7-2015

Design and Validation of Hardware-in-the-Loop Testbed for Proximity Operations Payloads

Kristia K. Harris

DESIGN AND VALIDATION OF A HARDWARE-IN-THE-LOOP TESTBED FOR PROXIMITY OPERATIONS PAYLOADS

A Thesis

Submitted to the Faculty

of

Embry-Riddle Aeronautical University

by

Harris, Kristia K.

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Mechanical Engineering

July 2015

Embry-Riddle Aeronautical University

Daytona Beach, Florida

DESIGN AND VALIDATION OF A HARDWARE-IN-THE-LOOP TESTBED FOR PROXIMITY OPERATIONS PAYLOADS

by

Harris, Kristia K.

A Thesis prepared under the direction of the candidate's committee chairman, Dr. Bogdan Udrea, Department of Aerospace Engineering, and has been approved by the members of the thesis committee. It was submitted to the School of Graduate Studies and Research and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering.

THESIS COMMITTEE

Chairman, Dr. Bogdan Udrea

Member, Dr. Troy Henderson

Member, Dr. Heidi Steinhauer

Department Chair, Dr. Jean-Michael Dhainaut
or Graduate Program Coordinator, Dr. Charles F. Reinholtz

7/29/15
Date

Dean of College of Engineering, Dr. Maj Mirmirani

8/4/15
Date

Associate VP for Academics, Dr. Mohamed S. Camara

8/6/15
Date

ACKNOWLEDGMENTS

I would like to give a special thanks to Dr. Dhainaut and Dr. White, the graduate program coordinators during my time in the program, for their assistance in navigating the program. I would like to thank the ARAPAIMA team and the OMATID team for all of their hard work and dedication to the mission. In addition, I would like to thank them for all of their assistance in testing as well as being the inspiration for the research. I would like to thank the Embry-Riddle Ignite Program and specifically Caroline Day for the funding and assistance with acquiring parts for the initial research and concepts. I would also like to thank the Florida Space Grant Consortium for their funding and support of my research.

I would like to give a special thanks to my thesis advisor, Dr. Udrea. Without his guidance I would have never thought to pursue research, especially not in the realm of satellites that I did. I would also like to thank my committee members, Dr. Henderson and Dr. Steinhauer for all of their assistance on the back end of my graduate work.

Last, but not least, I would like to thank my parents. Without their support I would never be where I am today.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ADCS	attitude determination and control system
AFRL	Air Force Research Laboratory
AIL	algorithm in the loop
ARAPAIMA	Application for RSO Proximity Analysis and IMAging
CAD	computer-aided design
CMOS	complementary metal-oxide semiconductor
CNC	computer numerical control
COTS	commercial off-the-shelf
CPU	central processing unit
CSD	Canisterized Satellite Dispenser
DARPA	Defense Advanced Research Projects Agency
DC	direct current
DOF	degrees of freedom
EDU	engineering design unit
EPOS	European Proximity Operations Simulator
ELaNa	Educational Launch of Nanosatellites
ELF	extremely low frequency
ERAU	Embry-Riddle Aeronautical University
ESA	European Space Agency
FOV	field of view
FREND	Front-end Robotics Enabling Near-term Demonstration
GaAs	gallium arsenide
GOT	Gravity Offset Test Bed
HIL	hardware in the loop
IDE	integrated development environment
IO	input/output
IR	infrared
ISIS	Innovative Solutions in Space
ISS	International Space Station
ITAR	International Trade and Arms Regulations
JAXA	Japan Aerospace Exploration Agency
LED	light-emitting diode
LEO	low Earth orbit
LRF	laser rangefinder
LV	launch vehicle
NASA	National Aeronautics and Space Administration
NEMA	National Electrical Manufacturers Association

NLAS	Nanosatellite Launch Adapter Systems
NRL	Naval Research Lab
OMATID	Opto-Mechanical Analysis Testing Integration and Development
PLA	polylactic acid
POT	Proximity Operations Test Bed
P-POD	Picosatelllite Orbital Deployer
PVC	polyvinyl chloride
PWM	pulse-width modulation
RMS	root mean square
RPO	rendezvous and proximity operations
RSO	resident space object
SDK	software development kit
SHOT	student hands-on training
SIL	software in the loop
STK	Systems Tool Kit
STL	stereolithography
SSPL	Space Shuttle Picosatellite Launcher
TCP	Transmission Control Protocol
TLE	two-line element
TLM	telemetry
UDP	User Datagram Protocol
UNP	University Nano-Satellite Program

ABSTRACT

Harris, Kristia K. MSME, Embry-Riddle Aeronautical University, July 2015. Design and Validation of a Hardware-in-the-Loop Testbed for Proximity Operations Payloads.

The research presented here is a new testbed design for CubeSat and payload testing and development. This research demonstrates a low-cost, hardware-in-the-loop testing apparatus for use with university CubeSat programs for testing throughout the different levels of the development process. The average university CubeSat program undergoes very little hardware-in-the-loop testing. Most of the focus is targeted towards performance testing and environmental testing which occur after the completion of the development process. This research shows that, for minimal schedule and cost impact, testing can occur early in the development process. The testbed presented here demonstrates suitable accuracy to be used for advanced mission testing and regularly throughout the process until completion. The testbed maintains a low-cost, modular design, and ease of integration into new and existing programs. In addition, some modifications and upgrades are suggested to further increase the performance of the testbed. The success of the testbed can be seen through the implementation of actual satellite telemetry with rendezvous and docking missions, the testbed performance, and the results of that experiment.

1. Introduction

1.1 Significance of the Study

The concept of CubeSats was developed from the need to have a standardized form factor for small satellites. The first batch of CubeSats was launched on June 30, 2003 from Plesetsk, Russia on an Eurockot launch vehicle. This batch included five CubeSats: CUTE-I (shown in Figure 1.1) by the Tokyo Institute of Technology, Japan; XI-IV by the University of Tokyo, Japan; CanX–1 by the University of Toronto, Canada; DTUsat by the Technical University of Denmark; AAU Cubesat by the Aalborg University, Denmark; and QuakeSat by Stanford University and Quakesat LLC, USA. These missions focused on demonstrating the CubeSat technology with the use of COTS components, space-testing technologies, orbit alteration with a tether, testing a color CMOS camera, and detecting the ELF radio emissions of seismic activity during earthquakes. From these missions, the trend had begun. The low cost of CubeSats allows for them to be designed and build by a more widespread community, with the bulk of the current community being universities. There have been over 98 universities, located all over the world, that have launched CubeSats (Adolphus & Jnr, 2003).

In addition to CubeSats being a low-cost satellite option, launch opportunities have been provided to many universities through programs such as ELANA, now NASA's

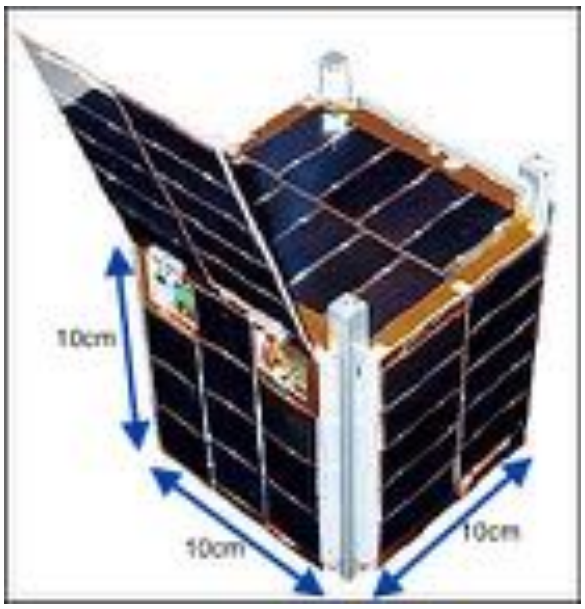


Figure 1.1. Image of the CUTE-I CubeSat by the Tokyo Institute of Technology (Adolphus & Jnr, 2003)

CubeSat Launch Initiative, and the AFRL UNP program, among other sources. In recent years, CubeSat programs have become widespread throughout the university. The draw of CubeSats is expansive to the university; there are numerous benefits to introducing a CubeSat program. One benefit is the hands-on experience that is achieved through the process of designing, building, and testing a CubeSat. An example of this hands-on work can be seen in Figure 1.2. In addition, many of the CubeSat design programs and competitions are designed to provide an environment that is very similar to the actual aerospace industry. Students participate in and lead the systems engineering, thhe design reviews, and the students are required to act and communicate on a professional level. As more and more universities are expanding and improving on their space programs and departments, the mission and design of the CubeSats produced by the universities have become increasingly advanced. Some

of the missions for CubeSats currently in development include satellite-to-satellite communication, hurricane observations, and studying the exosphere, among other technology and science missions. Universities are exploring completing missions that are normally done on a larger scale, in terms of budget and spacecraft size, in a much smaller and cheaper form factor. In addition, universities and companies alike are using the small size of the CubeSat to their advantage and inventing new missions that become much more of a reality in a smaller and more agile spacecraft. With all of the benefits that CubeSat programs have to offer, CubeSats have become a booming industry providing many new space missions and technologies to the aerospace industry and becoming a staple in the spacecraft community.

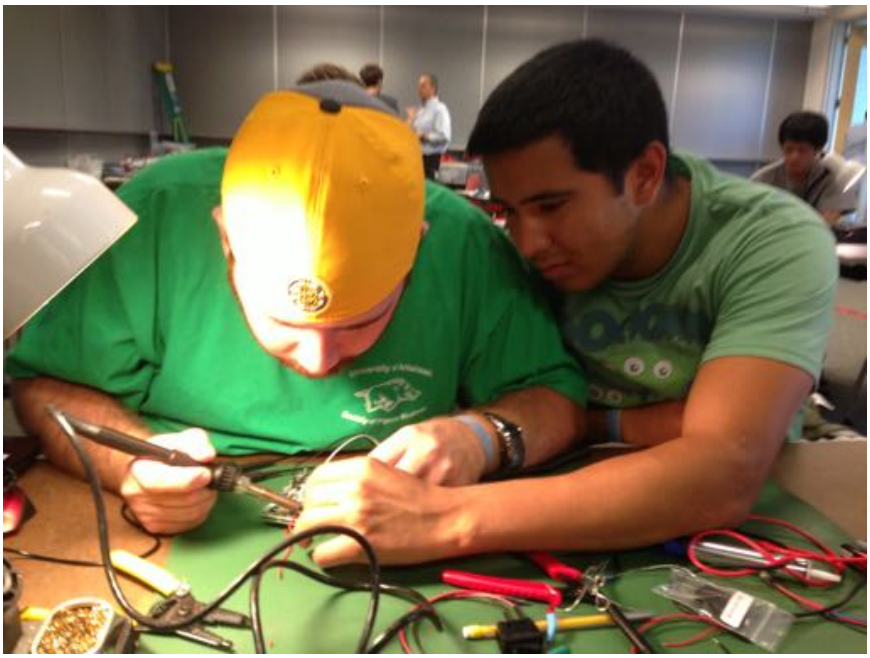


Figure 1.2. Students working together to solder a computer board at a UNP sponsored SHOT-I event

1.2 Statement of the Problem

CubeSats have become so advanced over the past few years that in some cases they can be used to replace actual satellites. This comes at a great advantage since gathering the same information or producing the same mission results allows sponsors to save money, conserve payload weight on the launch vehicle, and create smaller space debris. Although CubeSats provide multiple advantages, as CubeSats become more powerful and more complex they require more advanced planning and processes in order to be successful. Included in the additional required stages of development is additional testing on both the hardware and the software. This stage has the largest impact on universities.

Most of the current testing completed by universities is post engineering design unit (EDU) completion. Thus, the design is already locked and the hardware is purchased by the time that actual testing occurs. The concept and design are generally frozen by this stage and the investments have already been made. This process works if there are no fundamental problems with either the hardware or the software; however, if issues do arise it is often too late or very costly to fix. A much more streamlined approach to introducing testing into the university CubeSat development process is to be able to begin testing from an early stage of development. Ideally, testing would start when the design reaches a critical design review level. This would allow testing to occur at around the same time as the design is becoming more mature and concrete. Another improvement would be to test hardware and software together from the start

of their development. Currently, this is near impossible for universities to achieve due to the cost and availability of testing equipment and facilities. However, the benefits from the early testing of hardware and software interfacing are incalculable.

1.3 Purpose Statement

The purpose of this study is to solve the problem of early development testing, especially HIL testing, being difficult for universities to integrate into their development process. In addition, the simplification of testing hardware and software both individually and integrated will also be approached in this study.

The idea is that an apparatus can be designed that allows for universities to be able to start hardware-in-the-loop testing early in the development process. In addition, hardware and software will be able to be tested together and therefore developed in parallel. This will allow for many hardware and software conflicts to be identified and verification of requirements to occur early through testing.

The planned solution to accomplish the purpose of this study is to create a testing apparatus that can be widely used by universities. To ensure that this apparatus can be easily accessed and implemented by universities of all sizes and endowments, the testing apparatus must have a few key features. It is the belief of the author that these key features include maintaining a low cost, a small form factor, it must be simple to use, and easy to implement. In addition, the testing apparatus must be universal enough to serve as a platform for many different programs. As an applicability to the ARAPAIMA mission testing, the testbed shall be capable of moving the ARAPAIMA

payload or a payload of approximately the same mass and moment of inertia as the ARAPAIMA payload. Finally, the testing apparatus should be capable of performing HIL testing of a CubeSat payload and the accompanying algorithms for RPO. The goal of this study is to design, develop, and test an apparatus that meets all of these requirements.

1.4 Constraints

There are a few delimitations that must be adhered to in order to provide a product that maintains usability and accessibility to universities. The first limit discussed here is the cost of the testing apparatus. The cost limit is determined by two main constraints. The first of which is to maintain a cost that is deemed within easy reach of the majority of CubeSat programs. The second is limiting the cost of both the development and the cost to build the apparatus to the budget allotted by the main funding sources of this research.

The apparatus must be a reasonable size to fit in a typical laboratory used by CubeSat programs. This normally includes classrooms or small development labs as opposed to full buildings. The general goal in this case is to design an apparatus that is "desktop sized." Since the focus is on designing an apparatus for CubeSats, the apparatus can be scaled down accordingly.

The third constraint is designing an apparatus that is both easy to use and implement into programs. The goal is to have a design that is simple to integrate into not only the hardware, but also the software used for both the testing and the

flight software. Since the majority of universities use Matlab and Simulink to develop simulations and software, the software for the apparatus should also be in this fairly universal program for universities. This will allow for the program's pre-existing software to be easily ported over or called into the apparatus' software. Difficulty implementing or translating code to a different language is a common problem with many testing facilities that results in drastic schedule slippage.

These three items form the delimitations for the design of the testing apparatus. They also drive the base requirements for the concepts that can be developed further to create the apparatus.

1.5 Limitations and Assumptions

The largest limitations presented for this research are time and money. Due to the nature of the study, research in a broad number of topics is required. Due to the limited time available to produce results from this study, the author made the decision to obtain basic proficiency in each of the related topics instead of concentrating on mastery of just a few of the topics. In addition, due to the time required to order and ship parts, fewer iterations of the design were taken from the concept and modeling stage to the prototype stage. In an effort to save both time and money, rapid prototyping was used in place of machining in many instances, especially in the original prototypes that were made for proof of concept. Finally, the limited money available narrowed the available selection of components and ideas that could be explored.

It should also be noted that due to the size and cost constraints of the testing apparatus, the performance is expected to be limited as well. Due to the size and cost concerns, but especially the cost, it is expected that the performance in terms of accuracy will be limited due to the available parts in the cost range.

Some additional limitations are the author's knowledge of programming and robotics, along with access to supplies, software, and tools.

2. Review of Relevant Literature

The purpose of the literature review conducted here is to define the CubeSat class of spacecraft, demonstrate the importance of testing in space applications, and address the current spacecraft testing that is available. The literature review is a way to show the lack of available testing resources for small, university built satellites and its effect on the success of CubeSats. The research conducted hereinafter serves to provide the background information for the hypothesis of the research and demonstrate the importance of the research.

2.1 Overview of CubeSats

The CubeSat was developed as a "collaborative effort to continue developing the pico satellite, to provide a convenient low cost launch interface and coordinate launch activities" (Heidt, Puig-Suari, Moore, Nakasuka, & Twiggs, 2000). After many struggles with the early picosatellites, many lessons were learned, especially in the limitations of picosatellites. Due to the small size of the picosatellite, a modest 4 in x 3 in x 1in, power is a notable issue that commonly appears. The picosatellite is shown in Figure 2.1.

There was not enough available surface area on the picosatellite in order to power much more than the basic required electronics. The need to have enough available

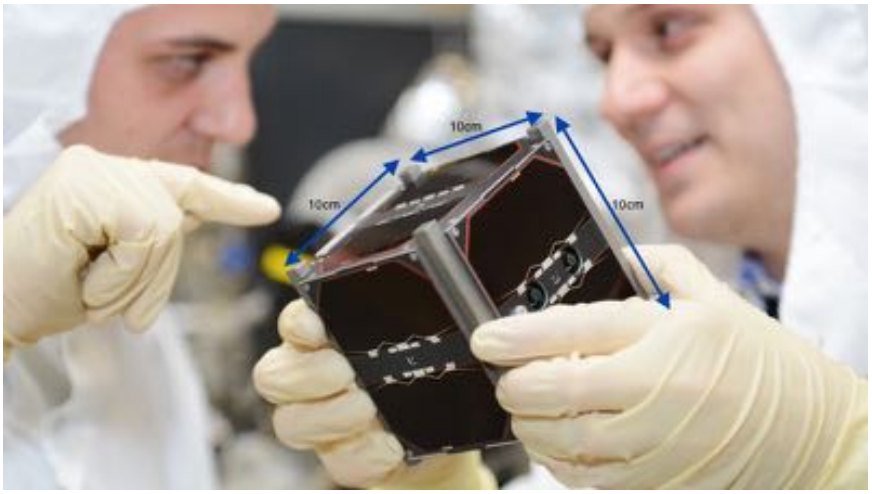


Figure 2.1. University of Würzburg's picosatellite UWE-3 being examined in a cleanroom (Bahr, 2013)

surface area for the solar cells to power the components along with the available components at the time outlines the main design decisions for the CubeSat.

Thus, the CubeSat standard was developed as a new small satellite concept by Stanford University as a cube with 4 in sides (Heidt et al., 2000). The first CubeSat was taken from a concept to reality in a collaboration between Stanford University and California Polytechnic State University (Cal Poly). This was the start of the "CubeSat approach," a CubeSat program designed to develop a market for launching picosatellites that are held to a specific standard (Toorian et al., 2008).

This standard is known as "The CubeSat specification." The CubeSat standard was developed as a product of The CubeSat Program by Cal Poly with a few key factors in mind. The first major factor was mass. Since launch costs are generally calculated by weight with the kilogram marking the unit of measurement, it was decided that a CubeSat would have a 1 kg mass. The 1U CubeSat, as it is now referred

to, has a maximum mass of 1kg and maximum dimensions of 10x10x10cm (Toorian et al., 2008). A exerpt of the dimensioned 1U CubeSat drawing is shown in Figure 2.2.

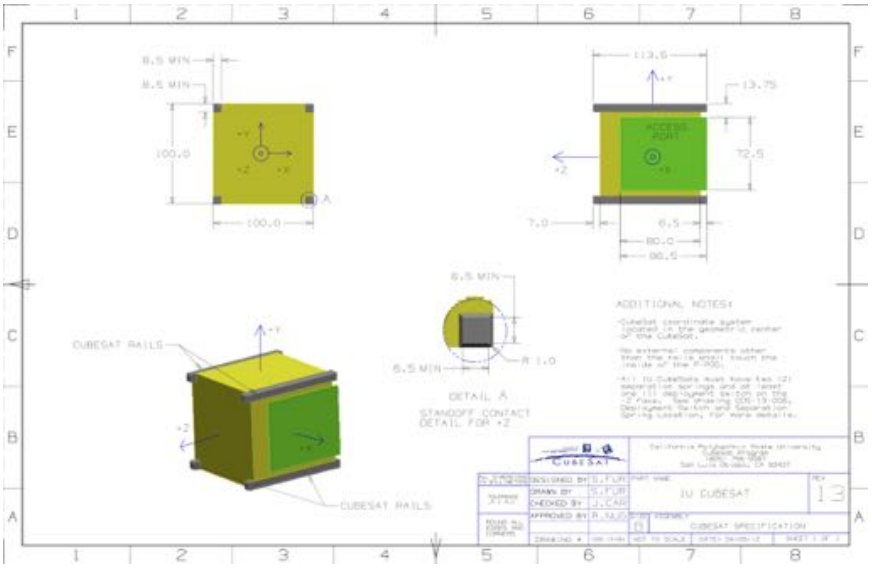


Figure 2.2. The dimensioned drawing of a 1U CubeSat from the CubeSat specification (Toorian et al., 2008)

Once these specifications became final, Cal Poly published the CubeSat Design Specification document which continues to be revised and updated to this day, with the last revision occurring in February 2014, marking revision 13. The document details everything including the interface with the launch container, the mechanical, electrical, and operational requirements for the CubeSat, the testing necessary for the CubeSat, and even a waiver process in case the CubeSat does not meet the requirements specified in the document (The CubeSat Program, 2009).

The CubeSat was eventually expanded to include several different classes of CubeSats, where each class is defined by the physical dimensions and the mass. There are currently four classes of CubeSats, the 1U, 2U, 3U, and 6U form factors. The

1U has been updated in more recent years to have dimensions of 10 cm x 10 cm x 11 cm with a mass of 1.33 kg (*CubeSat Launch Initiative Selectees*, 2013). This size scales with each form factor maintaining the idea that each U, or unit of space, is comparable to a 1U CubeSat. The 3U CubeSat consist of three 1U CubeSats and thus has dimensions of 10 cm x 10 cm x 33 cm with a maximum mass of 4 kg (rounded from 3.99 kg) and the 2U CubeSat has dimensions of 10 cm x 10 cm x 22 cm and a maximum mass of 2.66 kg. A variety of CubeSat sizes can be seen in Figure 2.3.

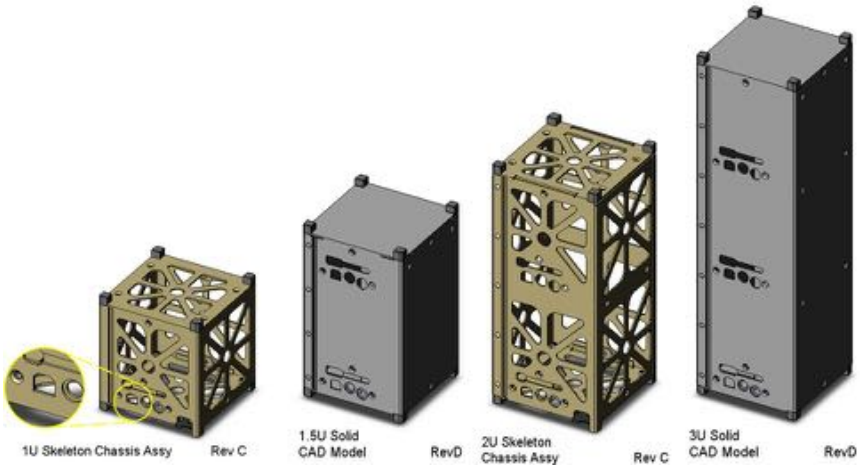


Figure 2.3. CubeSats in the 1U, 1.5U, 3U, and 3U form factors (Papadopoulos, 2014)

In more recent years, the CubeSat standard has been expanded to include a larger group of satellites. The new sizes of CubeSats added include the 6U, 12U, and 24U, which follow the same scaling form factor as the previous sizes.

CubeSat missions can be generalized into at least one of four categories. These categories include the technology class where the mission is the flight testing of components or subsystems, the science class where the mission is to gather or create

science data, the communications class where the mission is to provide communications services, and the educational class where the main mission is train and teach students. As of mid–2013, 1110 mission had flown with the vast majority of these CubeSats belonging to universities (Swartwout, 2013).

Over the years, especially in the time span since 2013, the missions of CubeSats have advanced rapidly along with the development processes for university CubeSats. Programs such as the University Nanosatellite Program headed by the Air Force Research Laboratory and NASA's ELaNa have allowed the university programs to expand and improve rapidly. These programs have guided students through the proper processes required in the concept, development, and testing phases of the CubeSat lifecycle.

2.2 Spacecraft Testing

When testing a spacecraft, the goal is to replicate the environment of space and the loads that the spacecraft will experience both in space and during launch as accurately as possible. Although a lot of money, time, and resources have been put into developing systems that replicate the space environment exactly, "economic, technological, and terrestrial limitations prevent [the] achievement of this objective" (New & Timmins, 1966). As a simple example, replicating the weightlessness of space is limited by the presence of gravity.

Despite these challenges, there are many tests and accompanying test facilities that have been designed to test spacecraft to the best of our abilities. There have

been facilities designed and built to "emulate the space environments particulate void, cold blackness, [and] solar radiation" (Outman, Wang, & Company, 1966). One such facility is the NRL Thermal Fabrication and Test Facility shown in Figure 2.4. These facilities can cost upwards of $40,000k; however this cost is small when compared to the amount of money saved through the problems identified in these facilities. The effect of a full round of testing on the success of a mission is astounding. At the Goddard test facilities alone, "16 prototype and 48 flight units [were evaluated]... [a] total of 855 problems have been uncovered and corrected, resulting in 27 successful satellites or probes and only one major failure ... [o]ver a four-year period" (New & Timmins, 1966).

The testing conducted in these facilities is generally referred to as protoflight testing, qualification testing, and environmental testing. The purpose of these tests is to verify that the spacecraft meets the requirements to survive both the launch and the space environment at different levels in the program. In order to best replicate the space environment, the following tests are generally completed: acceleration, vibration, shock, temperature cycling, and thermal-vacuum (Boeckel, 1963).

Although all large spacecraft undergo rigorous testing, very few CubeSats components are tested in space environmental conditions and even less CubeSats undergo full acceptance, protoflight, and qualification testing. This is the result of two main factors, the first of which is access to facilities capable of this type of testing and the second factor is that the orbit for a CubeSat is generally unknown until late in the development process (Corpino, 2014). As opposed to environmental testing, Cube-

Figure 2.4. The first MMS mission satellite being put into the thermal vaccuum test chamber at the NRL Thermal Fabrication and Test Facility (Parry, 2014)

Sats undergo functional testing at ambient conditions and greatly reduced functional environmental tests (Corpino, 2014).

As an addition to functional testing, many CubeSat developers have used an engineering system for the process of testing and integration. The stages of interactions between system elements are algorithm-in-the-loop, software-in-the-loop, and hardware-in-the-loop.

Hardware-in-the-loop (HIL) testing is a modeling and simulation method that combines computer simulations and hardware (Ledin, 1999). Applying this testing method to spacecraft allows for a greater understanding of how the components interface with each other in their flight configurations. HIL testing, unlike many other testing methods, does not require unique testing facilities to test components.

In addition, it is fairly simple to test both the software and the hardware through different phases of the mission, or operational modes (Corpino, 2014).

Unfortunately, there has not been a widespread application of HIL simulators in the CubeSat community. Although more HIL simulators are being designed for small satellites as compared to the rate of development in the past, these testbeds are typically designed for a sole purpose (Corpino, 2014). The vast majority of the simulators are designed to validate the attitude determination and control system (ADCS) for a particular satellite (Ure, Kaya, & Inalhan, 2011). Thus, the testbeds can almost be considered one-off testbeds. Once a satellite or series of similar satellites are completed, the simulator serves very little purpose since it was designed to be mission specific. In addition, there are no "off-the-shelf" HIL testbeds for proximity operations testing of neither CubeSats nor Nanosats.

2.3 Current Satellite Simulators

There are currently a few different HIL simulators in use today. These simulators are made by both industry companies and universities. One such simulator is the European Proximity Operations Simulator (EPOS) shown in Figure 2.5. This simulator belongs to the European Space Agency (ESA) and is currently operated by the German Space Operations Center. EPOS is a very large simulator, spanning larger than a typical room. It consists of two industrial robots onto which spacecraft mockups can be mounted. These robotic arms are capable of moving the spacecraft mockups in six degrees of freedom (DOF) independently. This allows EPOS to simulate relative

orientation between the two spacecraft. EPOS uses a combination of the Real-Time Operating System VxWorks and the Matlab/Simulink Real-Time Workshop to create the real-time control system used to operate the robots. Since EPOS's software uses this very specific real-time control, it requires that anyone seeking to use EPOS for HIL simulation spend the considerable time and effort required to modify their code to fit into EPOS's system (Rems, 2012).

