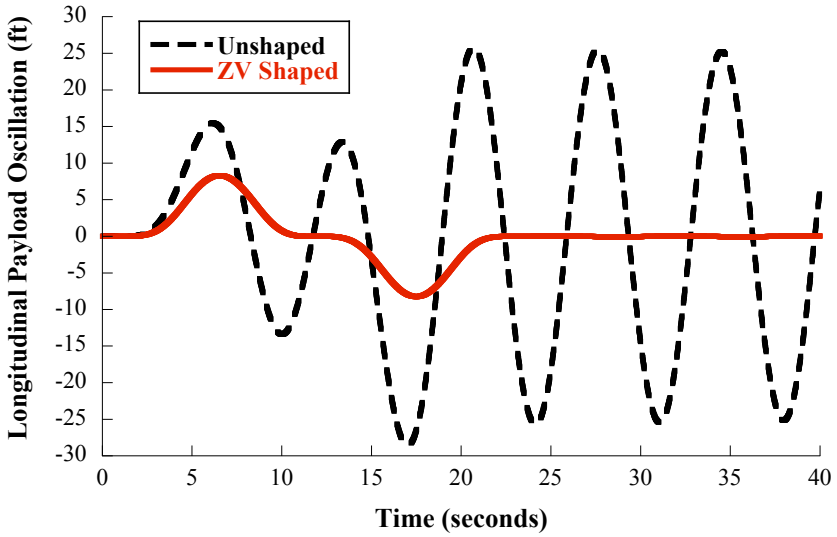
(a) Pitch channel.



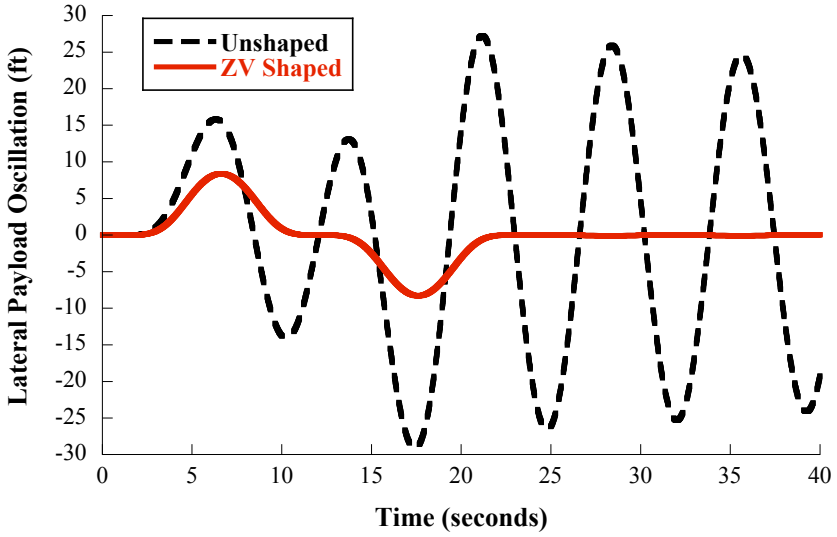
(b) Roll channel.

Figure 32: Input-Shaped pilot command, prescribed model response, and helicopter attitude response in the (a) pitch and (b) roll channels for the loaded Sikorsky S-61.

motion with and without input shaping. Input shaping significantly reduces the residual load oscillation in both the longitudinal and lateral directions. Note that there is a still some transient swing that occurs during the command. Input shapers are only designed to suppress residual oscillation, but they have the added benefit of greatly reducing transient swing as well.



(a) Unshaped and ZV-shaped longitudinal load swing.



(b) Unshaped and shaped lateral load swing.

Figure 33: Unshaped and ZV-shaped suspended load swing in the (a) longitudinal and (b) lateral directions.

The effectiveness of input shaping is even more clear when looking at the load oscillation in two dimensions. Figure 34 shows the two-dimensional load oscillation. The oscillation is measured relative to the position of the helicopter, so it is what the pilot would see when looking down on the load. The oscillation is significantly reduced by input shaping. The small amount of swing that occurs in the ZV-shaped case is the transient oscillation. This

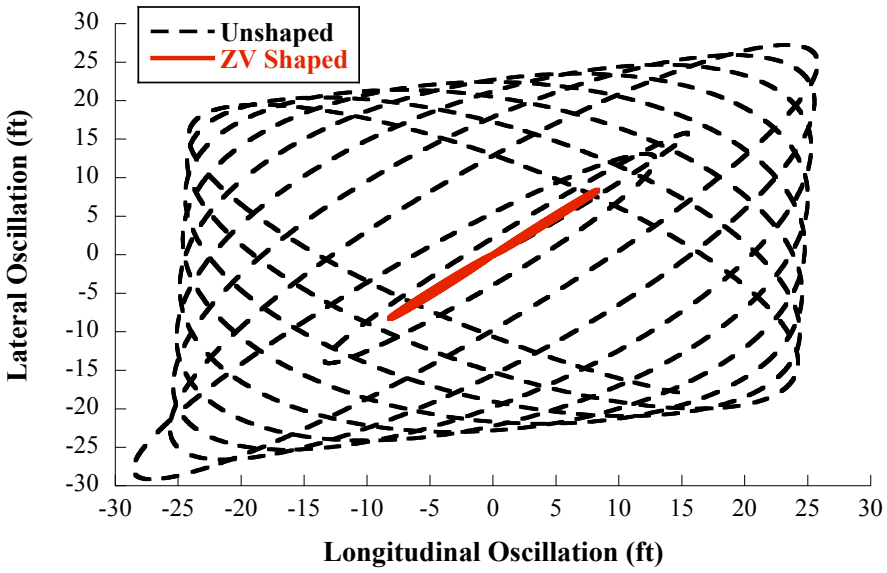


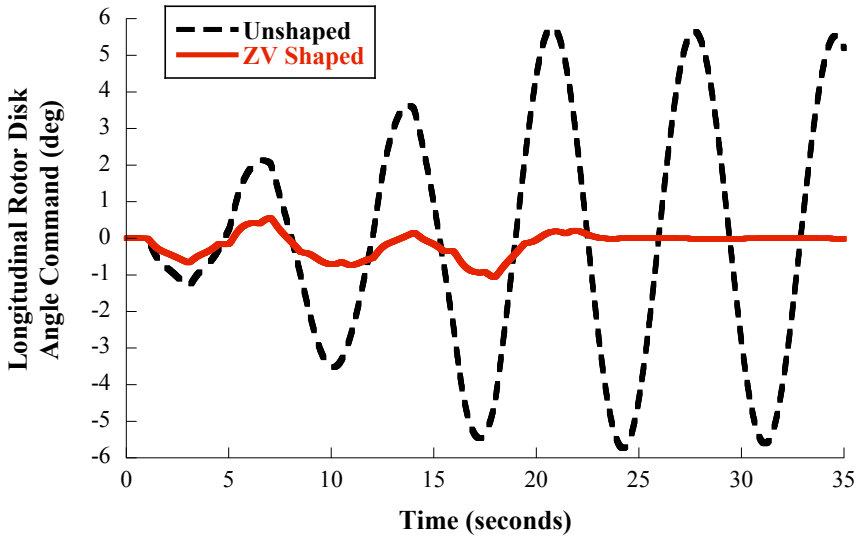
Figure 34: Unshaped and ZV-shaped two-dimensional load oscillation.

plot does not give a sense of time, but 80 seconds of data is shown in the plot. Because input shaping significantly reduces the residual load oscillation, it should decrease the time it takes for the pilot to transfer a load from one location to another.

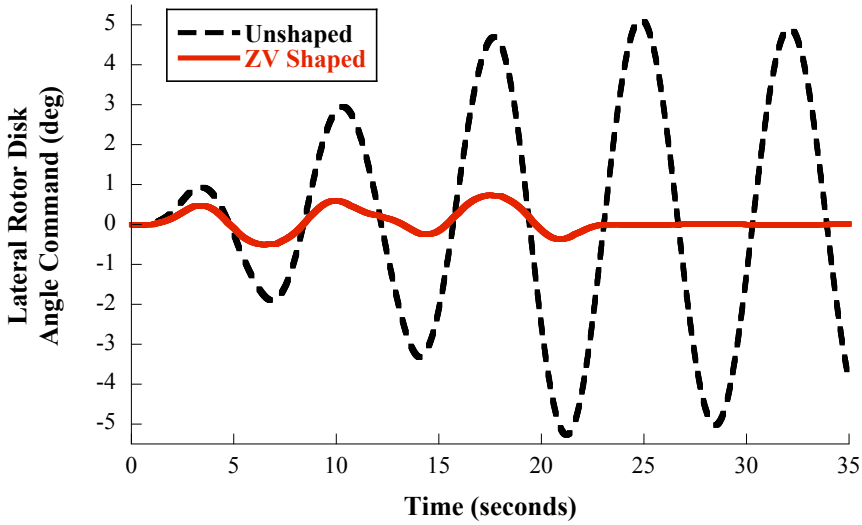
As with the controller for the unloaded helicopter, the control effort required to execute the maneuver should be examined. The command sent to the helicopter plant in the longitudinal and lateral channels is shown in Figure 35 with and without input shaping. Figure 35a shows the longitudinal command, and Figure 35b shows the lateral command. When there is no shaping, the command is oscillatory and has a larger amplitude compared to the case with input shaping. This is the case for both the longitudinal and the lateral command.

As discussed previously, the swing of the load acts like a disturbance applied to the helicopter. This causes error in the model tracking that the feedback controller tries to correct. This corrective action causes the command to have a larger amplitude and to be oscillatory. On the other hand, input shaping proactively suppresses the load swing before it disturbs the helicopter, causing no need for extra corrective action from the controller.

While model-following control has been suggested for use in helicopter flight controllers



(a) Longitudinal command.



(b) Lateral command.

Figure 35: Control effort required by the model-following controller in the (a) longitudinal and (b) lateral channels for the unloaded Sikorsky S-61.

since the 1960's, the implementation presented here for suspended-load control is unique. A model-following controller designed for an unloaded helicopter was combined with an input shaper designed to suppress the suspended load oscillation in the longitudinal and lateral directions. The results show that the combined controller suppresses oscillations of a suspended load and improves tracking of the prescribed model.

This approach works and is effective because the model inversion part of the model-following controller eliminates the undesirable dynamics of the helicopter, while the input shaper suppresses suspended load oscillation. This allows the model-following control design to be accomplished using a model of just the helicopter, while the input shaping is used to suppress the load oscillation.

Suppressing suspended load oscillation also improves the helicopter attitude response because the effects of the load on the helicopter are reduced when the load swing is reduced. As a result, the attitude better tracks the prescribed model response. Another benefit of this controller is that real-time measurement or estimation of the suspended load states is not required. The approach presented here simply requires an estimate of the load oscillation frequency and damping ratio.

It should be easier for the pilot to fly the helicopter with input shaping added to the model-following controller for a number of reasons. The response to attitude commands becomes more predictable in the sense that the pilot does not need to account for the slow-period and large amplitude load oscillations and their effects on the helicopter. In addition, the load oscillation following the command will be greatly reduced and the load will be positioned directly below the helicopter. This should make it easier, safer, and quicker for the pilot to transfer a suspended load from one location to another.

CHAPTER V

SUMMARY AND FUTURE WORK

Helicopters are often used as flying cranes to transport supplies and equipment to remote areas. Unfortunately, when a load is carried via suspension cables below a helicopter, the load oscillates in response to helicopter motion and disturbance forces such as wind. This oscillation is dangerous and adversely affects control of the helicopter, especially when carrying heavier loads. Keeping the swing at a low amplitude should allow the pilot to transfer loads more quickly and safely and improve control of the helicopter.

In this thesis, the effectiveness of input shaping at suppressing suspended load oscillation caused by helicopter motion was investigated. Designing an input shaper for this purpose requires an estimate of the natural frequency and damping ratio of the suspended load oscillation. Adequate estimates can be obtained from relatively simple models. As shown in Chapters III and IV, using input shaping to modify commands sent to the helicopter reduces oscillation of the suspended load.

In order to better understand the dynamics of suspended loads and their effects on the helicopter, several dynamic models of a helicopter carrying a suspended load were developed or discussed in Chapter II. The simplest model is that of a two-dimensional planar crane. This model was experimentally verified on a radio-controlled helicopter. Even this simple model was found to give good enough estimates of the natural frequency and damping to design input shapers that suppress the suspended load oscillation.

A linearized model of a near-hover Sikorsky S-61 with and without a suspended load from previous researchers was also examined in Chapter II. The flight modes of both configurations were identified. This model was used to design and simulate an attitude-command model-following controller in Chapter IV. The controller was first applied to the unloaded Sikorsky S-61, and simulation results were shown for example lateral and longitudinal repositioning movements. The same controller was then applied to the model of a Sikorsky S-61

carrying a suspended load. Simulation results showed that the performance of the controller degraded and that the load oscillation had a large amplitude and low period, suggesting that the helicopter and load would be difficult for a pilot to control.

Input shaping was then added to the model-following controller. The results showed that applying input shaping to the simulated pilot commands greatly reduced oscillation of the suspended load. By proactively eliminating load swing using input shaping, the helicopter response itself also improved because the helicopter and load are coupled. This analysis showed that input shaping can be combined with a helicopter flight control system to suppress suspended load oscillation and improve helicopter performance.

The combination of input shaping and a model-following controller should make it easier for the pilot to fly the helicopter when carrying a suspended load. The helicopter's response to pilot commands becomes more predictable because the pilot does not need to account for the slow-period and large-amplitude load oscillations and their effects on the helicopter. The residual load oscillations are greatly reduced and the load is positioned directly below the helicopter, once the helicopter stops. This should make transferring a suspended load from one location to another safer, faster, and easier for the pilot.

5.1 Future Work

In terms of helicopter performance, the best way to quantify the improvement offered by input shaping over other control systems would be to analyze input shaping in terms of handling qualities requirements, such as those in ADS-33E [59].

As was shown in Chapter IV, the residual oscillation in the roll attitude response when carrying a suspended load had an amplitude that exceeded the requirements in ADS-33E. Adding input-shaping to the model-following controller reduced the attitude oscillations to below the required amplitude.

However, input shaping may have a negative impact on other handling qualities, particularly those related to bandwidth. It remains to be studied whether or not the bandwidth criteria accurately accounts for the performance benefits gained by using input shaping to

suppress suspended load oscillation. There may be better ways of quantifying the performance requirements for suspended load oscillations. One such investigation by Hoh et al. [18] concluded that pilots' opinions on helicopter handling and controllability were correlated with the quality of the helicopter translational response, and the attitude requirements of ADS-33E do not accurately correlate with pilot rating of the controllability of the helicopter. In light of these advancements, the handling qualities requirements for suspended load operations may need to be further refined, particularly when analyzing the performance benefits offered by input shaping.

Input shaping should also be studied with real pilots in flight simulators and, once it is shown to be safe in the simulators, on actual helicopters carrying suspended loads. This would allow the pilots to evaluate the effectiveness of input shaping in terms of the controllability of the helicopter, and rating scales such as the Cooper-Harper scale may be used to assess the handling qualities.

The potential for input shaping to cause pilot-induced oscillations should also be studied. It should be noted that the oscillations of an uncontrolled suspended load can result in back-driving of the aircraft, and this back-driving can also result in pilot-induced oscillations [53]. If there is cause for concern about input shaping causing pilot-induced oscillations, then research should be done to investigate whether the potential for pilot-induced oscillations caused by input shaping is worse than the potential for those caused by an uncontrolled suspended load back-driving the helicopter.

Further studies should be done to better understand the effects of cable length and load-mass ratio on the damping ratio of the suspended load oscillation. While an approximate expression was found for natural frequency, a similar expression or correlation for the damping ratio would further assist in designing input shapers for suppressing suspended load swing. These parameter studies should also be accompanied by studies of the effectiveness of more robust shapers for suppressing oscillation for a larger range of cable lengths and payload types.

The model-following controller with input shaping should also be simulated with more complex models of helicopters carrying suspended loads to further verify its effectiveness.

Models that incorporate rotor dynamics and effects such as aerodynamic drag on the payload should be used. Furthermore, the combination of input shaping and a model-following controller may be implemented on an experimental radio-controlled helicopter to verify its effectiveness.

While input shaping is effective at reducing suspended load oscillation caused by helicopter motion, it does not help with oscillation caused by wind or other external disturbances. To also eliminate disturbance-induced oscillations, input shaping may be combined with a feedback controller that reacts to load disturbances. Such a feedback controller would require measuring the load swing angle, which can be accomplished using cable-angle feedback or a vision system that tracks the load position relative to the helicopter.

Regardless of the feedback controller architecture selected for this task, the input shaper should remain directly after the pilot commands in the overall control system, as was done in the model-following controller in this thesis. Because input shaping would suppress residual oscillation caused by helicopter motion resulting from pilot commands, any oscillation detected by the feedback controller could only be caused by external disturbances. This approach allows input shaping to suppress load swing caused by helicopter motion resulting from pilot commands, while the feedback controller can be designed to target load swing caused by disturbances.

