


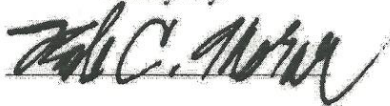
Active Control for Helicopter Sling Load Stabilization

A Major Qualifying Project Report
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science
in Aerospace Engineering

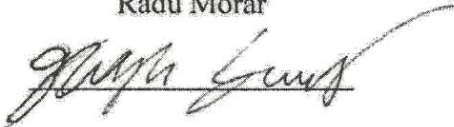
by



Dusty Cyr



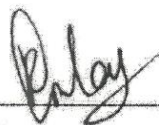
Radu Morar



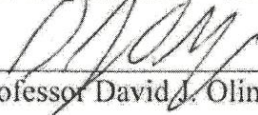
Joseph Sperry

April 30, 2015

Approved by:



Professor Raghvendra V. Cowlagi, Advisor



Professor David J. Olinger, Coadvisor

Aerospace Engineering Program
Mechanical Engineering Department, WPI

Abstract

Helicopter sling loads are widely used for cargo delivery by the military, but are inherently unstable in flight, which is hazardous. This project designed the SPARTA system to stabilize sling loads using active control methods by redirecting airflow over the sling load using a rudder and pipes with control vents, to create stabilizing forces and moments. This project involved mathematical modeling, simulation, and wind-tunnel testing of the SPARTA system for a 1/17 scale standard sling load cargo container. Stabilization of yaw motion and lateral sway motion of the sling load were investigated using the LQR control algorithm. An Arduino microcontroller was used to collect measurements and to actuate the control surfaces with servomotors.

“Certain materials are included under the fair use exemption of the U.S. Copyright Law and have been prepared according to the fair use guidelines and are restricted from further use.”

Contents

Acknowledgements.....	7
Table of Authorship	8
CHAPTER 1: INTRODUCTION.....	9
CHAPTER 2: BACKGROUND AND LITERATURE REVIEW	11
2.1 Helicopter Sling Loads	11
2.2 CONEX and TRICON Containers.....	12
2.3 Bluff Body Aerodynamics and Stability Issues	14
2.4 Previous Mathematical Models.....	15
2.5 Previous Active Stabilization Methods.....	15
CHAPTER 3: METHODOLOGY	18
3.1 Development of the SPARTA System (Stabilization with Pipes and Rudder TRICON Assembly)	18
3.1.1 Development of the “Pipes” Design	18
3.1.2 Development of the “Rudder” Design	22
3.2 Development of the Hardware and Actuation Scheme.....	27
3.2.1 Hardware.....	27
3.2.1 Actuation & Power	30
3.3 Mathematical Model Development.....	32
3.4 Control Law Development.....	35
3.5 Kalman Filters.....	36
3.6 Wind Tunnel Testing	38
3.7 Rapid Prototyping	42
3.8 Aerodynamic Analysis Verification	45
3.9 Final 1/17 th Scale Prototype Testing.....	47
CHAPTER 4: RESULTS	48
4.1 Initial 1/10 th Scale Pipes Testing	48
4.2 Initial 1/17 th Scale Poster-Board TRICON Testing	50
4.3 Final 1/17 th Scale SPARTA Prototype Testing.....	51
4.4 MATLAB Simulation Results	51
4.5 Kalman Filter Results	53

4.6 IMU-SD Card Data Logging Results.....	55
CHAPTER 5: CONCLUSIONS	57
Appendix A: Derivation of Moments of Angular Momentum	59
Appendix B: Aerodynamic Equations from <i>Modelling of static aerodynamics of helicopter underslung loads</i>	65
Appendix C: Full MATLAB Numerical Simulation Code and Results	67
Simulation Code.....	67
Model parameters.....	67
Simulation parameters	68
Control parameters and LQR gain	69
Kalman Parameters	69
Run simulation	70
Plot results.....	72
State Derivative Function	75
Main Function Code	75
Aerodynamic Forces	78
Kalman Filters.....	80
Aerodynamic Moments.....	81
Rudder Forces and Moments	82
Pipe Forces and Moments.....	83
Control Inputs	84
Moment Transform Function.....	85
Force Transform Function	85
Double Dot Solve Function	85
LQR Gain Function.....	88
Kalman Filter P Derivative Function.....	91
Define Filter Matricies.....	91
Calculate P_dot	92
Yaw and Yaw Rate Kalman Filter Function.....	92
Define Constants.....	92
Define Matricies.....	93

Matricies	94
Roll and Sway Velocity Kalman Filter Function.....	95
Define Constants.....	95
Define Matricies.....	95
Matricies	96
Appendix D: ARDUINO DUE Control Code	97
Code	97
Final Hardware Setup Reference	108
Appendix E: ARDUINO DUE SD Card IMU Data-logging Code	109
Appendix F: Pipes And Rudder Calculations MATLAB Code.....	120
Information	120
Conversion Factors and Scaling.....	120
Inputs.....	120
Pipe Equations	121
Vertical Stabilizer and Rudder Equations.....	122
Plotting.....	122
Works Cited	124

Table of Figures

Figure 1: Sling Set Components (Nyren, 2013).....	11
Figure 2: UH-60 Blackhawk sling-loading a Humvee	11
Figure 3: UH-60 Blackhawk in Flight	12
Figure 4: Various 8'x'6.5'x8' TRICON containers (CMCI, 2011)	13
Figure 5: Flow Separation vs. Incidence Angle (Greenwell, 2011)	14
Figure 6: Gera, Farmer Design	16
Figure 7 AFDD Design.....	17
Figure 8: First Pipes Design	18
Figure 9: Second Pipes Design ...	18
Figure 10: Third Design of the Pipes	19
Figure 11: Correcting Sway	20
Figure 12: Correcting Yaw	20
Figure 13: Self-Correcting Design	21
Figure 14: Fourth Design of the Pipes (with Rudder Included)	21
Figure 15: Final Design of the Pipes (with Rudder Included)	22
Figure 16: Initial Design	23
Figure 17: Final Rudder Design	23
Figure 18: Rudder System	24
Figure 19: Principles of Rudder Yaw Correction	26
Figure 20: Rudder in Final SPARTA System Prototype	27
Figure 21: Arduino DUE	28
Figure 22: Adafruit 9-DOF IMU	29
Figure 23: Adafruit Micro-SD Card Breakout.....	29
Figure 24 : HS-5065MG Servo	31
Figure 25: HS-7985MG Servo	31
Figure 26: Depiction of the Power Scheme Implementation.....	32
Figure 27: MATLAB Simulation Flowchart	33
Figure 28: MATLAB Simulation With Kalman Filter Flow Chart	37
Figure 29: Poster-board TRICON (note power cord - this was removed for subsequent tests) ...	38
Figure 30: Inside of Poster-Board TRICON with Components.....	39
Figure 31: Detachable Sling Legs on the Second TRICON Prototype.....	39
Figure 32: Basswood and Balsawood SPARTA Pre-Prototypes.....	40
Figure 33: A Screenshot from One of the Wind Tunnel Test Videos	41
Figure 34: Larger Rudder.....	41
Figure 35: Half-Sized Doors and Broken Pillar.....	42
Figure 36: Original Servo Gear Mechanism	43
Figure 37: Directly Attached Servo Mechanism	43
Figure 38: Servo Mechanism	44
Figure 39: Original Rudder with Nail.....	44
Figure 40: Final Rapid-Prototyped TRICON and SPARTA	45
Figure 41: 1/10th Scale Pipes with Doors	45
Figure 42: 1/10th Scale Vertical Stabilizer and Rudder.....	45

Figure 43: CAD model of support stand and with pipes in the wind tunnel 46

Figure 44: Rear Door X-Direction Force vs. Door Angle 48

Figure 45: Restoring Y-Direction Force vs. Door Angle 49

Figure 46: Rear Door Z-direction Moment vs. Door Angle 49

Figure 47: Average Maximum Yaw Angle for Various SPARTA Iterations..... 50

Figure 48: Yaw And Yaw Rate, Simulation Results Without SPARTA..... 51

Figure 49: Lateral And Longitudinal Swing Angles, Simulation Results Without SPARTA..... 52

Figure 50: Yaw And Yaw Rate, Simulation Results With SPARTA 52

Figure 51: Lateral And Longitudinal Swing Angles, Simulation Results With SPARTA..... 53

Figure 52: Yaw And Yaw Rate Kalman Filter Results Without Linked Control..... 54

Figure 53: Roll And Sway Velocity Kalman Filter Results Without Linked Control..... 54

Figure 54: Kalman Yaw And Yaw Rate With Linked Control 55

Figure 55: Kalman Roll And Sway Velocity With Linked Control 55

Figure 56: Accelerometer and Gyroscope Data-Logging Results 56

Figure 57: Pipes Diagram 59

Figure 58: Angular Momentum Moment..... 62

Figure 59: Rear Door Moment Arm 63

Figure 60: Front Door Moment Arm 64

Acknowledgements

We would like to thank the following individuals and groups for their help and support throughout the entirety of this project.

Project Advisors Professor Raghvendra V. Cowlagi
 Professor David J. Olinger

NSRDEC Sponsor Daniel Nyren

M.E. Dept. Staff Barbara Furhman

E.C.E. Dept. Staff Robert M. Boisse

Table of Authorship

Section	Author Initials
<i>Chapter 1: Introduction</i>	DC
<i>Chapter 2: Background and Literature Review</i>	
2.1, 2.2, 2.3	DC
2.5	RM
2.4	JPS
<i>Chapter 3: Methodology</i>	
3.1.1, 3.6, 3.7, 3.8, 3.9	DC
3.1.2, 3.2.1, 3.2.2	RM
3.3, 3.4, 3.5	JPS
<i>Chapter 4: Results</i>	
4.1, 4.2, 4.3	DC
4.6	RM
4.4, 4.5	JPS
<i>Chapter 5: Conclusions and Analysis</i>	DC, RM, JPS
<i>Appendix A: Derivation of Moments of Angular Momentum</i>	DC
<i>Appendix B: Aerodynamic Equations</i>	JPS
<i>Appendix C: Full MATLAB Numerical Simulation Code</i>	JPS
<i>Appendix D: ARDUINO DUE Control Code</i>	RM
<i>Appendix E: ARDUINO DUE SD Card IMU Data-logging Code</i>	RM
<i>Appendix F: Pipes And Rudder Calculations MATLAB Code</i>	DC, RM

CHAPTER 1: INTRODUCTION

Rapid and safe delivery of supplies is crucial to today's military operations. Hazardous terrain and/or passage through dangerous enemy territory can delay the delivery of supplies and place the delivery vehicle at great risk of capture or destruction. Aerial delivery systems, which negate the issue of hazardous terrain and reduce the time spent in enemy territory, have therefore proven to be invaluable for such tasks.

The helicopter sling load delivery system is “most accurate form of aerial delivery in the military today” due to the ability of the helicopter to bring cargo to the precise spot it is needed (Nyren, 2013). A helicopter sling load consists of cargo attached to the underside of the helicopter through a series of hooks and ropes. The ropes, called *sling legs*, are attached to the cargo via chains, and to the bottom of the helicopter via an apex fitting that attaches to the helicopter's cargo hook.

Most sling loads are not aerodynamically stable, and undergo undesirable motions that hinder flight safety and force the helicopter to fly at lower speeds. Commonly slung-loaded payloads such as rectangular CONEX containers and Humvees are bluff bodies that undergo undesirable pitch, sway and yaw motions that put the safety of both the helicopter and cargo at risk. Similar to a “swinging pendulum,” a sling load can make the helicopter harder to control, damage the cargo that is being sling-loaded, or in the worst case, collide with the helicopter (Potter, Singhose, & Costello, 2011). Helicopters with sling loads therefore have to fly with a significantly lower speed and altitude, making them more vulnerable to enemy fire and increasing delivery time. Due to the increased delivery time, recipients of the cargo have to wait longer to receive potentially vital supplies, the helicopter must remain over potentially dangerous territory for a longer time, and possibly excess fuel is needed to complete the (prolonged) mission.

Reduction of such undesirable pitch, sway and yaw motions through stabilization of the sling load will allow for an increased speed and altitude, as well as a decreased delivery time. Stabilization methods for sling loads can be classified into two broad categories: *passive* and *active*. Passive stabilization involves no controlled mechanisms: devices such as fins or tails are placed on the sling-loaded cargo to reduce the aerodynamic effects of the bluff body without any inputs. Active stabilization involves the manipulation of control surfaces or other mechanisms of the sling load system using control laws that are based on as the measured airspeed and angles of motion.. Controllable fins and rudders are examples of active stabilization systems.

This project, sponsored by the U.S. Army Natick Soldier Research Development and Engineering Center (NSRDEC), involves the development of an active stabilization method for helicopter sling load systems. The active stabilization system developed reduces in particular the two most significant motions of sling-loaded cargo: sway and yaw motion. To create this system, the team first researched the literature to study existing passive and active stabilization

techniques to form a foundation for this work. The team then developed a physics-based mathematical model of the proposed system, and developed a numerical simulation of this model using the MATLAB® software package. Lastly, the team designed a control scheme and the control laws to guide it, and built a small-scale prototype that was tested in a wind tunnel.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

This section discusses the fundamentals of helicopter sling loading, the various issues that sling loading causes, and past research into mathematical modelling of sling load systems. It also presents active stabilization techniques and ideas developed in the past that influenced the team's designs.

2.1 Helicopter Sling Loads

The basic components of a typical sling load set up are the sling legs, apex fitting and grabhook/grablink. The apex fitting is attached to the cargo hook of the helicopter, with a spacer needed for the UH-60 Blackhawk. This spacer provides additional assistance to the helicopter, “[reducing] the shock load to the cargo hook caused by oscillating and rotating loads” (Nyren, 2013). The sling legs, which connect to the apex fitting, are made out of double-braided nylon rope and are approximately twelve feet in length, with environmental effects and usage providing small alterations to their length. The grabhook and grab link, connecting the sling legs to the cargo, consist of chains and are adjustable to provide a three-to-five degree nose down orientation and to prevent the sling legs from touching the payload, which could result in damage to the sling legs.

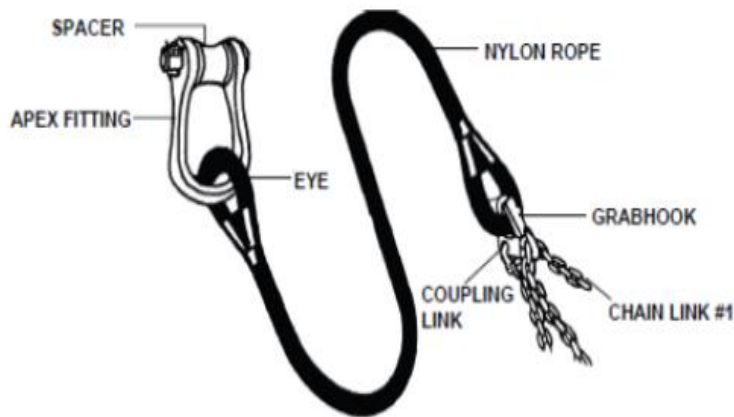


Figure 1: Sling Set Components (Nyren, 2013)



Figure 2: UH-60 Blackhawk sling-loading a Humvee

Helicopter sling load missions are undertaken in every branch of the United States military. The following table lists commonly-used helicopters in sling load operations, and several key parameters.¹

¹ Table from (Nyren, 2013)

Name	Maximum Airspeed	Maximum Safe Airspeed (Sling-loaded)	Max Hook Capacity (lbs)	Number of Hooks
UH-60 Blackhawk	193 knots	60 knots ²	9,000	1
CH-47 Chinook	170 knots	70-140 knots ³	26,000	3
UH-72 Lakota	145 knots	100 knots ⁴	4,000	1
CH-53E Super Stallion	170 knots	80-95 knots ⁵	32,000	3 swiveling

Table 1: Capabilities of Sling Load Helicopters

As shown in the table, helicopters with sling loads must fly at much lower maximum airspeeds to keep both the cargo and helicopter safe. The UH-60 Blackhawk in particular has a severe loss in maximum airspeed; while carrying a sling load the Blackhawk can safely attain just thirty-one percent of its actual maximum airspeed.

The UH-60 Blackhawk is the target helicopter for this project's active stabilization system. The Blackhawk is used extensively in all branches of the U.S. military, and only has one cargo hook, limiting it to single-point sling loading. The Chinook and Super Stallion both have multiple hooks, meaning that their sling loads are not prone to as severe motions and oscillations due to the self-stabilizing nature of multiple attachment points. Like the Blackhawk, the Lakota only has one hook, but can carry less than half the weight that the Blackhawk is able to, and thus will not be focused upon either.



Figure 3: UH-60 Blackhawk in Flight

2.2 CONEX and TRICON Containers

Helicopters sling load cargo including containers, vehicles and artillery, all with varying geometry. An active stabilization system designed to function for cargo of varying geometry, without touching the helicopter or sling legs, is beyond the scope of this project. Therefore, the active stabilization system is designed to accommodate a specific cargo geometry: a rectangular container.

² For 8'x8'x6.5' CONEX (Cicolani et al., 2009)

³ (Army, 2009a)

⁴ ("Army UH-72 Flight Limitations (EC-145/BK 117 C-2)," 2000)

⁵ (Army, 2009b)

Rectangular cargo containers come in a variety of sizes and are sling-loaded regularly. Two commonly-used sizes are the 8' x 6' x 6' CONEX and the 8' x 8' x 20' MILVAN⁶ containers, both of which have experienced stability issues while in flight (Greenwell, 2011). The 8' x 6' x 6' CONEX has been the focus of several studies into the stabilization of sling loads, such as Raz et al. (2011), McCoy (1998), and Nyren (2013). The CONEX (container express box) is a simple rectangular steel container with corrugated sides and a flat floor and roof (McCoy, 1998). The corrugated sides do not significantly affect the aerodynamics of the load and will therefore be idealized as flat sides (Nyren, 2013). The 8' x 6' x 6' CONEX is an ideal choice for sling load stabilization due to its widespread use, simple geometry for modelling and testing, and ability to carry heavy loads without altering the basic geometry (McCoy, 1998).

The sponsor for this project, NSRDEC, suggested using a slightly different sized container called the TRICON. The TRICON is slightly larger than the CONEX, with dimensions of 8' x 6.5' x 8'. The container is also heavier, with a tare weight of 2,600 pounds compared to the CONEX's tare weight of 1,800 pounds (CMCI, 2011; McCoy, 1998). The TRICON is used "extensively" by the United States military, and features an all steel construction, 346 cubic feet of internal capacity and 12,000 pounds of payload capacity (CMCI, 2011).



Figure 4: Various 8'x'6.5'x8' TRICON containers (CMCI, 2011)

NSRDEC is currently conducting small-scale stabilization testing of the TRICON at the Massachusetts Institute of Technology's (MIT) wind tunnel, so using a TRICON model as the test bed for this project's active stabilization system would provide an easy scaling-up factor for MIT wind tunnel testing. Therefore, a scale model of the TRICON model was used for the design and testing of the active stabilization system.

⁶ Dimensions are (Length x Width x Height)

2.3 Bluff Body Aerodynamics and Stability Issues

Helicopter sling load missions encounter two primary issues: instability due to the bluff body nature of the cargo and extra drag that causes power restraints on the helicopter (Nyren, 2013). According to Greenwell, three basic types of instability can occur: aerodynamic instability of the load, helicopter and load vertical oscillations, and sling cable flapping (Greenwell, 2011).⁷ Aerodynamic instability involves highly complex motions and oscillations that typically begin with “a period yaw oscillation which then couples into the sling and helicopter response” (Greenwell, 2011).

Aerodynamic issues occur even once the cargo reaches its destination, as crews sometimes have to wait until the load’s swing amplitude settles to an acceptable level before depositing the cargo, wasting time and efficiency (Potter et al., 2011).

According to the in-depth study by Greenwell, “containers are rectangular bluff bodies, with the aerodynamic loads dominated by normal pressure forces,” with “basic forces and moments... split into “attached flow” and “separated flow” components” (Greenwell, 2011). The attached flow component “comprises the loads from the attached flow on the front face, the attached flow underlying the side face separation bubbles, and the fully separated base flow, and is independent of box geometry,” while the separated flow component comprises the separated flow on the two side faces, which varies from full separation to a closed separation bubble depending on incidence and geometry” (Greenwell, 2011). The figure below provides a visualization of the flow separation as a function of incidence angle.

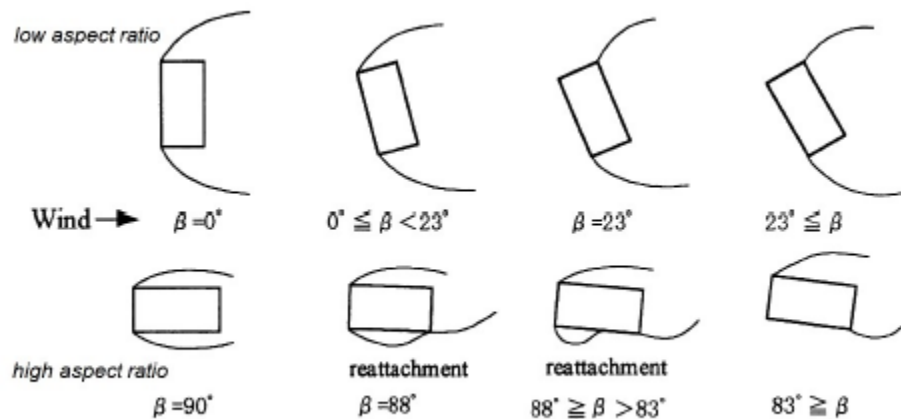


Figure 5: Flow Separation vs. Incidence Angle (Greenwell, 2011)

⁷ Helicopter oscillations and sling cable flapping are beyond the scope of this project, and not as significant as the aerodynamic instabilities, and therefore will not be focused on in this report.

2.4 Previous Mathematical Models

One of the first steps was to develop a mathematical model that would simulate a helicopter sling load system in MATLAB. This was needed for the development of the control laws. The mathematical model assumes that the slung loaded cargo is a rigid box with uniform density that was slung below a fixed attachment point at the origin. In the model the box is attached to the attachment point through four sling legs attached to the box at each of the corners on the top side and a flow of air is run over the box. All this is represented in the MATLAB model through a series of equations that predict the forces and moments that act on the simulated slung cargo. The forces and moments from the sling legs and gravity are easily represented using basic physics and stress analysis equations. However the equations for the aerodynamic forces and moments exerted on the container are a lot more complicated. The reason why these equations are more complex is because of the shape of the box. A box is a bluff body, and the aerodynamics of bluff bodies are a lot harder to model compared to other shapes. Instead of trying to derive the aerodynamic equations, it was decided that a previous mathematical model should be used.

At the beginning of the project several pieces of literature were provided by the project advisors. Among this literature was a paper by Greenwell (Greenwell, 2011) describing a mathematical model for the static aerodynamics of helicopter underslung loads. This paper was used to provide all the assumptions for aerodynamic forces and moments on the container load in the mathematical model. The equations are provided in Appendix B.

In this paper there are two sections that detail 3D aerodynamics of rectangular containers. These sections are Section 3.0, which discusses three dimensional aerodynamics, and the appendix, which shows the equations for the actual mathematical model. Both these sections were used heavily in our project to incorporate aerodynamic forces and moments into the mathematical model. In section 3.0 of Greenwell's paper the results of his models are compared to collected aerodynamic data from different types of rectangular containers. This demonstrates that the model does follow the general trend of the collected data. In the appendix Greenwell outlines the specific equations that he developed for his aerodynamic model of rectangular containers.

2.5 Previous Active Stabilization Methods

A variety of active helicopter sling load stabilization methods have been studied in the past. These systems focus on mitigating or eliminating the most detrimental forms of sling load instability, yaw motion, sway motion (lateral side-to-side motion), or both. Everything from active systems on board the helicopter, to systems that affect the sling legs, to systems that affect the load has been proposed. The focus of this project is a system that affects the container.

Systems that are on board the helicopter or affect the sling legs are outside the scope of this project and will only be briefly discussed.

There are three papers that outline methods of active helicopter sling load stabilization that are outside of the scope of this project. One such method is through the use of a winch system to control the sling legs (Asseo, 1973). This method uses control theory to design multiple systems which stabilize a variety of loads in different configurations. Whereas this had merit, one of the constraints of this design project was that the sling-legs were not to be interfered with; therefore such a similar design was not an option. Another proposed active control scheme involves introducing a control scheme into an existing helicopter control system along with vision-based sensor data (Bisgaard, 2010). This method is designed so that the helicopter would fly without inducing oscillations in the sling load system. A system involving the damping of oscillations in the sling legs via the use of linear actuators attached to the legs was also proposed (Smith, 1975). Both of these designs fall outside of the project constraints so similar systems were not considered.

The two main designs that were considered when first approaching this project were a rotational stabilization method proposed by the Aeroflightdynamics Directorate (AFDD) of AMRDEC and an active fin design proposed by Gera and Farmer. Gera and farmer used a rotational cup design mounted on top of a CONEX container coupled with a yaw rate feedback controller in order to spin up a swivel-hook sling load configuration to stabilize sway motion (Gera, 1974).



Figure 6: Gera, Farmer Design

The AFDD proposed controllable fins at both ends of a dual-point sling load configuration in order to also control sway motion (AFDD, 2011). Both these designs were considered during the initial design phase of this project; however one of the design constraints

for this particular project was a single-point fixed hook sling load. Controllable fins, a fin in the form of a rudder to be precise, did eventually become part of the final design.

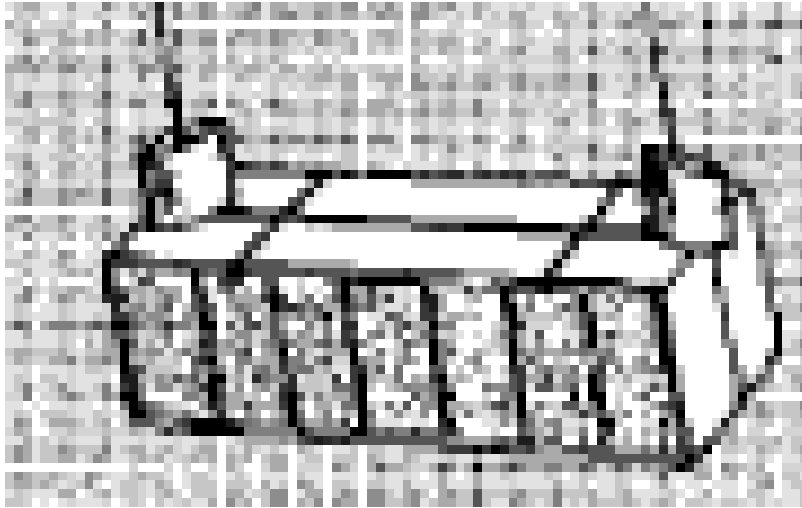


Figure 7 AFDD Design

CHAPTER 3: METHODOLOGY

3.1 Development of the SPARTA System (Stabilization with Pipes and Rudder TRICON Assembly)

The SPARTA System (formerly called PARAS: Pipes and Rudder Active Stabilization System) is a hybrid merged from two separate designs developed simultaneously during the first half of the project: the airflow-redirecting “pipes” design, and the airflow-deflecting “rudder” design.

3.1.1 Development of the “Pipes” Design

The “pipes” design is based on redirection of the freestream flow to produce stabilizing aerodynamic forces and moments. Inspired by the concept of thrust vectoring, incoming air is redirected in a direction of choice to provide a force or moment that counters any unwanted forces or moments that cause instability.

Originally, two pipes were placed along the longitudinal axis (parallel to the freestream flow) on top of the TRICON, with the outlets turned ninety degrees (perpendicular to the freestream flow) facing outward from the center of the TRICON and aligned with its center of mass. The outlets were allowed to rotate about ninety degrees, mostly towards the rear, to face any direction desired. The inlets also rotated to always face the freestream flow; the container could yaw up to fifty degrees in either direction and the inlets would counter-rotate to still directly face the flow of oncoming air. The entire unit would be held to the container using straps, and one or both of the inlets could be closed if no restoring forces were needed. This original design is shown in Figure (8).

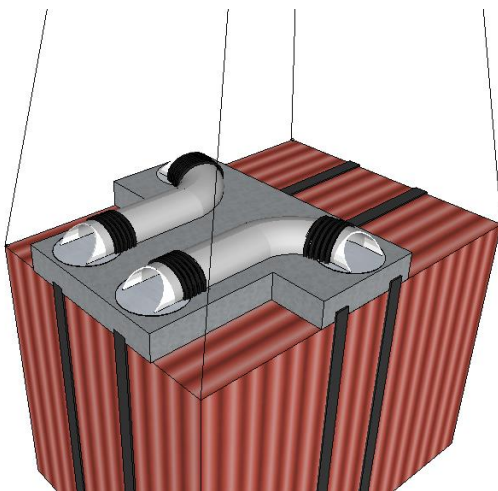


Figure 8: First Pipes Design

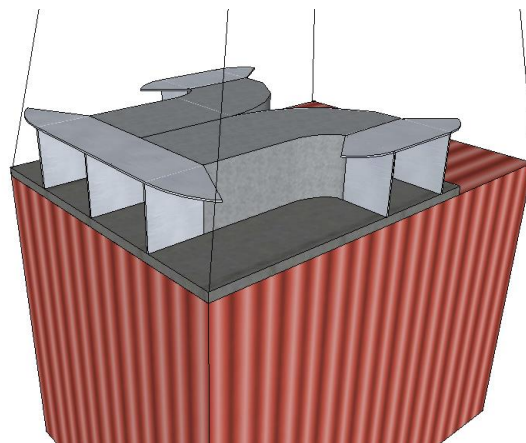


Figure 9: Second Pipes Design

It was determined that having the entire inlet and outlet rotate was not practical due to material requirements and complexity of the mechanism. Figure (9) shows a redesign of the system. The inlets and outlets have doors that rotate to redirect the flow in the same manner as the previous design. The flat plates above the doors function as a nozzle to focus the airflow into the inlets and out of the outlets, and also serve as an illustration of how much the inlets and outlets could rotate. However, at this point it was discovered that whereas the current design would provide restoring side forces to correct sway motion, it performed poorly at correcting yaw even if the outlets could rotate, because the forces were always applied in the plane of the center of mass of the container. The forces would need to be applied farther away from the center of mass to provide a moment arm long enough to produce a noticeable and effective moment. Also, rotating inlet doors were proven to provide no advantage to the design, as they did not actually provide any focusing of the freestream into the inlets. Thus, a second redesign was drafted.

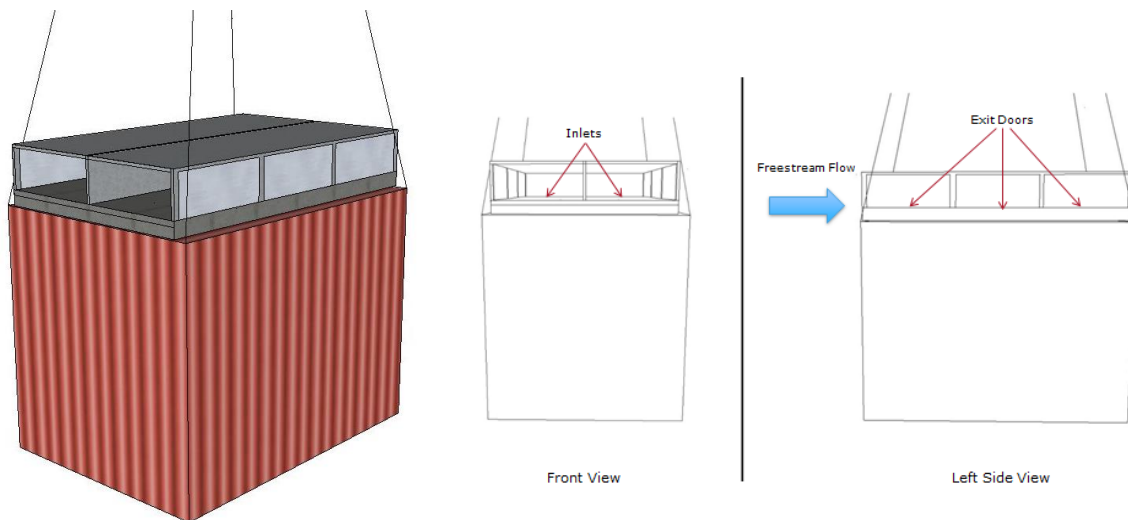


Figure 10: Third Design of the Pipes

The third design of the pipes, shown in Figure (10), solves all of the previous issues: three exit doors along the sides of the pipes open to redirect the freestream flow to either of their locations. Sway is corrected by opening the middle door (aligned with the center of mass of the container), and restoring moments are generated by opening either the front or rear doors. The inlets are fixed and the air is allowed to pass through the pipes and out the back, eliminating much of the drag that the previous designs would induce. The exit doors were originally designed to open up to ninety degrees inward.

The following two figures show top-down views of the pipes correcting both sway and yaw motions of the container. In Figure (11), the pipes are shown correcting for sway motion to the

left, with the middle door opening to provide a restoring force in the opposite direction to the movement of the container. Figure (12) shows the design correcting for counter-clockwise yaw motion, with the front-left and right-rear doors opening to maximize the restoring moment (or torque) to the container and realign the TRICON with the freestream flow.

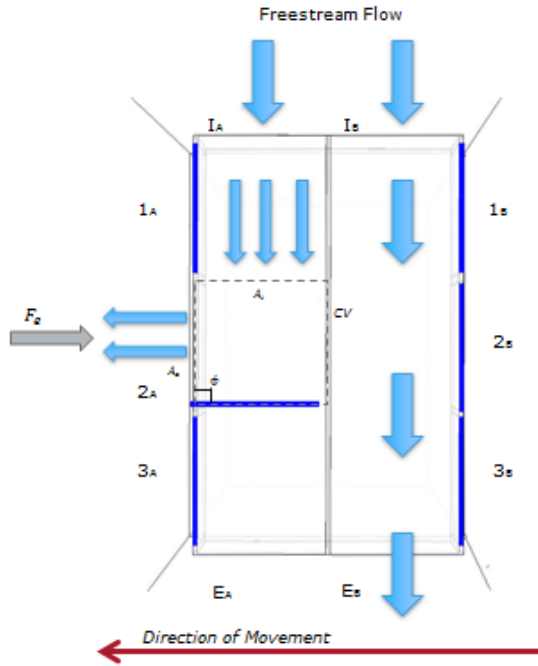


Figure 11: Correcting Sway

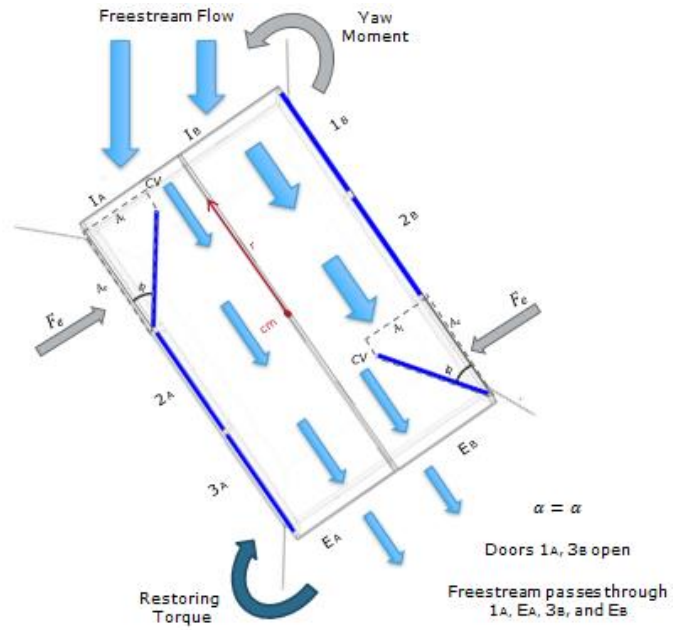


Figure 12: Correcting Yaw

Calculations of the restoring forces and moments the pipes system would deliver led to two adjustments to this design. These calculations, included in Appendix A, led to the following conclusions:

1. Due to moments of angular momentum (shown in Figure 11), the rear doors generate the largest restoring force for sway motion while minimizing unwanted moments
2. Unless a door is opened ninety degrees and aligned with the center of mass, it will produce an unwanted moment
3. Opening any door greater than approximately thirty degrees produces too much drag, side force, and moment to be effective
4. The front doors generate the largest restoring moments while keeping the angle of the doors relatively small (less than thirty degrees)
5. The middle doors are not as efficient at correcting sway as the rear doors and not as efficient at correcting yaw as the front doors, and are therefore unneeded

The calculations were run using the principle that opening one of the side doors changes the direction of the momentum of the freestream flow through the pipe. This now *angular* momentum exits the opened door at the angle that the door is opened. This generates a moment of angular momentum, with the moment arm r being the perpendicular distance between the

center of mass of the container to the angular momentum vector. The rear doors generate the smallest moment of angular momentum because, due to the geometry of the pipes, the perpendicular distance (the moment arm r) between the center of mass of the TRICON and the angular momentum vector is minimized (for small angles ~thirty degrees). The front doors generate the largest moment of angular momentum for the opposite reason: the moment arm r is maximized.