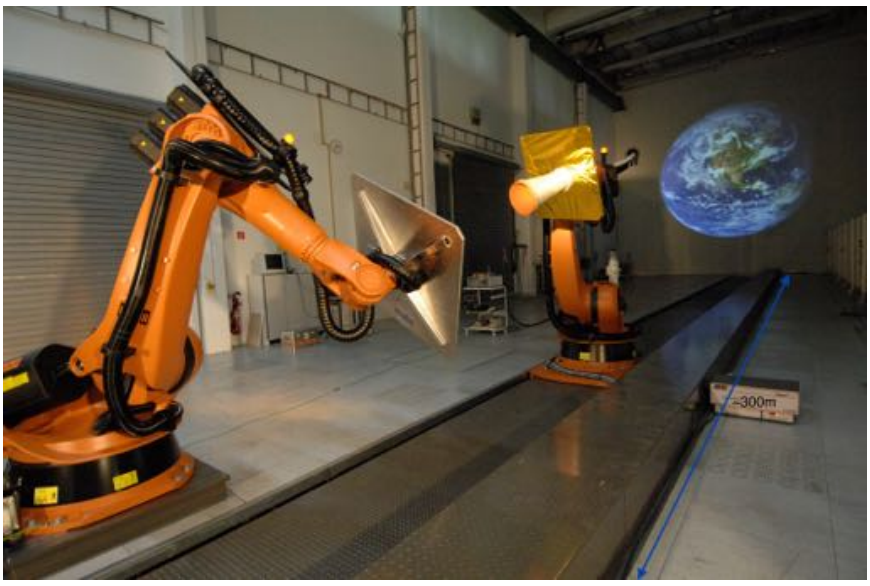


Figure 2.5. The EPOS belonging to ESA (Rems, 2012)

A simulator that is similar in nature to EPOS has also been designed and is currently operated through the United States Defense Advanced Research Projects Agency (DARPA) and the Naval Reseach Lab (NRL). This simulator is known as the Proximity Operations Test Bed (POT). POT is a dual platform motion simulator that performs spacecraft rendezvous and docking simulations with realistic dynamic conditions in three dimensions (Debus & Dougherty, 2009).

Also located to the NRL facility is the Front-end Robotics Enabling Near-term Demonstration (FREND). FREND features a robotic arm system developed by Alliance Spacesystems, which is shown in Figure 2.6. The robotic arm system uses gears and ball bearings along with other components that have a high stiffness to mass ratio in order to achieve a +/- 2 mm linear position accuracy. The software behind FREND is a spaceflight traceable control software (Debus & Dougherty, 2009). With these specifications, FREND has been able to successfully demonstrate the capability to use simulated orbital conditions to simulate autonomous rendezvous and docking in a ground test environment (Kelm et al., 2008).

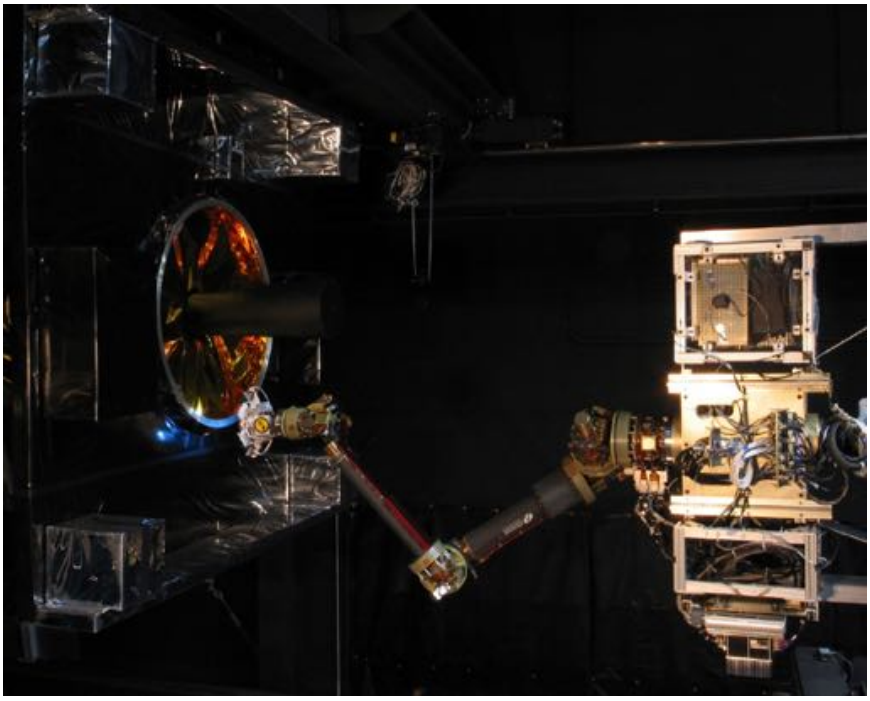


Figure 2.6. The FREND robotic arm system belonging to the NRL (Debus & Dougherty, 2009)

DARPA additionally has the Gravity Offset Test Bed (GOT) which is a three DOF air bearing facility. This simulator is specific to allowing "high fidelity contact dynamics simulation[s] to ensure [the] ability to grapple in simulated zero-G conditions" (Debus & Dougherty, 2009). Since this facility has such a specific use, it is typically used to work with the system integration and characterization of spacecraft with grapple related payloads and subsystems (Debus & Dougherty, 2009).

In addition to the simulators referenced here, large companies such as The Boeing Corporation and The Lockheed Martin Corporation have also been working on or have developed their own testing facilities for spacecraft which include HIL simulators.

Universities have also more recently moved into the area of research and design of HIL simulators. The Spacecraft Robotics Laboratory of the Naval Postgraduate School has developed a HIL architecture in order to simulate on-orbit docking between two spacecraft. While most of this system is simulated on a real-time computer, the target and chaser spacecraft are physically reproduced in the facility (Corpino, 2014). The Aerospace Engineering Department at Texas A&M university developed a six DOF autonomous mobile robotic system. In this system, "[a]n omni-directional robotic base provides unlimited 3-DOF planar motion with moderate precision, while a micron-class hexapod on top provides high precision, limited 6-DOF motion" (Doebbler, Davis, Valasek, & Junkins, 2008). This testbed features hardware-in-the loop sensors, processors, and docking hardware that allow for the closed-loop control algorithm to follow a generated trajectory in real-time (Doebbler et al., 2008). This testbed can be used both with and without the actual hardware.

Similar to the air bearing design of GOT, universities have also followed the air bearing trend in recent years. A prime example of this is Cornell University's FloatCube testbed shown in Figure 2.7. The FloatCube testbed is designed for CubeSat scale projects and technologies. FloatCube "provides a planar reduced-friction environment for multibody dynamics and controls technology development for spacecraft less than 6 kg and a 15 cm cube" (Wilson et al., 2013) making the testbed ideal for testing multibody dynamics for CubeSats. These FloatCubes have been applied to CubeSat technologies in order to "advance the maturity of a novel close-proximity spacecraft interface technology, validate performance for microgravity testing hardware, and examine the closed-loop multibody dynamics for the purposes of evaluating controller gain selection strategies and dynamic model validation" (Wilson et al., 2013). The FloatCube testbed also has the capability of testing docking and rendezvous maneuvers, along with formation flying for CubeSats at the system level (Wilson et al., 2013).

As stated by (Tsiotras, 2014), "A crucial element in advancing ARD and ProxOps state-of-the-art is the ability to test new relative navigation and guidance algorithms, grasping algorithms, vision processing and perception algorithms, etc that can support ProxOps in orbit." As opposed to other fields of study and development, it is comparatively very difficult to test algorithms and hardware in a realistic, or space-like, environment due to test availability, schedule, and cost (Tsiotras, 2014). It is for this reason that many testbeds have begun to be developed and tested in recent years. The concepts and missions of spacecraft, especially CubeSats are becoming

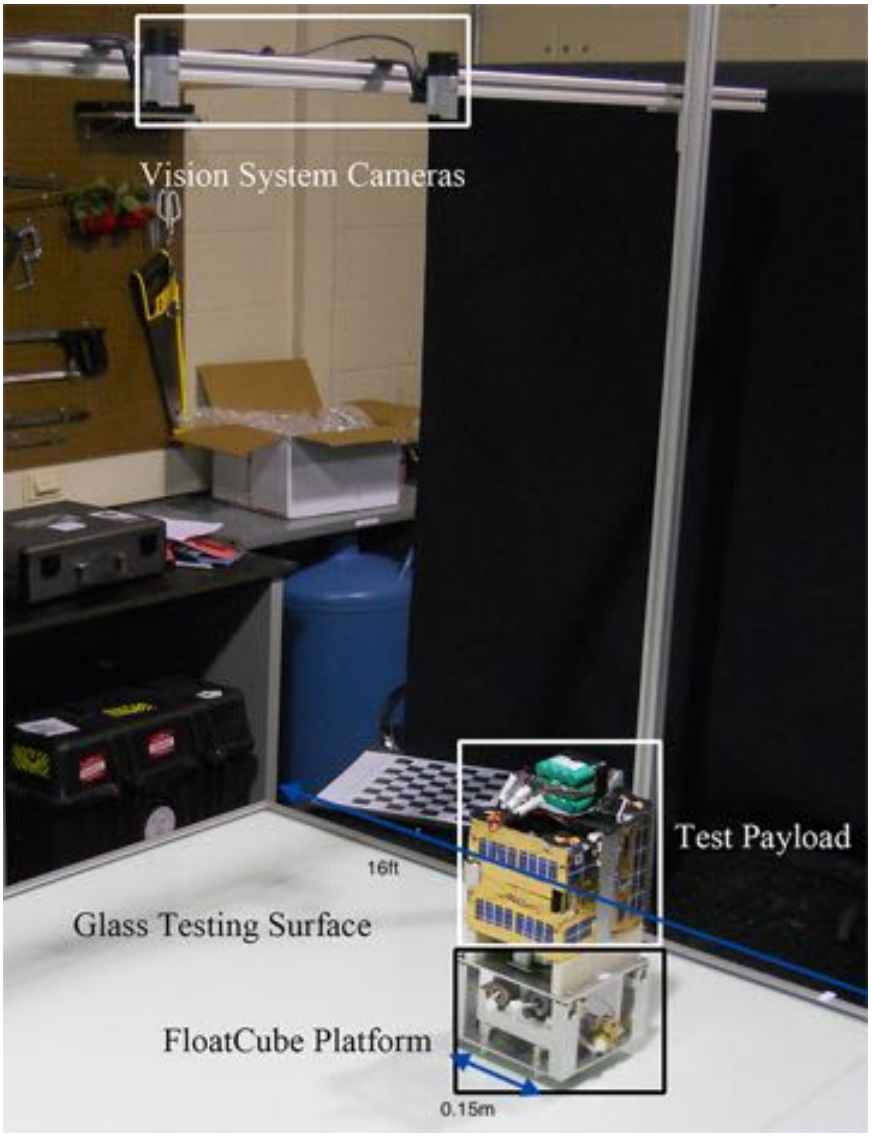


Figure 2.7. The FloatCube testbed with a test payload (Wilson et al., 2013)

increasingly more complex over the years, and in order for these concepts to become a reality, the available testing options need to be developed to match these new missions.

2.4 Advances in CubeSats and Payloads

Although CubeSats were originally purposed to be educational level missions, CubeSats have taken a turn from their original purpose to serve as avenues for more scientific and service missions (Corpino, 2014). To further the push for CubeSats in more complex missions, NASA and the US Air Force have both identified proximity operations and autonomous rendezvous and docking operations to be crucial technologies to both enable future missions in space and maintaining space superiority. These two areas of missions have become increasingly important in the last few years due to their applications. One of the necessary and pertinent applications is orbital debris removal, where "extensive robust capabilities for autonomous rendezvous, grasping, and docking in space (Tsiotras, 2014)" are required for success. An additional application is that of using proximity operations to both service satellites in orbit, protect friendly space assets and monitor non-friendly space assets (Tsiotras, 2014).

One particular CubeSat that maintains a focus on these crucial technologies as applied to orbital debris removal and space superiority is the ARAPAIMA CubeSat. ARAPAIMA is currently being developed at Embry-Riddle Aeronautical University under the guidance of Dr. Bogdan Udrea.

2.5 ARAPAIMA CubeSat

The Application for RSO and Proximity Analysis and Imaging (ARAPAIMA) CubeSat is a 6U CubeSat currently in development at Embry-Riddle Aeronautical

Univeristy, Daytona Beach Campus. The mission of this CubeSat is to "perform the in-orbit demonstration of autonomous proximity operations for visible, infrared, and point cloud generation of resident space objects (RSOs) from a nanosat platform"(Harris et al., n.d.). In order to successfully complete a mission of this magnitude, ARAPAIMA has to master a few key skills. These skills include rendezvous maneuvers with an uncooperative target, in-orbit attitude determination, and identification and tracking of a target. These skills provide the fundamentals to the previously identified needs of space superiority and orbital debris removal, by the U.S. Air Force and NASA. For Embry-Riddle, ARAPAIMA is the first mission of many and serves as a technology demonstration for the use of CubeSats to provide services previously only attempted on full-size spacecraft with multi billion dollar projects. ARAPAIMA plans to utilize the small, agile form factor, shown in Figure 2.8, to improve upon the capabilities of previous missions. ARAPAIMA is the first of many steps to move the CubeSat forward into the realm of orbital debris removal and proximity operations.

In order to achieve mission success, ARAPAIMA works its way through a series of steps. ARAPAIMA must acquire the target with assistance from ground control. Once the target is acquired, ARAPAIMA chases down its target to get within a reasonable range for the payload instruments. ARAPAIMA then enters a relative circular orbit with respect to the target, this orbit reduces in size through a series of autonomous angles-only navigation maneuvers performed by ARAPAIMA. Once within the appropriate range from the target, ARAPAIMA uses chaser attitude motion

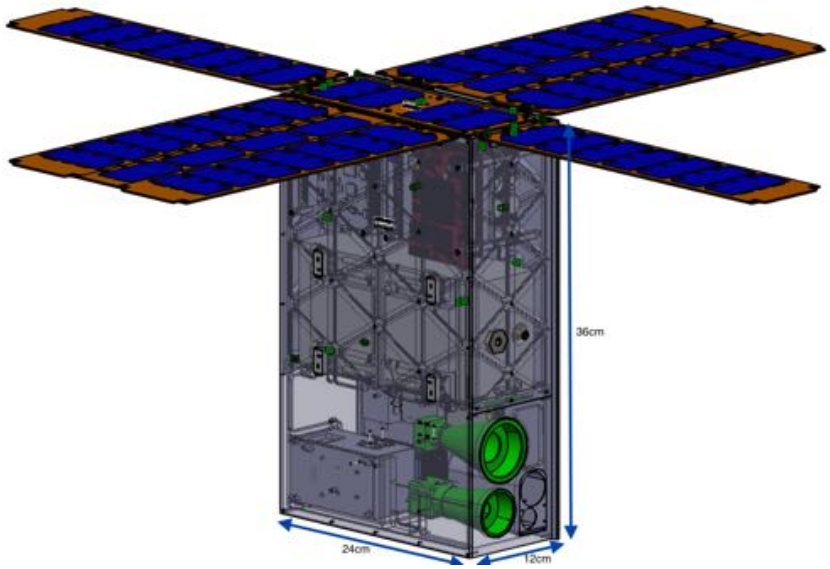


Figure 2.8. A CAD model of the ARAPAIMA CubeSat

and relative motion to perform the imaging and attitude reconstruction. This complex mission timeline requires ARAPAIMA to have a particular set of components.

The ARAPAIMA CubeSat is a 6U class CubeSat with dimensions of 12x24x36cm and a maximum mass of 6kg. ARAPAIMA features a unique propulsion system designed and built by the University of Arkansas that allows for precise navigation in space. In addition, ARAPAIMA has a host of payload instruments used to assist in the mission objectives. These instruments include a visible spectrum camera, an infrared camera, and a laser rangefinder. Unlike most other ADCS mission focused CubeSats, ARAPAIMA does not use reaction wheels. Instead, ARAPAIMA relies on its very accurate cold gas multiphase propulsion system to perform all of the required maneuvers. With the complex algorithms required to complete the mission objectives and the concept of operations, a significant amount of processing power is required on-

board the CubeSat. To meet the demand, ARAPAIMA has two on-board computers, one of them dedicated completely to the science objectives and algorithms. In order to power the payload, the propulsion system, and the on board computers, among other components, ARAPAIMA has a complex array of deployable and body-fixed solar panels.

All of these components need to be tested to ensure that their performance is up to par for the mission and that the components are compatible and complement each other. This feat is especially true of the payload subsystem where the instruments performing to specification is essential to mission success.

2.6 Summary of Relevant Literature

CubeSats are a class of satellites defined by size and mass. For the most part, this class varies in size but adheres to the basic principle that each 1U of size is the equivalent of ten cubic inches and a kilogram of mass. The CubeSat class is made of varying amounts of 1U building blocks. Originally intended for educational purposes, CubeSats have gone from simple university programs to becoming technology demonstrations and having their own influential missions.

As CubeSats have taken on more complex missions, more standardization in the development process and increased testing has become a necessity. Industry funded programs normally undergo quite a few different tests before the completion of the spacecraft. A vast majority of these tests are environmental tests. This series of testing includes vibration, vacuum, shock, and temperature testing. However, in

addition to these tests, spacecraft with more complex ADCS missions or payloads also generally undergo HIL testing. HIL testing allows for problems to be identified from the integration between components as well as serving as a way to test flight software and algorithms in the loop with the hardware. Performing HIL testing provides vast insight into the operation of the system as a whole and it allows for nuances that cannot be simulated in models to be identified.

Many universities have recognized the need to implement HIL testing and simulation into their own CubeSat programs. This can be seen in the development of testbeds such as FloatCube which allow for full motion testing to be conducted. While some of these testbeds are three to six DOF, they allow for the testing of grapple systems, rendezvous and docking maneuvers, and complex ADCS maneuver testing.

As more complex missions start being developed at the university CubeSat level, additional testing is required to ensure the mission success. The goal of the university CubeSat is no longer to just have them turn on and communicate in space, the goal is now to have groundbreaking technology and capabilities in a smaller scale satellite and for a drastically reduced price point. As missions such as the ARAPAIMA mission at ERAU demonstrate, CubeSats can become one of the greatest resources in space. Once the capability of CubeSats can be shown, the cost and size factor make CubeSats an unbeatable option as compared to multimillion or even billion dollar missions.

Universities have shown that the concepts and designs are there to use CubeSats to address some of the most pertinent topics for space. Now, the testing basis needs to be put in place to make these CubeSat missions as successful as possible.

2.7 Hypothesis

It is proposed that a low cost hardware-in-the-loop testbed can be designed such that it can be an easily accessible tool for university CubeSat programs. It is additionally proposed that the testbed can be developed in a way which makes it a universal testing option. This testbed should be applicable to various CubeSat mission types and university program styles. The testbed should be available to use starting from the early stages of development through the final stages of integration.

3. Methodology

The hypothesis is tested and studied through the creation and integration of hardware and software. The methodology used to test the hypothesis follows the process of developing and choosing the hardware in a way that best meets the requirements of the testing, then creating the software so that control of the hardware is simple and modular. Thus, the testbed is designed as a concept, a testing apparatus is built to fit into the testbed, and control software is coded to control the testbed and the apparatus. The testbed is then put through a series of tests to determine if the hypothesis is proven.

3.1 Experimental Apparatus

The testing apparatus designed, developed, and tested in this study will be here-on referred to as Chronos. In order to satisfy the delimitations of this study and work within the limitations and assumptions previously mentioned, the Chronos testing apparatus is developed. Chronos is a motorized system that operates on a Cartesian axis system. In a fashion similar to that of a 3D printer, Chronos uses a system of motors, rails, belts, and pulleys to drive the hardware in test around a scaled down space.

Chronos will be described in depth. Including the components that were selected, why those components were chosen, and the software that runs Chronos will also be discussed.

3.1.1 Hardware Design - Chapter Overview

The hardware in the testbed consists of two main categories, the robot testing apparatus and the motion tracking system. The parts for the robot testbed were individually chosen and each part combination was tested throughout the process to ensure that the testbed provides sufficient accuracy to test a CubeSat. In addition to the accuracy of the system, the hardware was chosen in a way to maximize the accuracy while minimizing the cost. Keeping the hardware costs down is essential to making the testbed accessible to university programs.

3.1.2 Robot

Chronos consists of three main subsystems, the frame, the inner motion frame, and the electronics. The frame provides the foundation for the entire apparatus. The frame size is the limiting factor for the range of motion and the frame serves as the mounting point for the other two subsystems. The inner motion frame moves within its standstill frame counterpart. This is the point to which the components under test are mounted. The electronic system consists of all the electrical components that allow the inner motion frame to maneuver and provides the means for controlling the

motion. The components will be discussed in the order that they were developed in the process.

Open-Builds

Since stability is critical to the testbed design, CNC machines shown in Figure 3.1, are used as a design example. A CNC machine was chosen because it has the ability to move heavy equipment very accurately in two to three axes. After doing some research into current CNC machine technologies, it was determined that most homebrew CNC machines consist of a rail system powered by a series of motors. This concept would form the basis for Chronos. In addition, CNC machines feature support for each of the rails and axes on both sides, not just a single side or point. This prevents the unsecured side from moving through using the secured side as a pivot point and ensures stability in the machine.

Based on the rail systems seen in CNC machines, a similar system needed to be designed to match the needs of Chronos. The first step is to find rails that are appropriate for the job. There are a few general options for rail types and sizes.

The most commonly used type of rail for more modular styles of builds is the T-Slot rail (Figure 3.2). This rail design is a part of the 80/20 framing system that uses 6105-T5 aluminum extruded beams. The T-Slot is cut down the length of each side of the beam. While the T-Slot beam has been the staple of modular builds, recently a KickStarter fund was started and completely funded to improve upon the

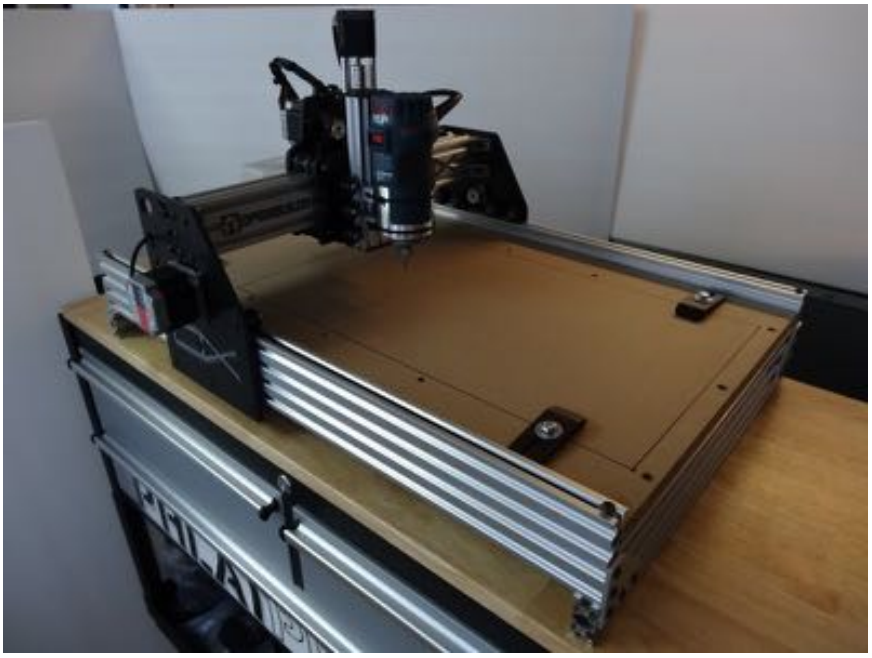


Figure 3.1. A CNC machine designed by the OpenBuilds team (Carew, 2015)

T-Slot. The OpenBuilds team designed a similar extrusion rail using a V-shaped slot as opposed to the T-Slot.

Figure 3.2. A sample of a T-Slot rail (*T-Slotted Extrusion, 10S, 72 Lx1 In H*, n.d.)

V-Slot Rails

V-slot rails, shown in Figure 3.3 are based off of the well known T-Slot design for rails. The advantage of the 'v' shaped grooves is that it keeps the modularity of the T-Slot rail while adding the ability for linear motion using the grooves.

The V-slot rail is made by simply taking the traditional T-Slot extrusion and notching the inside edge. The OpenBuilds team has released the rails under a Creative Commons fair share license. This means that many of the CAD models for the parts as well as drawings are open and available to the public. This allows for assemblies to be modeled before the parts are even purchased. The V-slots are made of 6063 aluminum T–5 and feature a clear anodized finish. This finish makes the extrusion extremely smooth which equates to a smoother linear motion.

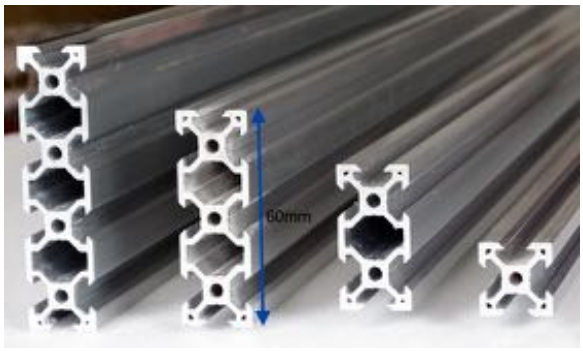


Figure 3.3. The various sizes of V-Slot rails available from OpenBuilds (*V-Slot Linear Rail*, 2015)

In addition, the rails are compatible with existing T-Slot rail components, widening the selection of possible components that can be used with the rail. The rails are now available in three different lengths, 500 mm, 1000 mm, and 1500 mm. Even with the

selection of lengths the rails can be easily cut to the desired length without changing the integrity of the rails. The rails also come in two colors, natural and black anodized, and five sizes. These sizes include 20 mm x 20 mm, 20 mm x 40 mm, 20 mm x 60 mm, 20 mm x 80 mm, and 40 mm x 40mm.

Working with the OpenBuilds team to get the parts necessary for Chronos is very convenient. One of the big causes for this convenience is that they not only designed the V-slots, but they also designed and are always adding to a collection of parts that match up with the V-slot rails. These components are accurately machined and use standard part sizes and hole patterns. This allows for the components to be used interchangeably. These parts include brackets and mounts, as well as the remaining parts required for a full moving rail system.

Belt and Pulley System

The belt and pulley system was chosen as the main means of moving the rails in the system. There are numerous reasons that the belt and pulley system was chosen for this task rather than the various other options. Some of the advantages in this case include the simplicity of using the system, the calculations involved in driving the system particular lengths (rotation to linear conversions), and the ease with which the parts could be chosen and installed. This system is simple to use based the on concept of once it is properly installed, it just works, with minimal maintenance. As for the calculations, there is a simple conversion between the rotation of the driving pulley in the system and the linear motion of the dependent rail.

Conveniently, once this factor is determined, it is constant across all of the systems as long as the driving pulley is of the same diameter. All of the parts in the belt and pulley system are supplied by OpenBuilds. Thus, the hole patterns and hole/screw sizes are consistent among all of the parts. Two different applications of the belt and pulley system are used in the design of Chronos. The first, referred to as the typical setup (Figure 3.4), consists of the pulleys mounted on both sides of a rail, one of the pulleys is smooth and the other pulley, the driving pulley, is toothed. In the center of the rail, there is the moving trolley or cart. The cart is connected to one end of the belt, the belt is threaded through the outside of both pulleys and secured to the opposite side of the cart. Thus, when the pulleys are rotating, the cart is the only part of the system that moves.