APPENDIX A

MATLAB INPUT SHAPING PROGRAM

A.1 Background

Through a project with the Vertical Lift Consortium and the National Rotorcraft Technology Center, Georgia Tech is working on flight control concepts and handling qualities requirements for suspended load operation. Through this program, an input shaping MATLAB program was prepared for and delivered to Boeing Helicopters, Kaman Aerospace, and Sikorsky. They will use this MATLAB program for testing and evaluation of the input-shaping technique as part of helicopter flight control systems. This project is still on-going and results are not yet available. Future collaboration should see input shaping tested in flight simulation with and without pilots, and eventually on actual helicopters carrying suspended loads.

The program was written as a MATLAB class using MATLAB's object-oriented programming capabilities. The following sections contain an overview of the program structure and available functions, with sample code. Additionally, demos illustrating the use of the program in a simulation environment are discussed. These demos were prepared as MATLAB scripts and Simulink models, and the required files to run these demos are included with the program. A few videos were also included in the program package to demonstrate the use of input shaping on real machines, including RC helicopters and cranes.

A.2 Program Functions and Sample Code

This section is an overview of some of the functions available in the MATLAB program. The core functionality is contained in a MATLAB class called `InputShaper` implemented using MATLAB's object-oriented programming capabilities. Object-oriented programming in MATLAB allows the programmer to create custom variable types called classes. Objects, or variables, of a class can then be created.

In the case of the `InputShaper` class, each object of the class contains an input shaper's impulse sequence, the type of input shaper (such as ZV), design natural frequency and damping ratio, and all other information needed by the input shaping process. The user can create objects with many input shaper types to suppress vibration at a specified natural frequency and damping ratio. When applicable to the type of input shaper, the tolerable residual vibration can also be specified.

An `InputShaper` object can then be implemented with a simulation or real-time physical system using a method, or function, available in the program that is passed the raw command to be shaped by the input shaper.

A.2.1 Directory Structure

The following folders are included with the program:

```
@InputShaper
HelperFunctions
```

In order to use the program, the `HelperFunctions` folder and the directory containing the `@InputShaper` folder *must* be on the MATLAB path.

A.2.2 Creating an Object of the `InputShaper` Class – The Constructor Function

An object of the `InputShaper` class is created using the class's constructor function. The syntax for this function is

```
obj = InputShaper(shaperType, designNatFreq, designDampingRatio, samplingTime);
```

where:

- `obj` is the created `InputShaper` object;
- `shaperType` is a 1x2 cell array with a string representing the type of shaper for the first entry and a double representing the tolerable vibration in percent for the second entry (when applicable). The string representing the type of shaper is simply the shaper's abbreviated name, for example `'ZV'`;

- `designNatFreq` and `designDampingRatio` are the natural frequency (in rad/s) and damping ratio of the vibratory mode to be suppressed by the input shaper;
- `samplingTime` is the sampling period (in seconds) of the system, or the size of the time step of the command to be input-shaped. A constant sampling time or time step size must be used.

Here is an example of creating a ZV shaper `InputShaper` object using the constructor function. To create a ZV shaper called `IS_1` to suppress a vibratory mode with a natural frequency of 1 rad/s and zero damping ratio to be used with a 10 ms (0.01 s) sampling time or time step size, the following command should be used:

```
IS_1 = InputShaper({'ZV'}, 1, 0, 0.01)
```

Using this command (without a semicolon) or using the MATLAB `disp` function on the `IS_1` variable results in the following output to the command window:

```
IS_1 =

Shaper Type:                    ZV
Design Natural Frequency:       1 [rad/s]
Design Damping Ratio:           0
Tolerable Residual Vibration:   Only applicable for EI and SI shapers.
Impulse Sequence:
  Times     Amplitudes
   0.0000    0.500
   3.142     0.500
```

This display contains all of the relevant information about the `InputShaper` object, including the shaper type, the full impulse sequence, and the design natural frequency and damping ratio. Once an `InputShaper` object such as `IS_1` has been created, the design natural frequency and damping ratio cannot be changed. In addition, `InputShaper` objects cannot be copied[1]. If an `InputShaper` object identical to `IS_1` is desired, then a new `InputShaper`

[1]The `InputShaper` class is a subclass of the handle superclass. Therefore, a variable representing an `InputShaper` object is merely a pointer to the actual object in memory, so a command such as `IS_2 = IS_1` does not create a copy of `IS_1` called `IS_2`. Rather, using such a command results in `IS_2` pointing to the same `InputShaper` object as `IS_1`. This is identical to the behavior of figure handles in MATLAB (figure objects are also a subclass of the handle superclass).

object should be created with a different variable name by calling the constructor function with the same shaper type, natural frequency, damping ratio, and sampling time.

When an input shaper is created by the constructor function, a technique from Murphy [33] is used to digitize the shaper impulses using the specified sampling time. This is why the constructor requires the sampling period to be specified. The digitization process ensures that the shaper impulses occur at an integer multiple of the sampling time. Therefore, an `InputShaper` object should only be used with commands that have a sampling time equal to the one specified when it was created. The amplitude and time locations of the digitized impulses are stored in the `InputShaper` object and are used to shape the commands. However, the continuous amplitudes and times are what is shown in the command window when using the `disp` function as in the above examples.

A.2.3 Creating Multi-Mode Input Shapers – The `convolveShapers` Method

The `InputShaper` class has a method that allows the user to create multi-mode shapers. The method is called `convolveShapers`. Alternatively, the mtimes operator `*` may be used. The syntax is

```
multiModeShaper = convolveShapers(shaper1, shaper2);
```

or, using the mtimes operator `*`

```
multiModeShaper = shaper1 * shaper2;
```

where `shaper1` and `shaper2` are the single-mode input shapers and `multiModeShaper` is the multi-mode shaper created by convolving `shaper1` and `shaper2` together.

Suppose one wants to create the multi-mode input shaper. First, a ZV shaper to suppress vibration at 1 rad/s and an EI shaper to suppress vibration at 2.5 rad/s with $V_{tol} = 5\%$ are created using the constructor function. This was done in the above examples, where `IS_1` is the ZV shaper and `IS_2` is the EI shaper. Next, the `convolveShapers` method or the mtimes operator `*` is used to create the multi-mode shaper `IS_2mode`

```
IS_2mode = convolveShapers(IS_1, IS_2)
% equivalent to IS_2mode = IS_1 * IS_2
```

This command results in the following output to the command window

```
IS_2mode =

Convolved Shaper: ZV-EI
 The properties of each shaper that were convolved to form this one are listed separately:
 Shaper Type (Shaper 1):                    ZV
 Design Natural Frequency (Shaper 1):       1 [rad/s]
 Design Damping Ratio (Shaper 1):           0
 Tolerable Residual Vibration (Shaper 1):   Only applicable for EI and SI shapers.
 Shaper Type (Shaper 2):                    EI
 Design Natural Frequency (Shaper 2):       2.5 [rad/s]
 Design Damping Ratio (Shaper 2):           0
 Tolerable Residual Vibration (Shaper 2):   5%
 Impulse Sequence:
  Times     Amplitudes
   0.0000    0.131
   1.256     0.238
   2.513     0.131
   3.142     0.131
   4.398     0.238
   5.655     0.131
```

A.2.4 Accessing Important Properties of `InputShaper` Objects – The `get` Methods

Several methods are available to access important properties of `InputShaper` objects. A list of these '`get`' methods and what property each returns is given in Table 3. For the structures returned by `getShaper` and `getDigitalShaper`, it should be clear what each field represents based on its name. Properities with a `Convolved` flag in the name involve the process of combining two shapers together using convolution.

Note that the user is only given read permissions to the property values via these methods – the user does not have write permissions to an `InputShaper` object's properties. Any desired changes to these properties should instead be accomplished by creating a new `InputShaper` object with the desired properties.

A.2.5 Applying `InputShaper` Objects to Commands – The `shapeCommand` Method

The `shapeCommand` method applies an input shaper to a reference command. It can be thought of as the input shaping control element or block in a control system block diagram. It can be used with a real-time command from a human operator or pilot, or be used with a predetermined command. In either case, the method calculates the shaped command one time step at a time. Because an input shaper is a series of time-delayed impulses, the

Table 3: `get` Methods for the `InputShaper` class.

`get` Methods	Properties Returned
`getShaperType`	String representing the type of input shaper
`getShaperTypeConvolved`	Cell array that is a list of strings indicating the types of shapers in a multi-mode shaper
`getModelingParameters`	Natural frequency (in rad/s) and damping ratio used to design the input shaper
`getShaperNumPulses`	Number of impulses in the input shaper
`getShaperPulseTimes`	Vector of input shaper impulse times (in seconds)
`getShaperPulseAmplitudes`	Vector of input shaper impulse amplitudes (unitless)
`getShaper`	Structure with the following fields: `shaperType`, `shaperTypeConvolved`, `natFreq_model`, `dampingRatio_model`, `numPulses`, `pulseTimes`, and `pulseAmplitudes`
`getDigitalShaper`	Structure with the following fields: `shaperType`, `shaperTypeConvolved`, `natFreq_model`, `dampingRatio_model`, `numPulsesDigital`, `pulseTimesDigital`, and `pulseAmplitudesDigital`

method stores the command history in a buffer so that command values at each time step can be accessed. Each value in the buffer corresponds to the command at a specific, discrete time step.

This command buffering technique for implementing input shaping in real-time is illustrated in Figure 36 [43]. There is a command value corresponding to each time step. The unshaped value at time 1 is broken into two pieces when the input shaper is applied to the command. The value scaled by the first impulse is stored at time 1, and the value scaled by the second impulse is stored at time 5. At time 5, the shaped command becomes the sum of the unshaped command at time 5 scaled by the first impulse plus the unshaped command at time 1 scaled by the second impulse. The output of the `shapeCommand` method at time n is the value at time n in the shaped-command buffer.

The syntax for the `shapeCommand` method is

```
u_shaped(curTimeStep) = obj.shapeCommand(u(curTimeStep));
```

where `curTimeStep` is the index of the current time step, `u(curTimeStep)` is the command at the current time step, `obj` is the `InputShaper` object applied to command `u`, and `u_shaped(curTimeStep)` is the resulting shaped command at the current time step.

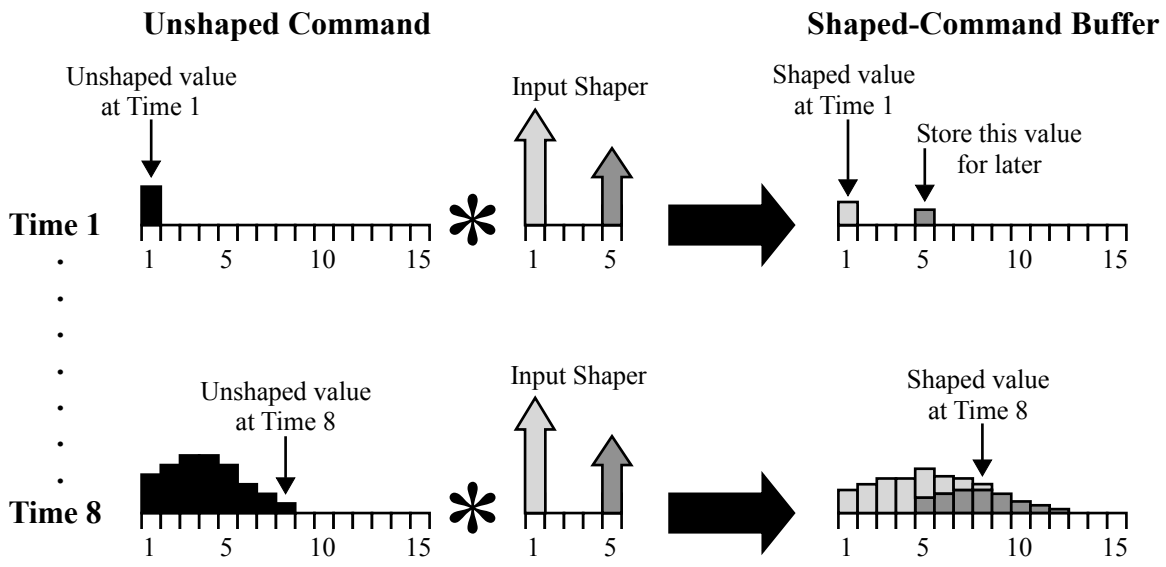


Figure 36: Illustration of the technique used to implement input shaping in real-time [43].

The `shapeCommand` method can be used either in a real-time control loop or with a predetermined command. Examples of both approaches are given in the provided demos. The demos are discussed in the next section. When using a predetermined command, first the command must be fully created before applying the input shaper. Next, `shapeCommand` method must be applied at each time step of the predetermined command. This requires looping through the entire duration of the command, and using the `shapeCommand` method with the command value at each time step. The resulting shaped command can then be used in simulation or applied to the physical system.

A.2.6 Plotting Sensitivity Curves to Analyze Robustness – The `sensitivityCurve` Method

The `InputShaper` class can also be used to calculate and plot the sensitivity curve of an input shaper. This is done using the `sensitivityCurve` method. The method plots the sensitivity curve in a new figure for a range of frequencies with percent residual vibration on the vertical axis and the natural frequency in rad/s on the horizontal axis. The method also returns the vectors of frequency and percent residual vibration used to create the plot. This allows the user to normalize the frequency by the natural frequency before plotting the sensitivity curve, to plot multiple sensitivity curves on one figure, or to export the data

from MATLAB to another program.

Also, the sensitivity curve of an input shaper can be calculated and plotted using either the continuous-time or discretized impulses. There will only be a noticeable difference between the two if the sampling time specified when creating the input shaper is not at least an order of magnitude smaller than the natural frequency used to design the input shaper. Therefore, plotting the digital representation is a good way of checking whether or not the selected sampling time is low enough to not cause significant degradation of the vibration-suppressing properties of the input shaper.

The syntax for the `sensitivityCurve` method is

```
[resVib, wVec] = obj.sensitivityCurve(freqRange, shaperDomainFlag);
```

where `obj` is the `InputShaper` object to plot the sensitivity curve of. `freqRange` is the frequency range over which to plot the sensitivity curve, and must be a 1×2 vector where the entries are the lower and upper end of the frequency range to plot, given in rad/s (*e.g.*, `freqRange = [`ω_{lower}`,` ω_{upper}`]`). `resVib` is a vector of residual vibration in percent, and `wVec` is a vector of frequencies in rad/s at which the residual vibration in `resVib` was plotted. The residual vibration at a given index of `resVib` corresponds to the frequency at that index in `wVec`. `shaperDomainFlag` is a string input flag used to indicate whether the continuous or digital representation of the input shaper should be used to calculate the residual vibration and plot the sensitivity curve. To plot the sensitivity curve using the continuous impulses, `shaperDomainFlag` should be either `'continuous'` or `''`. To plot the sensitivity curve using the discretized impulses, `shaperDomainFlag` should be either `'digital'` or `'discrete'`.

A.2.7 MATLAB and Simulink Demos

Several demos are provided with the program. These demos contain sample code that can be reused in other control programs, with some adjustments such as changing the natural frequency to match the frequency of the undesirable vibratory mode in a given application. The demos included with the program are documented in Table 4.

DEMO_1_SecondOrderSystem.m	Uses the `InputShaper` class and `shapeCommand` method to suppress the vibration of an underdamped second order dynamic system.
DEMO_2_control_loop_skeleton_code.m	Skeleton code that provides an outline for using the `InputShaper` class in conjunction with a control loop written in MATLAB. It illustrates how an input shaper can be created, used to shape commands obtained in real-time (from a pilot or human operator, for example), and passed on to other control elements or even hardware operated by MATLAB.
DEMO_3_Simulink.m, DEMO_3_Simulink_Model.mdl	Overview of how to use the `InputShaper` class in conjunction with Simulink (or another program). An input shaper is created using the constructor function, and the impulse amplitudes and times are extracted from the `InputShaper` object. The script then runs DEMO_Simulink_Model.mdl, which is a Simulink model of a generic input shaper and uses the impulse amplitudes and times to create a shaped command. Alternatively, the impulse amplitudes and times could be exported from MATLAB for use in another program.
DEMO_4_FourthOrderSystem.m	Uses the `InputShaper` class to suppress both underdamped vibratory modes of a fourth order dynamic system. This is done by creating two shapers and using the `convolveShapers` method to create a multi-mode input shaper.

Table 4: Demos included with the Input Shaping Program.

Each demo has comments that explain the code to users and will guide them through the demo's use. The demo files should be placed in and run from the same directory as the `@InputShaper` and `HelperFunctions` folders, and the `HelperFunctions` folder must be added to the MATLAB path. Alternatively, if the demos are placed in a different directory, then the `@InputShaper` folder must be in a directory that is on the MATLAB path and the HelperFunctions folder must be added to the MATLAB path.

