CLOSED-LOOP ATTITUDE CONTROL USING FLUID DYNAMIC VECTORING ON AN AEROSPIKE NOZZLE

by

Nathan M. Erni

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Doran J. Baker
Major Professor

Dr. Stephen A. Whitmore
Committee Member

Dr. Albert Einstein
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2011

Abstract

Closed-Loop Attitude Control Using Fluid Dynamic Vectoring on an Aerospike Nozzle

by

Nathan M. Erni, Master of Science

Utah State University, 2011

Major Professor: Dr. Doran J. Baker
Department: Electrical and Computer Engineering

Attitude control of a prototype satellite bus using fluid mechanical vectoring on an aerospike nozzle is demonstrated. The design achieves thrust vectoring by injecting propellant asymmetrically into the unconstrained aerospike exhaust plume near the nozzle base. The prototype system uses cold-gas thrusters to both spin-up and de-tumble the test article. The system is configured with axially directed annular flows that produce large longitudinal thrusts and smaller secondary lateral-injection flows for side thrusts. Both open and closed-loop attitude control, with and without main aerospike annular flow active, are demonstrated. Proportional, integral, derivative (PID) regulation is used for closed-loop attitude control. When the vectoring ports are operated with no primary plenum flow, very small impulse bits are generated. Based on the results presented in this paper, there exists a significant potential for three-degree of freedom (3-DOF) attitude control without mechanical nozzle gimbals. When extended to 3-DOF, the closed-loop control-law will allow the primary satellite propulsion system to be used for both larger-scale orbit change maneuvers and smaller-scale proximity operation maneuvers with the same system.

(112 pages)

Public Abstract

Closed-Loop Attitude Control Using Fluid Dynamic Vectoring on an Aerospike Nozzle

by

Nathan M. Erni, Master of Science

Utah State University, 2011

Major Professor: Dr. Doran J. Baker
Department: Electrical and Computer Engineering

Position control of a prototype satellite using fluid mechanical vectoring on an aerospike nozzle is demonstrated. The design achieves thrust positioning control by injection propellant asymmetrically into the aerospike nozzle exhaust plum near the nozzle base. The prototype system uses cold-gas aerospike nozzles to both spin-up and de-tumble the test article. The aerospike nozzle is positioned to provide a large amount of axially directed thrust and smaller secondary lateral-injection flows for side thrusts. Both open and closed-loop attitude control, with and without main aerospike annular flow active, are demonstrated. Proportional, integral, derivative (PID) regulation is used for closed-loop attitude control. When the vectoring ports are operated with no primary plenum flow, very small pulses of propellant are used to generate movement. Based on the results presented in this paper, there exists a significant potential for three-degree of freedom (3-DOF) attitude control without mechanical nozzle gimbals. When extended to 3-DOF, the closed-loop control-law will allow the primary satellite propulsion system to be used for both larger-scale orbit change maneuvers and smaller-scale proximity operation maneuvers with the same system.

Acknowledgments

I would like to thank Dr. Doran Baker, Dr. Stephen Whitmore, and Dr. Donald Cripps for dedicating their time and support to further the research of this project along with the Rocky Mountain NASA Space Grant Consortium and NASA's Office of Education, through their EPSCoR funds they contributed to make this project possible. A special thanks go to all the team members I worked with throughout this project, namely Crystal Frazier, Stewart Hansen, Shannon Eilers, and Nathan Madsen. I would also like to thank the machinists for the use of their equipment, machine time, and insights.

Nathan M. Erni

Contents

List of Tables

List of Figures

Chapter 1
Introduction

During the past decade, advances in miniature electronics and exponential growth in computational power have allowed the development of powerful satellites with a very small-scale form factor. Within the next decade these small spacecraft, referred to as textitCube-Sats, will develop the capability to perform a variety of in-space missions that previously could only be performed by very expensive, large-scale satellites. A wide variety of missions from science and exploration missions, telecommunications, to military reconnaissance and surveillance (R&S) would be enabled by a constellation of textitCubeSats precisely positioned to achieve a strategic objective. Launched as a constellation, this distributed swarm offers distinct advantages not achievable by single, larger-scale spacecraft.

For military R&S applications a constellation allows several member elements to fail and still achieve mission objectives. The multiple spacecraft allow for simultaneous spatial and temporal measurements once the target destination is achieved. A large constellation is multiply-redundant and would be nearly impossible to disable with airborne or ground based-weapon systems. Disabling a single member of the constellation still leaves a system with 95-99% functional capability. Most importantly, a highly-dispersed orbiting constellation would enable the United States Air Force (USAF) and the National Reconnaissance Office (NRO) to counter many enemy denial and deception practices. Such a constellation of spacecraft would provide virtually-undeniable critical intelligence for multiple purposes including detecting, locating, characterizing, and tracking: 1) Weapons of mass destruction and their support and production infrastructure; 2) Narcotics and terrorism activities, organizations, and leadership; 3) Potential adversary's air, land, and sea military activities; 4) Air and ground-moving targets and single entities; 5) Advanced weapons systems.

For civilian space missions, providing a capability of approximately 800 m/sec allows the proposed constellation to be launched as a ride-along payload and deployed onto interplanetary trajectories from a standard Geostationary Transfer Orbit (GTO). Accomplishing interplanetary missions as a secondary payload for commercial Geostationary Earth Orbit (GEO) launches offers a potentially game changing technology for National Aeronautics and Space Administration (NASA) science. Once the target destination is achieved, the distributed nature of this interplanetary swarm will allow simultaneous spatial and temporal measurements. A single large spacecraft cannot achieve such measurements.

Because of their small sizes, textitCubeSats must be constructed using the most efficient packaging possible. Thus, the design challenges associated with creating CubeSat-scale propulsion systems are greater than those associated with designing thrusters for conventional spacecraft. Deploying conventional propulsion systems with gimbaled bell-nozzles for attitude control is infeasible in such small form factors. This project researches non-standard propulsion systems designs that allow efficient packaging, and provides high impulse levels for orbital maneuvers, and still allows small impulse bits for orbital rendezvous and precision formation flying.

The aerospike nozzle is a key enabling technology to be investigated here. An aerospike nozzle is an altitude compensating nozzle that maintains aerodynamic efficiency across various altitudes. Because of the unconstrained external plume, it is possible to achieve thrust vectoring fluid-dynamically by injecting propellant asymmetrically at points along the nozzle contour. If this vectoring potential can be harnessed and incorporated into a closed-loop thrust-vectoring scheme, there exists a significant potential for three-degrees of freedom (3DoF) attitude control without mechanical nozzle gimbals. The result is a significant reduction of system complexity and a significant reduction of the overall propulsion system weight. Finally, when the vectoring ports are operated with no primary plenum flow and in a pulse-width mode, very small impulse bits can be generated and offers the potential for the system to also be used for precise proximity operations. This small-impulse option is not possible with a conventional nozzle.

Despite many potential advantages over conventional nozzle designs, the aerospike nozzle has never been deployed on an operational vehicle. This deployment gap is due to (justifiably or not) the perceived low overall technology readiness level (TRL) of the aerospike configuration. This low TRL is especially relevant with regard to very small thrusters on a scale useful for textitCubeSat applications. A single-axis prototype has been developed to demonstrate the proposed technology. Results of this demonstration project will be presented herein. A primary goal of the demonstration project is the rapid upgrade of the technology readiness level of the system from the current level of approximately 2-3 to a level of approximately 4. Further upgrading the system TRL to 6 by follow-on development efforts would allow an aerospike-based propulsion system with closed-loop vectoring control to be seriously considered as a viable alternative for future space deployments.

This thesis proposes the demonstration of a novel, compact propulsion system, scaled for textitCubeSat-sized spacecraft. Once developed, the system will provide the capability to precisely position textitCubeSats to form a large constellation whose members work collectively to accomplish a meaningful tactical objective. The distributed nature of this swarm offers distinct advantages not achievable by a single, large-scale spacecraft. Because of their small sizes, textitCubeSats must be constructed using the most efficient packaging possible. Thus the design challenges associated with creating textitCubeSat-scale propulsion systems are greater than those associated with designing thrusters for conventional spacecraft. Deploying conventional propulsion systems with gimbaled bell-nozzles for attitude control is not feasible with the necessary small form factors. The design, based upon the aerospike nozzle concept, ameliorates this difficulty. These advantages will be explored in the nest chapter.

Chapter 2
Objectives

The objectives, verification methods, and achievements are listed in Table 2.1 . This research is performed in conjunction with phase one of the *CubeSat*-Scale Propulsion System (CCSPS), which purpose is to minimize NASA's deployment gap by rapidly upgrading the TRL level from the current level 3 to approximately level 4. Tests were performed by a hardware in-the-loop simulation to demonstrate, measure, and assert the control characteristics of an aerospike nozzle. The approach was to use a single aerospike nozzle with two secondary injection ports for angular control.

Table 2.1: Table of objectives with verification methods and achievements.

Objective	Verification Method	Achievement
Open-Loop TVC Secondary Flow Only	The satellite starts at a resting state while a single secondary injection port is turned on causing the satellite to spin up to a desired angular rate of 2 rad/s (114.6 deg/s). The test is conducted without the presence of annular flow.	This test set will show the ability of rotational ramping using an aerospike nozzle showing the ability to de-tumble a satellite without annular flow.
Open-Loop TVC Annular Flow and Secondary	The satellite starts at a resting state while a single secondary injection port is turned on causing the satellite to spin up to a desired angular rate of 2 rad/s (114.6 deg/s). The test is conducted with the presence of annular flow.	This test set will show the ability of rotational ramping using an aerospike nozzle showing the ability to de-tumble a satellite with annular flow.
Closed-Loop TVC Secondary Flow Only	The satellite starts at a resting state while the secondary injection ports are controlled by a PID controller using pulse-width-modulation. The satellite will rotate to a desired angular position. The test is conducted without the presence of annular flow.	This test set will show the ability of closed-loop control using an aerospike nozzle used for positioning or maneuvering a satellite without annular flow.
Closed-Loop TVC Annular Flow and Secondary	The satellite starts at a resting state while the secondary injection ports are controlled by a PID controller using pulse-width-modulation. The satellite will rotate to a desired angular position. The test is conducted with the presence of annular flow.	This test set will show the ability of closed-loop control using an aerospike nozzle used for positioning or maneuvering a satellite with annular flow.

Chapter 3
Background

While the aerospike nozzle has long been known for its altitude compensation capability during endo-atmospheric flight, the aerospike also presents significant advantages for purely in-space applications. Because of its shape, the aerospike nozzle can be constructed with a higher area expansion ratio and more compact form factor than a conventional bell nozzle of the same mass. The higher expansion ratio provides better performance in a space environment; the compact form factor offers a very significant advantage for CubeSat-Scale spacecraft where volume efficiency is a key consideration.

Figures 3.1 and 3.2 [1] compare the aerospike nozzle theory of operation to the conventional bell nozzle typically used for space applications. Here the walls of the conventional bell nozzle constrain the flow and result in an over-expanded plume for launch conditions, and an under-expanded plume for in-space operating conditions. In both cases the performance is significantly lower than optimal. The aerospike nozzle, however, does not constrain the outer boundaries of the flow slipstream. The plume is free to expand or contract depending on the external pressure at the operating conditions. Compared to the bell nozzle, the achieved performance is significantly higher for both launch and in-space operating conditions.

Compared to the 100:1 expansion ratios typically available to bell nozzles, annular aerospike nozzles (of equivalent mass) with expansion ratios exceeding 250:1 can be easily fabricated. This higher expansion ratio increases vacuum specific impulse (Isp) by more than 10%. The increase in Isp results in a 10-18% reduction in the propellant mass, and both factors produce an 8-12% reduction of the total system weight.

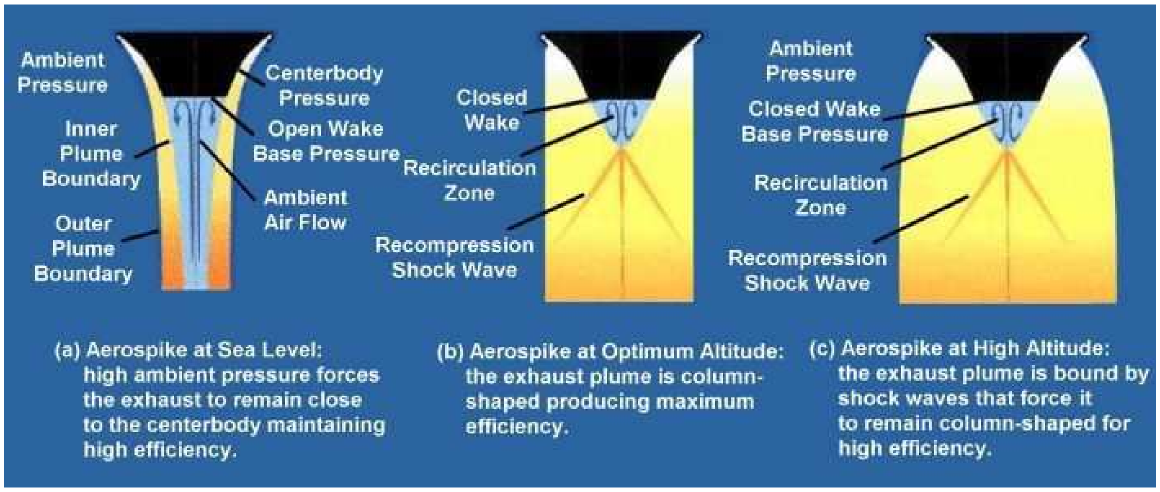

Ambient Pressure
Inner Plume Boundary
Outer Plume Boundary
Centerbody Pressure
Open Wake Base Pressure
Ambient Air Flow
Closed Wake
Recirculation Zone
Recompression Shock Wave
Ambient Pressure
Closed Wake Base Pressure
Recirculation Zone
Recompression Shock Wave
(a) Aerospike at Sea Level: high ambient pressure forces the exhaust to remain close to the centerbody maintaining high efficiency.
(b) Aerospike at Optimum Altitude: the exhaust plume is column-shaped producing maximum efficiency.
(c) Aerospike at High Altitude: the exhaust plume is bound by shock waves that force it to remain column-shaped for high efficiency.

Fig. 3.1: Aerospike operational comparison for (a) over-expanded, (b) optimally expanded, and (c) under-expanded conditions.

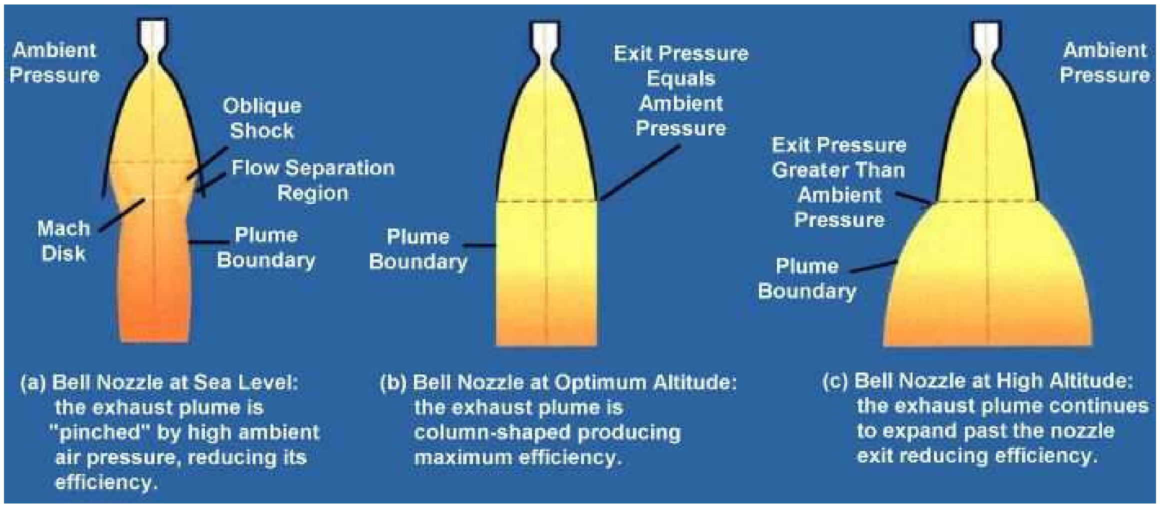

Ambient Pressure
Oblique Shock
Flow Separation Region
Mach Disk
Plume Boundary
Exit Pressure Equals Ambient Pressure
Plume Boundary
Ambient Pressure
Exit Pressure Greater Than Ambient Pressure
Plume Boundary
(a) Bell Nozzle at Sea Level: the exhaust plume is "pinched" by high ambient air pressure, reducing its efficiency.
(b) Bell Nozzle at Optimum Altitude: the exhaust plume is column-shaped producing maximum efficiency.
(c) Bell Nozzle at High Altitude: the exhaust plume continues to expand past the nozzle exit reducing efficiency.

Fig. 3.2: Conventional bell nozzle operational comparison for (a) over-expanded, (b) optimally expanded, and (c) under-expanded conditions.

Chapter 4
Literature Review

4.1 Prior Art

Testing on aerospike nozzles began in the 1950s in preparation for the Saturn V rocket [2], but after extensive research and test series, Rocketdyne concluded that aeropsike nozzles had less or equal thrust vectoring capabilities than a conventional nozzle. The series of tests performed were based on liquid injection only. The series indicated that an annular throat aerospike nozzle is at least comparable to, if not better than the 80% length bell nozzle at a given design pressure ratio [1, 3]; however, the nozzle had less or equal thrust vectoring capability. Research continued into the 1970s as an aerospike nozzle was under consideration for the Space Shuttle's main engine [4, 5], but again was not selected. Efforts on aerospike research declined until the 1990s when NASA proposed a linear aerospike as a propulsion system for the X-33 and the VentureStar [6]. These programs were later canceled.

More recently, analytical research has been conducted worldwide by several institutions for the evelopment and performance analysis of Thrust Vector Control (TVC) [7, 8], differential throttling [9], clustering performance [10–13], slipstream effects [14–16], base bleed injection [17–19], optimal contours [20], and acoustics [21]. Hardware experiments have been conducted by Arizona State University [22], University of Washington [23], and California Polytechnic State University [24]. California State University in conjunction with Garvey Spacecraft Corporation have conducted several experiments with liquid, clustered aerospike engines along with launching several sounding rockets [25, 26].

4.2 Recent Cold-Flow Vectoring Experiments

Utah State University recently performed analytical and experimental evaluations on aerodynamic thrust vectoring on aerospike nozzles using secondary injection [27]. These experiments included sizing an aerospike nozzle for slightly above optimal expansion for the testing altitude. This design allowed compression waves to impinge past the end of the nozzle to create a pressure distribution along the spike similar to vacuum conditions. For these tests carbon dioxide (CO_2) was used as the working fluid. Different spike configurations were designed and tested for an annular mass flow rate of approximately 1 kg/s with a secondary injection flow rate between 2 to 3% of the annular mass flow. Figure 4.1 shows the designed spikes truncated to 57% of the theoretical spike length. This level of truncation has minimal effect on the overall nozzle performance [28].

The spike nozzle housed the secondary injection ports. Tests were performed with secondary injection ports located at 20, 80, and 90% of the truncated spike. Figure 4.2 shows experimental results from secondary injection tests performed with and without main plenum flow. The key feature to take away from this figure is that the generated forces with secondary flow active are greater than those generated with secondary injection only. Test results presented in Figure 4.3, indicate the optimal injection site is located at approximately 90% of the truncated spike. This result implies that the same control impulse can be achieved for significantly less propellant when the main plenum flow is active.

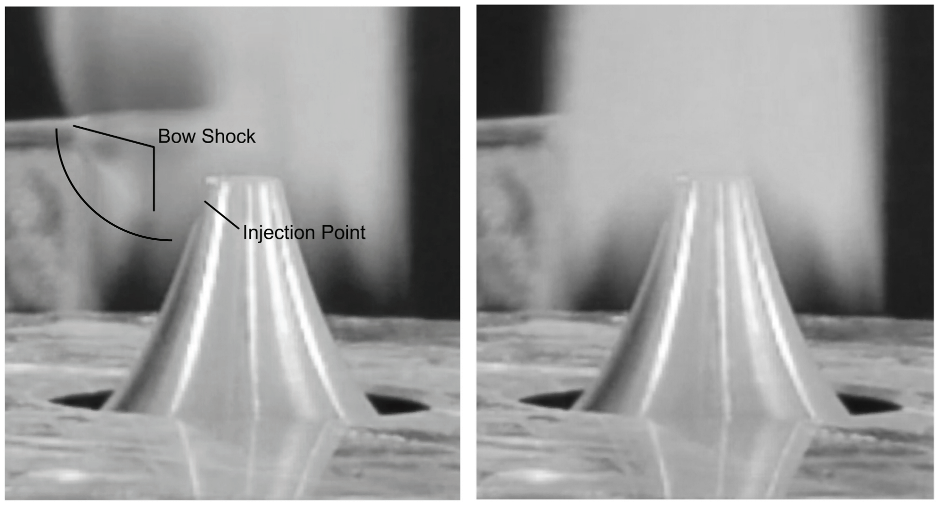


Fig. 4.1: Aerospike cold flow test with 4.4 mm diameter orifice located at 90% of the length of the truncated aerospike. a) Thrust vectoring active, showing bow shock. b) Thrust vectoring inactive.