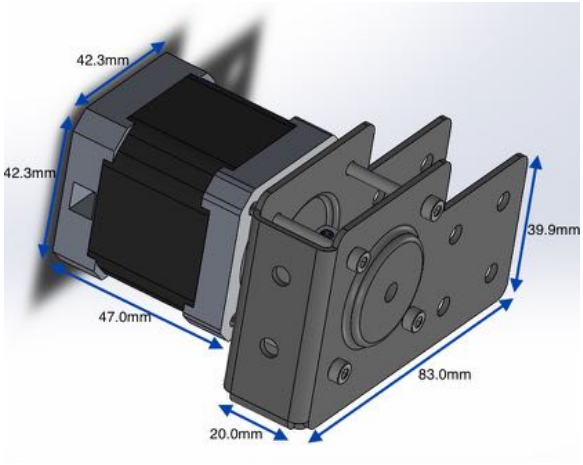


Figure 3.4. A CAD model of the assembly of the OpenBuilds endmount and the NEMA 17 stepper motor

The second setup of the belt and pulley system is the "belt and pinion" style shown in Figure 3.5. In this case, there are no pulleys at the sides of the rail. Instead,

there is only one driving pulley. The pulley is actually secured to the moving cart. The belt is installed over the top of the pulley, looped under the wheels of the cart that are to each side of the pulley, and secured to the ends of the rail using locking nuts. Each of these styles has advantages and disadvantages, hence why both are used in the design. The typical design has the advantage that the moving part is lighter since the motor and other driving parts are stationary. Thus, this was the primary style used in the design. The lighter the moving parts, the less stress on the motors. The belt and pinion style has the advantage of still maintaining accuracy in a rotational situation. Where the typical style would have quite a bit of slack in the belt, the belt and pinion style performs almost identically at any angle rail. Thus the belt and pinion style is used in the only rail in the design that rotates 360 degrees about the x-axis. The pulley systems are made up of numerous parts.

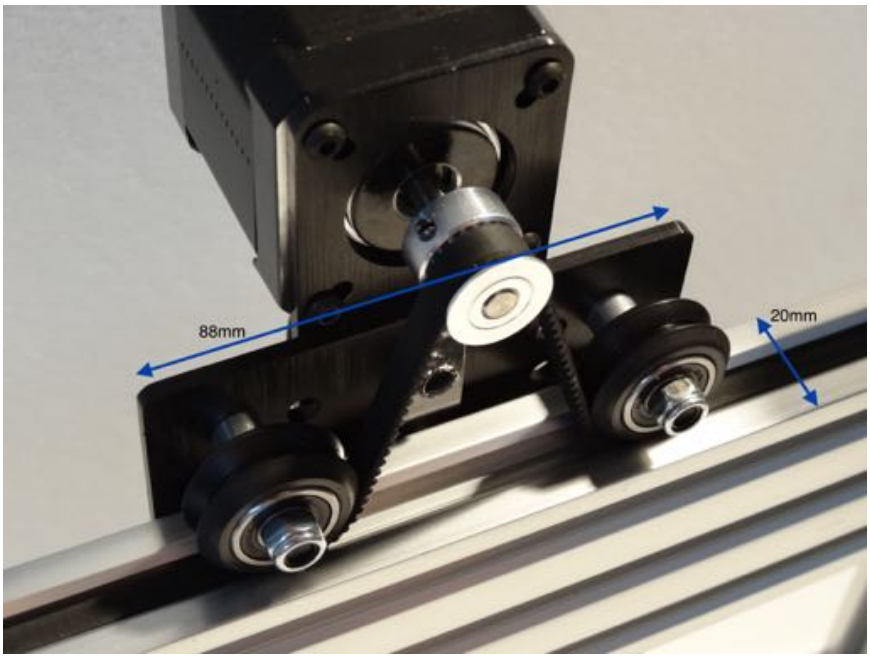


Figure 3.5. An assembled belt and pinion system (Carew, 2014)

The typical style consists of a toothed pulley, a smooth (idler) pulley, two bearings in order for the pulley to spin with minimal resistance, and washers and spacers to appropriately position the belt. All of this is secured in two corner brackets specially designed to hold the pulley system and the V-slot rails. The belt and pinion style contains the toothed pulley, the belt, and the locking T-nuts. The most important component of both of these styles is the tooth pulley. It is this part that determines the accuracy of the linear movement, through the tooth count and outer diameter of the pulley, and this is the part that connects the belt and pulley system to it's driving factor, the motor.

3.1.3 Electronics

The electronic system for the testbed is based on the concept that there is an actual computer in the loop for sending commands. Thus, the other electronic components need to be capable of completing the task of moving the trolleys as simply as possible. They need to be a pass through from the computer to the hardware. With this in mind, the idea was then developed into using a control board to take commands from the computer and translate them to a motor which drives the trolley. This concept is further developed into the electronics system presented here.

Arduino

The Arduino development board is chosen as the basis for the electronics involved in Chronos. The Arduino was an obvious pick as it is one of the best in its class. There is no need to have the board contain massive on-board processing power, the board just needs to serve as a middle man between the controlling computer and the motors that drive Chronos. The Arduino is simple to use and has a large developer community that serves as guidance and a helping-hand when working with the different types of boards. The Arduino boards work through a USB connection with a computer and have simple input and output pins around the board. For the coding that needs to be completed on the board, the company and the developer community provide plenty of tutorials and guidance for getting started with coding the Arduino. In addition, there are many pre-made codes available through open-source channels for use as demos and inspiration. The Arduino comes in many board types and sizes. The boards range from the Arduino Mini at 30 mm x 18 mm, 22 ports, and $14.81 to the Arduino Mega which is 101.52 mm x 53.3 mm, 70 pins, and $37.03. This allows Chronos to have the smallest board possible for the number of ports necessary. This is a crucial factor since the boards have to be mounted to the testbed. There have also been many accessories designed for the Arduino. There are many add-on boards that allow for the addition of wifi, ethernet, on-board memory cards, and much more. This makes Chronos expandable and upgradeable. The Arduino used for Chronos is an Arduino Micro shown in Figure 3.6.

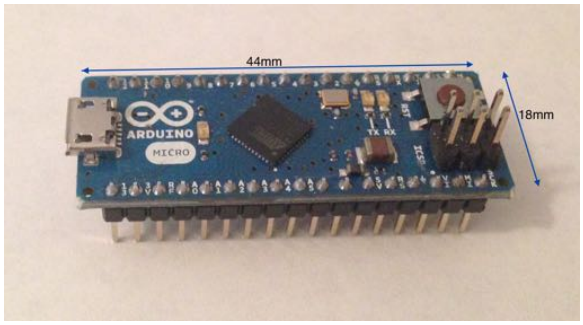


Figure 3.6. The Arduino Micro development board

The company that produces the Arduino boards also develops an accompanying software for the boards. This software, called the Arduino IO, is a simple program that uses a Matlab-like coding system. The correct board is chosen in the settings and the program uploads the commands to the Arduino through the USB connection. For the early testing of the board for Chronos, this is an easy way to quickly see if things are going to work and interface well. The Arduino IO software is used in all of the early phase testing and development of Chronos.

The Arduino boards are not capable of directly driving motors through the input and output pins. This is mainly due to the limited 40mA output per pin on the Arduino. Most motors require a lot more current in order to operate. Thus, a middle interface between the Arduino board and the motor needs to be added to the system. This component is generally a motor driver, which seperates the external power supplied to the motors from the Arduino board and allows for reversing the direction of the motor. Reversing the motor direction requires the current to be reversed, or negative, which is impossible for the Arduino to do alone. Since there are many types

of motor drivers, the motor needs to be selected before the driver can be narrowed down.

Stepper Motors

There are two main types of motors used in robotics, servo and stepper motors. After having experience with servo motors in the first iteration of Chronos, several of the drawbacks to using servo motors were experienced first hand. Thus, taking inspiration from both 3D printers and CNC machines, the stepper motor is chosen. The next deciding factors for the motors are size and holding torque. The motors need to be small enough to fit Chronos' form-factor while being strong enough to support the weight of any payload that Chronos would likely be testing. As for the size standard for the motor, the NEMA 17 standard (Figure 3.7) is chosen due to the availability of NEMA 17 mounting scheme compatible parts. After calculating the weight and distances that Chronos would see and adding the appropriate margin, the holding torque is determined. After comparing prices, the motors used on the Lulzbot 3D printers are selected for the job. While there are plenty of options to purchase motors that meet the minimum requirements, the motors made specifically for Lulzbot meet the requirements with the best price point.

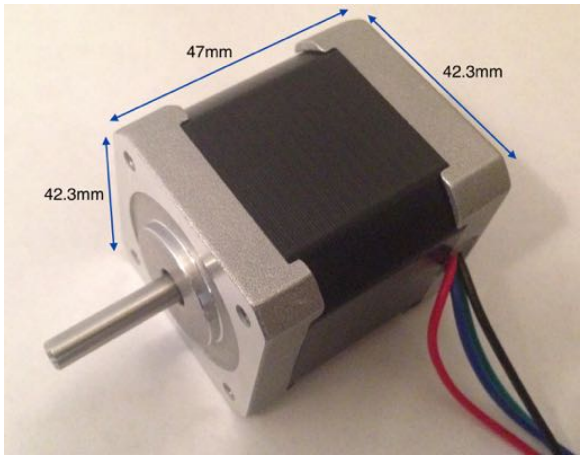


Figure 3.7. The NEMA 17 standard stepper motor used on Chronos

Motor Driver

Now that the motors are selected and the specifications are accessible, the motor drivers can be picked out to match the motors. After shopping around and finding all the available motor drivers that supply the appropriate voltage, current, and had 4 input ports as required by the motor, the DRV8834 motor driver supplied by Pololu is selected. The motor driver and corresponding wiring diagram can be seen in Figure 3.8. After selecting all the parts they are ordered in small quantities in order to facilitate testing and verify that the parts are compatible.

Motor Control

At this point, a power supply is necessary to power the motors as the power from the usb connection of the arduino boards is not strong enough. A dual output variable DC power supply is used throughout the testing. This is crucial since it allows for the

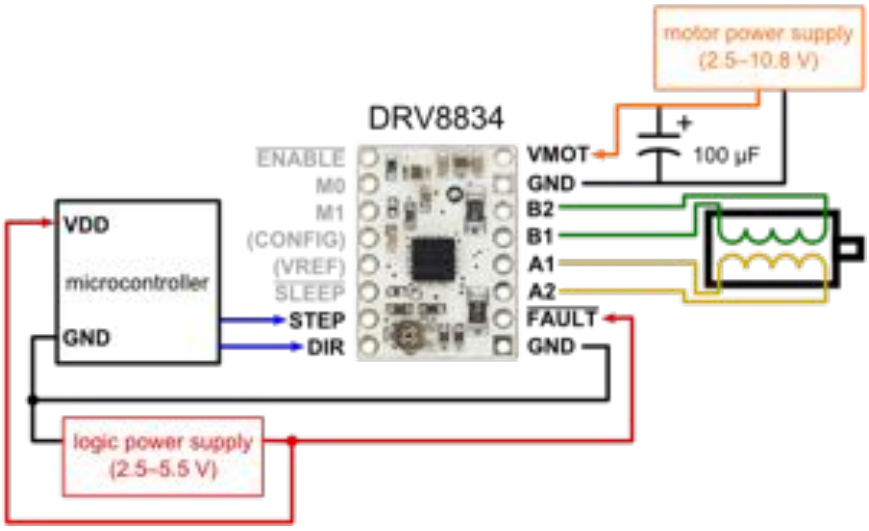


Figure 3.8. The DRV8834 motor driver wiring diagram (*DRV8834 Low-Voltage Stepper Motor Driver Carrier*, n.d.)

testing of different voltage and current levels on the performance of the motors. In addition, during testing, many different wiring configurations are tested and compared. After getting one motor to work, this is replicated on a second set. After this, a jump in design is made. The motor driver is advertised for use with one motor and the Arduino is typically interfaced with a single motor driver. After much testing, the ability is gained to use one Arduino board to run two motors on a single motor driver. This is a great leap in the Chronos development since it greatly minimizes the amount of boards necessary and the delays between two motors that need to be synchronous. The current wiring for the electronics is shown in Figure 3.9.

After some of the firsts tests of the moving trolleys with the motors, it iss determined that there needs to be a fail-safe for when the code tells the cart to move a longer distance than there is available space on the rail. Thus, limit switches are determined to be an appropriate solution to this problem. There are two options for using the

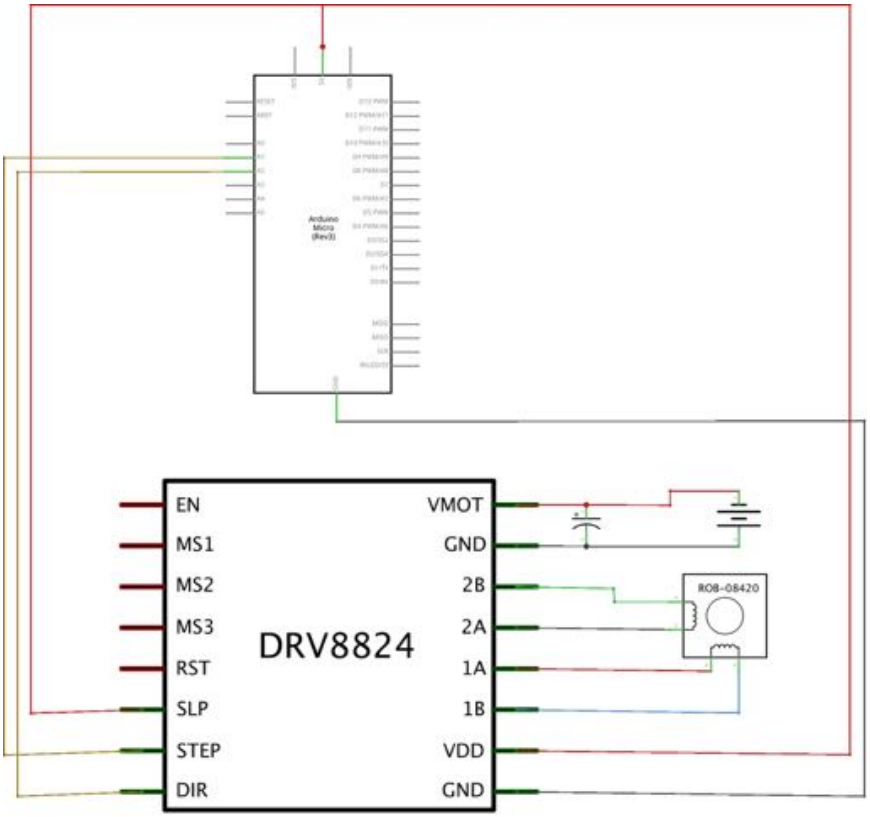


Figure 3.9. The circuit diagram of the electronics in Chronos

limit switches, they can be wired so that it sends a signal back to the Arduino when they are switched on (which occurs when something depresses the switch, in this case, the cart) or they can be wired to cut all power to the motors. The latter option is chosen because of the speed with which the motion would be stopped. By cutting the power, the reaction is almost instantaneous.

After the board configurations are finalized and purchased, the wiring needs to be completed. The motors are mounted on the appropriate mounts from OpenBuilds and the limit switches are mounted on the rails, and these locations serve as the starting points for all of the wires. Some of the wires are able to be run through the rails

themselves and covered with rail covers. However, most of the wires need to be able to move along with the payload. Due to this complexity, all of the motor drivers and Arduino boards are mounted in one place at the front of Chronos. This means that the wires can be grouped into three types, z moving, y moving, and x moving. The x and y axes, due to the small amount of wires that need to cover the distance, simply use wire covers to group all the wires together and allow there to be enough slack for the cart to freely move along the rail. The z-axis consists of significantly more wires since all of the boards are in the front. Thus, CNC wire rolls are used to house all of the wires. These work by putting all of the required wire and slack in a chain that rolls and unrolls as the rails move. With this system, all of the wires can safely be routed to the boards without impairing the movement of the rails/carts.

Power Supply System

The power supply system is designed to accommodate a varying number of motors, payload weights, and general motion speeds. This is accomplished through using an over-powered power supply and controlling the output voltage in a way that suits the needs of the system at the time. A LED power supply is used with a 24 V, 15 A output. The power supply has three output lines; however, only a single line is necessary to power Chronos. These additional lines can be used to power any additional testing apparatus as long as the total voltage required does not exceed 24 V. The power line being used for the system is routed through a custom voltage regulator circuit shown in Figure 3.10. A voltage divider, a resistor, and a potentiometer are used together to

control the voltage. The resistor has the resistance value recommended for use with the particular LM 330 Voltage Divider; 220 Ohms. The potentiometer is used to vary the output voltage from the circuit. The voltage divider circuit works by combining the two resistors in a way that results in the desired output voltage. The equation for calculating the output voltage based on the resistances is:

$$V_{out} = V_{in}\frac{R_2}{R_1 + R_2} \tag{3.1}$$

The potentiometer allows for the second resistance value to be easy changed to meet the output voltage needs of the circuit. In addition, a voltmeter is included in the circuit. The LED display of the voltmeter shows the user the current output voltage of the circuit. This helps to ensure that the user is not exceeding the maximum rated voltage for the motors and to see the effects of changing the potentiometer value on performance. The addition of this variable circuit has not only increased the safety of the power input to the electronics, but it also allows for fine adjustments to improve the performance of the apparatus. The amount of power going to the motors directly effects the speed that the motor can rotate as well as the holding torque of the motors. Adjusting the voltage can help if the apparatus is having problems maneuvering with a particular payload, if the motion seems slow, or if a slipping in position is noticed.

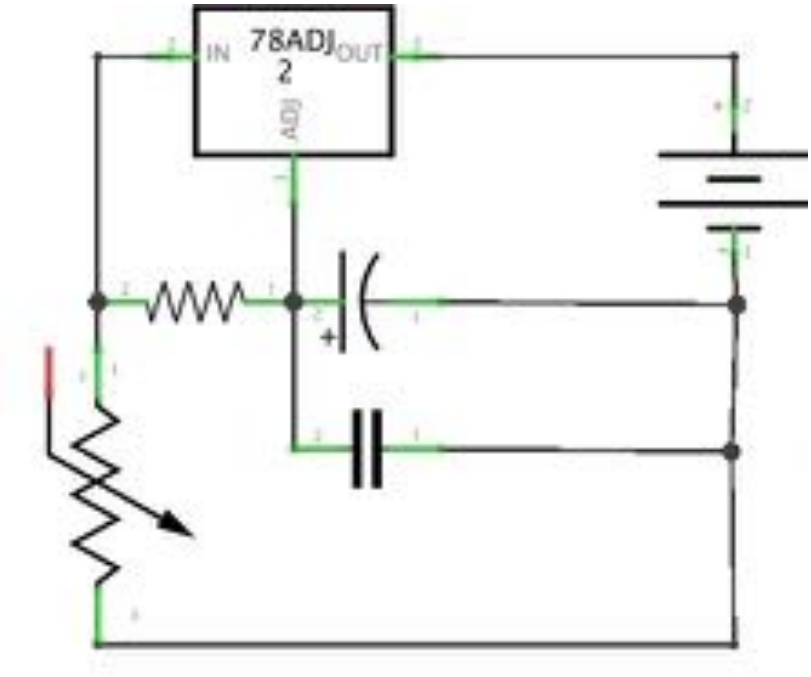


Figure 3.10. The voltage regulator circuit for managing the power input for Chronos

Control Box

All of the boards and the power supplies that are picked out based on the power and current results from testing, are housed in the control box. The control box is a 3D printed box designed specifically to hold each of the components. It includes vents for heat dissipation in the appropriate locations and a removable door. There is a shelf for each of the two power supplies and a shelf for the breadboards that hold up to four Arduino boards and 16 motor drivers. All of the wires run straight into the control box with the input cables from the Arduino to the computer and the power cords for the power supplies coming out of the control box. The three USB cables from the Arduino boards are plugged into a USB hub that has individual plugs for each usb device. This allows individual boards and thus motors to be turned on and

off when necessary. This also means that only one USB port needs to be utilized on the control computer. This completes the electronics system of Chronos.

3.1.4 RSO Models

Due to the mission of the first application of Chronos, ARAPAIMA, some external tools need to be constructed in addition to Chronos. The first of these tools is scale models of the upper stages of launch vehicles. ARAPAIMA's mission is to image the upper stage of its launch vehicle and complete a 3D reconstruction and pose estimation of that upper stage. In order to test some of ARAPAIMA's payload, flight software, and science algorithms, a few upper stages are constructed. First, the upper stages are modeled from various possible launch vehicles. Images and specifications are taken from online sources and between the basic details and many images, each of the upper stages is able to be modeled to full-scale in Catia v5 with reasonable accuracy. For the purposes of testing ARAPAIMA, the upper stages are required to be scaled down in size to match the available maneuver size on the testbed. Thus, once the maximum scale that could be used on the testbed is determined, the upper limit for the scale of the upper stages is set. From here, the scale is based on the availability of add-on parts for the upper stages, such as heaters and stands. A one twenty-fourth scale is determined to be appropriate due to its typical size for models and it falls within the size constraints.

In the next stage of the development, the upper stage models are appropriately scaled down to one twenty-fourth scale of the actual object. Then, the file is converted

to a STL file for its applicability to multiple programs. This STL file is imported into a program called MeshMixer, where it allows the model to be sliced into smaller parts capable of fitting on the 3D printer that is available for the printing of these models. Once the slicing is complete, the models are imported into the Makerware software, the main software for the Makerbot printers. In this program, the density, wall thickness, speed, and orientation of the print can be set and controlled. Each part is printed and once the individual parts for each model are completed, the pieces are sanded and glued together to produce a full model. In order to make the models look more life-like, the models are completely sanded, a primer/filler iss applied, and the models are hand painted to match their full-size counterparts. In this way, over eight models are completed, with five more modeled in CAD for further testing. Some of the completed models are shown in Figure 3.11.

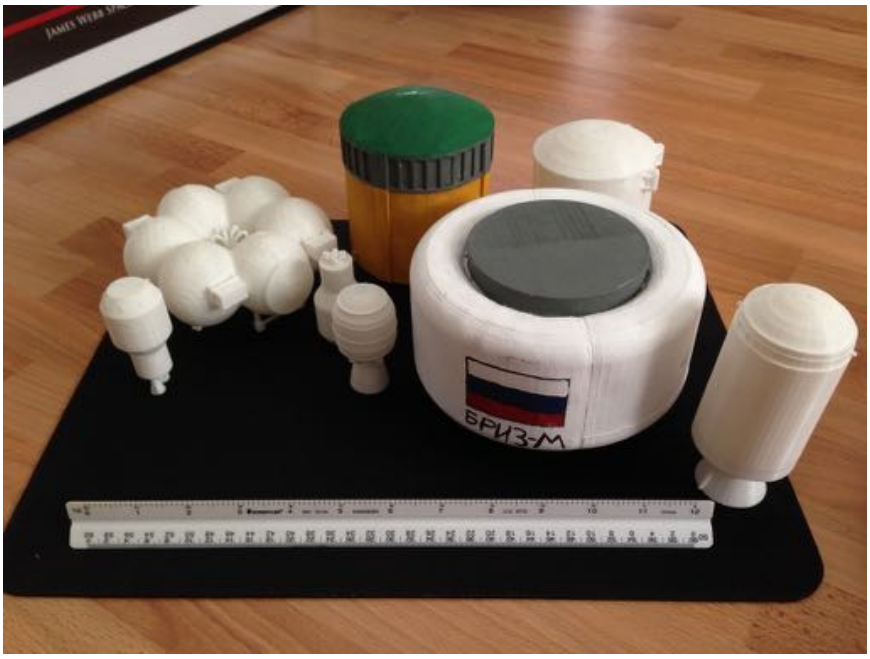


Figure 3.11. A few of the completed RSO models

3.1.5 Payload Emulator

In order to remain cost effective, the testbed features payload components that are similar (for test purposes) but lower priced than the ARAPAIMA flight hardware. The testbed consists of the three payload instrument counterparts, including the FLIR MLR100, the Sentech STC-CL33A, and the Sentech STC-CL202A; as well as a PC/104 Plus computer board and two Phoenix frame grabbers.

The first version of the payload emulator is designed to house the Sentech STC-CL33A, the MLR100 laser rangefinder, and a Logitech webcam as a stand-in for the infrared camera. The LRF comes without a protective box which makes it very sensitive to shock; therefore, the entire MLR100 is housed inside the emulator. In testing the payload emulator, having an infrared spectrum camera is important since it can be used to spot the LRF bloom on the target. With this capability, the camera can be used to ensure that the LRF is properly aligned within the emulator and that the bloom is not larger than the target and thus picking up erroneous measurements. The emulator is 3D printed in the Spacecraft Development Lab. Figure 3.12 shows the first version of the printed emulator with the LRF and two cameras attached.

The second version of the payload emulator houses the STC-CL33A, STC-CL202A, MLR100, VersaLogic CPU PC/104-Plus module, and two Active Silicon Phoenix frame grabbers. This version is outfitted to mount directly to the spacecraft simulator. Space will additionally be allocated to allow for both using the emulator with and without a separate computer to give inputs or analyze outputs.

Figure 3.12. The first generation payload emulator

For preliminary testing of the ARAPAIMA algorithms on Chronos, instead of the actual payload instruments being used, the algorithms are adjusted to allow for a GoPro Hero 3 camera to be used as a substitute. For the camera to be compatible with the Chronos mounting scheme, a special mount is designed for the GoPro. This allows the GoPro to be directly mounted to the cart that is the focal point of the design. In addition, the wireless capabilities of the GoPro camera have two distinct advantages. First, there are no wires to be concerned about routing. This allows Chronos to perform rotation maneuvers without having to account for wires twisting, and thus more accurately simulating a spacecraft's movement. The second advantage is the Wi-Fi capability. Live video from the GoPro can be seen on a computer or Wi-Fi

enabled device, such as a phone. This allows the controller to access and compare the visual results from the experiment to the simulation results very quickly.

3.1.6 Motion Tracking Hardware

The Microsoft Kinect for Xbox 360 is used for the motion tracking of Chronos. It is determined that motion tracking is necessary in order to provide feedback to the control system. Since encoded motors are not used for Chronos, due to the cost impact, an outside source of real-time, real-life position data is necessary to create the feedback loop. The Kinect is a low-cost alternative to options such as marker and sensor systems and pinging radar systems. The Microsoft Kinect, in the few years since its development has developed quite a library of software with some of this software allowing interfacing with all operating systems, making it a universal option. The Kinect is designed for mostly human tracking since it was developed to accompany the gaming console, Microsoft Xbox 360; however, the human tracking capabilities are used as a foundation for expanding the capabilities of the Kinect to specifically track Chronos.

The Microsoft Kinect provides several useful input devices for object tracking and distance sensing, which are used to monitor both the RSO and the simulation system as a whole. The available sensors are the Color Sensor and the IR Depth Sensor, both of which have a resolution of 640 pixels x 480 pixels. The range of the depth sensor in default mode is specified as 0.8 m to 4 m (2.6 ft to 13.1 ft), and in near mode, the range is 0.5 m to 3 m (1.6 ft to 9.8 ft).

Originally, it was planned to use three Kinects placed around Chronos in order to triangulate the results received from each individual system. However, this was quickly changed after it was discovered that the beams used by the depth sensor of each of the Kinects can interfere with each other and cause skewed results. Thus, one Kinect is used in the final design and it is placed in the optimal position based on research on the accuracy of the Kinect at varying distances and throughout its field of view. Thus, we are able to get, with calibration, very accurate results from this low-cost solution.

There are four versions of the Microsoft Kinect at the time of writing. The original pair of Microsoft Kinect versions are the Kinect for Xbox 360 and the Kinect for Windows. There are five main differences between these two particular versions, where all of the differences are additional functions present in the Kinect for Windows (Kerkhove, 2012).

- Near mode: Allows for the cameras to see objects starting from 40cm to a 3m distance
- Seated mode: Provides skeletal tracking of the user
- USB cable: Contains a USB port instead of the custom Xbox connector
- Camera settings: Adds additional settings including brightness and exposure
- Kinect Fusion: Maps an environement to 3D
- Handgrip: Adds hand detection allowing for gestures to be utilized

At a price difference of almost $200, it was decided that these added features were not necessary for this particular application. The other two versions of the Microsoft Kinect are the Kinect for Windows v2 and the Kinect for Xbox One. These versions are just updated versions of there version one counterparts that were released in late 2014.

3.1.7 Subsystem Descriptions

Now that each of the individual systems that make up the testbed has been discussed, the relationship between these systems is discussed. Chronos consists of a system of motion in three directional axis, x, y, and z. Each of these axis of motion is powered by the moving trolleys, where the y-axis and z-axis each have two synchronous moving trolleys (one on each side), and the x-axis has a single trolley using the belt and pinion motor style. Additional rotational movements are done using a single motor where the motor shaft is facing in the direction of the axis of rotation.