The provided MATLAB script *DEMO_2_control_loop_skeleton_code.m* demonstrates the use of the method in a real-time control loop. The rest of the provided demo scripts use a predetermined command. The *DEMO_3_Simulink.m* script is a demo that shows how to use the program with Simulink or another program outside of MATLAB.

A.2.8 How to Implement the Program

There are two main components to using the `InputShaper` class. First, the constructor is used to create an `InputShaper` object. Second, the `InputShaper` object is applied to a pre-planned or real-time command using the `shapeCommand` method. This takes the following form in MATLAB code:

```
% Create the input shaper
obj = InputShaper(shaperType, designNatFreq, designDampingRatio, samplingTime);

% Control loop, or loop through the pre-planned command
for curTimeStep = 1:length(time)

   % other code, if this loop is a control loop...

   % Apply the input shaper to the command u
   u_shaped(curTimeStep) = obj.shapeCommand(u(curTimeStep));

   % other code, if this loop is a control loop...

end
```

where `time` is the vector of times associated with the pre-planned command, or `length(time)` is equal to the total number of time steps in your simulation.

If a dynamic system has multiple inputs that need to be input-shaped to reduce the excitation of the same or different vibratory mode(s), then in the context of the program a unique `InputShaper` object must be created for each individual input. For example, the longitudinal and lateral cyclic inputs from a helicopter pilot both lead to helicopter motion that causes oscillation of a sling load. Therefore, if it is desirable to reduce the sling load oscillation using input shaping, then each of these inputs must be shaped by a separate input shaper. In the context of the program, this requires creating an input shaper for both of these inputs:

```
% Create the input shapers for each input:
```

```
IS_long = InputShaper(shaperType1, designNatFreq1, designDampingRatio1, samplingTime);
IS_lat = InputShaper(shaperType2, designNatFreq2, designDampingRatio2, samplingTime);

% Control loop, or loop through the pre-planned longitudinal and lateral commands
for curTimeStep = 1:length(time)

   % other code, if this loop is a control loop...

   % Apply the input shaper IS_long to the longitudinal cyclic command u
   u_shaped(curTimeStep) = IS_long.shapeCommand(u(curTimeStep));
   % Apply the input shaper IS_lat to the lateral cyclic command v
   v_shaped(curTimeStep) = IS_lat.shapeCommand(v(curTimeStep));

   % other code, if this loop is a control loop...

end
```

Note that the shaper types and parameters used to create the `InputShaper` objects may not necessarily be equal, but the sampling time is likely the same if the commands are issued from the same control loop or within the same simulation. The example above could be extended for as many inputs as desired simply by duplicating the process of creating `InputShaper` objects and applying them to commands using the `shapeCommand` method.