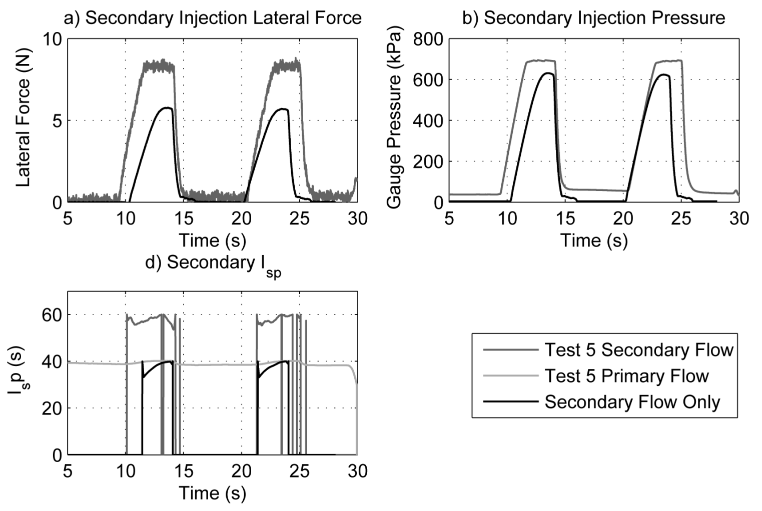


Fig. 4.2: Side force and secondary injection for 90% injection point for both primary flow active and secondary flow only configurations.

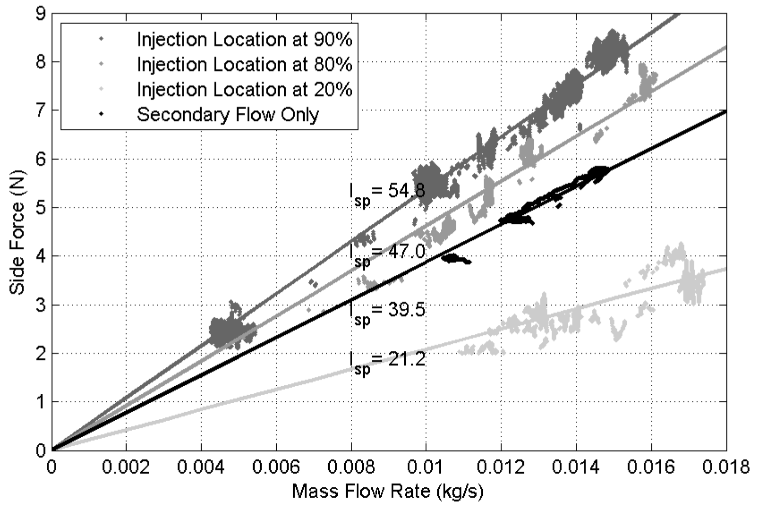


Fig. 4.3: Cold flow secondary injection results and regressed specific impulses for various hole locations.

Table 4.1 summarizes the test results including side-force amplification factors and achieved specific impulses. Here the side force amplification is defined as the ratio of side force with a main axial flow to the side force generated by the secondary injection without the primary flow. Results show a higher amplification factor is produced as the injection location is moved closer towards the end of the truncated region.

Table 4.1: Cold flow test specific impulse results.

Test Series	$I_{sp}(s)$	$I_{sp}(s)$ Uncertainty (s, 95 %)	Amplification Factor
Injection Location at 90 %	54.8	1.9	1.39
Injection Location at 80 %	47.0	1.9	1.19
Injection Location at 20 %	21.2	1.7	0.54
Secondary Flow Only	39.5	1.8	

Chapter 5
CubeSat Background

As described in the background section, the primary focus of this research is to demonstrate the feasibility for secondary injection thrust vectoring of a *CubeSat* using an aerospike nozzle. A *CubeSat* is a miniature satellite with a predefined external form factor. The size is usually described in terms of volume using 1U as a 10 cm cube. The sizing for a typical *CubeSat* is shown in Figure 5.1.

The current method for deploying *CubeSats* is the Poly-PicoSatellite Orbital Deployer (P-POD) developed by California Polytechnic State University [29]. Each P-POD is mounted as a secondary payload on the launch vehicle and carries several 1U *CubeSats*. Figure 5.2 shows a typical P-POD configuration. Once the desired orbit is reached, the P-POD releases the *CubeSats*. With the current state-of-the art size restrictions and hazards to the primary payload restrict *CubeSats* from having integrated propulsion systems. Thus *CubeSat* orbits are determined by the launch insertion and P-POD deployment velocities. Because of their small sizes, textitCubeSats must be constructed using the most efficient packaging possible. Deploying conventional propulsion systems with gimbaled bell-nozzles for attitude control is infeasible in such small form factors. The proposed design, based on the aerospike nozzle concept, overcomes this difficulty. Because of its shape, the aerospike nozzle can be constructed with a higher area expansion ratio and more compact form factor than a conventional bell nozzle of the same mass. The higher expansion ratio provides better performance in a space environment; the compact form factor offers a very significant advantage for textitCubeSat-Scale spacecraft where volume efficiency is a key consideration.

Figure 5.3 shows a proposed compact *CubeSat* scale propulsion system (CCSPS). Figure 5.4 shows the CCSPS integrated with the satellite bus and the P-POD dispensing system. The aerospike nozzle is housed in a fixed 10 cm diameter plenum containing hy-

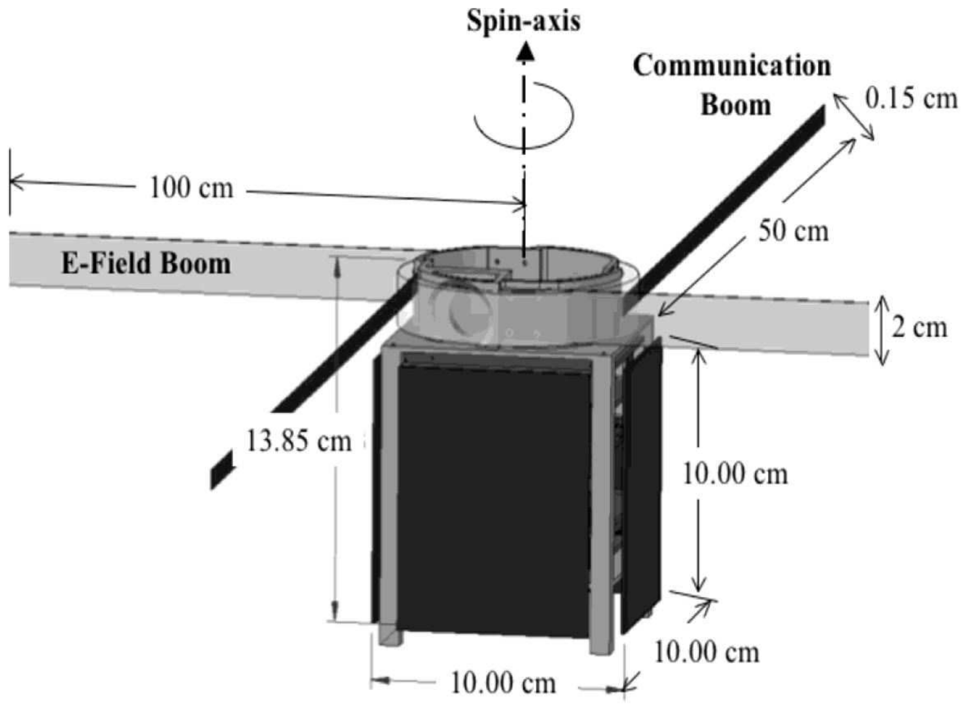


Fig. 5.1: Typical 1U *CubeSat* configuration.

brid propellant. The proposed design combines three emerging technologies 1) aerospike or external plume nozzles, 2) hybrid rocket systems, and 3) direct digital manufacturing to build a unique propulsion unit that can potentially enable the constellation described in the previous paragraphs. Because the system would be flown in an inert condition and would carry no ordnance, multiple payloads can be piggy backed together with no overall increase to mission risk. Two of these aerospike nozzles would be placed on the base of a 2U *CubeSat* module as shown in the aft view of Figure 5.4. The potential *CubeSat* propulsion system would be self-contained in a 2U *CubeSat*, housing all of the propellant, sensors, controllers, and valves. This design allows for any existing *CubeSat* to be attached allowing for orbital maneuverings. Approximately 50 grams of thermo-plastic hybrid fuel and 0.7 kg of liquid nitrous oxide as the oxidizer, would enable a 1-kg spacecraft to achieve a Delta V of 800 m/sec. This velocity change is sufficient to achieve escape velocity from a geosynchronous transfer orbit (GTO).

The experimental prototype demonstrated in this research approximates the CCSPS features by using a single-axis scaled aerospike nozzle for vectoring in the yaw axis.

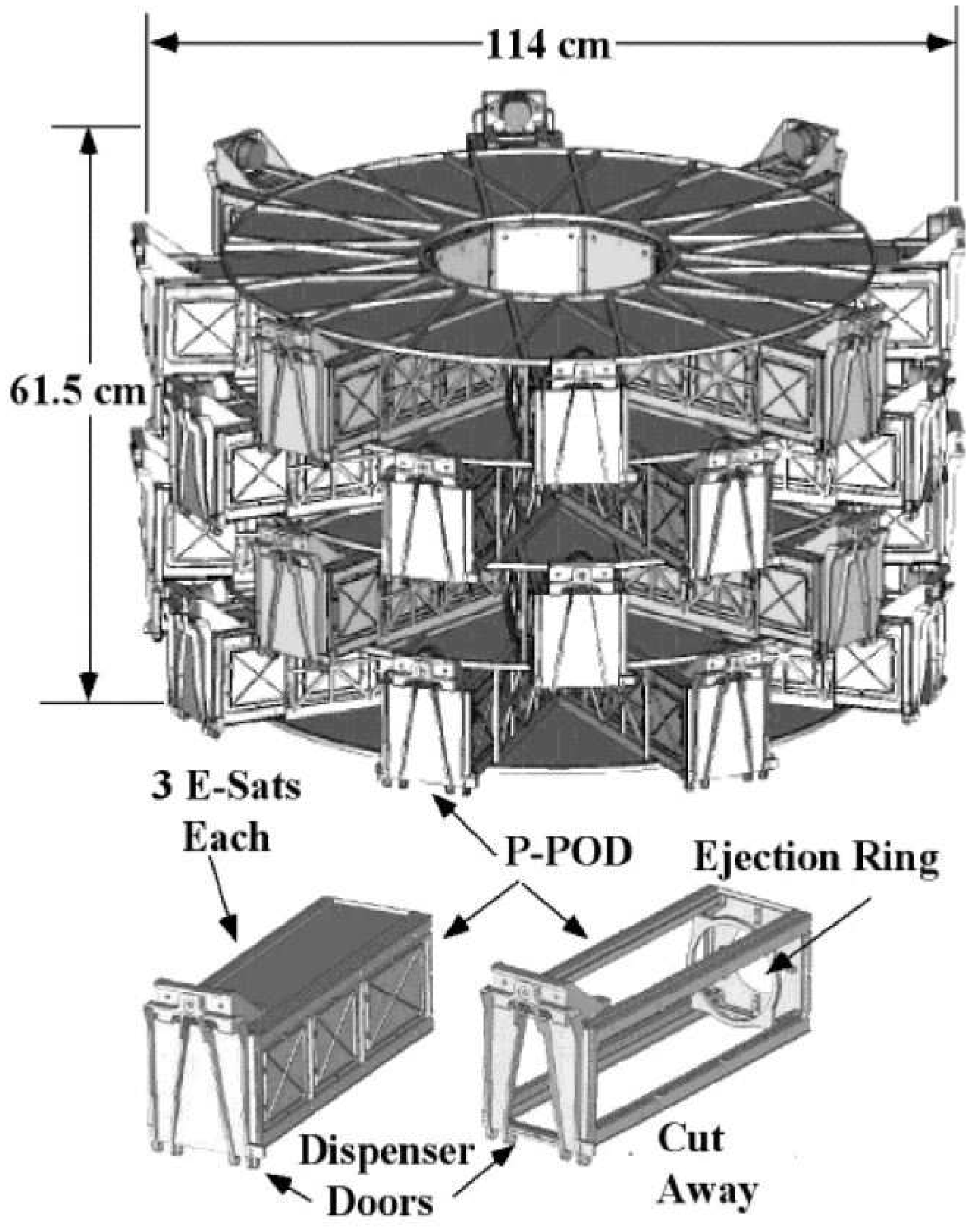


Fig. 5.2: Proposed compact textitCubeSat scaled propulsion system (CCSPS).

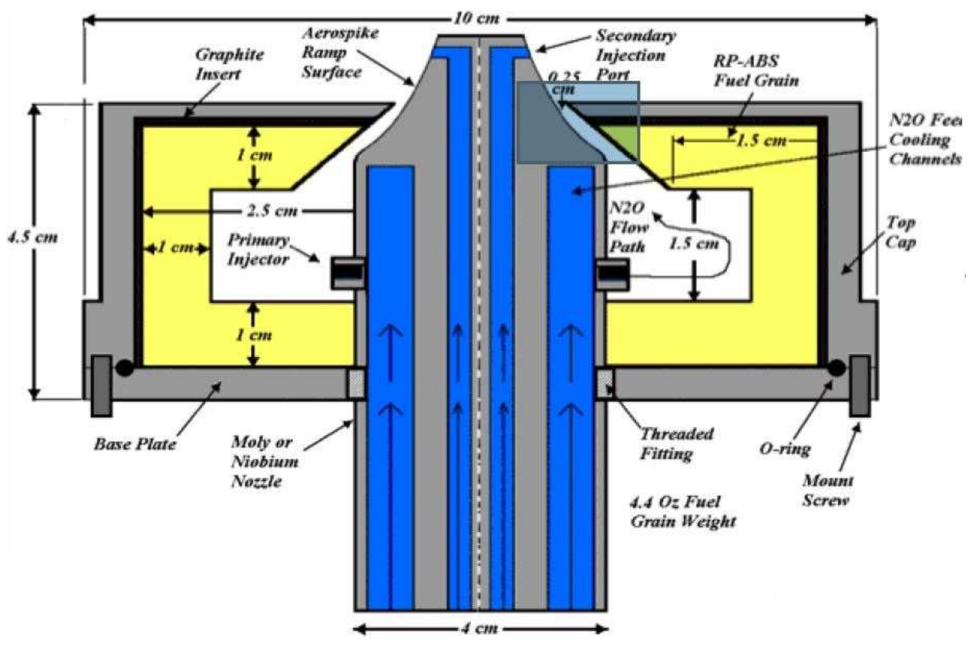


Fig. 5.3: Regeneratively cooled, low profile aerospike/hybrid motor.

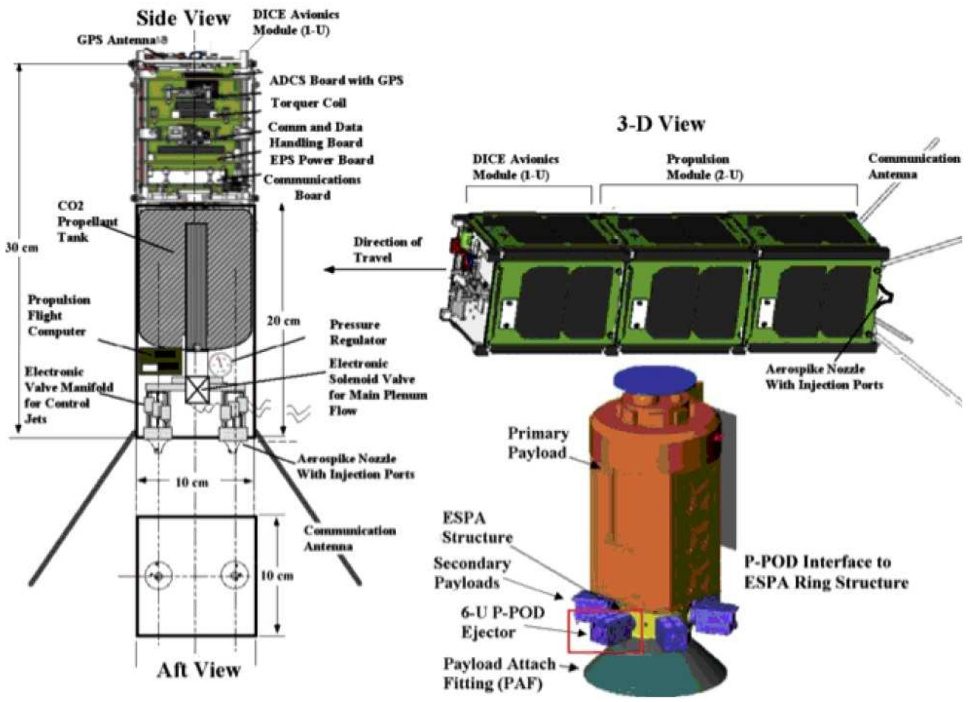

Side View
DICE Avionics Module (1-U)
GPS Antenna
ADCS Board with GPS
Torquer Coil
Comm and Data Handling Board
EPS Power Board
Communications Board
CO2 Propellant Tank
30 cm
20 cm
Propulsion Flight Computer
Pressure Regulator
Electronic Solenoid Valve for Main Plenum Flow
Electronic Valve Manifold for Control Jets
Aerospike Nozzle With Injection Ports
10 cm
Communication Antenna
10 cm
Aft View
3-D View
DICE Avionics Module (1-U)
Propulsion Module (2-U)
Communication Antenna
Direction of Travel
Aerospike Nozzle With Injection Ports
Primary Payload
ESPA Structure
Secondary Payloads
P-POD Interface to ESPA Ring Structure
6-U P-POD Ejector
Payload Attach Fitting (PAF)

Fig. 5.4: Aerospike configuration (a) attached to a 1U *CubeSat* and (b) inside a P-POD dispensing system.

Chapter 6

Methodology

This chapter describes all design aspects and tests for this project including the aerospike design, prototype satellite design, housing cage design, instrumentation, control methodology, and experimental tests.

6.1 Aerospike Design

The contour of the spike was calculated using the methods of Lee and Thompson [30]. Figure 6.1 shows the aerospike geometries. A preliminary trade-study was performed to estimate the nozzle parameters and operating systems that were most advantageous to accomplishing the proposed research objectives allowing for a sufficient mass flow rate, aerospike diameter, and overall propellant. Figure 6.2 shows the an aerospike sizing trade study including a detailed description comparing operating pressure, mass flow rate, produced thrust, plug diameter, outer throat diameter, and expansion ratio. This plot was used to calculate the aerospike's size for the given operating pressure of 1.315 kPa and mass flow rate of around 0.1 kg/s used. Table 6.1 summarizes the resulting design features.

Figure 6.3 shows the design features of the prototype test article. The system was sized for an annular flow rate of 0.1 kg/s with two secondary injection ports each with an approximate flow rate of about 0.002 kg/s. The plenum volume was sized to allow CO_2 gas to equalize throughout the chamber to maintain a symmetrical flow along the aerospike.

Figure 6.4 shows the aerospike configuration without the outer chamber attached to the base plate of the test apparatus. The annular flow port blows into a flat dispersing plate causing the flow to be uniform within the plenum.

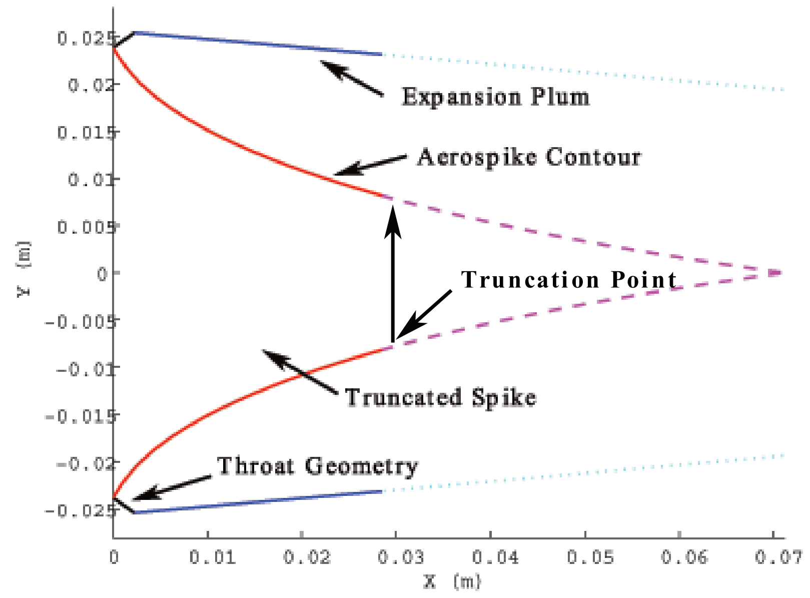


Fig. 6.1: Aerospike nozzle contour and dimensions.