Due to these conclusions, the middle doors were deemed useless and removed from the design, the rear doors were made larger to maximize the possible restoring side force while keeping unwanted moments small, and all doors are allowed to open up to an angle of thirty degrees. Since opening any doors, even the rear doors, will generate moments, the pipes are allowed to “self-correct” to minimize or eliminate any generated unwanted moments. As shown in Figure (13), opening the left rear door generates a small but noticeable counter-clockwise moment. The right front door then is opened to generate an equal and opposite clockwise moment to cancel the two out. Opening the right front door will generate a side force in the opposite direction as the left rear door, reducing the net effective side force that the pipes can generate. However, because the front door is smaller and does not need to be opened to as large an angle as the rear door, the negative counter-side force that the right front door generates is small. Figure (14) shows the modified pipes design, with the rudder (discussed in the next section) added on.

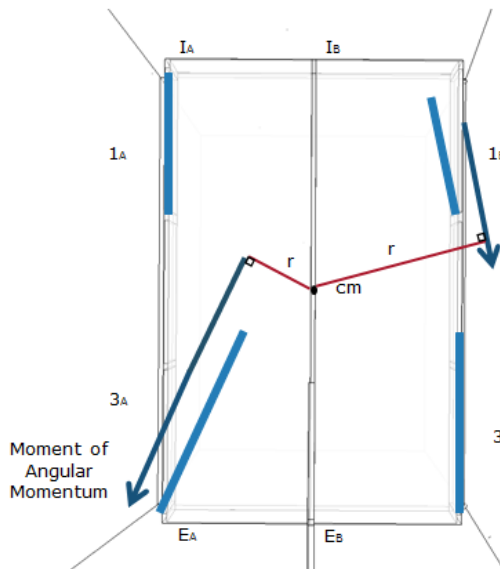


Figure 13: Self-Correcting Design

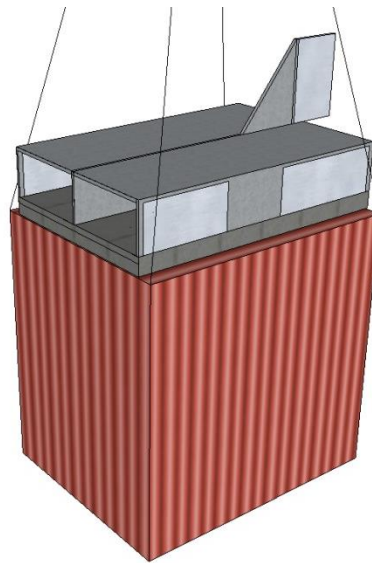


Figure 14: Fourth Design of the Pipes (with Rudder Included)

Later research showed that the rudder is a much more effective tool at correcting yaw than the pipes, and can easily counteract any unwanted moments generated by the rear doors of the pipes without producing any significant side forces. In effect, the pipes now solely correct sway motion, leaving all yaw motions to be corrected by the rudder. This led to the removal of the front doors of the pipes, simplifying the design and the hardware required for them, as now there are only two moving parts to the pipes design (the two rear doors). In final design of the pipes and rudder system, the pipes are also angled back to allow unimpeded flow into them when the TRICON rotates back and up in the flow; the final design is shown in Figure (15).

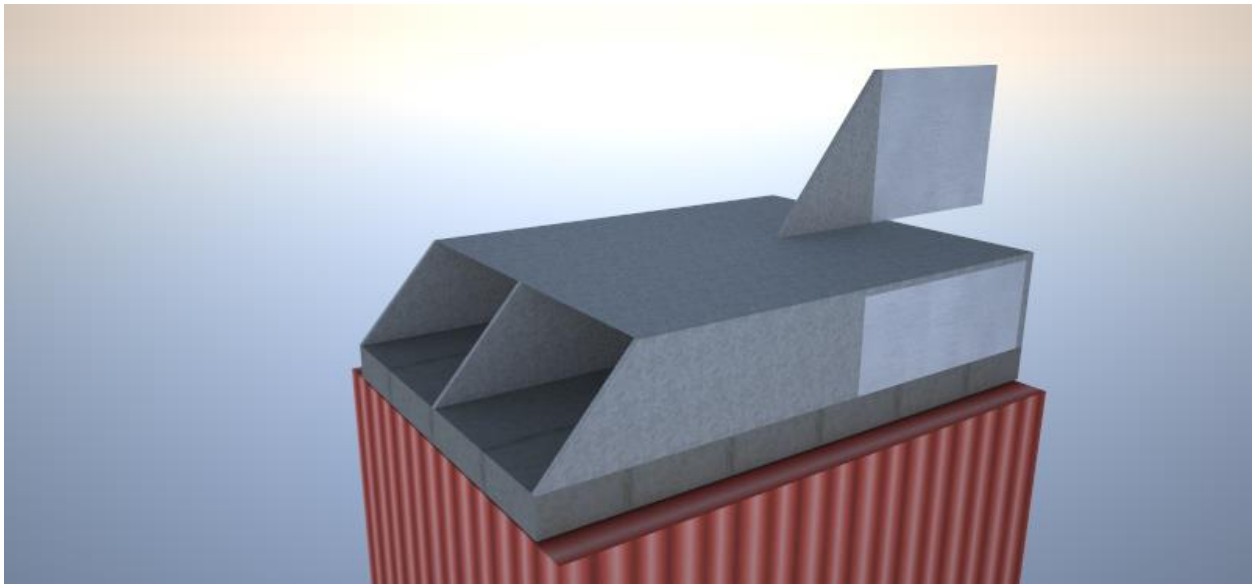


Figure 15: Final Design of the Pipes (with Rudder Included)

3.1.2 Development of the “Rudder” Design

The rudder and vertical stabilizer concept was derived out of an idea to actively stabilize a helicopter sling load in a manner similar to how a plane is stabilized using active control surfaces such as ailerons, rudder, elevators, etc. Focus was placed on simplicity and proven control surfaces that would be easy to integrate. The main instabilities that prompted the development of this part of the stabilization system were yaw instabilities and lateral side-to-side (sway) instability.

The rudder was derived from a standalone system which was comprised of a rudder and a pair of elevons. The goal of this was to be able to control yaw, pitch, and sway. However, a helicopter sling-load system has fewer degrees of freedom than a conventional aircraft. After careful analysis the elevons were deemed to be unneeded and were removed from the design. The elevons were found unnecessary because they would be ineffective in controlling sway since the sling load system is “hung” and cannot conventionally “roll” like an aircraft. Pitch control

was also deemed as superfluous as a sling load is generally rigged to fly 3 to 5 degrees nose-down and will natural move in the opposite direction of flight, causing it to pitch further.

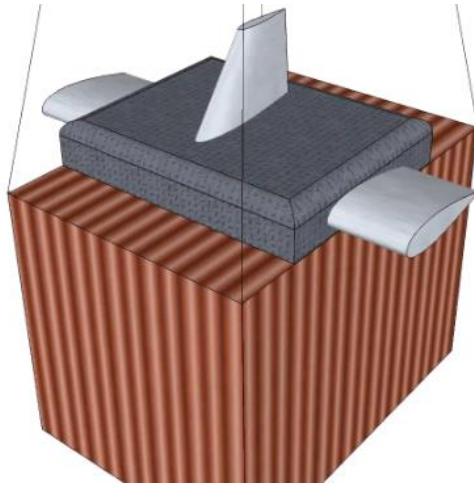


Figure 16: Initial Design

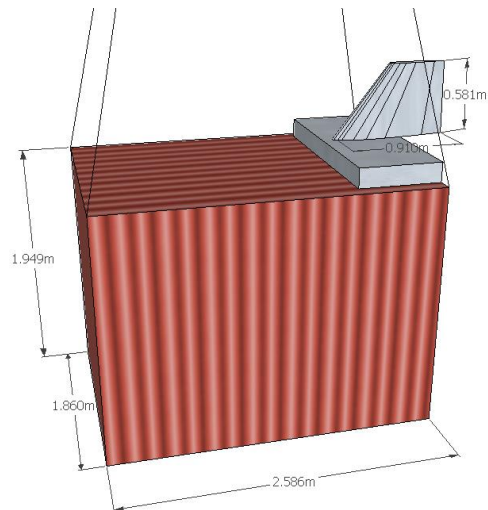


Figure 17: Final Rudder Design

The rudder in this system was initially a full, swept back, wing based on a symmetric airfoil. It was positioned towards the back of the sling load with the trailing edge meeting the rear edge of the load. The design featured no vertical stabilizer and the rudder was responsible for any and all yaw control. Once the elevons were removed the design of the system proceeded with only yaw stabilization in mind. This led to an almost complete redesign of the rudder system in order to increase its effectiveness. The size of the rudder, while very important in determining its effectiveness, was set to an arbitrary value for preliminary design purposes.

The effectiveness of the rudder also relied heavily on the horizontal distance between where the rudder force acted and the center of mass of the load, known as the moment arm. The longer the moment arm, the more moment the rudder can create. At this point the rudder was as far back on load as possible, while still keeping the trailing edge at the back edge of the load. In order to increase the length of the moment arm the rudder was redesigned without any sweep so that it could be positioned further back with the majority of it behind the back edge of the load. Only a small portion of the rudder would be in front of the back edge of the load so that it could be attached to the shaft that would control its position.

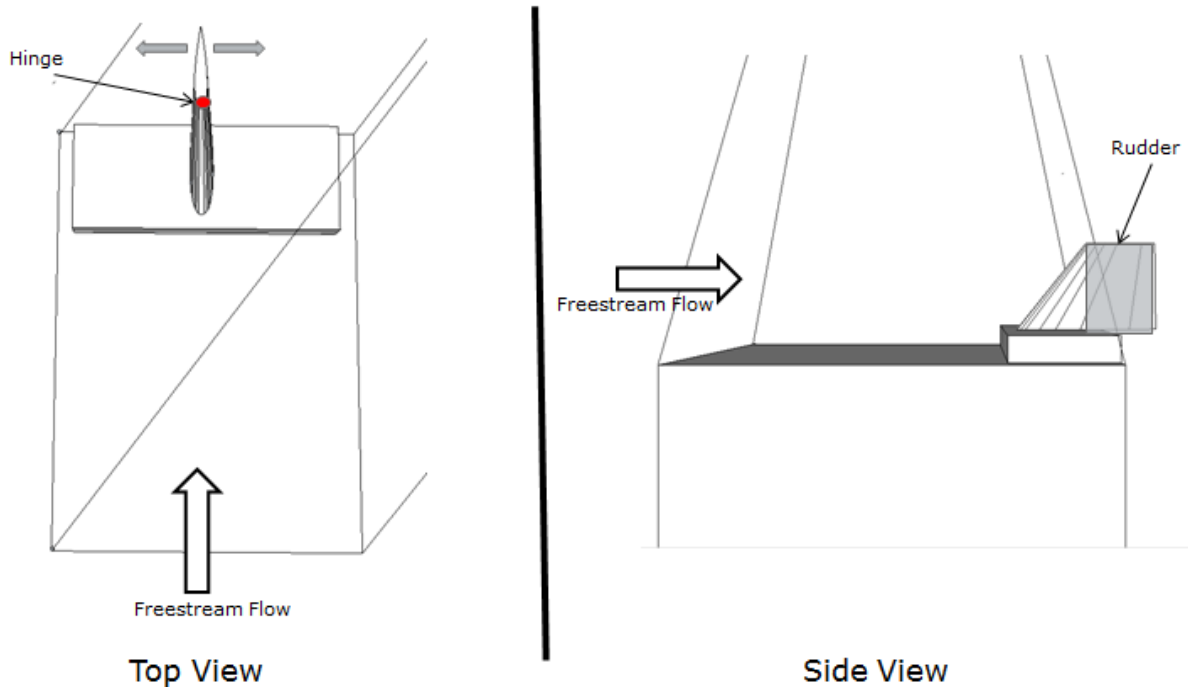


Figure 18: Rudder System

In order to supplement the rudder, a vertical stabilizer was also added to the design. While the focus of the design was active stabilization, a vertical stabilizer added an element of passive stability and enhanced the active rudder control in preventing adverse yawing of the load. The vertical stabilizer was placed in front of the rudder with a small gap in between the two. The vertical stabilizer was based off of the same chord length and wing span as the rudder; however it was swept back at 45 degrees from the leading edge. Once the preliminary design was completed, the system was put through a complete aerodynamic analysis in order to determine the forces and moments involved it generated.

The aerodynamic analysis also compared symmetric airfoil profiles to a thin flat plate profile in order to determine whether the vertical stabilizer and the rudder would be streamlined or whether they would remain thin flat plates. The NACA 0016 airfoil was used in this comparison since the rudder and vertical stabilizer must be modeled using a symmetric airfoil, one without camber, in order to generate the same forces in either direction. An analysis of the NACA 0016 airfoil was performed in XFLR5 for a range of Reynolds numbers that represented speeds up to the maximum flight speed for a sling load mission with angles of attack from 0 to 20 degrees. A similar analysis was performed using thin flat plate airfoil theory. The resulting lift curves were linear, since small angle assumptions were used, and had slopes that were almost identical. The thin flat plate design was chosen for both the rudder and vertical stabilizer because of the ease of prototyping. The symmetric airfoil had a significant advantage over the thin flat plate in terms of drag. Being a streamlined body, the symmetric airfoil produced less drag. This

advantage was not found to be enough to justify the prototyping and manufacturing complications as a sling load is a bluff body, generating large amounts of drag by nature. The drag produced by the rudder and vertical stabilizer were also along the center of mass in the horizontal plane, meaning that it would generate no adverse torque or yaw in a neutral position. For thin plate/airfoil theory to hold, the rudder and vertical stabilizer were designed to have a maximum thickness of 10% of their respective chord lengths.

After design analysis was performed, an aerodynamic analysis was performed using MATLAB in order to quantify the forces that the vertical stabilizer and rudder would generate at an average cruise speed for a helicopter sling load. Once the analysis was completed, the forces were altered through changes made to the chord length and the wing span of the rudder and vertical stabilizer to create the necessary amount of force. A MATLAB simulation of the helicopter sling load system set approximately 400 Newtons of force as the target maximum force for the rudder to produce at its maximum angle of incidence. Small angle assumptions were used for the purposes of analysis and simplicity. This limited the maximum angle of incidence, both positive and negative, to twenty degrees. The rudder was then sized to reach the target force with the goal of keeping the chord length longer than the wingspan in order to avoid encroaching on the sling legs. The vertical stabilizer was given the same dimensions as the rudder and produce roughly half the force as its area was reduced to half that of the rudder because of its 45 degree leading edge sweep angle.

Once the dimensions and the positions of the rudder and stabilizer were set, an analysis of the moments created by both of the surfaces was performed in MATLAB. The analysis was based on a rigid body of uniform density assumption. This set the container's center of gravity at its geometric center. Both the stabilizer and the rudder are directly behind the center of gravity on the horizontal plane. The moment arm lengths were calculated based on the point where the force from the stabilizer or rudder acts on the load. The rudder is a straight wing (no sweep), therefore it can be assumed that the rudder forces act at the quarter-chord point of the rudder. The vertical stabilizer is a swept wing, making it necessary to calculate the mean aerodynamic chord. The vertical stabilizer forces were assumed to be acting at the quarter-chord point of the mean aerodynamic chord. The sweep on the vertical stabilizer was beneficial because it moved the point through which the forces were acting further away from the center of gravity of the container. The sweep therefore lengthens the moment arm and makes the vertical stabilizer more effective at creating a moment about the center of gravity.

Further analysis of the rudder revealed that it would be beneficial to mass balance the rudder. Mass balancing the rudder involved moving both the vertical stabilizer and the rudder closer to the center of gravity. This move was necessary so that a quarter of the rudder's chord would be directly over the container and the rudder could be hinged at its quarter-chord point. Moving the hinge from the leading edge to the quarter chord point mass balanced the rudder which would serve to mitigate aerodynamic flutter and make the rudder a better overall control surface.

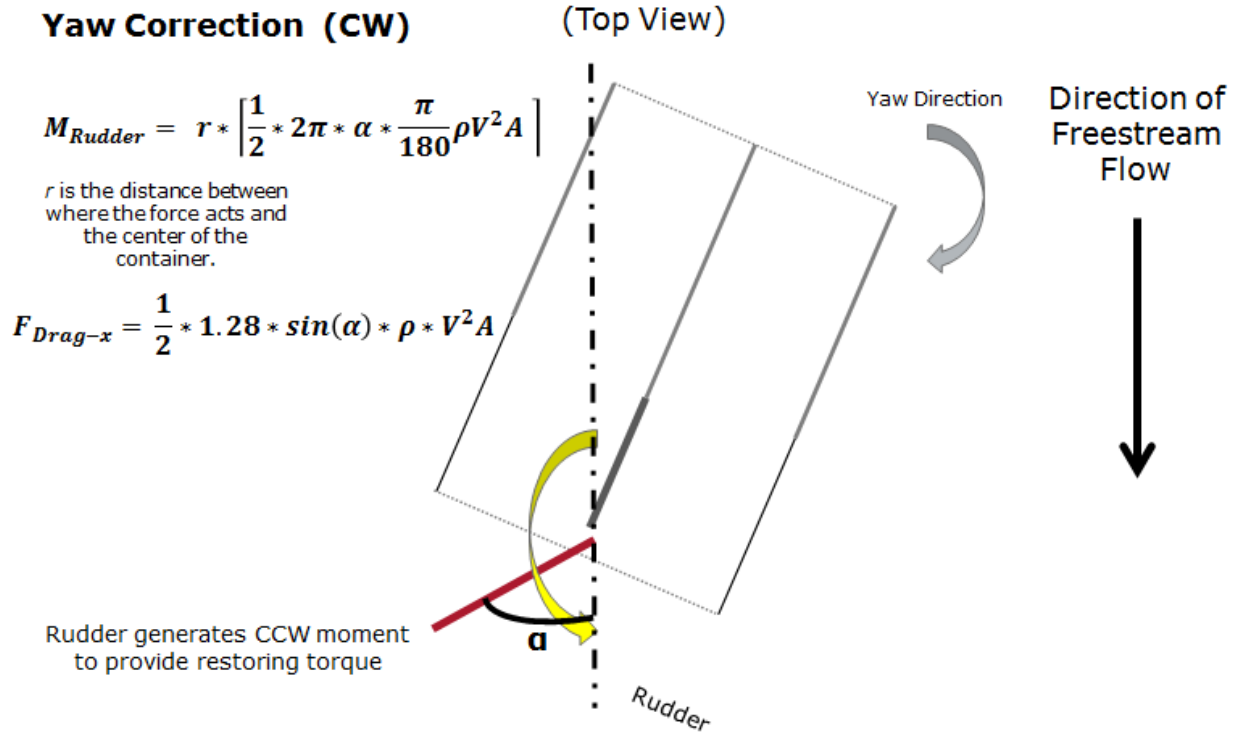


Figure 19: Principles of Rudder Yaw Correction

A final analysis of the entire system was performed after all of the modification to verify the forces and moments made by the rudder and the vertical stabilizer. Once the analysis was complete, work began on a 1/10th scale prototype to be tested in the WPI reciprocating wind tunnel in the fluids lab of Higgins Laboratories. A testing prototype was necessary in order to confirm the results of the aerodynamic analysis of the system. The testing prototype would not be actively controlled but it was still necessary for it to replicate a number of different rudder angles of incidence. In order to accomplish this, a model of the system was created with a rudder that had the ability to be pinned into a number of different angles of incidence. The test model was created in SolidWorks and rapid prototyped out of ABS plastic using a 3D printer.

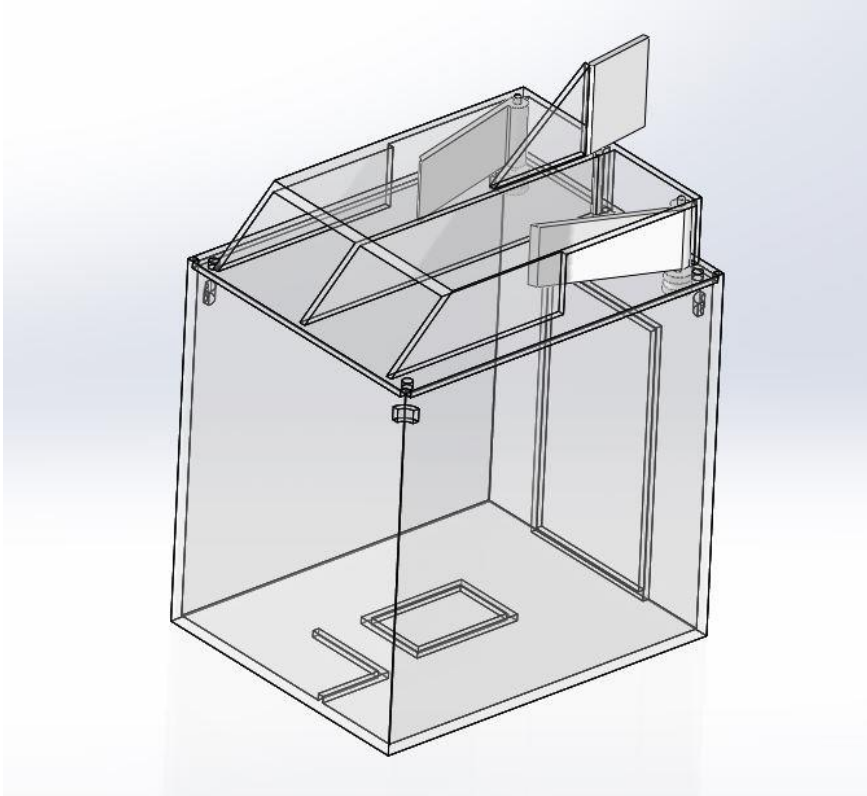


Figure 20: Rudder in Final SPARTA System Prototype

The vertical stabilizer was printed already attached to the model plate. The plate to which the test model was to be attached featured a hole through which the quarter chord pivot point of the rudder could be pinned through and a number of holes at various angles of attack to which the rear portion of the rudder could be pinned to. The test model of the rudder features an extra hole on it to allow it to be pinned into the proper angle of attack.

3.2 Development of the Hardware and Actuation Scheme

3.2.1 Hardware

The hardware and actuation scheme behind the SPARTA system was developed simultaneously with the actual system. Work on the hardware and actuation portions of the control system began when the pipes and rudder/vertical stabilizer systems were separate. However, the majority of the progress happened after the rudder and pipes were combined into one hybrid system.

From the start, the goal of this system was active stabilization. In order to accomplish that, control and actuation hardware was necessary. The main components of the system were actuators to move the control surfaces and a controller to properly move the actuators based on a set of control laws. The first task was finding a controller for the system. This had a lot to do with the control laws which were not developed at that point. This was not a major issue as the goal was to select a performant but easy to integrate controller. From the beginning of the

hardware and actuation design, it was decided that a single-board microcontroller would be best suited for this system. Research was performed to narrow down the options in terms of controllers until it was decided that an Arduino microcontroller would be used. Arduino was chosen to supply the microcontroller because the team had some previous experience with Arduino controllers as well as immediate access to Arduino's base model controller, the Arduino UNO.

Arduino proved to be a good decision because of its small sized controllers, vast libraries of pre-written code, and its convenient integrated development environment (IDE). The Arduino IDE is based on C and makes it easy to write a program for the Arduino and upload it the controller. Arduino offers many different controls with various features, performance characteristics, sizes, and weights. The final decision as to which Arduino controller would be used was postponed until the later stages of the system design. In the end, the decision came down to a number of different requirements. The size of the controller was an issue but none of the options that fit the system's needs were too large in any way. The other important factor was performance and capacity. Since size was not an issue, it was important to use the most performant Arduino with the largest capacity. The Arduino DUE was selected as the microcontroller for the system based on its large flash memory capacity (512 kb for code), 84 MHz clock speed, and its 32-bit core. The large flash memory capacity is very important in terms of coding because it allowed for the most leeway since the coding since the team did not have previous experience with C. The complexity of the setup and control code was also unknown and it was most prudent to have a controller with as much memory as possible. The 84 MHz clock speed also indicates a fast and powerful processor which is necessary for our control system to function well. The team's limited familiarity with Arduino microcontrollers required them to do a lot of practice coding, testing, and etc. with the Arduino UNO. Between the IDE and the very well written tutorials and examples that are provided by Arduino, the team was able to become proficient in the use of the Arduino and the coding involved.

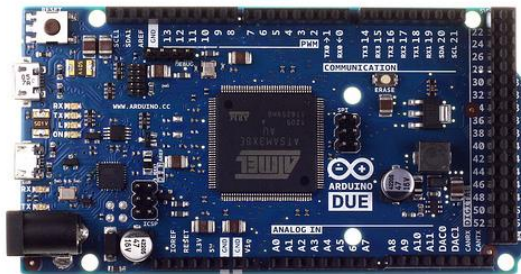


Figure 21: Arduino DUE

The SPARTA system was designed to correct unwanted sway and yaw of a helicopter sling load. This active control system relies on readings from an Inertial Measurement Unit (IMU). The IMU readings are used by the control law to calculate the necessary position of the control surfaces in order to stabilize the load. A 9 degree of freedom IMU, made by Adafruit,

was chosen for this system. The IMU featured a 3-axis gyroscope with a ± 250 , ± 500 , or ± 2000 degree-per-second scale, a 3-axis compass with a ± 1.3 to ± 8.1 gauss magnetic field scale, and a 3-axis accelerometer with a $\pm 2g/\pm 4g/\pm 8g/\pm 16g$ selectable scale. It was completely compatible with the 3.3 volt Arduino DUE and used an I2C two wire interface (TWI), making wiring, communication, and integration seamless. This IMU was designed with Arduino integration in mind and also featured a code library for easy access to the IMU data. Once purchased, the IMU was easily integrated into the hardware and actuation scheme.



Figure 22: Adafruit 9-DOF IMU

The IMU serves two purposes, the first being to supply the Arduino and its control laws with acceleration and angular rates, the second being to measure the overall stability of the system. In order to quantitatively measure the stability of the system, it was necessary to also record the IMU data, real-time, on a digital medium. A micro-SD card breakout board was needed in order to record this data. The board that was chosen was also made by Adafruit and was easily integrated into the hardware scheme using the Arduino DUE's SPI interface. This allowed the hardware scheme to log all of the IMU values as well as any other pertinent data.

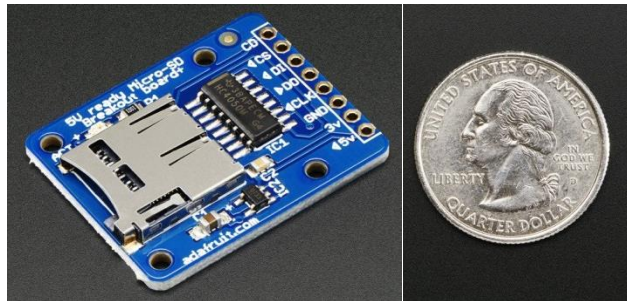


Figure 23: Adafruit Micro-SD Card Breakout

The IMU and micro-SD breakouts from Adafruit completed the hardware scheme and added minimal weight and size to the overall design. Initially, when the number of servos for the scheme was large, a servo driver was going to be used to take the load off of the Arduino. However, final development removed a number of servos to the point where it was decided that the servo would attach to and be directly controlled by the Arduino DUE itself. A template script was written in which the servos, IMU, and SD card datalogging were all set up and functioning, allowing for the control laws to be migrated and easily coded in and integrated onto the DUE.

Once the control laws were completed, they were manually ported onto the Arduino and integrated into the template script. MATLAB has the capabilities to port code directly into an Arduino script; however it was deemed that a manual port would be better for the overall simplicity. MATLAB uses countless libraries and imbedded functions which could take up too much space and create a less than optimal script for the Arduino. With high dynamic response in mind, the Arduino control code [Appendix D] was written as simply as possible. In order to further simplify the code porting process the control laws were broken down into their simple algebraic forms, omitting the need for matrix math and manipulation. This cut down on the complexity and size of the code as a matrix math library was no longer needed for the Arduino. The control code was optimized for speed and size with debugging code commented out of the final version. The final version of the code was written to be void of any serial port communication and delay functions. The SD card data-logging (included as a stand-alone script [Appendix E] was also deemed to be too processor power, memory, and time intensive and was not included in the final code. The final code ended up occupying roughly 10 percent of the Arduino DUE's total flash memory.

3.2.1 Actuation & Power

Another major aspect of the hardware and actuation scheme is the actuators. Early on, it was decided that linear actuators and motors of some sort would be moving whichever control surfaces the design needed. Once the hybrid design was created and chosen for development, it was apparent that the design would need one motor to control each of its control surfaces. The two most appropriate options were DC stepper motors and DC servo motors. Servo motors were chosen for the system's actuation scheme because stepper motors lacked positional feedback. Servos also provided better torque, due to integrated gear trains, and faster operation speeds. The drawback of the servos was that they were limited to 180 degrees of motion or less. This, however, was not a problem for this particular application.

The next step in the actuation scheme design was to decide whether to use digital or analog servos. The decision was made to use digital servos because they feature all of the same parts as an analog servo, but they include a microprocessor to process the pulse width modulated signals from the Arduino. This allows for higher frequency voltage pulses which make the servo respond faster, accelerate smoothly and more quickly, and provide much better, constant, holding power/torque. Digital servos also have a smaller dead band, usually around 2 microseconds. The only drawback to digital servos was the increased cost, which was not an issue for this application.

Servos manufactured by HiTec were chosen for this project because of their excellent quality. Servo size and weight was an issue that limited the servo selection to standard sized servos and smaller. A range of options was researched and discussed, but in the end the smallest and the largest servos from the available selection were purchased for testing. These were the HS-5065MG, the smallest, and the HS-7985MG. Both servos run off of any 4.8 or 6 volt power supply and draw a maximum of 2 amperes at full power. Using a 5 volt power supply, the 5065

can provide up to 1.8 kg·cm of torque and has an operating speed of .14 seconds per 60 degrees with no load. The 7985 can provide up to 10.4 kg·cm of torque and has an operating speed of .16 seconds per 60 degrees with no load using the same supply.

Servo	Weight [g]	Length [mm]	Width [mm]	Height [mm]
HS-5065MG Digital	12	24	12	24
HS-7985MG Digital	62	40	20	37

Table 2: Servo Specifications

The exact number of servos needed changed as the SPARTA system was continuously developed and improved, but either of the two options or a combination of both is acceptable options for the prototype. The final SPARTA system required only three servos. Due to weight and size constraints it was decided that all three servos would be HS-5065MGs. Using the SPARTA system analysis script [Appendix F], the servos proved to provide adequate torque for the design.



Figure 24 : HS-5065MG Servo



Figure 25: HS-7985MG Servo

Initially the prototype was to be battery powered. However, weight restrictions and possible hardware issues showcased the need for external power. An external power scheme was developed using two separate external AC to DC power supplies: a 10 Amp 5 VDC and a 1 Amp 9 VDC supply. The 5 VDC supply was chosen for the servo motors because it was easily available and could handle up to five servos at full power. The 9 VDC supply was chosen for the Arduino as it was readily available and most commonly used to power Arduinos of the DUE's size and specifications. The DUE, however, can be powered by any input voltage from 7 to 12 VDC according to manufacturer specifications. In order to power the system without interfering with the container or sling legs, the positive and negative components of each power source were split up and 4 wires, 2 (positive and negative) for each power supply, were loosely strung around the sling legs so as to power the system without interfering with the sling legs.

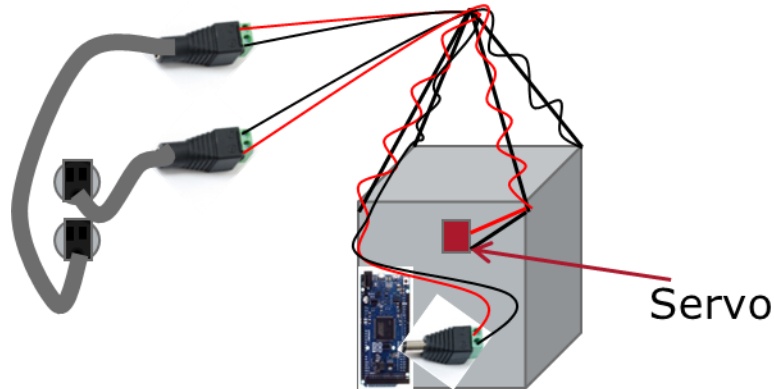


Figure 26: Depiction of the Power Scheme Implementation

3.3 Mathematical Model Development

The development of the mathematical model was an important first step in the project. The model is needed to develop and test our control laws. The model represents the dynamics of a slung loaded rectangular cargo container with a constant wind passing over it, slung beneath a fixed point through four sling legs attached to the corners of the top of the container. The model assumes that the container is rigid and of uniform density. It was decided that the easiest way to numerically simulate this model was to use the program MATLAB. The code for this numerical simulation can be seen in Appendix C.

In the numerical simulation the position orientation and movement of the simulated container is represented in a series of states. The values for each of the states are solved for over a provided time using the MATLAB function `ode45`. The `ode45` function does this by using a function called the state-derivative function. The state derivative function calculates the state derivatives using a system of equations that determine the forces and moments acting on the container based on the current state and the control inputs. The control inputs are calculated at each point in time by the control laws, which use the current state values to determine the necessary change in the control inputs to achieve the desired stabilization. The `ode45` function numerically integrates these derivatives over a timespan to determine the value of each state at each point in time. This process is shown below.