A.3 MATLAB Program Code

```
classdef InputShaper < handle
% classdef InputShaper < handle
%
% An object-oriented approach to defining and implementing the Input
% Shaping algorithm. Input shaping is a command generation technique that
% reduces motion-induced vibration in a flexible system. It does this by
% modifying, or shaping, reference commands in real time such that
% specified vibratory modes of the flexible system are canceled. It is
% implemented by convolving a sequence of impulses, called an input shaper,
% with a reference command. The product of the convolution is issued to the
% system or plant in place of the reference command.
%
% The InputShaper class is a subclass of the handle superclass. This means
% that objects of the InputShaper class are passed by reference, or handle,
% rather than by value like most MATLAB objects. This is similar to how
% figure objects work.
%
% ************************************************************************
```

```
% List of Input Shaper types currently available in this class:
%   ZV
%   ZVD
%   ZVDD
%   ZVDDD
%   EI
%   UMZV
%   UMZVD
%   UMEI
%   SI (requires Optimization toolbox)
%   SI2Mode (requires Optimization toolbox)
%   Unshaped
% ************************************************************************
%
% ************************************************************************
% The InputShaper class has the following public methods available:
%
%         InputShaper
%           Constructor function for the class. It creates an InputShaper
%           object of a specified type. The input shaper is then designed
%           to suppress vibration at a specified natural frequency
%           and damping ratio.
%
%         shapeCommand
%           Applies the InputShaper to the given command while storing the
%           partial command history.
%
%         mtimes
%           Overloads MATLAB default mtimes.m operator * with a method that
%           convolves, or combines, two shapers.
%
%         convolveShapers
%           Convolves, or combines, two shapers to create a multi-mode
%           multi-mode shaper.
%
%         sensitivityCurve
%           Plots the Sensitivity Curve of the InputShaper object
%
%         disp
%           Overloads MATLAB default disp.m
%
%         getShaper
%           Return all shaper properties as a struct
%
%         getDigitalShaper
%           Return all properties of the digitized shaper as a struct
%
%         getShaperType
%           Return type of shaper (a string)
%
%         getShaperTypeConvolved
%           Return types of shapers in the convolved shaper (a cell array
%           that is a list strings indicating shaper type)
%
%         getModelingParameters
%           Return modeling natural frequency and damping ratio of shaper
%
%         getShaperNumPulses
%           Return shaper numPulses
```

```
%
%         getShaperPulseAmplitudes
%           Return shaper pulseAmplitudes
%
%         getShaperPulseTimes
%           Return shaper pulseTimes
%
% More detailed descriptions of these methods, including descriptions of
% the methods' inputs and outputs, can be found in each method's help or
% in the function header in the method's .m file.
% ************************************************************************

    properties(GetAccess = 'private', SetAccess = 'private')

        % Shaper Design Parameters
        shaperType = 'OFF';
        natFreq_model = [];
        dampingRatio_model = [];
        vTol = 0;
        convolved = false;
        convolvedShaperTypes = {};

        % Shaper Impulse Sequence
        numPulses = [1];
        pulseAmplitudes = [1];
        pulseTimes = [0];
        % Digital representation of the above impulse sequence (the
        % pulseTimesDigital are correctly timed with the sampling period)
        numPulsesDigital = [1];
        pulseAmplitudesDigital = [1];
        pulseTimesDigital = [0];

        % Simulation Parameters
        T_sampling = [];
        n = 0;
        t_history = [];
        y_history = [];

    end % Properties

    methods (Access = public)
    % Public Methods

    %% CLASS CONSTRUCTOR:
        function obj = InputShaper(shaper, natFreq, dampingRatio, T, varargin)
        % Constructor function that creates an object of class InputShaper
        %
        % function obj = InputShaper(shaper, natFreq, dampingRatio, T)
        % is the constructor function for the InputShaper class. It creates
        % an InputShaper object of the specified input shaper type. The
        % input shaper suppresses vibration at the specified natural
        % frequency and damping ratio. The sampling time of the simulation
        % or real-time command that the input shaper will be used with must
        % also be provided.
        %
        % Inputs:
        %   shaper           A cell array where the first cell is a
        %                     string representing the type of input
        %                     shaper (see list below for options). The
```

```
%                     second cell is the tolerable vibration as
%                     a percentage (only applicable for EI and
%                     SI type shapers, and defaults to 5% for these
%                     shapers when it is not specified).
%   natFreq          The natural frequency of the vibrational
%                     mode to be suppressed, in rad/s.
%   dampingRatio     The damping ratio of the vibrational mode to
%                     be suppressed.
%   T                The sampling time, in seconds, of the
%                     simulation, physical system, real-time
%                     command, etc. that this input shaper will
%                     be used with.
%   varargin         Optional input arguments. Used internally
%                     in other class methods (see below).
%
% The form of some inputs change when using SI or SI2Mode shapers:
%  For SI shapers:
%  - natFreq should be a [1x2] vector of frequencies (in rad/s) of
%    the form [w_min, w_max] where w_min is the lower frequency
%    of the range to suppress vibration within and w_max is the
%    upper frequency of the range to suppress vibration within.
%  For SI2Mode shapers:
%  - The second cell of shaper should be a [1x2] vector of
%    tolerable percentage vibration of the form [vTol1, vTol2]
%  - natFreq should be a [1x4] vector of frequencies (in rad/s) of
%    the form [w1_min, w1_max, w2_min, w2_max]
%  - dampingRatio should be a [1x2] vector of damping ratios of the
%    form [damp1, damp2]
%  ...where...
%   FOR THE FIRST MODE:
%  w1_min = lower frequency of range to suppress vibration within
%  w1_max = upper frequency of range to suppress vibration within
%  damp1 = damping ratio
%  vTol1 = vibration level tolerated, in percent
%   FOR THE SECOND MODE:
%  w2_min = lower frequency of range to suppress vibration within
%  w2_max = upper frequency of range to suppress vibration within
%  damp2 = damping ratio
%  vTol2 = vibration level tolerated, in percent
%
% Outputs:
%   obj                 An InputShaper object.
% ********************************************************************
% List of Input Shaper types currently available:
%   'ZV'
%   'ZVD'
%   'ZVDD'
%   'ZVDDD'
%   'EI'
%   'UMZV'
%   'UMZVD'
%   'UMEI'
%   'SI'
%   'SI2Mode'
%   Unshaped (use 'OFF' for shaper type)
% ********************************************************************
% SAMPLE FUNCTION CALLS:
% ZV_shaper = InputShaper({'ZV'}, 2, 0, 0.01);
%     This forms a ZV shaper to suppress an undamped mode with
```

```
%     natural frequency = 2 rad/s. The sampling time is 10 ms
%     (input as 0.01 s). Note that no tolerable vibration percent
%     is specified in the first input cell array because ZV
%     shapers do not use tolerable vibration in the design process.
%
% EI_shaper = InputShaper({'EI', 5}, 2*pi, 0.1, 0.001)
%     This forms an EI shaper to suppress an underdamped mode with
%     natural frequency = 2*pi rad/s. The sampling time is 1 ms
%     (input as 0.001 s). Note that a tolerable vibration percent
%     of 5% is specified in the first input cell array.
%
% SI_shaper = InputShaper({'SI', 8}, [1.5, 4], 0.05, 0.01)
%     This forms a SI shaper to suppress vibration in the frequency
%     range from 1.5 rad/s to 4 rad/s with 0.05 damping ratio.
%     The sampling time is 10 ms (input as 0.01 s).
% ********************************************************************
% A NOTE ON THE OPTIONAL INPUT ARGUMENTS (varagin)
%   This constructor is also called by some of the other methods
%   in the InputShaper class to create InputShaper objects. One
%   example is in the convoleShapers method. This method combines
%   two input shapers to form a shaper that suppresses multilple
%   modes of vibration. This is known as a multi-mode shaper. The
%   constructor function must be called to create the multi-mode
%   shaper as a new InputShaper object. The constructor is called
%   with an additional input flag accepted by the varargin input to
%   identify that the new InputShaper object to be created should
%   be a combination of two other InputShaper objects.
% ********************************************************************

    if isempty(varargin)
    % Was the constructor called from the convolveShapers method?
    %  Set convolveFlag to false if it wasn't.
        convolveFlag = false;
    else
        convolveFlag = varargin{1};
    end
    if ~convolveFlag
    % The constructor was NOT called from the convolveShapers
    %  method, so perform the standard design procedure.
% Error checking and input management
  if ((iscell(shaper)) && (length(shaper)==1))
     obj.shaperType = shaper{1}; % Use given input shaper type
  if strcmp(obj.shaperType, 'ei') || strcmp(obj.shaperType, 'EI')
     warning('Using tolerable residual vibration of 5% (default for EI shaper).');
      obj.vTol = 5/100; % Use default vTol for EI shapers (5%)
  elseif strcmp(obj.shaperType, 'SI') || strcmp(obj.shaperType, 'SI')
      warning('Tolerable residual vibration must be specified for SI shapers.
       Using tolerable residual vibration of 5%.');
      obj.vTol = 5/100; % Use default vTol for SI shapers (5%)
elseif strcmp(obj.shaperType, 'si2mode') || strcmp(obj.shaperType, 'SI2Mode')
         warning('Tolerable residual vibration must be specified for SI2Mode
            shapers. Using tolerable residual vibration of 5% for both modes.');
           obj.vTol(1) = 5/100; % Use default vTol for SI2Mode shapers (5%)
           obj.vTol(2) = 5/100; % Use default vTol for SI2Mode shapers (5%)
        end
 elseif ((iscell(shaper)) && (length(shaper)==2))
           if ~strcmp(class(shaper{2}), 'double')
                error('Second entry in SHAPER cell array must be a double
                 representing tolerable residual vibration (as a percentage).');
```

```
            end
            obj.shaperType = shaper{1}; % Use given input shaper type
        if ~(strcmp(obj.shaperType, 'ei') || strcmp(obj.shaperType, 'EI') || ...
          strcmp(obj.shaperType, 'si') || strcmp(obj.shaperType, 'SI') || ...
          strcmp(obj.shaperType, 'si2mode') || strcmp(obj.shaperType, 'SI2Mode'))
                if strcmp(obj.shaperType, 'umei') || strcmp(obj.shaperType, 'UMEI')
                    obj.vTol = 0.05;
                warning('Value for tolerable vibration (Vtol) automatically set
                    to 5% (0.05).  This is the only Vtol value available for UMEI
                    shapers in this version of the shaper design program.');
            else
                warning(['Tolerable residual vibration cannot be specified
                        for ', obj.shaperType, ' shapers.']);
                end
            else
                obj.vTol = shaper{2}/100; % Use given input value of vTol
            end
    elseif iscell(shaper)
            warning('Improper shaper cell array length. Using default values
              of ''OFF'' (no shaping) and vTol = 5%.');
        else
            error('Improper type for SHAPER; input variable
                    SHAPER must be a cell array.');
        end
        % Simulation Parameters
        obj.T_sampling = T;
        obj.n = 1;
        % Shaper Design Parameters
        obj.natFreq_model = natFreq;
        obj.dampingRatio_model = dampingRatio;
        % Design Input Shaper of the appropriate type using the
        %  given system parameters
        if (strcmp(obj.shaperType, 'si') || strcmp(obj.shaperType, 'SI'))
        % Call the function that performs the SI shaper design
        %  routine using optimization.
            % But first some error checking...
            if length(natFreq) ~= 2
                error('Must specify a frequency range in the
                            form of [wmin, wmax] for SI shapers.')
            elseif length(dampingRatio) ~= 1
                error('Must specify only one damping ratio for SI shapers.')
            end
            ShaperLength = 3; % Redefined in si_fmincon function
            ShaperFlag = 0; % Positive SI shaper
            neg = -1;
            X0 = 0; % Unknown initial guess for optimization routine
            fmin = natFreq(1)./(2.*pi); % [Hz]
            fmax = natFreq(2)./(2.*pi); % [Hz]
            damp = dampingRatio(1);
            % Design SI shaper
            SI_shaper = si_fmincon(ShaperLength,ShaperFlag,neg,X0,...
            fmin,fmax,damp,obj.vTol,obj.T_sampling);
            obj.pulseAmplitudes = SI_shaper(:,2)';
            obj.pulseTimes = SI_shaper(:,1)';
            obj.numPulses = length(obj.pulseAmplitudes);
      elseif (strcmp(obj.shaperType,'si2mode')||strcmp(obj.shaperType,'SI2Mode'))
       % Call the function that performs the SI2Mode shaper design
       %  routine using optimization.
            % But first some error checking...
```

```
            if length(natFreq) ~= 4
                error('Must specify two frequency ranges in the form
                   of [w1min, w1max, w2min, w2max] for SI shapers.')
            elseif length(dampingRatio) ~= 2
                error('Must specify two damping ratios for SI2Mode shapers.')
            elseif length(obj.vTol) ~= 2
                error('Must specify exactly two tolerable residual vibrations
                         levels for SI2Mode shapers.')
            end
            ShaperLength = 3; % Redefined in si_fmincon function
            ShaperFlag = 0; % Positive SI2Mode shaper
            neg = -1;
            X0 = 0; % Unknown initial guess for optimization routine
            f1min = natFreq(1)./(2.*pi); % [Hz]
            f1max = natFreq(2)./(2.*pi); % [Hz]
            f2min = natFreq(3)./(2.*pi); % [Hz]
            f2max = natFreq(4)./(2.*pi); % [Hz]
            damp1 = dampingRatio(1);
            damp2 = dampingRatio(2);
            % Design SI2Mode shaper
            SI2Mode_shaper = si2mode_fmincon(ShaperLength,ShaperFlag,neg,X0,...
            f1min,f1max,damp1,obj.vTol(1),f2min,f2max,damp2,...
            obj.vTol(2),obj.T_sampling);
            obj.pulseAmplitudes = SI2Mode_shaper(:,2)';
            obj.pulseTimes = SI2Mode_shaper(:,1)';
            obj.numPulses = length(obj.pulseAmplitudes);
        else
            % Design shaper using closed form solutions for some
            %  shaper types and polynomial curve fits for other
            %  types
            ShaperDesign_Poly(obj);
        end
        % Create a digital respresentation of the shaper from its
        %  continuous-time representation
        if ~(strcmp(obj.shaperType, 'off') || strcmp(obj.shaperType, 'OFF'))
            digitizeShaper(obj);
        end
    if strcmp(obj.shaperType, 'off') || strcmp(obj.shaperType, 'OFF')
          obj.t_history = [0];
          obj.y_history = [0];
    else
         % Prebuffer t_history and y_history up to the time step
         %  at which the last impulse in the shaper's impulse
         %  sequence occurs
obj.t_history = 0:obj.T_sampling:(obj.pulseTimesDigital(obj.numPulsesDigital));
     obj.y_history = zeros(1, length(obj.t_history));
    end
        obj.convolvedShaperTypes{1} = obj.shaperType;
    else
    % The constructor was called from the convolveShapers method,
    %  so don't perform the shaper design process. Use values
    %  for some shaper parameters as inherited from the shapers
    %  being convolved together.
        % Shaper Parameters
        obj.convolvedShaperTypes = shaper{1};
        type_str = obj.convolvedShaperTypes{1};
        for i = 2:(length(obj.convolvedShaperTypes))
            type_str = [type_str, '-', obj.convolvedShaperTypes{i}];
        end
```

```
            obj.shaperType = type_str;
            obj.natFreq_model = sort(natFreq);
            obj.dampingRatio_model = sort(dampingRatio);
            obj.vTol = shaper{2}/100;
            obj.convolved = true;
            % Simulation Parameters
            obj.T_sampling = T;
            obj.n = 1;
            % NOTE: The convolveShapers method handles the continuous
            %  to digital conversion and prebuffers t_history and
            %  y_history. There is no real benefit to doing it there
            %  versus here.
        end
    end % InputShaper Constructor
%% COMMON INPUT SHAPING ACTIONS:
    y_shaped = shapeCommand(obj, y);
    % Applies the InputShaper to the given command while storing the
    %  partial command history.
    convolvedShaper = convolveShapers(obj1, obj2);
    % Convolves, or combines, two shapers to create a multi-mode
    %  multi-mode shaper.
    product = mtimes(obj1, obj2);
    % Overloads MATLAB default mtimes.m operator * with a method that
    %  convolves, or combines, two shapers.
    [resVib, wVec] = sensitivityCurve(obj, freqRange, shaperDomainFlag);
    % Plots the Sensitivity Curve of the InputShaper
    disp(obj);
    % Overloads MATLAB default disp.m
%% 'GET' METHODS:
    shaperOut = getShaper(obj);
    % Returns all shaper properties as a struct
    digitalShaperOut = getDigitalShaper(obj);
    % Returns all properties of the digitized shaper as a struct
    typeOut = getShaperType(obj);
    % Returns type of shaper (a string)
    typeConvolvedOut = getShaperTypeConvolved(obj);
    % Returns types of shapers in the convolved shaper (a cell array
    %  that is a list strings indicating shaper type)
    [w_model, zeta_model] = getModelingParameters(obj);
    % Returns modeling natural frequency and damping ratio of shaper
    numPulsesOut = getShaperNumPulses(obj);
    % Returns shaper numPulses
    pulseAmplitudesOut = getShaperPulseAmplitudes(obj);
    % Returns shaper pulseAmplitudes
    pulseTimesOut = getShaperPulseTimes(obj);
    % Returns shaper pulseTimes
end % Public Methods
methods (Access = private)
% Private Methods
    ShaperDesign_Poly(obj);
    % Used by the Constructor method to design the InputShaper
    digitizeShaper(obj);
    % Creates a digital respresentation of the shaper given in obj
    % from its continuous-time representation
    resVib = residualVibration(obj, w_n, zeta, shaperDomainFlag);
    % Returns the residual vibration resulting from the InputShaper's
    % impulse sequence at the given frequency w_n and damping ratio
    % zeta as a percentage of the amount of residual vibration caused
    % by a single, unity-magnitude impulse
```

```