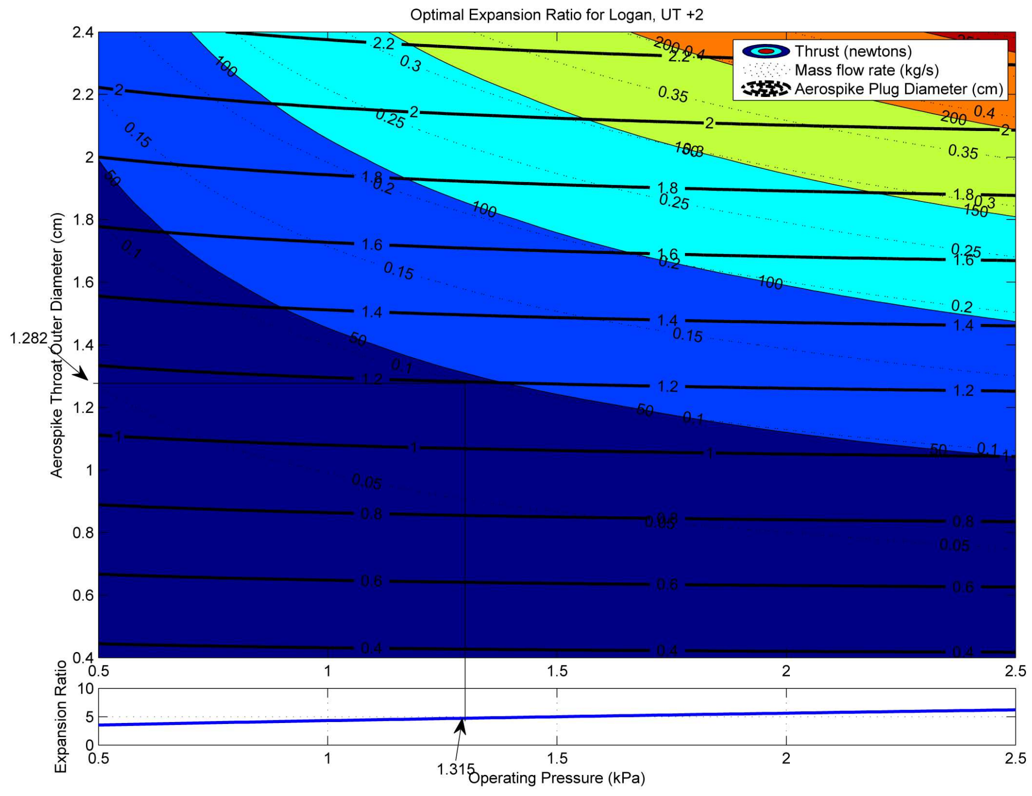


Fig. 6.2: Aerospike sizing trade study.

Table 6.1: Aerospike parameters.

Aerospike Parameter	Value
Operating Pressure	1315 kPa
Temperature	300 K
Radius Cowl	0.006415 m
Pressure Ambient	86.12625 kPa
Expansion Ratio	4.75
Gamma	1.28
MW	44.01
Sample Points	1500
Truncation	0.6

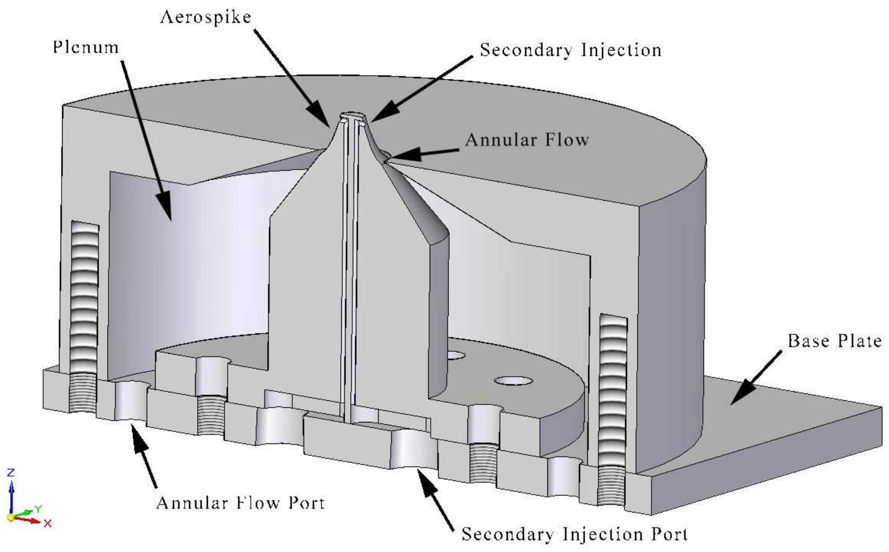


Fig. 6.3: CAD model of the prototype aerospike configuration.

6.2 Prototype Satellite Design

Figure 6.5 shows the prototype apparatus designed to approximate a scaled-up *CubeSat*. Figure 6.6 shows the yaw axis of rotation. This scale was required to economically house the non-flight weight propellant tanks and valves used to control the system. The 1'x1'x2' structure is supported by two aluminum base plates with aluminum rods connecting each corner. A square aluminum tubing placed in the center of the satellite supports four 24 ounce CO_2 tanks and internally houses two 14.8 VDC Li-Ion battery packs. The top of the tubing is bolted to a sheet of $\frac{1}{4}$" polycarbonate locking the CO_2 tanks in a downward position for the

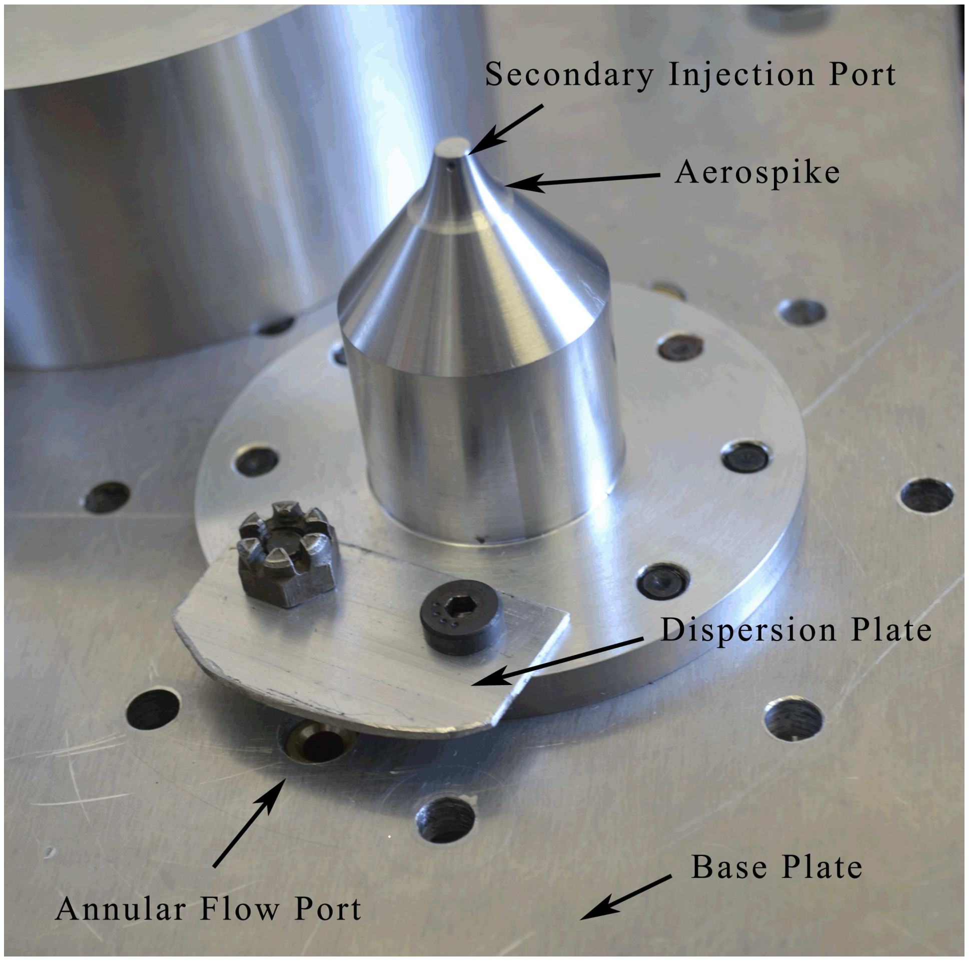


Fig. 6.4: View of the machined aerospike with the dispersion plate for annular flow.

use of liquid propellant. All instrumentation devices are housed on the polycarbonate sheet except for the inertial measurement unit and one pressure and temperature sensor housed on the bottom of the satellite. Included in the instrumentation are DC-DC regulators, pressure and temperature sensors, pressure regulator, annular and secondary flow valves, Load cell, Gumsitx for control code and communication, and a DAQ module for sensory input. Table 6.2 lists the parts with a further description of each major part. Other material not listed includes aluminum plumbing, wiring, and connectors. All communication to and from the satellite is transferred through a wireless link eliminating the need for a slip ring or excess cables.

Figure 6.7 shows the piping and instrumentation (P&ID) of the propellant flow system

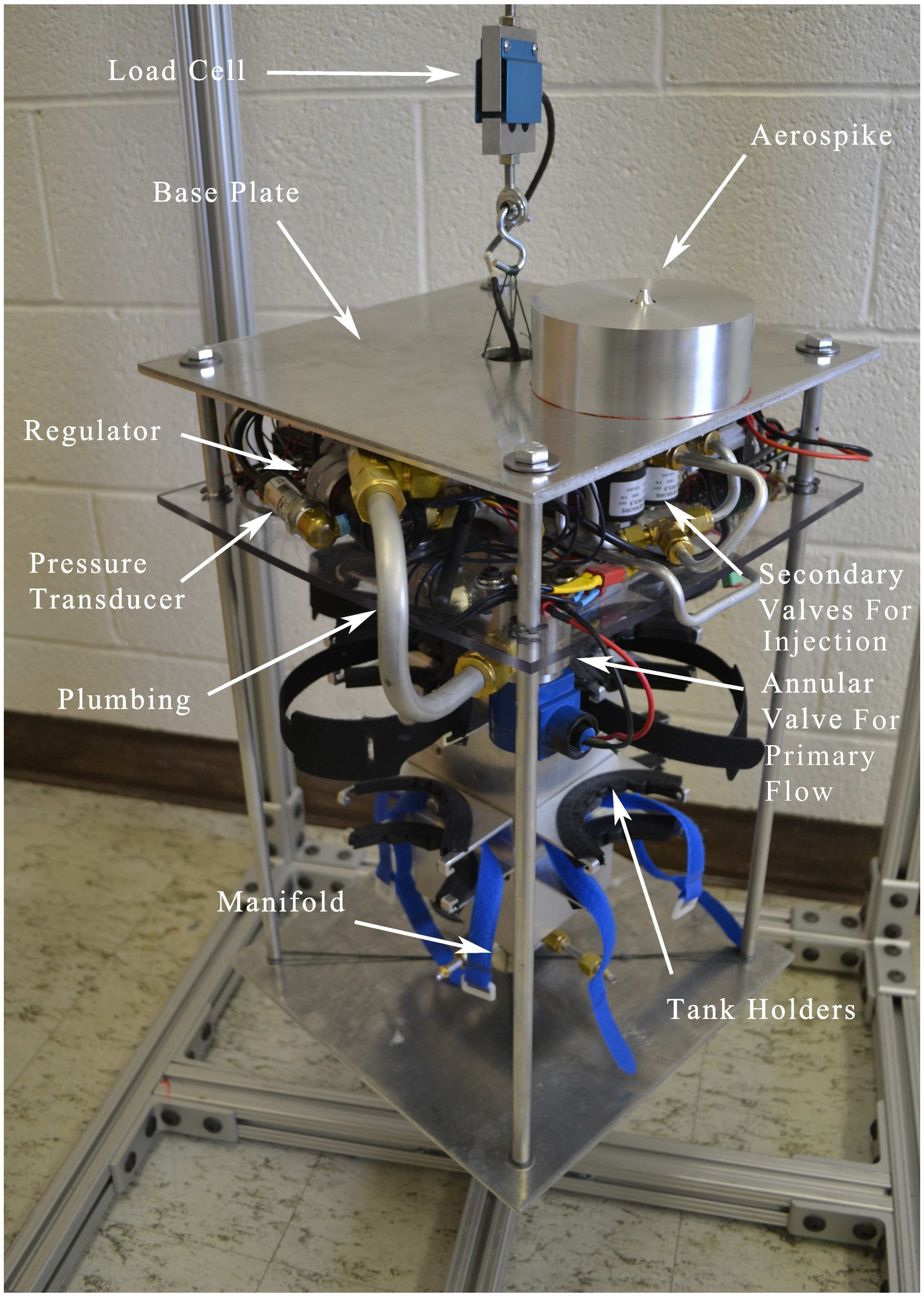


Fig. 6.5: Prototype satellite housing design and configuration.

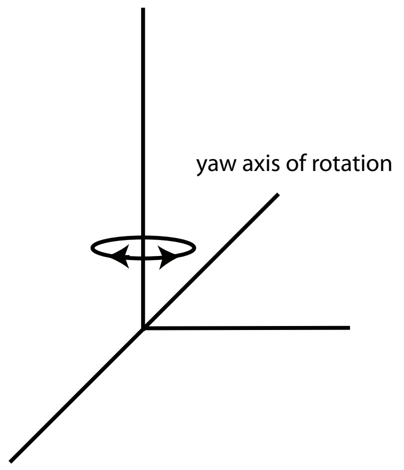


Fig. 6.6: Prototype satellite axis of rotation.

Table 6.2: Aerospike-controlled satellite system avionics parts list.

Satellite Parts	Description
Inertial Measurement Unit (IMU)	Micro-Strain's 3DM-GX3® -35
Customized Polymer Li-Ion Battery	14.8 VDC 6400 mAh (quantity of 2)
DC-DC Regulator Module	12 VDC used to power valves and NI module
DC-DC Regulator Module	3-5-12 VDC Used to power instrumentation
Pressure/Temperature Sensor	From 0 to 500 psiG and -40 to 125 C
Pressure/Temperature Sensor	From 0 to 1500 psiG and -40 to 125 C
Pressure Regulator	Power Tank Pro Series regulator
Annular Flow Valve	GC Valves H401GF15Z1BF5
Secondary Flow Valves	Gems Sensors cryogenic valve B2011-LCO2
Gumstix Overo Fire COM	OMAP 3530 Processor and wireless com
Tobi Expansion Board	Expansion board for the Gumstix processor
NI WLS-9205	Wireless voltage input for all sensor input

for the prototype satellite. Caps of four CO_2 tanks were disassembled and machined to increase the inner diameter to achieve the required mass flow rates for the aerospike. The tanks are plumed to converge into a manifold and up to a system regulator dropping the pressure to around 190 psiG. After pressure regulation the tubing is split three ways to an annular flow valve and two secondary injection valves that are directly connected to the aerospike. Pulsing these valves control rotational movement of the satellite.

6.3 Housing Design

Figure 6.8 shows the housing cage that was designed to suspend the satellite while

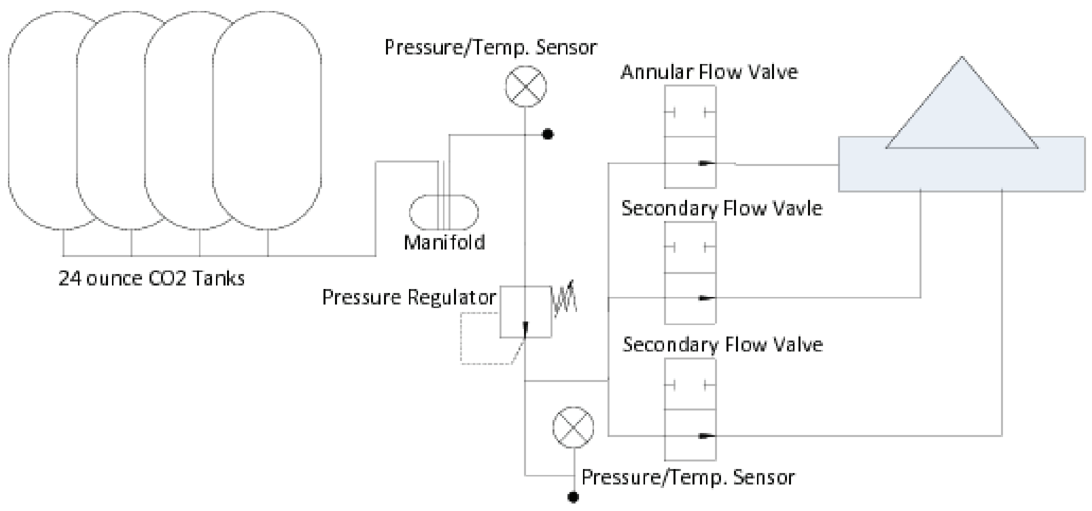


Fig. 6.7: Piping and instrumentation (P&ID) of the propellant flow system for the satellite system.

keeping all coordinates fixed except for yaw. The outer 2'x2'x4' frame was designed with 1" aluminum T-slotted framing with top and bottom supports and leveling feet. The housing cage encompasses the satellite leaving a gap of at least 6" on all sides of the satellite for maneuvering. Between the top of the satellite and the test cage a gap of 15" was necessary to allow undisturbed annular flow for main plenum flow testing. A professional speedbag swivel from Everlast was attached to the top of the housing cage with a piano wire strung to hold the weight of the satellite while allowing it to rotate freely. This design allows the pitch and roll coordinates to be locked while mitigating the friction about the yaw axis to better simulate space conditions.

6.4 Instrumentation

The satellite measures rotational angles, pressure, temperature, and various forces. Forces are measured at the manifold, pressure regulator, and plenum. An National Instruments wireless 9205 data acquisition module is used to transmit measurements to a LabVIEW GUI located on an off-satellite computer. The LabVIEW GUI is used for monitoring the system as well as for pre-check tests. An onboard computer, Overo Fire Gumstix,

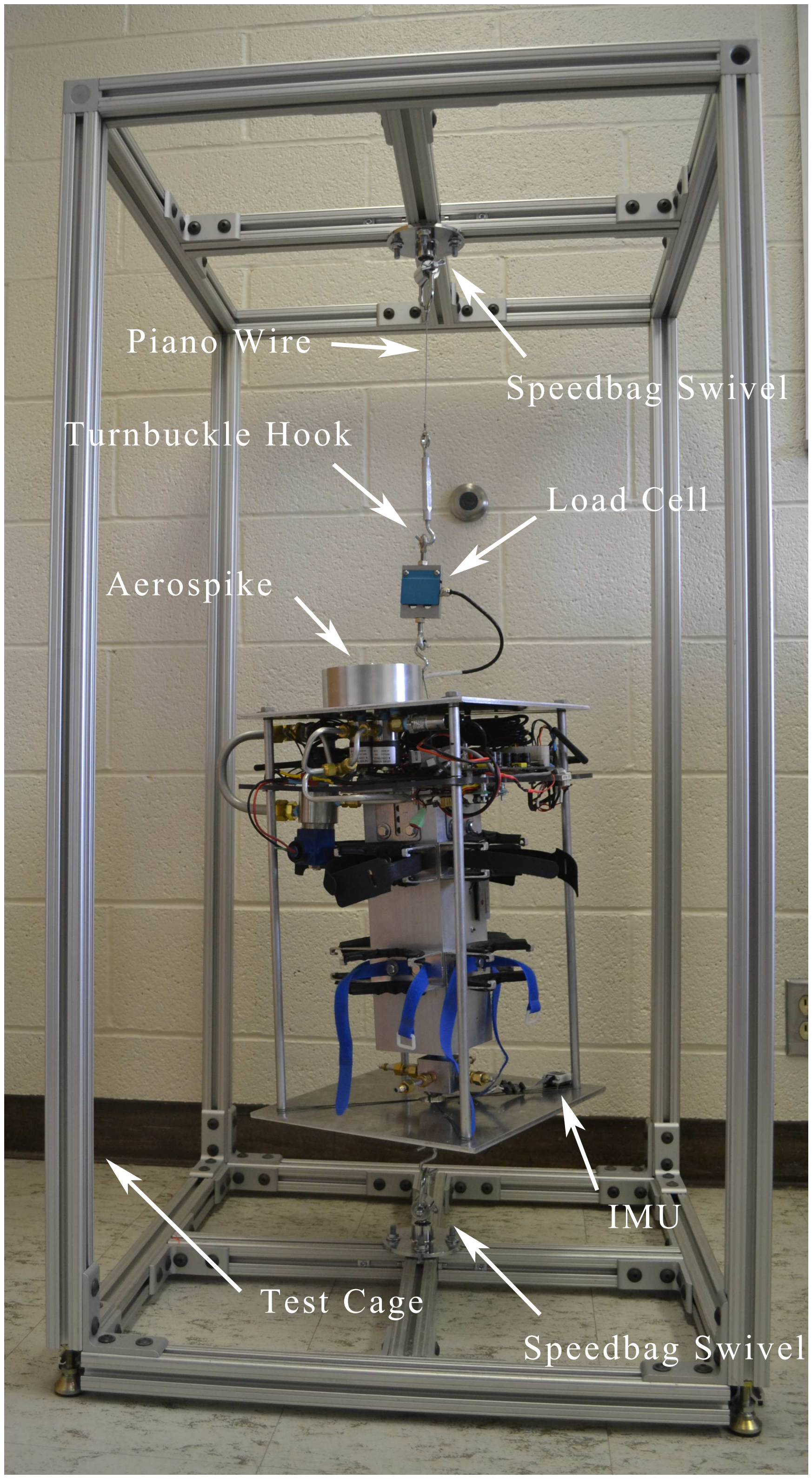


Fig. 6.8: Satellite test stand.

holds the control laws and is directly interfaced to an inertial measurement unit and Gems Sensors cryogenic valves to provide a fast response time. Table 6.2 lists the parts with a further description of each major part.