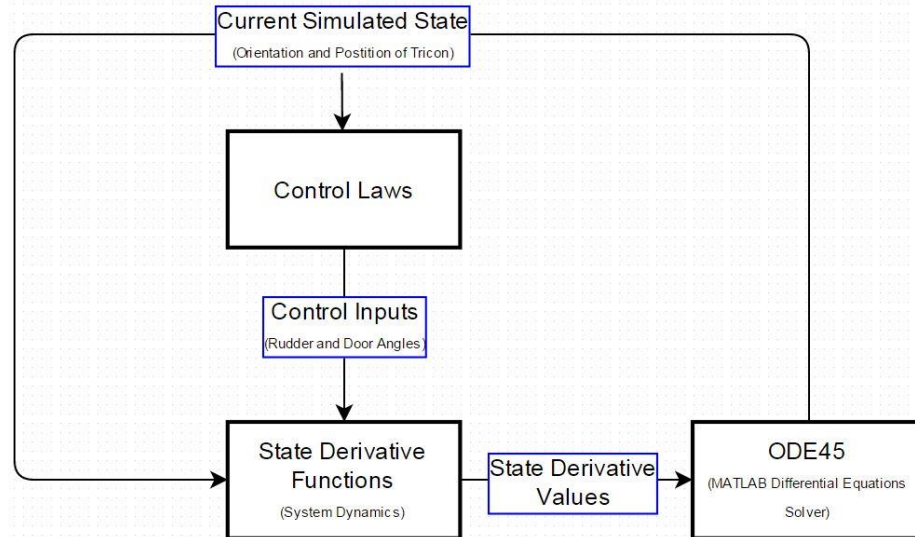


Figure 27: MATLAB Simulation Flowchart

The creation of the state derivative function is the central part of the numerical simulation. Over the course of this project there were two different versions of this function created. The first function represents the sling load system in a series of twelve states. These states are the X, Y, Z position and the X, Y, Z velocity of the container, as well as the yaw, pitch, roll Euler angles and the angular velocity of those angles. The second function represents the sling load system with only six states. These states are the yaw, pitch, roll Euler angles and the angular velocity of those angles. The main difference between these two functions is how the forces and moments from the sling legs are represented.

The first model calculates the forces and moments from the sling legs by assuming the sling legs have an un-stretched length and that they have a known modulus of elasticity. The attachment point of each sling leg is known in the body reference frame (the four top corners of the container). They are each represented in the code by a position vector originating from the center of the container and ending at each of the attachment points. This position vector is then converted to coordinates in the inertial reference frame by multiplying it by a 321 rotational cosine matrix whose values are determined from the three Euler angles of the container which represent the orientation with respect to the inertial reference frame. Once the position vector of each of the attachment points with respect to the center of the container is known in the inertial reference frame the distance between the attachment points on the container to the single fixed attachment point is easily calculated by summing the containers current position (X, Y, and Z values) and the attachment point positions. This distance is then used to calculate the forces from each sling leg by assuming the sling leg is elastic with known modulus of elasticity. Once the force vectors from each sling leg are known the moments acting on the container in the body reference frame are calculated by taking the cross product of the moment arm (the attachment point vectors) and the forces vectors at the respective attachment point. These forces and

moments are then summed with the other forces and moments in the body frame to calculate the state derivatives.

The second state derivative function assumes the sling legs are rigid, and because of this the Cartesian position and velocity of the container is directly related to the orientation and rates of rotation. Because of this, a traditional system of equations cannot be used to calculate the state so the Euler-Lagrange equation is used to determine the state derivatives. The Euler-Lagrange equation relates the kinetic and potential energy of the system to the forces and moments acting on the system for each state. The Euler-Lagrange equation is shown below where K is the kinetic energy and V is the potential energy.

$$L(q, \dot{q}) = K(q, \dot{q}) - V(q)$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \sum F_i + \sum M_i$$

Equation 1: Euler-Lagrange

The solution to this equation for each state produces a value for that state derivative. Since the system already takes into account the sling legs the forces and moments that are summed in these equations do not include the forces and moments from the sling legs.

Both these functions were created however the second function is used to calculate all results seen in the results section and the full model can be seen in the appendix. The next portion of the state derivative function is to determine the forces and moments from the aerodynamic effects, and the SPARTA system.

The calculation of the aerodynamic effects exert on the container is a very complicated step considering that the container is a rectangular box which is a bluff body. For these effects we decided to use equations from a model previously developed by a D.I. Greenwell at City University in London England and published in the journal article *Modelling of static aerodynamics of helicopter underslung loads*. These equations can be seen in detail in the appendix. This model is specifically designed to provide the aerodynamic forces and moments on a container based on the velocity vector of airflow over a static rectangular container. In our case we are modeling a dynamic container however since our goal is to stabilize it and keep it from moving we believe that the model can be used and trusted especially when it is stabilized.

The final step in the state derivative function is determining the forces and moments from the control scheme developed by this project (SPARTA system). Part of the design process of the SPARTA system was to determine what forces and moments the design would exert on the container based on how the pipe doors and rudder would move. These equations which can be seen in the appendix, are also used in the model to calculate the forces and moments exerted on the container from the SPARTA system.

3.4 Control Law Development

The creation of the control laws is an important part of this project. The control laws take a series of inputs and then calculate changes to the systems control inputs. In the case of this project and design the control inputs are the angles to which the pipe doors and rudder are moved to. This in turn changes the forces and moments that are exerted on the container in order to stabilize it. For this project the inputs to the control laws are three states, the containers side to side sway velocity, the containers yaw angle, and the containers yaw angular velocity. The full MATLAB code for our control laws can be seen in the appendix inside the full MATLAB numerical simulation code.

It was decided that the control process of Linearization of the system should be used to create the control laws. This involves representing the system in two different matrices and A matrix and a B matrix. Matrix A is the Jacobean of the state derivatives with respect to the states and matrix B is the Jacobean of the state derivatives with respect to the control inputs. These matrices were calculated by taking the derivative of the equations for the state derivatives developed in the mathematical model with respect to each of the different states for the A matrix and with respect to each of the control inputs for the B matrix.

The next step in the control laws is to select a point to linearize the system about. This means that the values for every other variable besides the inputs would correspond to a specific orientation and position that the container would reside in. This allows the control laws to estimate how a change in the control inputs would affect the system. The next and key part of the control laws is the calculation of the gain constant K. K is the constant that relates the difference in the desired inputs of the controller, which is the difference between the current state and the desired state, to the calculated change in the control inputs. For this project, the calculation of K was calculated using a linear quadratic regulator or LQR. LQR is an algorithm used in optimal control theory. It is a way to determine the optimal state control feedback based on the desired state and the system dynamics. The resulting equation is the form of the control laws used in this project.

$$(Change\ In\ Control\ Inputs) = K * (Difference\ Between\ Desired\ And\ Current\ State)$$

Equation 2: Control Laws General Form

For our project the gain matrix K is calculated using MATLAB's lqr function. This function has four inputs, the A and B matrix that were calculated earlier and an R and Q matrix. The R and Q matrices are values for determining how much each of the control inputs want to be used. The R matrix places a penalty on using different control inputs while the Q matrix encourages the use of them. In the case of this project we needed to limit the use of our rudder and increase the use of our pipe doors so we changed our R matrix value for the rudder to 10 instead of 1 to decrease the use, and we changed the Q matrix values for the pipe doors to 3.5 instead of 1 to increase their use. This resulted in our current control law equation shown below.

$$\begin{bmatrix} \Delta \theta_{rudder} \\ \Delta \theta_{left} \\ \Delta \theta_{right} \end{bmatrix} = \begin{bmatrix} -4.055 & -0.692 & -0.590 \\ 0.664 & 0.201 & -0.096 \\ -0.664 & -0.201 & 0.096 \end{bmatrix} * \begin{bmatrix} \Delta \psi \\ \Delta R \\ \Delta \dot{Y} \end{bmatrix}$$

θ = Control Angles

ψ = Yaw Angle

R = Yaw Angular Rate

\dot{Y} = Sway Velocity

Equation 3: Control Laws

3.5 Kalman Filters

Along with the development of the control laws, the mathematical model was also used to develop Kalman filters for the IMU. Kalman filters are filters that compare the current measurement to the current measurement estimate in order to determine the change in the current measurement estimate. Kalman filters were developed for this project for the purpose of calculating a more accurate input to the control laws instead of just using the noisy measurements from the IMU.

The mathematics behind Kalman filters are shown below in the following general expression.

$$\hat{X} = A * \hat{X} + B * U + K * (Z - H * \hat{X})$$

Equation 4: Kalman Filter General Form

For the purpose of our project two filters were created, one for the yaw angle and yaw angle rate, and the other for the sway (side to side) velocity and roll angle. The developed equations for the two filters are shown below.

$$\begin{bmatrix} \hat{\psi} \\ \hat{R} \end{bmatrix} = \begin{bmatrix} \frac{\partial \psi}{\partial \psi} & \frac{\partial \psi}{\partial R} \\ \frac{\partial \dot{\psi}}{\partial \dot{\psi}} & \frac{\partial \dot{\psi}}{\partial R} \end{bmatrix} * \begin{bmatrix} \hat{\psi} \\ \hat{R} \end{bmatrix} + \begin{bmatrix} \frac{\partial \psi}{\partial U_{rud}} & \frac{\partial \psi}{\partial U_{right}} & \frac{\partial \psi}{\partial U_{left}} \\ \frac{\partial \dot{\psi}}{\partial U_{rud}} & \frac{\partial \dot{\psi}}{\partial U_{right}} & \frac{\partial \dot{\psi}}{\partial U_{left}} \end{bmatrix} * \begin{bmatrix} U_{rud} \\ U_{right} \\ U_{left} \end{bmatrix} + K * \left(\begin{bmatrix} \psi_{imu} \\ R_{imu} \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} \hat{\psi} \\ \hat{R} \end{bmatrix} \right)$$

Equation 5: Yaw and Yaw Rate Kalman Filter

$$\begin{bmatrix} \hat{\phi} \\ \hat{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & L \end{bmatrix} * \begin{bmatrix} \hat{\phi} \\ \hat{v} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} * [A_{y_{imu}}] + K * \left(\begin{bmatrix} \phi_{imu} \\ P_{imu} \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} \hat{\phi} \\ \hat{v} \end{bmatrix} \right)$$

Equation 6: Roll Angle and Sway Velocity Kalman Filter

In these equations the subscript “imu” denotes a measurement from the inertial measurement unit, and L is the distance from the top of the inertial measurement unit to the sling leg attachment point on the helicopter or the wind tunnel.

The next part in developing the Kalman filters was determining the Kalman gain K. The Kalman gain K is solved for over time using the following equation where the matrix P is a function of time and R is the measurement error covariance matrix.

$$K = P(t) * H^T * R^{-1}$$

Equation 7: Kalman Gain

The since the matrix H and R are already known for each filter the only matrix that needs to be solved for is the state estimate covariance matrix (P) as a function of time. For this project the ordinary differential equation shown below is used to solve P(t) (Murray, 2010).

$$\dot{P} = A * P + P * A^T - P * H^T * R^{-1} * H * P$$

$$P(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Equation 8: State Estimate Covariance Matrix

With these filters designed they were added to the mathematical model to test their effectiveness.

Since the purpose of Kalman filters is to eliminate noise from a measurement modifications to the model were made to add a simulated noise profile. The modified model also linked the estimated Kalman values to the control laws to more accurately simulate how the SPARTA system would work. These modifications are illustrated in the flow chart below and the Kalman filter code that is part of the model can be seen in Appendix C.

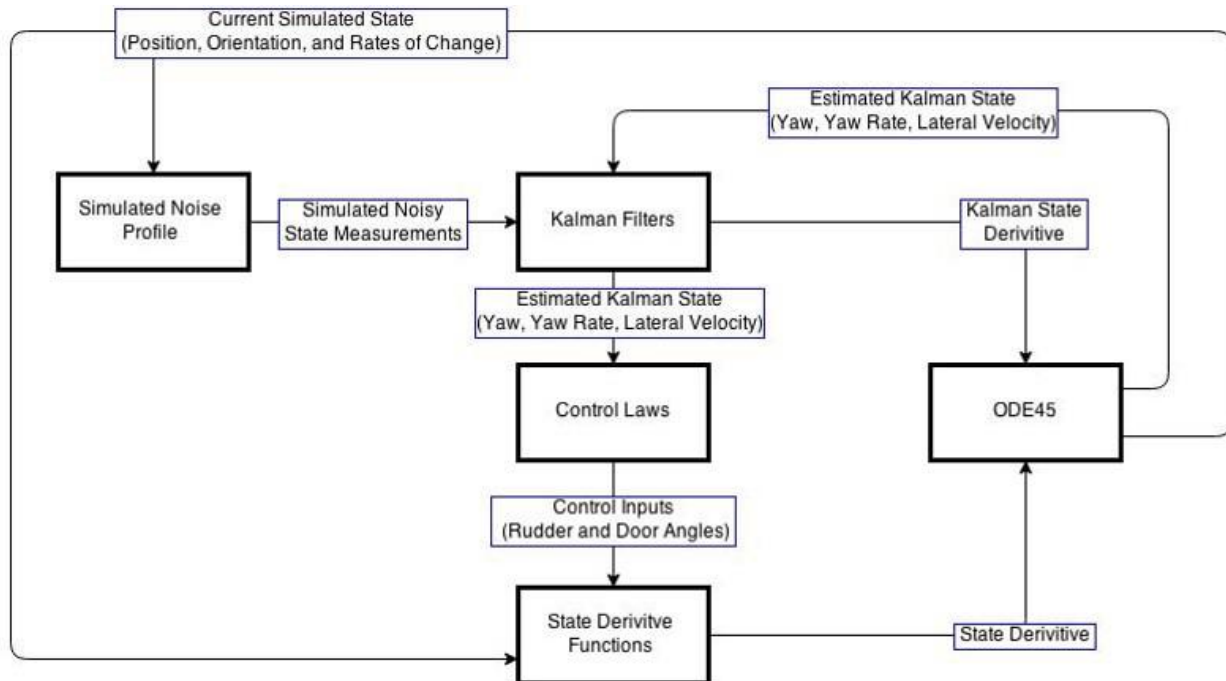


Figure 28: MATLAB Simulation With Kalman Filter Flow Chart

3.6 Wind Tunnel Testing

The team waited to rapid prototype the design until the majority of the aerodynamic analysis, Arduino programming, and control law creation was complete. In the meantime, two initial prototype TRICON containers were constructed, scaled to the exact dimensions of the eventual rapid prototyped version, and tested in the wind tunnel. The initial prototype TRICON containers were crafted using 0.25" poster-board, with the sling legs created from the same materials as Nyren's previous project: 2mm braided nylon rope, with 22-18 gauge O-rings connected to a simulated clevis, and 16-14 gauge O-rings connected to paper clips (simulating metal chains), which were attached to the TRICON. The setup can be seen in Figure 29.



Figure 29: Poster-board TRICON (note power cord - this was removed for subsequent tests)

During these early tests, many of the hardware components, including the Arduino and IMU, were included within the container to both weigh the poster-board down (since it is much lighter than ABS plastic), and to test the functionality of the electronics. The usual setup for these tests can be seen in Figure 30. The force transducer was tested as well; the results appeared promising and the team concluded that the transducer was working properly.

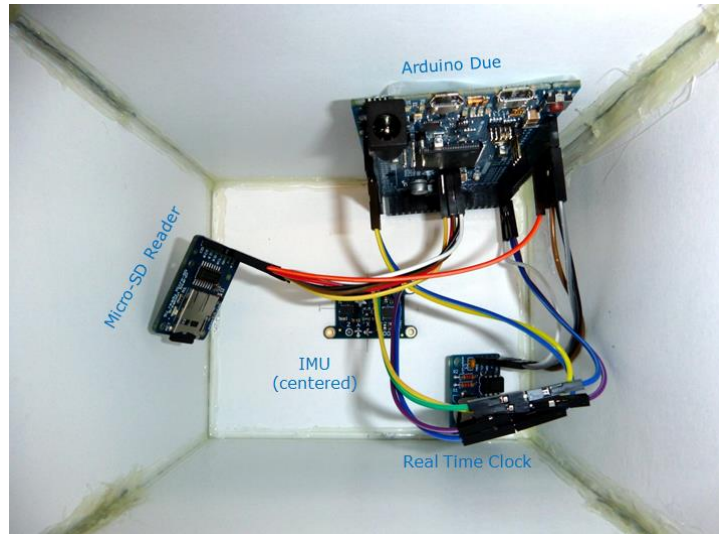


Figure 30: Inside of Poster-Board TRICON with Components

After some testing to see the behavior of the scaled-down TRICON in the wind tunnel, a second poster-board TRICON container was then created with detachable sling legs (using jewelry ring clasps), in order to test various wire configurations. It was determined from the tests with the first, non-detachable sling leg poster-board TRICON that having one thick power wire running up one sling leg severely altered the dynamics of the box. Thus, the wire would be split into four smaller wires, wrapped carefully around all four sling legs. The new poster-board TRICON is shown in Figure 31.



Figure 31: Detachable Sling Legs on the Second TRICON Prototype

Before submitting the final SPARTA prototype for rapid prototyping, the team constructed the SPARTA design using basswood and balsawood. These pre-prototypes served to test how the design would affect the stability of the TRICON, and whether any passive stability

was achieved. Two different designs were created; one without doors (to simulate the doors being closed), and one with the doors fixed at 10° , the newly-determined equilibrium position, since the control laws did not allow the team to optimize around 0° without greatly complicating the mathematics. The two designs are shown in Figure 32. All of the control surfaces (the doors and rudder) were fixed in place, so that the rudder essentially acted as an additional, larger vertical stabilizer.



Figure 32: Basswood and Balsawood SPARTA Pre-Prototypes

The team ran into difficulties weighing down these pre-prototypes to the weight of the rapid-prototyped container. Early results from the weighted-down container with the pre-prototype SPARTA system showed signs of some passive stabilization; however, the team later determined that the poorly-packed weights within the container had shifted and “pinned” one corner of the container in place during testing. Subsequent tests with the weight taped down and center properly generated less satisfying results. There was little passive stabilization; the TRICON’s average maximum yaw angle only reduced from 115° to 98° , and the SPARTA pre-prototype actually amplified sway instability, causing the TRICON to sway violently whenever the yaw was reduced, striking the side of the wind tunnel during one test. All of the tests were filmed for later analysis; a screenshot of one of these videos is shown in Figure 33.

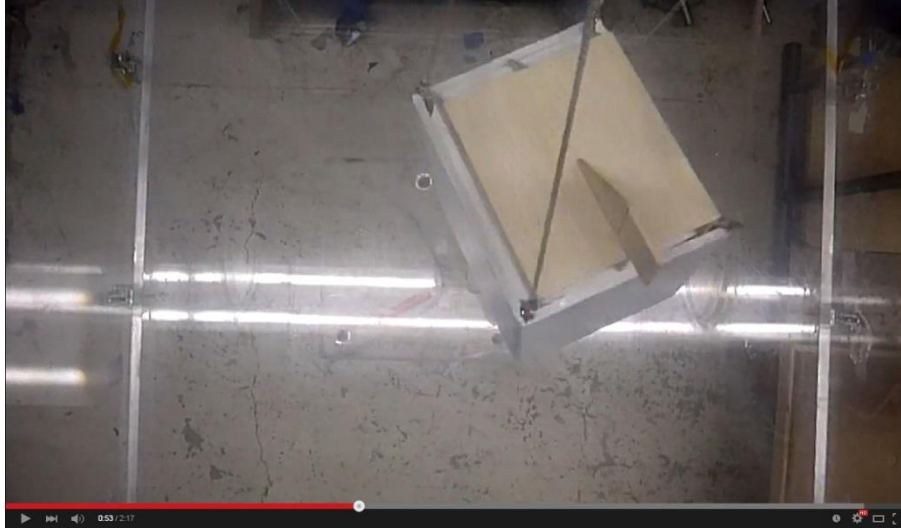


Figure 33: A Screenshot from One of the Wind Tunnel Test Videos

Due to the lack of suitable yaw correction by the pre-prototype SPARTA designs, the team decided to double the length of the rudder to increase the vertical stabilizer area. This new rudder is shown in Figure 34.

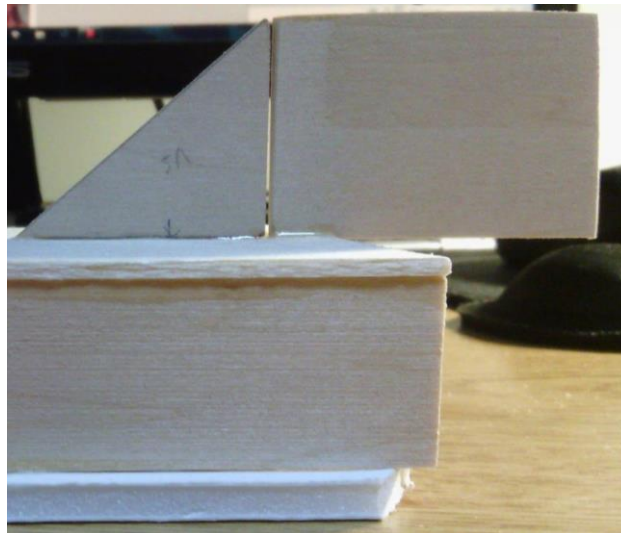


Figure 34: Larger Rudder

Subsequent wind tunnel tests with the larger rudder yielded the same results as the original rudder; little yaw stability was achieved, and sway was increased over the standalone TRICON. The team determined that the SPARTA design provides little to no passive stabilization and will therefore only rely on highly-precise actuation to provide active stabilization. While the results were not promising, the team noted that the actual active-control SPARTA would begin to correct the TRICON's movements starting in a freestream velocity of 0 m/s, where movements are much smaller. The SPARTA design would begin to correct

instabilities while they were very small, and thus the container would be prevented from reaching the magnitude of the large-scale instabilities seen in the preliminary wind tunnel tests.

3.7 Rapid Prototyping

The rapid prototyped TRICON container and SPARTA system was delivered to the team with a number of issues. None of the lids properly sat on top of container, one of the rear support pillars was broken before delivery, the pins for the rudder and doors were too thick and too fragile to use, and the doors were printed at half their proper height. Some of these issues can be seen in Figure 35. While the doors were reprinted quickly at their proper height, the team had to modify the design of SPARTA slightly. The intact pillar was removed from the other pipe; the doors themselves would act as support pillars for the back of the pipes.

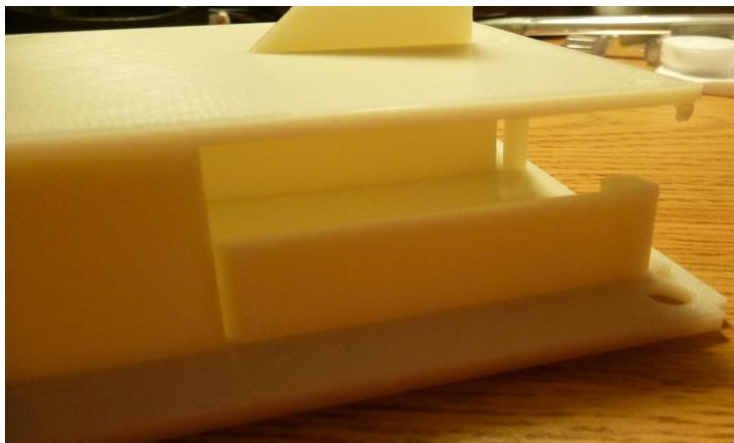


Figure 35: Half-Sized Doors and Broken Pillar

Additionally, a new pin system was developed using 2.5” x 8D nails from a local hardware store (for the doors), and a 0.125” aluminum rod for the rudder. The 8D nails fit snugly into the pin holes of the doors without any glue, while the rod is glued to the bottom of the rudder. The nails are mounted upside down, so that servo arms can be attached to their heads.

Originally, the servos were to be mounted upside down with gears attached to them, which would mesh with identical gears glued to the nails/rod, and rotate the doors and rudder, as outlined in Figure 36. However, the team learned in early D-Term that gears for the B1 spline size that these servos have are not manufactured. Therefore, an alternative servo mechanism was developed.

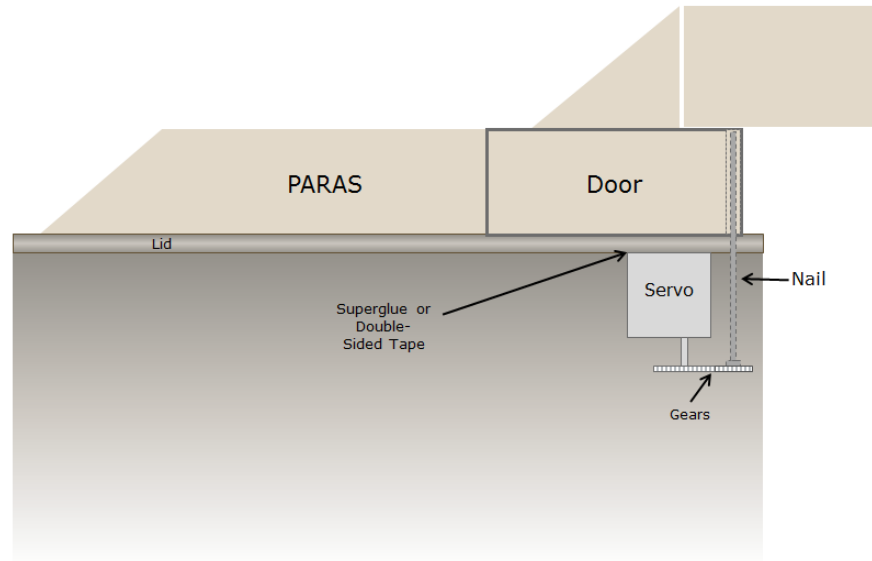


Figure 36: Original Servo Gear Mechanism

The second servo mechanism idea involved directly attaching the servos to the nails/rod using epoxy. The servos would be mounted upright and held in place by a lightweight support structure built from the same poster-board as the TRICON container mockups. However, this idea was scrapped after the team could not get the servos and support structure to fit within the container, and once the team realized that holes would need to be cut out of the rear of the TRICON to fit the protruding wires. One of these support structures is seen in Figure 37.

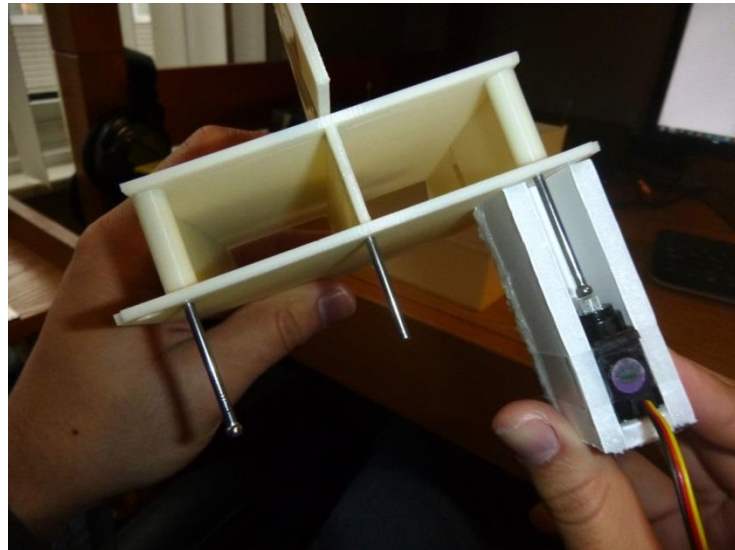


Figure 37: Directly Attached Servo Mechanism

In the third and final servo mechanism idea, the servos are again mounted upside down, superglued to the underside of the lid, this time with servo arms attached. Paper clips act as push rods connecting the servo arm on the servos to the servo arm glued to the nails and the servo arm

glued to the rod. When the servo turns its servo arm, it slides the paper clip up or down, which rotates the servo arm attached to the nail/rod, which turns the door or rudder. The servo arms are attached the nails/rod with JB Weld for a most permanent and strong bond. This final servo mechanism is shown in Figure 38.

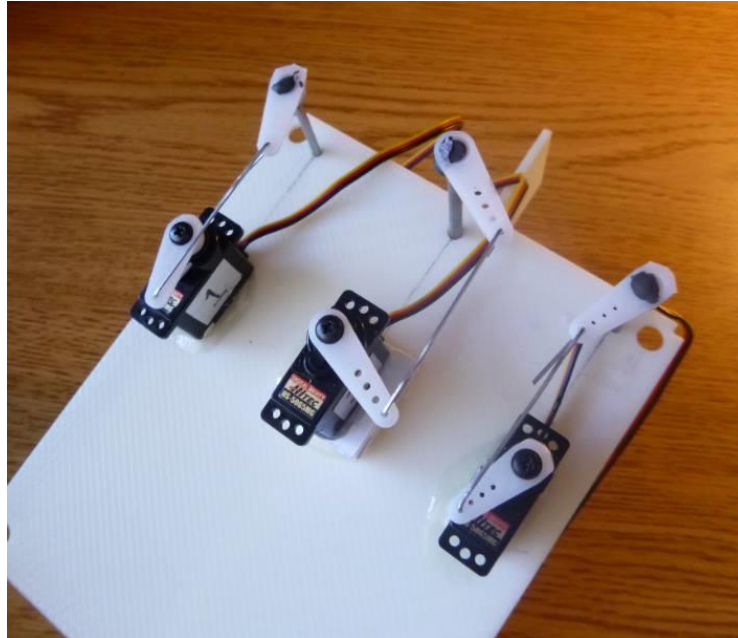


Figure 38: Servo Mechanism

After several failed attempts to attach the nail securely to the bottom edge of the rudder with superglue, Gorilla Glue, and general-purpose epoxy, the team cut a tab out in the rudder for the nail, and used an epoxy specifically for ABS plastic. This attachment method, seen in Figure 39, finally provided a strong bond between the nail and rudder.



Figure 39: Original Rudder with Nail

The final TRICON and SPARTA prototype is shown in Figure 40. Note that the sling legs from the previous poster-board designs have been reused for this design for consistency.

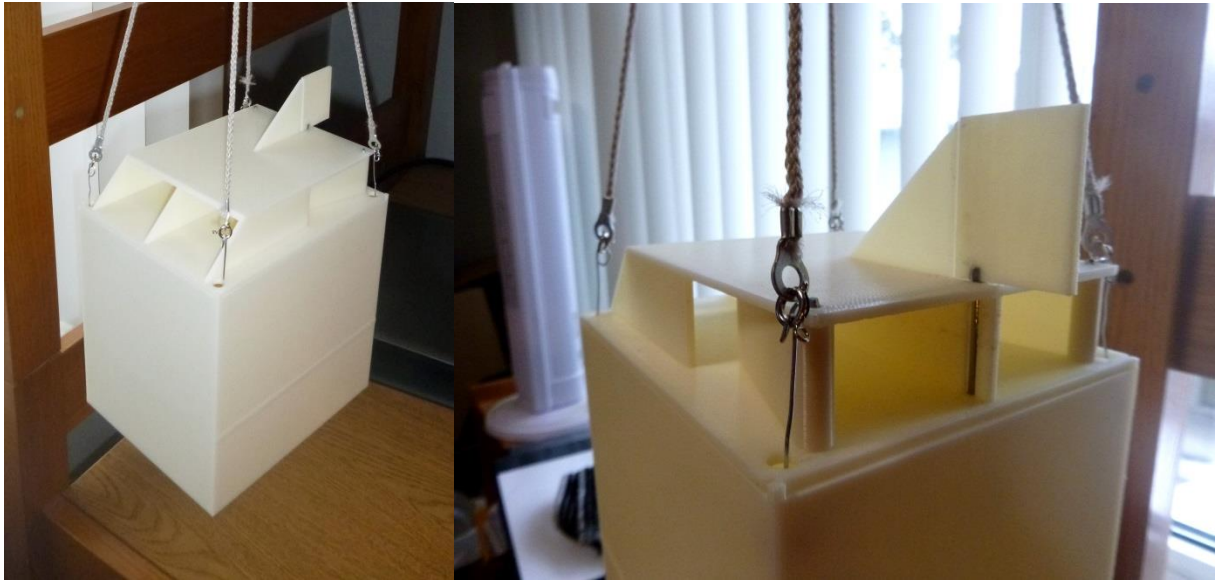


Figure 40: Final Rapid-Prototyped TRICON and SPARTA

3.8 Aerodynamic Analysis Verification

In order to test the accuracy of the aerodynamic analysis explained earlier, 1/10th scale versions of the pipes, vertical stabilizer and rudder were rapid prototyped, shown in Figures 41 and 42. However, the rudder model was manufactured with incorrect hole positions, and thus could not be used. The pipe model was mounted on a support structure attached to the force transducer, which measured the forces and moments generated by changing the angle of the rear door.



Figure 41: 1/10th Scale Pipes with Doors

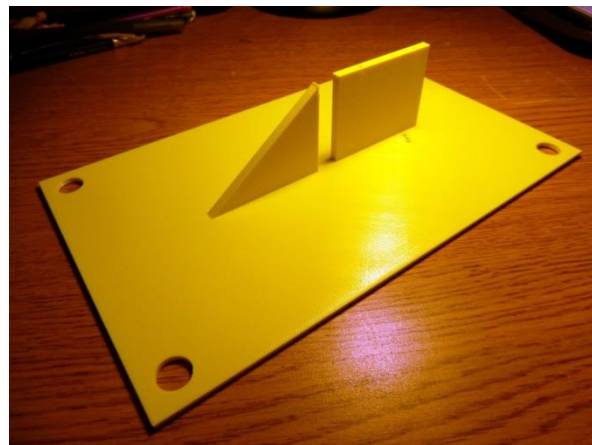


Figure 42: 1/10th Scale Vertical Stabilizer and Rudder

The support stand was machined in the WPI machine shop from aluminum purchased from MSC Industrial Supply Company. The stand screws directly into the force transducer and attaches to the 3D printed pipes using duct tape to reduce weight. Due to the difficulty of welding aluminum, particularly of the thicknesses of the support structure, JB Weld was again used for its strength and reliability.

For the wind tunnel tests, only the rear door was tested at various angles due to the front door having been scrapped by this point in the design. The door was held in place using the same 8D nail used in the 1/17th scale prototype, acting as a pin. The team encountered several issues during testing, including difficulty ensuring the force transducer was exactly aligned with the freestream flow, and difficulty in firmly affixing the support structure to the force transducer. The latter issue was determined to be caused by the upper plate of the support structure being slightly thinner than designed (about 1/10" thick versus the designed 1/8"). This caused the support structure and pipes to wobble slightly as the team adjusted the door angle during each test, which affected the reliability of the data.

An initial reading with the door closed was taken as a baseline offset. This offset was subtracted from the forces and moments measured at each door angles to produce the *change* in forces and moments versus door angle, yielding only the forces and moments caused directly by the door.