The z-axis is controlled on the bottom of the frame. Here, the end mounted motors control two mini v trolleys. It is off of this rectangular frame that the rest of the system is built. The y-axis uses end mounted motors; however, instead of the motors being in free space, the motors are mounted to the trolleys of the z-axis. From the end mounts, the rails are in a vertical direction. The y-axis trolleys are mounted on these rails as well. The end mounts at the top of the vertical rails are attached to trolleys on a second rectangular frame. This second frame is secured to the original, bottom frame using corner brackets. Having the frame create a cube like shape allows

for the vertical components to be secured at both the top and the bottom of the rail, increasing stability. The x-axis is designed a bit differently than the other two axes due to its rotational capabilities. A motor mount is placed on each of the trolleys belonging to the y-axis. Motors are placed in these mounts and a rail is placed in between these two motors using mounting hubs which lock the rail in a particular orientation on the motor shaft. This allows the motors to spin the rail. The x-axis movement is provided by a trolley mounted on the x-axis rail, with a belt and pinion motor setup. Thus, in this fashion, all three directions of motion are established. The completely assembled Chronos CAD model is shown in Figure 3.13.

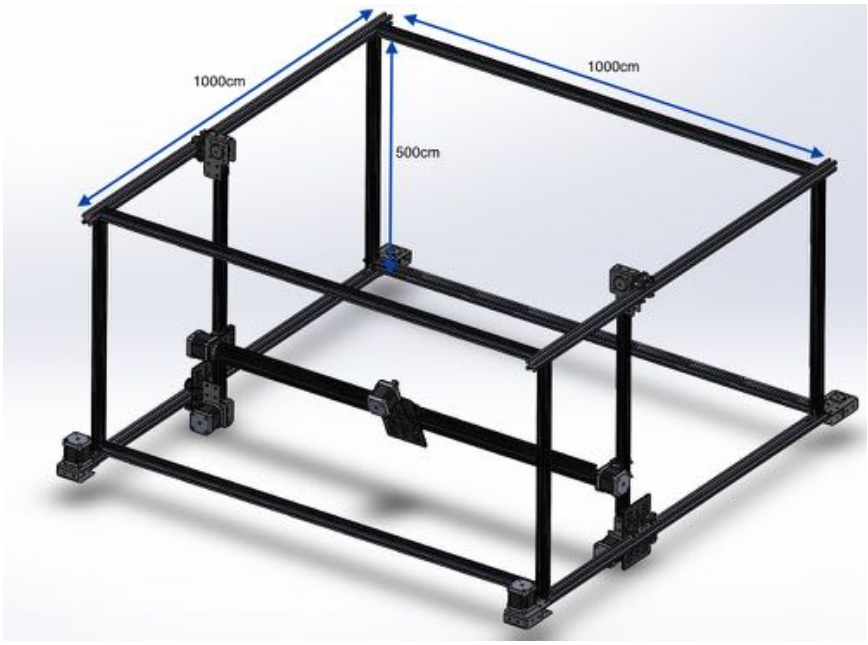


Figure 3.13. The SolidWorks CAD assembly of the Chronos testing apparatus

Additional axes of rotation can be added as necessary. Although it is recommended to use the least amount necessary for the maneuvers, due to the extra weight being added to the system, additional axes of rotation are simple to add. Through utilizing

a variety of mounts available from OpenBuilds, more motors can be added in the necessary directions. The most difficult part of adding motors is simply running the wires back to the control box.

The control box, once completed, contains at least a single breadboard and a single power supply. These values can change with the number of motors added due to the motor drivers required and the amount of power needed to drive the motors. In the current Chronos configuration, the LED power supply is used with the voltage divider circuit and a single breadboard. The breadboard holds four motor drivers and a single Arduino board. Each of the motor drivers is routed to a different set of output pins in the Arduino. Each motor driver operates a single axis of motion or rotation. If the axis has dual motors, the motors are connected to the same pins in the motor driver, just one of the connections is flipped. This rotates the motors simultaneously in opposite directions, allowing the motors to be facing each other in the design.

Overall, Chronos is designed to be a stationary cubic frame with a moving interior and a house for all of the electronics. This is a simple design that keeps the testing apparatus modular, strong, steady, and accurate.

3.1.8 Summary of Hardware Design

The hardware previously discussed is the foundation of the testbed. This hardware is picked for its modularity and simplicity. The hardware chosen is all from open source companies and are all open source products with the exception of the Kinect for which the hardware is proprietary. This allows for the hardware to constantly be

upgraded, updated, and modified to match the current technologies and the current mission needs of different CubeSats. At this point, the motion tracking system, the RSO simulator, and the spacecraft simulator have finalized hardware designs. Not only do these systems work as separate entities, but they work smoothly together and easily accept the modification or addition of other hardware. The functional architecture of the testbed can be seen in Figure 3.14.

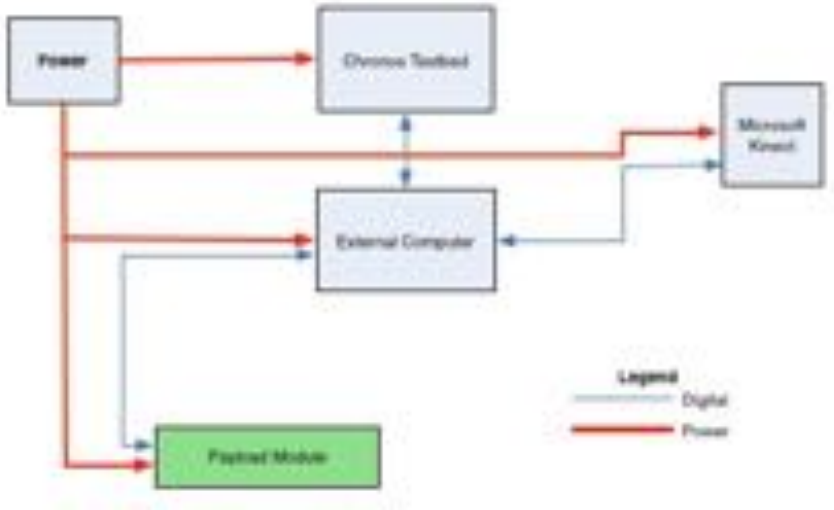


Figure 3.14. The block diagram of the functional architecture of the testbed

3.2 Software Design - Chapter Overview

Once the hardware is finalized, software is developed to meet the needs of the hypothesis and match the hardware chosen. The software to be used for the testbed is based on both open source software and the Matlab/Simulink environment. Although Matlab is not open source, it is a commonly available tool at universities and it is commonly used to develop software for university CubeSats.

Utilizing these tools, software is developed to control the testing apparatuses and the motion tracking system. In addition to these core tasks, a few software additions are made to simplify the process of using the testbed.

3.2.1 Arduino Software

There are a few different options for working with the software side of the Arduino boards. Arduino boards respond to specific Arduino commands and reference data through different groups of libraries. While Arduino has its own recommended software, it is possible to use external software to work with the Arduino boards. In this process, once a sketch is written for the board, the sketch must be burnt to the board using the external program. However, Arduino simplifies that process by providing an intuitive interface for the Arduino and designing add-ons for other programs to be able to read and write to the Arduino board. The two programming options used for Chronos in the development process and for the final software control system are discussed in this section.

Arduino IDE

Arduino provides an open-source software hosted on GitHub for use with the Arduino boards. The Arduino IDE is a simplistic code editor that integrates directly with the board via a USB connection or Wi-Fi. Within the software interface, working with different types of Arduino boards is as simple as selecting the correct board from a

list of currently available boards. This allows the user to quickly switch between boards when running software. The Arduino IDE is used in the early stages of testing for Chronos since it is quick and easy to go from an idea to a coded piece of software that is sent to the board for testing. Coding in the Arduino language is very similar to that of Matlab, this allows for a smooth and easy transition between rapid software testing in the IDE and the actual Matlab software. One large advantage to using the Arduino IDE in the early stage testing of the motor control is the variety of demostration codes that accompany the software. There are four different demoonstration codes focused on different controls of stepper motors alone. Thus, if simple interface checking of the motors needs to be completed, a demonstration code can be selected and a couple of the parameters (steps per revolution, speed, etc.) changed to match the motor. Once this is completed, the code can be sent to the Arduino for immediate testing. Working with the Arduino boards in the IDE as opposed to matlab or another code editor also allows for easy feedback from the board. Directly in the IDE interface, it shows the board's status. This allows the user to quickly discriminate between the possible sources of issues that may arise during testing.

In addition to just using the Arduino IDE software, there is also a large community of developers online. These developers have many different projects published online, allowing new developers to easily access a vast amount of relevant tested code. There are many different codes for slightly different purposes, but they are similar enough in the basic coding structure that allows for them to assist with the first few motor tests. This includes using openly available code to learn the commands, the tools

that are available to the user, and to use as a starting point as the code is being developed. The developer community is also easily accessible for any questions that may arise while developing the software for Arduino based projects. There are quite a few forums and websites dedicated to experienced users answering freely posted questions and assisting with development issues.

While it may not be the best option for projects that are mostly computational as opposed to a simulation heavy program such as Matlab, the Arduino IDE provides a truly accessible interface for working with the Arduino boards. Although most of the software for Chronos is Matlab and Simulink based, the code that allows Simulink to interface with the Arduino boards is written in and sent to the Arduino Micro boards through the Arduino IDE. This code simply opens up the ports on the board so that an external program, i.e. Matlab can have access to read and write to each of the pins.

Simulink Arduino Toolboxes

Currently, Simulink offers a support package for working with Arduino boards. This support package gives the user a library of Simulink blocks as well as direct deployment to the board. Some of the blocks included in the library are both analog and digital inputs and outputs, blocks for servo motors, and blocks for UDP and TCP send and receive. These blocks allow for interfacing with the Arduino board from completely within the Matlab/Simulink environment.

This support package is limited in its available uses. One of the big hindrances is the way in which the package uses inputs and outputs to/from the board. The output

block is just that, a single block that outputs the specified code to the pin specified in the block information. Thus, you can only output to a single pin at a time. This is a constraint that made it almost impossible to output the correct information for the Arduino board to interface with the motor driver. Especially since the motor driver requires two inputs from the arduino. Thus, a non-natively supported Simulink package for Arduino is used as an alternative.

ArduinoIO, is a Simulink support package that was created as an alternative to the native package. ArduinoIO uses serial port communication to work with the Arduino boards. The package works by having a sketch previously installed on the Arduino boards. In Chronos, this sketch is uploaded using the Arduino IDE. This sketch is a server program that opens all of the ports on the Arduino for external access. Once the ports are open, a simple read/write block can be used in Simulink to communicate with the Arduino. In Simulink, the sole purpose of the package is to supply a Simulink block that either reads or writes to/from a specified pin on the Arduino board. This allows for any output to be generated through Simulink and uploaded to the board.

The advantage of the ArduinoIO package over the previously mentioned Simulink support package is that all of the ports on the Arduino can be accessed simultaneously. Thus, as an application to Chronos, each Arduino board and motor driver combination has 2 output blocks in Simulink. If there are two motor drivers connected to a single board, there are four output blocks. This idea is replicated for each Arduino board in the system, in a single Simulink code. Once the position of the robot is determined in Matlab for the next position of Chronos, the appropriate PWM curve is generated for

each axis of movement. This curve is then sent to both output ports for the Arduino board matching that axis. This allows for each DOF to be controlled simultaneously and simply in the Simulink interface.

Since the sketch on the Arduino tells the pins on the Arduino to constantly be looking for an input signal, the above process can ideally be repeated endlessly. The Simulink output can keep sending PWM curve to the pins on the Arduino corresponding to the position and velocity for each required motion.

3.2.2 Virtual Robot

The "Virtual Robot" is a matlab simulation which models the Chronos testbed. It serves as a first testing place for all code before it gets sent to Chronos. It allows for any problems with the code to be seen and processed before it reaches the actual hardware. The greatest advantage of this is that any incorrect movements that could result in damage to the Chronos hardware or the hardware being tested can be seen and corrected before they occur, preventing damage from simple missteps.

Before code is ever sent out to the physical Chronos, the user has the option to see the simulation or bypass it. It is recommended that the simulation is run especially before any new code is sent to Chronos. The code is then run with the simulation where the CAD modeled components of Chronos are moved with appropriate connections in SimMechanics.

CAD Modeling

For the CAD modeling of Chronos, multiple CAD programs are used in the process. The original designs and models are completed in Catia. This was simply chosen due to the proximity of the program to the author. Catia however, does have limitations when it comes to interfacing with other programs. Since the virtual robot is to be designed for Matlab, a change in the CAD format is required. SolidWorks allows for simple integration with Matlab through a plug-in for SimMechanics provided directly from Mathworks. Thus, the final Chronos model was created in SolidWorks. Since the parts were in-house when the models were started, the CAD model is more accurate then the average CAD model due to using the actual dimensions of the parts. Each part is individually measured to the thousandth of a millimeter to allow for the model to take into account the fine nuances of Chronos. Two versions are modeled, one with subassemblies and another without them. These two versions use the same parts and the final CAD models look identical; however, it is the SimMechanics model in Matlab that sees the difference. SimMechanics transfers subassemblies as subsystems in Simulink. This makes the model more organized in Simulink rather than seeing every point connection and contact between every part in the model at one time. The SolidWorks CAD model of the RSO testing apparatus, a smaller version of Chronos, is shown in Figure 3.15.

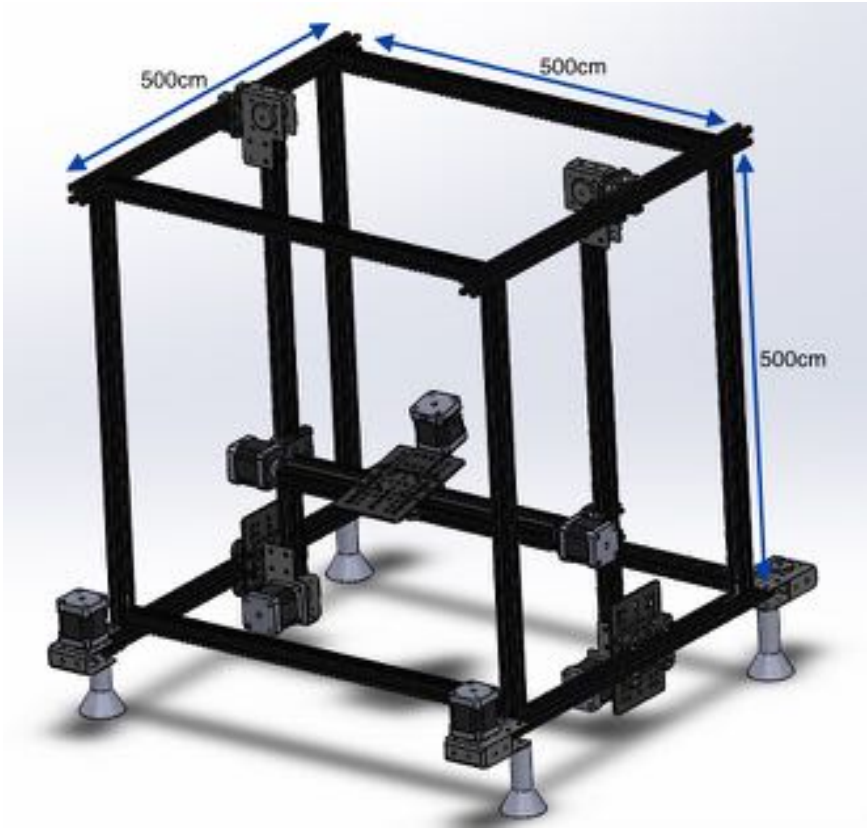


Figure 3.15. The SolidWorks CAD model of the RSO testing apparatus

SimMechanics

SimMechanics is the interface through which Matlab and the CAD model are integrated. The plugin for SolidWorks allows for the CAD models to be quickly converted to the necessary file type for SimMechanics integration. Once the plugin is activated, all of the CAD files are converted and saved as STL type files. In addition, a file with the type of .xml is created, this file contains the simulink model that features all of the connections between the parts from the CAD model. These connections include the constraints between the parts along with the relative positions between the parts. Once this file is opened, the CAD model can be viewed and altered as a

Simulink model. The Chronos SimMechanics model is shown in detail in appendix C, Figure C.8.

The constraints such as contact, concentric, etc. that are used in SolidWorks are converted to the connection types recognized by SimMechanics. These constraint types include angle, coincident, concentric, distance, parallel, and perpendicular which can be applied to the following mate entities: circle/arc, cone, cylinder, line, plane, and point. Once any issues that may have arisen from the conversion process have been solved, additional features may be added to the model in SimMechanics. For elements to move in the model, the joints need to be individually moved. In SimMechanics this is done using the joint actuation block, which takes position, velocity, and acceleration inputs and moves the assigned joint accordingly.

This method is used to convert the ideal coordinates for Chronos to virtual movement. The PWM curve used to drive the motor, can also be used to actuate the virtual model. Once the model is moving, there are many options for what to do with the model. A video of the actuation can be saved, the speed with which the model runs can be adjusted, the model can be rotated, and the zoom can be adjusted.

Besides just moving the parts in the model, more features can be added. An initial position condition can be applied to the model, advanced modeling can also be added. Some of this modeling includes adding friction into the model, material properties, and effects such as vibrations can be analyzed. The SimMechanics program becomes an indispensable tool for analyzing the system.

3.2.3 Kinect Software

The software behind the motion tracking system, OpenNI (Open Natural Interaction) was originally designed by PrimeSense, the company that built the depth sensing reference design that the Kinect is based on. Although the OpenNI project was stopped when PrimeSense was acquired, it has been since forked as OpenNI2 and development continues in the open-source community. The framework for OpenNI and OpenNI2 is referred to as the OpenNI SDK. This SDK provides the APIs for support with voice and voice command recognition, hand gestures, and body motion tracking. However, this SDK is simply the framework for the natural interface data processing. There are two generally agreed upon softwares in the community for working with the SDK. The first software is NITE, a freeware library from PrimeSense that provides the code that allows for actual tracking to take place. This software processes the depth images from the camera and gets the data from those images. The second software is SensorKinect. This software simply provides the driver for using the Kinect with the SDK.

There is a third software that is often used with this grouping. Libfreenect provides additional drivers that allow for the control of the Kinect's motors and lights. This can come in handy since it allows the user to change the direction and motion of the Kinect which can increase the FOV of the cameras or allow the Kinect to re-adjust itself to be pointing at a calibration position.

The code is written in C++. So all of the setup to design and simplify the motion tracking to meet the needs of the system is completed in C++. At this point, there are a few routes that can be followed to integrate the position data into the Matlab feedback loop. The first is to use the toolboxes that are provided by the open-source community for pulling the camera video into Simulink. The second is to use the Mex abilities of Matlab to read and/or run the C++ code. The third is to completely obtain the results in C++, write the continuous results to a file, and have Matlab continuously read that file and integrate the position data into the feedback loop.

Kinect Toolboxes

The Kinect, since it is made by Microsoft, is designed for use with the Kinect SDK provided by Microsoft. The SDK provides a simple way to interface with the Kinect by allowing the user to install the drivers and work with the Kinect control software directly. Microsoft has also worked with Matlab to provide a set of simulink toolboxes that allow direct access to the cameras and the motors in the Kinect through Simulink. While this is a very simple way to get the feedback necessary from the Kinect in the motion tracking system, it is not completely modular. Since one of the goals of the testbed is to work with any operating system, the SDK was not a suitable software since it only works with the windows operating system.

With this limitation imposed by the software on the system, another way to interface with the Kinect is needed. Before the Kinect SDK was developed, an open-source alternative was created to allow users to interface with their Xbox Kinects.

This software is the OpenKinect software which uses the libfreenet libraries to talk with the Kinect. While this software is no longer kept up by the original developing company, it is still updated through the Git where is it available to download and fork. Thus, the author deemed that this software is reliable enough and that there are enough users of the software dedicated to improving the software that the software is sufficient and reliable enough to be used in the testbed.

Although this software does not directly interface through the simulink toolboxes provided by MathWorks, there are other ways to interact with the Kinect through Simulink. Since the software is based in controlling the Kinect through C++ code, Matlab can use that code through the Mex tools.

Object Isolation and Triangulation

Originally, multiple Kinects were to be used for the motion tracking system; however, for the purposes of this research, it was limited to a single Kinect. The reasoning behind this decision was the uncertainty of the of interference between multiple Kinects. The depth sensor uses infrared beams to determine depth; however, if multiple of these beams overlap on the same objects, the data can become skewed. The beam reflected back into the Kinect may have not originated from that same Kinect. For this reason, the original plan to use multiple Kinects to triangulate the position of the test payload to achieve more accurate results is bypassed at this stage in the research.

In this iteration of the testbed, a single Kinect is used for the motion tracking system. The payload to be tracked on the testbed is identified by the motion tracking system using object isolation. The Kinect software is trained to recognize not the payload itself, but the components of the testbed. This allows the payload to be interchangeable without altering the software. The C++ code recognizes each of the rails and the trolleys in the testbed and basically just ignores their existence. Even though these components are moving, the Kinect will continue to ignore them in the tracking process. Instead, the Kinect will only track the payload as it is moving. The software tracks the movement of Chronos and the payload in its entirety. Then, it ignores the results from the recognized components, i.e. the testbed, leaving only the movement results of the payload.

The object isolation is accomplished by having the Kinect recognize a combination of colors and shapes. The software then associates the combinations with different components of the testbed. The recognition uses both of the Kinect's cameras since the regular camera can see both color and shape and the depth camera is used to validate the shape as parts are moving. In this way, the motion tracking system is capable of sending the depth data of only the moving payload back into the feedback loop through validating the depth data with object recognition.

Accuracy

Since its release, the Microsoft Kinect has been widely studied. Within these studies, the accuracy of the Kinect has been looked at in depth. (Dutta, 2012) studied

the accuracy of the Kinect when the targets were at different x, y, and z coordinate positions relative to the Kinect. There were 104 data points collecting during the study, with any distance greater than 3m away from the Kinect being discarded. The RMS error is calculated to be as shown in Table 3.1. Figure 3.16 shows "a. Stem plot of RMS errors for the x coordinate plotted with respect to the x and y coordinates of the target marker position. x stems represent discarded data points while o stems denote points that remain within the final 3D capture volume. b. Stem plot of RMS errors for the y coordinate plotted with respect to the x and y coordinates of the target marker position. x stems represent discarded data points while o stems denote points that remain within the final 3D capture volume. c. Stem plot of RMS errors for the z coordinate plotted with respect to the y and z coordinates of the target marker position. x stems represent discarded data points while o stems denote points that remain within the final 3D capture volume."(Dutta, 2012).

Table 3.1. The RMS error and Standard Deviation of the Kinect on each axis (Dutta, 2012)

Axis	**RMS Error [m]**	**Standard Deviation [m]**
X	0.0074	0.0061
Y	0.0120	0.0112
Z	0.0074	0.0075

In addition to characterizing the accuracy of the Kinect camera, (Dutta, 2012) also determined the FOV of both the color camera and the depth sensor. These results are shown in Table 3.2 where the advertised values for the FOV are shown as well.

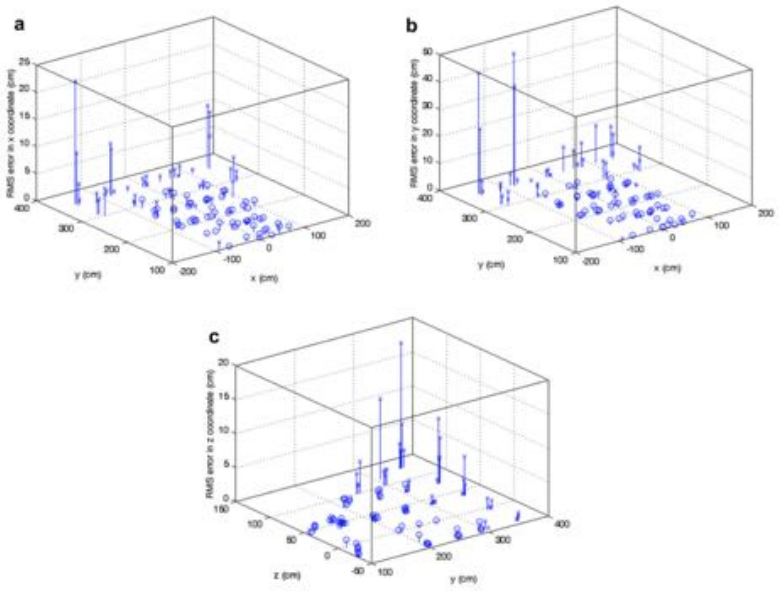


Figure 3.16. A graphical representation of the RMS values (Dutta, 2012)

Table 3.2. The measured FOVs of the Kinect compared to the specified values from Microsoft (Dutta, 2012)

Sensor	**Horizontal [degrees]**	**Cited Horizontal [degrees]**	**Vertical [degrees]**	**Cited Vertical [degrees]**
Color Camera	61.8	57	47.4	43
Depth Sensor	58.6	57	43.6	43

Having an accurate FOV for the Kinect allows for proper positioning of the Kinect relative to the testbed. The Kinect should be placed so that the entire testing appratus falls within the FOV.

3.3 Testbed Control Software

The control software for the entire testbed consists of two main components. The first component is the initialization files, a few .m files that setup of the software appropriately based on the user input. The second major component is the Simulink file that conducts the calculations and deploys the simulator. The initial setup required to interface the simulation code of interest with the control software can be completed in one of two ways. The first option is to simply add a function call to the end of the code of interest if the code is based in matlab. If the code is based in simulink, the appropriate input/out blocks can be added and the initialization file can be run separately. Once the interfacing is complete, the code of interest can be run. Once the initialization code is run, it prompts the user for a number of inputs. These inputs include, which apparatus is in use, whether or not the virtual simulator should be run, if the calibration should be run, etc. The responses to these prompts allow for the Simulink code to follow the correct path. The main simulink file consists of four major subsystems, the "robot specifics, "virtual robot," "calibration," and "robot" blocks.

The robot specifics block uses the user responses to properly scale down any position coordinates to a scale appropriate for the apparatus in use. Two types of scaling occur in this block, the first is length scaling. Here, any commanded position is scaled based on the largest coordinate value compared to the corresponding rail length. The second stage of scaling is the delta scaling. The scaled lengths are taken and the delta between each position is calculated. The sum of the deltas is used to determine

if over time the movements will exceed the available rail length. If so, the coordinates will again be scaled to prevent this from occurring. The final scaled coordinates are then sent to the next block.

If the user has chosen to run the virtual simulator, the scaled coordinates are then applied to either the RSO or the Chronos SimMechanics model. Here, the model demonstrates the motion defined by the position coordinates. The next block is the calibration block. This block executes motor movement of the testing apparatus based on user feedback. For example, if the user indicates that the cart is to the right of the zero point, the block will move the cart slightly to the left. In this manner, the calibration block can match each axis to the zero points.

The final block is the robot block. This block performs 2 functions. The first function is creating a custom sine wave based on the scaled positions and timing and sends this curve to the Arduino board to be performed on the testing apparatus. The second function is to output an Arduino IDE compatible file for running on the Arduino board at a later time. This file can be directly deployed to the Arduino board and the requested positions will be run on the testbed.

In this way, the testbed control software interfaces with the user's software, makes adjustments based on the user's needs, and runs the testbed accordingly.

3.4 Modular System

Chronos is designed as a modular system, a testbed that can be easily integrated into most CubeSat projects with minimal alterations. In order to create a system that

is consistently easy to integrate with new projects, every feature, both hardware and software, has to be simple at the higher level. On the hardware side, the key is to make a system that can be used for many types of testing and many types of payloads. Thus, Chronos features a mounting platform. This mounting platform allows for any payload to be mounted by simply adding a connector piece. This piece can be 3D printed based off the hole scheme of the OpenBuilds products. The piece simply needs to match this pattern and have a way to interface with the payload that wants to be attached. This can be a plate or a box, for example.

The second key to keeping the hardware modular is to make a system that can test multiple features. This is why the entire testbed with Chronos features two actual testing apparatus with the option to add more. The apparatus can be used together or as separate systems. This allows for both singular CubeSat missions to be tested where just pointing or payload performance is being tested and for interactive missions to be tested. This includes docking maneuvers, rendezvous, approach, and many orbital debris missions. The advantage of this design is that the number of interfacing components can vary depending on the mission. For example, ARAPAIMA only uses a single target, thus two apparatus are in use, but let's say there is a mission that requires two targets. An additional apparatus can cheaply and easily be added to the testbed. Thus, many different missions or just tests can be conducted with the same testbed.

The next key to modularity is the software. Many testbeds have a specific format that the input code needs to output or even be written in for the testbed to be of

use. The goal with Chronos was to be able to use any Matlab code or any code that could be read by Matlab through a compiler or the like. All that the testbed requires from the code is where the subject should be in space and how long it should take to get there (thorough timestamps, velocities, etc.). The code then takes this data and constructs the proper code for the testbed autonomously. This allows for "plug and play" software. There is no need to rewrite the code in a new language, just import the code and make sure that the software knows what portion of the code to read. This means that instead of it taking days or even weeks, like with some of the larger testbeds, the software integration should only take a matter of hours.