The primary onboard navigation instrument is a miniature inertial measurement unit (IMU) built by Micro-Strain®, Inc [31]. The IMU features a high-performance attitude heading reference system that includes embedded tri-axial accelerometers, axis rate-gyros, axis magnetometers, and a temperature sensor. The form factor and weight are very small, and this device is mounted on the inner platform of the vehicle without significantly affecting the weight and inertia of the platform. The IMU sensor data is blended internally running a sensor fusion algorithm to provide filtered data. User-selectable output parameters include Euler angles, direction cosine matrix components, acceleration vector components, 3-axis angular rates, and 3-axis magnetic field components. A interface circuit board was designed for the communication link between the IMU and Gumstix along with holding relays to actuate control valves. Figure 6.9 shows the block diagram for this communication. There is a two-way communication link between the Gumstix and the IMU to ensure correct data packets are being sent by comparing packet headings and checksums. Figure 6.10 shows the schematic diagram of the board, and Figure 6.11 shows the layout with the circuitry implemented.

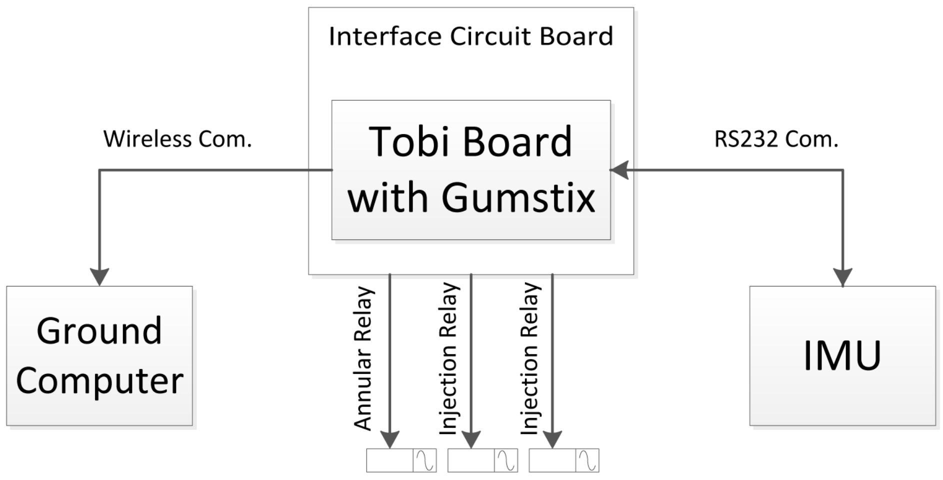


Fig. 6.9: Block diagram of the communication link between the Gumstix, IMU, ground station computer, and relay blocks.

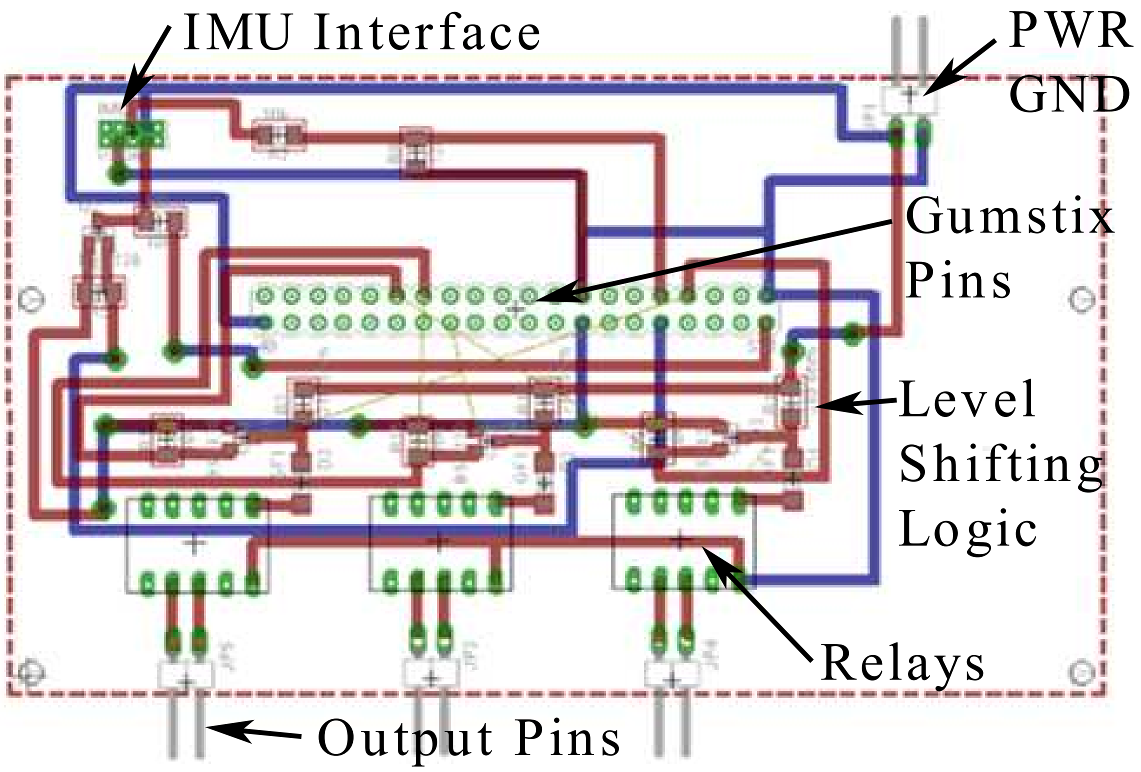


Fig. 6.10: Interface circuit design for establishing a communication link between the IMU and the gumstix along with relay outputs for solenoid valves.

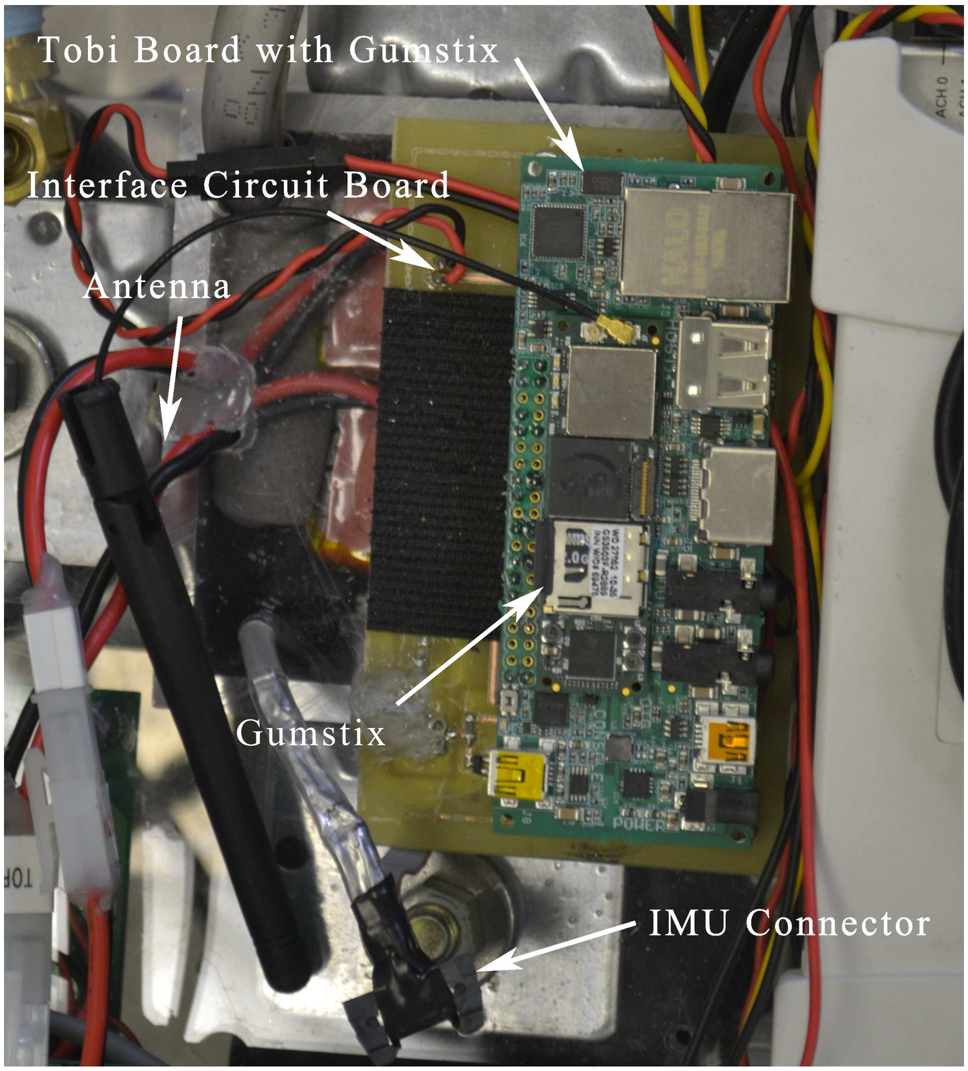


Fig. 6.11: Interface circuit board with gumstix, tobi expansion board, wifi antenna, and IMU connector.

Chapter 7

Thrust Vectoring Tests

Both open and closed-loop thrust vectoring tests were performed. The open-loop tests were used to quantify the delivered side force and moment levels and to workout system bugs. Following the initial open-loop tests, rotational inertias of the system were measured, and a closed-loop Proportional, Integral, Derivative (PID) control law was developed and coded on the Gumstix avionics computer. Follow-on closed-loop control tests were used to adjust control gains.

7.1 Open-Loop Thrust Vectoring Tests

Initial tests consisted of placing the satellite at a resting state and turning on a single secondary injection port causing the satellite to spin up to a desired angular rate of 2 rad/s (114.6 deg/s). Annular flow was included in the second test to compare the achieved amplification. These initial two test sets demonstrated the potential of aerospike vectoring to de-tumble artificial satellites. The third set of tests were used to ascertain a desired angle for satellite maneuvering. These tests were conducted with and without the presence of annular flow. Each ensemble of tests included at least eight tests to allow statistical verification of system performance and repeatability.

7.1.1 Open-Loop Tests with Secondary Injection Only

Each secondary injection test started at a fixed angular position with a rotational rate of 0 rad/s and ramps up until an angular measurement of 2 rad/s (114.6 deg/s) is read by the IMU. Once the desired rotational rate is reached, the test automatically turns off all valves and the data are saved in a packaged data set. A script is run to unpack the data into a format that can be imported into MatLab. Figure 7.1 shows the angular rate of the satellite compared with the time duration of each test. The secondary injection port produced a mean force of 4.19 newtons.

The pressure and temperature in the plenum are at ambient conditions as the annular flow is not being used during these tests. All other pressures and temperatures vary slightly between tests but have no significant effect on the angular rate of the satellite. The time required to achieve 2 rad/s varied by 3.2% between the eight conducted tests.

7.1.2 Open-Loop Tests Annular Flow with Secondary Injection

Each annular flow test was conducted like the secondary injection test with the exception of the usage annular flow. As the test was initiated, the annular flow was turned on while employing secondary injection until a rate of 2 rad/s was read by the IMU. Figure 7.2 displays the angular rate time history for each of the tests performed. With the annular flow active, the test results were moderately noisy. The filtered slope of each test was averaged to give an ensemble mean response curve. Figure 7.3 shows this result. The mean time to achieve a 2 rad/s rotational rate is approximately 15 seconds. This time difference was due to the swivel friction and off-axis swaying of the satellite produced by annular flow.

During this test series the main plenum pressure and temperature varied considerably from test to test. No correlation between pressure vs. temperature was observed. Figure 7.4 shows this result. The observed time differences were primarily due to the swivel friction and off-axis swaying of the satellite. The aerospike is located off the center axis of rotation caused a swaying motion during the tests. Figure 7.5 shows this off-axis motion.

7.2 Closed-Loop Thrust Vectoring Tests

Closed-loop thrust vectoring tests consist of placing the satellite at a resting state and controlling the secondary injection ports to move the satellite to a desired angular position. Tests conducted without annular flow run for a 80 second duration whereas the tests conducted with annular flow run for 35 seconds due to propellant constraints. These tests demonstrate the ability to vector the satellite in any direction with or without the presence of annular flow. Before any tests could occur it is important to understand the

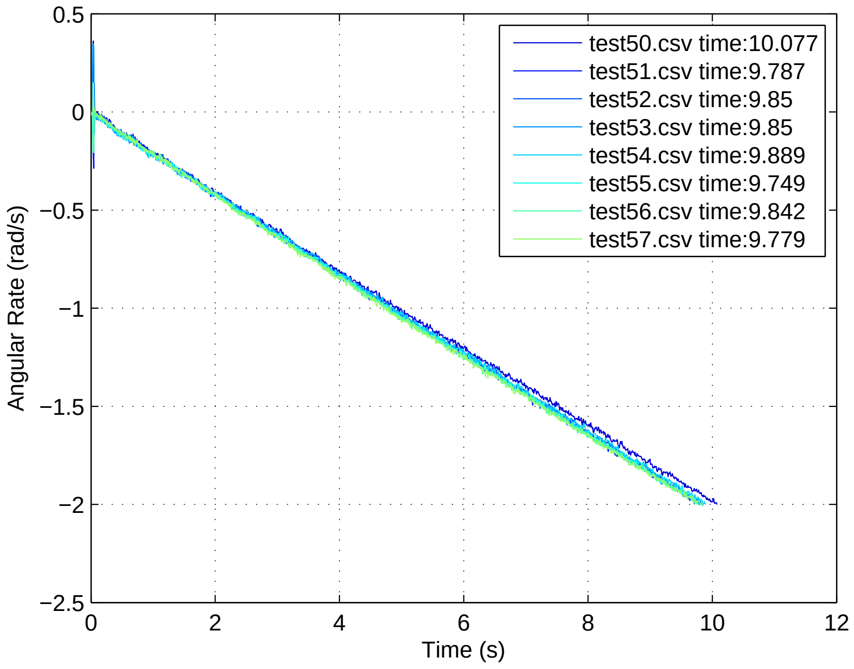


Fig. 7.1: Spin-up tests conducted using only secondary injection for satellite acceleration from 0 rad/s to 2 rad/s.

system dynamics in order to design good control laws for the system being controlled. An analysis of the system is provided in the below sections.

7.2.1 Moment of Inertial Measurements

It is essential to determine the natural frequency and damping ratio of the satellite before control theory is applied to the system. These constants were found by using a strong extension spring as a torsional spring. Figure 7.6 shows the 48.5 lb. satellite being suspended by the extension spring and rotated by weights on a pulley. The first test was used to calculate the spring torsion constant. Eight tests were conducted each placing a series of 10 to 100 grams of weight onto the pulley system while measuring the displacing of satellite from its resting position. By the displacement the spring torsion coefficient was found by using Equation (7.1) where τ = torque, θ = angle, r = radius, and F = force.

$$K = -\tau/\theta, \ \ where \ \ \tau = r * F \tag{7.1}$$

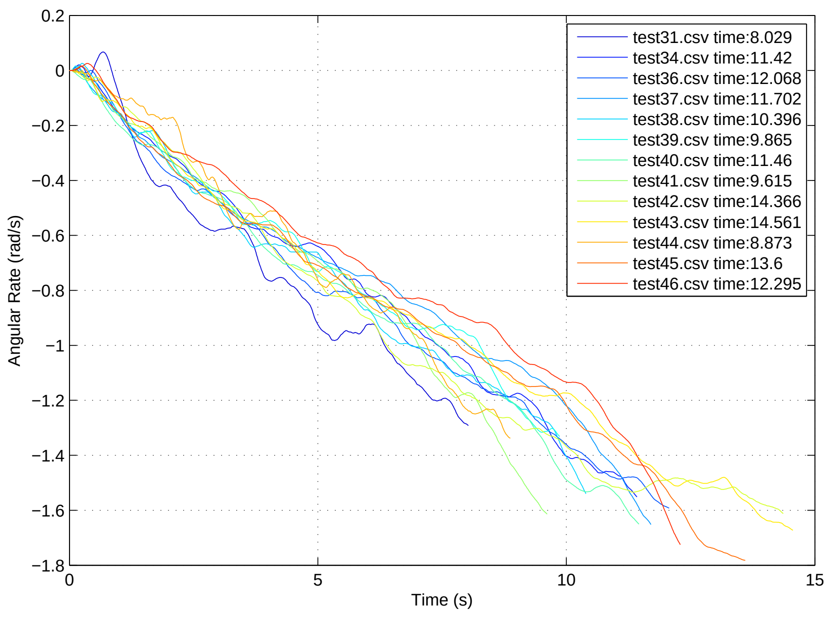


Fig. 7.2: Spin-up tests with annular flow active and secondary injection accelerating from 0 rad/s to 2 rad/s.

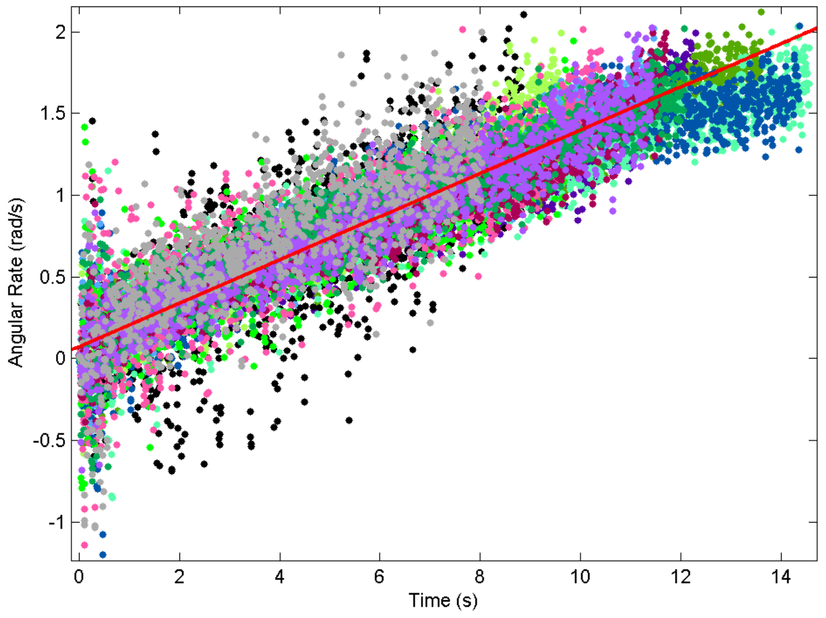


Fig. 7.3: Angular rates for each data set with an averaged angular rate of 2 rad/s.

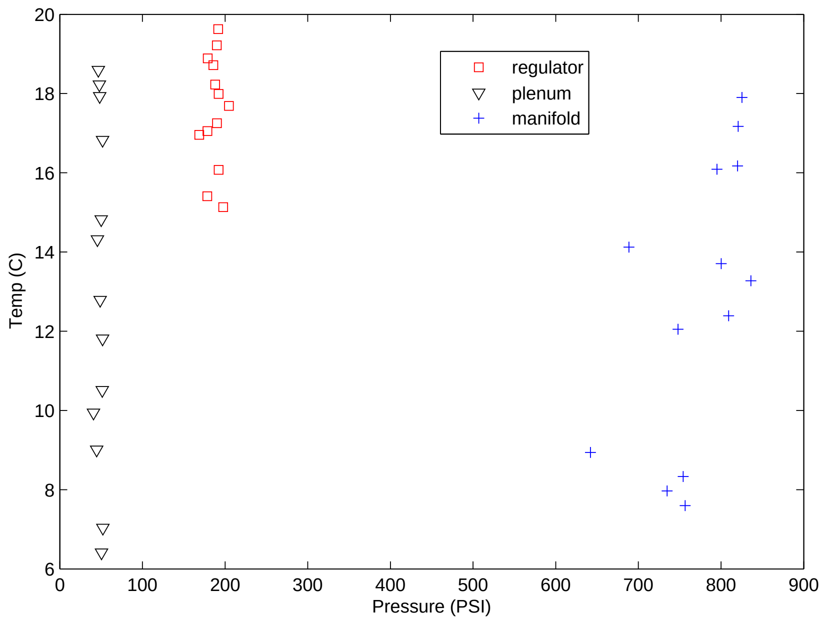
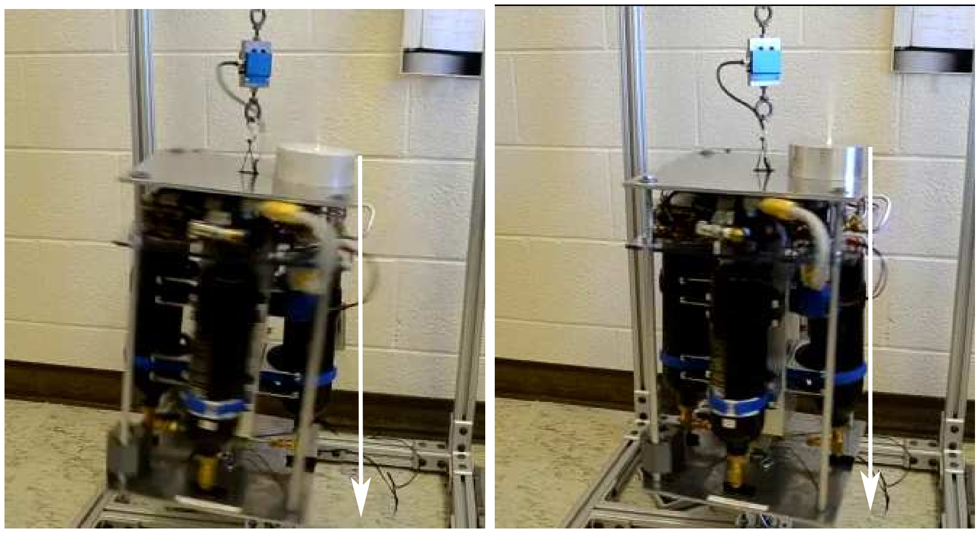


Fig. 7.4: Pressure vs. temperature data for tests conducted using annular flow with a secondary injection for the regulator, plenum, and manifold.