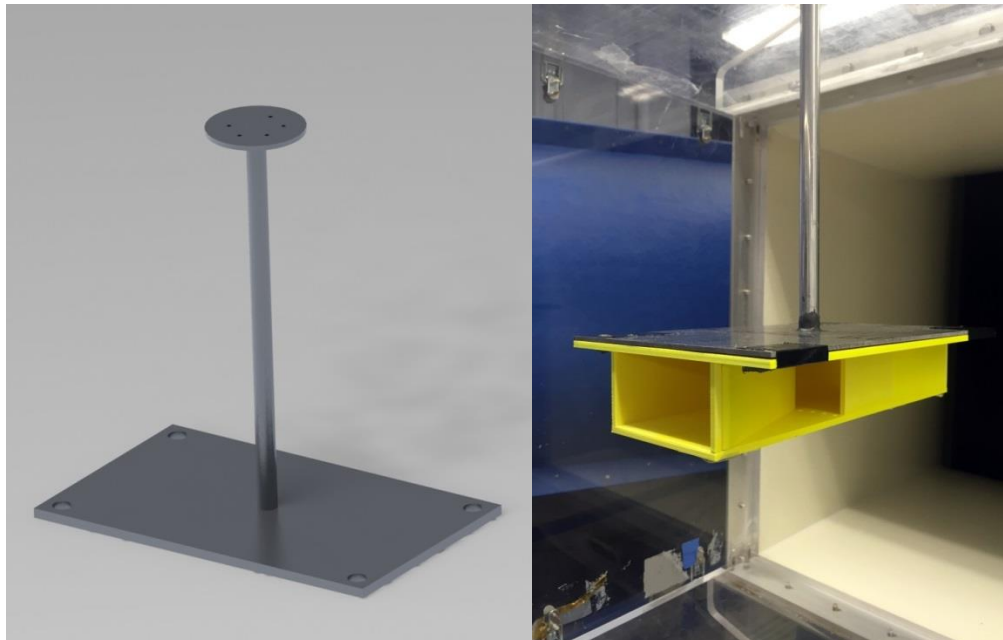


Figure 43: CAD model of support stand and with pipes in the wind tunnel

3.9 Final 1/17th Scale Prototype Testing

Once the pushrod style servo mechanism was crafted, the team connected the servos to the Arduino and began to troubleshoot the hardware. During this troubleshooting, the doors occasionally made contact with the inside center wall of the pipes as the orientation of the servos was worked out. The power of the servos became a liability as they were found to shear the top layer of ABS plastic off the container (where they were glued to) when the doors contacted the center wall of the pipes and prevented the servo arm from turning further, resulting in the team having to re-glue one of the door servos. Additionally, the other door servo failed midway through troubleshooting and no longer operated. The rudder servo did not encounter these issues and performed well; however due to the noise in the IMU data the team found the rudder to rapidly twitch in place during testing. This twitching was deemed minor as the rudder performed as expected.

The team discovered that the arm connected to the rod (and thus the rudder) protruded out too far when the rudder turned to the left, and would thus contact the inside rear wall of the container. Plans were made to cut a small hole in the rear of the container to allow the servo arm to extend further and allow the rudder to turn the full amount. However, following the loss of one of the door servos and the limited time remaining in the project to order a new one, the team cancelled the final testing of the prototype in the wind tunnel.

CHAPTER 4: RESULTS

4.1 Initial 1/10th Scale Pipes Testing

Three experimental values were recorded during the testing of the 1/10th scale pipes and compared with their theoretical values: the force due to the doors in the x-direction (in the direction of the freestream flow), the restoring force in the y-direction that corrects for sway, and the moment in the z-direction generated by the opening of the door. The rear door was pinned in five-degree increments from 0° to 45°. The results for the x-direction force are shown in Figure 44.

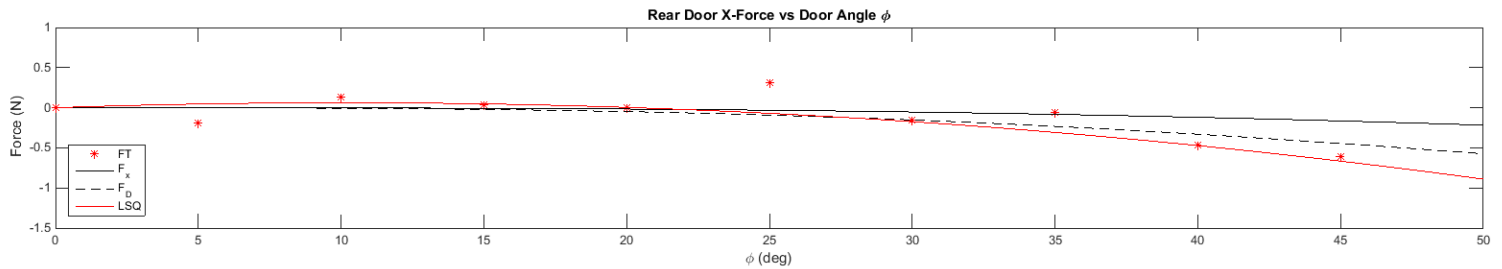


Figure 44: Rear Door X-Direction Force vs. Door Angle

The solid black line in the theoretical values for the force, with the dashed black line being the same equation with drag calculations also included. The red points are the actual data points taken by the force transducer, and the red line is a least-squares fit of those points, to compensate for the significant noise in the measurements. Each of the least-squares fit curves for the three graphs had their y-axis intercept set to 0 Newtons. The least-squares fit of the x-direction force was found to match the theoretical x-direction force with drag included (the dashed black line) fairly closely, albeit diverging with a slightly greater magnitude for door angles past 35°. Since the doors do not reach these door angles in the final control laws, this was ignored.

Some of the data points in this graph as well as the following two showed a positive force when the forces should have been zero or negative due to the orientation of the force transducer. Removing these positive points drastically altered the least-squares fit curve and in some cases created an impossible positive slope. Thus, the positive readings were left in and attributed to the overall noise in the measurements. However, one outlier was removed from each graph; in the x-direction force graph, the data point at 25° was removed due to having a relatively extremely positive value that contrasted significantly with the other measured values. The final least-squares fit curve for the x-direction force is given below.

$$F_{x_{exp}} = -0.0006\phi^2 + 0.0122\phi$$

ϕ is the door angle, measured in degrees. The R^2 correlation coefficient was 0.788, which the team was satisfied with given the noise of the measurements.

The restoring y-direction force graph is shown in Figure 45. The least-squares curve fit was found to match the theoretical results quite well, with again a slight divergence forming after the door angle exceeded 35°.

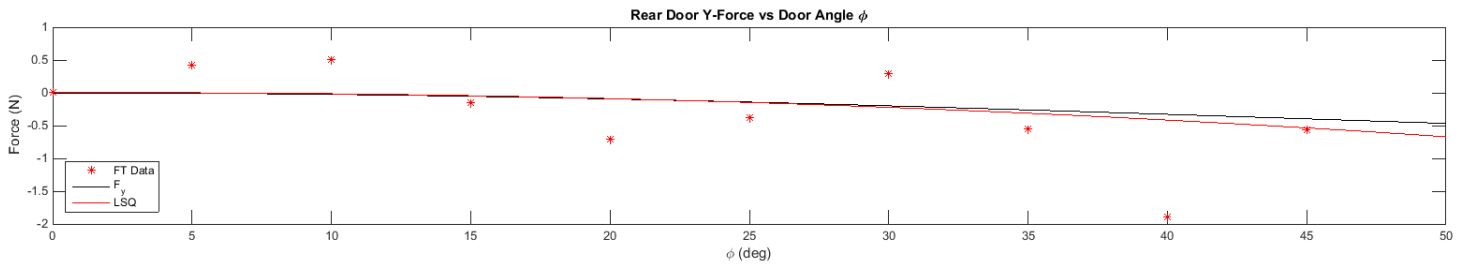


Figure 45: Restoring Y-Direction Force vs. Door Angle

An outlier at 40° was removed before the least-squares fit was taken. The least-squares fit equation for the y-direction is shown below.

$$F_{y_{exp}} = -0.0003\phi^2 + 0.0016\phi$$

Despite the least-squares fit matching the experimental values, the R^2 correlation coefficient was found to be much lower than the in the x-direction, with a value of 0.275. This was again determined to be due to the noise in the measurements.

The z-direction moment values were the most difficult to analyze, with the theoretical values hovering just below 0 Newton-meters. The resulting graph and least-squares fit equation is shown below.

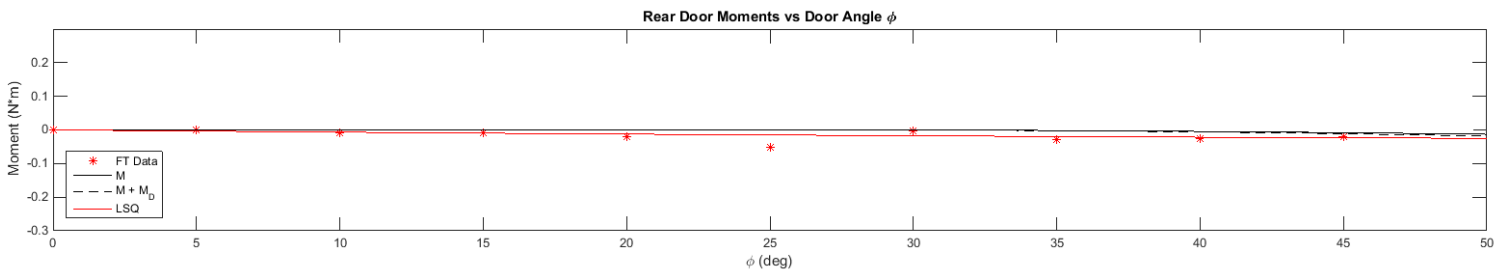


Figure 46: Rear Door Z-direction Moment vs. Door Angle

$$M_{z_{exp}} = 0.0004\phi^2 - 0.0007\phi$$

The R^2 correlation coefficient for the Z-direction moment was 0.61, an improvement upon the y-direction force R^2 value. Due to the noise in the measurements, accuracy concerns due to the team being unable to exactly align the force transducer, and issues with the support stand, all while measuring relatively tiny forces and moments, the team concluded that results were promising, and further testing would be necessary to fully verify the experimental calculations.

4.2 Initial 1/17th Scale Poster-Board TRICON Testing

The poster-board TRICON container was fitted with three iterations of a basswood and balsa SPARTA system: one with the doors closed (flush with the sides of the pipes), one with both doors open 10°, and one with the doors closed and a rudder with a double chord length. Each was tested in the wind tunnel at 12.7 m/s (scaled down from a 60 knot full scale speed) several times to determine if the SPARTA system exerted any innate passive stabilization upon the container.

The figure below is a screenshot taken from videos of the iterations in the wind tunnel. The average maximum yaw angle is displayed for the TRICON without the SPARTA system, one with the doors closed, and one with the doors open 10 degrees.

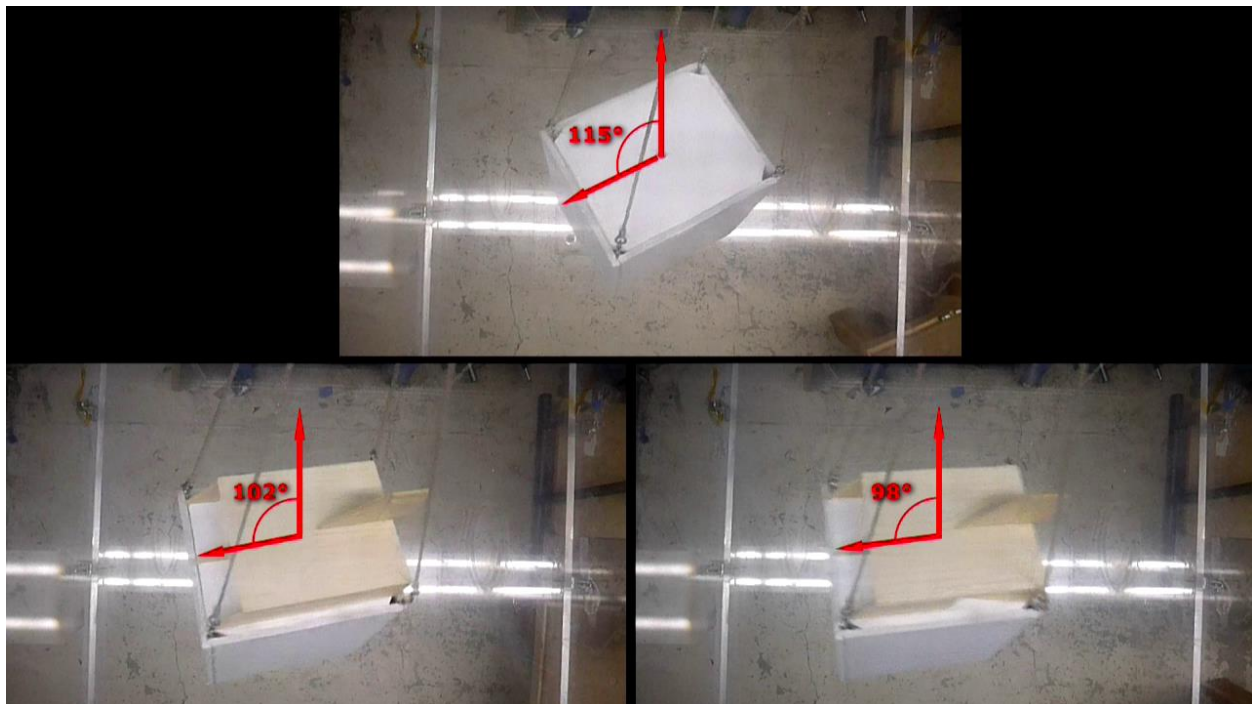


Figure 47: Average Maximum Yaw Angle for Various SPARTA Iterations

The team found that the SPARTA system did produce a small reduction in average maximum yaw amplitude, reducing it from 115° to 98° with the doors open 10°. However, the team was hoping for much more significant yaw magnitude direction, and a reduction in yaw rate, which remained relatively unchanged through each iteration. The double chord length rudder produced similar unpromising results. Additionally, the team found that when the SPARTA system occasionally stabilized the container and eliminated yaw (usually only for a second or two), the container would suddenly dramatically increase in sway motion. In one instance of prolonged negligible yaw, the container began to sway so violently it bumped the side of the wind tunnel. This behavior was consistent with the team's previous research; when one mode of instability is corrected, the other intensifies.

The team also tested the setup at various flow speeds; however, no reliable and significant passive stabilization was found. Thus, the team concluded that the SPARTA system was a purely active stabilization system, with little to no innate passive stabilization.

4.3 Final 1/17th Scale SPARTA Prototype Testing

The final 1/17th scale SPARTA prototype testing, controlled by the Arduino using measurements from the IMU gyroscope, and magnetometer, and actuated using the three servos, was cut short due to hardware issues. During the initial setup of the hardware, one of the door servos ceased to operate, and the other door servo ripped the top layer of ABS plastic off its attachment point twice. The rudder and doors were found to work properly, moving appropriately as the IMU was rotated, albeit with jitter due to the noisy IMU measurements. Due to the hardware issues and limited time remaining in the project, the team was unable to complete the final prototype testing of the SPARTA system. However, as the servos did all briefly function, and the rudder and doors behaved as expected with the IMU, the team viewed the initial setup testing as a proof of concept, with further testing required in a future project.

4.4 MATLAB Simulation Results

The numerical simulation was used to estimate how effective our prototype SPARTA system could be at stabilizing a TRICON or CONEX container in our wind tunnel. While the model is not a substitute for real world testing it is effective at estimating the performance. The final results of the testing were done to simulate the 17th scale testing in the wind tunnel with speeds of 12.5 m/s which is the scaled down airspeed. There is also an initial disturbance of a 15 degree initial yaw angle and 20 lateral swing angle.

Without the SPARTA system the simulation produces the following results.

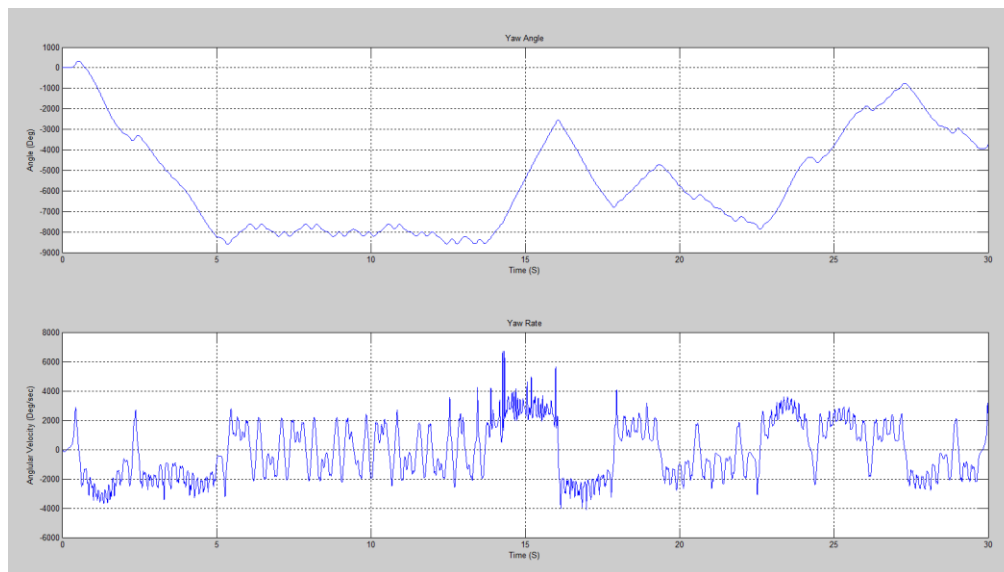


Figure 48: Yaw And Yaw Rate, Simulation Results Without SPARTA

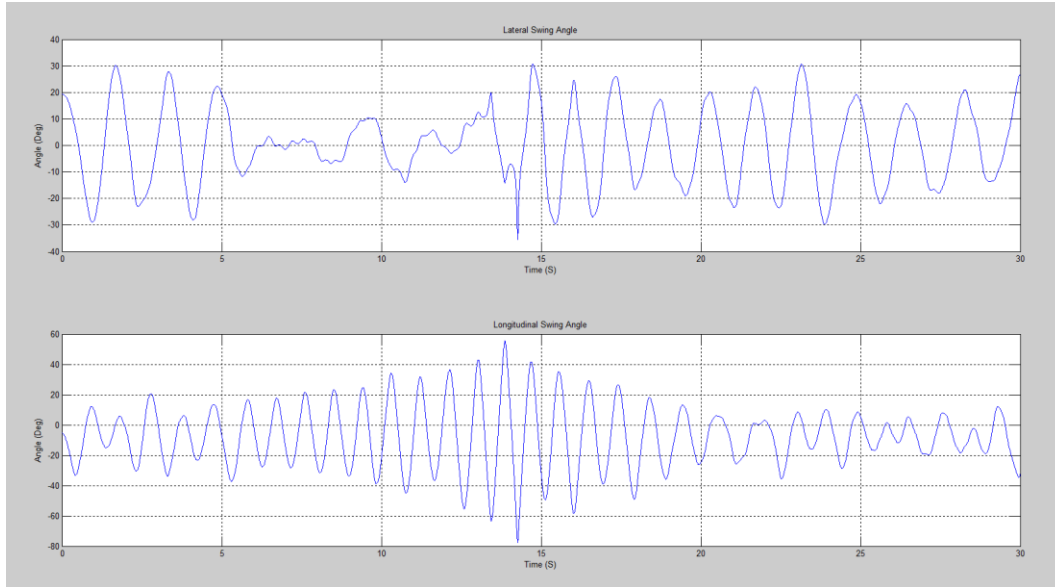


Figure 49: Lateral And Longitudinal Swing Angles, Simulation Results Without SPARTA

These results show what we are expecting, random yawing motion of spinning up and down, and lateral swing amplitude of approximately 30 degrees. Now with the SPARTA system the model produces the following results.

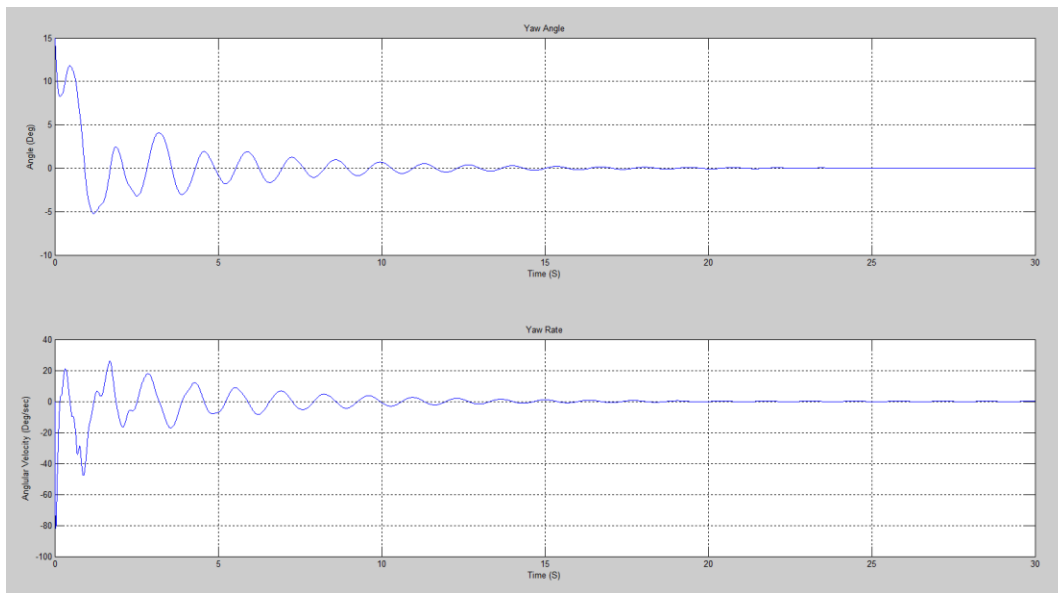


Figure 50: Yaw And Yaw Rate, Simulation Results With SPARTA

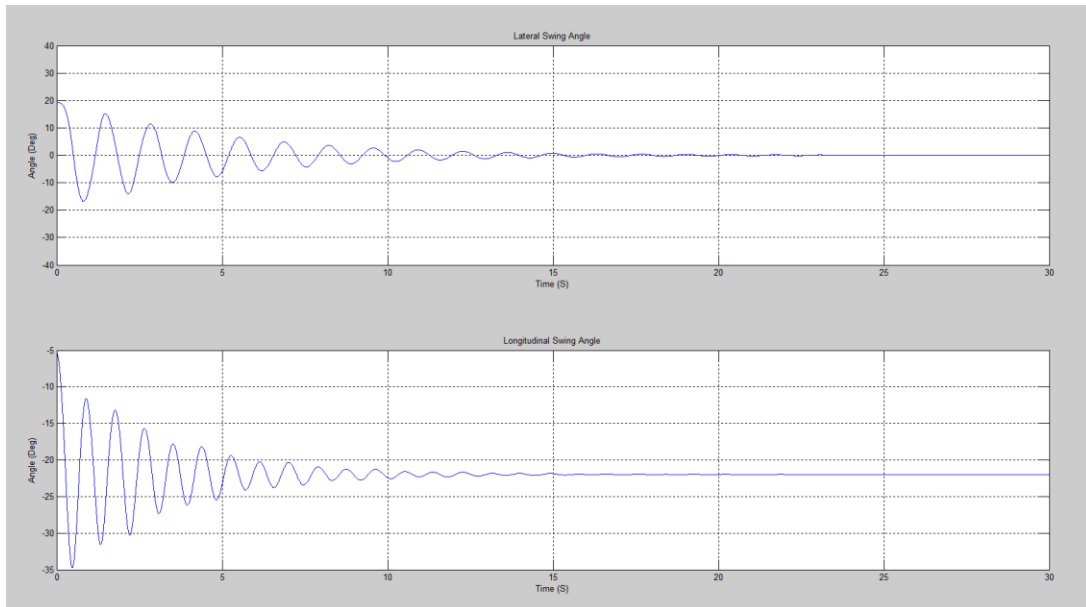


Figure 51: Lateral And Longitudinal Swing Angles, Simulation Results With SPARTA

These results show an almost perfect stabilization of the sling load over time. These results are promising however it is important to keep in mind that the simulation is in an ideal environment and with a relatively small initial disturbance. The model shows similar stabilization effects with initial yaw angle disturbances up to approximately 35 degrees. After that the system is not able to stabilize the container since it is less and less effective at imparting forces and moments on the container the further away from a zero degree yaw angle it is.

4.5 Kalman Filter Results

The simulation with the Kalman filters inserted was also tested both with the filters providing inputs to the controls and without. The testing shows that when the Kalman filters are not linked to the controls both filters appear to be working. This is shown in the plots below which are the differences between the actual and Kalman estimated states as a function of time.

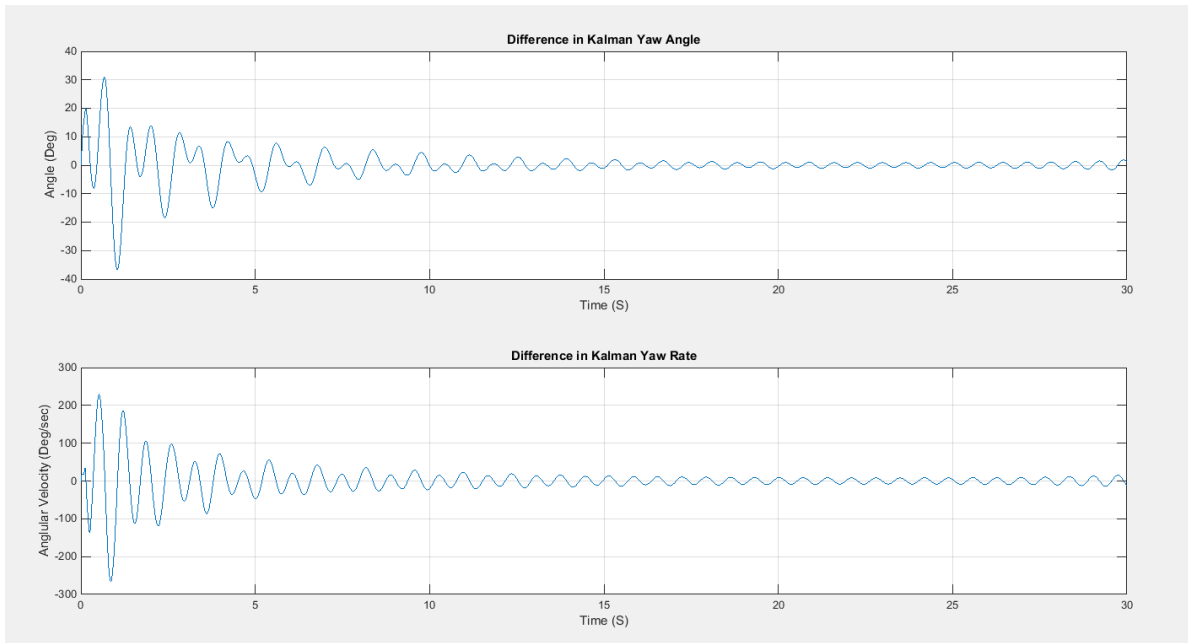


Figure 52: Yaw And Yaw Rate Kalman Filter Results Without Linked Control

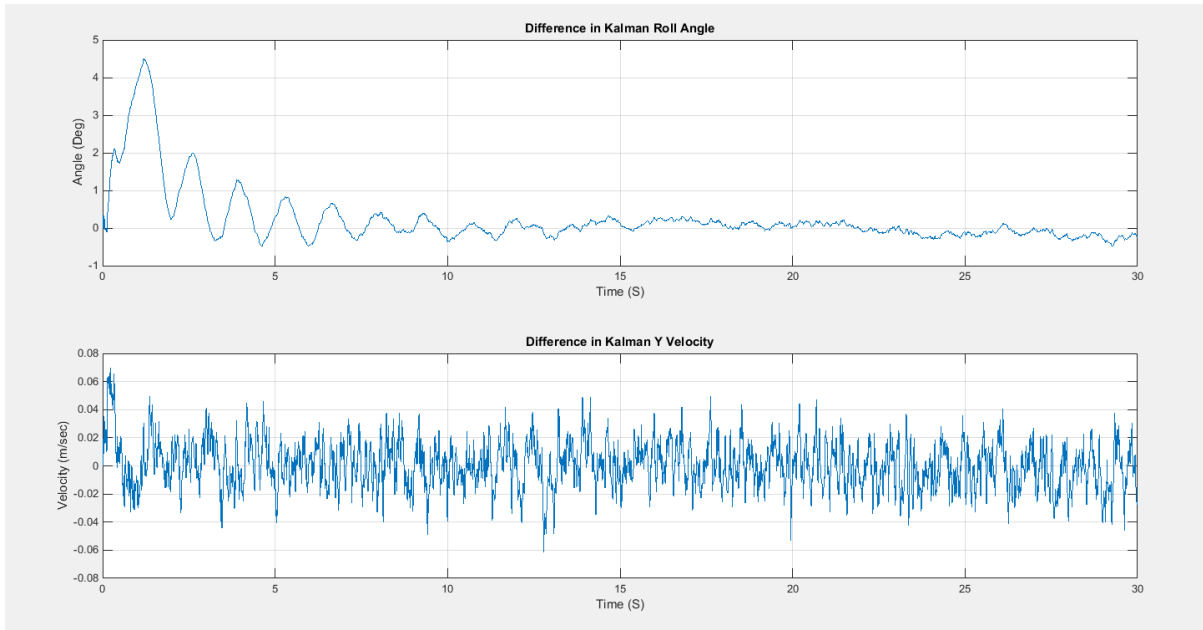


Figure 53: Roll And Sway Velocity Kalman Filter Results Without Linked Control

Both filters do effectively converge to zero when the controls are not linked to the Kalman estimates however when they are linked neither filter effectively estimates the actual value of the measured states. The results from the linked Kalman filter simulation are shown below.

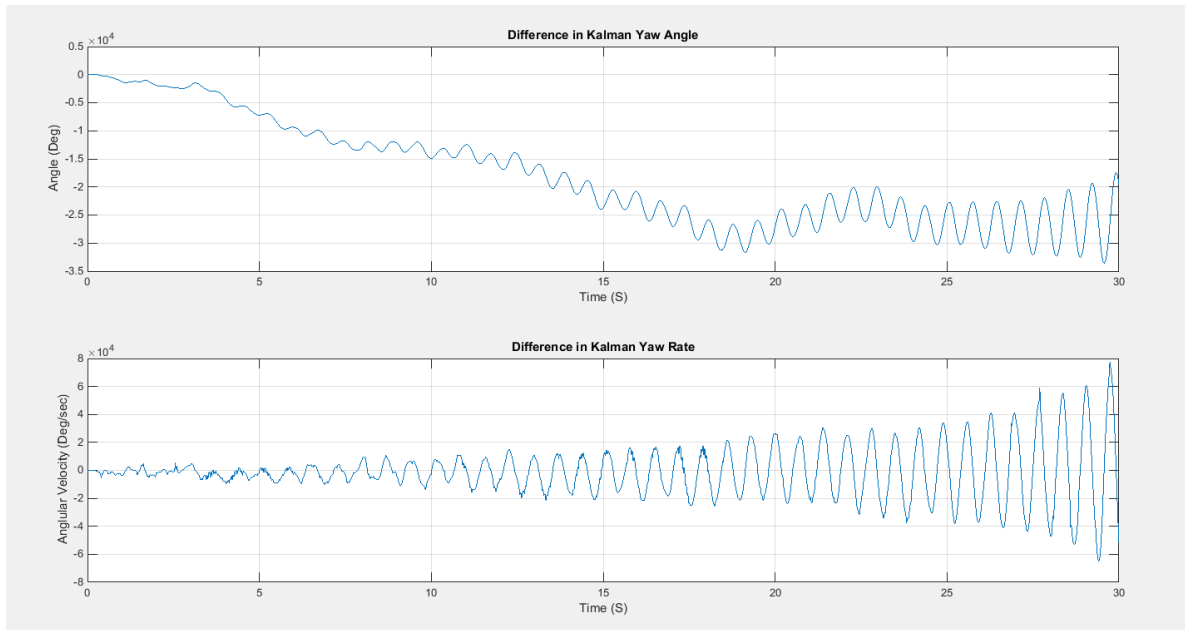


Figure 54: Kalman Yaw And Yaw Rate With Linked Control

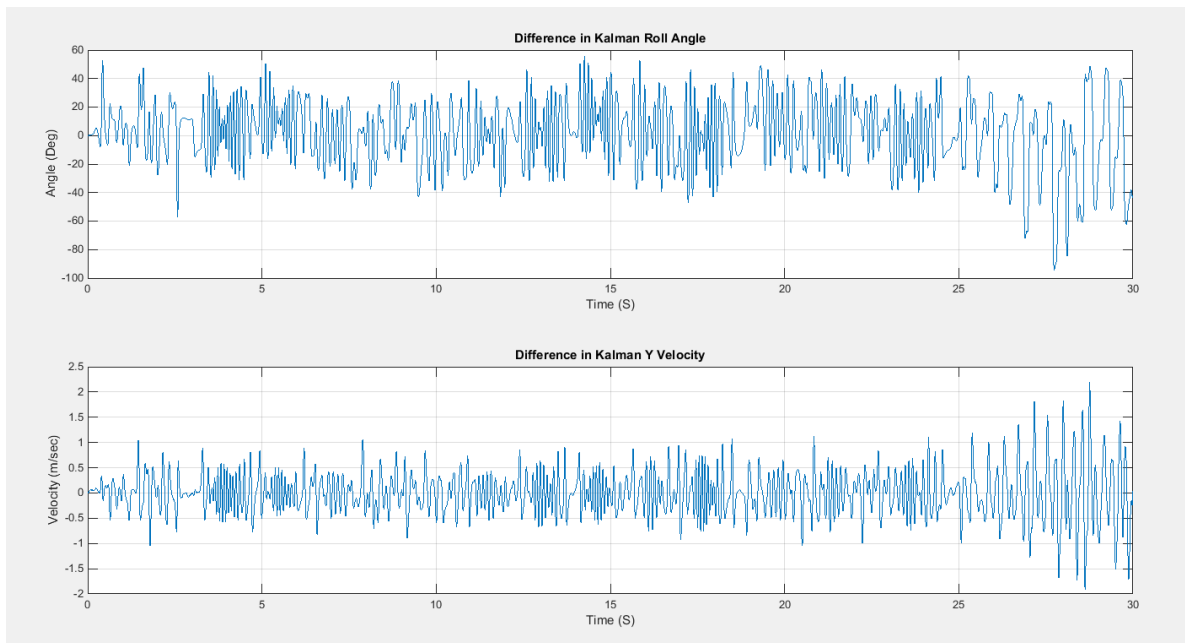


Figure 55: Kalman Roll And Sway Velocity With Linked Control

This shows that the Kalman filters do need further development before real world testing should be tried.

4.6 IMU-SD Card Data Logging Results

The IMU-SD Card data-logging function was tested and used during a wind tunnel testing session. This was done to test the effectiveness of the code and the setup. The results

properly convey the movement of the container during the testing and also showcase the noise that the IMU records as it is taking measurements. Only the accelerometer (3-axes) and the gyroscope (3-axes) were used in this testing.