    end % Private Methods
end % InputShaper classdef

function ShaperDesign_Poly(obj)
%
% List of Input Shapers available in this method:
%   ZV
%   ZVD
%   ZVDD
%   ZVDDD
%   EI
%   UMZV
%   UMZVD
%   UMEI
%   Unshaped (use OFF or off for shaperType)


%% Define anonymous function ----------------------------------------------
    valPoly = @(M0,M1,M2,M3,T,zeta) T*(M0 + M1*zeta + M2*zeta^2 + M3*zeta^3);
%% Define structures ------------------------------------------------------
    shaperContinuous = struct;
%% Calculate system parameters --------------------------------------------
    freqNaturalIn = obj.natFreq_model;
    dampingIn = obj.dampingRatio_model;
    K = exp((-dampingIn*pi)/sqrt(1-dampingIn^2));
    freqDamped = freqNaturalIn*sqrt(1-dampingIn^2);
    periodDamped = (2*pi)/freqDamped;
    periodUndamped = (2*pi)/freqNaturalIn;
%% Design specified input shaper ------------------------------------------
    if strcmp(obj.shaperType, 'zv') || strcmp(obj.shaperType, 'ZV')
        denom = K + 1;
        shaperContinuous.numPulses = 2;
        shaperContinuous.pulseAmps =   [   1/denom      K/denom ];
        shaperContinuous.pulseTimes =  [   0            periodDamped/2  ];
    elseif strcmp(obj.shaperType, 'zvd') || strcmp(obj.shaperType, 'ZVD')
        denom = K^2 + 2*K + 1;
        shaperContinuous.numPulses = 3;
        shaperContinuous.pulseAmps =   [ 1/denom     (2*K)/denom     (K^2)/denom ];
        shaperContinuous.pulseTimes =  [ 0        periodDamped/2     periodDamped ];
    elseif strcmp(obj.shaperType, 'zvdd') || strcmp(obj.shaperType, 'ZVDD')
        denom = K^3 + 3*K^2 + 3*K + 1;
        shaperContinuous.numPulses = 4;
        shaperContinuous.pulseAmps =   [   1/denom     (3*K)/denom   ...
                  (3*K^2)/denom       K^3/denom           ];
        shaperContinuous.pulseTimes =  [   0           (1/2)*periodDamped  ...
            periodDamped        (3/2)*periodDamped  ];
    elseif strcmp(obj.shaperType, 'zvddd') || strcmp(obj.shaperType, 'ZVDDD')
        denom = K^4 + 4*K^3 + 6*K^2 + 4*K + 1;
        shaperContinuous.numPulses = 5;
        shaperContinuous.pulseAmps =   [1/denom (4*K)/denom (6*K^2)/denom...
               (4*K^3)/denom           (K^4)/denom     ];
        shaperContinuous.pulseTimes =  [0         (1/2)*periodDamped ...
              periodDamped        (3/2)*periodDamped      2*periodDamped  ];
    elseif strcmp(obj.shaperType, 'ei') || strcmp(obj.shaperType, 'EI')
        A1 =    0.24968 ...
                + 0.24962*obj.vTol ...
                + 0.80008*dampingIn ...
                + 1.23328*obj.vTol*dampingIn ...
                + 0.49599*dampingIn^2 ...
```

```
            + 3.17316*obj.vTol*dampingIn^2;
    A3 =    0.25149 ...
            + 0.21474*obj.vTol ...
            - 0.83249*dampingIn ...
            + 1.41498*obj.vTol*dampingIn ...
            + 0.85181*dampingIn^2 ...
            - 4.90094*obj.vTol*dampingIn^2;
    t2 =    periodDamped*(0.49990 ...
            + 0.46159*obj.vTol*dampingIn ...
            + 4.26169*obj.vTol*dampingIn^2 ...
            + 1.75601*obj.vTol*dampingIn^3 ...
            + 8.57843*obj.vTol^2*dampingIn ...
            - 108.644*obj.vTol^2*dampingIn^2 ...
            + 336.989*obj.vTol^2*dampingIn^3);
    shaperContinuous.numPulses = 3;
    shaperContinuous.pulseAmps =   [   A1      1-(A1+A3)   A3              ];
    shaperContinuous.pulseTimes =  [   0       t2          periodDamped    ];
elseif strcmp(obj.shaperType, 'umzv') || strcmp(obj.shaperType, 'UMZV')...
 || strcmp(obj.shaperType, 'um-zv') || strcmp(obj.shaperType, 'UM-ZV')
    t1 = valPoly(0,       0,       0,       0,       periodUndamped, dampingIn);
    t2 = valPoly(0.16724, 0.27242, 0.20345, 0.00000, periodUndamped, dampingIn);
    t3 = valPoly(0.33323, 0.00533, 0.17914, 0.20125, periodUndamped, dampingIn);
    if dampingIn == 0
        t1 = 0;
        t2 = periodUndamped/6;
        t3 = periodUndamped/3;
    end
    shaperContinuous.numPulses = 3;
    shaperContinuous.pulseAmps =   [    1       -1      1   ];
    shaperContinuous.pulseTimes =  [    t1      t2      t3  ];
    if dampingIn > 0.4
        warning('Damping ratio is very high: polynomial fit for UM-ZV shaper design
         is outside of its ideal range.  Shaper effectiveness may be impaired.');
    end
elseif strcmp(obj.shaperType, 'umzvd') || strcmp(obj.shaperType, 'UMZVD')...
 || strcmp(obj.shaperType, 'um-zvd') || strcmp(obj.shaperType, 'UM-ZVD')
    t1 = valPoly(0,       0,       0,       0,       periodUndamped, dampingIn);
    t2 = valPoly(0.08945, 0.28411, 0.23013, 0.16401, periodUndamped, dampingIn);
    t3 = valPoly(0.36613, -0.08833,0.24048, 0.17001, periodUndamped, dampingIn);
    t4 = valPoly(0.64277, 0.29103, 0.23262, 0.43784, periodUndamped, dampingIn);
    t5 = valPoly(0.73228, 0.00992, 0.49385, 0.38633, periodUndamped, dampingIn);
    shaperContinuous.numPulses = 5;
    shaperContinuous.pulseAmps =   [    1       -1      1       -1      1   ];
    shaperContinuous.pulseTimes =  [    t1      t2      t3      t4      t5  ];
    if dampingIn > 0.4
        warning('Damping ratio is very high: polynomial fit for UM-ZVD shaper design
         is outside of its ideal range.  Shaper effectiveness may be impaired.');
    end
elseif strcmp(obj.shaperType, 'umei') || strcmp(obj.shaperType, 'UMEI')...
 || strcmp(obj.shaperType, 'um-ei') || strcmp(obj.shaperType, 'UM-EI')
    t1 = valPoly(0,       0,       0,       0,       periodUndamped, dampingIn);
    t2 = valPoly(0.09374, 0.31903, 0.13582, 0.65274, periodUndamped, dampingIn);
    t3 = valPoly(0.36798, -0.05894,0.13641, 0.63266, periodUndamped, dampingIn);
    t4 = valPoly(0.64256, 0.28595, 0.26334, 0.24999, periodUndamped, dampingIn);
    t5 = valPoly(0.73664, 0.00162, 0.52749, 0.19208, periodUndamped, dampingIn);
    shaperContinuous.numPulses = 5;
    shaperContinuous.pulseAmps =   [    1       -1      1       -1      1   ];
    shaperContinuous.pulseTimes =  [    t1      t2      t3      t4      t5  ];
    if dampingIn > 0.4
```

```
            warning('Damping ratio is very high: polynomial fit for UM-EI shaper design
             is outside of its ideal range.  Shaper effectiveness may be impaired.');
        end
    elseif strcmp(obj.shaperType, 'off') || strcmp(obj.shaperType, 'OFF')
        shaperContinuous.numPulses = 1;
        shaperContinuous.pulseAmps = 1;
        shaperContinuous.pulseTimes = 0;
    else
        error('Not an expected kind of shaper!');
    end
    obj.numPulses = shaperContinuous.numPulses;
    obj.pulseAmplitudes = shaperContinuous.pulseAmps;
    obj.pulseTimes = shaperContinuous.pulseTimes;
end % ShaperDesign_Poly

function digitizeShaper(obj)
% Creates a digital respresentation of the shaper given in obj from its
% continuous-time representation
%
% ************************************************************************
% References:
%   Murphy, B.R. and Watanabe, I., "Digital Shaping Filters for Reducing
%   Machine Vibration," IEEE Transactions on Robotics and Automation, Vol.
%   8, No. 2, April 1992.
% ************************************************************************

    % System parameters
    w_n = obj.natFreq_model(1);
    zeta = obj.dampingRatio_model(1);
    % Calculate damped natural frequency
    w_d = w_n.*sqrt(1-zeta.^2);
    % Move impulses onto multiples of the sampling period (except the first
    %  impulse)
    B = [obj.pulseAmplitudes(1)]; % digitized pulseAmplitudes
    t = [obj.pulseTimes(1)]; % digitized pulseTimes in [s]
    for i=2:(obj.numPulses)
        if(mod(obj.pulseTimes(i), obj.T_sampling)==0)
            % If an impulse happens exactly at a sampling time, do nothing
            B = [B, obj.pulseAmplitudes(i)];
            t = [t, obj.pulseTimes(i)];
        else
            % Find sampling times on either side of each impulse
            t_k = floor(obj.pulseTimes(i)./obj.T_sampling).*obj.T_sampling;
            t_k_plus_1 = ceil(obj.pulseTimes(i)./obj.T_sampling).*obj.T_sampling;
            t = [t, t_k, t_k_plus_1];
            % Use technique in Murphy (1992) to calculate amplitudes of two
            % impulses that fall on sampling times on either side of each
            % original impulse
            phi_k = abs(t_k - obj.pulseTimes(i)).*w_d;
            phi_k_plus_1 = abs(t_k_plus_1 - obj.pulseTimes(i)).*w_d;
            B_k = obj.pulseAmplitudes(i).*exp(-zeta.*(t_k-obj.pulseTimes(i))...
            .*w_n).*sin(phi_k_plus_1)./(sin(phi_k_plus_1).*cos(phi_k)...
            +sin(phi_k).*cos(phi_k_plus_1));
            B_k_plus_1 = obj.pulseAmplitudes(i).*exp(-zeta..
            .*(t_k_plus_1-obj.pulseTimes(i)).*w_n)...
            .*sin(phi_k)./(sin(phi_k_plus_1).*cos(phi_k)...
            +sin(phi_k).*cos(phi_k_plus_1));
            B = [B, B_k, B_k_plus_1];
        end
```

```
    end
    % Scale each impulse so that the sum of the digitzed impulse sequence is 1.
    B = B./sum(B);
    obj.numPulsesDigital = length(B);
    obj.pulseAmplitudesDigital = B;
    obj.pulseTimesDigital = t;
end % digitzeShaper

function y_shaped = shapeCommand(obj, y)
% Applies the InputShaper to the given command while storing the
%  partial command history
%
% function y_shaped = shapeCommand(obj, y) applies the input shaper
% given in obj to the command y (note that the input command y is
% a single value). Be sure to apply shapeCommand to each command value
% or time step only once.
%
% Inputs:
%   obj             InputShaper object to use to shape the given command.
%   y               Command (at time t) to be input shaped. Units should be
%                    consistent from one function call to the next for a
%                    given InputShaper object. For example, do not call
%                    shapeCommand with a y in meters in one time step and
%                    then use a y in m/s in the next time step for the same
%                    InputShaper object.
%
% Outputs:
%   y_shaped        The input shaped command at the current time step.

    obj.y_history = [obj.y_history, 0];
    % Input Shape y
    for i = 1:obj.numPulsesDigital
        cur_timeStep = round(obj.pulseTimesDigital(i)./obj.T_sampling) + obj.n;
        obj.y_history(cur_timeStep) = y.*obj.pulseAmplitudesDigital(i)...
         + obj.y_history(cur_timeStep);
    end
    % Update index for next time step
    obj.n = obj.n + 1;
    % Return the shaped command at the current time step only
    y_shaped = obj.y_history(obj.n - 1);
end % shapeCommand

function product = mtimes(obj1, obj2)
% Overloads MATLAB default mtimes.m operator * with a method that
%  convolves two shapers together.
%
% function product = mtimes(obj1, obj2) combines InputShaper object obj1
% with InputShaper object obj2 to create a multi-mode shaper that
% suppresses vibration at the design modes of both obj1 and obj2.
%
% Inputs:
%   obj1            First InputShaper object
%   obj2            Second InputShaper object
%
% Outputs:
%   product         A multi-mode InputShaper object. It is created by
%                    convolving InputShaper objects obj1 and obj2.
product = convolveShapers(obj1, obj2);
end % mtimes
```

```
function convolvedShaper = convolveShapers(obj1, obj2)
% Convolves, or combines, two shapers to create a multi-mode shaper.
%
% function convolvedShaper = convolveShapers(obj1, obj2) combines
% InputShaper object obj1 with InputShaper object obj2 to create a
% multi-mode shaper that suppresses vibration at the design modes of both
% obj1 and obj2.
%
% Inputs:
%   obj1                First InputShaper object
%   obj2                Second InputShaper object
%
% Outputs:
%   convolvedShaper     A multi-mode InputShaper object. It is created by
%                        convolving InputShaper objects obj1 and obj2.

%% Create new shaper. Uses modeled w_n and
%% zeta from both obj1 and obj2 for bookkeeping purposes.

    if (obj1.convolved || obj2.convolved)
        warning('Not recommended to convolve a shaper with another
         that was created by convolving two shapers.
          May result in unexpected performance.')
    elseif (strcmp(obj1.shaperType, 'si') || strcmp(obj1.shaperType, 'SI')...
     || strcmp(obj2.shaperType, 'si') || strcmp(obj2.shaperType, 'SI'))
        warning('Use a SI2Mode shaper instead of convolving an SI shaper
         with another shaper')
    elseif (strcmp(obj1.shaperType, 'si2mode') ||...
     strcmp(obj1.shaperType, 'SI2Mode') || strcmp(obj2.shaperType, 'si2mode')...
      || strcmp(obj2.shaperType, 'SI2Mode'))
        warning('SI2Mode shapers already account for 2 modes. If more than
        two modes are to be suppressed, it would be better to convolve
         simpler shapers together')
    end
    % Hyphenate shaper names together (put the shaper with the lower
    %  design natural frequency first). Also, concatenate vTol levels
    %  for both shapers in the same order.
    if (obj1.natFreq_model(1) <= obj2.natFreq_model(1))
        shaperType = {obj1.shaperType, obj2.shaperType};
        vTol = [obj1.vTol, obj2.vTol];
    else
        shaperType = {obj2.shaperType, obj1.shaperType};
        vTol = [obj2.vTol, obj1.vTol];
    end
    % Both obj1 and obj2 should have the same sampling times. If not, the
    % function uses the larger sampling time.
    if obj1.T_sampling >= obj2.T_sampling
        T_s = obj1.T_sampling;
    else
        T_s = obj2.T_sampling;
    end
    % Use modified version of constructor to create the new shaper object
    convolvedShaper = InputShaper({shaperType, vTol.*100}, ...
    [obj1.natFreq_model, obj2.natFreq_model],...
     [obj1.dampingRatio_model, obj2.dampingRatio_model], T_s, true);

%% Convolve impulse sequences of both shapers
    % Extract impulse sequences for both shapers from the object properties
```

```
    A1 = obj1.getShaperPulseAmplitudes;
    A2 = obj2.getShaperPulseAmplitudes;
    T1 = obj1.getShaperPulseTimes;
    T2 = obj2.getShaperPulseTimes;
    seq1 = [T1; A1];
    seq2 = [T2; A2];
    % Convolve impulse sequences
    seq = seqconv(seq1', seq2');
    A = seq(:,2)';
    T = seq(:,1)';

%     % Convolution of a series of impulses:
%     k = 1;
%     for j=1:obj1.numPulses
%         for i=1:obj2.numPulses
%             A(k) = A1(j)*A2(i);
%             T(k) = T1(j)+T2(i);
%             k=k+1;
%         end
%     end
%
%     % Sort new impulses sequence based on time locations
%     [T, loc] = sort(T,'ascend');
%     % Sort amplitudes using the mapping from the time locations sort
%     A = A(loc);

%% Assign the impulse sequence to the new shaper object
    convolvedShaper.numPulses = length(A);
    convolvedShaper.pulseAmplitudes = A;
    convolvedShaper.pulseTimes = T;
%% Create a digital respresentation of the new shaper from its
%% continuous-time representation
    digitizeShaper(convolvedShaper);
%% Assign remaining class properties of the new shaper object
    % Prebuffer t_history and y_history up to one time step before the time
    %  of the last impulse in the InputShaper's impulse sequence
    convolvedShaper.t_history = 0:convolvedShaper.T_sampling...
      :(convolvedShaper.pulseTimesDigital(convolvedShaper.numPulsesDigital));
    convolvedShaper.y_history = zeros(1, length(convolvedShaper.t_history));
end % convolveShapers

function seq = seqconv(seq1,seq2)
% SEQUENCECONVOLVE Convolve two continuous sequences together.
%
% seq = seqconv(seq1,seq2)
%
% Convolves two sequences together.
% A Sequence is an n*2 matrix with impulse times (sec) in
% the first column and amplitudes in the second column.
%
% Parameters:
%  seq1, seq2 the two sequences to convolve together.
%
% Returns:
%  seq, the sequence resulting from the convolution.

index = 1;
tempseq = [];
for i=1:length(seq1),
```

```
  for j=1:length(seq2),
     tempseq(index,1) = seq1(i,1)+seq2(j,1);
     tempseq(index,2) = seq1(i,2)*seq2(j,2);
     index = index+1;
  end
end
seq = seqsort(tempseq);

function sortedseq = seqsort(unsortedseq)
% SEQUENCESORT  Sort a continuous sequence into correct
%  order and combine impulses at the same time.
%
% sortedseq = seqsort(unsortedseq)
%
% A sequence is an n*2 matrix with times (sec) in the first
%  column and amplitudes in the second column.
%
% PARAMETERS:
%  unsortedseq is the continuous sequence to sort
%
% RETURNS:
%  sortedseq, the sorted sequence.

[badsortseq,sortingindex] = sort(unsortedseq);

for nn=1:length(unsortedseq),
  goodsortseq(nn,1) = badsortseq(nn,1);
  goodsortseq(nn,2) = unsortedseq(sortingindex(nn,1),2);
end
% Combine impulses with the same times
% (creates seq from goodsortseq)
idx = 1;
seq = [];
seq(1,:) = goodsortseq(1,:);
for nn=2:length(goodsortseq),
  if    goodsortseq(nn,1)==seq(idx,1),
          seq(idx,2) = seq(idx,2)+goodsortseq(nn,2);
  else
    idx = idx+1;
    seq(idx,:) = goodsortseq(nn,:);
  end
end
% Eliminate any impulse with amplitude less than 0.0001
% (creates sortedseq from seq)
idx = 1;
for nn=1:length(seq),
  if    abs(seq(nn,2))>=0.0001,
    sortedseq(idx,:) = seq(nn,:);
    idx = idx+1;
  end
end

function [resVib, wVec] = sensitivityCurve(obj, freqRange, shaperDomainFlag)
% Plots the Sensitivity Curve of the InputShaper object
%
% function [resVib, wVec] = sensitivityCurve(obj, normFreqRange,
% shaperDomainFlag) plots the sensitivity curve of the InputShaper obj. The
% sensitivity curve is used as a measure of the robustness of the input
% shaper to changes in the system parameters. This is typically plotted as
```

```
% residual vibration in percent caused by a command shaped by the input
% shaper on the vertical axis and system natural frequency on the
% horizontal axis. Residual vibration is defined as the ratio (at a given
% frequency and damping ratio) of the vibration caused by the input
% shaper's impulse sequence to the amount of residual vibration caused by
% a single, unity-magnitude impulse.
%
% Inputs:
%   obj                 InputShaper object to plot the sensitivity curve
%                        of.
%   FreqRange           Range of frequencies to plot the residual vibration
%                        at
%   shaperDomainFlag    Input flag used to indicate whether the continuous
%                        or digital representation of the input shaper
%                        should be used to calculate the residual
%                        vibration and plot the sensitivity curve.
%                        For continuous: use 'continuous' or ''.
%                        For digital: use 'digital' or 'discrete'.
%                        Plotting the digital representation is a good way
%                        of checking whether or not the selected sampling
%                        time is low enough to not cause significant
%                        degradation of the vibration-suppressing
%                        properties of the input shaper.
%
% Outputs:
%  resVib               Vector of residual vibration, in percent.
%  wVec                 Vector of frequencies, in rad/s, at which the
%                        residual vibration was plotted. The
%                        residual vibration at a given index of resVib
%                        corresponds to the frequency in wVec at that
%                        index.

    if(length(freqRange) ~= 2)
        error('Range of frequencies to plot must be a 1x2 vector')
    end
    % System parameters