The natural frequency of the system was found by placing 100 grams on the pulley moving the satellite back to an angle of around 15 degrees. Once the satellite was a rest the weight was quickly removed and the satellite was allowed to oscillated for 300 seconds. Angular position data was gathered and plotted in Figure 7.7(a). The Fast Fourier Transform (FFT) of the data was taken to find the natural frequency of the system as shown in Figure 7.7(b). Once the natural frequency was observed the damping ratio was found by implementing an exponential line fit and solving for ς in Equation (7.2). I_{xx} was also found by solving Equation (7.3). Results was compared to and agreed within 3% of the moments of inertia calculated by Solid Edge.

$$y = X \exp(-\varsigma \omega_n t) \tag{7.2}$$

$$\omega_o = \sqrt{\frac{k}{I_{xx}}} \tag{7.3}$$



(a) (b)

Fig. 7.5: Comparison of the swaying motion produced by annular flow. a) Sway which occurred near the end of a test. b) Sway during the beginning of a test.

7.2.2 Secondary Injection Port Hardware Modifications

The initial machining of the aerospike was not completed according to specification. The base hole of the secondary injection port did not line up properly with the intersecting drill hole leading to the contour of the spike resulting in almost no side force. To resolve this issue a bigger drill bit was used to bore out the base hole wide enough to allow ample flow to the intersecting drill hole. With this modification, the open-loop tests were performed again and 2 rad/s response times were approximately 1 second faster than the previously observed response times. Figure 7.8 shows this result.

7.2.3 Closed-Loop Control Law Development

The control law was developed using a commonly used feedback controller known as a proportional-integral-derivative (PID) controller. Figure 7.9 [32] shows a block diagram of a this controller. The PID controller is used to calculate and minimize the error by taking the difference between the actual and the desired process and adjusts the control output accordingly. For the satellite system, the desired process to control is the angular position. The feedback loop used the actual angular position from the IMU sensors and compare it

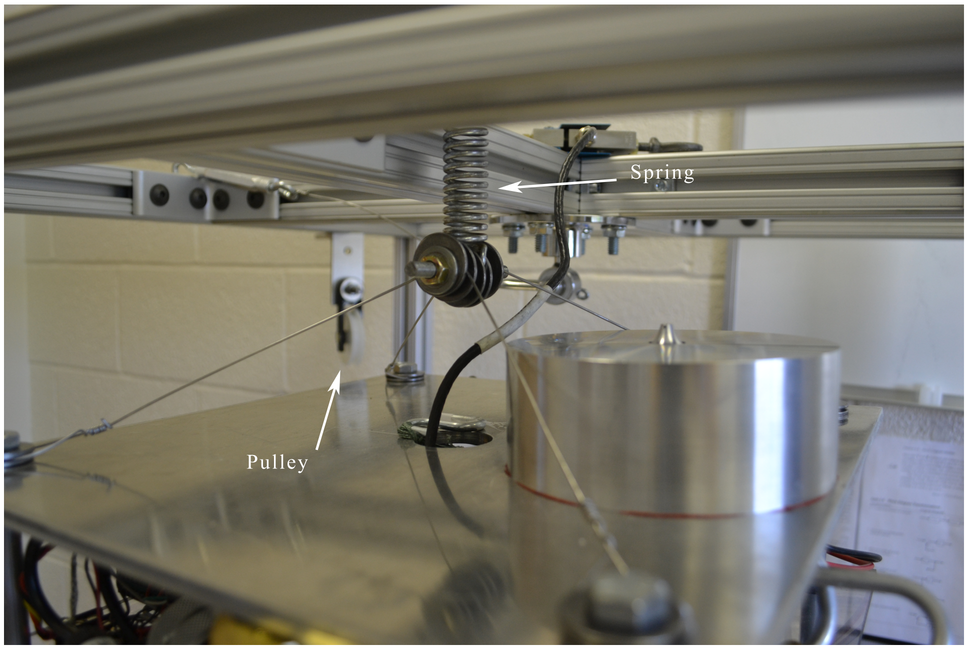
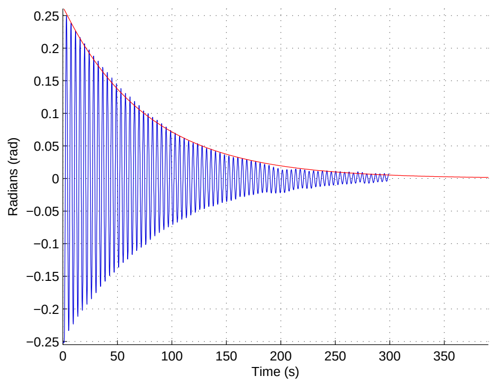
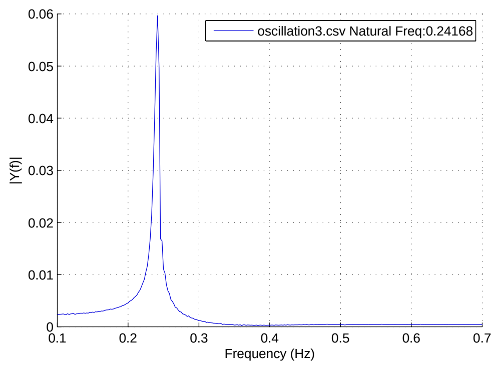


Fig. 7.6: Test hardware to find the natural frequency and damping ratio of the satellite.



(a)



(b)

Fig. 7.7: Test data collected for finding the natural frequency and damping ratio of the satellite. a) Average oscillation of the satellite with the an exponential line fit. b) FFT of the average oscillation plot.

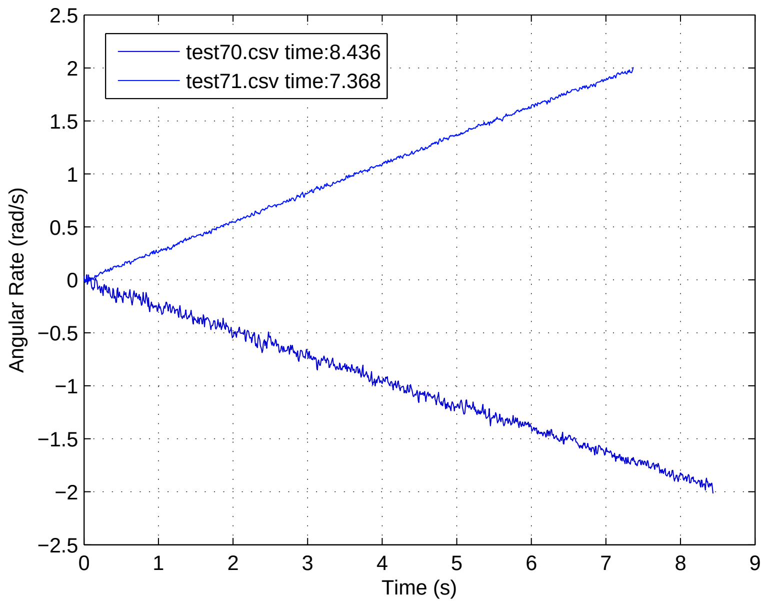


Fig. 7.8: Test to compare the secondary injection port #1 with the secondary injection port #2. Port #1 was re-drilled to allow a greater mass flow to inject onto the contour of the aerospike.

to the desired position to calculate the total angular error. This error will be multiplied by each section of the PID controller and summed to produce the control output for the secondary injection ports.

Before applying control laws the system must be modeled. The satellite system is modeled by the relationship between desired angel and output torque. Equation (7.4) defines the systems open-loop transfer function where $0 \leq \varsigma \leq 1$ where K is the system gain, ω_n is the system's natural frequency, and ς is the system's damping ratio.

$$G(s) = K \frac{\omega_n^2}{s^2 + 2\varsigma\omega_n s + \omega_n^2} \tag{7.4}$$

A PID controllers transfer function is defined in Equation (7.5), where K_p is the controller's proportional gain, K_i is the controller's integral gain, and K_d is the controllers derivative gain. Equation (7.6) is the systems characteristic equation.

$$Gc(s) = \frac{K_d s^2 + K_p s + K_i}{s} \tag{7.5}$$

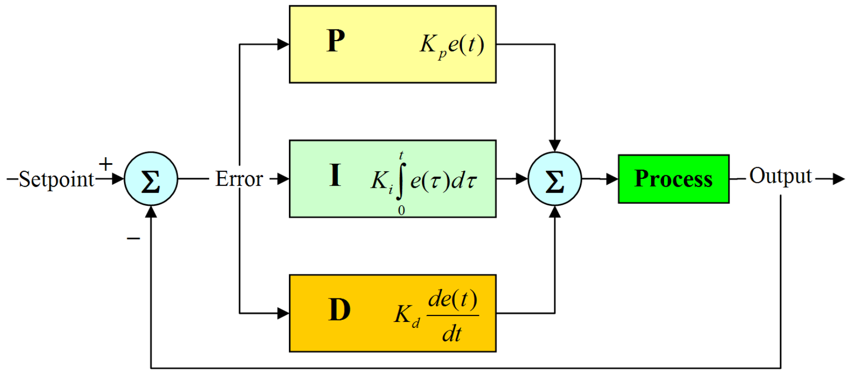


Fig. 7.9: Block diagram of a typical PID controller.

$$q(s) = \alpha s^3 + (2\varsigma\omega_n\alpha + K_d\alpha\omega_n^2)s^2 + (\omega_n^2 + K_p\alpha\omega_n^2)s + \omega_n^2 K_i \tag{7.6}$$

Control algorithms were developed and simulated using mathsoft's MATLAB. Figure 7.10 shows a root locus plot of the system and plant while Figure 7.11 shows the controlled position using the PID controller described above. After developing the controller in MATLAB, the algorithms were implemented and processed on the Overo Gumstix via the interface board. After running through the PID controller the control output is converted to a pulse-width modulation (PWM) signal for the system output.

7.2.4 Control Algorithm Implementation

An Overo Fire Gumstix was used as the onboard computer for implementing the control algorithm. Upon this sequence the communication between the IMU sensor and gumstix is established along with initialization of the PID control gains and control input. The control input is defined in *rad* as the angular position of the satellite. After initializing the system the process steps into the control feedback loop. First, the control feedback loop requests information from the IMU sensor to describing the satellites current angular position and angular rates. Subsequent to this request the information is read from the IMU and stored into buffers for computing. The current angular position is compared to the desired angular position resulting in a total angular error. The PID controller computes the proportional,

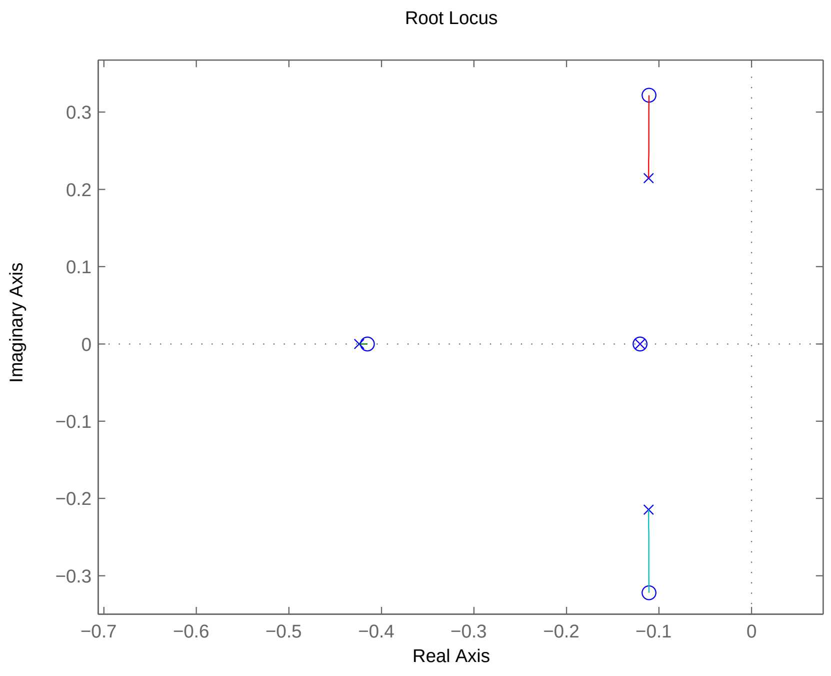


Fig. 7.10: Root locus plot of the PID controller and plant. The x represents the poles as the o represent the zeros. The controller can be selected by selecting a point along the root locus that coincides with damping ratio and natural frequency.

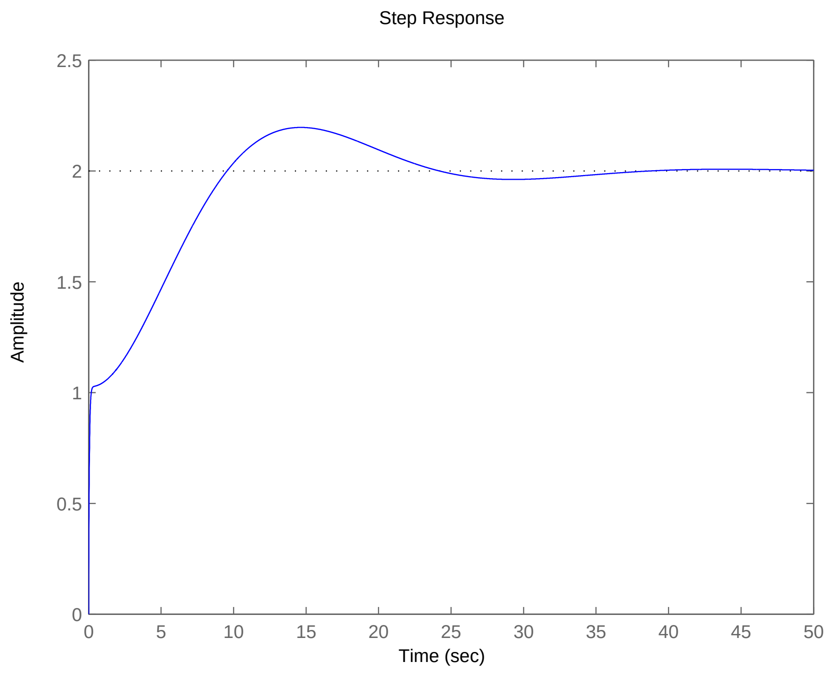


Fig. 7.11: Controlled position using the PID controller tested in MATLAB.

integral, and derivative errors separately. The proportional controller multiplies the error by the proportional gain. The integral controller multiplies total tests accumulated angular error by the integral gain. The derivative controller multiplies the angular velocity by the derivative gain. The PID calculations are summed to produce a the controller output for the satellite system shown in Equation (7.7).

$$mb = K_d e(t) + K_i \int e(t) + K_d \frac{de(t)}{dt} \tag{7.7}$$

Because the valve is either active or inactive a PWM conversion is implemented. This conversion is implemented by converting the PID control output to a set pulse length within the operating frequency range. The control feedback loop runs at 2 Hz. This frequency is set by the B-Series Cryogenic valves which have an opening response time of 50 ms, thus the shortest pulse achievable for the valve to fully open and close is 0.1 seconds. The frequency is set to 2 Hz to allow the longest pulse to be five times the shortest pulse length. The PID control output becomes a factor of how long the valve is open during the 0.5 second cycle interval. The conversion is defined in Equation (7.8) and shown in Figure 7.12. The test runs for an predetermed time. After the time is reached all valves and communication ports are closed. The code was developed in python and located in the appendixes. Equation (7.8) limits commanded moments to positive values. Negative moment commands are generated by firing the opposing vectoring port.

$$V_o = \left\{ \begin{array}{cc|c} if & mb \leq 0 & V_o = 0 \\ elseif & 0 < mb < max & V_o = mb \\ elseif & mb \geq max & V_o = max \\ else & & V_o = 0 \end{array} \right. \tag{7.8}$$

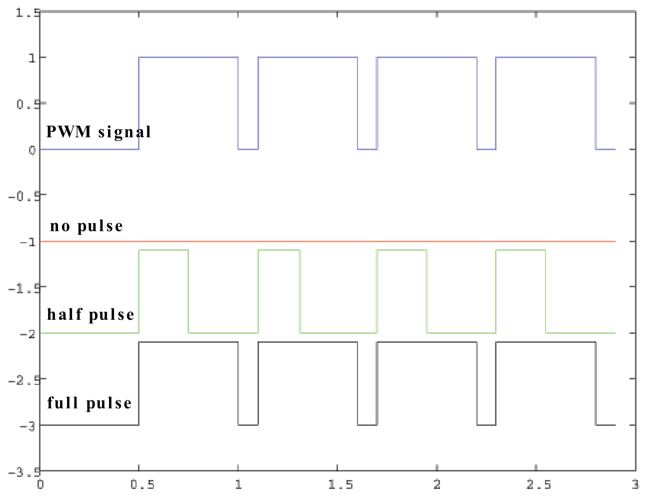


Fig. 7.12: PWM signal shaping for (a) no pulse, (b) half pulse, and (c) full pulse.

7.2.5 Simulated Closed-Loop Positioning Results

Figures 7.13, 7.14, and 7.15 show results from MATLAB simulations using a PWM controller. The satellite is simulated without the presence of annular flow. Simulation results include the angular position, solenoid pulse, and angular error.

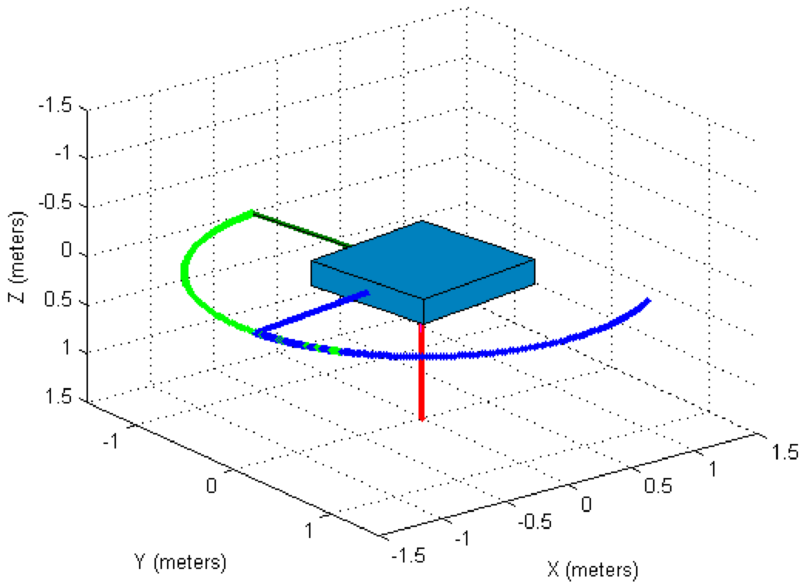


Fig. 7.13: MATLAB simulation of graphical output using a PID controller.

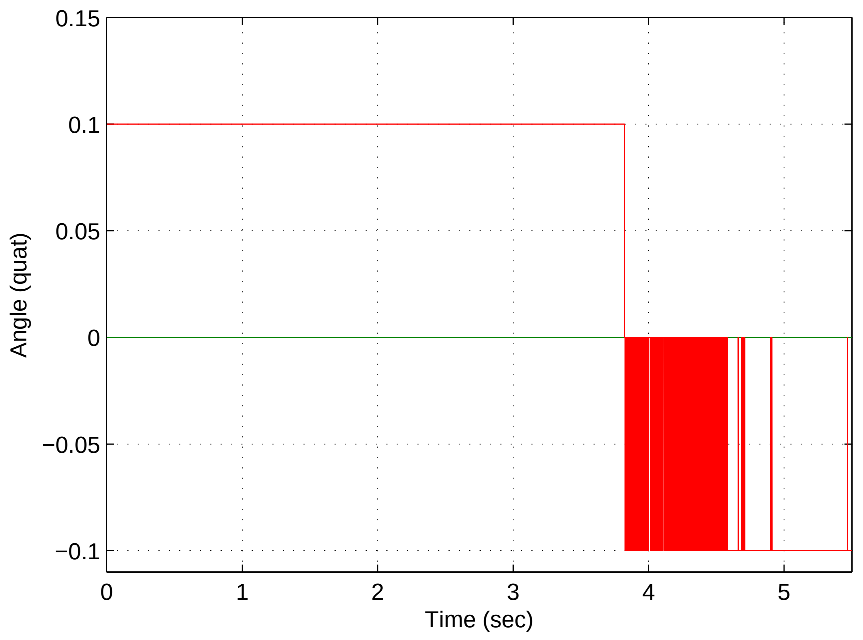


Fig. 7.14: MATLAB simulation pulses sent to solenoid valves after PWM conversion.