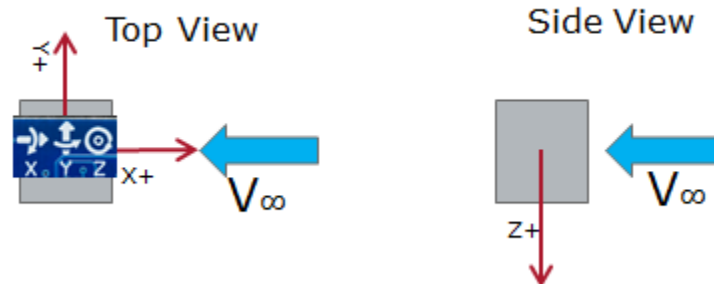
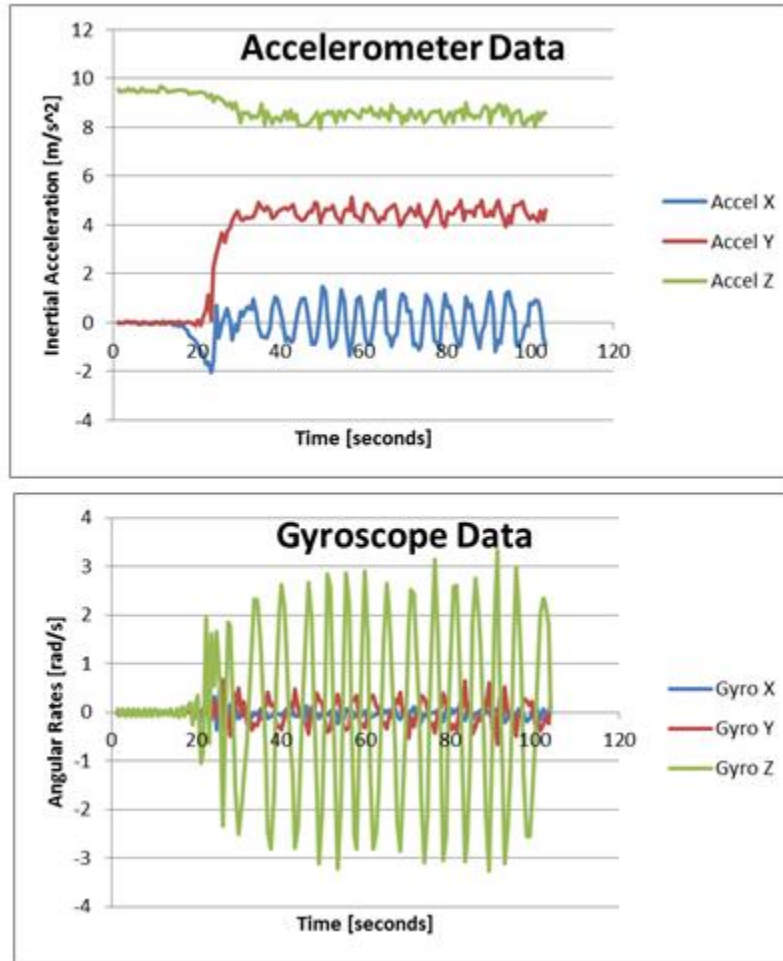


Figure 56: Accelerometer and Gyroscope Data-Logging Results

CHAPTER 5: CONCLUSIONS

In this project, the SPARTA system for active stabilization of helicopter sling loads was developed. Using IMU data, two pipes with doors were designed to correct sway, while a rudder and vertical stabilizer correct yaw. This design was tested in MATLAB via a numerical simulation, based on a mathematical model of a sling load system that was also developed. A 1/10th scale version of one side of the pipes was rapid-prototyped and tested in a wind tunnel using a force transducer to verify that the forces and moments generated by the door opening matched the theoretical calculations. Additionally, a 1/17th version of the TRICON container and a non-moving SPARTA system were constructed from poster-board and basswood/balsa to test for any passive stabilization.

A 1/17th scale prototype of the SPARTA system (with TRICON container) was created for testing. An actuation and control scheme for the 1/17th scale rapid-prototyped SPARTA system was developed using an Arduino, IMU, and three servomotors. The Arduino was programmed to use the control laws developed for the numerical simulation as well as the IMU data to actuate the SPARTA system and reduce instabilities. Wind tunnel testing of the overall system was not performed due to time constraints..

The SPARTA system was shown to be highly effective at stabilizing the sling load in the numerical simulation. This observation does however depend on the accuracy of the mathematical model used for numerical simulation of the system. The hardware and actuation scheme worked well for the final prototype, with the Arduino DUE able to perform using both the IMU data-logging script and the control law script. The poster-board and basswood/balsa SPARTA tests revealed no passive stabilization, and were a good check of the baseline behavior predicted by the numerical simulation. However, the 1/17th scale of the prototype, constrained by the size of the wind tunnel, was found to be too small to properly test the SPARTA system. Even with the smallest servos available, the system added significant weight to the container, and the limited volume within the container presented issues when creating a mechanism to turn the doors and rudder via the servomotors. Additionally, the power of the servos proved to be a liability, shearing off the top layer of ABS plastic a door contacted the inside wall of the pipes, and damaging another servo in the same manner.

Future work in the mathematical model, numerical simulation, hardware setup, and practical testing is recommended. The theoretical calculations behind the mathematical model and SPARTA system were based on several simplifying assumptions that could be eliminated through the use of computational fluid dynamics (CFD). Further experimental validation of the mathematical model needs to be performed, as the model only takes into account steady level flight and does not model changes in altitude or changes in speed. The SPARTA system would also need a different IMU setup to compare the orientation and motion of the sling load to the orientation and motion to the helicopter. An onboard power supply would also need to be

developed since the current SPARTA prototype draws power externally and it is not possible for the helicopter to supply power. Larger scale testing would allow for a better hardware and actuation scheme using better components without such stringent size and weight requirements. With a larger scale prototype and more coding knowledge, a more powerful microcontroller such as a Raspberry Pi could have increased performance and allowed for a more complex control code, including the use of a Kalman filter. A more powerful microcontroller with larger built-in memory would allow the SD card breakout to be used without compromising a system that requires such high dynamic response, and a more precise and accurate IMU could be used to obtain better readings, thus making the control system function more effectively.

Appendix A: Derivation of Moments of Angular Momentum

A_i	Intake area of door
A_e	Exit area of door
C_D	Drag coefficient of door (flat plate)
cm	Center of mass
D	Drag force
E_A, E_B	Exit area of pipes
F_x, F_y, F_z	Forces in Cartesian coordinates
I_A, I_B	Inlet area of pipes
M_{1A}, M_{3A}	Moment of doors $1_A, 3_A$
m	Mass of TRICON container
\dot{m}_i	Mass flow rate of intake area A_i
\dot{m}_e	Mass flow rate of exit area A_e
ρ	Density of air
r	Moment arm (perpendicular to center of mass)
ϕ	Door angle with respect to x axis
V_i	Velocity of air at door intake
V_e	Velocity of air at door exit
V_∞	Freestream flow velocity

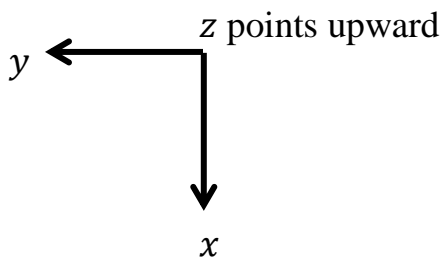
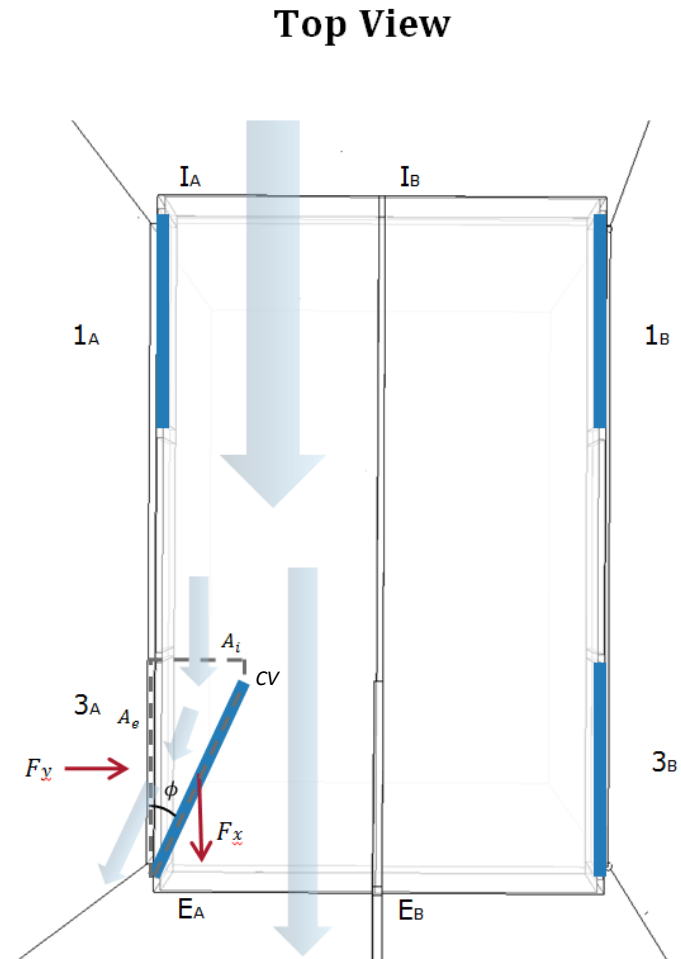


Figure 57: Pipes Diagram



Force is equal to the change in momentum over time

$$F = \frac{dp}{dt} = \frac{d(mV)}{dt} \quad (1)$$

For a fluid, Equation (1) is written using mass flow rate \dot{m} , the change in mass over time

$$F = d(\dot{m}V) \quad (2)$$

Where

$$\dot{m}_1 = \rho_1 V_1 A_1 \quad (3)$$

For intake area A_i . With an intake area A_i and exit area A_e , Equation (2) becomes

$$F = \dot{m}_e V_e - \dot{m}_i V_i$$

As shown in Figure 57, for the specified control volume, there is no mass flow intake in the y -component (neglect small area above door). Effective velocity in the x -component will change (increase) as door angle increases ($V_i \cos(\phi)$ is the velocity in the x -component due to the door opening)

$$F_x = \dot{m}_i (V_i - V_i \cos(\phi)) \quad (4)$$

$$F_y = \dot{m}_e V_e \quad (5)$$

Conservation of mass dictates that the mass flow rate of the intake must equal the mass flow rate of the exit, therefore

$$\dot{m}_i = \dot{m}_e$$

Expanding using the definition of mass flow rate shown in Equation (3)

$$\rho_i V_i A_i = \rho_e V_e A_e$$

Intake and exit densities are equal due to no compressibility effects for air at low speeds

$$\rho V_i A_i = \rho V_e A_e$$

Assuming that intake velocity V_i is equal to freestream velocity V_∞

$$\rho V_\infty A_i = \rho V_e A_e$$

Relating A_i to A_e using basic geometry (neglect small horizontal area above door)

$$\rho V_\infty (\sin(\phi) A_e) = \rho V_e A_e$$

Cancelling out like terms

$$V_{\infty} \sin(\phi) = V_e \quad (6)$$

Using Equation (3) for \dot{m}_e

$$\dot{m}_e = \rho V_e A_e$$

Plugging in Equation (6)

$$\dot{m}_e = \rho (V_{\infty} \sin(\phi)) A_e \quad (7)$$

Plugging Equation (6) and Equation (7) into Equation (5)

$$F_y = (\rho (V_{\infty} \sin(\phi)) A_e) (V_{\infty} \sin(\phi))$$

Combining like terms

$$F_y = \rho V_{\infty}^2 \sin^2(\phi) A_e \quad (8)$$

Equation (8) quantifies the restoring force F_y that is used for sway motion correction. For extra force F_x , Equation (4) states

$$F_x = \dot{m}_i (V_i - V_i \cos(\phi))$$

Expanding using Equation (3) for \dot{m}_i

$$F_x = \rho V_i A_i (V_i - V_i \cos(\phi))$$

Combining like terms and substituting $V_i = V_{\infty}$

$$F_x = \rho V_{\infty}^2 A_i (1 - \cos(\phi)) \quad (9)$$

Equation (9) quantifies the extra force F_x that results from turning the flow. This simplified equation holds reasonably well for small door angles ($\phi \leq 20^\circ$). For larger angles, drag due to the door (which is approximated as a flat plate) must be taken into account.

Drag D for a door is equal to

$$D = \frac{1}{2} \rho V_{\infty}^2 C_D A_i \quad (10)$$

Where, for $0^\circ \leq \phi \leq 90^\circ$ ⁸

$$C_D = 2 \sin(\phi)$$

⁸ <http://mekside.com/wings-redux/>

Therefore, Equation (10) becomes

$$D = \rho V_{\infty}^2 \sin(\phi) A_i$$

Adding this to Equation (9)

$$F_{x+drag} = \rho V_{\infty}^2 A_i (1 - \cos(\phi)) - D \tag{11}$$

Or

$$F_{x+drag} = \rho V_{\infty}^2 A_i (1 - \sin(\phi) - \cos(\phi))$$

For the process of simplification and clarity, Equation (9) will be used for F_x for deriving the moment equations.

Angular momentum L was found in a different way than was previously taught, using forces F_x, F_y and the Pythagorean Theorem

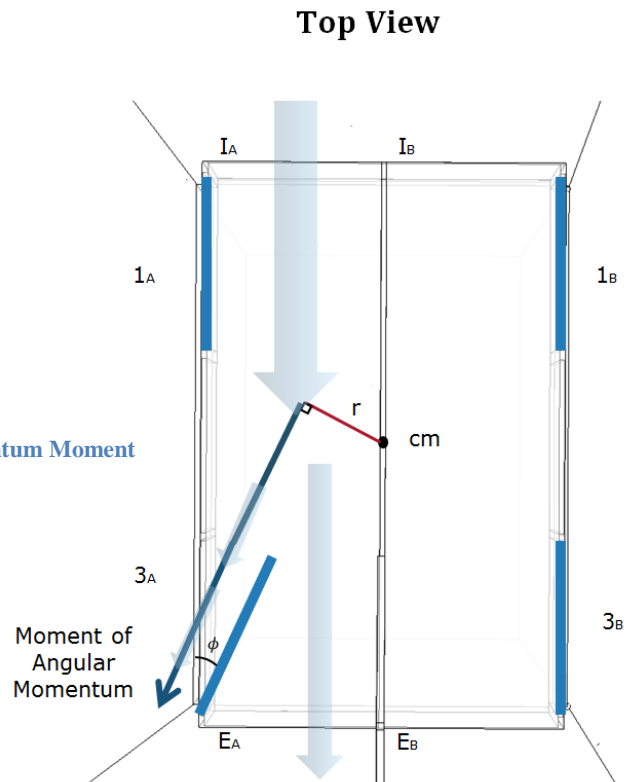
$$L = \sqrt{F_x^2 + F_y^2} \tag{12}$$

The cross product of Equation (12) and a moment arm r was used to generate a moment of angular momentum M

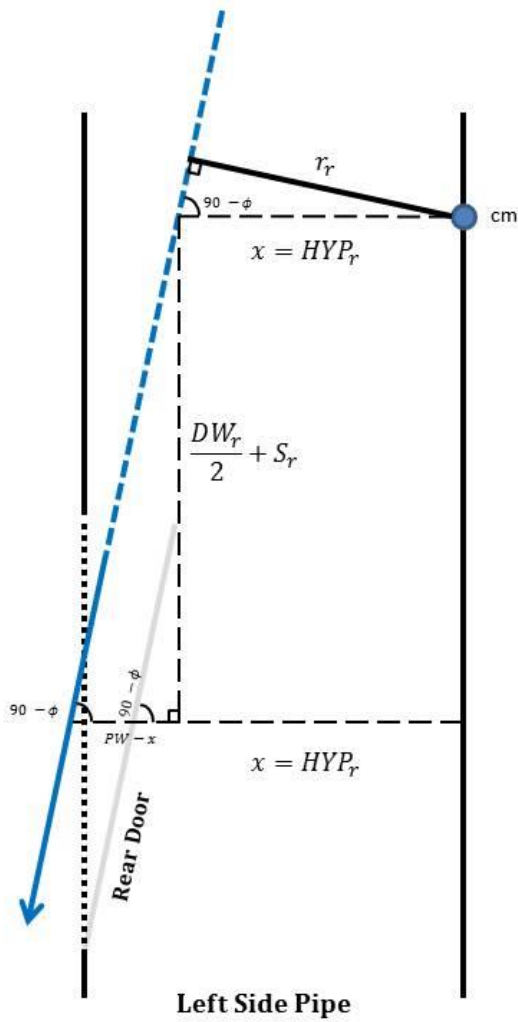
$$M = r \times L$$

Where r is the perpendicular distance between the center of mass of the TRICON and the angular momentum vector, shown in Figure 58.

Figure 58: Angular Momentum Moment



The moment arm equations are described and derived in Figures 59 and 60 below.



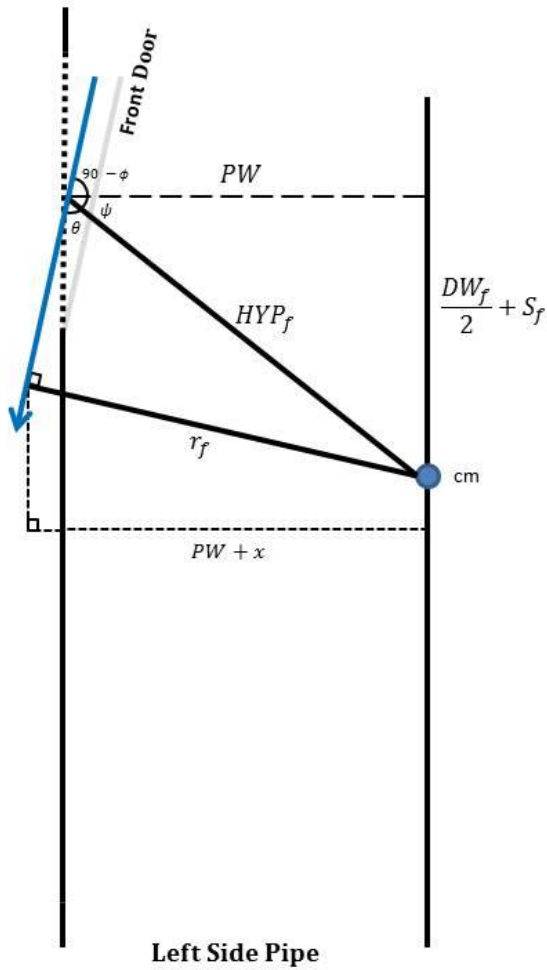
- ϕ door angle
- DW_r rear door width
- S_r longitudinal distance between center of rear door opening and center of mass
- PW pipe width
- BTW_r "Big Triangle Width" of rear door
- HYP_r hypotenuse of moment arm triangle
- r_r moment arm

$$\tan(90 - \phi) = \frac{\frac{DW_r}{2} + S_r}{PW - x} \rightarrow PW - x = \frac{\frac{DW_r}{2} + S_r}{\tan(90 - \phi)} \rightarrow BTW_r$$

$$BTW_r = PW - x \rightarrow x = PW - BTW_r = HYP_r$$

$$\sin(90 - \phi) = \frac{r_r}{HYP_r} \rightarrow r_r = HYP_r * \sin(90 - \phi)$$

Figure 59: Rear Door Moment Arm



- φ door angle
- DW_f front door width
- S_f longitudinal distance between center of front door opening and center of mass
- PW pipe width
- θ angle between angular momentum vector and hypotenuse of moment arm triangle
- ψ angle between the horizontal and hypotenuse of moment arm triangle
- HYP_f hypotenuse of moment arm triangle
- r_f moment arm

$$HYP_f = \sqrt{PW^2 + \left(\frac{DW_f}{2} + S_f\right)^2}$$

$$\psi = \cos^{-1}\left(\frac{PW}{HYP_f}\right)$$

$$\theta = 180 - (90 - \varphi) - \psi$$

$$r_f = HYP_f * \sin(\theta)$$

Figure 60: Front Door Moment Arm

Appendix B: Aerodynamic Equations from *Modelling of static aerodynamics of helicopter underslung loads*

Nomenclature:

A_y	Side face area
C_Y^*	Body-axis sideforce component, based on side face area
u, v, w	Cartesian velocity components
V	Freestream velocity
ρ	Density of air (1.225 kg/m ³)
α	Angle of attack, = $Tan^{-1}\left(\frac{w}{u}\right)$
θ^*	Side face crossflow angle, = α
C_{YM}^*	Body-axis pitching moment component, based on side face area and length
β	Sideslip angle, $Sin^{-1}v/V$
φ^*	Front face crossflow angle, = $Tan^{-1}w/v$
θ^*	Side face crossflow angle, = $Tan^{-1}wu \equiv \alpha$

Equations:

$$C_Y^* = \frac{Y}{\frac{1}{2}\rho V^2 A_y} = C_{Y,att}^* + \Delta C_Y^* = -1.4 \left(\frac{v}{V}\right) + (C_{Y,bubble}^* + C_{Y,base\ dip}^*)$$

$$C_{Y,base\ dip}^* = \begin{cases} 0; & \frac{v}{V} < 0.82 \\ 1.94 \left(\frac{v}{V} - 0.82\right); & \frac{v}{V} \geq 0.82 \end{cases}$$

$$C_{Y,bubble,0}^* = \begin{cases} 2.5 \frac{v}{V}; & \frac{v}{V} \leq 0.18 \\ 0.45 - 0.703 \left(\frac{v}{V} - 0.18\right); & 0.18 < \frac{v}{V} < 0.82 \\ 0; & \frac{v}{V} \geq 0.82 \end{cases}$$

$$C_{Y,bubble}^* = C_{Y,bubble,0}^* \times |\cos^3(2\theta^*)|$$

$$C_Y^* = \frac{YM}{\frac{1}{2}\rho V^2 A_y L} = C_{YM,att}^* + \Delta C_{YM}^*$$

$$\beta_{side} = \sin^{-1}\left(\frac{v}{V}\right), \quad \beta_{front} = \sin^{-1}\left(\frac{u}{V}\right)$$

$$C_{YM,att}^* = (-0.075 * \sin(2\beta_{side}) + 0.03 \left(\frac{8}{6}\right) \sin(4\beta_{side}) \cos(\theta^*) \\ + (0.075 \left(\frac{8}{6}\right)^2 \sin(2\beta_{front}) \cos(\varphi^*))$$

$$C_{YM,bubble,side}^* = \begin{cases} 0.25 \frac{v}{V}; & \frac{v}{V} \leq 0.28 \\ 0.07 - 0.219 \left(\frac{v}{V} - 0.28\right); & 0.28 < \frac{v}{V} < 0.6 \\ 0; & \frac{v}{V} \geq 0.6 \end{cases}$$

$$C_{YM,bubble,front}^* = \begin{cases} -0.9 \frac{u}{V}; & \frac{u}{V} \leq 0.2 \\ -0.18 + 0.219 \left(\frac{u}{V} - 0.2\right); & 0.2 < \frac{u}{V} < 0.7 \\ 0; & \frac{u}{V} \geq 0.7 \end{cases}$$

$$\Delta C_{YM}^* = C_{YM,bubble,side}^* \times \cos^2(\theta^*) + C_{YM,bubble,front}^* \times \cos^2(\varphi^*)$$

Appendix C: Full MATLAB Numerical Simulation Code and Results

Simulation Code

```
clear variables; close all; clc;
```

Model parameters

```
params_model.scale = 17;

%----- Height of sling load cm below hook
params_model.sling_height_cmass = -5 / params_model.scale;

%----- Sling-loaded cargo
params_model.load.mass = 2000 / (params_model.scale^3); % kg
params_model.load.width = 1.9812 / params_model.scale; % m
params_model.load.length = 2.4384 / params_model.scale; % m
params_model.load.height = 2.4384 / params_model.scale; % m

params_model.load.area_x = params_model.load.height*params_model.load.width;
params_model.load.area_y = params_model.load.length*params_model.load.height;
params_model.load.area_z = params_model.load.length*params_model.load.width;

params_model.load.w_by_l = params_model.load.width/params_model.load.length;

params_model.load.Ixx = (1/12)*params_model.load.mass*...
    (params_model.load.width^2 + params_model.load.height^2);
params_model.load.Iyy = (1/12)*params_model.load.mass*...
    (params_model.load.height^2 + params_model.load.length^2);
params_model.load.Izz = (1/12)*params_model.load.mass*...
    (params_model.load.length^2 + params_model.load.width^2);

%----- Pipes
params_model.pipes.height = 0.4 / params_model.scale; % m
params_model.pipes.width = 0.8 / params_model.scale; % m
params_model.pipes.length = params_model.load.length;
params_model.pipes.area = params_model.pipes.height*params_model.pipes.width;

params_model.pipes.door.length = 0.8 / params_model.scale; % m
params_model.pipes.door.area = params_model.pipes.door.length*params_model.pipes.height;
params_model.pipes.door.loc = (1.143/params_model.scale) - params_model.pipes.door.length;

%----- Rudder
params_model.rudder.height = 0.6 / params_model.scale; % m;
```

```

params_model.rudder.length = 1 / params_model.scale;
    % m;
params_model.rudder.area = params_model.rudder.height*params_model.rudder.length;
params_model.rudder.CLO = 2*pi;
params_model.rudder.CDO = 1.28;
params_model.rudder.loc = params_model.load.length / 2;

%----- Vertical stabilizer
params_model.vstab.height = params_model.rudder.height;
params_model.vstab.length = 0.5 / params_model.scale;
    % m;
params_model.vstab.area = 0.5*params_model.vstab.height*params_model.vstab.length;
params_model.vstab.loc = params_model.load.length / 2;

%----- Sling legs
params_model.legs.youngs = 0.64*10^9;
    % Young's modulus, N/m^2
params_model.legs.cs_area = pi*(0.06/(2*params_model.scale))^2; % m^2
params_model.legs.length0 = 5.3 / params_model.scale;
    % Unstretched length, m
params_model.legs.stiffness = params_model.legs.youngs*params_model.legs.cs_area /
params_model.legs.length0;

params_model.legs.C_wind_up = -0;
    % Coefficient of wind-up restoring torque, N.m/rad

params_model.legs.loc_e = 0.5*[...
    params_model.load.length, -params_model.load.width, -params_model.load.height;
    -params_model.load.length, -params_model.load.width, -params_model.load.height;
    -params_model.load.length, params_model.load.width, -params_model.load.height;
    params_model.load.length, params_model.load.width, -params_model.load.height]';

```

Simulation parameters

```

params_simulation.tf = 0.5*60; % simulation duration, s
params_simulation.use_rudder = true;
params_simulation.use_pipes = true;
params_simulation.use_control = true;
params_simulation.use_ekf = false;
params_simulation.use_legs = false;
params_simulation.use_kalman = true;
params_simulation.use_kalmancontrols = false;

params_simulation.solver_options= odeset('RelTol', 1e-4, 'AbsTol', 1e-4);

params_simulation.V_freestream = 12.5; % freestream velocity, m/s
params_simulation.rho_atm = 1.225; % atmospheric density, kg/m^3
params_simulation.g = 9.81; % m/s

params_simulation.deg2rad = pi/180;

params_simulation.initial_state_system =

```

```
[35*params_simulation.deg2rad;0;20*params_simulation.deg2rad;0;0;0;15*params_simulation.deg2rad;0;
;20*params_simulation.deg2rad;0];
% params_simulation.initial_state_system(2) = params_model.load.width/10;
% params_simulation.initial_state_system(3) = params_model.sling_height_cmass;
% params_simulation.initial_state_system(5) = -0.1 / params_model.scale;

params_simulation.initial_state      = params_simulation.initial_state_system;

params_model.legs.tension0          = 0.25*params_model.load.mass*params_simulation.g*...
norm([(params_model.sling_height_cmass - params_model.load.height/2); ...
params_model.load.length/2; params_model.load.width/2 ]) / ...
(params_model.sling_height_cmass - params_model.load.height/2);
```

Control parameters and LQR gain

```
params_control.psi0                 = 0;
params_control.r0                   = 0;
params_control.yd0                  = 0;
params_control.rudder0              = 0;
params_control.door_left0           = 15*params_simulation.deg2rad;
params_control.door_right0          = 15*params_simulation.deg2rad;
params_control.theta0               = -22*params_simulation.deg2rad;
params_control.phi0                 = 0;
params_control.v0                   = params_simulation.v_freestream;
params_control.lqr                  = calc_lqr_gain(params_model, params_control, params_simulation);

params_control.sat.rudder_min        = -45*params_simulation.deg2rad;
params_control.sat.rudder_max        = 45*params_simulation.deg2rad;
params_control.sat.door_left_min     = 0;
params_control.sat.door_left_max     = 30*params_simulation.deg2rad;
params_control.sat.door_right_min    = 0;
params_control.sat.door_right_max    = 30*params_simulation.deg2rad;
```

Kalman Parameters

```
params_kalman.error = randn(1,params_simulation.tf*100);
params_kalman.std_accel = 1;
params_kalman.std_mag = 1*params_simulation.deg2rad;
params_kalman.std_gyro = 1*params_simulation.deg2rad;
params_kalman.L = (-1*params_model.sling_height_cmass);

params_kalman.phi_v.A = kalmanFilter_v_phi_cc( 'A', params_kalman);
params_kalman.phi_v.B = kalmanFilter_v_phi_cc( 'B', params_kalman);
params_kalman.phi_v.H = kalmanFilter_v_phi_cc( 'H', params_kalman);
params_kalman.phi_v.Q = kalmanFilter_v_phi_cc( 'Q', params_kalman);
params_kalman.phi_v.R = kalmanFilter_v_phi_cc( 'R', params_kalman);
params_kalman.phi_v.F = [0,0;0,1];
% params_kalman.phi_v.K = kalmanFilter_v_phi_cc( 'K', params_kalman);
```

```

params_kalman.psi_r.A = KalmanFilter_Yaw_R_cc( 'A', params_kalman, params_control, params_model,
params_simulation);
params_kalman.psi_r.B = KalmanFilter_Yaw_R_cc( 'B', params_kalman, params_control, params_model,
params_simulation);
params_kalman.psi_r.H = KalmanFilter_Yaw_R_cc( 'H', params_kalman, params_control, params_model,
params_simulation);
params_kalman.psi_r.Q = KalmanFilter_Yaw_R_cc( 'Q', params_kalman, params_control, params_model,
params_simulation);
params_kalman.psi_r.R = KalmanFilter_Yaw_R_cc( 'R', params_kalman, params_control, params_model,
params_simulation);
params_kalman.psi_r.F = 0;
% params_kalman.psi_r.K = KalmanFilter_Yaw_R_cc( 'K', params_kalman, params_control,
params_model, params_simulation);

%----- Calculate P for both filters as a function of time
P_initial = eye(2);

[t_psi_r, P_psi_r] = ode45(@(t,P)ODE_KALMAN_P(t, P, params_kalman.psi_r),...
[0 params_simulation.tf], P_initial, params_simulation.solver_options);

[t_phi_v, P_phi_v] = ode45(@(t,P)ODE_KALMAN_P(t, P, params_kalman.phi_v),...
[0 params_simulation.tf], P_initial, params_simulation.solver_options);

params_kalman.phi_v.P = P_phi_v;
params_kalman.phi_v.t_P = t_phi_v;

params_kalman.psi_r.P = P_psi_r;
params_kalman.psi_r.t_P = t_psi_r;

```

Run simulation

```

[t_sim, xi_sim] = ode45(@(t,xi)ode_active_sling(t, xi, ...
params_simulation, params_model, params_control, params_kalman), [0
params_simulation.tf], ...
params_simulation.initial_state, params_simulation.solver_options);

%----- Results in readable form
e321_psi_sim= xi_sim(:, 1);
psi_dot_sim = xi_sim(:, 4);

H321_e_inv = zeros(3,3*numel(t_sim));
R321_eh = zeros(3,3*numel(t_sim));

for m = 1:numel(t_sim)
    H321_e_inv(1:3,3*m-2:3*m) = (1/cos(xi_sim(m,2)))*[...
        0, sin(xi_sim(m,3)),
        cos(xi_sim(m,3));...
        0, cos(xi_sim(m,3))*cos(xi_sim(m,2)), -
sin(xi_sim(m,3))*cos(xi_sim(m,2));...
        cos(xi_sim(m,2)), sin(xi_sim(m,3))*sin(xi_sim(m,2)),
        cos(xi_sim(m,3))*sin(xi_sim(m,2))];

```

```

R321_eh(1:3,3*m-2:3*m) = [...
    cos(xi_sim(m,1))*cos(xi_sim(m,2)),    cos(xi_sim(m,2))*sin(xi_sim(m,1)),    -
sin(xi_sim(m,2));...
    %
    cos(xi_sim(m,1))*sin(xi_sim(m,3))*sin(xi_sim(m,2)) - cos(xi_sim(m,3))*sin(xi_sim(m,1)),...
    cos(xi_sim(m,3))*cos(xi_sim(m,1)) +
sin(xi_sim(m,3))*sin(xi_sim(m,1))*sin(xi_sim(m,2)),...
    cos(xi_sim(m,2))*sin(xi_sim(m,3));...
    %
    sin(xi_sim(m,3))*sin(xi_sim(m,1)) + cos(xi_sim(m,3))*cos(xi_sim(m,1))*sin(xi_sim(m,2)),...
    cos(xi_sim(m,3))*sin(xi_sim(m,1))*sin(xi_sim(m,2)) -
cos(xi_sim(m,1))*sin(xi_sim(m,3)),...
    cos(xi_sim(m,3))*cos(xi_sim(m,2))];
end

PQR_sim = zeros(3,numel(t_sim));
for m = 1:numel(t_sim)
PQR_sim(1:3,m) = H321_e_inv(1:3,3*m-2:3*m)'*xi_sim(m,4:6)';
end

xyz_h_dot_sim = zeros(3,numel(t_sim));
xyz_e_dot_sim = zeros(3,numel(t_sim));
for m = 1:numel(t_sim)
    xyz_h_dot_sim(1,m) =
params_model.sling_height_cmass*((cos(xi_sim(m,3))*xi_sim(m,6))*sin(xi_sim(m,1))+sin(xi_sim(m,3))
*(cos(xi_sim(m,1))*xi_sim(m,4))...
    +(-sin(xi_sim(m,3))*xi_sim(m,6))*sin(xi_sim(m,2))*cos(xi_sim(m,1))...
    +cos(xi_sim(m,3))*(cos(xi_sim(m,2))*xi_sim(m,5))*cos(xi_sim(m,1))...
    +cos(xi_sim(m,3))*sin(xi_sim(m,2))*(-sin(xi_sim(m,1))*xi_sim(m,4)));

    xyz_h_dot_sim(2,m) = params_model.sling_height_cmass*(-
((cos(xi_sim(m,3))*xi_sim(m,6))*cos(xi_sim(m,1))+sin(xi_sim(m,3))*(-
sin(xi_sim(m,1))*xi_sim(m,4))...
    +(-sin(xi_sim(m,3))*xi_sim(m,6))*sin(xi_sim(m,2))*sin(xi_sim(m,1))...
    +cos(xi_sim(m,3))*(cos(xi_sim(m,2))*xi_sim(m,5))*sin(xi_sim(m,1))...
    +cos(xi_sim(m,3))*sin(xi_sim(m,2))*(cos(xi_sim(m,1))*xi_sim(m,4)));

    xyz_h_dot_sim(3,m) = params_model.sling_height_cmass*(-
sin(xi_sim(m,3))*xi_sim(m,6))*cos(xi_sim(m,2))+cos(xi_sim(m,3))*(-sin(xi_sim(m,2))*xi_sim(m,5)));

    xyz_e_dot_sim(1:3,m) = R321_eh(1:3,3*m-2:3*m)*xyz_h_dot_sim(1:3,m);

end

xyz_h_sim = zeros(3,numel(t_sim));
for m = 1:numel(t_sim)
    xyz_h_sim(1,m) =
params_model.sling_height_cmass*(sin(xi_sim(m,3))*sin(xi_sim(m,1))+cos(xi_sim(m,3))*sin(xi_sim(m,
2))*cos(xi_sim(m,1)));
    xyz_h_sim(2,m) = params_model.sling_height_cmass*(-
sin(xi_sim(m,3))*cos(xi_sim(m,1))+cos(xi_sim(m,3))*sin(xi_sim(m,2))*sin(xi_sim(m,1)));