    if strcmp(obj.shaperType, 'SI') || strcmp(obj.shaperType, 'SI')
        w_n = (obj.natFreq_model(1)+obj.natFreq_model(2))/2;
        zeta = obj.dampingRatio_model(1);
    elseif strcmp(obj.shaperType, 'si2mode') || strcmp(obj.shaperType, 'SI2Mode')
        w_n_1 = (obj.natFreq_model(1)+obj.natFreq_model(2))/2;
        w_n_2 = (obj.natFreq_model(3)+obj.natFreq_model(4))/2;
        if w_n_1 > w_n_2
            w_n = w_n_2;
        else
            w_n = w_n_1;
        end
        zeta_1 = obj.dampingRatio_model(1);
        zeta_2 = obj.dampingRatio_model(2);
        if zeta_1 > zeta_2
            zeta = zeta_2;
        else
            zeta = zeta_1;
        end
    else
        w_n = obj.natFreq_model(1);
        zeta = obj.dampingRatio_model(1);
    end
%     % Calculate damped natural frequency
```

```
%     w_d = w_n.*sqrt(1-zeta.^2);
    % Determine number of frequencies to plot
    divs = 1000; % points per 1 unit of frequency
    numFreqs = round(divs.*(freqRange(2)-freqRange(1)));
    % Create vector of frequencies at which the residual vibration
    %  will be plotted
    wVec = linspace(freqRange(1), freqRange(2), numFreqs);
    % Prebuffer resVib
    resVib = zeros(1, numFreqs);
    for k = 1:numFreqs
        resVib(k) = residualVibration(obj, wVec(k), zeta, shaperDomainFlag);
    end
    % Plot
    if nargout == 0
        figure;
        plot(wVec, resVib);
        % Figure formatting
        xlabel('Frequency (rad/s)');
        xlim([freqRange(1), freqRange(2)]);
        ylabel('Percentage Residual Vibration');
        ylim([0, ceil(max(resVib))]);
        titleStrLine1 = ['Shaper Type: ', obj.shaperType];
        titleStrLine2 = ['Modeled Natural Frequency and Damping Ratio: \omega_m = ',...
         num2str(w_n), ' rad/s, \zeta_m = ', num2str(zeta)];
        if strcmp(shaperDomainFlag, 'continuous') || strcmp(shaperDomainFlag, '')
            title({'Sensitivity Curve', titleStrLine1, titleStrLine2})
        elseif strcmp(shaperDomainFlag, 'digital') || strcmp(shaperDomainFlag, 'discrete')
            title({'Sensitivity Curve for the Shaper''s Digital Representation', ...
            titleStrLine1, titleStrLine2})
        else
            error('Input shaperDomainFlag is not a recognized flag.')
        end
    end % else Don't plot, return function outputs resVib and wVec
end % sensitivityCurve

function resVib = residualVibration(obj, w_n, zeta, shaperDomainFlag)
% Returns the residual vibration resulting from the input shaper's
% impulse sequence at the given frequency w_n and damping ratio zeta as a
% percentage of the amount of residual vibration caused by a single,
% unity-magnitude impulse

    % % Calculate damped natural frequency
    w_d = w_n.*sqrt(1-zeta.^2);
    if strcmp(shaperDomainFlag, 's') || strcmp(shaperDomainFlag, 'continuous')...
     || strcmp(shaperDomainFlag, '')
    % Use continuous (normal) shaper representation

        C = zeros(1, obj.numPulses);
        S = zeros(1, obj.numPulses);
        for k = 1:obj.numPulses
        C(k) = obj.pulseAmplitudes(k).*exp(zeta.*w_n.*obj.pulseTimes(k))...
        .*cos(w_d.*obj.pulseTimes(k));
        S(k) = obj.pulseAmplitudes(k).*exp(zeta.*w_n.*obj.pulseTimes(k))...
        .*sin(w_d.*obj.pulseTimes(k));
        end
        C = sum(C);
        S = sum(S);
        % Calculate residual vibration (normalized as a percentage of
        %  the vibration caused by a single, unity-magnitude impulse)
```

```
        resVib = 100.*exp(-zeta.*w_n.*obj.pulseTimes(obj.numPulses))...
        .*sqrt(C.^2+S.^2); % [%]
    elseif strcmp(shaperDomainFlag, 'z') || strcmp(shaperDomainFlag, 'digital')...
     || strcmp(shaperDomainFlag, 'discrete')
    % Use digital shaper representation
        C = zeros(1, obj.numPulsesDigital);
        S = zeros(1, obj.numPulsesDigital);
        for k = 1:obj.numPulsesDigital
        C(k) = obj.pulseAmplitudesDigital(k).*exp(zeta.*w_n.*obj.pulseTimesDigital(k))...
        .*cos(w_d.*obj.pulseTimesDigital(k));
        S(k) = obj.pulseAmplitudesDigital(k).*exp(zeta.*w_n.*obj.pulseTimesDigital(k))...
        .*sin(w_d.*obj.pulseTimesDigital(k));
        end
        C = sum(C);
        S = sum(S);
        % Calculate residual vibration (normalized as a percentage of
        %  the vibration caused by a single, unity-magnitude impulse)
        resVib = 100.*exp(-zeta.*w_n.*obj.pulseTimesDigital(obj.numPulsesDigital))...
        .*sqrt(C.^2+S.^2); % [%]
    else
        error('Input shaperDomainFlag is not a recognized flag.')
    end
end % residualVibration

function disp(obj)
% Overloads MATLAB default disp.m

    if strcmp(obj.shaperType, 'SI') || strcmp(obj.shaperType, 'SI')
        fprintf('Shaper Type:\t\t\t\t\t%s\n', obj.shaperType);
        fprintf('Natural Frequency Range: \t\t%0.5g - %0.5g [rad/s]\n',...
         obj.natFreq_model(1), obj.natFreq_model(2))
        fprintf('Design Damping Ratio: \t\t\t%0.5g\n', obj.dampingRatio_model(1))
        fprintf('Tolerable Residual Vibration: \t%0.5g%% \n', obj.vTol(1)*100);
    elseif strcmp(obj.shaperType, 'si2mode') || strcmp(obj.shaperType, 'SI2Mode')
        fprintf('Shaper Type:\t\t\t\t\t%s\n', obj.shaperType);
        fprintf('Natural Frequency Range (Mode 1): \t\t%0.5g ...
        - %0.5g [rad/s]\n', obj.natFreq_model(1), obj.natFreq_model(2))
        fprintf('Design Damping Ratio (Mode 1): \t\t\t%0.5g\n', obj.dampingRatio_model(1))
        fprintf('Tolerable Residual Vibration (Mode 1): \t%0.5g%% \n', obj.vTol(1)*100);
        fprintf('Natural Frequency Range (Mode 2): \t\t%0.5g ...
        - %0.5g [rad/s]\n', obj.natFreq_model(3), obj.natFreq_model(4))
        fprintf('Design Damping Ratio (Mode 2): \t\t\t%0.5g\n', obj.dampingRatio_model(2))
        fprintf('Tolerable Residual Vibration (Mode 2): \t%0.5g%% \n', obj.vTol(2)*100);
    else
        if obj.convolved
            fprintf('Convolved Shaper: %s\n The properties of each shaper that were...
             convolved to form this one are listed separately:\n', obj.shaperType)
            for i = 1:length(obj.natFreq_model)
                fprintf(' Shaper Type (Shaper %d):\t\t\t\t\t%s\n', i,...
                 obj.convolvedShaperTypes{i});
                fprintf(' Design Natural Frequency (Shaper %d): \t\t%0.5g [rad/s]\n', i,...
                 obj.natFreq_model(i))
                fprintf(' Design Damping Ratio (Shaper %d): \t\t\t%0.5g\n', i,...
                 obj.dampingRatio_model(i))
        if ~(strcmp(obj.convolvedShaperTypes{i}, 'ei') || ...
           strcmp(obj.convolvedShaperTypes{i}, 'EI') ||...
           strcmp(obj.convolvedShaperTypes{i}, 'si') ||...
           strcmp(obj.convolvedShaperTypes{i}, 'SI') ||...
           strcmp(obj.convolvedShaperTypes{i}, 'si2mode') ||...
```

```
            strcmp(obj.convolvedShaperTypes{i}, 'SI2Mode') ||...
            strcmp(obj.convolvedShaperTypes{i}, 'umei') ||...
             strcmp(obj.convolvedShaperTypes{i}, 'UMEI'))
                    fprintf(' Tolerable Residual Vibration (Shaper %d): \tOnly...
                     applicable for EI and SI type shapers. \n', i);
                else
                    fprintf(' Tolerable Residual Vibration (Shaper %d): \t%0.5g%%...
                     \n', i, obj.vTol(i)*100);
                end
            end
            fprintf(' ')
        else
            fprintf('Shaper Type:\t\t\t\t\t%s\n', obj.shaperType);
            fprintf('Design Natural Frequency: \t\t%0.5g [rad/s]\n', obj.natFreq_model)
            fprintf('Design Damping Ratio: \t\t\t%0.5g\n', obj.dampingRatio_model)
            if ~(strcmp(obj.shaperType, 'ei') || strcmp(obj.shaperType, 'EI') || ...
            strcmp(obj.shaperType, 'si') || strcmp(obj.shaperType, 'SI') || ...
            strcmp(obj.shaperType, 'si2mode') || strcmp(obj.shaperType, 'SI2Mode') ||...
             strcmp(obj.shaperType, 'umei') || strcmp(obj.shaperType, 'UMEI'))
                fprintf('Tolerable Residual Vibration: \tOnly applicable for EI and ...
                SI type shapers. \n');
            else
                fprintf('Tolerable Residual Vibration: \t%0.5g%% ...
                \n', obj.vTol*100);
            end
        end
    end
    fprintf('Impulse Sequence:\n')
    fprintf('  Times \tAmplitudes\n')
    for i = 1:obj.numPulses
        fprintf('   %#0.4g \t %#0.3g\n', obj.pulseTimes(i), obj.pulseAmplitudes(i))
    end
end % disp

function shaperOut = getShaper(obj)
% Returns all shaper properties as a struct
    shaperOut.shaperType = obj.shaperType;
    shaperOut.shaperTypeConvolved = obj.convolvedShaperTypes;
    shaperOut.natFreq_model = obj.natFreq_model;
    shaperOut.dampingRatio_model = obj.dampingRatio_model;
    shaperOut.numPulses = obj.numPulses;
    shaperOut.pulseAmplitudes = obj.pulseAmplitudes;
    shaperOut.pulseTimes = obj.pulseTimes;
end % getShaper

function typeOut = getShaperType(obj)
% Returns type of shaper (a string)
    typeOut = obj.shaperType;
end % getShaperType

function pulseAmplitudesOut = getShaperPulseAmplitudes(obj)
% Returns shaper pulseAmplitudes
    pulseAmplitudesOut = obj.pulseAmplitudes;
end % getShaperPulseAmplitudes

function pulseTimesOut = getShaperPulseTimes(obj)
% Returns shaper pulseTimes
    pulseTimesOut = obj.pulseTimes;
end % getShaperPulseTimes
```

```
function numPulsesOut = getShaperNumPulses(obj)
% Returns shaper numPulses
    numPulsesOut = obj.numPulses;
end % getShaperNumPulses

function [w_model, zeta_model] = getModelingParameters(obj)
% Returns modeling natural frequency and damping ratio of shaper
    w_model = obj.natFreq_model;
    zeta_model = obj.dampingRatio_model;
end % getModelingParameters

function typeConvolvedOut = getShaperTypeConvolved(obj)
% Return stypes of shapers in the convolved shaper (a cell array
%  that is a list of strings indicating shaper type)
    typeConvolvedOut = obj.convolvedShaperTypes;
end % getShaperType

function digitalShaperOut = getDigitalShaper(obj)
% Returns all properties of the digitized shaper as a struct
    digitalShaperOut.shaperType = obj.shaperType;
    digitalShaperOut.shaperTypeConvolved = obj.convolvedShaperTypes;
    digitalShaperOut.natFreq_model = obj.natFreq_model;
    digitalShaperOut.dampingRatio_model = obj.dampingRatio_model;
    digitalShaperOut.numPulsesDigital = obj.numPulsesDigital;
    digitalShaperOut.pulseAmplitudesDigital = obj.pulseAmplitudesDigital;
    digitalShaperOut.pulseTimesDigital = obj.pulseTimesDigital;
end % getDigitalShaper

function [exactshaper,exitflag] = si_fmincon(ShapLength,ShaperFlag,...
    neg,X0,fmin,fmax,damp,V_tol,deltaT)
% [exactshaper,exitflag] = si_fmincon(ShapLength,ShaperFlag,...
%    neg,X0,fmin,fmax,damp,V_tol,deltaT)
%
% function to determine amplitudes and time locations
% for an SI, UM-SI, or SNA-SI shaper
%
% necessary files: optimization toolbox
%
% Can be either a SNA, UM, or positive shaper, depending on the
% value of UMFlag.
%
% The user enters desired frequency range to suppress vibration;
% the program calculates insensitivity values and generates input shaper.
%
% ShaperLength = number of impulses, will be redefined if X0 = 0
% ShaperFlag = 0 for positive, 1 for UM shaper, 2 for SNA
% neg = max negative amplitude for SNA shaper, ignored otherwise
% X0 = initial guess - [a1 a2 ... an t1 t2 ... tn] or 0 if unknown
% fmin = lower frequency of range to suppress vibration within (Hz)
% wmin = lower frequency of range to suppress vibration within (rad/s)
% fmax = upper frequency of range to suppress vibration within (Hz)
% wmax = upper frequency of range to suppress vibration within (rad/s)
% damp = damping ratio
% V_tol = vibration level tolerated (0.05 = 5%)
% deltaT = sampling period (s)
%
% This function generates the exact impulse sequence.
```

```
ShaperLength = ShapLength;  % if X0 = 0 this will be redefined during intial guess
Vtol = V_tol;
UMFlag = ShaperFlag;
zeta = damp;
if UMFlag == 1;
    neg = 1;
end
%***** Determine median frequecny of range to be suppressed
%***** and Insensitivity.
wmin = 2*pi*fmin;                   % convert to rad/s
wmax = 2*pi*fmax;                   % convert to rad/s
wn = (wmin+wmax)/2;                 % median frequency
Ins = (wmax-wmin)/wn;               % Insensitivity
%***** Calculate period; use for initial time guess ****
T=2*pi/wn;
if X0==0                    % if I.C.'s unknown, create X0
    if UMFlag==0            % For pos shaper
        if Ins <= 0.3992
            ShaperLength = 3;       % EI
            X0 = [0.28 0.46 0.28 0 T/2 T];
        elseif Ins <= 0.7;%0.7262
            ShaperLength = 4;       % 2hump EI
            X0 = [0.16 0.34 0.34 0.16 0 T/2 T 1.5*T];
        elseif Ins <= 0.9654
            ShaperLength = 5;       % 3 hump EI
            X0 = [0.0625 0.25 0.375 0.25 0.0625 0 T/2 T 1.5*T 2*T];
        else
            disp('Code only work for positive shapers up to I(5%) = 0.9654')
        end
    else      % for negative shapers
        if Ins <= 0.0333*neg^2-0.0672*neg+0.3956
            ShaperLength = 5;
            t(1) = 0;
            t(2) = -0.0091*neg^2 - 0.041*neg + 0.1466;
            t(3) = 0.0923*neg^2 - 0.2134*neg + 0.4881;
            t(4) = 0.1933*neg^2 - 0.3856*neg + 0.8297;
            t(5) = 0.1843*neg^2 - 0.4267*neg + 0.9763;
            a(1) = 0.4067*neg^3 - 0.5703*neg^2 + 0.9112*neg + 0.25;
            a(2) = -neg;
            a(3) = -0.8135*neg^3 + 1.1401*neg^2 + 0.1782*neg + 0.4998;
            a(4) = -neg;
            a(5) = a(1);
            X0 = [a t*T]; % EI node
        elseif Ins <= 0.0604*neg^2 - 0.1061*neg + 0.7186;
            ShaperLength = 7;
           t(1) = 0;
           t(2) = -0.0698*neg^2 + 0.0185*neg + 0.1171;
           t(3) = 0.1952*neg^2 - 0.3032*neg + 0.4949;
           t(4) = 0.1571*neg^2 - 0.3053*neg + 0.7294;
           t(5) = 0.1190^neg^2 - 0.3075*neg + 0.9639;
           t(6) = 0.3839*neg^2 - 0.6290*neg + 1.3416;
           t(7) = 0.3138*neg^2 - 0.6102*neg + 1.4587;
           a(1) = 1.6343*neg^4 - 2.4423*neg^3 + 1.0978*neg^2 + 0.5355*neg + 0.1772;
           a(2) = -neg;
           a(3) = -1.6343*neg^4 + 2.4423*neg^3 - 1.0978*neg^2 + 0.9645*neg +0.3228;
           a(4) = -neg;
           a(5) = a(3);
           a(6) = -neg;
           a(7) = a(1);
```

```
            X0 = [a t*T]; % 2Hump EI Node
        elseif Ins <= .2895*neg^4 - 0.6258*neg^3 + 0.5211*neg^2 - 0.2382*neg + 0.9654
            ShaperLength = 9;
            t(1) = 0;
            t(2) = -0.0856*neg^2 + 0.0235*neg + 0.1148;
            t(3) = 0.3095*neg^2 - 0.4085*neg + 0.4947;
            t(4) = 0.1438*neg^2 - 0.3033*neg + 0.7011;
            t(5) = 0.2201*neg^2 - 0.3867*neg + 0.9686;
            t(6) = 0.2973*neg^2 - 0.4709*neg + 1.237;
            t(7) = 0.1308*neg^2 - 0.3648*neg + 1.4424;
            t(8) = 0.5266*neg^2 - 0.7978*neg + 1.8226;
            t(9) = 0.4407*neg^2 - 0.7738*neg + 1.9372;
            a(1) = 0.3615*neg^5 + 2.2773*neg^4 - 4.501*neg^3 + 2.5652*neg^2 ...
            + 0.1458*neg + 0.1537;
            a(2) = -neg;
            a(3) = -4.7821*neg^5 + 10.014*neg^4 - 7.4091*neg^3 + 2.4361*neg^2 ...
            + 0.492*neg + 0.2475;
            a(4) = -neg;
            a(5) = 8.7667*neg^5 - 24.359*neg^4 + 23.578*neg^3 - 9.8884*neg^2 ...
            + 2.7027*neg + 0.1989;
            a(6) = -neg;
            a(7) = a(3);
            a(8) = -neg;
            a(9) = a(1);
            X0 = [a t*T]; %3 Hump EI Node
        elseif Ins <= 1.2
            ShaperLength = 11; % ???
            X0 = [1 -1 1 -1 1 -1 1 -1 1 -1 1 ...
                0 0.0427*T 0.4242*T 0.5635*T 0.8305*T 1.0976*T 1.2371*T...
                1.6189*T 1.6619*T 1.85*T 2.4*T];
        else
            disp('Code only works for negative shapers up to I(5%) = 1.2')
        end
    end
end
% Increase ShapLen temporarily... it gets reduced during first iteration of
%   optimization
if UMFlag == 0
    ShaperLength = ShaperLength + 1;
else
    ShaperLength = ShaperLength + 2;
end
% Create zero vectors to store amplitudes and times
% These also trigger the while loop to execute the first time
amps=zeros(1,ShaperLength);
tms=zeros(1,ShaperLength);
seek_solution = 1;      % flag for while loop about solution
%   Check for zero amplitudes and repeated impulse times
%   If they exist, decrease the shaper length by one and solve again
while (seek_solution == 1)
    if UMFlag == 0
        %Reduce number of impulses if min(amps) is small
%         ShaperLength = ShaperLength-1;
        ShaperLength = length(X0)/2;
    else
        ShaperLength = ShaperLength-2;
    end
    % Define Linear Constraints based on UMFlag
    if UMFlag == 0      % All Positive Impulses
```

```
    %Linear Equality Constraints
    Aeq=zeros(2,2*ShaperLength);      % define to speed comp.
    for ii=1:ShaperLength
        Aeq(1,ii)=1;                  % all impulses sum to one
    end
    Aeq(2,ShaperLength+1)=1;          % first impulse at t=0
    Beq=[1;0];
    %Linear Inequality Constraints
    A=zeros(ShaperLength-1,2*ShaperLength); % define to speed comp.
    B=[zeros(3*ShaperLength-1,1)];
    % loop is constraints that t(i+1) > t(i)
    for ii=1:ShaperLength-1
        A(ShaperLength+ii,ShaperLength+ii)=1;
        A(ShaperLength+ii,ShaperLength+1+ii)=-1;
    end
    for ii=1:ShaperLength
        A(ii,ii)=-1;                                    %All Imp.  pos
        A((2*ShaperLength+ii-1),ShaperLength+ii)=-1;    %All times pos
    end
elseif UMFlag == 1  % Unity Magnitude
    %Linear Equality Constraints
    Aeq=zeros(ShaperLength+2,2*ShaperLength); % define to speed comp.
    for ii=1:ShaperLength
        Aeq(1,ii)=1;                      % all impulses sum to one
        Aeq(ii+1,ii)=(-1)^(ii+1);         % Alternating signs and unity magnitude constraints
    end
    Aeq(ShaperLength+2,ShaperLength+1)=1;  % first impulse at t=0
    Beq=[1;ones(ShaperLength,1);0];
    %Linear Inequality Constraints
    A=zeros(2*ShaperLength-1,2*ShaperLength);   % define to speed comp.
    B=[zeros(2*ShaperLength-1,1)];
    % loop is constraints that t(i+1) > t(i)
    for ii=1:ShaperLength-1
        A(ii,ShaperLength+ii)=1;
        A(ii,ShaperLength+1+ii)=-1;
    end
    for ii=1:ShaperLength
        A((ShaperLength+ii-1),ShaperLength+ii)=-1;    %All times pos
    end
elseif UMFlag == 2  % Specified Negative Amplitude
    %Linear Equality Constraints
    count = 0;                            % counter for #neg imp.
    for ii=2:2:ShaperLength
        count = count+1;
    end
    Aeq=zeros(2+count,2*ShaperLength);      % define to speed comp.
    for ii=1:ShaperLength
        Aeq(1,ii)=1;                      % all impulses sum to one
    end
    count = 0;
    for ii=2:2:ShaperLength
        count = count + 1;
        Aeq(1+count,ii) = -1/neg;         % alternating negative amps
    end
    Aeq(2+count,ShaperLength+1)=1;          % first impulse at t=0
    Beq=[1;ones(count,1);0];
     %Linear Inequality Constraints
    A=zeros(2*ShaperLength-1,2*ShaperLength);
    B=[zeros(2*ShaperLength-1,1);neg*ones(ShaperLength,1);...
```

```
    ones(ShaperLength,1);zeros(ShaperLength,1);ones(ShaperLength,1)];
    % loop is constraints that t(i+1) > t(i)
    for ii=1:ShaperLength-1
        A(ii,ShaperLength+ii)=1;
        A(ii,ShaperLength+1+ii)=-1;
    end
    for ii=1:ShaperLength
        A((ShaperLength+ii-1),ShaperLength+ii)=-1;    %All times are pos