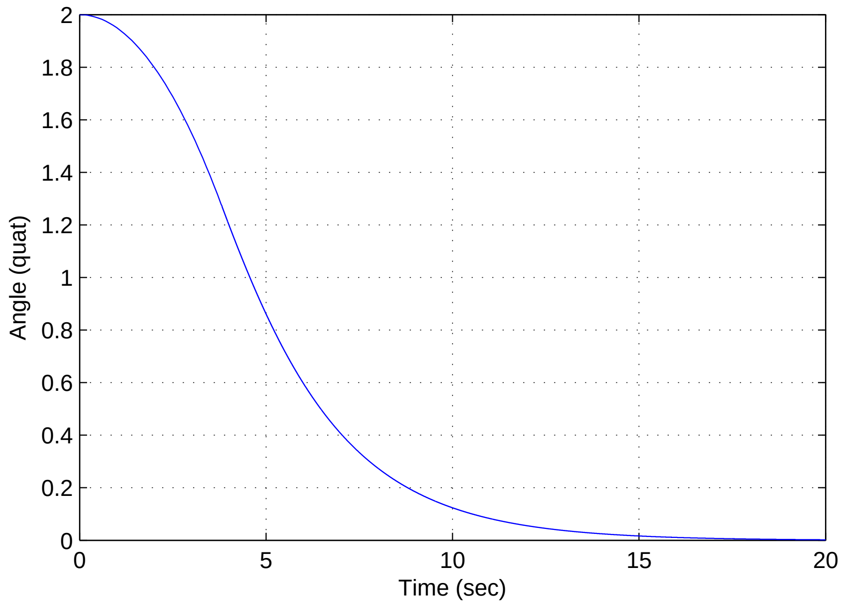


Fig. 7.15: MatLab simulation angular error.

7.2.6 Closed-Loop Tests with Secondary Injection Only

Closed-loop vectoring tests with secondary injection were conducted using both a PI and PID controller. Damping was initially neglected. Figure 7.16 shows the PI controller in a oscillatory state, due to the lack of appreciable damping. As the derivative term was added into the controller the plants oscillation significantly reduced. Figure 7.17 shows this result. The initial gains were derived from the natural frequency of the system, but later manually tweaked to provide better control. The gains for the PID controller are $K_p = 5$, $K_i = 1$, and $K_d = -2.7$. The tests started around 0 *rad* with a control input angle of 1.56 *rad*. The tests were very reliable and repeatable. All pressures and temperature were consistent throughout the test duration.

7.2.7 Closed-Loop Tests with Annular Flow and Secondary Injection

Closed-loop thrust vectoring tests with annular flow and secondary injection were performed using a PID controller with a control input of 1.56 *rad*. Initially gains from the previous tests were used and the derivative gain was manually adjusted to give the desired response. This damping factor slowed down the rotation of the satellite when annular flow was on. Figure 7.18 shows the angular position of the PID controller vs. time.

The angular rates were also plotted in Figure 7.19 to show when the secondary pulsing took place. Each sharp ridge indicates a pulse. A single angular rate was plotted in Figure 7.20 to better view the pulses of a single test. As time increases, or the test satellite moves closer to the desired position, the pulses become more rapid and multidirectional to hold the position.

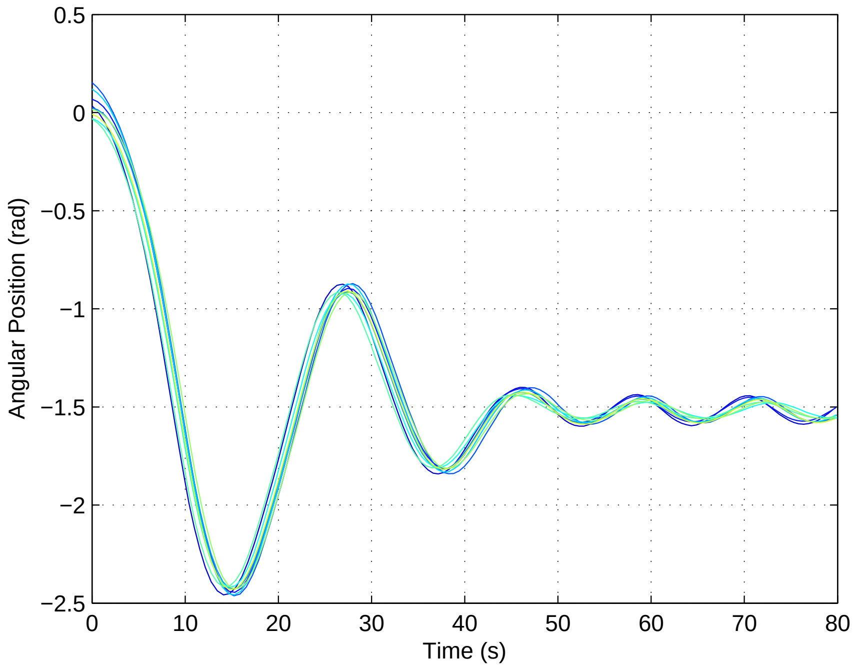


Fig. 7.16: Position test using only the secondary injection ports with a PI controller.

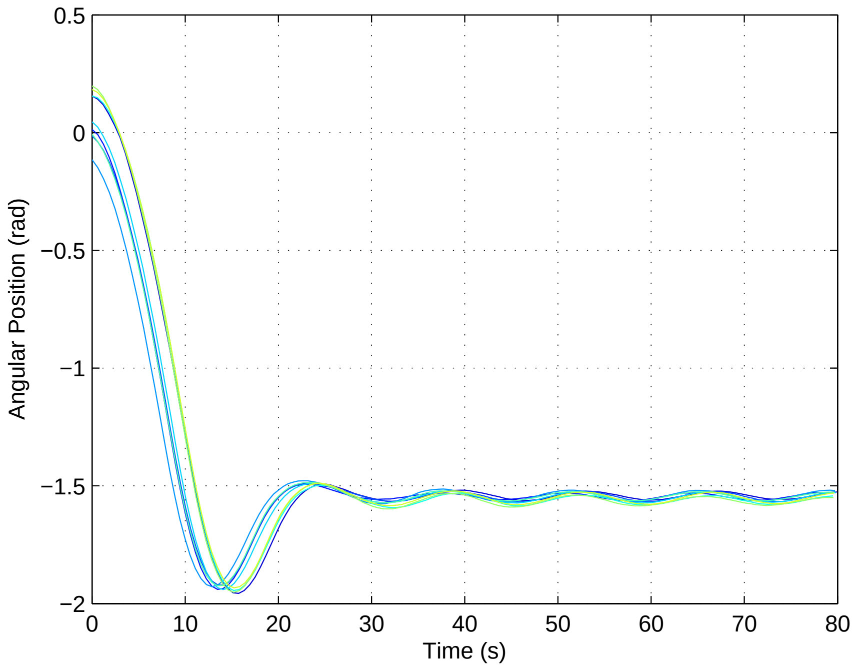


Fig. 7.17: Position test using only the secondary injection ports with a PID controller.

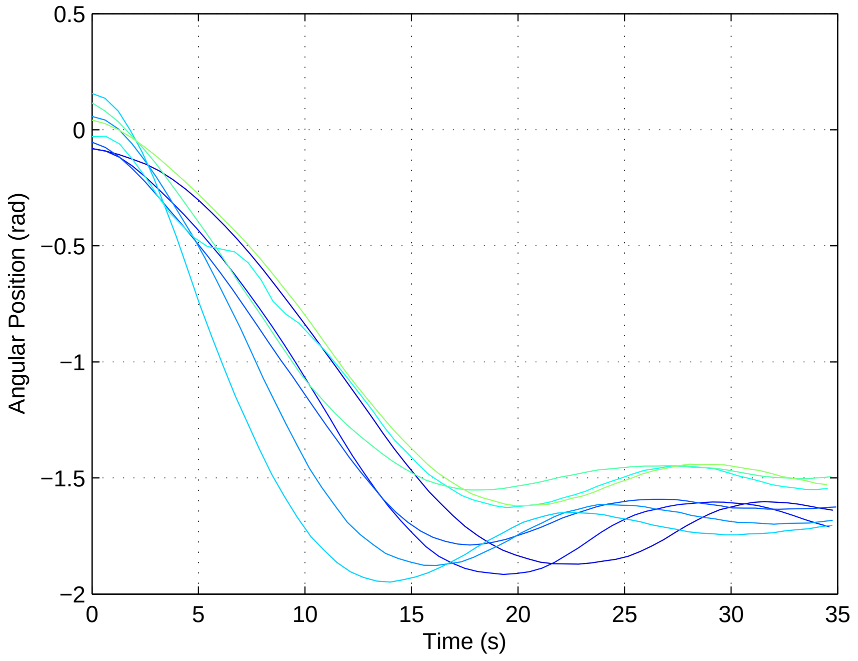


Fig. 7.18: Position tests using annular flow and secondary injection with a PID controller.

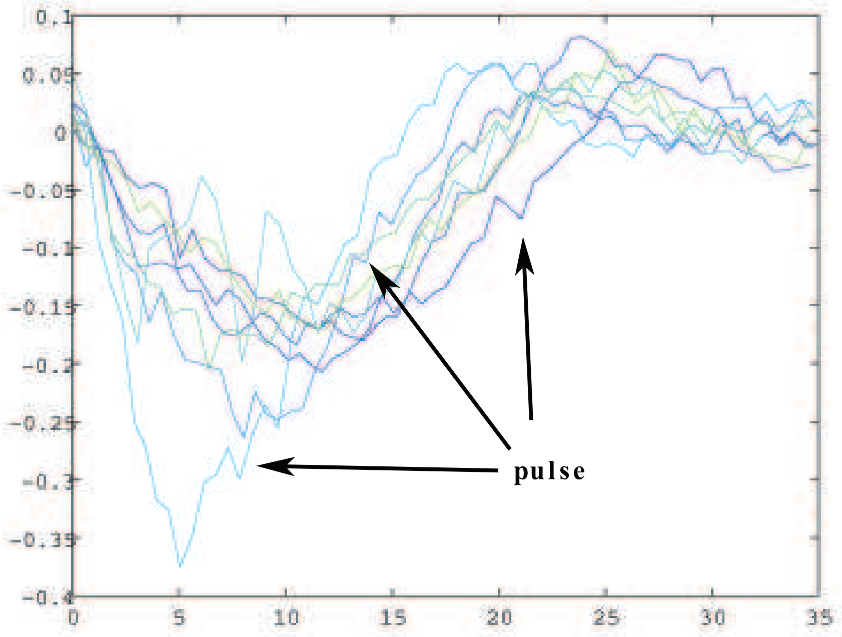


Fig. 7.19: Angular rates during the position test using annular flow and secondary injection with a PID controller.

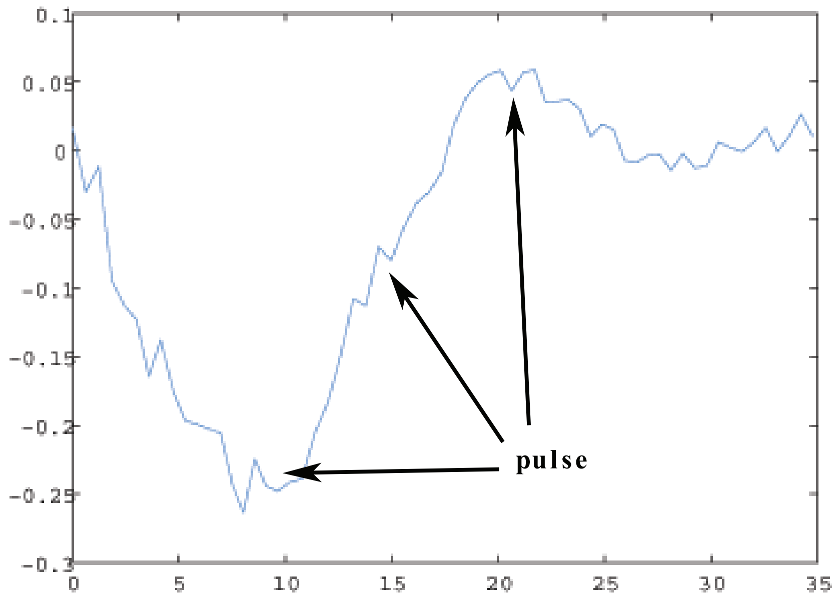


Fig. 7.20: Angular rates from a single position test using annular flow and secondary injection with a PID controller.

Chapter 8

Summary Of Experimental Results

This chapter summarizes the experimental tests conducted during this research. Comparisons are made for secondary injection angular control with and without annular flow for open-loop and closed-loop tests.

8.1 Secondary Injection Comparison

Tests conducted without annular flow were very smooth and repeatable even with a slight variance in pressure and temperature. Predicted results concluded that the tests with annular flow would reach 2 rad/s faster than without annular flow. This however, was not the case as tests with primary flow took an average of 5 seconds longer to reach the desired angular rate with the secondary injection port producing a mean force of 4.19 newtons. Figure 7.5 shows a swaying motion on the test rig during annular flow caused by the downward force produced by main plenum flow. Because the satellite hangs from the test stand the off-centered aerospike causes the satellite to precess in the direction of the force exerted from the plenum. Because of this precession a direct comparison cannot accurately be made. Numeric values of test performed with annular flow are shown in Table 8.1. The tests proved the ability to rotate the satellite with and without annular flow. Pulsing from secondary injection port only can maneuver a satellite.

8.2 Experimental Positioning Results

Tests were conducted by releasing CO_2 through secondary injection ports moving the satellite from an initial position of 0 rad to a final position of 1.56 rad. Figure 7.17 shows that without annular flow the system is very reliable and repeatable, indicating that attitude control using an aerospike nozzle on a small satellite is feasible. Control positioning tests with annular flow are also repeatable and reliable within tolerance. These results are not as clean as the results produced without annular flow due to the swaying motion produced in the test stand. This inconsistent test aparatus artifact must be corrected when developing a full 3-DoF test article. Generally, the presented results demonstrate the feasability of controlling a small satellite using secondary injection ports on an aerospike nozzle.

Table 8.1: Numeric values for open-loop annular flow tests. All temperatures are in C°.

Test #	Load Cell (LBS)			Pressure Plenum (PSI)				Pressure Regulator (PSI)				Pressure Manifold (PSI)			
	Max.	Avg.	Min.	Start	Avg.	End	Temp.	Start	Avg.	End	Temp.	Start	Avg.	End	Temp.
31	3.54	2.73	0.00	21.86	51.85	51.84	16.82	207.82	204.74	203.96	16.09	840.86	792.11	779.41	17.68
34	2.45	1.48	-1.33	44.56	52.15	43.29	7.03	196.75	197.61	190.82	14.12	750.90	688.54	635.38	15.13
36	2.45	1.37	-1.23	23.64	47.89	49.10	18.22	195.03	188.03	183.76	17.90	881.75	825.35	788.40	18.22
37	2.07	1.02	-1.78	23.19	51.72	49.28	11.81	191.79	192.24	189.61	12.05	808.49	748.00	703.79	16.07
38	2.77	1.72	-0.78	26.58	48.27	48.25	17.93	193.24	185.69	182.37	17.17	868.57	820.78	790.37	18.71
39	2.20	1.09	-1.56	24.03	51.39	50.80	10.51	189.96	190.01	186.68	8.33	807.24	754.12	718.63	19.21
40	2.59	1.46	-1.15	26.04	48.78	47.12	12.78	196.10	190.11	188.15	12.38	865.98	809.14	773.43	17.25
41	2.65	1.59	-0.95	21.43	50.51	46.20	6.41	194.37	192.18	185.42	7.96	788.77	734.67	697.26	17.98
42	1.83	0.68	-2.44	26.97	50.16	55.68	14.82	195.21	191.56	198.76	16.17	879.16	819.90	773.12	19.62
43	2.64	1.39	-1.09	24.36	46.55	44.93	18.59	183.06	178.42	169.56	13.70	871.65	800.25	736.79	15.40
44	2.76	1.52	-0.35	23.27	40.71	37.09	9.94	182.32	168.64	148.25	8.93	765.98	642.05	515.24	16.95
45	2.50	1.20	-1.12	23.19	45.55	48.99	14.31	185.61	178.55	174.63	13.27	890.24	836.15	797.32	17.05
46	2.83	1.69	-0.54	48.60	44.57	40.54	9.01	186.41	178.97	178.57	7.59	817.52	756.50	714.13	18.88

Chapter 9

Conclusion

This paper supports the concept of using secondary injection ports on an aerospike nozzle to control a prototype *CubeSat*. Thrust vectoring tests were performed using open-loop and closed-loop control. Open-loop control tests were used to quantify the delivered side fore and moment levels as well as correcting any system bugs. During these tests the secondary injection port was used to prototype satellite spin up to an angular rate of 2 rad/s. Consistently achieved angular rate demonstrated high control fidelity and repeatability. The time required to achieve 2 rad/s varied by 0.328 seconds between the eight conducted tests with a mean time of 9.85 seconds.

The same tests conducted with annular flow were moderately noisy and demonstrated less control fidelity. The mean time to reach a angular rate of 2 rad/s was 15 seconds. The observed time differences between tests with and without annular flow were primarily due to the swivel friction and off-axis swaying of the satellite caused by annular flow.

Closed-loop thrust vectoring tests consist of placing the satellite at a resting state and controlling the secondary injection ports to move the satellite to a desired known position. A PID controller was used for closed-loop vectoring control. Tests without annular flow were conducted for 80 seconds each. It was observed that positioning control is repeatable and reliable.

Closed-loop thrust vectoring tests with annular flow were run for 35 seconds. These tests were moderately noisy and varied more than the tests without annular flow due to the swivel friction and off-axis swaying of the satellite. Nonetheless the prototype system is controllable and repeatable with a closed-loop PID controller.

The hardware in the loop demonstration realized the capability to precisely position a satellite in a single axis coordinate frame with and without annular flow. The initial tests have verified objectives 1 and 2 showing that rotational ramping is achieved from the secondary injection ports simulating the ability to de-tumble or position a satellite. Final tests have verified objectives 3 and 4, proving the ability to precisely position a satellite for a one-axis attitude control simulation. The above tests indicates that using an aerospike nozzle with secondary injection ports for attitude control is feasible.

Immediate future work includes developing different control algorithms with a comparison test to see which provides the best desired control. More extensive future work for developing the aerospike nozzle to achieve a sufficient TRL level for flight is described below.

- Developing a prototype satellite integrating two aerospike nozzles for controlling one-axis of rotation.
- Closed-loop control demonstration using the aerospike nozzles for a 3DoF simulation using cold gas.
- Hanging the prototype to a tethered balloon showing 3DoF control using cold gas.
- Developing a scaled prototype satellite integrating two aerospike nozzles using hot gas.
- Testing control algorithms for a 3DoF ground demonstrations.
- Development of a satellite system to fit within a 2U *CubeSat*.
- Testing control algorithms for a 3DoF ground demonstration.
- Designing actual flight hardware for a *CubeSat* propulsion unit.

Chapter 10
Recommendations

Below is a list of recommendations for those perusing future research and development in this area of study. The list contains areas of the project that can be improved as well as lessons learned from the development process.

- Use High Pressure Air (HPA) instead of CO2. This would elevate any freezing in the system and two phase flow.

- Develop a better method for frictionless rotation. For example consider developing an air table to suspend the satellite.

- If the propellant tanks are going to be taken on and off the system for refilling consider using a stronger plumbing material than aluminum. Some of the piping was replaced several times due to the continuous stress of refilling propellant bottles.

- If running future tests with only one aerospike nozzle consider possible solutions to mitigate the precession in the system. This would allow for a comparison to be made between the tests with annular flow and without.

References

[1] Boeing, “Advanced aerodynamic spike configurations, volume 1,” Rocketdyne Advanced Projects, Tech. Rep., 1967.

[2] ——, “Studies of improved Saturn V vehicles and intermediate payload vehicles,” The Boeing Company - Space Division, Tech. Rep., 1996.

[3] ——, “Advanced aerodynamic spike congifurations, volume 2,” Rocketdyne Advanced Projects, Tech. Rep., 1967.

[4] C. Bendersky, *Space Shuttle Propulsion Issue, Staged Combustion Bell Versus Tap-Off or Gas-Generator Aerospike*, Bellcomm. Inc. Std.

[5] K. C. Hendershot, R. J. Sergeant, and H. B. Wilson, “A new approach for evaluating the performance and base environment characteristics of nonconventional rocket propulsion systems,” *AIAA*, vol. AIAA-67-256, 1967.

[6] J. J. Korte, A. O. Salas, H. J. Dunn, N. M. Alexandrov, W. W. Follett, G. E. Orient, and A. H. Hadid, “Multidisciplinary approach to aerospike nozzle design,” National Aeronautics and Space Administration, Langley Research Center, Tech. Rep., 1997.

[7] K. Higdon and D. B. Landrum, “Analysis of annular plug nozzle performance and TVC,” *AIAA paper*, vol. AIAA-2003-4908, 2003.

[8] H. F. R. Schoyer, “Thrust vector control for (clustered modules) plug nozzles: Some considerations,” *Journal of Propulsion and Power*, vol. 16, pp. 196–201, 2000.

[9] J. Ruf and D. McDaniels, “Experimental results for an annular aerospike with differential throttling,” in *5th International Symposium on Liquid Space Propulsion*, 2003.

[10] M. Fick and R. H. Schmucker, “Remarks on plug cluster nozzles,” in *31st AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, vol. AIAA 95-2694, 1995.

[11] G. Hagemann, C.-A. Schley, E. Odintsov, and A. Sobatchkine, “Nozzle flowfield analyses with particular regard to 3d-plug cluster configurations,” *AIAA*, vol. AIAA-1996-2954, 1996.

[12] R. Sorge, C. Carmicino, and A. Nocito, “Design of a lab-scale cooled two-dimensional plug nozzle for eperimental tests,” *38th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2002.

[13] F. Nasuti, M. Geron, and R. Paciorri, “Three dimensional features of clustered plug nozzle flows,” in *39th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2003.

[14] T. Ito, K. Fujii, and G. Hagemann, “Numerical investigation of the side-fence effect on linear plug nozzle perfomrance,” *AIAA*, vol. AIAA 2004-4018, 2004.