In addition, adjusting the software to different testbed setups is also simple and quick. When prompted by the code, the user simply selects which form factor apparatus are in use. This ensures that the scaling in the code is correct for the testbed being used. This also allows for apparatus of different sizes to be created as they are necessary, with the creation of a new size, only a handful of noted variables need to be adjusted.

In order to make a testbed that can easily be integrated for use with multiple cubesat missions with varying schedule margin, a system that could both meet the testing needs of the missions and the limitied time available to both test and setup the test wis necessary. Thus, Chronos was designed to be a modular system wherein it can easily be improved upon or changed to meet the mission needs and can perform tests with very little input from the code. This means that more testing can be integrated

at all levels of the development process with very little restrictions on the software or hardware maturity.

3.5 Protyping

Rapid protyping is used as a way to produce low-cost parts throughout the design process and many of these parts are used in the final version of the testbed. The type of rapid protyping used here is additive manufacturing where the parts are built up layer-by-layer rather than by cutting away material from a larger block. The more familiar name for the type of additive manufacturing used is 3D printing. Three different 3D printers were used to print the components used throughout this research. These printers are the Makerbot 2, the Makerbot Z18, and the Cubify. All three of these printers are using PLA at the time of printing. PLA stands for polylactic acid, this is a type of plastic or rather a polyester that is derived from Cornstarch (in the United States). For printing, it is molded into a solid cylinder with an diameter of around 1.75mm. For 3D printing, this is wrapped around a spool, generally consisting of 1kg of material, and referred to as filament. The printing works by melting this filament, which has a melting temperature of just above 300 °F, and moving the position of the melting filament as it falls. This allows designs to be layed out layer by layer with the melting filament.

Multiple components for the protypes are 3D printed. In the first iteration, the cart (with the exception of the wheels and the nuts and bolts) us completely printed. It is printed to determine if the parts fit and interacted as expected; however, it is

found that the cart is able to hold sufficient weight to perform tests. Thus, this printed cart is used for all initial testing of that particular design. In addition, a lot of the connector pieces in the second iteration are printed. While these pieces serve their purpose of allowing the prototype to be assembled, they are not up to par for any actual tests of the prototype. The roughness of the material made sliding across the pieces difficult.

Outside of the structural pieces that are printed, a few additions to the designs are also printed. One of the main additions is the phone case that is printed for the tests with the robotic arm. This part is modeled to fit a phone in a case and matched the mounting pattern to install the phone safely on the robot arm.

The final design of Chronos still features a couple of printed components. The feet that lift the rails to the appropriate height and level the apparatus are printed. Also, the control box that houses the electronics is printed. Finally, the mount used to connect the GoPro to the testbed is also 3D printed.

3D printing is a faster and most importantly, cheap way to work with design concepts and test the ideas in the early design process.

3.6 Calibration

In order to operate at the highest accuracy level, the testbed requires calibration at multiple levels. There are two main components that require calibration, Chronos and the motion tracking system. The calibration for Chronos is integrated into the main software. Before each run of the code, the user is given the option of whether or not

to calibrate Chronos first. The software then walks the user step by step through the calibration process. The process consists of both responding to prompts in Matlab and being hands-on with the Chronos hardware (with guidance from the software). For calibrating each joint, it starts with a question and answer prompt. It asks whether or not the plates are at the zeroed position on the axis. The markings for the zeroed position are shown in Figure 3.17. If the answer is yes, the software proceeds to the next joint; however, if the answer is no the software walks the user through fixing it. Fixing an unzeroed joint is completed by telling the code which direction it is off, if it is towards the positive or negative side of the axis. Once that information is fed back to the software, it will let you step the motor in the correct direction. With each answer, the software with have the motor take a single additional step until it is aligned. If the zero target is overshot, the prompt will ask and allow you to step back in the opposite direction. Once the joint is zeroed, it moves to the next joint. This process is repeated until all the joints are zeroed. At this point, the software gives you the option to start clean or run the code that you originally wanted to run.

The question here is once Chronos is calibrated, does it ever have to be calibrated again. The answer to this question is yes, the author recommends calibrating the machine after moving the location of Chronos, having an emergency stop occur, or before running any test where results are being recorded. This is due to the fact that small jolts to the machine can knock the rails off from the position where they were left off. Even if this jolt is not from an external force, if anything clashes or gets out of sync within Chronos, this can force the rails to become misaligned.

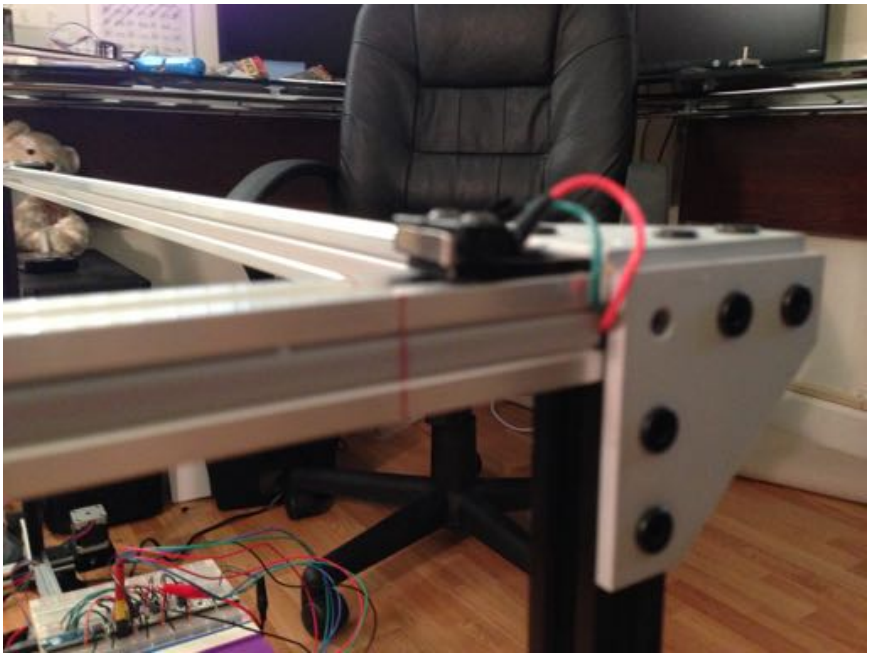


Figure 3.17. The markings on Chronos indicating the zeroed positions

The second area that requires calibration is the motion tracking system. While the motion tracking system does not need to be calibrated due to forces, it should be calibrated on first use. The calibration of the motion tracking system is the calibration required by the Kinect in order to read accurate depth measurements. This calibration procedure will tend to be quicker and simpler than calibrating Chronos. The Kinect calibration is a well-known and well-tested procedure. It consists of using a checkerboard pattern that can be printed out on a piece of paper and hanging it from a wall. The Kinect is then pointed so that the paper is within the field of view of the camera. The calibration code just needs to be run. Once the code is run and completed, the calibration process is complete.

4. Experiments

Many experiments are conducted throughout the development of the testbed. These experiments are crucial to choosing the best components and designs for the testbed. While smaller tests are conducted to test the performance of different components, there are also experiments to determine and document the performance of the system. All of these tests are conducted to prove the capability of the testbed and determine if the functionality meets the requirements to prove the hypothesis.

4.1 Design Optimization Tests

Many tests are conducted in order to create a stable, accurate system. The design is optimized through testing the differences caused by small design changes. While most of the tests conducted are too small to mention here, there are a few tests that are of more consequence to the overall design.

One such test solidified the decision of whether to have 2 motors powering the main, z-axis or a singular motor. After studying the performance results, the decision is obvious. Although there are additional factors to consider when using dual motors in synchronization (such as timing differences and jolting), there is was a marked improvement over the single motor performance. When using a single motor, the sides move unequally, causing position offsets and jamming. This is caused by the driving

motor favoring the side that it is located on. This issue is prevented by placing a motor on both sides to ensure that each side has the same amount of control. Many quick tests such as this one were conducted to prevent performance losses.

Another example of this type of testing is the testing for the motor placement design choices. This test is a bit more considerable and results in small changes in the design. Although these changes are small, they have a great compounded effect on the performance.

4.1.1 Axis Design Comparison

Before finalizing the setup for the motor placement in Chronos, a test was performed to determine the best option for the motor placement. This test looks at the accuracy of the movement when the rail is both level and placed at a variety of angles. This test is conducted using a single motor and single rail and the rail is rotated while the trolley is in motion. In addition, the level tests were conducted two ways, with no additional weight and with a 10 lb power supply serving as the maximum test weight. The test setup with the additional weight is shown in Figure 4.1. A GoPro was used to record all of the testing. The best performing motor setup is determined by calculating the distance that the trolley actually moved on the rail to the theoretical distance that it should have moved based on the rotation of the motor. A meterstick is placed behind the rail during testing which allowed for an accurate distance measurement to be determined using the meterstick for scale in the GoPro footage. From this point, tests of varying distances and angles could be completed. Though this test was simple

in nature and quick to execute, it was the basis for making Chronos as accurate as possible in the available constraints by considering accuracy in each design choice.

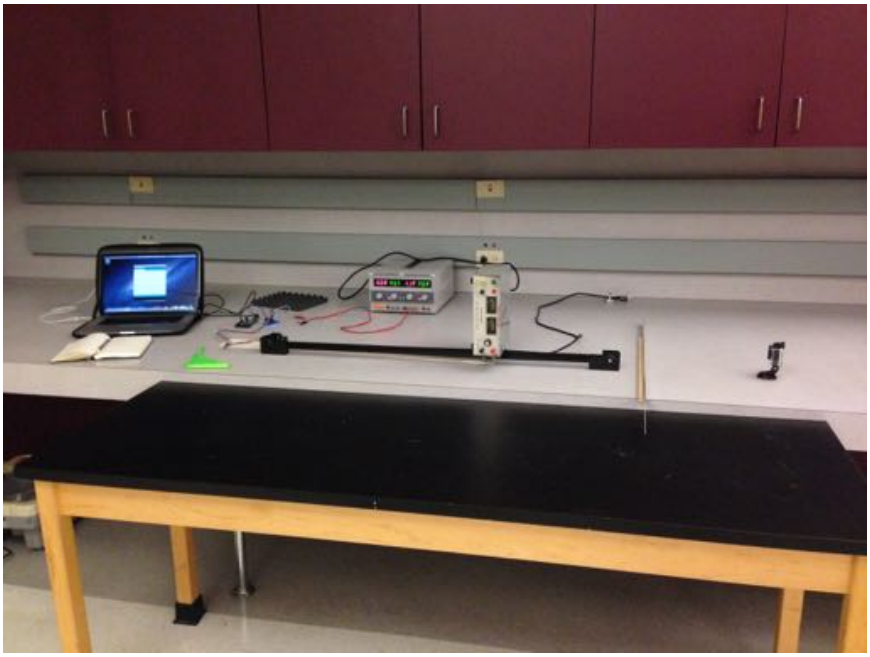


Figure 4.1. The axis design comparison testing with added weight

4.2 Accuracy Testing

Accuracy testing is completed on Chronos to determine the accuracy of the transformation between stepper motor steps or rotations and the linear motion, the accuracy of the actual movement of Chronos along each axis of motion, and the precision of these movements. Testing is completed through performing a series of motor commands from the Arduino IO interface to the motors. From here, the response of the linear motion is recorded. There are two main test performed along each axis, the first is a simple precision test. Chronos is commanded to move either 4

cm or 2 cm (depending on the overall rail length) on each axis individually. For each command it moves 4 cm or 2 cm, pauses, moves again, pauses, etc. until it reaches the end of the rail. In this way, the consistency of the motion can be seen with the move command. The second test performed on each axis is a longer motion. The axis is commanded to move at increasing lengths for each movement. Between each motion the axis is reset to the zero point before the next movement occurs. Each movement is 4 cm longer than the previous motion. The longest command is the length of available rail space.

On the z-axis, the precision test performed 18 repetitions of 4 cm motions. This occured in both the positive and negative z-axis. This test was repeated five times. On the y-axis, the test contained 17 repetitions of 2 cm motions and on the x-axis, the test contained 18 repetitions of 4 cm motions.

The length accuracy test on the z-axis ranged from 8 cm to 72 cm with a difference of 4 cm. The y-axis ranged from 8 cm to 36 cm and the x-axis ranged from 8 cm to 72 cm, both with 4 cm differences in length between each movement in the series.

In addition to these two tests, a couple of maneuvers are also tested to study the accuracy of the testbed in a full motion simulation. The two maneuvers studied include replicating a helix shape and a repeated straight line climb and decline.

4.3 Simulation Experiment

4.3.1 Concept

This experiment is used to demonstrate the capability of the Chronos testing apparatus in a real-life situation, modeling an actual satellite using accurate positioning data. The satellite, or rather satellite pair used in this case study is Orbital Express. Orbital Express is a DARPA program used to "validate the technical feasibility of robotic, autonomous on-orbit refueling and reconfiguration of satellites"(Initiative, 2007). Orbital Express consists of two spacecraft, ASTRO which is the servicing satellite and NextSat which is the client satellite. The Orbital Express program demonstration includes transferring propellant between the two satellites, detaching and reattaching components both to the same satellite and between them, satellite grappling, and multiple rendezvous and capture scenarios (Initiative, 2007).

For the studies, the TLM data for both satellites is acquired from space-track.org. This website stores and updates TLE data for many different satellites and store them in a catalog. In order to get the data for a particular satellite, the catalog number can be used in the search. The catalog numbers for ASTRO and NextSat are 30772 and 30774, respectively. The TLE format contains a multitude of information about a satellite's position at a particular time. Some of this information includes the element set epoch, the orbit inclination, the right ascension of ascending node, the eccentricity, the argument of perigee, and the mean anomaly. From this information, it is possible to recreate the motion of the satellite over time.

4.3.2 Code

Through the use of sample code from (Vallado & McClain, 2001), a code is generated to convert the TLEs into a variable holding multiple satellite orbital elements. Using a code from (Vannatta, 2009) as a baseline, the correct variable and orbital elements are extracted from the twoline2rv code. These elements are then used in a second code based on (Vallado & McClain, 2001)'s randv code to convert from the elements to positions in the i,j,k frame. Once in the coordinate frame, the positions can be directly interfaced with the testbed code, along with the array of corresponding times from the twoline2rv code. This code is created such that it is a versatile addition to the testbed code library. The code will work for any set of TLEs saved in a text file.

4.3.3 Application

In this study, three different scenarios are modeled on both the virtual simulator and Chronos. The first scenario is the relative motion between the two satellite over a one year period. Due to the nature of the mission, the relative motion is generally very small, thus most of the motions are hardly noticable. The second scenario is the motion of ASTRO also over an one year period and the third scenarios is the motion of NextSat over the same time period.

This study allows for a true representation of a testbed orbit simulation. As opposed to recreating patterns or shapes, the testbed has to replicate actual satellite motion. While the performance of Chronos is not measured numerically during the

simulations of the individual satellites, it is visually checked for accuracy. The results of the relative motion test are studied numerically. Both the virtual simulator and Chronos motion were compared to a STK simulation of the same satellite motion based on the same telemetry points.

5. Results

5.1 Accuracy Results

In the precision tests, the z-axis demonstrated an overall 0.5% error. The error is 0.4% in the positive z direction (Table D.1) and 0.6% in the negative z direction (Table D.2). The highest error seen on any one step is 2.5% error, which correlates to a 0.1cm offset. The mode of the data is 0.0% error. The y-axis has an overall 2.3% error, with 2.4% in the positive y (Table D.3) and 2.1% in the negative y(Table D.4) directions. There is a lot more variance in the y-axis data as compared to the z-axis, values ranged from 1.7 cm to 2.2 cm for the commanded 2 cm movement. It should be noted that the error increases in the last few centimeters of the positive y-axis. The x-axis shows an average percent error of 0.5%. The average error is 0.6% in the positive x direction (Table D.5) and 0.4% in the negative x direction (Table D.6). The error is always 0.1 cm less than the target when there is an error. The error for the precision tests for the x, y, and z axes can be seen in Figure 5.1, Figure 5.2, and Figure 5.3, respectively.

In the length accuracy tests, in the z-axis the overall accuracy was 0.1% (Table D.7). The range of accuracy errors is -0.6% to 0.6%. The y-axis has an overall average error of 1.9% (Table D.8) and the x-axis has an overall average error of 0.1% (Table D.9).

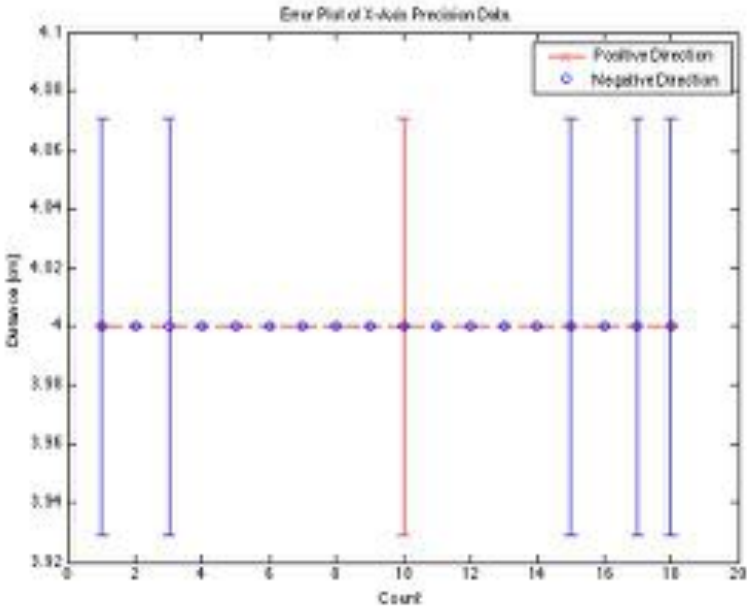


Figure 5.1. The plot of error for the precision tests for the x-axis

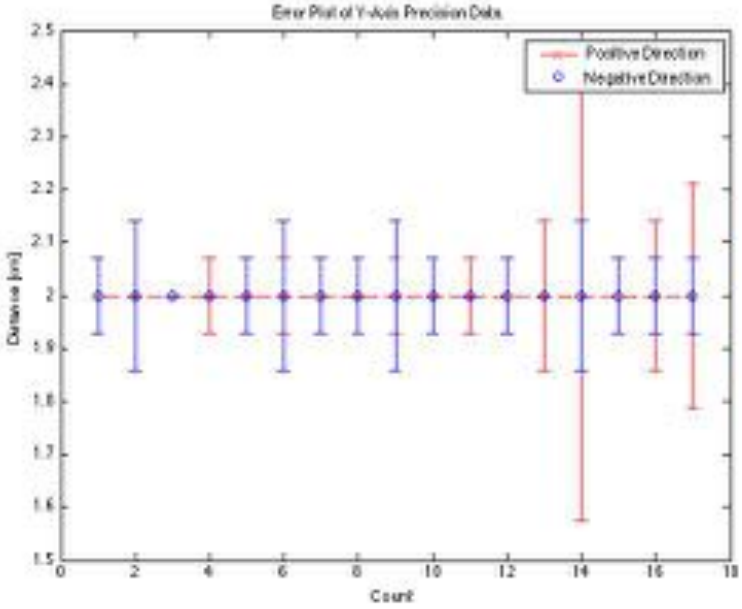


Figure 5.2. The plot of error for the precision tests for the y-axis

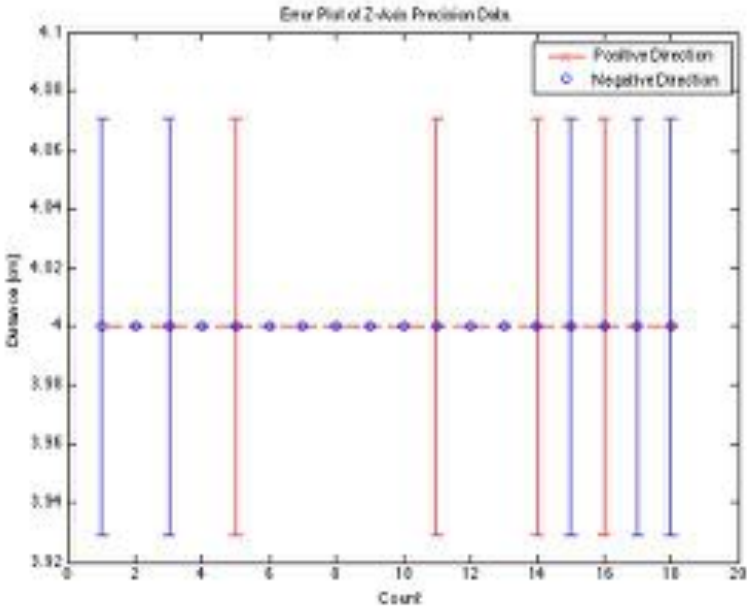


Figure 5.3. The plot of error for the precision tests for the z-axis

The error for the length accuracy tests for the x, y, and z axes can be seen in Figure 5.4, Figure 5.5, and Figure 5.6, respectively.

For the stright line and helix maneuver testing, the accuracy at each data point ranges from approximately -0.2 cm to 0.2 cm for each of the tests. The overlay of the actual testbed movement over the commanded positions can be seen in Figure 5.8 for the helix and Figure 5.7 for the straight lines. Additionally, plots of the actual error, the difference between the command and actual positions can be seen in Figure 5.9, Figure 5.10, Figure 5.11 for the helix maneuver x, y, and z axes, respectively and Figure 5.12, Figure 5.13, Figure 5.14 for the x, y, and z axes for the straight line maneuver, respectively.

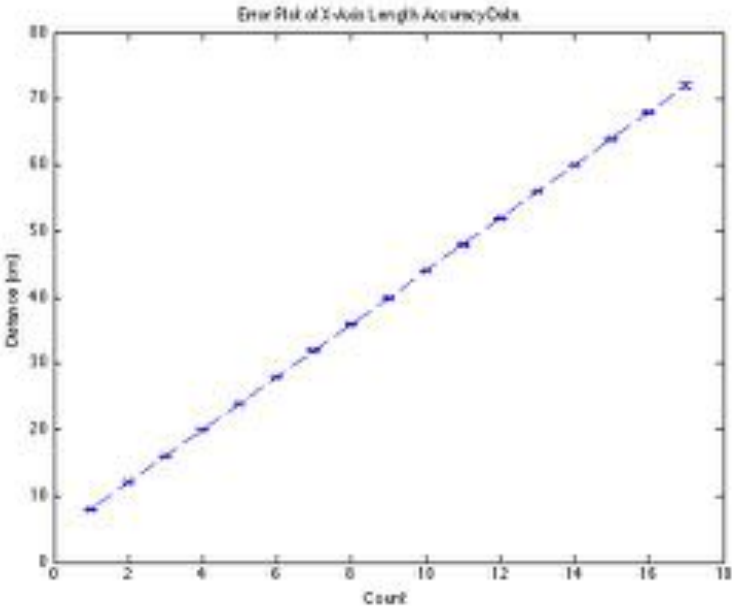


Figure 5.4. The plot of error for the length accuracy tests for the x-axis

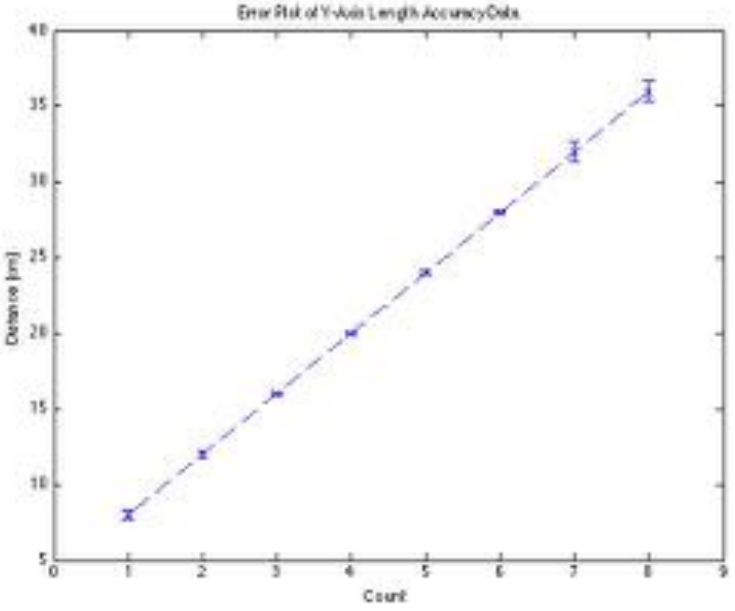


Figure 5.5. The plot of error for the length accuracy tests for the y-axis

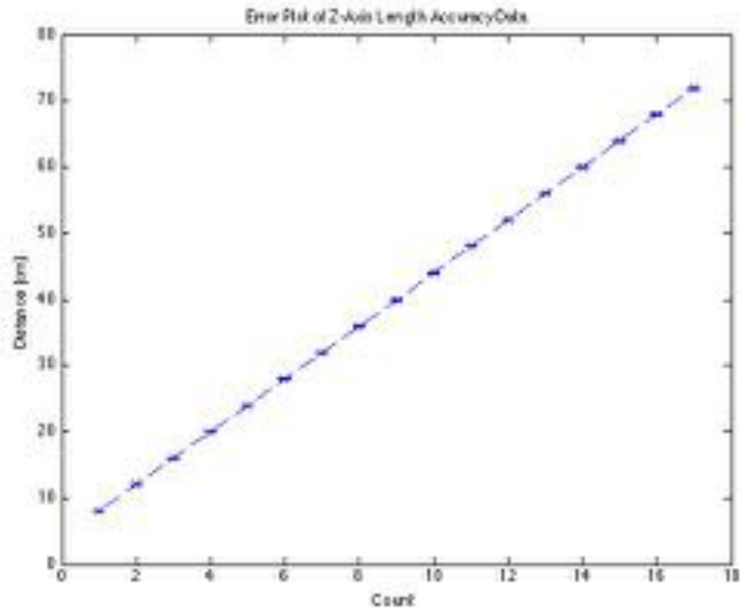


Figure 5.6. The plot of error for the length accuracy tests for the z-axis

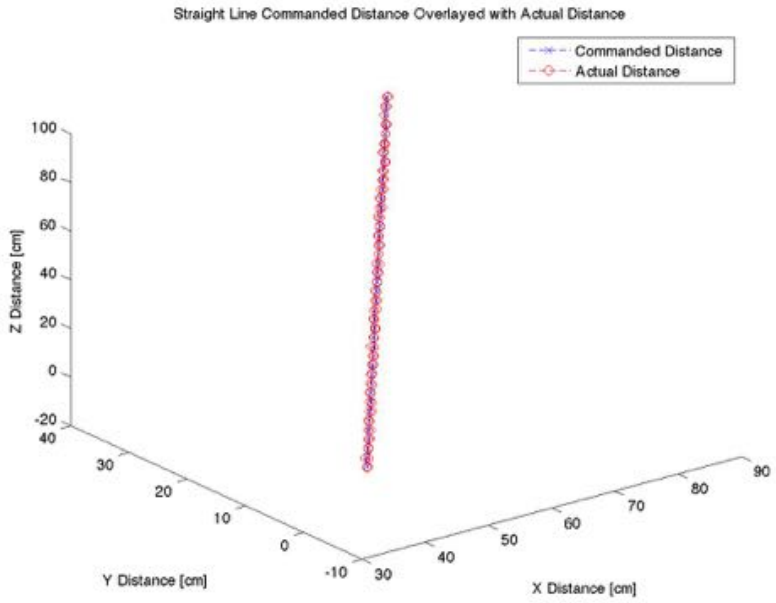


Figure 5.7. An overlay of the commanded and actual positions for the straight line maneuver