        A(2*ShaperLength+ii-1,ii) = -1;
        A(3*ShaperLength+ii-1,ii) = 1;
        % running total >= 0
        A(4*ShaperLength+ii-1,1:ii) = -1;
        A(5*ShaperLength+ii-1,1:ii) = 1;
    end
else
    error('ZV2mode:UMFlagCheck',...
    '\nPlease enter a correct ShaperFlag value: \n   ShaperFlag = 0 -> ...
    Only positive impulses \n   ShaperFlag = 1 -> Unity Magnitude shaper \n ...
      ShaperFlag = 2 -> Specified Negative Amplitude')
end
%******* Set maximum number of iterations ****************
options = optimset('Algorithm','active-set','MaxIter',5000*ShaperLength,...
    'Display','off','MaxFunEvals',5000*ShaperLength,'TolFun',1e-9);
%******* make call to optimizer ***************************
%x=fmincon(@myfun,x0,A,b,Aeq,beq,lb,ub,@mycon) - format of call to
%fmincon
[x,fval,exitflag]=fmincon(@si_fmincon_fun,X0,A,B,Aeq,Beq,...
    [],[],@si_fmincon_const,options);
i=2*ShaperLength;
clear tms_X0 amps_X0
amps_X0=x(1:ShaperLength);
tms_X0=x(ShaperLength+1:i);
% uncomment to see incremental solutions
% shaper = [tms_X0' amps_X0']
% check for zero amplitudes or identical times
% if so, create new intial guess from current solution and re-solve
if (min(abs(amps_X0)) < 1e-4) | (min(diff(tms_X0)) < 1e-4)
    disp(' ')
    disp('Nearly-identical times or zero-amplitude impulses found. ')
    disp('Reducing the number of impulses and re-solving...')
    disp(' ')
    %Generate next initial guess
    if UMFlag == 0 || UMFlag == 2
        % check for impulses occuring at the same time
        % if so, remove one, sum amplitudes, and resolve
            for ii = length(tms_X0)-1:-1:1;
                if tms_X0(ii+1)-tms_X0(ii) < 1e-4
                    tms_X0(ii) = [];
                    amps_X0(ii+1) = amps_X0(ii)+amps_X0(ii+1);
                    amps_X0(ii) = [];
                end
            end
            for ii = length(amps_X0):-1:1
                if abs(amps_X0(ii)) < 1e-4
                    amps_X0(ii) = [];
                    tms_X0(ii) = [];
                end
            end
        % create new initial Guess
```

```
            X0=[amps_X0 tms_X0];
          else % "smart" simplification for negative shapers
            for ii = length(tms_X0)-1:-1:1;
                if tms_X0(ii+1)-tms_X0(ii) < 1e-3
                    tms_X0(ii:ii+1) = [];
                    amps_X0(ii:ii+1) = [];
                end
            end
            X0=[amps_X0 tms_X0];
            ShaperLength = length(X0)/2;
        end
    else
        % there are zero amp impulses or repeated times
        % use current solution
        seek_solution = 0;
    end
end
% Check for convergence and assign shaper
if exitflag <= 0
    disp('ERROR: optimization did not converge')
    exactshaper = [];
    shaper = [];
else
    i=2*ShaperLength;
    amps=x(1:ShaperLength);
    time=x(ShaperLength+1:i);
    time(1) = 0;
    exactshaper = [time' amps'];
    % shaper = digseq(exactshaper,deltaT);
end
%%%%%  Cost Function to Minimize  %%%%%%
function [f,g]=si_fmincon_fun(x)
i=2*ShaperLength;           % length of x
f=x(i);                     % function to be minimized (time of tn)
end     %end cost function
%%%%%  Nonlinear Constraints  %%%%%%%%
function [c,ceq]=si_fmincon_const(x)
i=2*ShaperLength;               % length of x
ceq=[];                         % Nonlinear equality constraints
StepMax=50;                     % # of points at which vibration is limited
% Generate constraints to satisfy insensitivity parameters
for jj=1:StepMax
    w(jj)=wn*((1-Ins/2)+Ins/(StepMax-1)*(jj-1));
    w_damp(jj) = w(jj)*sqrt(1-zeta^2);
    % cosine terms of vibration equation
    csum=sum(x(1:i/2).*exp(zeta*w(jj)*x(i/2+1:i))...
        .*cos(-w_damp(jj)*x(i/2+1:i)));
    % sine terms of vibration equation
    ssum=sum(x(1:i/2).*exp(zeta*w(jj)*x(i/2+1:i))...
        .*sin(-w_damp(jj)*x(i/2+1:i)));
    % if last impulse amplitude is zero, measure vib. at previous impulse
    if abs(x(i/2)) < 1e-6
        g(jj,:)=exp(-zeta*w(jj)*x(i-1))*sqrt(csum^2+ssum^2)-Vtol;
    else
        g(jj,:)=exp(-zeta*w(jj)*x(i))*sqrt(csum^2+ssum^2)-Vtol;
    end
end
% uncomment next line to watch convergence
% shap = [x(ShaperLength+1:2*ShaperLength)' x(1:ShaperLength)']
```

```
c=g;
end     %end nonlinear constraint function
end % end main function

function [exactshaper,exitflag] = si2mode_fmincon(ShapLength,ShaperFlag,...
    neg,X0,f1min,f1max,damp1,V_tol_1,f2min,f2max,damp2,V_tol_2,deltaT)
% [exactshaper,exitflag] = si2mode(ShapLength,ShaperFlag,...
%    neg,X0,f1min,f1max,damp1,V_tol_1,f2min,f2max,V_tol_2,deltaT)
%
% Function determines the amplitudes and time locations
% for a Two-mode SI shaper.
%
% Can be either a SNA, UM, or positive shaper, depending on UMFlag value.
% (As of 11/8/10 - negative and UM shapers sometimes fail to converge)
%
% Necessary auxilliary files: optimization toolbox
%       EI2M_und.m and negumei.m also needed if no initial guess is given
%
% ShapLength = number of impulses - Redefined if X0 = 0
% ShaperFlag = 0 for positive, 1 for UM shaper, 2 for SNA
% neg = desired negative amplitude for SNA shaper, no function othersise
% X0 = initial guess - [a1 a2 ... an t1 t2 ... tn] or 0 if unknown
% deltaT = sampling period (s)
%
% FOR FIRST MODE
% f1min = lower frequency of range to suppress vibration within (Hz)
% f1max = upper frequency of range to suppress vibration within (Hz)
% damp1 = damping ratio
% V_tol_1 = vibration level tolerated (0.05 = 5%)
%
% FOR SECOND MODE
% f2min = lower frequency of range to suppress vibration within (Hz)
% f2max = upper frequency of range to suppress vibration within (Hz)
% damp2 = damping ratio
% V_tol_2 = vibration level tolerated (0.05 = 5%)

% Convert user input to function variables
ShaperLength = ShapLength;
UMFlag = ShaperFlag;
Vtol1 = V_tol_1;
zeta1 = damp1;
Vtol2 = V_tol_2;
zeta2 = damp2;

%***** Determine median frequecny of range to be suppressed
%***** and Insensitivity for FIRST mode.
f1 = (f1min+f1max)/2;                   % median freq in Hz
w1min = 2*pi*f1min;                     % convert to rad/s
w1max = 2*pi*f1max;                     % convert to rad/s
wn1 = (w1min+w1max)/2;                  % median frequency
Ins1 = (w1max-w1min)/wn1;               % Insensitivity
%*****************************************************

%***** Determine median frequecny of range to be suppressed
%***** and Insensitivity for SECOND mode.
f2 = (f2min+f2max)/2;                   % median freq in Hz
w2min = 2*pi*f2min;                     % convert to rad/s
w2max = 2*pi*f2max;                     % convert to rad/s
wn2 = (w2min+w2max)/2;                  % median frequency
```

```
Ins2 = (w2max-w2min)/wn2;               % Insensitivity
%*****************************************************

% Generate intial guess if none given
if X0 == 0
    if UMFlag == 0
    % Use 2-mode EI as Initial Guess
        [temp,EI2mode] = ei2m_und(f1,f2,Vtol1,Vtol2,deltaT);
        X0 = [EI2mode(:,2)' EI2mode(:,1)'];
        ShaperLength = length(EI2mode);
    else
    % Use 2-mode UM-EI (Vtols = 0.05) as Initial Guess
        [temp,UMEImode1] = negumei(f1,zeta1,0.05,deltaT);
        [temp,UMEImode2] = negumei(f2,zeta2,0.05,deltaT);
        UMEI_tot = seqconv(UMEImode1,UMEImode2);
        X0 = [UMEI_tot(:,2)' UMEI_tot(:,1)'];
        ShaperLength = length(UMEI_tot);
    end
end

% seek_solution = 1 until solution is found
% will remain = 1 if zero amplitudes and repeated impulse times are found
seek_solution = 1;

while (seek_solution == 1)

    %**** Generate linear constraints according to shaper-type
    if UMFlag == 0   % Positive Impulse Shapers

        %**** Linear Equality Constraints (of form Aeq*x = Beq)
        Aeq=zeros(2,2*ShaperLength);
        for ii=1:ShaperLength
            Aeq(1,ii)=1;                 % all impulses sum to one
        end
        Aeq(2,ShaperLength+1)=1;         % first impulse at t=0
        Beq=[1;0];
        %**** Linear Inequality Constraints (of form A*x <= B)
        A=zeros(ShaperLength-1,2*ShaperLength); % define matrix to speed comp.
        % loop is constraints that t(i+1) > t(i)
        for ii=1:ShaperLength-1
            A(ShaperLength+ii,ShaperLength+ii)=1;
            A(ShaperLength+ii,ShaperLength+1+ii)=-1;
        end
        for ii=1:ShaperLength
          A(ii,ii)=-1;                                    %All Imp. are pos
          A((2*ShaperLength+ii-1),ShaperLength+ii)=-1;    %All times are pos
        end
        B=[zeros(3*ShaperLength-1,1)];
      elseif UMFlag == 1  % Unity Magnitude
        %**** Linear Equality Constraints (of form Aeq*x = Beq)
        Aeq=zeros(ShaperLength+2,2*ShaperLength); % define to speed comp.
        for ii=1:ShaperLength
          Aeq(1,ii)=1;                   % all impulses sum to one
          Aeq(ii+1,ii)=(-1)^(ii+1);% Alternating signs and unity magnitude constraints
        end
        Aeq(ShaperLength+2,ShaperLength+1)=1;  % first impulse at t=0
        Beq=[1;ones(ShaperLength,1);0];
        %**** Linear Inequality Constraints (of form A*x <= B)
        A=zeros(2*ShaperLength-1,2*ShaperLength);   % define to speed comp.
```

```
    B=[zeros(2*ShaperLength-1,1)];
    % loop is constraints that t(i+1) > t(i)
    for ii=1:ShaperLength-1
        A(ii,ShaperLength+ii)=1;
        A(ii,ShaperLength+1+ii)=-1;
    end
    for ii=1:ShaperLength
        A((ShaperLength+ii-1),ShaperLength+ii)=-1;    %All times pos
    end
  elseif UMFlag == 2  % Specified Negative Amplitude
    %**** Linear Equality Constraints (of form Aeq*x = Beq)
    count = 0;                                % counter for #neg imp.
    for ii=2:2:ShaperLength
        count = count+1;
    end
    Aeq=zeros(2+count,2*ShaperLength);      % define to speed comp.
    for ii=1:ShaperLength
        Aeq(1,ii)=1;                          % all impulses sum to one
    end
    count = 0;
    for ii=2:2:ShaperLength
        count = count + 1;
        Aeq(1+count,ii) = -1/neg;           % alternating negative amps
    end
    Aeq(2+count,ShaperLength+1)=1;          % first impulse at t=0
    Beq=[1;ones(count,1);0];
    %**** Linear Inequality Constraints (of form A*x <= B)
    % define matrices to speed comp.
    A=zeros(2*ShaperLength-1,2*ShaperLength);
    B=[zeros(2*ShaperLength-1,1);neg*ones(ShaperLength,1);...
        ones(ShaperLength,1);zeros(ShaperLength,1);ones(ShaperLength,1)];
    % loop is constraints that t(i+1) > t(i)
    for ii=1:ShaperLength-1
        A(ii,ShaperLength+ii)=1;
        A(ii,ShaperLength+1+ii)=-1;
    end
    for ii=1:ShaperLength
        % All times are pos
        A((ShaperLength+ii-1),ShaperLength+ii)=-1;
        A(2*ShaperLength+ii-1,ii) = -1;
        A(3*ShaperLength+ii-1,ii) = 1;
        % running total >= 0
        A(4*ShaperLength+ii-1,1:ii) = -1;
        A(5*ShaperLength+ii-1,1:ii) = 1;
    end
else
   error('SI2mode:UMFlagCheck',...
        '\nPlease enter a correct ShaperFlag value: \n   ShaperFlag = 0...
         -> Only positive impulses \n   ShaperFlag = 1 ->...
          Unity Magnitude shaper \n   ShaperFlag = 2 -> Specified Negative Amplitude')
end % End formation of linear constraints
%******* Set optimization options  ****************
options = optimset('MaxIter',1e4,'Display','off',...
    'Algorithm','active-set','MaxFunEvals',1e4,'TolFun',1e-4);

%***** make call to optimizer ***************************
%x=fmincon(@myfun,x0,A,b,Aeq,beq,lb,ub,@mycon) - format of call to fmincon
[x,fval,exitflag]=fmincon(@si2mode_fmincon_fun,X0,A,B,Aeq,Beq,...
    [],[],@si2mode_fmincon_const,options);
```

```
    i=2*ShaperLength;
    clear tms_X0 amps_X0
    amps_X0=x(1:ShaperLength);
    tms_X0=x(ShaperLength+1:i);
% %     Uncomment below for helpful debugging
%         disp(' ')
%         disp(['exitflag = ',num2str(exitflag)])
%         disp('Run "help fmincon" for meaning of flag')
%         disp(' ')
%         disp('fmincon solution - may have closely spaced times and 0 ampl. pulses')
%         fmincon_solution = [tms_X0' amps_X0']
% %     End debugging comments
    % check for zero amplitudes or nearly-identical times
    % if so, create new intial guess from current solution and re-solve
    if (min(abs(amps_X0)) < 1e-3) || (min(diff(tms_X0)) < 1e-3)
        disp(' ')
        disp('Nearly-identical times or zero-amplitude impulses found. ')
        disp('Reducing the number of impulses and re-solving...')
        disp(' ')
        % Generate next initial guess
        % "smart" solution simplification for positive shapers
        if UMFlag == 0 || UMFlag == 2
            % check for impulses occuring at the same time
            % if so, remove one, sum amplitudes, and resolve
                for ii = length(tms_X0)-1:-1:1;
                    if tms_X0(ii+1)-tms_X0(ii) < 1e-3
                        tms_X0(ii) = [];
                        amps_X0(ii+1) = amps_X0(ii)+amps_X0(ii+1);
                        amps_X0(ii) = [];
                    end
                end

                for ii = length(amps_X0):-1:1
                    if abs(amps_X0(ii)) < 1e-3
                        amps_X0(ii) = [];
                        tms_X0(ii) = [];
                    end
                end
            % create new initial Guess
            X0=[amps_X0 tms_X0];
            ShaperLength = length(X0)/2;
        else % "smart" simplification for negative shapers
            for ii = length(tms_X0)-1:-1:1;
                if tms_X0(ii+1)-tms_X0(ii) < 1e-3
                    tms_X0(ii:ii+1) = [];
                    amps_X0(ii:ii+1) = [];
                end
            end
            X0=[amps_X0 tms_X0];
            ShaperLength = length(X0)/2;
        end
    else
        % there are no zero amp impulses or repeated times
        % use current solution
        seek_solution = 0; % breaks while loop
    end
end  % end seek_solution while loop
%**** Check for convergence and assign shaper
```

```
if exitflag < 0
    disp('ERROR: optimization did not converge')
    exactshaper = [];
    shaper = [];
else
    i=2*ShaperLength;
    amps=x(1:ShaperLength);
    times=x(ShaperLength+1:i);
    times(1) = 0;

    % assign exact shaper
    exactshaper = [times' amps'];

    % digitize sequence according to deltaT
    %shaper = digseq(exactshaper,deltaT);
end
%%%%%%% Cost Function for Optimization %%%%%%%%%%%%%
function [f,g]=si2mode_fmincon_fun(x)
 i=2*ShaperLength;           % length of x
f=x(i);                     % function to be minimized (time of tn)
end % end cost function
%%%%%%% Nonlinear Constrain Function for Optimization %%%%%%
function [c,ceq]=si2mode_fmincon_const(x)

i=2*ShaperLength;               % length of x
StepMax=50;                     % # of points at which vibration is limited
% Define matrices to speed computation (see MATLAB doc for reason)
w1 = zeros(1,StepMax);          % Frequencies to test around FIRST mode
w2 = zeros(1,StepMax);          % Frequencies to test around SECOND mode
w_damp1 = zeros(1,StepMax);
w_damp2 = zeros(1,StepMax);
csum1 = zeros(1,StepMax);       % cosine terms for 1st mode vib equation
csum2 = zeros(1,StepMax);       % cosine terms for 2nd mode vib equation
ssum1 = zeros(1,StepMax);       % sine terms for 1st mode vib equation
ssum2 = zeros(1,StepMax);       % sine terms for 2nd mode vib equation
 %**** No nonlinear equality constraints (of form ceq*x = 0)
ceq=[];

%**** Nonlinear inequality constraints (of form c*x <= 0)
% These constraints are the insensitivity inequalities (Vib <= Vtol)
for jj=1:StepMax
    % For FIRST mode
    w1(jj)=wn1*((1-Ins1/2)+Ins1/(StepMax-1)*(jj-1));
    w_damp1(jj) = w1(jj)*sqrt(1-zeta1^2);
    csum1=sum(x(1:i/2).*exp(zeta1*w1(jj)*x(i/2+1:i))...
        .*cos(-w_damp1(jj)*x(i/2+1:i)));
    ssum1=sum(x(1:i/2).*exp(zeta1*w1(jj)*x(i/2+1:i))...
        .*sin(-w_damp1(jj)*x(i/2+1:i)));
    % For SECOND Mode
    w2(jj)=wn2*((1-Ins2/2)+Ins2/(StepMax-1)*(jj-1));
    w_damp2(jj) = w2(jj)*sqrt(1-zeta2^2);
    csum2=sum(x(1:i/2).*exp(zeta2*w2(jj)*x(i/2+1:i))...
        .*cos(-w_damp2(jj)*x(i/2+1:i)));
    ssum2=sum(x(1:i/2).*exp(zeta2*w2(jj)*x(i/2+1:i))...
        .*sin(-w_damp2(jj)*x(i/2+1:i)));
    % if nth impulse is 0, use n-1th impulse of damping envelope
    if abs(x(i/2)) < 1e-4
        g(jj,:)=exp(-2*zeta1*w1(jj)*x(i-1))*(csum1^2+ssum1^2)-Vtol1^2;
        g(StepMax+jj,:)=exp(-2*zeta2*w2(jj)*x(i-1))*(csum2^2+ssum2^2)-Vtol2^2;
```

```
    else
        g(jj,:)=exp(-2*zeta1*w1(jj)*x(i))*(csum1^2+ssum1^2)-Vtol1^2;
        g(StepMax+jj,:)=exp(-2*zeta2*w2(jj)*x(i))*(csum2^2+ssum2^2)-Vtol2^2;
    end
end
c=g;
end  % end nonlinear constraint function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end  % end main function

function [shaper,exactshaper] = ei2m_und(f1,f2,Vlim1,Vlim2,deltaT)
% EI2M_UND(f1,f2,Vlim1,Vlim2,deltaT)-- Bill Singhose
% Generates an EI shaper 2 UNDAMPED modes.
% (The vibration at the modeling frequency is limited to Vlim
%  and the insensitivity is maximized)
% The shapers for each mode are convolved together to give the
% final shaper.
% f1,f2 - frequency (Hz) of vibration being controlled.
% Vlim1,Vlim2 - vibration limits, for 5% vibration, Vlim=0.05.
% deltaT - time spacing at which input to system is updated.

% Mode 1
Wn1=2*pi*f1;
shaperdeltaT1=pi/Wn1;
T21=shaperdeltaT1;
T31= 2*shaperdeltaT1;
A11=(1+Vlim1)/4;
A21=2*(1-Vlim1)/4;
A31=(1+Vlim1)/4;
% shaper for mode1
exactshaper1=[0 A11;T21 A21;T31 A31];
% Mode 2
Wn2=2*pi*f2;
shaperdeltaT2=pi/Wn2;
T22=shaperdeltaT2;
T32= 2*shaperdeltaT2;
A12=(1+Vlim2)/4;
A22=2*(1-Vlim2)/4;
A32=(1+Vlim2)/4;
% shaper for mode2
exactshaper2=[0 A12;T22 A22;T32 A32];
% Convolve the two shapers together
exactshaper=seqconv(exactshaper1,exactshaper2);
shaper = [];%digseq(exactshaper,deltaT);

function [shaper,exactshaper,minBB] = negumei(f,z,V,deltaT)
% [shaper,exactshaper,minBB] = NEGUMEI(f,zeta,V,deltaT)-- Bill Singhose
% Generates a negative Bang-Bang EI shaper for 1 mode
% that when used with a bang-bang unshaped command
% will NOT cause over-currenting if the pulses in the
% bang-bang are longer than the shaper length.
%
% Returns the digital sequence by default, and
% the exact sequence as the second variable.
% The minimum duration for the unshaped bang-bang command to
% avoid over-currenting is also returned(minBB=2*shaper length)
%
% f - undamped frequency (Hz) =Wn/2*pi.
% z - damping ratio
```

```
% deltaT - sampling period
%
% This function generates the exact sequence and then uses
% DigitizeSeq to convert the exact sequence to digital format.
%
% This function produces good answers over the range 0<z<0.4.
% Only V=0.05 and V=0.0125 are available at this time.
amps=[1;-1;1;-1;1];
T=1/f;
if z>0.4,
   fprintf('WARNING: Damping Ratio is probably too large.\n');
end
if V==0.05,
  t(1)=0;
  t(2)=(0.09374+0.31903*z+0.13582*z^2+0.65274*z^3)*T;
  t(3)=(0.36798-0.05894*z+0.13641*z^2+0.63266*z^3)*T;
  t(4)=(0.64256+0.28595*z+0.26334*z^2+0.24999*z^3)*T;
  t(5)=(0.73664+0.00162*z+0.52749*z^2+0.19208*z^3)*T;
elseif V==0.0125,
  t(1)=0;
  t(2)=(0.09051+0.29315*z+0.20436*z^2+0.29053*z^3)*T;
  t(3)=(0.36658-0.081044*z+0.21524*z^2+0.27994*z^3)*T;
  t(4)=(0.64274+0.28822*z+0.25424*z^2+0.34977*z^3)*T;
  t(5)=(0.73339+0.006322*z+0.51595*z^2+0.29764*z^3)*T;
else
   fprintf('Only V=0.05 or V=0.0125 can be used at this time.\n');
end
exactshaper=[t' amps];
shaper = [];%digseq(exactshaper,deltaT);
minBB=2*length(shaper)*deltaT;
```