[15] T. Ito, K. Fujii, and A. Hayashi, "Computations of the axisymmetric plug nozzle flow fields: Flow structures and thrust performance," in *17th AIAA Applied Aerodynamics Conference*, no. AIAA 1999-3211, 1999.

[16] H. Miyamoto, A. Matsuo, and T. Kojima, "Effects of sidewall configurations on rectangular plug nozzle performance," *AIAA*, vol. AIAA-2006-4373, 2006.

[17] T. Ito and K. Fujii, "Flow field and performance analysis of an annular-type aerospike nozzle with base bleeding," *Transactions of the Japan Society for Aeronautical and Space Sciences*, vol. 46-151, pp. 17–23, 2003.

[18] ——, "Numerical analysis of the base bleed effect on the aerospike nozzles," *Transactions of the Japan Society for Aeronautical and Space Sciences*, vol. 46-151, pp. 17–23, 2003.

[19] ——, "Flow field analysis of the base region of axisymmetric aerospike nozzles," in *39th AIAA Aerospace Sciences Meeting & Exhibit*, no. AIAA 2001-1051, 2001.

[20] A. N. Kraiko and N. I. Tillyayeva, "Contouring spike nozzles and determining the optimal direction of their primary flows," *Fluid Dynamics*, vol. 42-2, pp. 321–329, 2007.

[21] N. Karthikeyan, S. B. Verma, and L. V. Venkatakrishnan, "Experimental investigation of the acoustics of an annular aerospike nozzle flow," in *15th AIAA/CEAS Aeroacoustics Conference (30th AIAA Aeroacoustics Conference)*, 2009.

[22] S. C. Shark, J. D. Dennis, and J. K. Villarreal, "Experimental performance analysis of a toroidal aerospike nozzle integrated with a n2o/htpb hybrid rocket motor," in *46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2010.

[23] J. R. Stoffel, "Experimental and theoretical investigation of aerospike nozzles in a hybrid rocket propulsion system," in *47th AIAA Aerospace Sciences Meeting*, 2009.

[24] P. Lemieux, "Nitrous oxide cooling in hybrid rocket nozzles," *Progress in Aerospace Sciences*, vol. 46, pp. 106–115, 2009.

[25] E. Besnard and J. Garvey, "Aerospike engines for nanosat and small launch vehicles (nlv/slv)," in *Space 2004 Conference & Exhibit*, 2004.

[26] ——, "Development and flight-testing of liquid propellant aerospike engines," in *40th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 2004.

[27] S. D. Eilers, M. D. Wilson, D. S. A. Whitmore, and Z. Thicksten, "Analytical and experimental evaluation of aerodynamic thrust vectoring on an aerospike nozzle," *AIAA Journal*, 2011.

[28] A. Naghib-Lahouti and E. Tolouei, "Investigation of the effect of base bleed on thrust performance of a truncated aerospike nozzle in off-design conditions," in *European Conference on Computational Fluid Dynamics*, 2006.

[29] W. A. Jordi Puig-Suari, Clark Turner, “Development of the standard cubesat deployer and a cubesat class picosatellite,” in *Aerospace Conference*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=931726: IEEE, 2001.

[30] T. Angstadt, “Fortran program for plug nozzle design,” 2001, a written assignment for TECH 591.

[31] M. Inc., “3DM-GX3-25, minature attitude, heading reference system (AHRS),” June 2010. [Online]. Available: http://www.microstrain.com/3dm-gx3-25.aspx

[32] SilverStar. (2006, October) Depicts a traditional pid controller. [Online]. Available: http://en.wikipedia.org/wiki/File:Pid-feedback-nct-int-correct.png

Appendices

Appendix A

Gumstix Python Programming Code

A.1 Main For Gyro

The *main.py* function calls all other functions including the *imu.py*, *aerospike.py*, *data_handler.py*, and *ground_communication.py*. This code also holds the control algorithms for turning on the annular flow and secondary injection ports.

```
# runs from ./run_avionics.sh
# inputs none
# outputs IMU data, control signals

import imu
import aerospike
import data_handler
import math
import time

def main():
    data = data_handler.DataHandler()
    data.open()

    IMU = imu.IMU() #call imu.py
    IMU.open() #opens serial port

    spike = aerospike.Aerospike()
    spike.initialize()
    ON = int(aerospike.ON)
    OFF = int(aerospike.OFF)

    imu_data = []
    setAccelSpeed = 2 # test to run from 0 rad/s to this set speed
    AngRatez = 0
    sleeptime = 1
    sleepstring = 'Test will begin in ' + repr(sleeptime) + ' sec.'
    print sleepstring
    time.sleep(sleeptime)
    starttime = time.time()
    #spike.on() #annular flow on or off

    while math.fabs(AngRatez) < math.fabs(setAccelSpeed):
        imu_data, packed_imu_data = IMU.read() #read IMU data
        AngRatez = imu_data[5] #parse angular rate Z

        print AngRatez
        #spike.on1() #secondary1 flow
        spike.on2() #secondary2 flow use this one

# Save the data
        data.save( packed_imu_data )

    endtime = time.time() #test endtime tag
    timestring = 'Test time: ' + repr(endtime-starttime) + ' sec.'
    print timestring
    spike.off1() #secondary1 off
    spike.off2() #secondary2 off
    spike.off() #annular flow       off
    data.close() #close serial
    IMU.close() #close serial
    spike.close() #turn off spike

if __name__ == "__main__":
    main()
```

A.2 Main For Control

The is the main function for the control code. It runs the same functionality as the above main code for the gyro tests. This code is a lot more involved because of the control algorithms.

```
# runs from ./run_avionics.sh
# inputs none
# outputs IMU data, control signals

import imu
import aerospike
import data_handler
import math
import time

def main():
    data = data_handler.DataHandler()
    data.open()
```

```
IMU = imu.IMU()
IMU.open()

spike = aerospike.Aerospike()
spike.initialize()
ON = int(aerospike.ON)
OFF = int(aerospike.OFF)

####Initializations####
imu_data = []
YawCommand = 0 #rad
PulseGain = .1
testtime = 80 #test duration in sec.
pastYawDot = 0
maxPulse = 0.51 # max pulse (on/off time)
sleeptime = 1 #wait time for start of test
#gains
Kp = 5
Ki = 1
Kd = -5.7
#######################
sleepstring = 'Test will begin in ' +
      repr(sleeptime) + ' seconds'
print sleepstring
time.sleep(sleeptime)
pastTime = time.time()
testDuration = testtime + pastTime
#spike.on() #annular flow on/off
```

```
while testDuration > pastTime:
 #read IMU and get data
 imu_data, packed_imu_data = IMU.read()
 Derivative = imu_data[5] #angular rate
 magX = imu_data[6] #magX
 Yaw = magX
 YawError = YawCommand - Yaw #error



 #find shortest distance between Desired and
     #Actual position
 if YawError > math.pi:
   Proportional = YawError - (2*math.pi)
 elif YawError < -math.pi:
   Proportional = YawError + (2*math.pi)
 else:
 Proportional = YawError #proportional error

 #Controller Gains
 deltaTime = time.time() - pastTime
 Integral = (YawError * deltaTime) + pastYawDot
 m_b = Kp*Proportional + Ki*Integral + Kd*Derivative
 waitTime = math.fabs(m_b)*PulseGain

     #set pulse ON time to limits if above
 if waitTime > maxPulse:
   waitTime = maxPulse
```

```
else:
   waitTime = waitTime

      #set pulse OFF time
closeTime = maxPulse - waitTime
if closeTime < maxPulse/2 and closeTime != 0:
   closeTime = maxPulse/2
   waitTime = maxPulse/2

#Command injecion ON pulse and direction
if m_b <= 0:
   spike.off2()
   spike.on1()
   time.sleep(waitTime)
elif m_b > 0:
   spike.off1()
   spike.on2()
   time.sleep(waitTime)
else:
   print "how did I get here in the code?"
   waitTime = 0
   closeTime = 0

      #turn off injections and wait for OFF time
spike.off1()
spike.off2()
time.sleep(0.5)#closeTime)
```

```
        pastTime = time.time()
        pastYawDot = Integral

# Save the data
        data.save( packed_imu_data )

    #close everything
    spike.off()
    spike.off1()
    spike.off2()
    data.close()
    IMU.close()
    spike.close()
    print "test is over"

if __name__ == "__main__":
    main()
```

A.3 IMU For Gyro

The *imu.py* code initializes the IMU with the correct serial interface. It also reads the IMU and writes packed and unpacked data as outputs.

```
#inputs none
#outputs imu packed data and unpacked data in the data format

import serial
import struct

IMU_PORT = '/dev/ttyS0' #port
```

```
IMU_BAUD = 115200 #baud rate
IMU_TIME_CONVERSION = 62500.0 #IMU time converstion

CMD_ACCEL_ANG_ORIENT = '\xD2' #for gyro-stabilized acceleration'
CMD_ACCEL_ANG_ORIENT_SIZE = 43 # response bytes
CMD_ACCEL_ANG_ORIENT_DATA_FORMAT = '>cffffffffIH' #data format


class IMU:
  def __init__(self):
    self.IMU_PORT = IMU_PORT
    self.IMU_BAUD = IMU_BAUD

    self.IMU_COMMAND = CMD_ACCEL_ANG_ORIENT
    self.IMU_MESSAGE_SIZE = CMD_ACCEL_ANG_ORIENT_SIZE
    self.IMU_COMMAND_DATA_FORMAT=CMD_ACCEL_ANG_ORIENT_DATA_FORMAT

  def open(self): #open serial port
    self.imu = serial.Serial(self.IMU_PORT, self.IMU_BAUD)

  def close(self): #close serial port
    self.imu.close()

  def read(self): #read IMU data
    self.imu.write(self.IMU_COMMAND)

    #TODO check IMU write

    data = []
```

```
        data = self.imu.read(self.IMU_MESSAGE_SIZE)

        #TODO check read status, check first char, checksum
        #unpack IMU data
        imu_parsed_data=struct.unpack(self.IMU_COMMAND_DATA_FORMAT,
                     data)

        #handle clock rollover outside of function
        secs = imu_parsed_data[-2] / IMU_TIME_CONVERSION # to sec.
        processed_data = imu_parsed_data[1:-2] + ( secs, )

        return processed_data, data

def main():
    imu = IMU()
    imu.open()

    data  = imu.read()

    print data

    imu.close()

if __name__ == "__main__":
    main()
```

A.4 IMU For Control

The *imu.py* code for control is the same as the imu.py for the gyro, but with different command and response bytes.

```
#inputs none
#outputs imu packed data and unpacked data in the data format

import serial
import struct

IMU_PORT = '/dev/ttyS0' #port
IMU_BAUD = 115200 #baud rate
IMU_TIME_CONVERSION = 62500.0  #IMU time converstion

CMD_ACCEL_ANG_ORIENT = '\xCC' #IMU command byte
CMD_ACCEL_ANG_ORIENT_SIZE = 79 # response message byte size
CMD_ACCEL_ANG_ORIENT_DATA_FORMAT = '>cffffffffffffffffffIH' #fmt

class IMU:
  def __init__(self):
    self.IMU_PORT = IMU_PORT
    self.IMU_BAUD = IMU_BAUD

    self.IMU_COMMAND = CMD_ACCEL_ANG_ORIENT
    self.IMU_MESSAGE_SIZE = CMD_ACCEL_ANG_ORIENT_SIZE
    self.IMU_COMMAND_DATA_FORMAT=CMD_ACCEL_ANG_ORIENT_DATA_FORMAT

  def open(self): #open serial port
    self.imu = serial.Serial(self.IMU_PORT, self.IMU_BAUD)
  def close(self): #close serial port
    self.imu.close()

  def read(self): #read IMU data
    self.imu.write(self.IMU_COMMAND)

    #TODO check IMU write

    data = []
    data = self.imu.read(self.IMU_MESSAGE_SIZE)

    #TODO check read status, check first char, checksum
    #unpack IMU data
    imu_parsed_data=struct.unpack(self.IMU_COMMAND_DATA_FORMAT,
           data)

    #handle clock rollover outside of function
    secs = imu_parsed_data[-2] / IMU_TIME_CONVERSION # to sec.
    processed_data = imu_parsed_data[1:-2] + ( secs, )

    return processed_data, data

def main():
  imu = IMU()
  imu.open()

  data  = imu.read()
```

```
    print data

    imu.close()

if __name__ == "__main__":
    main()
```

A.5 Aerospike

The *aerospike.py* code sets the gpio pins for the valves and holds the definitions that are called to turn on and off the annular and secondary injection flows.

```
#inputs none
#outputs voltages to control spike

from time import sleep

ON = '1'
OFF = '0'

class Aerospike:
    def __init__(self):
        self.SPIKE_PIN = "147"

    def initialize(self):
        #annular flow initialization...set gpio and turn off
        self.gpio_value = open('/sys/class/gpio/gpio173/value', 'w')
        self.gpio_value.write( OFF )
        #secondary port 1 initialization...set gpio and turn off
        self.gpio_value1 = open('/sys/class/gpio/gpio147/value', 'w')
        self.gpio_value1.write( OFF )
        #secondary port 2 initialization... set gpio and turn off
        self.gpio_value2 = open('/sys/class/gpio/gpio146/value', 'w')
        self.gpio_value2.write( OFF )

    def on(self): #annular flow on
        self.gpio_value.write( ON )
        self.gpio_value.flush()

    def on1(self): #secondary port 1 on
        self.gpio_value1.write( ON )
        self.gpio_value1.flush()

    def on2(self): #secondary port 2 on
        self.gpio_value2.write( ON )
        self.gpio_value2.flush()

    def off(self): #annular flow off
        self.gpio_value.write( OFF )
        self.gpio_value.flush()

    def off1(self): #secondary port 1 off
        self.gpio_value1.write( OFF )
        self.gpio_value1.flush()

    def off2(self): #secondary port 2 off
        self.gpio_value2.write( OFF )
```

```
        self.gpio_value2.flush()

    def close(self): #close all valves
        self.gpio_value.close()
        self.gpio_value1.close()
        self.gpio_value2.close()

#runs a setup to initialize and pusle the spike
def main():
    spiky = Aerospike()
    spiky.initialize()
    spiky.off()
    sleep(2)
    spiky.on()
    sleep(2)
    spiky.off()
    spiky.close()

if __name__ == "__main__":
    main()
```

A.6 Data Handler

The *data_handler.py* code takes the packed IMU data and writes it to a *avionics_data.dat* file.

```
# inputs IMU data
# outputs IMU data to ./avionics_data.dat file

import struct
import ground_communication

FILENAME = './avionics_data.dat'

class DataHandler:
    def __init__(self):
        pass
    #opens file
    def open(self):
        self.data_file = open(FILENAME, 'ab')
        self.comm = ground_communication.GroundCommunication()
        self.comm.open()
    #close file
    def close(self):
        self.comm.close()
        self.data_file.close()
    #save IMU packed data
    def save(self, packed_imu_data, m_b):
        message = packed_imu_data + m_b
        self.comm.communicate(message)
        self.data_file.write(message)
        self.data_file.flush()
```

A.7 Ground Communication

The *ground_communication.py* code transmits the packed IMU data through a wireless signal to the desired host IP. This piece of code does not need to be use for the system to work, however allows the user to plot real time data.

```
# inputs IMU data
# outputs UDP packet to remote host
import socket


class GroundCommunication:
  def __init__(self):
    self.RECEIVER_HOST = '192.168.1.100'    # The remote host
    self.PORT =  3000 # The same port as used by the server


  def open(self):
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    #self.sock.connect((self.RECEIVER_HOST, self.PORT))


  def close(self):
    self.sock.close()


  def communicate(self, data):
    try:
      self.sock.sendto(data, (self.RECEIVER_HOST, self.PORT))
    except Exception, err:
      pass
```

A.8 Convert Gyro Test

The *convert_avionics$_d$ata$_t$o$_c$sv.py* code takes the packed IMU data and unpackes it to another file called *nSAT_data.csv*. It also gives headers to all of the columns containing data. This code only supports the main gyro function.

```
# inputs avionics_data.dat
# outputs nSAT_data.csv
```

```
import csv
import struct

writer = csv.writer(open('nSAT_data.csv', 'wb'))

#headers
writer.writerow( ['','Accel_X', 'Accel_Y', 'Accel_Z',
                'AngRate_X','AngRate_Y', 'AngRate_Z', 'Mag_X',
                'Mag_Y', 'Mag_Z','timer', 'checksum'] )

data = open('avionics_data.dat', 'rb')

#reads first 43 bytes
datum = data.read(43)

#writes data and continutes to read next 43 bytes until done
while data:
  numbers = struct.unpack('>cfffffffffIH', datum )
  writer.writerow(numbers)
  datum = data.read(43)
```

A.9 Convert Controls Test

This *convert_test.py* code takes the packed IMU data and unpackes it to another file called *nSAT_data.csv*. It also gives headers to all of the columns containing data. This code only supports the main control function.

```
import csv
import struct
```

```
writer = csv.writer(open('nSAT_data1.csv', 'wb'))

writer.writerow( ['','Accel_X', 'Accel_Y', 'Accel_Z',
                'AngRate_X','AngRate_Y', 'AngRate_Z', 'Mag_X',
                'Mag_Y', 'Mag_Z', 'M11','M12', 'M13', 'M21',
                'M22', 'M23', 'M31', 'M32', 'M33', 'timer',
                'checksum'] )

data = open('avionics_data.dat', 'rb')

datum = data.read(79)

while data:
  numbers = struct.unpack('>cffffffffffffffffffIH', datum )
  writer.writerow(numbers)
  datum = data.read(79)
```

Appendix B

Gumstix Bourne Shell Programming Code

All of the below code is installed onto the Gumstix and has had the permissions changed to run as executables using *chmod+xname_of_flie*. Most of the code has also been installed into startup commands. All scripts can be run by logging into the Gumstix using putty.ece and typing the following command *./name_of_script*.

B.1 GPIO

The *gpio.sh* script initializes all of the gpio pins used for controlling the annular flow and secondary injection valves. This code runs in the Gumstix startup commands.

```
# initialize gpio173
# gpio173 found at 0x480021cc h
# find value of this address to be 0x100
# need change to the following value 0x10c
# white (main) = 173
# blue () = 147
# yellow () = 146

devmem2 0x480021cc h 0x10c

echo 173 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio173/direction
echo 0 > /sys/class/gpio/gpio173/value
```

```
# initialize gpio146
echo 146 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio146/direction
echo 0 > /sys/class/gpio/gpio146/value

# initialize gpio 147
echo 147 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio147/direction
echo 0 > /sys/class/gpio/gpio147/value
```

B.2 Off

The *off.sh* script turns all gpio pins off.

```
echo 0 > /sys/class/gpio/gpio173/value
echo 0 > /sys/class/gpio/gpio146/value
echo 0 > /sys/class/gpio/gpio147/value
```

B.3 On

The *on.sh* script turns all gpio pins on.

```
echo 1 > /sys/class/gpio/gpio173/value
echo 1 > /sys/class/gpio/gpio146/value
echo 1 > /sys/class/gpio/gpio147/value
```

B.4 Wireless

The *wireless.sh* script turns on the wireless communications on the Gumstix and connects to the smallsat router. This code runs in the Gumstix startup commands.

```
ifconfig wlan0 up
iwlist wlan0 scan
iwconfig wlan0 essid nSAT key d1be867212
dhclient wlan0
```

B.5 Testing

The *testing.sh* script runs through a series of pulses to verify that all valves are in working condition.

```
# white (main) = 173
# blue () = 147
# yellow () = 146
./off.sh
sleep 2
echo 1 > /sys/class/gpio/gpio173/value
sleep 10
echo 1 > /sys/class/gpio/gpio146/value
sleep .5
echo 0 > /sys/class/gpio/gpio146/value
sleep .25
echo 1 > /sys/class/gpio/gpio146/value
sleep .25
echo 0 > /sys/class/gpio/gpio146/value
sleep .05
echo 1 > /sys/class/gpio/gpio146/value
sleep .05
echo 0 > /sys/class/gpio/gpio146/value
sleep 10
echo 0 > /sys/class/gpio/gpio173/value
echo 0 > /sys/class/gpio/gpio146/value
echo 0 > /sys/class/gpio/gpio147/value
```

B.6 Run Avionics.sh

The *run_avionics.sh* script starts all testing by running the *main.py* code.

```
#!/usr/bin/env sh


python avionics/main.py &
```

Appendix C
MatLab Parsing Code

The MatLab code is used for parsing and plotting data collected from the IMU and LabVIEW file extensions. This code is specific to the tests and described in further detail below.