```



```

xyz_h_sim(3,m) = params_model.sling_height_cmass*(cos(xi_sim(m,3))*cos(xi_sim(m,2)));
end

```

Plot results

```

%----- Plot 3D Motion
figure;
plot3(xyz_h_sim(1, :), xyz_h_sim(2, :), xyz_h_sim(3, :), 'k');
grid on; title('Position'); xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');

%----- Plot Yaw and Yaw Rate
figure;
subplot(211); plot(t_sim, e321_psi_sim/params_simulation.deg2rad);
grid on; title('Yaw Angle'); xlabel('Time (S)'); ylabel('Angle (Deg)');

subplot(212); plot(t_sim, psi_dot_sim/params_simulation.deg2rad);
grid on; title('Yaw Rate'); xlabel('Time (S)'); ylabel('Angular velocity (Deg/sec)');

%----- Plot Orientation Angles from Helicopter
swing_angles = zeros(2, numel(t_sim));
for m = 1:numel(t_sim)
    swing_angles(1, m) = atan2(xyz_h_sim(2,m), -xyz_h_sim(3,m));
    swing_angles(2, m) = atan2(xyz_h_sim(1,m), -xyz_h_sim(3,m));
end

figure;
subplot(211); plot(t_sim, swing_angles(1,:)/params_simulation.deg2rad);
grid on; title('Lateral Swing Angle'); xlabel('Time (S)'); ylabel('Angle (Deg)');
axis([0,params_simulation.tf,-40,40]);

subplot(212); plot(t_sim, swing_angles(2,:)/params_simulation.deg2rad);
grid on; title('Longitudinal Swing Angle'); xlabel('Time (S)'); ylabel('Angle (Deg)');

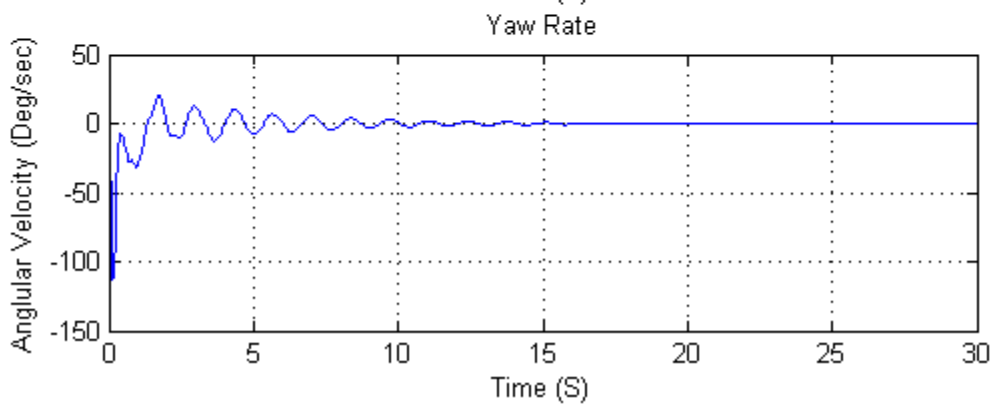
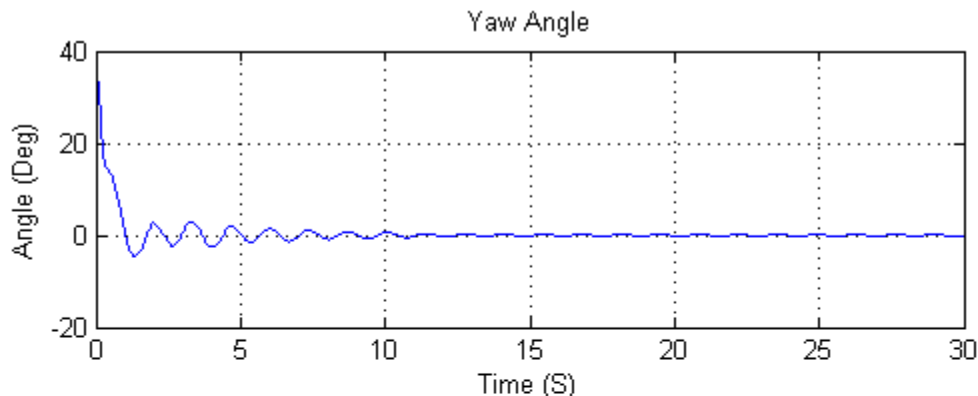
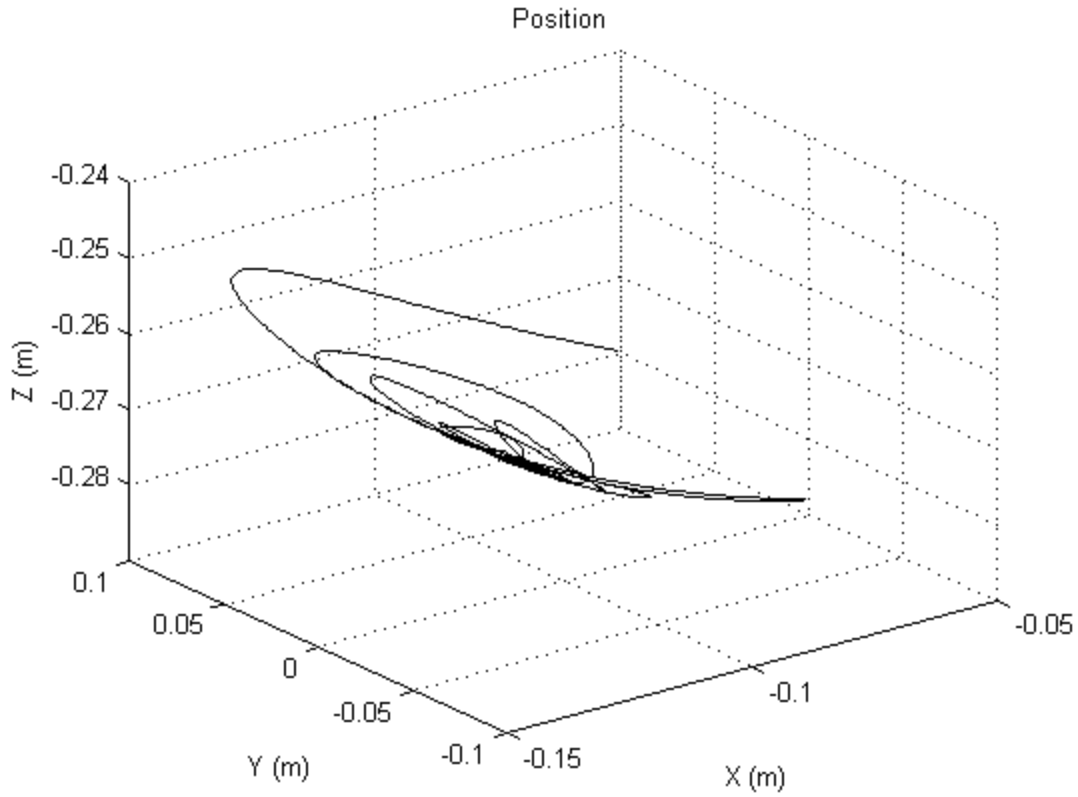
%----- Plot Difference in Kalman Filter States

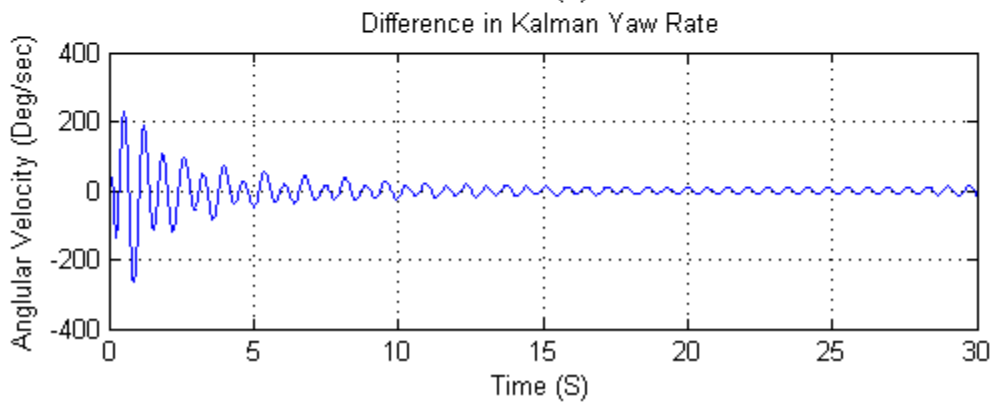
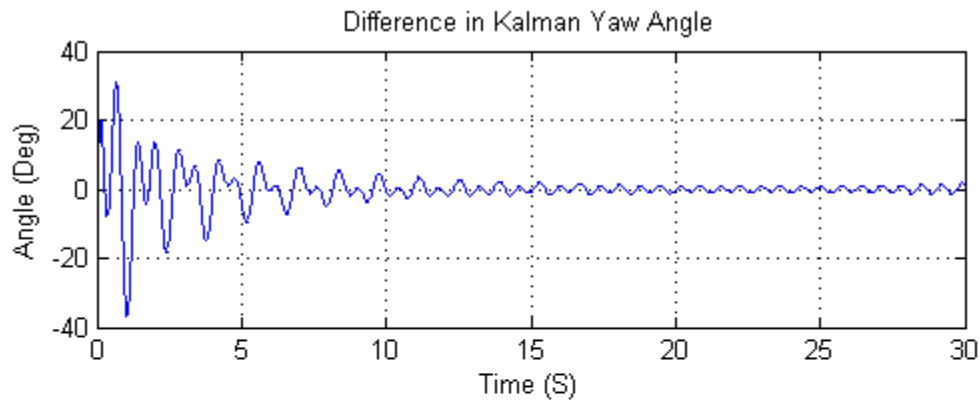
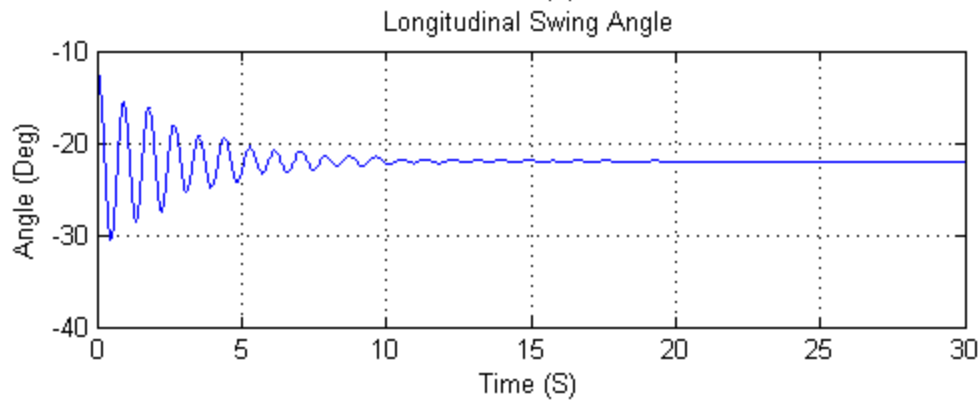
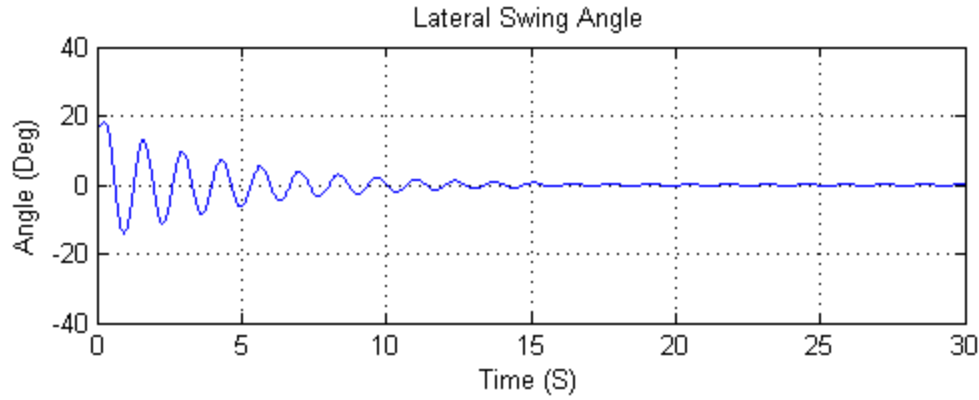
if params_simulation.use_kalman
figure
subplot(211); plot(t_sim, (xi_sim(:,7)-e321_psi_sim)/params_simulation.deg2rad); grid on;
title('Difference in kalman Yaw Angle'); xlabel('Time (S)'); ylabel('Angle (Deg)');
subplot(212); plot(t_sim, (xi_sim(:,8)-PQR_sim(3,:))/params_simulation.deg2rad); grid on;
title('Difference in kalman Yaw Rate'); xlabel('Time (S)'); ylabel('Angular velocity (Deg/sec)');

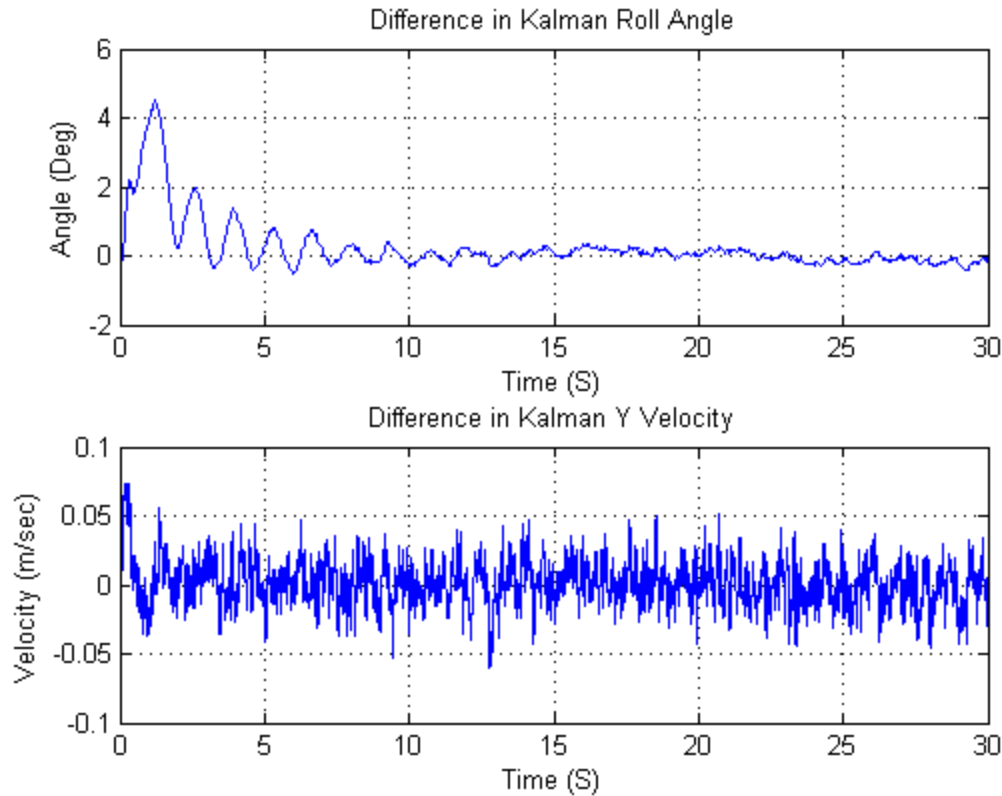
figure
subplot(211); plot(t_sim, (xi_sim(:,9)-xi_sim(:,3))/params_simulation.deg2rad); grid on;
title('Difference in kalman Roll Angle'); xlabel('Time (S)'); ylabel('Angle (Deg)');
subplot(212); plot(t_sim, (xi_sim(:,10)-xyz_e_dot_sim(2,:))); grid on; title('Difference in kalman Y velocity'); xlabel('Time (S)'); ylabel('Velocity (m/sec)');

else
end

```







Published with MATLAB® R2013b

State Derivative Function

```
function xi_dot = ode_active_sling(t, xi, params_simulation, params_model, params_control,
params_kalman)
```

Main Function Code

```
%{
state variable description
-----
    xi(1:3)      - angular position of c.m. of load
    xi(4:6)      - angular velocity of c.m. of load
    xi(7:10)     - kalman estimates of states

    Helicopter-fixed coordinates have subscript 'h'
    Load-fixed coordinates have subscript 'e'
%}
%----- Progress Counter (I know it slows it down, but I like it)
Percent_Complete = 100*(t/params_simulation.tf);
```

```

%----- States in readable form

e321_psi      = xi(1);
e321_theta   = xi(2);
e321_phi     = xi(3);
e321_psi_dot = xi(4);
e321_theta_dot = xi(5);
e321_phi_dot = xi(6);
kalman_psi_est = xi(7);
kalman_r_est  = xi(8);
kalman_phi_est = xi(9);
kalman_v_est  = xi(10);

%----- Establish Position and Velocity in Cartesian Coordinates

H0 = params_model.sling_height_cmass;

X_h = H0*(sin(e321_phi)*sin(e321_psi)+cos(e321_phi)*sin(e321_theta)*cos(e321_psi));
Y_h = H0*(-sin(e321_phi)*cos(e321_psi)+cos(e321_phi)*sin(e321_theta)*sin(e321_psi));
Z_h = H0*(cos(e321_phi)*cos(e321_theta));

X_h_dot =
H0*((cos(e321_phi)*e321_phi_dot)*sin(e321_psi)+sin(e321_phi)*(cos(e321_psi)*e321_psi_dot)...
    +(-sin(e321_phi)*e321_phi_dot)*sin(e321_theta)*cos(e321_psi)...
    +cos(e321_phi)*(cos(e321_theta)*e321_theta_dot)*cos(e321_psi)...
    +cos(e321_phi)*sin(e321_theta)*(-sin(e321_psi)*e321_psi_dot));

Y_h_dot = H0*(-((cos(e321_phi)*e321_phi_dot)*cos(e321_psi)+sin(e321_phi)*(-
sin(e321_psi)*e321_psi_dot))...
    +(-sin(e321_phi)*e321_phi_dot)*sin(e321_theta)*sin(e321_psi)...
    +cos(e321_phi)*(cos(e321_theta)*e321_theta_dot)*sin(e321_psi)...
    +cos(e321_phi)*sin(e321_theta)*(cos(e321_psi)*e321_psi_dot));

Z_h_dot = H0*((-sin(e321_phi)*e321_phi_dot)*cos(e321_theta)+cos(e321_phi)*(-
sin(e321_theta)*e321_theta_dot));

%----- Rotation matrices and such
R321_eh = [...
    cos(e321_psi)*cos(e321_theta),      cos(e321_theta)*sin(e321_psi),      -
sin(e321_theta);...
    %
    cos(e321_psi)*sin(e321_phi)*sin(e321_theta) - cos(e321_phi)*sin(e321_psi),...
    cos(e321_phi)*cos(e321_psi) + sin(e321_phi)*sin(e321_psi)*sin(e321_theta),...
    cos(e321_theta)*sin(e321_phi);...
    %
    sin(e321_phi)*sin(e321_psi) + cos(e321_phi)*cos(e321_psi)*sin(e321_theta),...
    cos(e321_phi)*sin(e321_psi)*sin(e321_theta) - cos(e321_psi)*sin(e321_phi),...
    cos(e321_phi)*cos(e321_theta)];

H321_e_inv = (1/cos(e321_theta))*[...
    0,                                sin(e321_phi),
    cos(e321_phi);...
    0,                                cos(e321_phi)*cos(e321_theta),      -

```

```

sin(e321_phi)*cos(e321_theta);...
    cos(e321_theta),      sin(e321_phi)*sin(e321_theta),
    cos(e321_phi)*sin(e321_theta)];

%----- Define omga_he_e and XYZ Dots in Body Frame

omga_he_e    = inv(H321_e_inv)*xi(4:6);
xyz_e_dot   = R321_eh*[X_h_dot;Y_h_dot;Z_h_dot];

P_phidot    = omga_he_e(1);
Q_thetadot  = omga_he_e(2);
R_psidot    = omga_he_e(3);

X_e_dot     = xyz_e_dot(1);
Y_e_dot     = xyz_e_dot(2);
Z_e_dot     = xyz_e_dot(3);

%----- Relative velocity and airspeed
V_e_air     = R321_eh*([-params_simulation.V_freestream;0;0] -[X_h_dot;Y_h_dot;Z_h_dot]);
    % velocity of the air flowing over the container in body frame
airspeed    = norm(V_e_air);
    % velocity magnitude
aoa_x       = atan2(V_e_air(3), V_e_air(2));
aoa_z       = atan2(V_e_air(2), -V_e_air(1));

%----- Control inputs
control_inputs =
control_law(e321_psi,R_psidot,Y_e_dot,kalman_psi_est,kalman_r_est,kalman_v_est,params_simulation)
;

%----- Forces and torques from control surfaces and sling legs
% [accel_tension_e, torque_tension_e]           = acceltorque_tension_e();
[accel_rudder_vstab_e, torque_rudder_vstab_e] = acceltorque_rudder_vstab_e();
[accel_pipes_e, torque_pipes_e]               = acceltorque_pipes_e();

%----- Collect all accelerations
accel_external_e = accel_aero_e() + ...
    accel_rudder_vstab_e + accel_pipes_e;

%----- Collect all moments
% torque_tension_e
% torque_aero_e()
% torque_rudder_vstab_e
% torque_pipes_e

torque_windup_e = [0;0;e321_psi*params_model.legs.C_windup];

torque_external_e = torque_aero_e() + ...
    torque_rudder_vstab_e + torque_pipes_e + torque_windup_e;
torque_gyroscopic_e = [...
    (params_model.load.Iyy - params_model.load.Izz)*omga_he_e(2)*omga_he_e(3); ...

```

```

(params_model.load.Izz - params_model.load.Ixx)*omega_he_e(3)*omega_he_e(1); ...
(params_model.load.Ixx - params_model.load.Iyy)*omega_he_e(1)*omega_he_e(2)];

%----- Kalman Filter Derivatives
kalman_state_dot = kalman_est_function();

%----- Euler Lagrange
force_external_e = params_model.load.mass*accel_external_e;
force_external_e_psi_theta_phi =
force_transform(force_external_e,e321_psi,e321_theta,e321_phi,H0);
force_external_h_psi_theta_phi = inv(R321_eh)*force_external_e_psi_theta_phi;

torque_external_e_psi_theta_phi =
moment_transform(torque_external_e+torque_gyroscopic_e,e321_psi,e321_theta,e321_phi,H0);
torque_external_h_psi_theta_phi = inv(R321_eh)*torque_external_e_psi_theta_phi;

%----- Collect all state derivatives
xi_dot          = zeros(6, 1);
% xi_dot(1:3)    = xi(4:6);
% xi_dot(4:6)    = R321_eh' * accel_external_e - [0; 0; params_simulation.g];
xi_dot(1:3)     = xi(4:6);
xi_dot(4:6)     = doubledot_solve(xi, force_external_h_psi_theta_phi,
torque_external_h_psi_theta_phi, params_model, params_simulation);
xi_dot(7:10)    = kalman_state_dot;

% (torque_external_e + torque_gyroscopic_e) ./ ...
% [params_model.load.Ixx; params_model.load.Iyy; params_model.load.Izz];

%=====

```

Error using ode_active_sling (line 14)
Not enough input arguments.

Aerodynamic Forces

```

function accel_e = accel_aero_e()

    aoa_x = atan2(v_e_air(3), v_e_air(2));
% angle of attack
    aoa_y = atan2(v_e_air(3), v_e_air(1));
% angle of attack
    aoa_z = atan2(v_e_air(2), v_e_air(1));
% angle of attack
    Rx      = abs(v_e_air(1) / airspeed);
    Ry      = abs(v_e_air(2) / airspeed);
    Rz      = abs(v_e_air(3) / airspeed);

%----- Force in body Y direction
if Ry < 0.82;
    C_ybasedip = 0;
else

```

```

        C_ybasedip = 1.94*(Ry-0.82);
    end

    if Ry <= 0.18
        C_ybubble0 = 2.5*Ry;
    elseif Ry < 0.82
        C_ybubble0 = 0.45-0.703*(Ry-0.18);
    else
        C_ybubble0 = 0;
    end
    C_ybubble = C_ybubble0*abs(cos(aoa_y)^3);
    C_y = -1.4*(Ry) + C_ybubble + C_ybasedip;
% side force coefficient

%----- Force in body Z direction
    if Rz < 0.82;
        C_zbasedip = 0;
    else
        C_zbasedip = 1.94*(Rz-0.82);
    end

    if Rz <= 0.18
        C_zbubble0 = 2.5*Rz;
    elseif Rz < 0.82
        C_zbubble0 = 0.45-0.703*(Rz-0.18);
    else
        C_zbubble0 = 0;
    end
    C_zbubble = C_zbubble0*abs(cos(aoa_z)^3);
    C_z = -1.4*(Rz)+C_zbubble+C_zbasedip;
% side force coefficient

%----- Force in X direction
    if Rx < 0.82;
        C_xbasedip = 0;
    else
        C_xbasedip = 1.94*(Rx-0.82);
    end

    if Rx <= 0.18
        C_xbubble0 = 2.5*Rx;
    elseif Rx < 0.82
        C_xbubble0 = 0.45-0.703*(Rx-0.18);
    else
        C_xbubble0 = 0;
    end
    C_xbubble = C_xbubble0*abs(cos(aoa_x)^3);
    C_x = -1.4*(Rx)+C_xbubble+C_xbasedip;
% side force coefficient

    accel_e = 0.5*params_simulation.rho_atm*(airspeed^2)*[...
        sign(V_e_air(1))*abs(C_x*params_model.load.area_x); ...
        sign(V_e_air(2))*abs(C_y*params_model.load.area_y); ...

```



```

sign(v_e_air(3))*abs(C_z*params_model.load.area_z)] / ...
params_model.load.mass;
end

```

Kalman Filters

```

function kalman_est_dot = kalman_est_function()
    if ~params_simulation.use_kalman
        kalman_est_dot(1,1) = 0;
        kalman_est_dot(2,1) = 0;
        kalman_est_dot(3,1) = 0;
        kalman_est_dot(4,1) = 0;
        return
    end

    % Standard Deviation

    std_accel = params_kalman.std_accel;
    std_mag = params_kalman.std_mag;
    std_gyro = params_kalman.std_gyro;

    % Error Counter
    if t == 0
        error_number = 1;
    else
        error_number = ceil(numel(params_kalman.error)*t/params_simulation.tf);
    end

    % P value counter

    t_phi_v = params_kalman.phi_v.t_P;

    P_phi_v_number = find( abs(t_phi_v-t) == min( abs(t_phi_v-t) ) );

    t_psi_r = params_kalman.psi_r.t_P;

    P_psi_r_number = find( abs(t_psi_r-t) == min( abs(t_psi_r-t) ) );

    % Psi and R

    A_k_psi_r = params_kalman.psi_r.A;
    B_k_psi_r = params_kalman.psi_r.B;
    H_k_psi_r = params_kalman.psi_r.H;
    Q_k_psi_r = params_kalman.psi_r.Q;
    R_k_psi_r = params_kalman.psi_r.R;

    P_k_psi_r_row = params_kalman.psi_r.P(P_psi_r_number,1:4);
    P_k_psi_r(1,1:2) = P_k_psi_r_row(1,1:2);
    P_k_psi_r(2,1:2) = P_k_psi_r_row(1,3:4);

    K_k_psi_r = P_k_psi_r*H_k_psi_r'*inv(R_k_psi_r); %params_kalman.psi_r.K;

```

```

Z_k_psi_r(1,1) = e321_psi+std_mag*params_kalman.error(error_number);
Z_k_psi_r(2,1) = R_psidot+std_gyro*params_kalman.error(error_number);

U_k_psi_r =
[control_inputs.total.rudder;control_inputs.total.door_left;control_inputs.total.door_right];

X_hat_psi_r = [xi(7);xi(8)];

kalman_est_dot(1:2,1) = A_k_psi_r*X_hat_psi_r+B_k_psi_r*U_k_psi_r+K_k_psi_r*(Z_k_psi_r-
H_k_psi_r*X_hat_psi_r);

% Phi and v
A_k_phi_v = params_kalman.phi_v.A;
B_k_phi_v = params_kalman.phi_v.B;
H_k_phi_v = params_kalman.phi_v.H;
Q_k_phi_v = params_kalman.phi_v.Q;
R_k_phi_v = params_kalman.phi_v.R;

P_k_phi_v_row = params_kalman.phi_v.P(P_phi_v_number,1:4);
P_k_phi_v(1,1:2) = P_k_phi_v_row(1,1:2);
P_k_phi_v(2,1:2) = P_k_phi_v_row(1,3:4);

K_k_phi_v = P_k_phi_v*H_k_phi_v'*inv(R_k_phi_v); %params_kalman.phi_v.K;

Z_k_phi_v(1,1) = e321_phi+std_mag*params_kalman.error(error_number);
Z_k_phi_v(2,1) = P_phidot+std_gyro*params_kalman.error(error_number);

U_k_phi_v = accel_external_e(2,1)+std_accel*params_kalman.error(error_number);

X_hat_phi_v = [xi(9);xi(10)];

kalman_est_dot(3:4,1) = A_k_phi_v*X_hat_phi_v+B_k_phi_v*U_k_phi_v+K_k_phi_v*(Z_k_phi_v-
H_k_phi_v*X_hat_phi_v);

end

%=====

```

Aerodynamic Moments

```

function torque_e = torque_aero_e()

    beta_side      = asin(v_e_air(2)/airspeed);
%side slip angle side (Yaw)
    beta_front     = asin(v_e_air(1)/airspeed);
% side slip angle front (Yaw - 90deg)
    phi_star       = atan2(v_e_air(3), v_e_air(2));
% front face crossflow angle (Roll)
    theta_star     = atan2(v_e_air(3), v_e_air(1));
% side face crossflow angle (Pitch)
    Rvy            = abs(v_e_air(2)/airspeed);
    Ruy            = abs(v_e_air(1)/airspeed);

```

```

if Rvy <= 0.28
    C_ymbubbleside = sign(v_e_air(2))*0.25*Rvy;
elseif Rvy < 0.6
    C_ymbubbleside = 0.07 - 0.219*(sign(v_e_air(1))*Rvy-0.28);
else
    C_ymbubbleside = 0;
end

if Ruy <= 0.2
    C_ymbubblefront = sign(v_e_air(1))*0.9*Ruy;
elseif Ruy < 0.7
    C_ymbubblefront = -0.18 + 0.219*(sign(v_e_air(1))*Ruy - 0.2);
else
    C_ymbubblefront = 0;
end

C_ymatt      = (-0.09*(1 - params_model.load.w_by_l^2)* ...
    sin(2*beta_side) + 0.04*params_model.load.w_by_l*...
    sin(4*beta_side))*cos(theta_star) + ...
    (0.075*(params_model.load.w_by_l^2)*sin(2*beta_front))*cos(phi_star);
deltaC_ym    = C_ymbubbleside*cos(theta_star)^2 + ...
    C_ymbubblefront*cos(phi_star)^2;
C_ym         = C_ymatt + deltaC_ym;
% yawing moment coefficient

torque_e     = [0; 0; C_ym*0.5*params_simulation.rho_atm*(airspeed^2)*...
    params_model.load.area_y*params_model.load.length];

end

%=====

```

Rudder Forces and Moments

```

function [accel_e, torque_e] = acceltorque_rudder_vstab_e()
    if ~params_simulation.use_rudder
        accel_e = zeros(3, 1);
        torque_e = zeros(3, 1);
        return
    end

    aoa      = atan2(v_e_air(2), -v_e_air(1));
    v_rud    = norm(v_e_air(1:2));

    CL_rud   = params_model.rudder.CL0*sin(aoa + control_inputs.total.rudder);
    CL_vs    = params_model.rudder.CL0*sin(aoa);

    CD_rud   = params_model.rudder.CD0*sin(e321_psi + control_inputs.total.rudder);
    CD_vs    = params_model.rudder.CD0*sin(e321_psi);

    q_rud    = 0.5*params_simulation.rho_atm*(v_rud^2);

```

```

accel_e = q_rud*[...
    -CD_rud*params_model.rudder.area - CD_vs*params_model.vstab.area; ...
    CL_rud*params_model.rudder.area + CL_vs*params_model.vstab.area; ...
    0] / params_model.load.mass;

torque_e= -q_rud*[0; 0; ...
    (CL_rud*params_model.rudder.area*params_model.rudder.loc + ...
    CL_vs*params_model.vstab.area*params_model.vstab.loc)];

end

%=====

```

Pipe Forces and Moments

```

function [accel_e, torque_e] = acceltorque_pipes_e()
    if ~params_simulation.use_pipes
        accel_e = zeros(3, 1);
        torque_e = zeros(3, 1);
        return
    end

    q_pip = params_simulation.rho_atm*(v_e_air(1)^2)*params_model.pipes.area;

    %----- Left door (Sperry's Door 2a)
    Fy_left = -q_pip*(sin(control_inputs.total.door_left))^2;
    Fx_left = -q_pip*sin(control_inputs.total.door_left)*(1 -
cos(control_inputs.total.door_left));
    M_2a = norm([Fx_left; Fy_left])*...
        (params_model.pipes.width - (params_model.pipes.door.length/2 + ...
        params_model.pipes.door.loc) / ...
        (tan(pi/2 - control_inputs.total.door_left)))* ...
        sin(pi/2 - control_inputs.total.door_left);

    %----- Right door (Sperry's Door 2b)
    Fy_right = q_pip*(sin(control_inputs.total.door_right))^2;
    Fx_right = -q_pip*sin(control_inputs.total.door_right)*(1 -
cos(control_inputs.total.door_right));
    M_2b = -norm([Fx_right; Fy_right])*...
        (params_model.pipes.width - (params_model.pipes.door.length/2 + ...
        params_model.pipes.door.loc) / ...
        (tan(pi/2 - control_inputs.total.door_right)))* ...
        sin(pi/2 - control_inputs.total.door_right);

    accel_e = [Fx_left + Fx_right; Fy_left + Fy_right; 0] / params_model.load.mass;
    torque_e = [0; 0; M_2a + M_2b];

end

%=====

```

Control Inputs

```

function control_inputs =
control_law(e321_psi,R_psidot,Y_e_dot,kalman_psi_est,kalman_r_est,kalman_v_est,params_simulation)
    if ~params_simulation.use_control
        control_inputs.delta.rudder      = 0;
        control_inputs.delta.door_left   = 0;
        control_inputs.delta.door_right  = 0;
        control_inputs.total.rudder     = params_control.rudder0;
        control_inputs.total.door_left   = params_control.door_left0;
        control_inputs.total.door_right  = params_control.door_right0;
        return
    end

    if ~params_simulation.use_kalmancontrols
        psi_est = e321_psi;
        R_est = R_psidot;
        V_est = Y_e_dot;
    else
        psi_est = kalman_psi_est;
        R_est = kalman_r_est;
        V_est = kalman_v_est;
    end

    delta_zeta      = [psi_est - params_control.psi0; ...
                      R_est - params_control.r0; ...
                      V_est - params_control.yd0];

    delta_u         = params_control.lqr*delta_zeta;

    control_inputs.delta.rudder      = delta_u(1);
    control_inputs.delta.door_left   = delta_u(2);
    control_inputs.delta.door_right  = delta_u(3);

    control_inputs.total.rudder     = min( (max( ...
        (control_inputs.delta.rudder + params_control.rudder0), ...
        params_control.sat.rudder_min )), params_control.sat.rudder_max);

    control_inputs.total.door_left   = min( (max( ...
        (control_inputs.delta.door_left + params_control.door_left0), ...
        params_control.sat.door_left_min )), params_control.sat.door_left_max);

    control_inputs.total.door_right  = min( (max( ...
        (control_inputs.delta.door_right + params_control.door_right0), ...
        params_control.sat.door_right_min )), params_control.sat.door_right_max);

end
end

```

Moment Transform Function

```
function moment_psi_theta_phi = moment_transform(M,psi,theta,phi,h0)

Mt = h0*[-sin(theta),sin(phi)*cos(theta),cos(phi)*cos(theta);...
         0,cos(phi),-sin(phi);...
         1,0,0];

moment_psi_theta_phi = Mt*M;
end
```

Error using moment_transform (line 3)
Not enough input arguments.