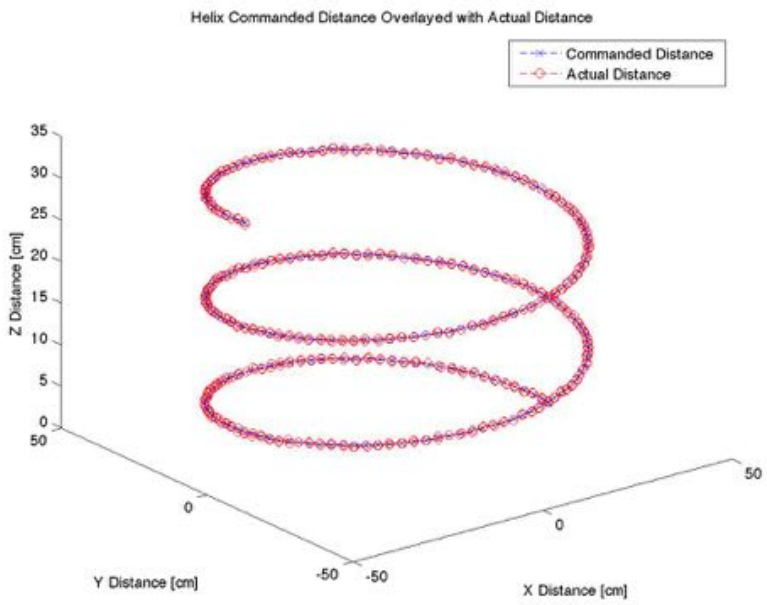


Figure 5.8. An overlay of the commanded and actual positions for the straight line maneuver

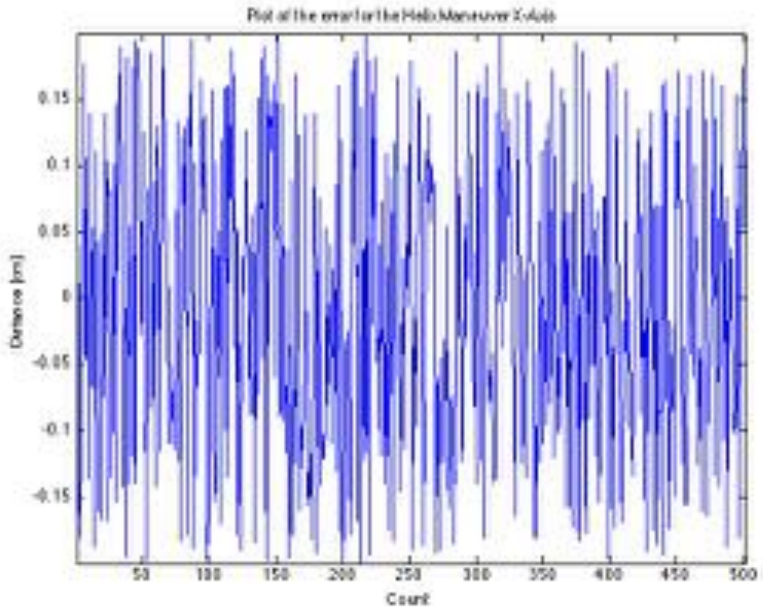


Figure 5.9. The x-axis error for the helix maneuver

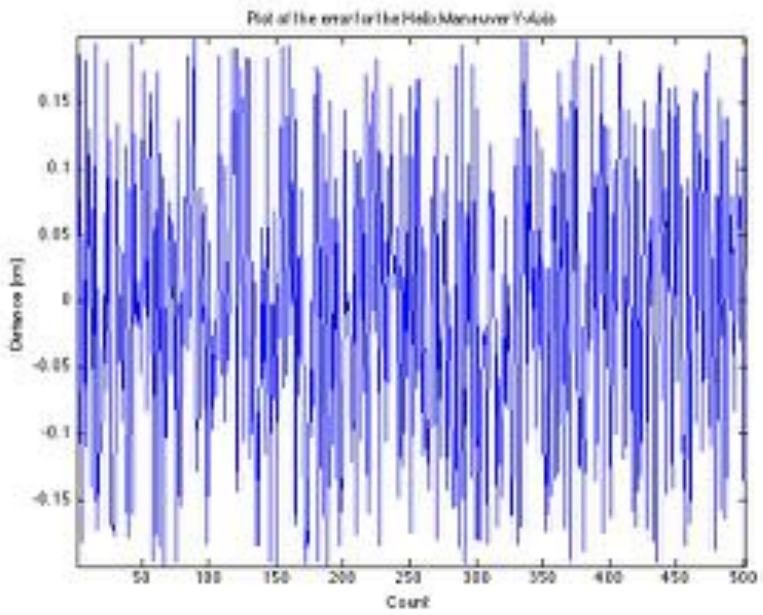


Figure 5.10. The y-axis error for the helix maneuver

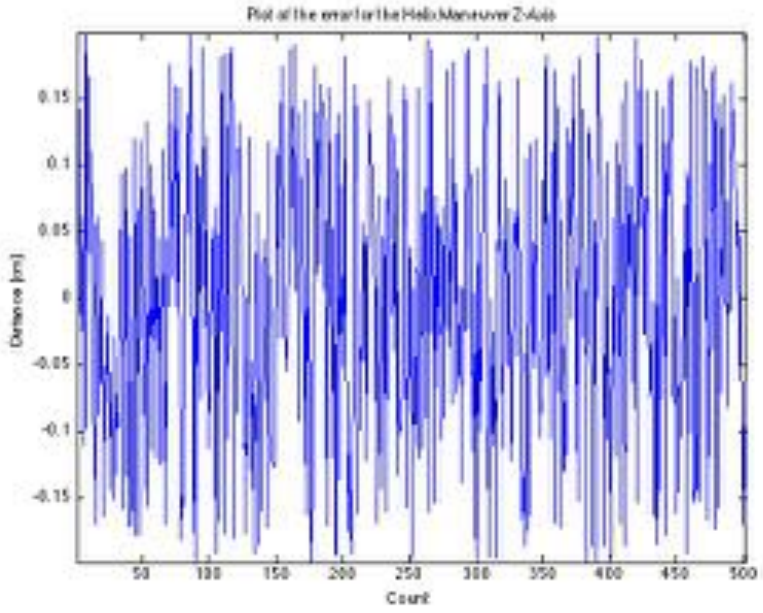


Figure 5.11. The z-axis error for the helix maneuver

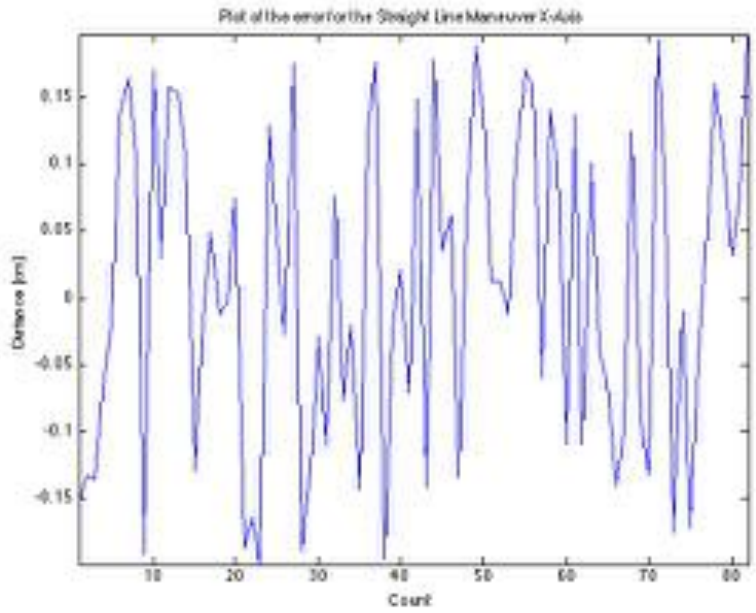


Figure 5.12. The x-axis error for the straight line maneuver

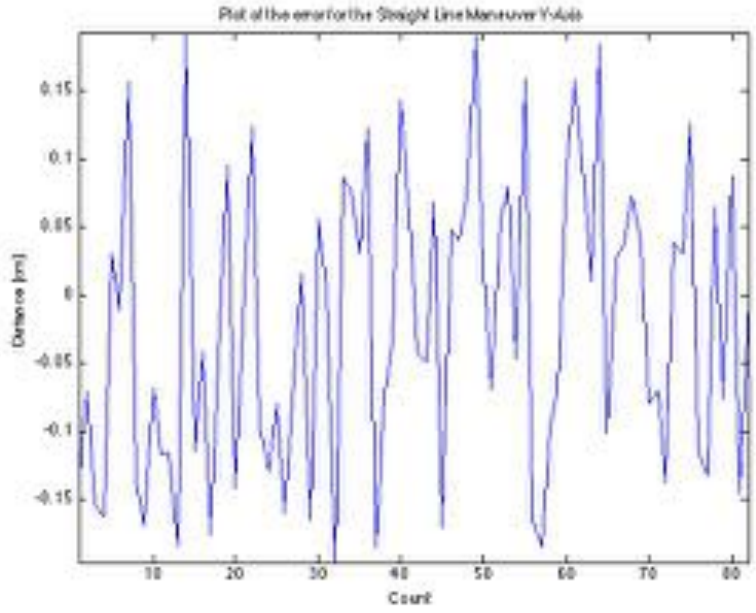


Figure 5.13. The y-axis error for the straight line maneuver

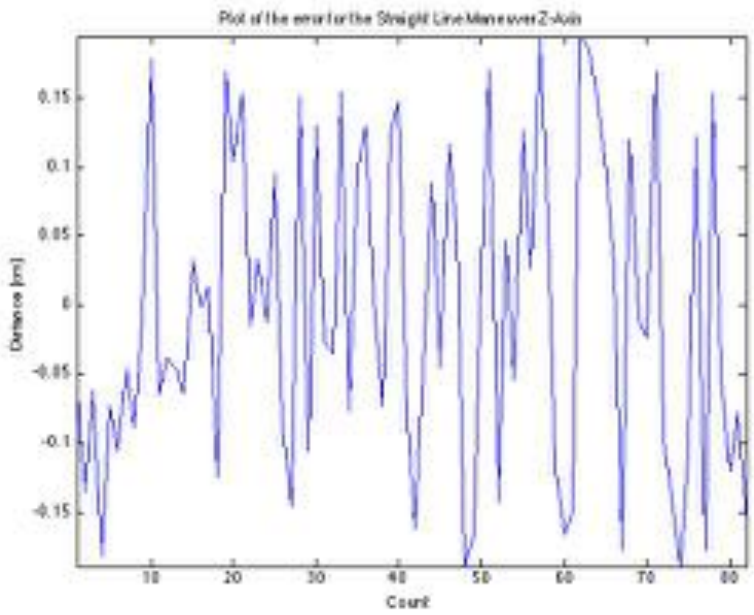


Figure 5.14. The z-axis error for the straight line maneuver

5.2 Orbital Express Results

In the relative motion test of Orbital Express, the motion between Astro and NextSat are plotted in the x (Figure 5.15), y (Figure 5.16), and z (Figure 5.17) directions. The error is then studied over each of these axes. The error never exceeds 0.2 cm or -0.2 cm, with the bulk of the error centering around 0 cm. The error in each axis can be seen in Figure 5.18 for the x-axis, Figure 5.19 for the y-axis, and Figure 5.20 for the z-axis.

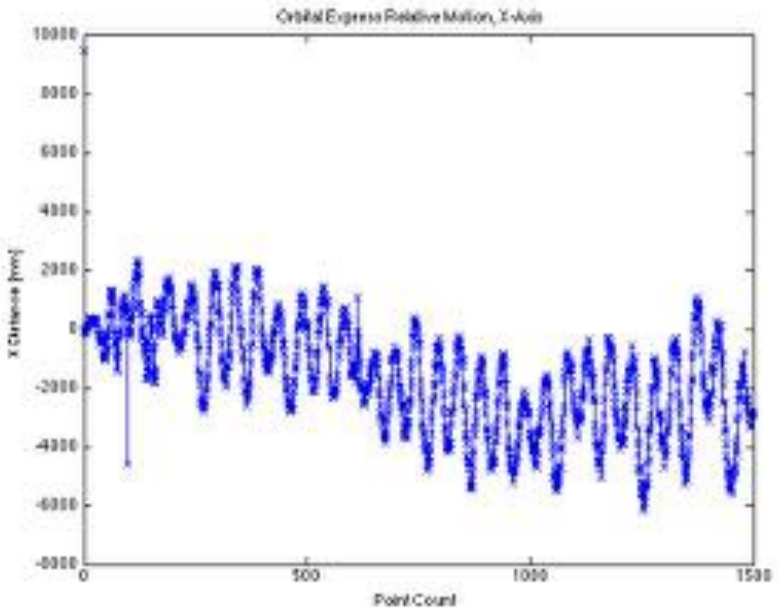


Figure 5.15. The x-axis motion of the Orbital Express relative motion test

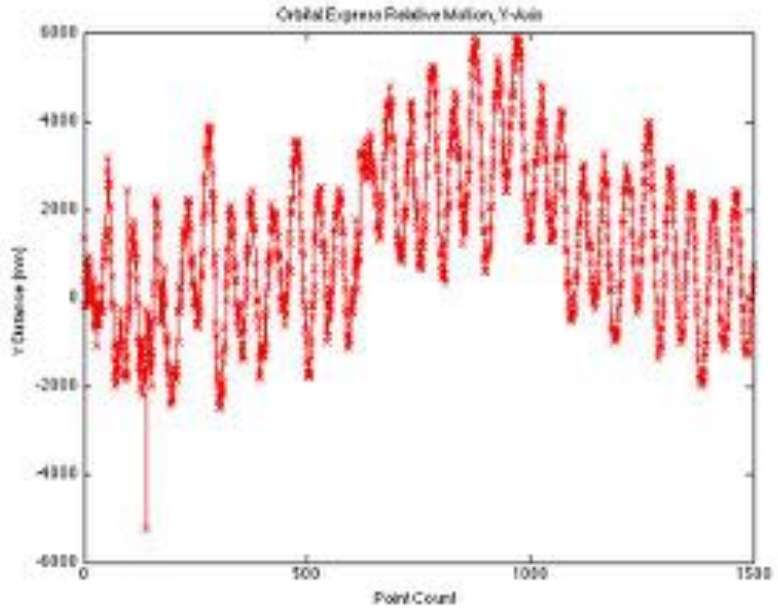


Figure 5.16. The y-axis motion of the Orbital Express relative motion test

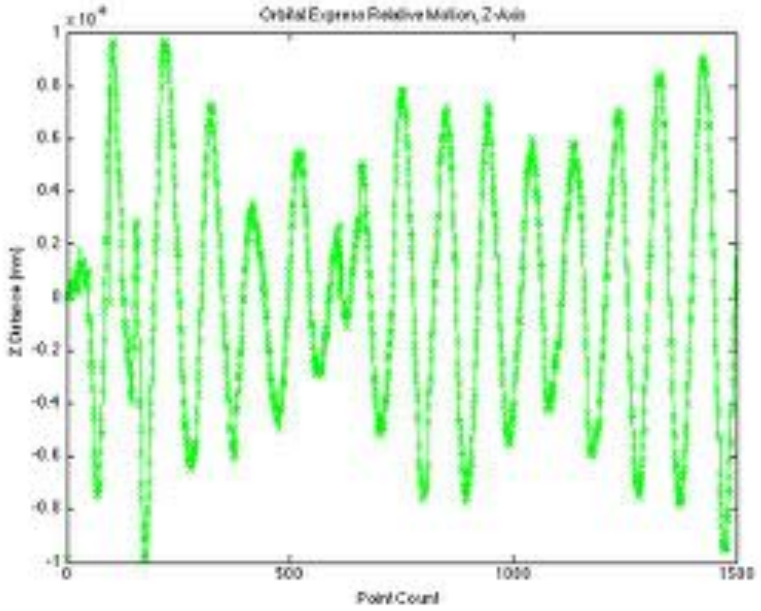


Figure 5.17. The z-axis motion of the Orbital Express relative motion test

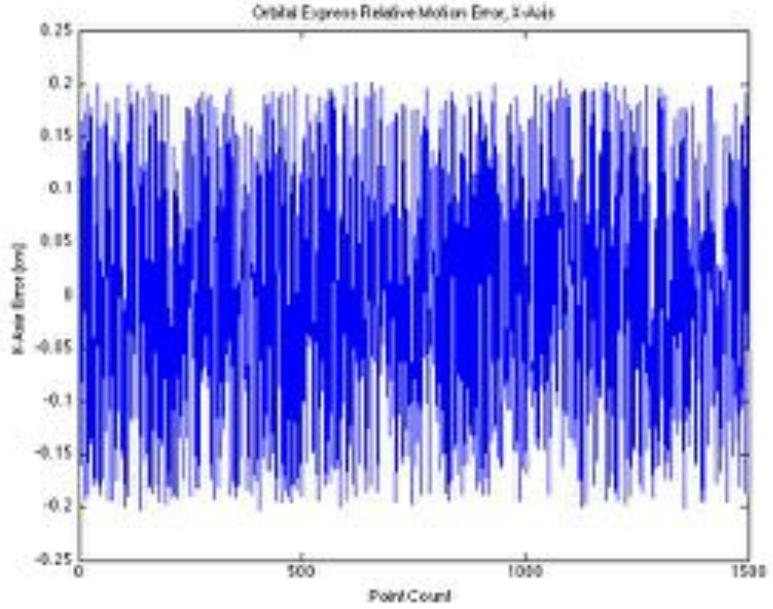


Figure 5.18. The x-axis error of the motion for the Orbital Express relative motion test

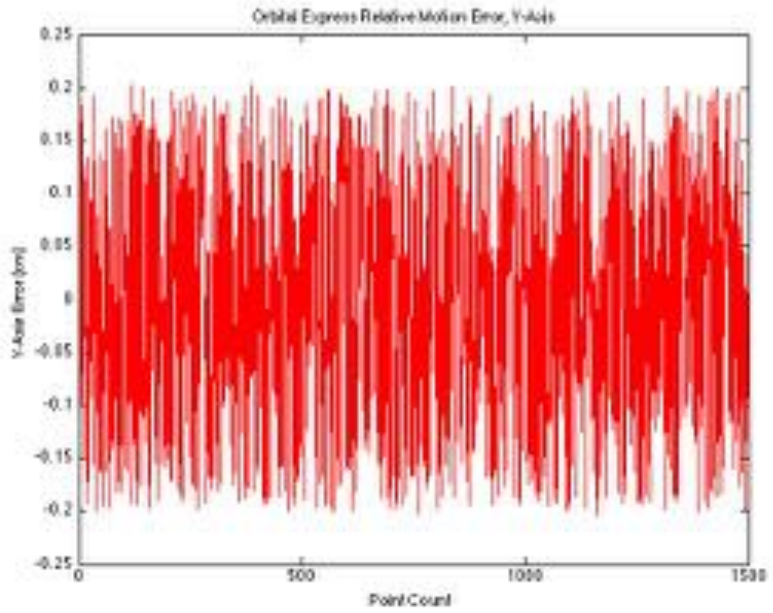


Figure 5.19. The y-axis error of the motion for the Orbital Express relative motion test

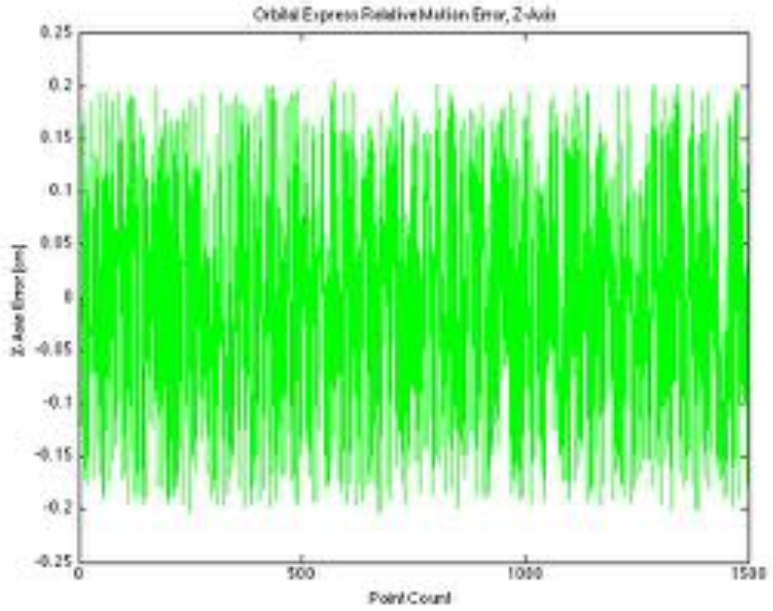


Figure 5.20. The z-axis error of the motion for the Orbital Express relative motion test

6. Discussion, Conclusions, and Recommendations

6.1 Performance

The overall performance of the testbed is as expected. The x and z axes, have an accuracy of less than 1%, which matched with the specified accuracy of the motors minus a slight loss in the belt and pulley system. The y-axis demonstrates a significantly depreciated accuracy as compared to the x and x axes results. A probable cause of this loss in accuracy can be attributed to the additional weight that is present on the y-axis. The weight of the motors and the components that are moved along with the movement along the y-axis could place more of a strain on the holding torque of the motors. A good indicator of this is that the tests where continuous motion over longer distances occurred has a increased accuracy when compared to those where the motors are repeated stopped and restarted.

For the basic maneuver testing and the replicating of Orbital Express, the testbed performed satisfactorily. It is able to accurately replicate the maneuvers in real-time. In addition, the testbed is able to remain consistent over longer tests. The Orbital Express tests had over 6000 position commands. Overall, the performance of the testbed is on par with the expectations.

6.2 Good Practices

While Chronos has its own specific upkeep, the testbed as a whole has a few recommendations for good practices to follow. The first is to keep the testbed as dust-free as possible. Dust buildup can add slipping or additional friction along the tracks. Dust can also skew the results of the Kinect when there is dust on the camera lens. When moving the testbed there are two recommended options, depending on how far the testbed is being moved. If the move is short, like from one room to another, it is recommended to move each piece separately. It is recommended to use two people to move the testbed, keeping it as level as possible while moving. It is also suggested to use a rolling cart rather than hand carrying Chronos, this is to reduce the stress on the joints. For longer moves, it is recommended to wrap Chronos in bubble wrap in a way that keeps any of the parts from moving and protects the rails from dents or scratches. Optionally, Chronos can be bubble wrapped and packed in a large box, allowing for shipping. The Kinect should also be protected while moving to prevent scratches on the lenses. Regular maintenance is suggested on where the Kinect is positioned compared to the testbed. Check to make sure the distances are correct, make sure the Kinect is not tilted, and check the overall state of the testbed.

6.2.1 Expected Robot Upkeep

All screws on the testbed will require periodic tightening. It would be expected that the screws will start to loosen noticeably after every dozen times that the machine

is used. Whether it is planned on a regular basis to perform tightening of the screws or only when wiggling on the testbed is noticed, tightening will be required at some point. The number of cycles that the testbed can be put through varies with the length of the tests and the number of direction changes that occur, thus the exact point at which the screws will require attention is impossible to pin-point. The time between necessary checks can be lengthened through the use of Loctite; however, even with the Loctite applied the screws do still loosen.

In order to tighten all of the screws, three hex keys are required. The sizes of these key are M3, M5, and M1.5. The smallest key will be used on the timing pulley set screws and the mounting hub set screws. The M3 hex key will be used on the screws for the motors and the M5 hex key will be used for all remaining screws. It is only recommended to make the screws hand tight, if they are over tightened, especially unevenly, the testbed will start to shake and become misaligned at the corresponding joints, as well as the possibly of bending different parts.

Calibration is also a part of the expected robot upkeep. The two items to be calibrated, the motion tracking system and Chronos, have different suggested periods of time between calibrations. It is recommended that the motion tracking system be calibrated once every 50 uses. This assumes that the same Kinect is in use the entire time. Calibration is also required for each new Kinect in the system. It is recommended that Chronos be calibrated every time that a test with recorded result is being run. This will increase the accuracy of the results dramatically. It is also

recommended to calibrate Chronos whenever it is moved or the moving rails are bumped.

In addition to calibration, Chronos requires leveling for the best performance. The adjustable feet can be loosened or tightened depending on how level the surface is that Chronos is sitting on. Levels are integrated into the frame of Chronos to allow for easy visual inspection of the current state.

The final expected upkeep for Chronos is periodic tightening of the belts. Since a belt and pulley system is for the basis for movement, the belts being tight around the pulleys on each end is integral to the performance of the system. Having the belts too loose can result in inaccurate movement of the trolley or at worst, the trolley won't move at all because the pulleys won't catch the belt. There are two ways to solve the problem of a loose belt. The first is to install a torsion spring, this will ensure that the belt is tight until the excess length reaches a certain point. The second option is to unsecure the belt at its ends and trim the belt to size, then re-secure the belt. I would consider both of these solutions to be temporary solutions, both in that they will have to be performed periodically and that after a while the solutions will cease working. After the belt loosens, the grips that fit into the timing pulley will start to become stretched as well. This results in the pulley/belt interface missing interactions since they won't line up. On the trolley side, this missing interaction is seen as unsmooth motion, or skipping. This yields very inaccurate movements. At this point, the belt will need to be replaced with a new one.

6.3 Conclusions

The purpose of this study is to prove the hypothesis that a low cost hardware-in-the-loop testbed can be designed such that it can be an easily accessible tool for university CubeSat programs while being developed in a way which makes it a universal testing option for varying CubeSat mission types and university programs styles.

The results shown in this research demonstrate that at a final total cost of $1299.03 and having dimensions of 1x1x0.5 meters, the testing apparatus is capable of replicating spacecraft maneuvers on a small-scale. The apparatus has suitable accuracy levels to simulate satellite maneuvers to a reasonable level for basic hardware-in-the-loop testing, while maintaining a small form-factor. In addition, the testbed has proven capable of performing multiple types of maneuvers concluding in replicating an actual rendezvous and docking mission.

The hypothesis of this study has been proven to be true. The testbed developed through this research has be proven to be cost efficient and reasonably accurate while maintaining a small dimensional footprint.

6.4 Recommendations and Future Work

While the testbed design is currently sufficient to serve the testing needs of a low budget cubesat mission, there are many improvements and expansions that would improve the testbed overall. There a quite a few additions that would improve

the performance of the testbed, as well as some additions that would increase the capabilities of the testbed. In addition, based on the results of the testing completed with the testbed, some changes have also been recommended to improve the results of the tests.

6.4.1 Improvements

There are a few improvements that would increase the overall performance of the testbed. One of the main issues with the testbed is the vibrations. Although the vibrations of the motors themselves are barely noticeable when the are standalone, the vibrations are amplified through the testbed. This has two major unintended consequences; the first of these consequences is the noise. The vibrations cause the testbed to produce a very loud, low hum when it is in operation. The second consequence is that the vibrations cause the screws and nuts to loosen quicker than normal. Nuts can frequently be seen falling off during a test. The improvement necessary here is to dampen the motor vibrations.

The second improvement is in the control software. With the current available interfaces between Simulink and the Arduino, microstepping of the motor occurs, unintended, whenever the motor is in operation. The result is that more steps of the motor need to occur in the same span of time to complete the maneuvers properly. Unfortunately, due to the rotational speed limits of the motors, in many cases it is impossible to step fast enough to account for the presence of microstepping. The improvement here would be to either create a software option to prevent the

microstepping or work with a different electronics setup in the system. For example, using an Arduino motor shield instead of the motor driver to control the motors.

6.4.2 Enhancements and Expansions

Since the development of Chronos, the OpenBuilds group has since developed a new type of rail. This rail is known as the C-Beam and features a rounded edge as compared to the V-Slot rail. This means increased strength in the rail itself. In addition, this rail is constructed of 6063 T–5 Aluminum with a smooth, clear finish. This new finish allows for smoother interfacing with the wheels. Since the C-Beam is compatible with any of the parts used for the V-Slot, it can be easily exchanged with the current rails. It is recommended that this rail be used as an alternative to the 20x40mm V-Slot rail. The C-Beam allows for new configurations which may be better suited for the rotating rail.

Another enhancement would be the addition of ping sensors. These sensors connect with the Arduino and use ultrasonic pings to locate objects. If these sensors are used in conjunction with the limit switches, additional safety features can be added. Since the sensors can determine distance, the motors can be slowed down once they pass a certain threshold. This would prevent the trolleys from crashing into the limit switches and abruptly cutting off power to the motors while the software is still running. A large benefit is that this could be integrated as feedback into the software to allow for the system to be self-correcting during testing. If the trolleys are traveling more or less distance than expected, the ping sensor can alert the software to the distance

differential. This allows for two sources of feedback in the system, the ping sensors and the Kinect.

An additional benefit to self-correction using the ping sensors is using them as a way to self-calibrate Chronos. Since the ping sensors can determine the distance from a set point, instead of walking through the calibration procedure step-by-step, the user can simply tell the system to calibrate. Using the distance measurements from the ping sensors, the software would be able to command the motors to step the equivalent number of steps to the difference in distance being measured by the sensor from the trolley to the zero point. This would make calibration easier and thus make it more likely that the user will calibrate as often as recommended or even more often; this would mean more consistent, accurate results.