REFERENCES

[1] Alag, G. S., Kempel, R. W., Pahle, J. W., Bresina, J. J., and Bartoli, F., "Model-following control for an oblique-wing aircraft," Technical Memorandum 88269, NASA Ames Research Center, August 1986.

[2] Banerjee, A. K., "Dynamics and control of the WISP shuttle-antennae system," *J. Astronaut. Sci.*, vol. 41, no. 1, pp. 73–90, 1993.

[3] Bisgaard, M., Bendtsen, J. D., and Cour-Harbo, A. l., "Modeling of generic slung load system," *AIAA J. of Guid. Control Dyn.*, vol. 32, no. 2, 2009.

[4] Bisgaard, M., Cour-Harbo, A. l., and Bendtsen, J. D., "Full state estimation for helicopter slung load system," in *AIAA Guidance, Navigation, and Control Conference*, vol. 4, (Hilton Head, SC), pp. 4036–4050, 2007.

[5] Bisgaard, M., Cour-Harbo, A. l., Johnson, E. N., and Bendtsen, J. D., "Vision aided state estimator for helicopter slung load system," in *17th IFAC Symposium on Automatic Control in Aerospace*, (Toulouse, France), 2007.

[6] Bisgaard, M., la Cour-Harbo, A., and Dimon Bendtsen, J., "Adaptive control system for autonomous helicopter slung load operations," *Control Engineering Practice*, vol. 18, no. 7, pp. 800–811, 2010.

[7] Blackburn, D., Singhose, W., Kitchen, J., Patrangenaru, V., Lawrence, J., Kamoi, T., and Taura, A., "Command shaping for nonlinear crane dynamics," *J. Vib. Control*, vol. 16, pp. 477–501, Apr. 2010.

[8] Brown, E., "The Flying Crane," in *The Helicopter in Civil Operations*, ch. 6, pp. 56–70, New York: Van Nostrand Reinhold Co., 1981.

[9] Cooper, G. E. and Harper Jr., R. P., "The use of pilot rating in the evaluation of aircraft handling qualities," AGARD Report 567, North Atlantic Treaty Organization Advisory Group for Aerospace Research and Development, April 1969.

[10] Drapeau, V. and Wang, D., "Verification of a closed-loop shaped-input controller for a five-bar-linkage manipulator," in *IEEE Int. Conf. Rob. Autom.*, (Atlanta, GA), pp. 216–221, May 1993.

[11] Dukes, T. A., "Maneuvering heavy sling loads near hover part I: Damping the pendulous motion," *J. American Helicopter Soc.*, vol. 18, no. 2, pp. 2–11, 1973.

[12] Dukes, T. A., "Maneuvering heavy sling loads near hover part II: Some elementary maneuvers," *J. American Helicopter Soc.*, vol. 18, no. 3, pp. 17–22, 1973.

[13] Fantoni, I. and Lozano, R., *Non-linear Control for Underactuated Mechanical Systems*. Springer Verlag, 2001.

[14] Grosser, K. and Singhose, W., "Command generation for reducing perceived lag in flexible telerobotic arms," *JSME Int J., Ser. C*, vol. 43, no. 3, pp. 755–761, 2000.

[15] Gupta, N. K. and Bryson Jr., A. E., "Near-hover control of a helicopter with a hanging load," *J. Aircraft*, vol. 13, pp. 217–222, March 1976.

[16] Hall Jr., W. E. and Bryson Jr., A. E., "Inclusion of rotor dynamics in controller design for helicopters," *J. Aircraft*, vol. 10, pp. 200–206, April 1973.

[17] Hamel, P. G. and Kaletka, J., "Advances in rotorcraft system identification," *Progress in Aerospace Sciences*, vol. 33, pp. 259–284, March-April 1997.

[18] Hoh, R. H., Heffley, R. K., and Mitchell, D. G., "Development of handling qualities criteria for rotorcraft with externally slung loads," U.S. Army RDECOM No. AFDD/TR-06-003, NASA Ames Research Center, 2006.

[19] Hohenemser, K., "Dynamic stability of a helicopter with hinged rotor blades," Technical Memorandum 907, NACA, 1939.

[20] Hong, S. W. and Curtiss Jr, H. C., "An analytic modeling and system identification study of rotor/fuselage dynamics at hover," *Mathematical and Computer Modelling*, vol. 19, pp. 47–67, February 1994.

[21] Horn, J. F. and Bridges, D. O., "A model following controller optimized for gust rejection during shipboard operations," in *American Helicopter Society 63rd Annual Forum*, (Virginia Beach, VA), American Helicopter Society, May 2007.

[22] Hyde, J. and Seering, W., "Using input command pre-shaping to suppress multiple mode vibration," in *IEEE Int. Conf. Rob. Autom.*, vol. 3, (Sacramento, CA), pp. 2604–2609, Apr. 1991.

[23] Johnson, W., *Helicopter Theory.* New York: Dover, 1994.

[24] Jones, S. and Ulsoy, A. G., "An approach to control input shaping with application to coordinate measuring machines," *J. Dyn. Syst. Meas. Control*, vol. 121, pp. 242–247, Jun. 1999.

[25] Khalid, A., Huey, J., Singhose, W., Lawrence, J., and Frakes, D., "Human operator performance testing using an input-shaped bridge crane," *J. Dyn. Syst. Meas. Control*, vol. 128, pp. 835–841, Dec. 2006.

[26] Kim, D. and Singhose, W., "Performance studies of human operators driving double-pendulum bridge cranes," *Control Eng. Pract.*, vol. 18, pp. 567–576, Jun. 2010.

[27] Landis, K. H., Davis, J. M., Dabundo, C., and Keller, J. F., "Advanced flight control research and development at Boeing Helicopters," in *Advances in Aircraft Flight Control* (Tischler, M. B., ed.), ch. 4, pp. 103–141, Bristol, PA: Taylor & Francis, 1996.

[28] Lucassen, L. R. and Sterk, F. J., "Dynamic stability analysis of a hovering helicopter with a sling load," *J. American Helicopter Soc.*, vol. 10, no. 2, pp. 6–12, 1965.

[29] Magee, D. P. and Book, W. J., "Filtering micro-manipulator wrist commands to prevent flexible base motion," in *American Control Conf.*, (Seattle, WA), pp. 924–928, Jun. 1995.

[30] Manning, R., Clement, J., Kim, D., and Singhose, W., "Dynamics and control of bridge cranes transporting distributed-mass payloads," *J. Dyn. Syst. Meas. Control*, vol. 132, p. 014505, Jan. 2010.

[31] Morikawa, C. and Murakami, T., "Vibration suppression control for a hanging load of autonomous helicopter using simplified transfer function," in *Asia International Symposium on Mechatronics*, (Hokkaido University, Japan), Aug. 2008.

[32] Morrow, L. D. and Balasubramanian, R., "Real model following control," *J. Aircraft*, vol. 12, no. 12, pp. 996–998, 1975.

[33] Murphy, B. R. and Watanabe, I., "Digital shaping filters for reducing machine vibration," in *Robotics and Automation, IEEE Transactions on*, vol. 8, pp. 285–289, 1992.

[34] Murphy, R. D. and Narendra, K. S., "Design of helicopter stabilization systems using optimal control theory.," *J. Aircraft*, vol. 6, no. 2, pp. 129–136, 1969.

[35] Ottander, J. A. and Johnson, E. N., "Precision slung cargo delivery onto a moving platform," in *AIAA Modeling and Simulation Technologies Conference*, (Toronto, Canada), 2-5 August 2010.

[36] Padfield, G. D., *Helicopter Flight Dynamics: The Theory and Application of Flying Qualities and Simulation Modeling*. AIAA Educational Series, Washington, DC: American Institute of Aeronautics and Astronautics, Inc., 1996.

[37] Potter, J., Singhose, W., and Costello, M., "Reducing swing of model helicopter sling load using input shaping," in *9th IEEE International Conference on Control and Automation*, (Santiago, Chile), pp. 348–353, 19-21 Dec. 2011.

[38] Prouty, R. W., *Helicopter Performance, Stability, and Control*. Malabar, FL: Krieger Publishing Company, Inc., 1990.

[39] Singer, N. C. and Seering, W. P., "Preshaping command inputs to reduce system vibration," *J. Dyn. Syst. Meas. Control*, vol. 112, pp. 76–82, Mar. 1990.

[40] Singh, T. and Heppler, G. R., "Shaped input control of a system with multiple modes," *J. Dyn. Syst. Meas. Control*, vol. 115, pp. 341–347, Sep. 1993.

[41] Singh, T. and Vadali, S. R., "Robust time-delay control," *J. Dyn. Syst. Meas. Control*, vol. 115, pp. 303–306, Jun. 1993.

[42] Singhose, W., Kim, D., and Kenison, M., "Input shaping control of double-pendulum bridge crane oscillations," *J. Dyn. Syst. Meas. Control*, vol. 130, p. 034504, May 2008.

[43] Singhose, W. and Seering., W., *Command Generation for Dynamic Systems*. Lulu, 2009.

[44] Singhose, W., Seering, W., and Singer, N., "Residual vibration reduction using vector diagrams to generate shaped inputs," *J. Mech. Des.*, vol. 116, pp. 654–659, Jun. 1994.

[45] Singhose, W., Bohlke, K., and Seering, W., "Fuel-efficient pulse command profiles for flexible spacecraft," *AIAA J. of Guid. Control Dyn.*, vol. 19, no. 4, pp. 954–960, 1996.

[46] Singhose, W., Crain, E., and Seering, W., "Convolved and simultaneous two-mode input shapers," *IEE Control Theory Appl.*, vol. 144, pp. 515–520, Nov. 1997.

[47] Singhose, W., Seering, W., and Singer, N., "Input shaping for vibration reduction with specified insensitivity to modeling errors," in *Japan-USA Sym. Flexible Autom.*, vol. 1, (Boston, MA), pp. 307–313, 1996.

[48] Singhose, W., Singer, N., and Seering, W., "Improving repeatability of coordinate measuring machines with shaped command signals," *Precis. Eng.*, vol. 18, pp. 138–146, Apr. 1996.

[49] Smith, J. H., Allen, G. M., and Vensel, D., "Design, fabrication, and flight test of the active arm external load stabilization system for cargo handling helicopters," tech. rep., Boeing Vertol Co., September 1973.

[50] Smith, O. J. M., *Feedback Control Systems*. New York, NY: McGraw-Hill Book Co., Inc., 1958.

[51] Sorensen, K. L., Singhose, W., and Dickerson, S., "A controller enabling precise positioning and sway reduction in bridge and gantry cranes," *Control Eng. Pract.*, vol. 15, pp. 825–837, Jul. 2007.

[52] Stevens, B. L. and Lewis, F. L., "Modern design techniques," in *Aircraft Control and Simulation*, ch. 5, pp. 421–437, Hoboken, NJ: John Wiley & Sons, Inc., 1992.

[53] Szustak, L. S. and Jenney, D. S., "Control of large crane helicopters," *J. American Helicopter Soc.*, vol. 16, pp. 11–22, July 1971.

[54] Tischler, M. B., "System identification requirements for high-bandwidth rotorcraft flight control system design," in *American Control Conference*, (Boston, MA), pp. 2494–2502, June 1991.

[55] Trentini, M. and Pieper, J. K., "Model-following control of a helicopter in hover," in *IEEE International Conference on Control Applications*, (Dearborn, MI), pp. 7–12, 15–18, September 1996.

[56] Tuttle, T. and Seering, W., "Experimental verification of vibration reduction in flexible spacecraft using input shaping," *AIAA J. of Guid. Control Dyn.*, vol. 20, pp. 658–664, Jul. 1997.

[57] Tyler Jr., J., "The characteristics of model-following systems as synthesized by optimal control," *IEEE Transactions on Automatic Control*, vol. 9, no. 4, pp. 485–498, 1964.

[58] UK Civil Aviation Authority Safety Regulation Group, *Helicopter External Load Operations*. The Stationery Office, April 2006. CAP 426.

[59] United States Army Aviation and Missile Command, *Aeronautical Design Standard 33E: Handling Qualities Requirements for Military Rotorcraft*. Aviation Engineering Directorate, Redstone Arsenal, Alabama, 21 March 2000. ADS-33E-PRF.

[60] Vaughan, J., Yano, A., and Singhose, W., "Comparison of robust input shapers," *J. Sound Vib.*, vol. 315, pp. 797–815, Sep. 2008.

[61] Vaughan, J., Yano, A., and Singhose, W., "Robust negative input shapers for vibration suppression," *J. Dyn. Syst. Meas. Control*, vol. 131, p. 031014, May 2009.

[62] Vicon Motion Systems, *Vicon MX Hardware System Reference*. Oxford, UK, 2007. Revision 1.6.

[63] Vicon Motion Systems, "Vicon | Products | Cameras | MX-3+." [Online]. Available: http://www.vicon.com/products/mx3.html, Date Accessed: 15 March 2012.

[64] Vicon Motion Systems, "Vicon | Products | Hardware | MX Ultranet HD." [Online]. Available: http://www.vicon.com/products/ultranethd.html, Date Accessed: 15 March 2012.

[65] Wie, B., Sinha, R., Sunkel, J., and Cox, K., "Robust fuel- and time-optimal control of uncertain flexible space structures," in *AIAA Guid. Navig. Control Conf.*, (Monterey, CA), pp. 939–948, Aug. 1993.

[66] Winsor, C. and Roy, R., "The application of specific optimal control to the design of desensitized model following control systems," *IEEE Transactions on Automatic Control*, vol. 15, no. 3, pp. 326–333, 1970.