[*Published with MATLAB® R2014b*](#)

Force Transform Function

```
function force_psi_theta_phi = force_transform(F,psi,theta,phi,h0)

Ft = h0*[sin(phi)*cos(psi)-
cos(phi)*sin(theta)*sin(psi),sin(phi)*sin(psi)+cos(phi)*sin(theta)*cos(psi),0;...
        cos(phi)*cos(theta)*cos(psi),cos(phi)*cos(theta)*sin(psi),-cos(phi)*sin(theta);...
        cos(phi)*sin(psi)-sin(phi)*sin(theta)*cos(psi),-cos(phi)*cos(psi)-
sin(phi)*sin(theta)*sin(psi),sin(phi)*cos(theta)];

force_psi_theta_phi = Ft*F;
end
```

Error using force_transform (line 3)
Not enough input arguments.

[*Published with MATLAB® R2014b*](#)

Double Dot Solve Function

```
function doubledot = doubledot_solve(xi, Fe_psi_theta_phi, Me_psi_theta_phi, params_model,
params_simulation)

psi          = xi(1);
theta       = xi(2);
phi         = xi(3);
psi_dot     = xi(4);
theta_dot   = xi(5);
phi_dot     = xi(6);

m           = params_model.load.mass;
Ixx         = params_model.load.Ixx;
Iyy         = params_model.load.Iyy;
Izz         = params_model.load.Izz;
```

```

H0                = params_model.sling_height_cmass;

g                 = params_simulation.g;

A = [
                                (m*(4*H0^2*(sin(phi)*sin(psi) +
cos(phi)*cos(psi)*sin(theta))^2 + 2*H0^2*(cos(psi)*sin(phi) - cos(phi)*sin(psi)*sin(theta))^2))/2
+ Ixx*sin(theta)^2 + Izz*cos(phi)^2*cos(theta)^2 + Iyy*cos(theta)^2*sin(phi)^2,
(m*(4*H0^2*cos(phi)*cos(theta)*sin(psi)*(sin(phi)*sin(psi) + cos(phi)*cos(psi)*sin(theta)) +
2*H0^2*cos(phi)*cos(psi)*cos(theta)*(cos(psi)*sin(phi) - cos(phi)*sin(psi)*sin(theta))))/2 +
Iyy*cos(phi)*cos(theta)*sin(phi) - Izz*cos(phi)*cos(theta)*sin(phi), -
(m*(4*H0^2*(cos(phi)*cos(psi) + sin(phi)*sin(psi)*sin(theta))*(sin(phi)*sin(psi) +
cos(phi)*cos(psi)*sin(theta)) - 2*H0^2*(cos(phi)*sin(psi) -
cos(psi)*sin(phi)*sin(theta))*(cos(psi)*sin(phi) - cos(phi)*sin(psi)*sin(theta)))/2 -
Ixx*sin(theta);...
(m*(4*H0^2*cos(phi)*cos(theta)*sin(psi)*(sin(phi)*sin(psi) + cos(phi)*cos(psi)*sin(theta))
+ 2*H0^2*cos(phi)*cos(psi)*cos(theta)*(cos(psi)*sin(phi) - cos(phi)*sin(psi)*sin(theta)))/2 +
Iyy*cos(phi)*cos(theta)*sin(phi) - Izz*cos(phi)*cos(theta)*sin(phi),
(m*(2*H0^2*cos(phi)^2*cos(psi)^2*cos(theta)^2 + 4*H0^2*cos(phi)^2*cos(theta)^2*sin(psi)^2))/2 +
Iyy*cos(phi)^2 + Izz*sin(phi)^2,
(m*(4*H0^2*cos(phi)*cos(theta)*sin(psi)*(cos(phi)*cos(psi) + sin(phi)*sin(psi)*sin(theta)) -
2*H0^2*cos(phi)*cos(psi)*cos(theta)*(cos(phi)*sin(psi) - cos(psi)*sin(phi)*sin(theta)))/2;...
- (m*(4*H0^2*(cos(phi)*cos(psi) + sin(phi)*sin(psi)*sin(theta))*(sin(phi)*sin(psi) +
cos(phi)*cos(psi)*sin(theta)) - 2*H0^2*(cos(phi)*sin(psi) -
cos(psi)*sin(phi)*sin(theta))*(cos(psi)*sin(phi) - cos(phi)*sin(psi)*sin(theta)))/2 -
Ixx*sin(theta),
(m*(4*H0^2*cos(phi)*cos(theta)*sin(psi)*(cos(phi)*cos(psi) + sin(phi)*sin(psi)*sin(theta)) -
2*H0^2*cos(phi)*cos(psi)*cos(theta)*(cos(phi)*sin(psi) - cos(psi)*sin(phi)*sin(theta)))/2,
Ixx + (m*(4*H0^2*(cos(phi)*cos(psi) + sin(phi)*sin(psi)*sin(theta))^2 + 2*H0^2*(cos(phi)*sin(psi)
- cos(psi)*sin(phi)*sin(theta))^2))/2];

D = [Izz*phi_dot*theta_dot*cos(theta) - Iyy*phi_dot*theta_dot*cos(theta) -
Ixx*phi_dot*theta_dot*cos(theta) + H0^2*m*phi_dot^2*sin(2*psi) + H0^2*m*psi_dot^2*sin(2*psi) +
Ixx*psi_dot*theta_dot*sin(2*theta) - Iyy*psi_dot*theta_dot*sin(2*theta) -
Iyy*theta_dot^2*cos(phi)*sin(phi)*sin(theta) + Izz*theta_dot^2*cos(phi)*sin(phi)*sin(theta) -
4*H0^2*m*phi_dot*theta_dot*cos(theta) + 2*H0^2*m*phi_dot*psi_dot*sin(2*phi) +
2*Iyy*phi_dot*theta_dot*cos(phi)^2*cos(theta) - 2*Izz*phi_dot*theta_dot*cos(phi)^2*cos(theta) -
4*H0^2*m*phi_dot^2*cos(phi)^2*cos(psi)*sin(psi) - 4*H0^2*m*psi_dot^2*cos(phi)^2*cos(psi)*sin(psi)
- H0^2*m*theta_dot^2*cos(phi)^2*cos(psi)*sin(psi) -
H0^2*m*phi_dot^2*cos(psi)*cos(theta)^2*sin(psi) +
5*H0^2*m*phi_dot*theta_dot*cos(phi)^2*cos(theta) +
2*H0^2*m*phi_dot*theta_dot*cos(psi)^2*cos(theta) +
2*Iyy*phi_dot*psi_dot*cos(phi)*cos(theta)^2*sin(phi) -
2*Izz*phi_dot*psi_dot*cos(phi)*cos(theta)^2*sin(phi) +
2*Iyy*psi_dot*theta_dot*cos(phi)^2*cos(theta)*sin(theta) -
2*Izz*psi_dot*theta_dot*cos(phi)^2*cos(theta)*sin(theta) -
2*H0^2*m*phi_dot^2*cos(phi)*sin(phi)*sin(theta) - 2*H0^2*m*psi_dot^2*cos(phi)*sin(phi)*sin(theta)
- 2*H0^2*m*theta_dot^2*cos(phi)*sin(phi)*sin(theta) +
4*H0^2*m*phi_dot^2*cos(phi)*cos(psi)^2*sin(phi)*sin(theta) +
4*H0^2*m*psi_dot^2*cos(phi)*cos(psi)^2*sin(phi)*sin(theta) +
H0^2*m*theta_dot^2*cos(phi)*cos(psi)^2*sin(phi)*sin(theta) -
8*H0^2*m*phi_dot*psi_dot*cos(phi)*cos(psi)^2*sin(phi) +
H0^2*m*phi_dot*psi_dot*cos(phi)*cos(theta)^2*sin(phi) +
H0^2*m*psi_dot*theta_dot*cos(phi)^2*cos(theta)*sin(theta) +

```

$$\begin{aligned}
& 2*H0^2*m*phi_dot^2*cos(phi)^2*cos(psi)*cos(theta)^2*sin(psi) + \\
& 2*H0^2*m*psi_dot^2*cos(phi)^2*cos(psi)*cos(theta)^2*sin(psi) + \\
& 2*H0^2*m*theta_dot^2*cos(phi)^2*cos(psi)*cos(theta)^2*sin(psi) - \\
& 4*H0^2*m*phi_dot*theta_dot*cos(phi)^2*cos(psi)^2*cos(theta) - \\
& 4*H0^2*m*phi_dot*psi_dot*cos(psi)*sin(psi)*sin(theta) + \\
& 8*H0^2*m*phi_dot*psi_dot*cos(phi)^2*cos(psi)*sin(psi)*sin(theta) + \\
& 4*H0^2*m*phi_dot*psi_dot*cos(phi)*cos(psi)^2*cos(theta)^2*sin(phi) + \\
& 4*H0^2*m*psi_dot*theta_dot*cos(phi)^2*cos(psi)^2*cos(theta)*sin(theta) + \\
& 4*H0^2*m*psi_dot*theta_dot*cos(phi)*cos(psi)*cos(theta)*sin(phi)*sin(psi) - \\
& 4*H0^2*m*phi_dot*theta_dot*cos(phi)*cos(psi)*cos(theta)*sin(phi)*sin(psi)*sin(theta); \dots \\
& \quad Izz*phi_dot*psi_dot*cos(theta) - Iyy*phi_dot*psi_dot*cos(theta) + \\
& H0^2*m*phi_dot^2*sin(2*theta) - Iyy*phi_dot*theta_dot*sin(2*phi) + \\
& Izz*phi_dot*theta_dot*sin(2*phi) - 2*H0^2*m*phi_dot*psi_dot*cos(theta) + \\
& 2*Iyy*phi_dot*psi_dot*cos(phi)^2*cos(theta) - 2*Izz*phi_dot*psi_dot*cos(phi)^2*cos(theta) + \\
& H0^2*m*phi_dot*theta_dot*sin(2*phi) - Iyy*psi_dot*theta_dot*cos(phi)*sin(phi)*sin(theta) + \\
& Izz*psi_dot*theta_dot*cos(phi)*sin(phi)*sin(theta) - \\
& 4*H0^2*m*phi_dot^2*cos(phi)^2*cos(theta)*sin(theta) - \\
& H0^2*m*psi_dot^2*cos(phi)^2*cos(theta)*sin(theta) - \\
& H0^2*m*phi_dot^2*cos(psi)^2*cos(theta)*sin(theta) - \\
& 4*H0^2*m*theta_dot^2*cos(phi)^2*cos(theta)*sin(theta) + \\
& 5*H0^2*m*phi_dot*psi_dot*cos(phi)^2*cos(theta) + H0^2*m*phi_dot*psi_dot*cos(psi)^2*cos(theta) - \\
& H0^2*m*phi_dot*theta_dot*cos(phi)*cos(psi)^2*sin(phi) - \\
& H0^2*m*psi_dot*theta_dot*cos(phi)^2*cos(psi)*sin(psi) - \\
& 8*H0^2*m*phi_dot*theta_dot*cos(phi)*cos(theta)^2*sin(phi) + \\
& 2*H0^2*m*phi_dot^2*cos(phi)^2*cos(psi)^2*cos(theta)*sin(theta) + \\
& 2*H0^2*m*psi_dot^2*cos(phi)^2*cos(psi)^2*cos(theta)*sin(theta) + \\
& 2*H0^2*m*theta_dot^2*cos(phi)^2*cos(psi)^2*cos(theta)*sin(theta) - \\
& 4*H0^2*m*phi_dot*psi_dot*cos(phi)^2*cos(psi)^2*cos(theta) - \\
& 2*H0^2*m*psi_dot*theta_dot*cos(phi)*sin(phi)*sin(theta) + \\
& H0^2*m*psi_dot*theta_dot*cos(phi)*cos(psi)^2*sin(phi)*sin(theta) + \\
& H0^2*m*phi_dot*theta_dot*cos(phi)^2*cos(psi)*sin(psi)*sin(theta) + \\
& 4*H0^2*m*phi_dot*theta_dot*cos(phi)*cos(psi)^2*cos(theta)^2*sin(phi) + \\
& 4*H0^2*m*psi_dot*theta_dot*cos(phi)^2*cos(psi)*cos(theta)^2*sin(psi) + \\
& 2*H0^2*m*phi_dot^2*cos(phi)*cos(psi)*cos(theta)*sin(phi)*sin(psi) + \\
& 2*H0^2*m*psi_dot^2*cos(phi)*cos(psi)*cos(theta)*sin(phi)*sin(psi) - \\
& 4*H0^2*m*phi_dot*psi_dot*cos(phi)*cos(psi)*cos(theta)*sin(phi)*sin(psi)*sin(theta); \dots \\
& \quad - (m*(2*H0^2*(phi_dot*cos(phi)*sin(psi) + psi_dot*cos(psi)*sin(phi) + \\
& \quad theta_dot*cos(phi)*cos(psi)*cos(theta) - phi_dot*cos(psi)*sin(phi)*sin(theta) - \\
& \quad psi_dot*cos(phi)*sin(psi)*sin(theta))*(phi_dot*sin(phi)*sin(psi) - psi_dot*cos(phi)*cos(psi) + \\
& \quad phi_dot*cos(phi)*cos(psi)*sin(theta) + theta_dot*cos(psi)*cos(theta)*sin(phi) - \\
& \quad psi_dot*sin(phi)*sin(psi)*sin(theta)) + 4*H0^2*(psi_dot*sin(phi)*sin(psi) - \\
& \quad phi_dot*cos(phi)*cos(psi) + psi_dot*cos(phi)*cos(psi)*sin(theta) + \\
& \quad theta_dot*cos(phi)*cos(theta)*sin(psi) - \\
& \quad phi_dot*sin(phi)*sin(psi)*sin(theta))*(phi_dot*cos(phi)*sin(psi)*sin(theta) - \\
& \quad psi_dot*cos(phi)*sin(psi) - phi_dot*cos(psi)*sin(phi) + psi_dot*cos(psi)*sin(phi)*sin(theta) + \\
& \quad theta_dot*cos(theta)*sin(phi)*sin(psi)) + 2*H0^2*(cos(phi)*sin(psi) - \\
& \quad cos(psi)*sin(phi)*sin(theta))*(phi_dot^2*sin(phi)*sin(psi) + psi_dot^2*sin(phi)*sin(psi) + \\
& \quad phi_dot^2*cos(phi)*cos(psi)*sin(theta) + psi_dot^2*cos(phi)*cos(psi)*sin(theta) + \\
& \quad theta_dot^2*cos(phi)*cos(psi)*sin(theta) - 2*phi_dot*psi_dot*cos(phi)*cos(psi) + \\
& \quad 2*phi_dot*theta_dot*cos(psi)*cos(theta)*sin(phi) + \\
& \quad 2*psi_dot*theta_dot*cos(phi)*cos(theta)*sin(psi) - \\
& \quad 2*phi_dot*psi_dot*sin(phi)*sin(psi)*sin(theta)) - 4*H0^2*(cos(phi)*cos(psi) + \\
& \quad sin(phi)*sin(psi)*sin(theta))*(phi_dot^2*cos(phi)*sin(psi)*sin(theta) -
\end{aligned}$$


```

psi_dot^2*cos(psi)*sin(phi) - phi_dot^2*cos(psi)*sin(phi) +
psi_dot^2*cos(phi)*sin(psi)*sin(theta) + theta_dot^2*cos(phi)*sin(psi)*sin(theta) -
2*phi_dot*psi_dot*cos(phi)*sin(psi) - 2*psi_dot*theta_dot*cos(phi)*cos(psi)*cos(theta) +
2*phi_dot*psi_dot*cos(psi)*sin(phi)*sin(theta) +
2*phi_dot*theta_dot*cos(theta)*sin(phi)*sin(psi))/2 - Ixx*psi_dot*theta_dot*cos(theta)];

E = [H0^2*m*(phi_dot*cos(phi)*sin(psi) + psi_dot*cos(psi)*sin(phi) +
theta_dot*cos(phi)*cos(psi)*cos(theta) - phi_dot*cos(psi)*sin(phi)*sin(theta) -
psi_dot*cos(phi)*sin(psi)*sin(theta))*(psi_dot*sin(phi)*sin(psi) - phi_dot*cos(phi)*cos(psi) +
psi_dot*cos(phi)*cos(psi)*sin(theta) + theta_dot*cos(phi)*cos(theta)*sin(psi) -
phi_dot*sin(phi)*sin(psi)*sin(theta));...
H0*g*m*cos(phi)*sin(theta) - Ixx*psi_dot*cos(theta)*(phi_dot - psi_dot*sin(theta)) -
(m*(2*H0^2*(phi_dot*cos(psi)*cos(theta)*sin(phi) + psi_dot*cos(phi)*cos(theta)*sin(psi) +
theta_dot*cos(phi)*cos(psi)*sin(theta))*(phi_dot*cos(phi)*sin(psi) + psi_dot*cos(psi)*sin(phi) +
theta_dot*cos(phi)*cos(psi)*cos(theta) - phi_dot*cos(psi)*sin(phi)*sin(theta) -
psi_dot*cos(phi)*sin(psi)*sin(theta)) + 4*H0^2*(phi_dot*cos(theta)*sin(phi)*sin(psi) -
psi_dot*cos(phi)*cos(psi)*cos(theta) +
theta_dot*cos(phi)*sin(psi)*sin(theta))*(psi_dot*sin(phi)*sin(psi) - phi_dot*cos(phi)*cos(psi) +
psi_dot*cos(phi)*cos(psi)*sin(theta) + theta_dot*cos(phi)*cos(theta)*sin(psi) -
phi_dot*sin(phi)*sin(psi)*sin(theta)))/2 + Izz*psi_dot*cos(phi)*sin(theta)*(theta_dot*sin(phi) -
psi_dot*cos(phi)*cos(theta)) - Iyy*psi_dot*sin(phi)*sin(theta)*(theta_dot*cos(phi) +
psi_dot*cos(theta)*sin(phi));...
Izz*(theta_dot*cos(phi) + psi_dot*cos(theta)*sin(phi))*(theta_dot*sin(phi) -
psi_dot*cos(phi)*cos(theta)) - Iyy*(theta_dot*cos(phi) +
psi_dot*cos(theta)*sin(phi))*(theta_dot*sin(phi) - psi_dot*cos(phi)*cos(theta)) -
(m*(2*H0^2*(phi_dot*cos(phi)*sin(psi) + psi_dot*cos(psi)*sin(phi) +
theta_dot*cos(phi)*cos(psi)*cos(theta) - phi_dot*cos(psi)*sin(phi)*sin(theta) -
psi_dot*cos(phi)*sin(psi)*sin(theta))*(phi_dot*sin(phi)*sin(psi) - psi_dot*cos(phi)*cos(psi) +
phi_dot*cos(phi)*cos(psi)*sin(theta) + theta_dot*cos(psi)*cos(theta)*sin(phi) -
psi_dot*sin(phi)*sin(psi)*sin(theta)) + 4*H0^2*(psi_dot*sin(phi)*sin(psi) -
phi_dot*cos(phi)*cos(psi) + psi_dot*cos(phi)*cos(psi)*sin(theta) +
theta_dot*cos(phi)*cos(theta)*sin(psi) -
phi_dot*sin(phi)*sin(psi)*sin(theta))*(phi_dot*cos(phi)*sin(psi)*sin(theta) -
psi_dot*cos(phi)*sin(psi) - phi_dot*cos(psi)*sin(phi) + psi_dot*cos(psi)*sin(phi)*sin(theta) +
theta_dot*cos(theta)*sin(phi)*sin(psi)))/2 + H0*g*m*cos(theta)*sin(phi)];

doubledot = inv(A)*(Fe_psi_theta_phi + Me_psi_theta_phi + E - D);
end

```

Error using doubledot_solve (line 3)
Not enough input arguments.

[Published with MATLAB® R2014b](#)

LQR Gain Function

```

function K_lqr = calc_lqr_gain(params_model, params_control, params_simulation)
%----- Control Parameters
psi      = params_control.psi0;
R        = params_control.r0;
Ydot    = params_control.yd0;

```

```

Urud      = params_control.rudder0;
phi2a     = params_control.door_left0;
phi2b     = params_control.door_right0;
theta     = params_control.theta0;
phi       = params_control.phi0;
V         = params_control.v0;

%----- Sling-loaded cargo Parameters
mass      = params_model.load.mass;
WCC       = params_model.load.width;
LCC       = params_model.load.length;
HCC       = params_model.load.height;

AX        = params_model.load.area_x;
AY        = params_model.load.area_y;
AZ        = params_model.load.area_z;

W_L       = params_model.load.w_by_l;

Ixx       = params_model.load.Ixx;
Iyy       = params_model.load.Iyy;
Izz       = params_model.load.Izz;

rho       = params_simulation.rho_atm;
wuc       = params_model.legs.C_wind_up;

%----- Pipe Parameters
pipe_height      = params_model.pipes.height;
pipe_width      = params_model.pipes.width;
pipe_length     = params_model.pipes.length;
pipe_area       = params_model.pipes.area;

door_widthr    = params_model.pipes.door.length;
A_backdoor     = params_model.pipes.door.area;
Spacer         = params_model.pipes.door.loc;

%----- Rudder Parameters
H_rud          = params_model.rudder.height;
L_rud          = params_model.rudder.length;
A_rud          = params_model.rudder.area;
Cl_0           = params_model.rudder.Cl0;
Cd_0           = params_model.rudder.Cd0;
R_rud          = params_model.rudder.loc;

%----- Vertical stabilizer Parameters
H_vs = params_model.vstab.height;
L_vs = params_model.vstab.length;
A_vs = params_model.vstab.area;
R_vs = params_model.vstab.loc;

%----- A matrix
A_11 = 0;

```

```

% partial derivative of psidot with respect to psi
A_12 = 1;

% partial derivative of psidot with respect to R
A_13 = 0;

% partial derivative of psidot with respect to Ydot

A_21 = (1/Izz)*[...
    ((0.5*rho*vA2*Ay*Lcc)*[(-0.18*(1-(wcc/Lcc)^2)*cos(2*psi)+0.16*(wcc/Lcc)*cos(4*psi)]*...
    cos(theta)+(0.15*(wcc/Lcc)^2)*cos(2*(psi-pi/2)))*cos(phi)] + ...
    [(0.25*cos(psi))*cos(theta)^2]-(0.5*rho*vA2)*(c1_0*A_rud*R_rud+c1_0*A_vs*R_vs)+wuc];
%partial derivative of Rdot with respect to psi
A_22 = 0; %partial derivative of Rdot with respect to R
A_23 = 0; %partial derivative of Rdot with respect to Ydot

A_31 = (1/mass)*[(0.5*rho*Ay*(vA2))*(-
    1.4*cos(psi)+2.5*cos(psi)*abs(cos(theta)^3))+(0.5*rho*vA2)*(c1_0*A_rud+c1_0*A_vs)]; %partial
derivative of Ydoubledot with respect to psi
A_32 = 0; %partial derivative of Ydoubledot with respect to R
A_33 = 0; %partial derivative of Ydoubledot with respect to Ydot

A = [A_11,A_12,A_13;...
    A_21,A_22,A_23;...
    A_31,A_32,A_33];

%----- B Matrix
% 3 States psi, R, and Ydot. 5 Inputs Urud_0, U1a_0, U1b_0, U2a_0, U2b_0
A2 = rho*(vA2)*A_backdoor;
B2 = pipe_width;
C2 = pi/2;
D2 = (door_widthr/2+Spacer);

B_11 = 0; %partial derivative of psidot with respect to Urud
B_12 = 0; %partial derivative of psidot with respect to U2a
B_13 = 0; %partial derivative of psidot with respect to U2b

B_21 = (1/Izz)*[-0.5*rho*(vA2)*A_rud*c1_0*R_rud]; %partial derivative of Rdot with respect to
Urud

if sqrt(sin(phi2a)^2+cos(phi2a)^2-2*cos(phi2a)+1) == 0
    B_22 = 0;
else
    B_22 = (1/Izz)*[(A2*sin(phi2a)^2*(B2*sin(C2-phi2a)-D2*cos(C2-
    phi2a)))/sqrt(sin(phi2a)^2+cos(phi2a)^2-
    2*cos(phi2a)+1)+A2*cos(phi2a)*sqrt(sin(phi2a)^2+cos(phi2a)^2-2*cos(phi2a)+1)*(-B2*cos(C2-phi2a)-
    D2*sin(C2-phi2a))+A2*cos(phi2a)*sqrt(sin(phi2a)^2+cos(phi2a)^2-2*cos(phi2a)+1)*(B2*sin(C2-phi2a)-
    D2*cos(C2-phi2a))]; %partial derivative of Rdot with respect to U2a
end

if sqrt(sin(phi2b)^2+cos(phi2b)^2-2*cos(phi2b)+1) == 0
    B_23 = 0;
else
    B_23 = -(1/Izz)*[(A2*sin(phi2b)^2*(B2*sin(C2-phi2b)-D2*cos(C2-
    phi2b)))/sqrt(sin(phi2b)^2+cos(phi2b)^2-

```

```

2*cos(phi2b)+1)+A2*sin(phi2b)*sqrt(sin(phi2b)^2+cos(phi2b)^2-2*cos(phi2b)+1)*(-B2*cos(C2-phi2b)-
D2*sin(C2-phi2b))+A2*cos(phi2b)*sqrt(sin(phi2b)^2+cos(phi2b)^2-2*cos(phi2b)+1)*(B2*sin(C2-phi2b)-
D2*cos(C2-phi2b)); %partial derivitive of Rdot with respect to U2b
end

B_31 = (1/mass)*[0.5*rho*(V^2)*A_rud*2*pi]; %partial derivitive of Ydoubledot with respect to
Urud
B_32 = -(1/mass)*[A2^2*sin(phi2a)*cos(phi2a)]; %partial derivitive of Ydoubledot with respect to
U2a
B_33 = (1/mass)*[A2^2*sin(phi2b)*cos(phi2b)]; %partial derivitive of Ydoubledot with respect to
U2b

B = [B_11,B_12,B_13;...
     B_21,B_22,B_23;...
     B_31,B_32,B_33];

%----- Solve for input alterations

Q = [1,0,0;...
     0,3.5,0;...
     0,0,3.5];

R = [10,0,0;...
     0,1,0;...
     0,0,1];

N = 0;

%----- Calculate LQR Gain K
K_lqr = lqr(A,B,Q,R,N);

```

Error using calc_lqr_gain (line 3)
Not enough input arguments.

[*Published with MATLAB® R2014b*](#)

Kalman Filter P Derivative Function

```
function P_dot = ODE_KALMAN_P(t, P, params_kalman_FILTER)
```

```
%Calculation of P as a function of time for ODE45
```

Define Filter Matricies

```

P_matrix = [P(1),P(2);P(3),P(4)];
A = params_kalman_FILTER.A;
B = params_kalman_FILTER.B;
C = params_kalman_FILTER.H;
R_w = params_kalman_FILTER.Q;
R_v = params_kalman_FILTER.R;

```

```
F = params_kalman_FILTER.F;
L = P_matrix*C'*inv(R_v);
```

Error using ODE_KALMAN_P (line 6)
Not enough input arguments.

Calculate P_dot

```
P_dot_matrix = A*P_matrix+P_matrix*A'-P_matrix*C'*inv(R_v)*C*P_matrix+F*R_w*F';

P_dot(1:2,1) = P_dot_matrix(1,1:2)';
P_dot(3:4,1) = P_dot_matrix(2,1:2)';

end
```

[Published with MATLAB® R2014b](#)

Yaw and Yaw Rate Kalman Filter Function

```
function Matrix = KalmanFilter_Yaw_R_cc( Desired_Matrix, params_kalman, params_control,
params_model, params_simulation)
```

```
%Kalman Filter For Active Control Helicopter Sling Load
```

Define Constants

```
mass = params_model.load.mass; % mass of the conex container (kg)
wcc = params_model.load.width; % width of the conex container (m)
LCC = params_model.load.length; % Length of the conex container (m)
HCC = params_model.load.height; % Height of the conex container (m)

V = params_simulation.v_freestream; % velocity of the air flowing over the container in the -X
direction
rho = params_simulation.rho_atm; % Density of air (kg/m^3)

A_rud = params_model.rudder.area;
A_vs = params_model.vstab.area;
Cl_0 = params_model.rudder.Cl0;
R_rud = params_model.rudder.loc;
R_vs = params_model.vstab.loc;

pipe_width = params_model.pipes.width;
door_widthr = params_model.pipes.door.length;
A_backdoor = params_model.pipes.door.area;
Spacer = params_model.pipes.door.loc;

psi = params_control.psi0;
theta = params_control.theta0;
phi = params_control.phi0;
```

```

Urud = params_control.rudder0;
phi2a = params_control.door_left0;
phi2b = params_control.door_right0;

Ay = LCC * HCC;
Ax = HCC * WCC;
Az = WCC * LCC;

Ixx = (1/12)*mass*(wcc^2+hcc^2);
Iyy = (1/12)*mass*(hcc^2+lcc^2);
Izz = (1/12)*mass*(lcc^2+wcc^2);

std_accel = params_kalman.std_accel;
std_mag = params_kalman.std_mag;
std_gyro = params_kalman.std_gyro;

FC = 0.5*rho*v^2; %Force Constant

```

Error using KalmanFilter_Yaw_R_cc (line 5)
Not enough input arguments.

Define Matrices

A Matrix

```

A_11 = 0; %partial derivitive of psidot with respect to psi
A_12 = 1; %partial derivitive of psidot with respect to R

%partial derivitive of Rdot with respect to psi
A_21 = (-1/Izz)*[((0.5*rho*v^2*Ay*Lcc)*[(-0.18*(1-(wcc/Lcc)^2)*cos(2*psi)...
+0.16*(wcc/Lcc)*cos(4*psi)]*cos(theta)+(0.15*((wcc/Lcc)^2)*...
cos(2*(psi-pi/2)))*cos(phi)]+[0.25*cos(psi))*cos(theta)^2]-...
(0.5*rho*v^2)*(C1_0*A_rud*R_rud+C1_0*A_vs*R_vs)]; %+wuc];

A_22 = 0; %partial derivitive of Rdot with respect to R

A = [A_11,A_12;...
     A_21,A_22];

% B Matrix

A2 = rho*(v^2)*A_backdoor;
B2 = pipe_width;
C2 = pi/2;
D2 = (door_widthr/2+Spacer);

B_11 = 0; %partial derivitive of psidot with respect to Urud
B_12 = 0; %partial derivitive of psidot with respect to U2a
B_13 = 0; %partial derivitive of psidot with respect to U2b

%partial derivitive of Rdot with respect to Urud

```

```

B_21 = (1/Izz)*[-0.5*rho*(V^2)*A_rud*2*pi*R_rud];

%partial derivative of Rdot with respect to U2a
B_22 = (1/Izz)*[(A2*sin(phi2a)^2*(B2*sin(C2-phi2a)-D2*cos(C2-phi2a)))/...
    sqrt(sin(phi2a)^2+cos(phi2a)^2-2*cos(phi2a)+1)+A2*sin(phi2a)*...
    sqrt(sin(phi2a)^2+cos(phi2a)^2-2*cos(phi2a)+1)*(-B2*cos(C2-phi2a)-...
    D2*sin(C2-phi2a))+A2*cos(phi2a)*sqrt(sin(phi2a)^2+cos(phi2a)^2-...
    2*cos(phi2a)+1)*(B2*sin(C2-phi2a)-D2*cos(C2-phi2a))];

%partial derivative of Rdot with respect to U2b
B_23 = -(1/Izz)*[(A2*sin(phi2b)^2*(B2*sin(C2-phi2b)-D2*cos(C2-phi2b)))/...
    sqrt(sin(phi2b)^2+cos(phi2b)^2-2*cos(phi2b)+1)+A2*sin(phi2b)*...
    sqrt(sin(phi2b)^2+cos(phi2b)^2-2*cos(phi2b)+1)*(-B2*cos(C2-phi2b)-...
    D2*sin(C2-phi2b))+A2*cos(phi2b)*sqrt(sin(phi2b)^2+cos(phi2b)^2-...
    2*cos(phi2b)+1)*(B2*sin(C2-phi2b)-D2*cos(C2-phi2b))];

B = [B_11,B_12,B_13;...
    B_21,B_22,B_23];

% H Matrix

H = [1,0;...
    0,1];

% Q Matrix

Q = [0,0;...
    0,0];

% R Matrix

R = [std_mag,0;...
    0,std_gyro];

% % P Matrix
%
% A_re = A';
% B_re = H'*inv(R)*H;
% C_re = Q;
% P = are(A_re,B_re,C_re);

% K Matrix (gain)

% K = P*H'*inv(R);

```

Matricies

```

if strcmp(Desired_Matrix, 'A')
    Matrix = A;
elseif strcmp(Desired_Matrix, 'B')
    Matrix = B;
elseif strcmp(Desired_Matrix, 'H')

```

```

    Matrix = H;
elseif strcmp(Desired_Matrix, 'Q')
    Matrix = Q;
elseif strcmp(Desired_Matrix, 'R')
    Matrix = R;
else
    Matrix = 0;
end
end
end

```

Published with MATLAB® R2014b

Roll and Sway Velocity Kalman Filter Function

```
function Matrix = KalmanFilter_v_phi_cc(Desired_Matrix, params_kalman)
```

```
%Kalman Filter For Active Control Helicopter Sling Load
```

Define Constants

```

L = params_kalman.L;
std_accel = params_kalman.std_accel;
std_mag = params_kalman.std_mag;
std_gyro = params_kalman.std_gyro;

```

Error using KalmanFilter_v_phi_cc (line 6)
Not enough input arguments.