The addition of multiple Kinects into the system has been an idea throughout the development process of the testbed. The idea is that having more Kinects could mean the use of triangulation to get more accurate depth measurements. Each additional data point would add to the accuracy of the overall feedback loop. The difficulty here is making sure that the depth sensors don't interfere with each other. It has been seen that the accuracy results from the Kinects actually decrease as more Kinects are added. Thus, the trick here would be to either position the Kinects so that they do not interfere or toggle the Kinects so that they are never on simultaneously.

An alternative to using the Kinects is to use a professional level motion tracking system. Some of these systems include little sensors that you mount of the target. These sensors are specifically sought out by the tracking system so that the system

tracks only the motion of the sensors and is not sidetracked by the presence of other objects or lighting levels amongst other conditions. As a bonus, some of the sensors also contain gyroscopes and accelerometers. This means that not only are you externally tracking the sensors, but they can also tell the user the velocity with which they are moving. This system would give more types of feedback and more accurate feedback then the Kinect system.

SimMechanics is a wide range tool that has many useful features that the control software is yet to take full advantage of. One of the features that will add accuracy to the virtual robot is the use of friction. Since each part is modeled with the correct material properties and finishes, the properties get transferred into SimMechanics as well. Currently, nothing is being done with these properties, they are just being stored. SimMechanics has the power to use these properties to calculate items such as static and kinetic frictional coefficients in order to more accurately simulate movement. In order to take advantage of this feature a few things would have to be altered in the model. The largest change would be that currently the joints that are being actuated are those of the actual rails; thus the rails are being moved back and forth rather than the rails moving because they are connected to the moving trolley. In concurrence, the virtual robot can be maneuvered through using friction and actuating the wheels of the trolley. As an addition, belt systems can be added to account for all the factors that may come into play by actually stepping the motor, having the motor spin the belt, and having the belt move the trolley. This would add some additional factors such as belt stretching to the simulation, which can be a big deal.

Another enhancement that can be completed in SimMechanics is taking advantage of the video features that it has to offer. One major application of this feature would be to place the "camera" on the cubesat simulator and be able to view what the simulator or payload is pointing at. This allows the user to ensure that the cubesat is actually pointing at its target properly and give the user the ability to create live demonstration videos. This can also be used for other purposes such as viewing docking maneuvers or practicing joystick maneuvering.

The upgrade from the current stepper motors to encoded motors is another recommendation. The encoded motors will give feedback on the positioning of the stepper motors. This is invaluable information since in the current setup, it is reliant on the idea that the motors are always turning the correct amount and never slipping. This is a dangerous assumption, many factors can attribute to slipping or under-stepping. One of the largest factors of concern is the weight that the motors are holding, the more weight, the higher the risk. Encoded motors account for these errors by monitoring the movement of the shaft and being able to implement corrections based on the feedback.

Upgraded power supplies are another enhancement to the current design. The suggestion would be to use professional quality power supplies with individually controlled outputs. This would allow the user to ensure that each motor is receiving the optimal amount of power. The benefits to this enhancement include making sure that the motors are capable of stepping at the desired speeds, if the motor isn't receiving enough power; it won't perform to the specifications. Additionally, the user

would be able to monitor the current draw of the motors, alerting the user to when a motor is over stressed. This allows for corrective actions to be taken before any failures occur.

An expansion of the system would be to make it wireless. In the current configuration, the wires prevent full rotations and certain maneuvers. The drag chains housing the wires also provide resistance to the rails during motion. Ideally there would be no wires in the system, but even reducing the number of wires would be an improvement. One concept for going wireless is to electrify a metal strip within the inside slots of the rails. The receiving component of the electricity would use brushes that drag along the inside of the rails to conduct the necessary power. This would eliminate all of the wires in use for power. The data wires can be eliminated through another concept. The Arduino boards have the ability to add Wi-Fi, this could be used to communicate with each of the boards. Currently, this communication is completed through a USB transfer from the main computer. Instead, the computer and each of the boards would be on a specified Wi-Fi network over which the data will be transferred to the appropriate Arduino board. With this addition, data wires would no longer need to run back to the main computer. Another alternative to using Wi-Fi is to use Bluetooth. The Arduino boards also have Bluetooth add-ons to add Bluetooth capability. Thus, as long as the computer is Bluetooth enabled, the computer would be able to communicate with the Arduino boards.

This particular enhancement does not add to any of the existing components, it instead adds another component to the testbed. A sunlight simulator would be a

motorized sunlamp that moves as programmed by the main computer. The goal of the sunlight simulator would be to mock the position and lighting effects of the sun. Although it is nearly impossible to replicate the effects of the sun, this simulator would allow the user to see new data that might not have been previously considered. Some of this data could include blinding by the sun and where the payload cannot operate or cannot operate accurately due to the sun's position. Another aspect is the shadows cast from the sunlights position, causing eclipses. The addition of a sunlight simulator would add new resources for missions.

Another enhancement is a way to solve both a safety and a mission condition dilemma. In many cases, the payload being tested should not be operated with people in the target range or the room needs to be completely dark for testing, among other conditions where being right next to the testbed might not be advised. The suggested enhancement is wireless monitoring where cameras are placed around the testbed and live footage is streamed to a nearby computer or recorded for later use. This will allow the user to monitor the testbed from another location and know if the test is going awry.

6.5 Applications

The applications of the testbed are really the purpose of its development. While the hardware and design can be applied to other devices, such as a 3D printer or CNC machine, the main application of the testbed is to test CubeSats in a university setting. Since the design of the testbed is very modular, the same technology is very

easily applied to different CubeSats at different universities with a variety of missions. The main application for the particular testbed development for this research is the testing for the ARAPAIMA CubeSat.

6.5.1 ARAPAIMA Integration

Since Chronos was designed around the needs of a mission such as that of ARAPAIMA, there are many plans for how Chronos will be used throughout the development process. The first major use will be to use Chronos to serve as the ARAPAIMA CubeSat and have the secondary apparatus, the RSO apparatus, mimic the movements expected of the upper stage target for ARAPAIMA. This particular concept is broken down into multiple levels of testing, but the main goal of these tests is to confirm that the algorithms work and that the payload instruments can perform as a unit.

The first level of testing includes using just the Chronos main testing apparatus to mimic the planned approach movements of ARAPAIMA. The approach maneuvers will be completed with a 3D printed RSO as the target. In this test case, the target will be stationary. The purpose of this test is to simply test the payload components on the approach. The next level of testing is to add rotation to the RSO. Thus, the RSO apparatus will be used to simulate a single degree of rotation about the y-axis. The third level of testing is to have the RSO apparatus fully emulating the expected tumbling of the rocket body upper stage. During this time, the mock ARAPAIMA is emulating approach and proximity operations between itself and the upper stage. The payload is collecting data through the images taken of the RSO and the distance

information from the laser rangefinder. This allows the ARAPAIMA team to see the types of information gathered about the mock RSO and how the algorithms are processing the data. Completing this testing on an interactive testbed allows for problems with the code to be weeded out early.

The next stage of testing takes this idea a bit further. Once the flight software is complete, the on-board computer will be added to the hardware on the testing apparatus. The on-board computer will be communicating with the control computer for the testbed. This means that the ARAPAIMA hardware is in a complete loop with Chronos. It can not only perform operations, but it can also send feedback into the main control code. The idea here is that the mock ARAPAIMA will be imaging the mock RSO with the image control being provided by the on-board computer. Once the team thinks that a satisfactory amount of information has been collected by the on-board computer, the control software will allow for input from the on-board computer. This is the true test of whether or not the flight software is working.

The flight software is required to calculate the next position required to travel to, or rather a planned path to perform particular maneuvers to image the entire mock RSO based on how the RSO is moving. These planned paths will be sent to the control computer and the simulator will actually run through these planned routines. At this point, the testbed control software is no longer commanding the movement of the mock ARAPAIMA, the on-board computer has taken over, just as it would in space. The testbed is completely hardware in-the-loop; however, the testing eases into this full test as the hardware and software is developed. The testbed is adjustable to the

needs and progress of the design and development team so that it can be integrated at multiple levels of the process.

REFERENCES

Adolphus, C., & Jnr, E. (2003). Comprehensive List of CubeSat Missions.

Bahr, J. (2013). *University of Würzburg's picosatellite UWE-3 successfully launched and transmitting.* Retrieved 2015-05-24, from http://spaceboard.eu.temp-url.se/modx/news/spacerelated/2013/11/26/university-of-wurzburgs-picosatellite-uwe-3-successfully-launched-and-transmitting/

Boeckel, J. H. (1963, February). *The Purposes of Environmental Testing for Scientific Satellites* (Tech. Rep.). Retrieved from http://ntrs.nasa.gov/search.jsp?R=19630010328http://hdl.handle.net/2060/19630010328

Carew, M. (2014). *V-Slot Belt & Pinion Example Build.* Retrieved 2015-05-25, from http://www.openbuilds.com/builds/v-slot-belt-pinion-example-build.97/

Carew, M. (2015). *OpenBuilds OX CNC Machine.* Retrieved 2015-05-25, from http://openbuilds.org/builds/openbuilds-ox-cnc-machine.341/

Corpino, S. (2014). Verification of a CubeSat via Simulation. , *50*(4).

CubeSat Launch Initiative Selectees. (2013). Retrieved 2015-04-01, from https://www.nasa.gov/directorates/heo/home/CSLI_selections.html#.VZb7PXjZoqg

Debus, T., & Dougherty, S. (2009). Overview and Performance of the Front-End Robotics Enabling Near-Term Demonstration (FREND) Robotic Arm. In (p. 12). American Institute of Aeronautics and Astronautics. Retrieved from http://arc.aiaa.org/doi/abs/10.2514/6.2009-1870

Doebbler, J., Davis, J. J., Valasek, J., & Junkins, J. L. (2008). Mobile Robotic System for Ground Testing of Multi-Spacecraft Proximity Operations. In (Vol. 6548). Retrieved from http://arc.aiaa.org/doi/pdf/10.2514/6.2008-6548

DRV8834 Low-Voltage Stepper Motor Driver Carrier. (n.d.). Retrieved 2015-05-25, from https://www.pololu.com/product/2134

Dutta, T. (2012). Evaluation of the Kinect sensor for 3-D kinematic measurement in the workplace. *Applied ergonomics*, *43*(4), 645–649.

Harris, K., Mcgarvey, M., Chang, H. Y., Ryle, M., Ii, T. R., Udrea, B., & Nayak, M. (n.d.). SSC13-WK-6 Application for RSO Automated Proximity Analysis and IMAging (ARAPAIMA): Development of a Nanosat-based Space Situational Awareness Mission.

Heidt, H., Puig-Suari, J., Moore, A., Nakasuka, S., & Twiggs, R. (2000). CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation. Retrieved from http://digitalcommons.usu.edu/smallsat/2000/All2000/32/

Initiative, a. G. H. (2007). Fact sheet. (February). doi: 10.1016/S0002-8223(97)00353-2

Kelm, B. E., Angielski, J. A., Butcher, S. T., Creamer, N. G., Harris, K. A., Henshaw, C. G., ... Others (2008). *FREND: Pushing the Envelope of Space Robotics* (Tech. Rep.). DTIC Document. Retrieved from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA517473

Kerkhove, T. (2012). *What is the difference between Kinect for Windows & Kinect for Xbox360?* Retrieved 2015-06-27, from http://www.kinectingforwindows.com/2012/09/07/what-is-the-difference-between-kinect-for-windows-kinect-for-xbox360/

Ledin, J. A. (1999). Hardware-in-the-loop simulation. *Embedded Systems Programming*, *12*, 42–62.

New, J. C., & Timmins, a. R. (1966). Effectiveness of environment-simulation testing for satellites. , *D*(December), 1711–1715. doi: 10.2514/3.28735

Outman, V., Wang, E. S. J., & Company, M. (1966). Simulation Testing Space EnvironmentAn Assessment. , *D*(1697), 1697–1711.

Papadopoulos, L. (2014). NASA's Cube Quest Challenge Aims to Send Nanosatellites to Lunar Orbit and Beyond on First SLS Flight. *AmericaSpace*. Retrieved from http://www.americaspace.com/?p=72686

Parry, D. (2014). NRL Hosts NASA Administrator Charles Bolden. *NRL*. Retrieved from http://www.nrl.navy.mil/media/news-releases/2014/nrl-hosts-nasa-administrator-charles-bolden

Rems, F. (2012). Integration of the Formation Flying Testbed with the European Proximity Operations Simulator.. Retrieved from http://elib-v3.dlr.de/84071/1/Rems_Diplomarbeit_2011.pdf

Swartwout, M. (2013). The First One Hundred CubeSats : A Statistical Look. *Journal of Small Satellites*, *2*(2), 213–233.

The CubeSat Program. (2009). Cubesat design specification. *The CubeSat Program, California Polytechnic State ...*, *8651*, 22. Retrieved from http://www.cubesat.org/images/developers/cds_rev12.pdf$\delimiter"026E30F$nhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:CubeSat+Design+Specification#0

Toorian, A., Diaz, K., & Lee, S. (2008). The CubeSat approach to space access. *IEEE Aerospace Conference Proceedings*, *1*(1). doi: 10.1109/AERO.2008.4526293

Tsiotras, P. (2014, January). A 5dof Experimental Platform for Research in Spacecraft Proximity Operations. *AAS Guidance and Control Conference*, *14*(114), 14. Retrieved from http://soliton.ae.gatech.edu/labs/dcsl/papers/gncc14.pdf

T-Slotted Extrusion, 10S, 72 Lx1 In H. (n.d.). Retrieved 2015-05-25, from http://www.grainger.com/product/2RCP8?cm_sp=HIO-_-Home-_-VTV70300505&cm_vc=HPPVZ11&zoneId=HPBTZ11

Ure, N. K., Kaya, Y. B., & Inalhan, G. (2011). The development of a Software and Hardware-in-the-Loop Test System for ITU-PSAT II nano satellite ADCS. In *Aerospace conference, 2011 ieee* (pp. 1–15).

Vallado, D. A., & McClain, W. D. (2001). *Fundamentals of astrodynamics and applications* (Vol. 12). Springer Science & Business Media.

Vannatta, C. (2009). *Use of Electrodynamic Tethers to Counteract Drag Effects on the International Space Station.* Retrieved from http://ccar.colorado.edu/asen5050/projects/projects_2009/vannatta/

V-Slot Linear Rail. (2015). Retrieved 2015-05-25, from http://openbuildspartstore.com/v-slot-linear-rail/

Wilson, W. R., Jones, L. L., & Peck, M. a. (2013). A Multimodule Planar Air Bearing Testbed for CubeSat-Scale Spacecraft. *Journal of Dynamic Systems, Measurement, and Control*, *135*(4), 045001. Retrieved from http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?doi=10.1115/1.4023767 doi: 10.1115/1.4023767

A. Apparatus Cost

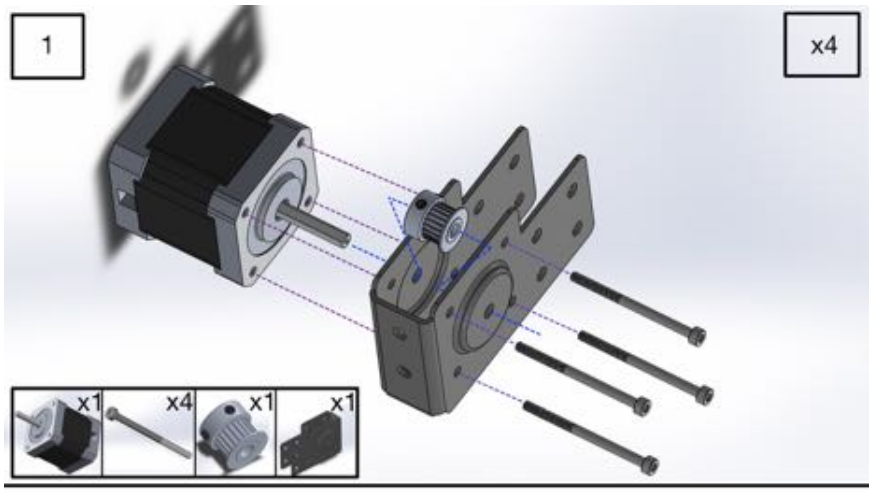
This section details the part and cost breakdown of Chronos. In addition, the vendor for the part is also listed.

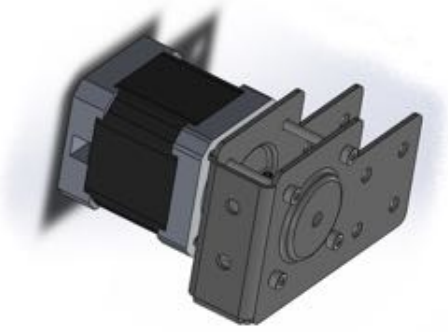
Part Name	**Quantity**	**Source**	**Price Per [$]**
V-Slot Linear Rail (20x20x1000mm)	8	OpenBuilds	10.00
V-Slot Linear Rail (20x20x500mm)	6	OpenBuilds	5.00
V-Slot Linear Rail (20x40x1000mm)	1	OpenBuilds	13.00
Smooth Idler Pulley Wheel Kit	4	OpenBuilds	4.85
Delrin Mini V Wheel Kit	16	OpenBuilds	3.25
Delrin V Wheel Kit	12	OpenBuilds	3.85
Belt Clamp Crimp Style	8	OpenBuilds	0.60
GT2 (2mm) Aluminum Timing Pulley - 20 Tooth	5	OpenBuilds	5.50
Linear Actuator End Mount	8	OpenBuilds	9.95
Mounting Hub (5mm)	2	OpenBuilds	3.50
90 Degree Joining Plate	12	OpenBuilds	4.80
V-Slot Gantry Plates	3	OpenBuilds	12.00
Mini V Wheel Plate	4	OpenBuilds	8.95
Motor Mount Plate for Nema 17 Stepper Motor	3	OpenBuilds	6.95
Micro Limit Switch with Mounting Plate	10	OpenBuilds	4.50
12V/30A Power Supply	2	OpenBuilds	35.00
Eccentric Spacer (6mm)	14	OpenBuilds	2.00
Double Tee Nut	33	OpenBuilds	0.85
Inside Hidden Corner Bracket	12	OpenBuilds	1.80
Slot Cover/Panel Holder (1000mm)	7	OpenBuilds	3.00
Slot Cover/Panel Holder (500mm)	4	OpenBuilds	2.00
M3 Cap Head Screws (45mm)	24	OpenBuilds	0.25
Aluminum Spacers (6mm)	28	OpenBuilds	0.20
Low Profile Screws M5 (6mm)	20	OpenBuilds	4.00
Low Profile Screws M5 (8mm)	60	OpenBuilds	4.50
Low Profile Screws M5 (10mm)	2	OpenBuilds	4.50

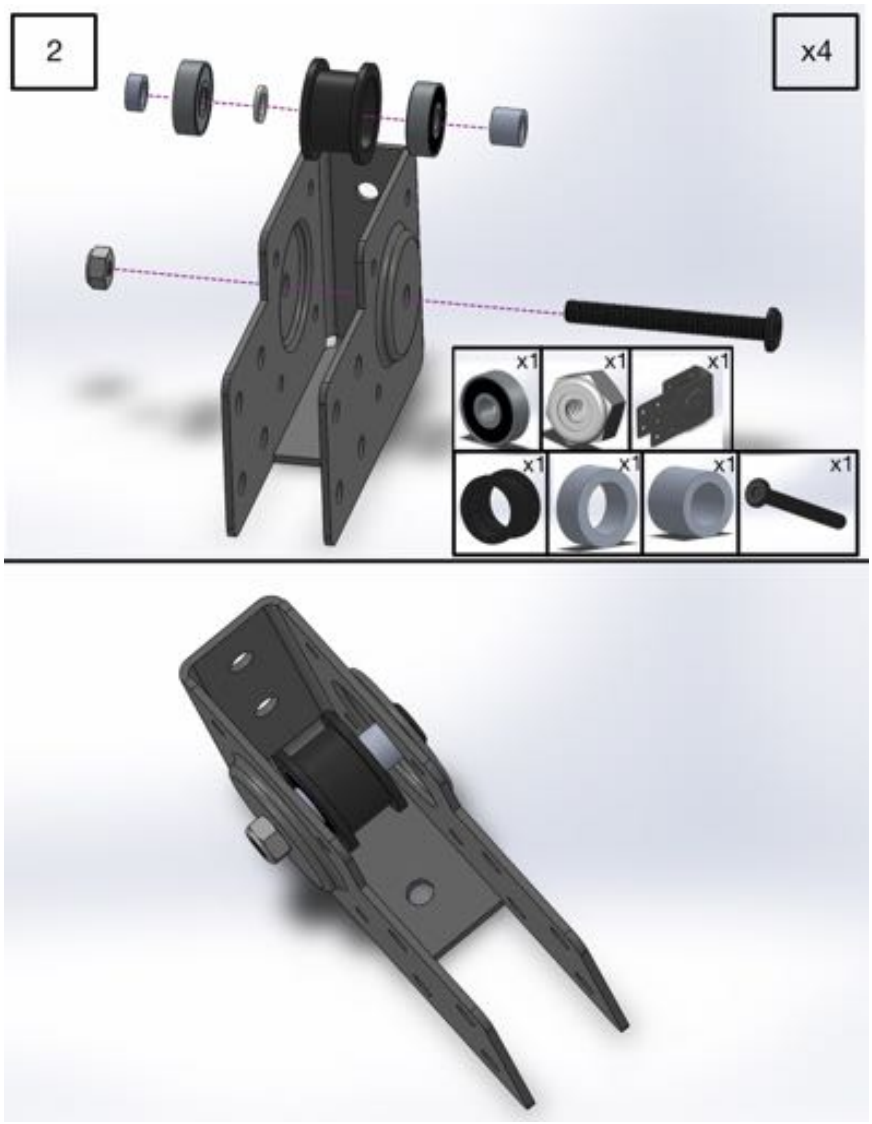
Part Name	**Quantity**	**Source**	**Price Per [$]**
DRV8834 Low-Voltage Stepper Motor Driver Carrier	4	Pololu	5.95
Stranded Wire: Black, 22 AWG, 50 Feet	2	Pololu	5.00
Stranded Wire: Red, 22 AWG, 50 Feet	2	Pololu	5.00
Stranded Wire: Green, 22 AWG, 50 Feet	1	Pololu	5.00
Stranded Wire: Blue, 22 AWG, 50 Feet	1	Pololu	5.00
Stranded Wire: White, 22 AWG, 50 Feet	1	Pololu	5.00
Amazon Basics USB 2.9 Extension Cable (3 meters)	2	Amazon	5.79
Amazon Basics USB 2.9 Extension Cable (3 meters)	1	Amazon	4.99
Monoprice 107028 30mmx1.5m Spiral Wrap Bands	2	Amazon	7.70
CNC Machine R18 10x10mm 100cm Cable Carrier Drag Chain	2	Amazon	12.13
Generic 7-Port USB Hub with ON/OFF Switch	1	Amazon	4.99
iXCC 5Pack 3ft Premium USB 2.0 - Micro USB to USB Cable	1	Amazon	7.99
Breadboard - Full-Size (Bare)	2	Sparkfun	5.95
Arduino Micro	3	Arduino	19.04
Total:			**1299.03**

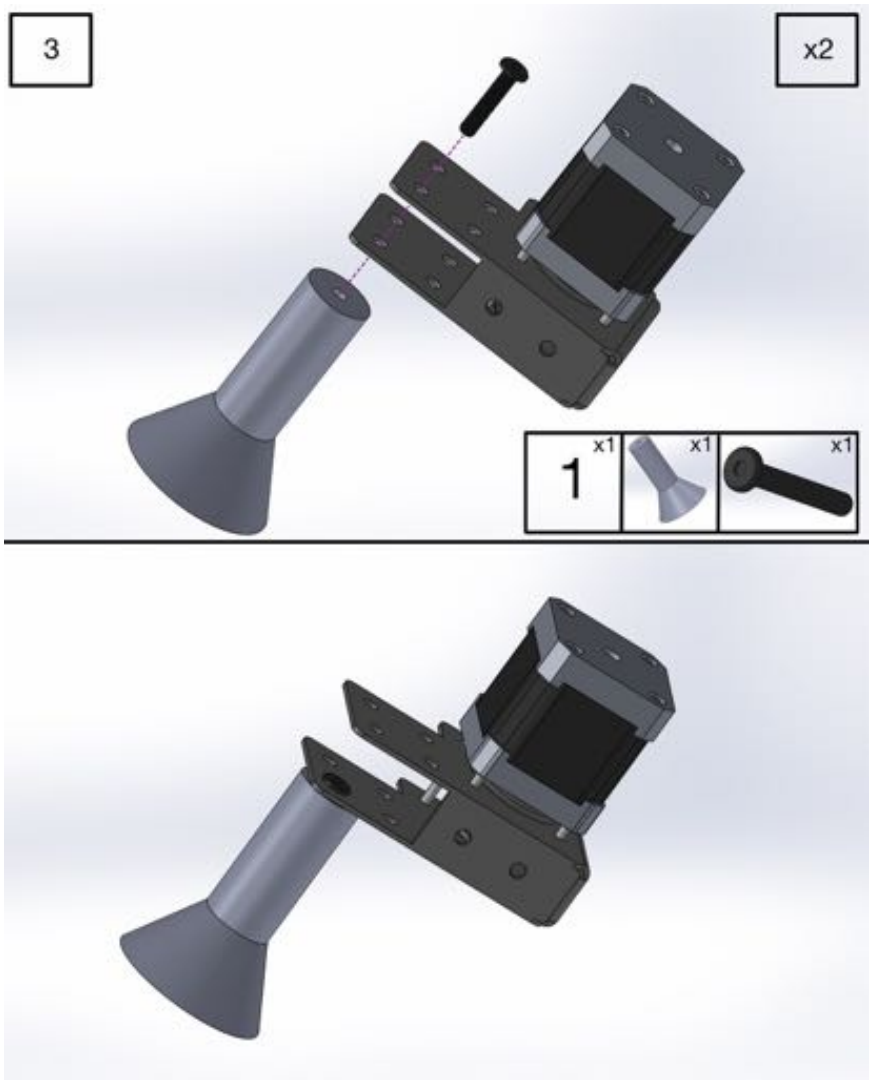
B. Assembly Instructions

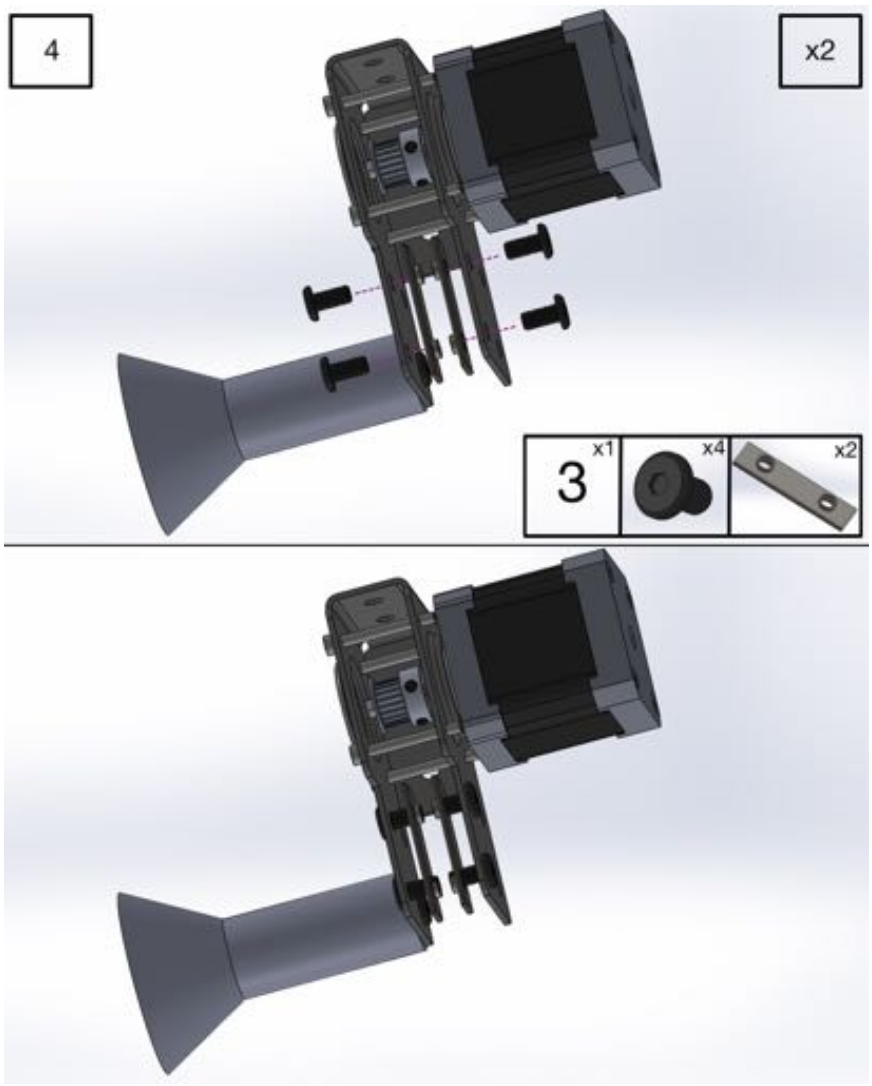
The follwing images serve as a guideline to the assembly of Chronos. The images go step-by-step through the assemblys process using the CAD model.