C.1 Compare Gyro Rate

The *compare.m* code takes all of the IMU data from the selected tests and plots the angular rate onto a single plot.

```
% Same Plot comparison for Force, Pressure, Temp data


% Read in LabView data
% For LabView Plots
clc
close all
clear all
addpath data


figuretype = 'eps';


%%% -- annular flow -- %%%%%%
infiles{1}='test31.lvm';  infiles2{1}='test31.csv'; %was 31
infiles{end+1}='test34.lvm';infiles2{end+1}='test34.csv';
infiles{end+1}='test36.lvm';infiles2{end+1}='test36.csv';
infiles{end+1}='test37.lvm';infiles2{end+1}='test37.csv';
```

```
infiles{end+1}='test38.lvm'; infiles2{end+1}='test38.csv';
infiles{end+1}='test39.lvm'; infiles2{end+1}='test39.csv';
infiles{end+1}='test40.lvm'; infiles2{end+1}='test40.csv';
infiles{end+1}='test41.lvm'; infiles2{end+1}='test41.csv';
infiles{end+1}='test42.lvm'; infiles2{end+1}='test42.csv';
infiles{end+1}='test43.lvm'; infiles2{end+1}='test43.csv';
infiles{end+1}='test44.lvm'; infiles2{end+1}='test44.csv';
infiles{end+1}='test45.lvm'; infiles2{end+1}='test45.csv';
infiles{end+1}='test46.lvm'; infiles2{end+1}='test46.csv';
plotlength = 19000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% -- secondary only -- %%%%
% infiles{1}='test50.lvm'; infiles2{1}='test50.csv';
% infiles{end+1}='test51.lvm'; infiles2{end+1}='test51.csv';
% infiles{end+1}='test52.lvm'; infiles2{end+1}='test52.csv';
% infiles{end+1}='test53.lvm'; infiles2{end+1}='test53.csv';
% infiles{end+1}='test54.lvm'; infiles2{end+1}='test54.csv';
% infiles{end+1}='test55.lvm'; infiles2{end+1}='test55.csv';
% infiles{end+1}='test56.lvm'; infiles2{end+1}='test56.csv';
% infiles{end+1}='test57.lvm'; infiles2{end+1}='test57.csv';
% plotlength = 15000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


timeof = length(infiles);
colors = jet(45);
[b,a] = butter(2,0.002,'low'); % butterworth filter
```

```
for num = 1:1:length(infiles)

    file = infiles{num};
    fid = fopen(file,'rt');

    fmt = '';
    nChan = 23;
    for i = 1:nChan
        fmt = [fmt '%s\t'];
    end

    nSkip = 23;
    for i = 1:nSkip
        str = fgetl(fid);
    end
    names = regexp(str,'\t','split');
    Tmp = importdata(file,'\t',nSkip);

    %Pare Time Data
    t = Tmp.data(:,1);
    %Parse Temperature Data
        %temp range -40 to 125C (41.25 C/V)
    temp_regulator = ((Tmp.data(:,3)-1).*41.25)-40;
        %temp range -40 to 125C (41.25 C/V)
    temp_plenum = ((Tmp.data(:,5)-1).*41.25)-40;
        %temp range -40 to 125C (41.25 C/V)
    temp_manifold = ((Tmp.data(:,7)-1).*41.25)-40;
```

```
%Parse Pressure Data
    %pressure range 0 to 500 (125 psi/V)
pressure_regulator = (Tmp.data(:,2)-1).*125+15.13;
    %pressure range 0 to 500 (125 psi/V)
pressure_plenum = (Tmp.data(:,4)-1).*125+15.13;
    %pressure range 0 to 1500 (375 psi/V)
pressure_manifold = (Tmp.data(:,6)-1).*375+15.13;
%Parse Load Cell Data
offset = Tmp.data(1,10)*3125;
load_cell = (Tmp.data(:,10).*3125)-offset;
load_cell = filter(b,a,load_cell);

figure(2)
subplot(4,1,1);
hold on
plot(t(1:plotlength),(temp_regulator(1:plotlength)),
            'Color',colors(num*3,:))

figure(1)
subplot(4,1,2);
hold on
plot(t(1:plotlength),(temp_plenum(1:plotlength)),
            'Color',colors(num*3,:))

figure(2)
subplot(4,1,3);
hold on
plot(t(1:plotlength),(temp_manifold(1:plotlength)),
                'Color', colors(num*3,:))

    figure(2)
    subplot(4,1,2);
    hold on
    plot(t(1:plotlength),(pressure_regulator(1:plotlength)),
                'Color', colors(num*3,:))

    figure(1)
    subplot(4,1,3);
    hold on
    plot(t(1:plotlength),(pressure_plenum(1:plotlength)),
                'Color', colors(num*3,:))

    figure(2)
    subplot(4,1,4);
    hold on
    plot(t(1:plotlength),(pressure_manifold(1:plotlength)),
                'Color', colors(num*3,:))

    figure(1)
    subplot(4,1,4);
    hold on
    plot(t(1:plotlength),(load_cell(1:plotlength)),
                'Color', colors(num*3,:))


end
```

```
h = figure(1);
subplot(4,1,2);
ylabel('Temp Plenum (C)');
xlabel('Time (s)');
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(1);
subplot(4,1,3);
ylabel('Pressure Plenum (Psi)');
xlabel('Time (s)');
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(1);
subplot(4,1,4);
ylabel('Load Cell (lbs)');
xlabel('Time (s)');
grid on
%legend([infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)


h = figure(2);
subplot(4,1,1);
ylabel('Temp Regulator (C)');
xlabel('Time (s)');
```

```
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(2);
subplot(4,1,2);
ylabel('Pressure␣Regulator␣(C)');
xlabel('Time␣(s)');
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(2);
subplot(4,1,3);
ylabel('Temp␣Manifold␣(Psi)');
xlabel('Time␣(s)');
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(2);
subplot(4,1,4);
ylabel('Pressure␣Manifold␣(lbs)');
xlabel('Time␣(s)');
grid on
%legend([infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)

[b,a] = butter(2,.01,'low'); % butterworth filter

for num = 1:1:length(infiles)
```

```
    file = infiles2{num};
    DELIMITER = ',';
    HEADERLINES = 1;
    newData1 = importdata(file, DELIMITER, HEADERLINES);
    vars = fieldnames(newData1);

    for i = 1:length(vars)
        assignin('base', vars{i}, newData1.(vars{i}));
    end

    %Parse IMU data I want to look at
    %AngRate_z = filter(b,a,data(:,6));
    %magX = data(:,7);
    M11 = data(:,10);
    M12 = data(:,11);
    yaw = atan2(M12,M11);
    %convert time to seconds and offset to 0
    time_offset = data(1,10)/62500;
    time = (data(:,10)./62500)-time_offset;
    timeof(num) = time(end);

    figure(1)
    subplot(4,1,1);
    %plot(time,(AngRate_z*-1),'Color',colors(num*3,:))
    plot(time,(yaw),'Color',colors(num*3,:))
    hold on
end
```

```
h = figure(1);
subplot(4,1,1);
ylabel('Angular␣Rate␣(rad/s)');
xlabel('Time␣(s)');
grid on
for num = 1:1:length(infiles)
    legendof{num} = ([infiles2{num},'␣time:' ,
        num2str(timeof(num))]);
end
%legend([legendof, 'Location', 'Best']);
saveas(h,'figures/AngularRate',figuretype)
```

C.2 Compare Gyro Sensors

The *mergeandplot.m* code takes all of the saved data from the selected tests and plots the temperatures, pressures, and angular rates for the gyro based tests.

```
% Read in IMU data from multiple Ubuntu unpacked files
% For Angular Rate

clc
close all
clear all
addpath data
figuretype = 'eps';

%%% —— annular flow —— %%%%%%
% infiles{1}='test31.csv';
% infiles{end+1}='test34.csv';
% infiles{end+1}='test36.csv';
% infiles{end+1}='test37.csv';
% infiles{end+1}='test38.csv';
% infiles{end+1}='test39.csv';
% infiles{end+1}='test40.csv';
% infiles{end+1}='test41.csv';
% infiles{end+1}='test42.csv';
% infiles{end+1}='test43.csv';
% infiles{end+1}='test44.csv';
% infiles{end+1}='test45.csv';
% infiles{end+1}='test46.csv';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%% -- secondary only -- %%%%
infiles{1}='test50.csv';
infiles{end+1}='test51.csv';
infiles{end+1}='test52.csv';
infiles{end+1}='test53.csv';
infiles{end+1}='test54.csv';
infiles{end+1}='test55.csv';
infiles{end+1}='test56.csv';
infiles{end+1}='test57.csv';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


timeof = length(infiles);
colors = jet(45);
[b,a] = butter(2,.01,'low'); % butterworth filter
```

```
%[b,a] = butter(2,.91,'low'); % butterworth filter

for num = 1:1:length(infiles)
    file = infiles{num};
    DELIMITER = ',';
    HEADERLINES = 1;
    newData1 = importdata(file, DELIMITER, HEADERLINES);
    vars = fieldnames(newData1);

    for i = 1:length(vars)
        assignin('base', vars{i}, newData1.(vars{i}));
    end

    %Parse IMU data I want to look at
    AngRate_z = filter(b,a,data(:,6));
    %convert time to seconds and offset to 0
    time_offset = data(1,10)/62500;
    time = (data(:,10)./62500)-time_offset;
    timeof(num) = time(end);

    figure(1)
    plot(time,(AngRate_z*-1),'Color',colors(num*3,:))
    hold on
end

h = figure(1);
ylabel('Angular Rate (rad/s)');
xlabel('Time (s)');
```

```
grid on
for num = 1:1:length(infiles)
    legendof{num} = ([infiles{num},'␣time:' ,
        num2str(timeof(num))]);
end
legend([legendof, 'Location', 'Best']);
saveas(h,'figures/AngularRate',figuretype)
```

C.3 Oscillation

The *oscillation.m* code takes the data from the oscillation tests and plots the position along with the last FFT of the data set.

```
% import oscillation file to find spring constant
clear all
close all
clc

addpath springdata

infiles{1} = 'oscillation3.csv';
% infiles{end+1}='oscillation4.csv';
% infiles{end+1}='oscillation5.csv';
% infiles{end+1}='oscillation6.csv';
% infiles{end+1}='oscillation7.csv';
% infiles{end+1}='oscillation8.csv';
% infiles{end+1}='oscillation9.csv';
% infiles{end+1}='oscillation10.csv';

colors = jet(27);
```

```
for num = 1:1:length(infiles)
    file = infiles{num};
    % Import the file
    newData1 = importdata(file);

    % Create new variables in the base workspace from
        % those fields.
    vars = fieldnames(newData1);
    for i = 1:length(vars)
        assignin('base', vars{i}, newData1.(vars{i}));
    end

    yaw_hold = atan2(data(:,11),data(:,10));
    yaw_mean = mean(yaw_hold);
    yaw = yaw_hold-yaw_mean;
    starttime = data(1,19)/62500;
    time = data(:,19)./62500-starttime;

    figure(1)
    hold on
    plot(time,yaw,'Color',colors(num*3,:))

    Fs = length(time)/max(time);   % Sampling frequency
    T = 1/Fs;                       % Sample time
    NFFT = 2^nextpow2(length(time));% 2 pwr from length y
    Y = fft(yaw,NFFT)/length(time);
    f = Fs/2*linspace(0,1,NFFT/2+1);
    % Plot single-sided amplitude spectrum.
    figure(2)
    hold on
    plot(f,2*abs(Y(1:NFFT/2+1)),'Color',colors(num*3,:))
    fft = 2*abs(Y(1:NFFT/2+1));
    freq(num) = f(find(fft==max(fft)));



end


h = figure(1);
hold on; plot(.26*exp(-0.2417*time*.46),'r')
ylabel('Radians (rad)');
xlabel('Time (s)');
grid on
legend([infiles, 'Location', 'Best']);

h = figure(2);
title('Single-Sided Amplitude Spectrum of y(t)')
xlabel('Frequency (Hz)')
ylabel('|Y(f)|')
grid on
for num = 1:1:length(infiles)
    legendof{num} = ([infiles{num},' Natural Freq:' ,
        num2str(freq(num))]);
end
legend([legendof, 'Location', 'Best']);
```

C.4 Compare Controlled Angular Position

The *compare_control.m* code takes all of the IMU data from the selected tests and plots the angular rate and position.

```
clc
close all
clear all
addpath data
%figuretype = 'eps';



%% -- secondary only -- %%%%%%
% infiles{1}='control_test61.csv';
% infiles{end+1}='control_test62.csv';
% infiles{end+1}='control_test63.csv';
% infiles{end+1}='control_test64.csv';
% infiles{end+1}='control_test65.csv';
% infiles{end+1}='control_test66.csv';
% infiles{end+1}='control_test67.csv';
% infiles{end+1}='control_test68.csv';
% infiles{end+1}='control_test69.csv';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% -- annular only -- %%%%%%
infiles{1}='control_test81.csv';
infiles{end+1}='control_test82.csv';
infiles{end+1}='control_test83.csv';
% infiles{end+1}='control_test84.csv';
% infiles{end+1}='control_test85.csv';
% infiles{end+1}='control_test86.csv';
% infiles{end+1}='control_test87.csv';
% infiles{end+1}='control_test88.csv';
infiles{end+1}='control_test71.csv';
infiles{end+1}='control_test72.csv';
infiles{end+1}='control_test73.csv';
infiles{end+1}='control_test76.csv';
infiles{end+1}='control_test77.csv';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

timeof = length(infiles);
colors = jet(45);
%[b,a] = butter(2,.01,'low');% butterworth filter
[b,a] = butter(2,.91,'low'); % butterworth filter

for num = 1:1:length(infiles)
    file = infiles{num};
    DELIMITER = ',';
    HEADERLINES = 1;
    newData1 = importdata(file, DELIMITER, HEADERLINES);
    vars = fieldnames(newData1);

    for i = 1:length(vars)
        assignin('base', vars{i}, newData1.(vars{i}));
    end

    %Parse IMU data I want to look at
    magX = data(:,7);
    M11 = data(:,10);
    M12 = data(:,11);
    yaw = atan2(M12,M11);
    angrateX = data(:,6);
    %convert time to seconds and offset to 0
    time_offset = data(1,19)/62500;
    time = (data(:,19)./62500)-time_offset;
    timeof(num) = time(end);

    figure(1)
    plot(time,(yaw),'Color',colors(num*3,:))
    hold on
    figure(2)
    plot(time,angrateX,'Color',colors(num*3,:))
    hold on
end

h = figure(1);
ylabel('Angular␣Position␣(rad)');
xlabel('Time␣(s)');
grid on
for num = 1:1:length(infiles)
    legendof{num} = ([infiles{num},'␣time:' ,
        num2str(timeof(num))]);
end
legend([legendof, 'Location', 'Best']);
%saveas(h,'figures/AngularRate',figuretype)
```

C.5 Compare Controlled Sensors

The *control_mergeandplot.m* code takes all of the saved data from the selected tests and plots the temperatures, pressures, and angular positions for the control based tests.

```
% Same Plot comparison for Force, Pressure, Temp data


% Read in LabView data
% For LabView Plots
clc
close all
clear all
addpath data

figuretype = 'eps';

%%%%-- control secondary only -- %%%%
% infiles{1}='control_test61.lvm';
% infiles2{1}='control_test61.csv';
% infiles{end+1}='control_test62.lvm';
% infiles2{end+1}='control_test62.csv';
% infiles{end+1}='control_test63.lvm';
% infiles2{end+1}='control_test63.csv';
% infiles{end+1}='control_test64.lvm';
% infiles2{end+1}='control_test64.csv';
% infiles{end+1}='control_test65.lvm';
% infiles2{end+1}='control_test65.csv';
% infiles{end+1}='control_test66.lvm';
% infiles2{end+1}='control_test66.csv';
% infiles{end+1}='control_test67.lvm';
% infiles2{end+1}='control_test67.csv';
% infiles{end+1}='control_test68.lvm';
% infiles2{end+1}='control_test68.csv';
% infiles{end+1}='control_test69.lvm';
% infiles2{end+1}='control_test69.csv';
% plotlength = 30000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%-- control secondary only -- %%%%
infiles{1}='control_test81.lvm';
infiles2{1}='control_test81.csv';
infiles{end+1}='control_test82.lvm';
infiles2{end+1}='control_test82.csv';
infiles{end+1}='control_test83.lvm';
infiles2{end+1}='control_test83.csv';
% infiles{end+1}='control_test84.lvm';
% infiles2{end+1}='control_test84.csv';
% infiles{end+1}='control_test85.lvm';
% infiles2{end+1}='control_test85.csv';
% infiles{end+1}='control_test86.lvm';
% infiles2{end+1}='control_test86.csv';
% infiles{end+1}='control_test87.lvm';
% infiles2{end+1}='control_test87.csv';
% infiles{end+1}='control_test88.lvm';
% infiles2{end+1}='control_test88.csv';
infiles{end+1}='control_test71.lvm';
infiles2{end+1}='control_test71.csv';
infiles{end+1}='control_test72.lvm';
```

```
infiles2{end+1}='control_test72.csv';
infiles{end+1}='control_test73.lvm';
infiles2{end+1}='control_test73.csv';
infiles{end+1}='control_test76.lvm';
infiles2{end+1}='control_test76.csv';
infiles{end+1}='control_test77.lvm';
infiles2{end+1}='control_test77.csv';
plotlength = 30000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

timeof = length(infiles);
colors = jet(45);
[b,a] = butter(2,0.9,'low'); % butterworth filter

for num = 1:1:length(infiles)

    file = infiles{num};
    fid = fopen(file,'rt');

    fmt = '';
    nChan = 23;
    for i = 1:nChan
        fmt = [fmt '%s\t'];
    end

    nSkip = 23;
    for i = 1:nSkip
        str = fgetl(fid);
```

```
end
names = regexp(str,'\t','split');
Tmp = importdata(file,'\t',nSkip);

%Pare Time Data
t = Tmp.data(:,1);
%Parse Temperature Data
    %temp range -40 to 125C (41.25 C/V)
temp_regulator = ((Tmp.data(:,3)-1).*41.25)-40;
    %temp range -40 to 125C (41.25 C/V)
temp_plenum = ((Tmp.data(:,5)-1).*41.25)-40;
    %temp range -40 to 125C (41.25 C/V)
temp_manifold = ((Tmp.data(:,7)-1).*41.25)-40;
%Parse Pressure Data
    %pressure range 0 to 500 (125 psi/V)
pressure_regulator = (Tmp.data(:,2)-1).*125+15.13;
    %pressure range 0 to 500 (125 psi/V)
pressure_plenum = (Tmp.data(:,4)-1).*125+15.13;
    %pressure range 0 to 1500 (375 psi/V)
pressure_manifold = (Tmp.data(:,6)-1).*375+15.13;
%Parse Load Cell Data
offset = Tmp.data(1,10)*3125;
load_cell = (Tmp.data(:,10).*3125)-offset;
load_cell = filter(b,a,load_cell);

figure(2)
subplot(4,1,1);
hold on
```

```
plot(t(1:plotlength),(temp_regulator(1:plotlength)),
          'Color',colors(num*3,:))

figure(1)
subplot(4,1,2);
hold on
plot(t(1:plotlength),(temp_plenum(1:plotlength)),
          'Color',colors(num*3,:))

figure(2)
subplot(4,1,3);
hold on
plot(t(1:plotlength),(temp_manifold(1:plotlength)),
          'Color',colors(num*3,:))

figure(2)
subplot(4,1,2);
hold on
plot(t(1:plotlength),(pressure_regulator(1:plotlength)),
          'Color',colors(num*3,:))

figure(1)
subplot(4,1,3);
hold on
plot(t(1:plotlength),(pressure_plenum(1:plotlength)),
          'Color',colors(num*3,:))

figure(2)
```

```
    subplot(4,1,4);
    hold on
    plot(t(1:plotlength),(pressure_manifold(1:plotlength)),
                'Color',colors(num*3,:))

    figure(1)
    subplot(4,1,4);
    hold on
    plot(t(1:plotlength),(load_cell(1:plotlength)),
                'Color',colors(num*3,:))


end

h = figure(1);
subplot(4,1,2);
ylabel('Temp_Plenum_(C)');
xlabel('Time_(s)');
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(1);
subplot(4,1,3);
ylabel('Pressure_Plenum_(Psi)');
xlabel('Time_(s)');
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
```

```
h = figure(1);
subplot(4,1,4);
ylabel('Load Cell (lbs)');
xlabel('Time (s)');
grid on
%legend([infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)



h = figure(2);
subplot(4,1,1);
ylabel('Temp Regulator (C)');
xlabel('Time (s)');
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(2);
subplot(4,1,2);
ylabel('Pressure Regulator (C)');
xlabel('Time (s)');
grid on
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(2);
subplot(4,1,3);
ylabel('Temp Manifold (Psi)');
xlabel('Time (s)');
grid on
```

```
%legend([infiles, infiles, infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)
h = figure(2);
subplot(4,1,4);
ylabel('Pressure␣Manifold␣(lbs)');
xlabel('Time␣(s)');
grid on
%legend([infiles, 'Location', 'Best']);
%saveas(h,'figures/LoadCell',figuretype)

[b,a] = butter(2,.001,'low'); % butterworth filter

for num = 1:1:length(infiles)
    file = infiles2{num};
    DELIMITER = ',';
    HEADERLINES = 1;
    newData1 = importdata(file, DELIMITER, HEADERLINES);
    vars = fieldnames(newData1);

    for i = 1:length(vars)
        assignin('base', vars{i}, newData1.(vars{i}));
    end

    %Parse IMU data I want to look at
    magX = data(:,7);
    M11 = data(:,10);
    M12 = data(:,11);
    yaw = atan2(M12,M11);
    %convert time to seconds and offset to 0
    time_offset = data(1,19)/62500;
    time = (data(:,19)./62500)-time_offset;
    timeof(num) = time(end);

    figure(1)
    subplot(4,1,1);
    plot(time,(yaw),'Color',colors(num*3,:))
    hold on
end

h = figure(1);
subplot(4,1,1);
ylabel('Angular␣Position␣(rad)');
xlabel('Time␣(s)');
grid on
for num = 1:1:length(infiles)
    legendof{num} = ([infiles2{num},'␣time:' ,
        num2str(timeof(num))]);
end
%legend([legendof, 'Location', 'Best']);
saveas(h,'figures/AngularRate',figuretype)
```