Define Matrices

```

% A Matrix

A_11 = 0; %partial derivitive of phidot with respect to phi
A_12 = 1/L; %partial derivitive of phidot with respect to v
A_21 = 0; %partial derivitive of vdot with respect to phi
A_22 = 0; %partial derivitive of vdot with respect to v

A = [A_11,A_12;...
     A_21,A_22];

% B Matrix

B_11 = 0; %partial derivitive of phidot with respect to A
B_21 = 1; %partial derivitive of vdot with respect to A

```



```

B = [B_11;...
     B_21];

% H Matrix

H = [1,0;...
     0,1/L];

% Q Matrix

Q = [0,0;...
     0,std_acce1^2];

% R Matrix

R = [std_mag,0;...
     0,std_gyro];

% % P Matrix
%
% A_re = A';
% B_re = H'*inv(R)*H;
% C_re = Q;
% P = are(A_re,B_re,C_re);

% K Matrix (gain)

% K = P*H'*inv(R);

```

Matrices

```

if strcmp(Desired_Matrix, 'A')
    Matrix = A;
elseif strcmp(Desired_Matrix, 'B')
    Matrix = B;
elseif strcmp(Desired_Matrix, 'H')
    Matrix = H;
elseif strcmp(Desired_Matrix, 'Q')
    Matrix = Q;
elseif strcmp(Desired_Matrix, 'R')
    Matrix = R;
else
    Matrix = 0;
end

end

```

[Published with MATLAB® R2014b](#)

Appendix D: ARDUINO DUE Control Code

Code

```
// WPI ACTIVE HELICOPTER SLING LOAD MQP
// ACTIVE CONTROL SCHEME FOR SPARTA SYSTEM
// MADE FOR ARDUINO DUE

// =====
// =====

// I2C Library
#include <Wire.h>

// Sensor Library
#include <Adafruit_Sensor.h>

// Accelerometer & Magnetometer
#include <Adafruit_LSM303_U.h>

// Gyroscope
#include <Adafruit_L3GD20_U.h>

// IMU API
#include <Adafruit_9DOF.h>

// Servo Library
#include <Servo.h>

// Create Servo Objects
Servo leftcontrol; // Left Door Servo
Servo rightcontrol; // Right Door Servo
Servo ruddercontrol; // Rudder Servo
```

```

// Assign a unique ID to the sensors
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_L3GD20_Unified gyro = Adafruit_L3GD20_Unified(20);
Adafruit_9DOF dof = Adafruit_9DOF();

// =====
// =====

// DECLARE AND/OR DEFINE VARIABLES

// Initial State Setup

int psi0 = 0;
int r0 = 0;
int vdot0 = 0;

// Left Door Initial Setup

double k21 = 0.014214081337476;
double k22 = 0.030863696983994;
double k23 = -0.047156356256757;

int posleftc0 = 15; // Initial Position

// Right Door Initial Setup

```

```
double k31 = -0.014214081337476;
double k32 = -0.030863696983994;
double k33 = 0.047156356256757;

int posrightc0 = 15; // Initial Position

// Rudder Initial Setup

double k11 = -5.937456420108622;
double k12 = -1.113428092780142;
double k13 = -0.997773800081550;

int posrudderc0 = 0; // Initial Position

// Servo Position Offsets based on current setup
int posrudderchard = 60; // Rudder
int posleftchard = 180; // Left Door
int posrightchard = 0; // Right Door

// Max Actuation positions with respect to servo position offsets
int maxrudder1 = posrudderchard + 40; // Rudder Direction 1
int maxrudder2 = posrudderchard - 40; // Rudder Direction 2
int maxleftd = posleftchard - 30; // Left Door
int maxrightd = posrightchard + 30; // Right Door
```



```
int finalposrudder, finalposleftc, finalposrightc; // Declare Final Position variables for Rudder,
Left and Right doors
```

```
// Create Data String for Printing to Serial Monitor (Used for viewing real-
time values and debugging)
```

```
// String dataString;
```

```
// =====
=====
```

```
// Initial Setup (Only Run Once)
```

```
void setup() {
```

```
    // Open Serial Monitor Communication at an 115200 baud rate (Only necessary for reading
real-time values or debugging)
```

```
    // Serial.begin(115200);
```

```
    // =====
```

```
    // ACCEL
```

```
    // Enable auto-ranging
```

```
    accel.enableAutoRange(true);
```

```
    if (!accel.begin())
```

```
    {
```

```
        // There was a problem detecting the ADXL345 ... check your connections
```

```
        // Serial.println(F("Oops, no LSM303 detected ... Check your wiring!"));
```

```
        while (1);
```

```
}  
  
// ACCEL END  
  
  
// MAG  
  
// Enable auto-ranging  
mag.enableAutoRange(true);  
  
if (!mag.begin())  
{  
  // There was a problem detecting the LSM303 ... check your connections  
  // Serial.println("Ooops, no LSM303 detected ... Check your wiring!");  
  while (1);  
}  
  
// MAG END  
  
  
// GYRO  
  
// Enable auto-ranging  
gyro.enableAutoRange(true);  
  
if (!gyro.begin())  
{  
  // There was a problem detecting the L3GD20 ... check your connections  
  // Serial.print("Ooops, no L3GD20 detected ... Check your wiring or I2C ADDR!");  
  while (1);  
}  
  
// GYRO END  
  
// =====
```

```
// Attatch Servos to Arduino
leftcontrol.attach(5); // Arduino I/O 5 with PWM
rightcontrol.attach(6); // Arduino I/O 6 with PWM
ruddercontrol.attach(7); // Arduino I/O 7 with PWM

// =====

// Send servos to their initial positions
ruddercontrol.write(posrudderc0);
rightcontrol.write(posrightc0);
leftcontrol.write(posleftc0);

// Get a new sensor event
sensors_event_t event;

// read mag and accel
sensors_event_t accel_event;
sensors_event_t mag_event;
sensors_vec_t orientation;

// Update events
accel.getEvent(&accel_event);
mag.getEvent(&mag_event);

// Sets reference "zero" heading point based on current tilt compesated heading
```



```

if (dof.magTiltCompensation(SENSOR_AXIS_Z, &mag_event, &accel_event))
{
  if (dof.magGetOrientation(SENSOR_AXIS_Z, &mag_event, &orientation))
  {
    initorientation = orientation.heading; // Define Initial Orientation
  }
}

}

// =====
// =====

// Control Loop (Run Continously after initial setup)
void loop() {

  // Get a new sensor event
  sensors_event_t event;

  // read mag and accel
  sensors_event_t accel_event;
  sensors_event_t mag_event;
  sensors_vec_t orientation;

  // read sensors and append data to the string:
  gyro.getEvent(&event);

  // assign angular rate (x, z) variables

```

```
z_ang_v = event.gyro.z ;
x_ang_v = event.gyro.x ;

// append angular rates to datastring
//  dataString += String(gyrox);
//  dataString += ", rad/s (x), ";
//  dataString += String(gyroz);
//  dataString += ", rad/s (z), ";

// Update events
accel.getEvent(&accel_event);
mag.getEvent(&mag_event);

// Use tilt compensated heading in order to calculate yaw angle
if (dof.magTiltCompensation(SENSOR_AXIS_Z, &mag_event, &accel_event))
{
  if (dof.magGetOrientation(SENSOR_AXIS_Z, &mag_event, &orientation))
  {
    heading = orientation.heading; // Define Heading
    yaw_est = (heading - initorientation); // Calculate Yaw
  }
}

//  dataString += "Heading, ";
//  dataString += String(heading);
//  dataString += ", Yaw, ";
//  dataString += String(yaw_est);
```

```

// *** CONTROL LAWS***

posleftc = posleftc0 + yaw_est * k21 + z_ang_v * k22 + H * x_ang_v * k23;
posrightc = posrightc0 + yaw_est * k31 + z_ang_v * k32 + H * x_ang_v * k33;
posrudderc = posrudderc0 + yaw_est * k11 + z_ang_v * k12 + H * x_ang_v * k13;

// Calculate position of servos based on control law values and servo positioning offsets
finalposrudder = posrudderchard + posrudderc;
finalposleftc = posleftchard - posleftc;
finalposrightc = posrightchard + posrightc;

// Prevents rudder from actuating more than 40 degrees in either direction
// ELSE: The rudder is commanded to go to calculated position
if ( finalposrudder > maxrudder1 || finalposrudder < maxrudder2 )
{
    // Do Nothing
}
else
{
    ruddercontrol.write(finalposrudder); // Send servo to calculated position
}

// Prevents left door from actuating more than 30 degrees
// ELSE: The left door is commanded to go to calculated position
if (finalposleftc < maxleftd)
{

```

```
// Do Nothing
}
else
{
    leftcontrol.write(finalposleftc); // Send sevo to calculated position
}

// Prevents right door from actuating more than 30 degrees
// ELSE: The right door is commanded to go to calculated positon
if (finalposrightc > maxrightd)
{
    // Do Nothing
}
else
{
    rightcontrol.write(finalposrightc); // Send sevo to calculated position
}

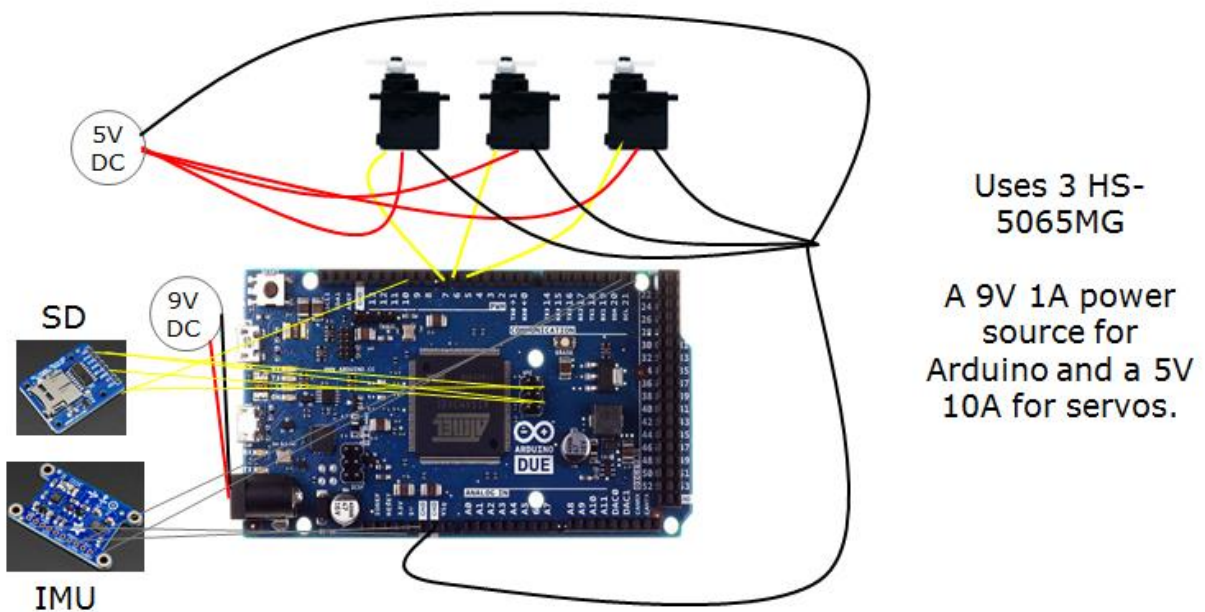
// dataString += ", C.L. Rudder Pos, ";
// dataString += String(posrudderc);
// dataString += ", Rudder Pos, ";
// dataString += String(finalposrudder);
// dataString += ", Left Door Pos, ";
// dataString += String(finalposleftc);
// dataString += ", RightDoor Pos, ";
// dataString += String(finalposrightc);
```

```
// Print DataString to Serial Monitor
// Serial.println(dataString);

// Insert Delay of control loop, if necessary
// delay(100);

}
```

Final Hardware Setup Reference



The Left Door Servo data wire is hooked up to I/O 5 (PWM)
 The Right Door Servo data wire is hooked up to I/O 6 (PWM)
 The Rudder Servo data wire is hooked up to I/O 7 (PWM)
 The IMU is hooked up using the TWI (Two Wire Interface) and a 5V input from the Arduino
 The SD card is not used in the final prototype setup. However, it is connected using the SPI communication interface on the Arduino and I/O 10.

Appendix E: ARDUINO DUE SD Card IMU Data-logging Code

[code]

```

// WPI ACTIVE HELICOPTER SLING LOAD MQP

// SD CARD IMU DATALOGGING

// MADE FOR ARDUINO DUE

// Measures Inertial Acceleration (m/s2) in 3 Axes (x,y,z), Angular Rates (rad/s) in 3 Axes, and
// Magnetic Field ( $\mu$ T) in 3 Axes, and time since start. Calculates heading angle, roll angle, and
// pitch angle. Records all data to a 'datalog.txt' file on any SD card.

// =====
// =====

// I2C Library
#include <Wire.h>

// Sensor Library
#include <Adafruit_Sensor.h>

// Accelerometer & Magnetometer
#include <Adafruit_LSM303_U.h>

// Gyroscope
#include <Adafruit_L3GD20_U.h>

// IMU API
#include <Adafruit_9DOF.h>

// SPI Library
#include <SPI.h>

// SD Card Library
#include <SD.h>

// SD Card Setup

// On the Ethernet Shield, CS is pin 4. Note that even if it's not
// used as the CS pin, the hardware CS pin (10 on most Arduino boards,

```

```

// 53 on the Mega) must be left as an output or the SD library
// functions will not work.
const int chipSelect = 10;
// SD Card Setup END

// Assign a unique ID to the sensors
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_L3GD20_Unified gyro = Adafruit_L3GD20_Unified(20);
Adafruit_9DOF dof = Adafruit_9DOF();

// =====
=====

void displaySensorDetails(void)
{
  sensor_t sensor;

  accel.getSensor(&sensor);
  Serial.println(F("----- ACCELEROMETER -----"));
  Serial.print (F("Sensor:  ")); Serial.println(sensor.name);
  Serial.print (F("Driver Ver:  ")); Serial.println(sensor.version);
  Serial.print (F("Unique ID:  ")); Serial.println(sensor.sensor_id);
  Serial.print (F("Max Value:  ")); Serial.print(sensor.max_value); Serial.println(F(" m/s^2"));
  Serial.print (F("Min Value:  ")); Serial.print(sensor.min_value); Serial.println(F(" m/s^2"));
  Serial.print (F("Resolution:  ")); Serial.print(sensor.resolution); Serial.println(F(" m/s^2"));
  Serial.println(F("-----"));
  Serial.println(F(""));
}

```

```

gyro.getSensor(&sensor);
Serial.println(F("----- GYROSCOPE -----"));
Serial.print (F("Sensor:   ")); Serial.println(sensor.name);
Serial.print (F("Driver Ver:  ")); Serial.println(sensor.version);
Serial.print (F("Unique ID:  ")); Serial.println(sensor.sensor_id);
Serial.print (F("Max Value:  ")); Serial.print(sensor.max_value); Serial.println(F(" rad/s"));
Serial.print (F("Min Value:  ")); Serial.print(sensor.min_value); Serial.println(F(" rad/s"));
Serial.print (F("Resolution:  ")); Serial.print(sensor.resolution); Serial.println(F(" rad/s"));
Serial.println(F("-----"));
Serial.println(F(""));

```

```

mag.getSensor(&sensor);
Serial.println(F("----- MAGNETOMETER -----"));
Serial.print (F("Sensor:   ")); Serial.println(sensor.name);
Serial.print (F("Driver Ver:  ")); Serial.println(sensor.version);
Serial.print (F("Unique ID:  ")); Serial.println(sensor.sensor_id);
Serial.print (F("Max Value:  ")); Serial.print(sensor.max_value); Serial.println(F(" uT"));
Serial.print (F("Min Value:  ")); Serial.print(sensor.min_value); Serial.println(F(" uT"));
Serial.print (F("Resolution:  ")); Serial.print(sensor.resolution); Serial.println(F(" uT"));
Serial.println(F("-----"));
Serial.println(F(""));

```

```

delay(500);
}

```

```

// =====
=====

```



```
void setup() {

  Serial.begin(115200);
  Serial.println(F("AHSL")); Serial.println("");

  // SD Card Setup
  Serial.print("Initializing SD card...");
  // make sure that the default chip select pin is set to
  // output, even if you don't use it:
  pinMode(10, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("card initialized.");
  // SD Card Setup END

  // Initialise the sensors

  // ACCEL
  // Enable auto-ranging
  accel.enableAutoRange(true);
```

```
if (!accel.begin())
{
  // There was a problem detecting the ADXL345 ... check your connections
  Serial.println(F("Oops, no LSM303 detected ... Check your wiring!"));
  while (1);
}
// ACCEL END

// MAG
// Enable auto-ranging
mag.enableAutoRange(true);

if (!mag.begin())
{
  // There was a problem detecting the LSM303 ... check your connections
  Serial.println("Oops, no LSM303 detected ... Check your wiring!");
  while (1);
}
// MAG END

// GYRO
// Enable auto-ranging
gyro.enableAutoRange(true);

if (!gyro.begin())
{
```

```

// There was a problem detecting the L3GD20 ... check your connections
Serial.print("Oops, no L3GD20 detected ... Check your wiring or I2C ADDR!");
while (1);
}
// GYRO END

// Display some basic information on this sensor
displaySensorDetails();

}

// =====
=====

void loop() {

// Get a new sensor event
sensors_event_t event;

// make a string for assembling the data to log:
String dataString = "";

// Setup Timestamp
double currentMillis = millis();
double timestamp = currentMillis / 1000;

// read sensors and append data to the string:

```

```
accel.getEvent(&event);  
  
// create acceleration (x, y, z) variables  
double accelx = event.acceleration.x ;  
double accely = event.acceleration.y ;  
double accelz = event.acceleration.z ;  
  
  
// append accelerations to datastring  
dataString += String(accelx);  
dataString += ", m/s^2 (x), ";  
dataString += String(accely);  
dataString += ", m/s^2 (y), ";  
dataString += String(accelz);  
dataString += ", m/s^2 (z), ";  
  
  
// read sensors and append data to the string:  
gyro.getEvent(&event);  
  
// create angular rate (x, y, z) variables  
double gyroX = event.gyro.x ;  
double gyroY = event.gyro.y ;  
double gyroZ = event.gyro.z ;  
  
  
// append angular rates to datastring  
dataString += String(gyroX);  
dataString += ", rad/s (x), ";  
dataString += String(gyroY);  
dataString += ", rad/s (y), ";
```

```

dataString += String(gyroz);
dataString += ", rad/s (z), ";

// read sensors and append data to the string:
mag.getEvent(&event);
// create magnetic field (x, y, z) variables
double magx = event.magnetic.x ;
double magy = event.magnetic.y ;
double magz = event.magnetic.z ;

// append magnetic field values to datastring
dataString += String(magx) ;
dataString += ", uT (x), " ;
dataString += String(magy) ;
dataString += ", uT (y), " ;
dataString += String(magz) ;
dataString += ", uT (z), " ;

// read mag and accel
sensors_event_t accel_event;
sensors_event_t mag_event;
sensors_vec_t orientation;

// // append attitude and heading variables to datastring
// /* Calculate pitch and roll from the raw accelerometer data */
// accel.getEvent(&accel_event);
// mag.getEvent(&mag_event);

```

```

//
//

// MAG HEADING WITH ACCEL (INCLINO) PITCH COMPENSATION
// read mag and accel data
accel.getEvent(&accel_event);
mag.getEvent(&mag_event);

//// create tilt compensated heading
//double truehead;

// tilt compensation
if (dof.magTiltCompensation(SENSOR_AXIS_Z, &mag_event, &accel_event))
{
    // Do something with the compensated data in mag_event!
    if (dof.fusionGetOrientation(&accel_event, &mag_event, &orientation))
    {
        dataString += String(orientation.roll) ;
        dataString += ", roll angle, " ;
        dataString += String(orientation.pitch) ;
        dataString += ", pitch angle, " ;
        dataString += String(orientation.heading) ;
        dataString += ", heading angle, " ;
    }
}
}

```

```
// append measurement second timestand to datastring
dataString += String(timestamp);
dataString += " , seconds";

//// Delay .015 seconds
//delay(15);

// open the file. note that only one file can be open at a time,
// so you have to close this one before opening another.
File dataFile = SD.open("datalog.txt", FILE_WRITE);

// if the file is available, write to it:
if (dataFile) {
  dataFile.println(dataString);
  dataFile.close();
  // print to the serial port too:
  Serial.println(dataString);
}
// if the file isn't open, pop up an error:
else {
  Serial.println("error opening datalog.txt");
}
delay (500);
}
```

[/code]

Appendix F: Pipes And Rudder Calculations MATLAB Code

Information

```
%This program was created for the Active Sling Load Stabilization MQP at
%Worcester Polytechnic Institute. With inputs of freestream velocity,
%dimensions, and door angles, it calculates the theoretical forces and
%moments caused by the "Pipes" design.
```

```
close all; clear all; clc
```

Conversion Factors and Scaling

```
s = 1;
; %scaling factor

convert = 0; %switch for conversion
if convert == 1
    n2lbf = 0.224808943871; %conversion factor from Newtons to Pound-force
    m2in = 39.3701; %convert m to in
    nm2lbf_in = 0.005710147162769201; %conversion factor from Newton-meters to Pound-force-inches
else
    n2lbf = 1;
    m2in = 1;
    nm2lbf_in = 1;
end
```

Inputs

```
%Pipe Constants
rho = 1.225; %density of air
if s==1
    v = 30.87; %freestream velocity
else
    v = 12.7;
end
door_height = 0.402/s; %height of door
door_widthf = 0.800/s; %width of front door
door_widthr = 1.000/s; %width of rear door
phi = 0:0.01:50; %door angle from vertical (clockwise from 12)

%Vertical stabilizer Constants
aoa = 0:0.01:45; %angle of attack of VS and rudder in degrees
aoar = aoa*(pi/180); %... in radians
vs_root_chord = .7/s; %vertical stabilizer root chord
vs_tip_chord = 0/s; %vertical stabilizer tip chord
vs_span = .6/s; %vertical stabilizer span

%Rudder Constants
rd_chord = .5/s; %rudder chord
```

```
rd_span = .6/s;           %rudder span
```

% Container Dimensions

```
L = 2.4384/s; %length of TRICON (m)
W = 1.9812/s; %width of TRICON (m)
H = 2.4384/s; %height of TRICON (m)
```

Pipe Equations

%Geometry

```
pipe_width = (0.787-0.04)/s;           %width of pipes
Spacef     = (1.143/s)-door_widthf;    %space between front doors and cm
Spacer     = (1.143/s)-door_widthr;    %space between rear doors and cm
A_ef       = door_height*door_widthf;  %area of front door
A_er       = door_height*door_widthr;  %area of rear door
C_d        = 2*sin((pi/180)*phi).*sin((pi/180)*phi); %drag coeff vs. door angle from
http://mekside.com/wings-redux/
A_if       = A_ef.*(sin((pi/180)*phi)); %capture area of front door
A_ir       = A_er.*(sin((pi/180)*phi)); %capture area of rear door
```

%Force Equations

```
Df = n2lbf*0.5*rho*(v^2)*C_d.*A_if;           %drag force due to
seperated flow (front door)
Dr = n2lbf*0.5*rho*(v^2)*C_d.*A_ir;           %drag force due to
seperated flow (rear door)
F_yf = n2lbf*rho*(v^2)*(sin((pi/180)*phi)).*sin((pi/180)*phi)*A_ef; %restoring (y) force
(FRONT)
F_xf = -n2lbf*rho*(v^2).*A_if.*(1-cos((pi/180)*phi)); %attached flow (x) force
for small angles of attack (<20 degrees) (FRONT)
F_xxf = -n2lbf*rho*(v^2).*A_if.*(1-cos((pi/180)*phi))-Df; %attached flow (x) force
for all angles (drag included - estimate)(FRONT)
F_yr = n2lbf*rho*(v^2)*(sin((pi/180)*phi)).*sin((pi/180)*phi)*A_er; %restoring (y) force
(REAR)
F_xr = -n2lbf*rho*(v^2).*A_ir.*(1-cos((pi/180)*phi)); %attached flow (x) force
for small angles of attack (<20 degrees)(REAR)
F_xxr = -n2lbf*rho*(v^2).*A_ir.*(1-cos((pi/180)*phi))-Dr; %attached flow (x) force
for all angles (drag included - estimate)(REAR)
```

%Moment Arm Equations

```
BTWr = (door_widthr/2+Spacer)./(tan((pi/180).*(90-phi))); %intermediate calculation of Big
Triangle width (rear door)
HYPf = sqrt((pipe_width.^2)+((door_widthf/2+Spacef).^2)); %intermediate calculation of
Hypotenuse of Big Triangle (front door)
psi = acosd(pipe_width/HYPf); %intermediate calculation of
extra angle psi (front door)
HYPr = pipe_width-BTWr; %intermediate calculation of
Hypotenuse of Smaller Triangle (rear door)
theta = 180-(90-phi)-psi; %intermediate calculation of
extra angle theta (front door)
r_f = HYPf.*sin((pi/180)*theta); %moment arm of FRONT door
angular momentum
r_r = HYPr.*sin((pi/180)*(90-phi)); %moment arm of REAR door angular
```

momentum

%Left Side Moments

```

M_1_a = nm2lbfm*sqrt(F_xf.^2+F_yf.^2).*(m2in*r_f); %door 1A moment of angular momentum
M_3_a = nm2lbfm*sqrt(F_xr.^2+F_yr.^2).*(m2in*r_r); %door 3A moment of angular momentum
M_1_ad = nm2lbfm*sqrt(F_xxf.^2+F_yf.^2).*(m2in*r_f); %door 1A moment of angular momentum
(attached + seperated flow)
M_3_ad = nm2lbfm*sqrt(F_xxr.^2+F_yr.^2).*(m2in*r_r); %door 3A moment of angular momentum
(attached + seperated flow)

```

%Right Side Moments

```

M_1_b = -nm2lbfm*sqrt(F_xf.^2+F_yf.^2).*(m2in*r_f); %door 1B moment of angular momentum
M_3_b = -nm2lbfm*sqrt(F_xr.^2+F_yr.^2).*(m2in*r_r); %door 3B moment of angular momentum
M_1_bd = -nm2lbfm*sqrt(F_xxf.^2+F_yf.^2).*(m2in*r_f); %door 1B moment of angular momentum
M_3_bd = -nm2lbfm*sqrt(F_xxr.^2+F_yr.^2).*(m2in*r_r); %door 3B moment of angular momentum

```

Vertical Stabilizer and Rudder Equations

```

A = vs_root_chord; %same as vs_root_chord
B = vs_tip_chord; %same as vs_tip_chord
a_vs = 0.5*vs_span*vs_root_chord; %area of vertical stabilizer
a_rd = rd_chord*rd_span; %area of rudder
MAC = A-(2*(A-B)*(0.5*A+B) / (3*(A+B))); %mean aerodynamic chord
QMAC = MAC*0.25; %quarter-chord mean aerodynamic chord
Cd = 1.28*sin(aoar); %drag coefficient
Cl = 2*pi*aoar; %lift coefficient

```

%Drag & Lift Forces

```

Drag_vs = 0.5*rho*(V^2)*a_vs*Cd; %drag force from vertical stabilizer
Drag_rd = 0.5*rho*(V^2)*a_rd*Cd; %drag force from rudder
Lift_vs = 0.5*rho*(V^2)*a_vs*Cl; %lift force from vertical stabilizer
Lift_rd = 0.5*rho*(V^2)*a_rd*Cl; %lift force from rudder

```

%Moment Arms

```

r_rd = (0.5*L); %moment arm of rudder
r_vs = (0.5*L)-(0.25*rd_chord)-(0.75*MAC); %moment arm of vertical stabilizer

```

%Moments

```

Moment_vs = r_vs*Lift_vs; %moment caused by vertical stabilizer
Moment_rd = r_rd*Lift_rd; %moment caused by rudder

```

Plotting

%Note: x points downwards, y points to the left, z points upwards

```

figure
subplot(2,2,1)
plot(phi,-F_yr,phi,F_xr,phi,F_xxr,':',phi,0,':') %F_y reaction force points to the right (-)
title('Left Rear Door Forces as a Function of Door Angle \phi');
xlabel('\phi (deg)');
if convert == 1

```

```

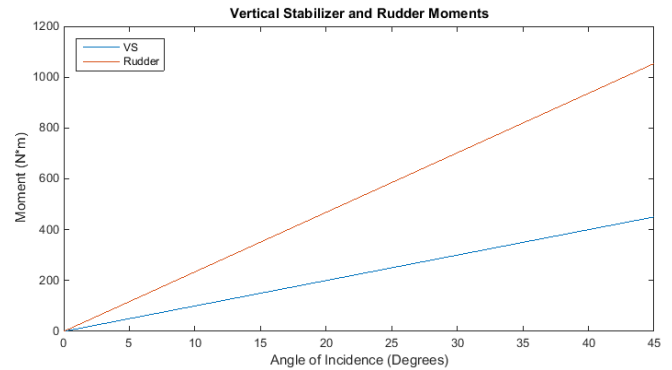
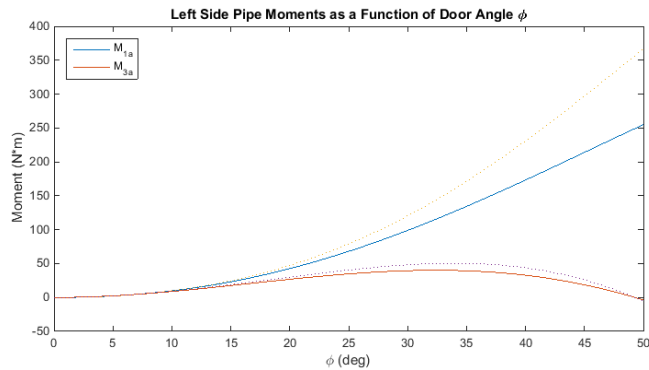
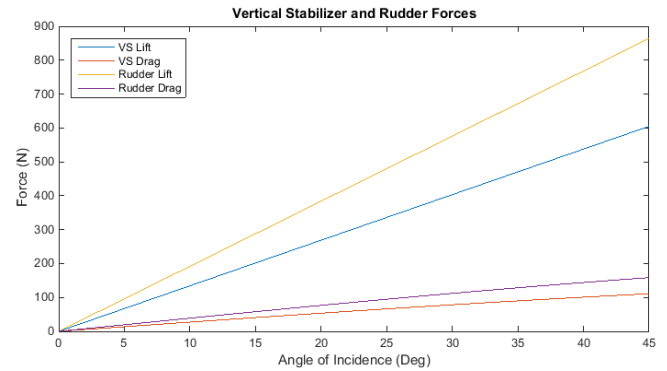
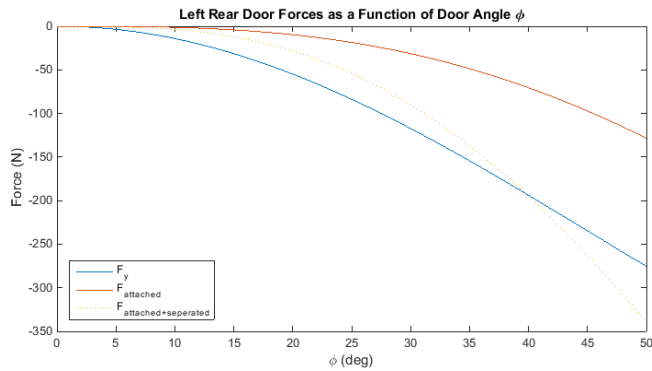
    ylabel('Force (lbf)');
else
    ylabel('Force (N)');
end
legend('F_y','F_a_t_t_a_c_h_e_d','F_a_t_t_a_c_h_e_d+_s_e_p_e_r_a_t_e_d','Location','southwest');

subplot(2,2,3)
plot(phi,M_1_a,phi,M_3_a,phi,M_1_ad,':',phi,M_3_ad,':')
title('Left Side Pipe Moments as a Function of Door Angle \phi');
xlabel('\phi (deg)');
if convert ==1
    ylabel('Moment (lbf*in)');
else
    ylabel('Moment (N*m)');
end
legend('M_1_a','M_3_a','Location','northwest');

subplot(2,2,2)
plot(aoa,Lift_vs,aoa,Drag_vs,aoa,Lift_rd,aoa,Drag_rd)
title('Vertical Stabilizer and Rudder Forces')
xlabel({'Angle of Incidence (Deg)'});
ylabel({'Force (N)'});
legend('VS Lift','VS Drag','Rudder Lift','Rudder Drag','Location','northwest');
xlim([0,45]);

subplot(2,2,4)
plot(aoa,Moment_vs,aoa,Moment_rd)
title('Vertical Stabilizer and Rudder Moments')
xlabel({'Angle of Incidence (Degrees)'});
ylabel({'Moment (N*m)'});
legend('VS','Rudder','Location','northwest');
xlim([0,45]);

```



Published with MATLAB® R2014b

Works Cited

- Army, D. o. t. (2009a). Multiservice Helicopter Sling Load: Dual-Point Load Rigging Procedures: Department of the Army.
- Army, D. o. t. (2009b). Multiservice Helicopter Sling Load: Single-Point Load Rigging Procedures: Department of the Army.
- Army UH-72 Flight Limitations (EC-145/BK 117 C-2). (2000).
- Cicolani, L., Cone, A., Theron, J., Robinson, D., Lusardi, J., Tischler, M., . . . Raz, R. (2009). Flight Test and Simulation of a Cargo Container Slung Load in Forward Flight *Journal of the American Helicopter Society*, 54.
- CMCI. (2011). Tricons. Retrieved October 14, 2014, 2014, from <http://www.cmci.com/Tricons.aspx>
- Greenwell, D. I. (2011). Modelling of static aerodynamics of helicopter underslung loads. *The Aeronautical Journal*, 115(1166).
- McCoy, A. (1998). Flight Testing and Real-Time System Identification Analysis of a UH-60A Black Hawk Helicopter With an Instrumented External Sling Load Ames Research Center.
- Nyren, D. (2013). Innovated Concepts for Passively Increasing the Stability of Helicopter Sling Load Payloads: Worcester Polytechnic Institute.

Potter, J., Singhose, W., & Costello, M. (2011). *Reducing Swing of Model Helicopter Sling Load Using Input Shaping*. Paper presented at the 9th IEEE International Conference on Control and Automation (ICCA), Santiago, Chile.

Murray, R. M. (2010). *Optimization-Based Control*: California Institute of Technology