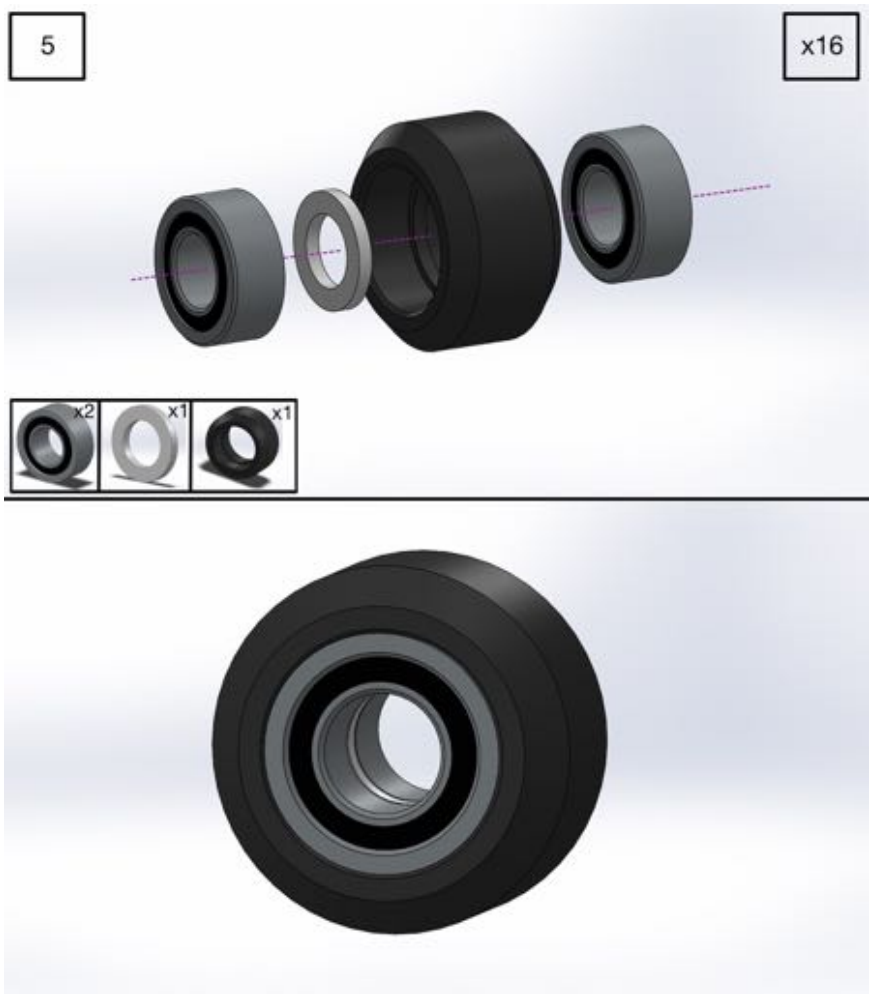


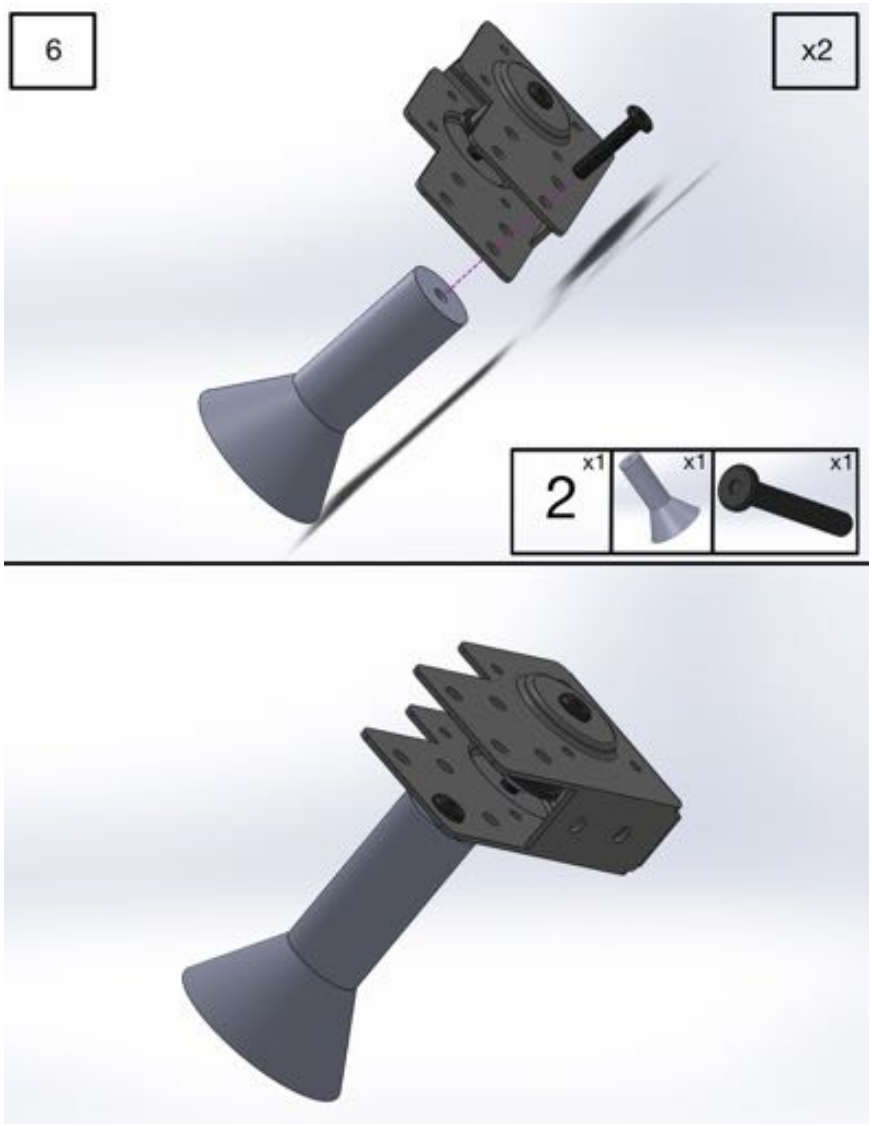


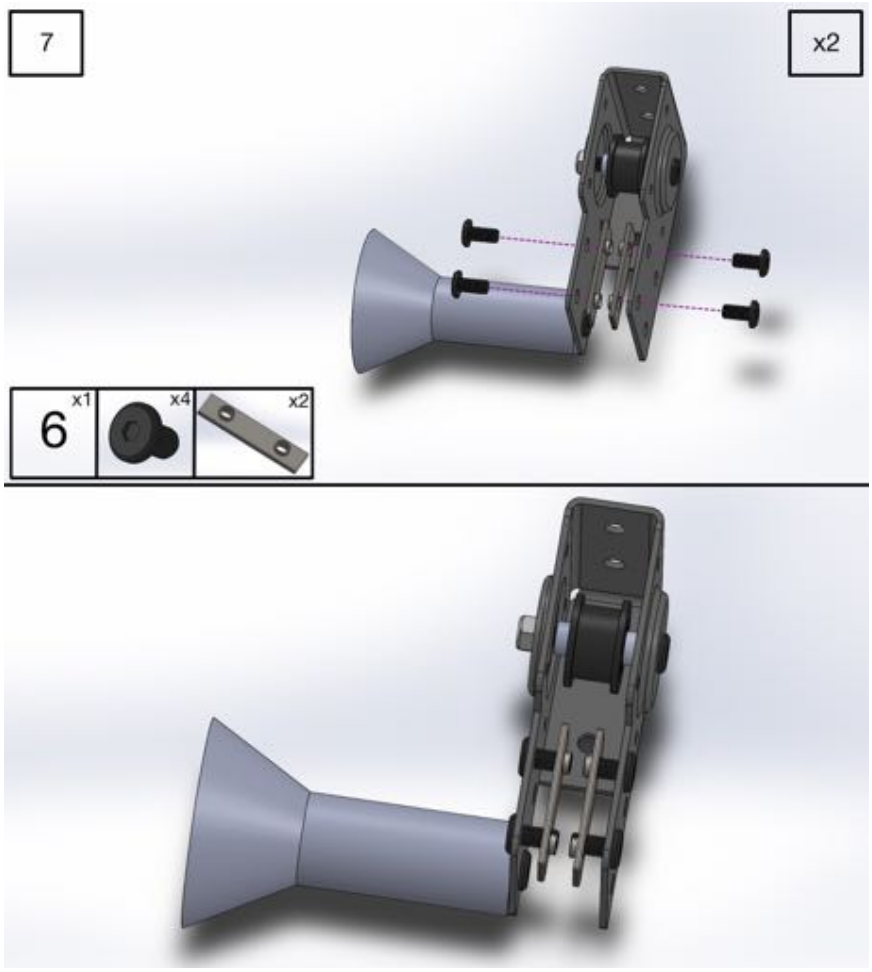

2
x4
x1
x1
x1
x1
x1
x1
x1


3
x2
1
x1
x1
x1


4
x2
3
x1
x4
x2


5
x16
x2
x1
x1

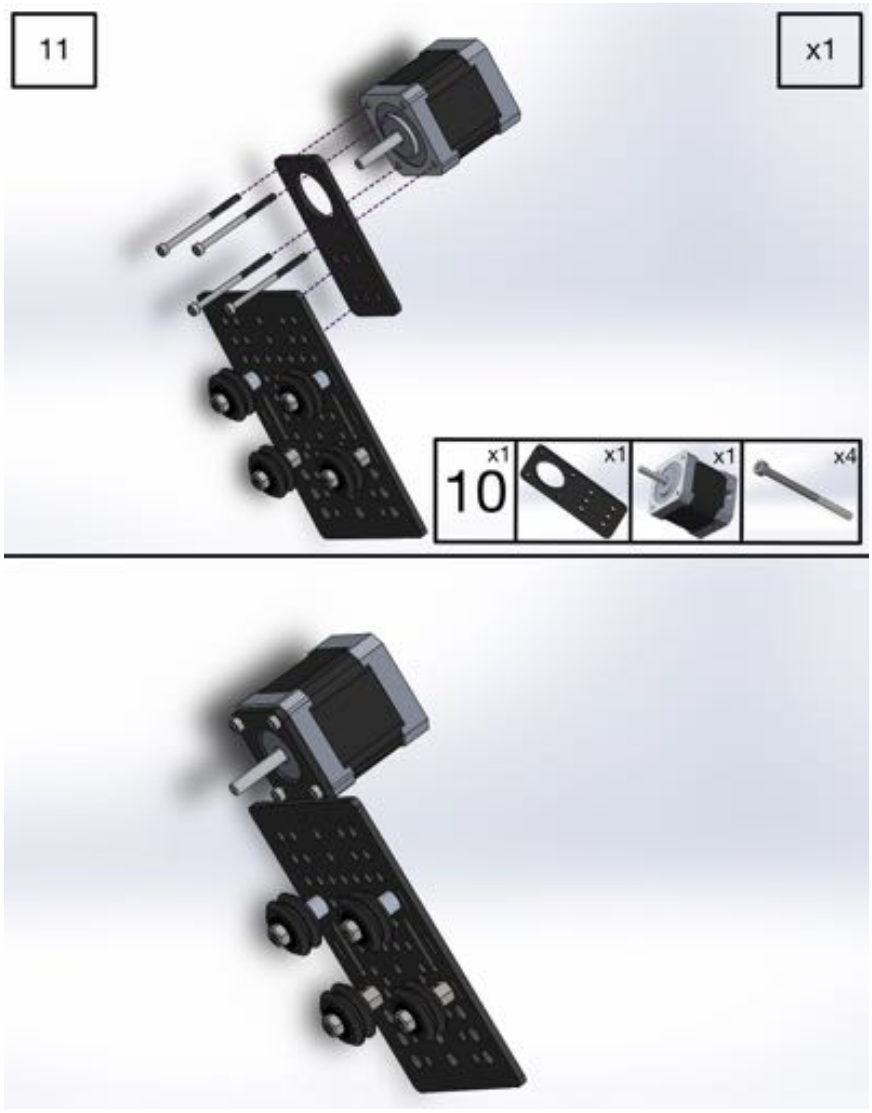

6
x2
2
x1
x1
x1


7
x2
6
x1
x4
x2


8
x12
x1
x1
x2

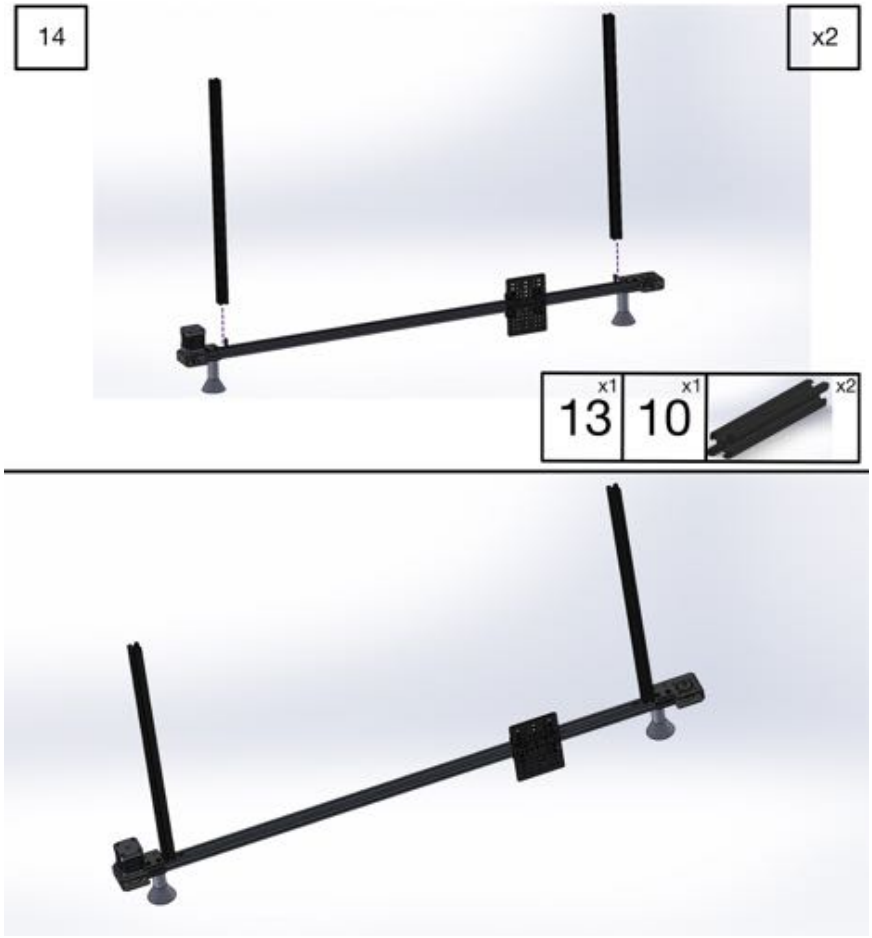

9
x4
5
x4
x4
x2
x2
x4
x1
x4

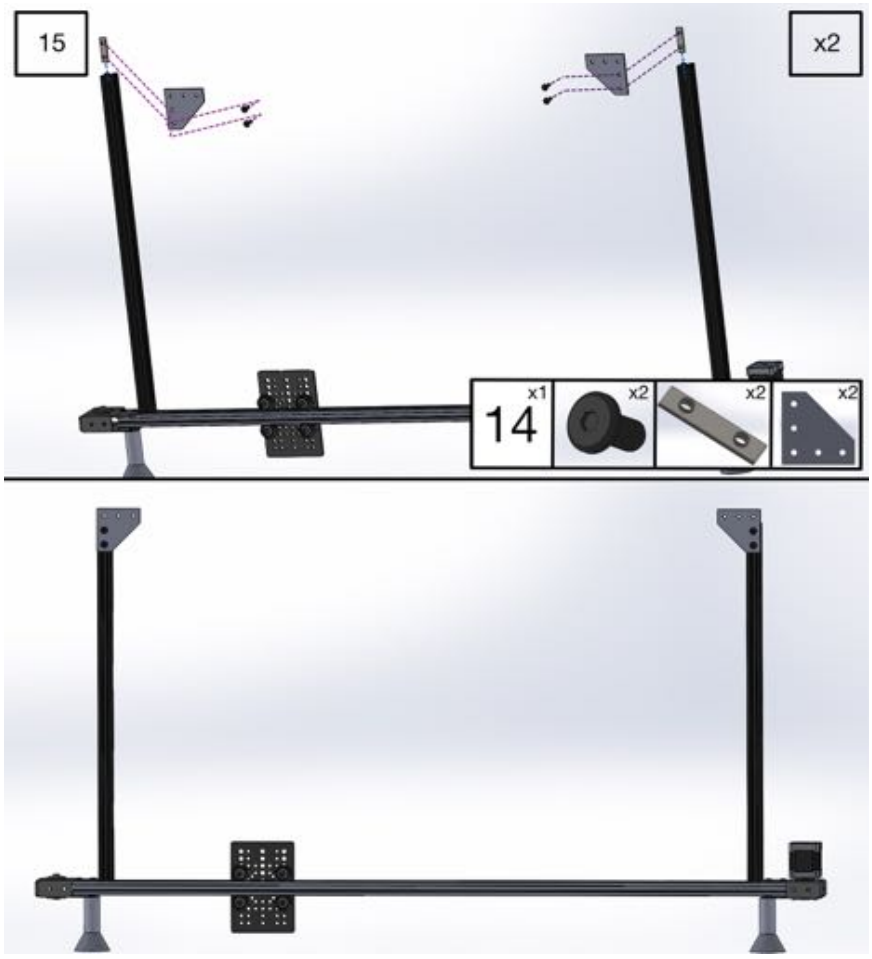

10
x3
x4
8
x4
x2
x2
x4
x1
x4


11
x1
10
x1
x1
x1
x4


12
x1
1 x1
2 x1
x1


13
x2
4
x1
7
x1
x1
x2


14
x2
x1
13
x1
10
x2


15
x2
x1
14
x2
x2
x2


16
x2
15
x1
9
x1
x2
x4
x1


17
x2
12
x1
16
x1
x4
x1


18
x2
x1
10
x1
17
x1
x2
x4

C. Simulink Code

This document contains the Simlink models from the Testbed Control Software discused in Chapter 3. These models demonstrate the modularity and simplicity of the control software for the testbed. The models serve as representations of the software as a whole, as every model and submodel is not present here.

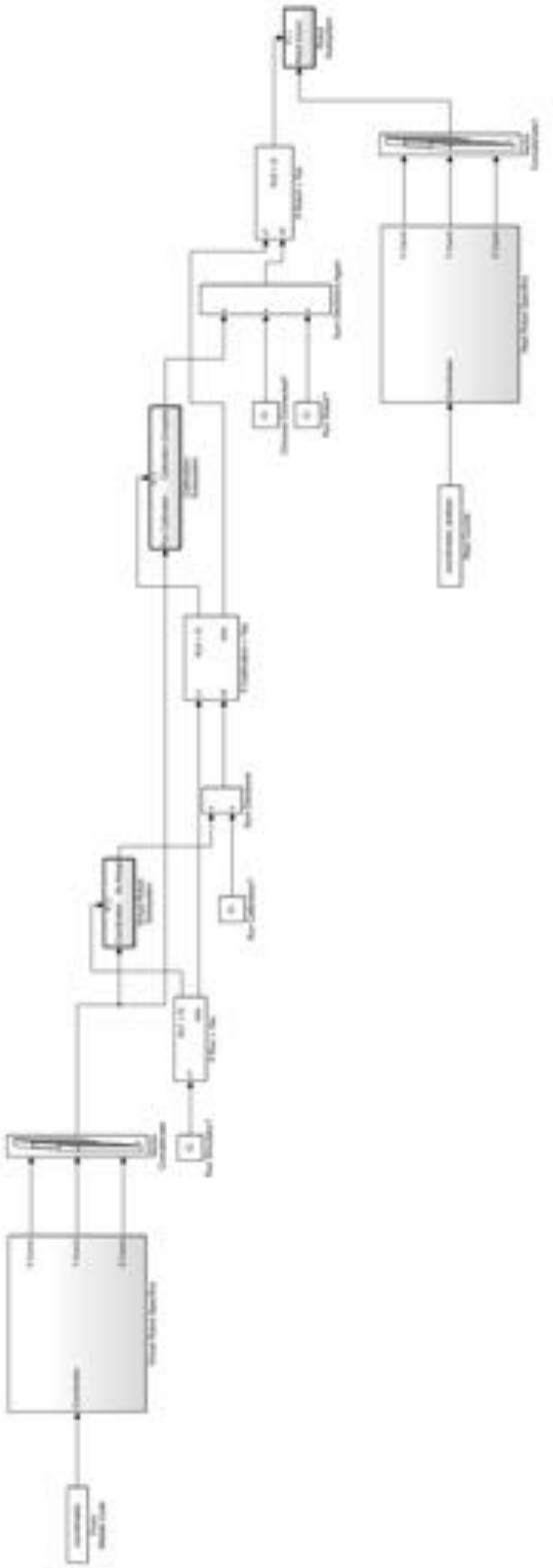


Figure C.1. Simulink block diagram showing the connections between each of the functions of the control software

Figure C.2. Simulink block diagram showing the two scaling processes for the virtual simulator

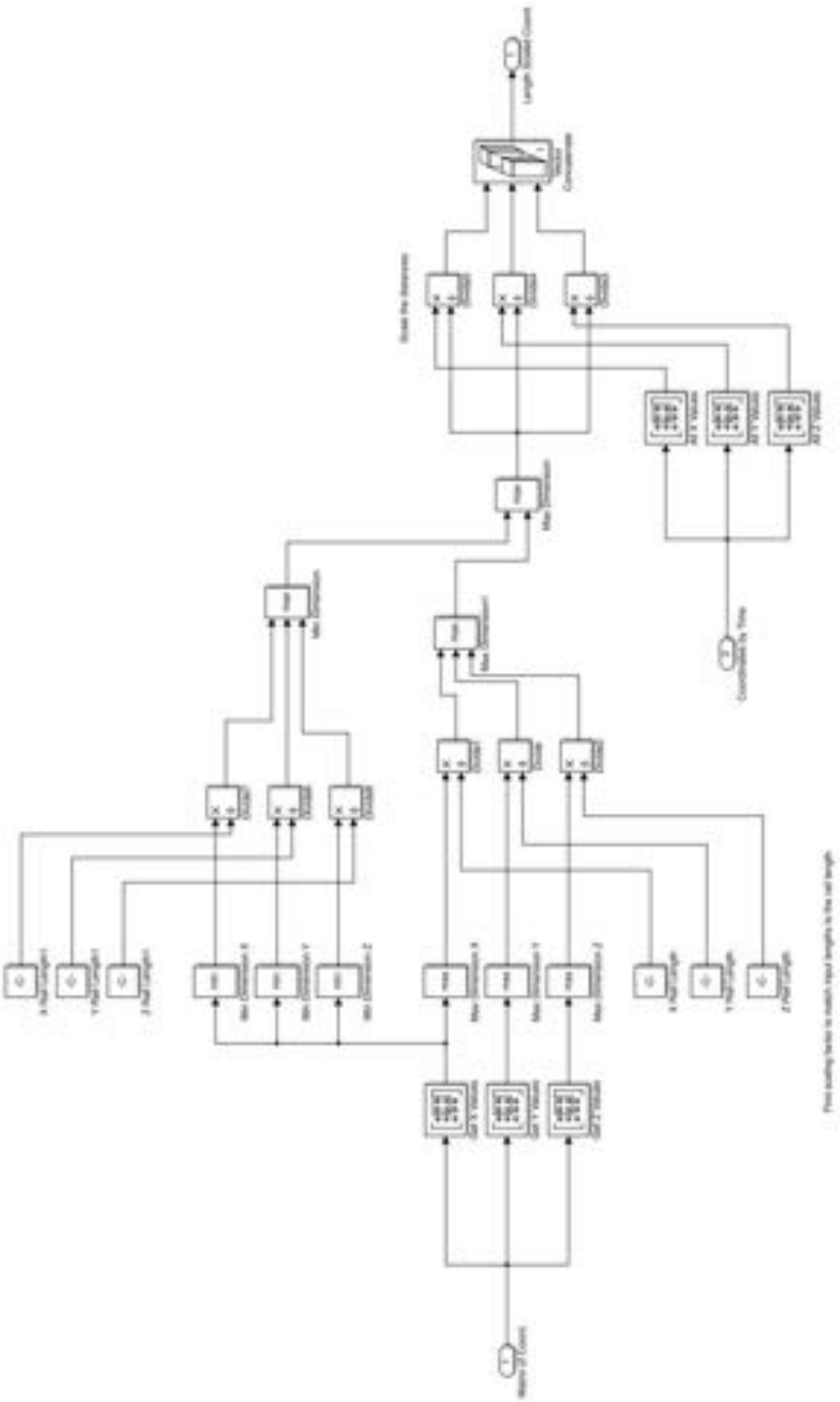


Figure C.3. Simulink block diagram scaling the position coordinates based on the available rail length

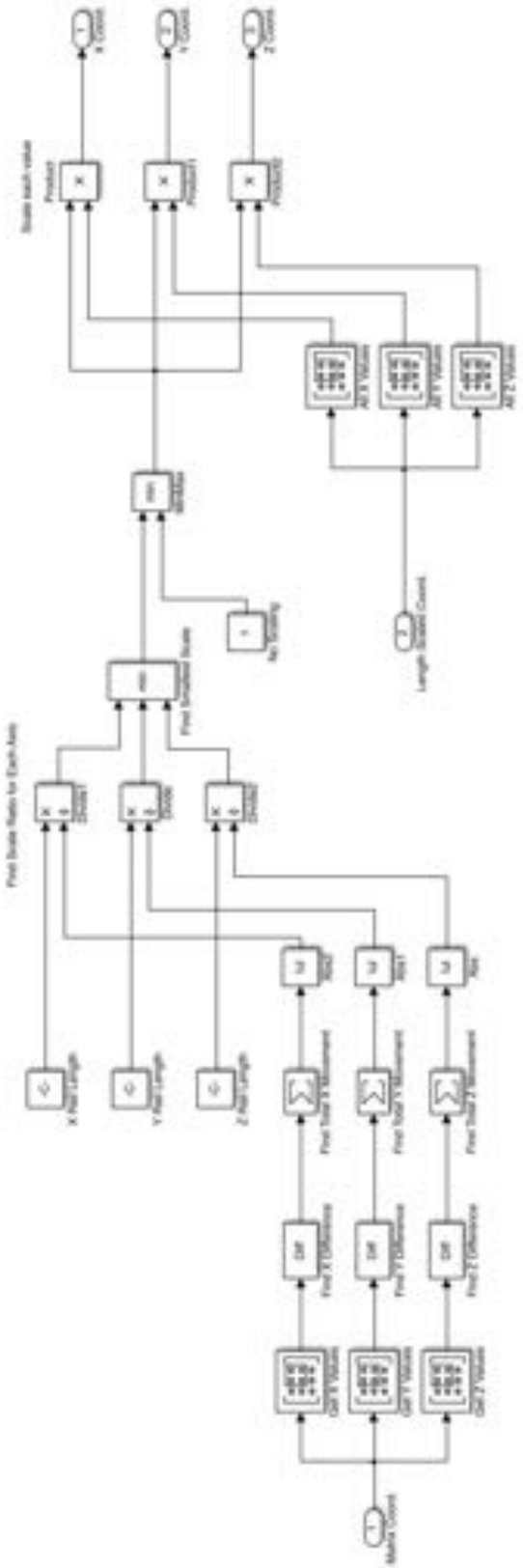


Figure C.4. Simulink block diagram scaling the position coordinates based on the overall delta movement

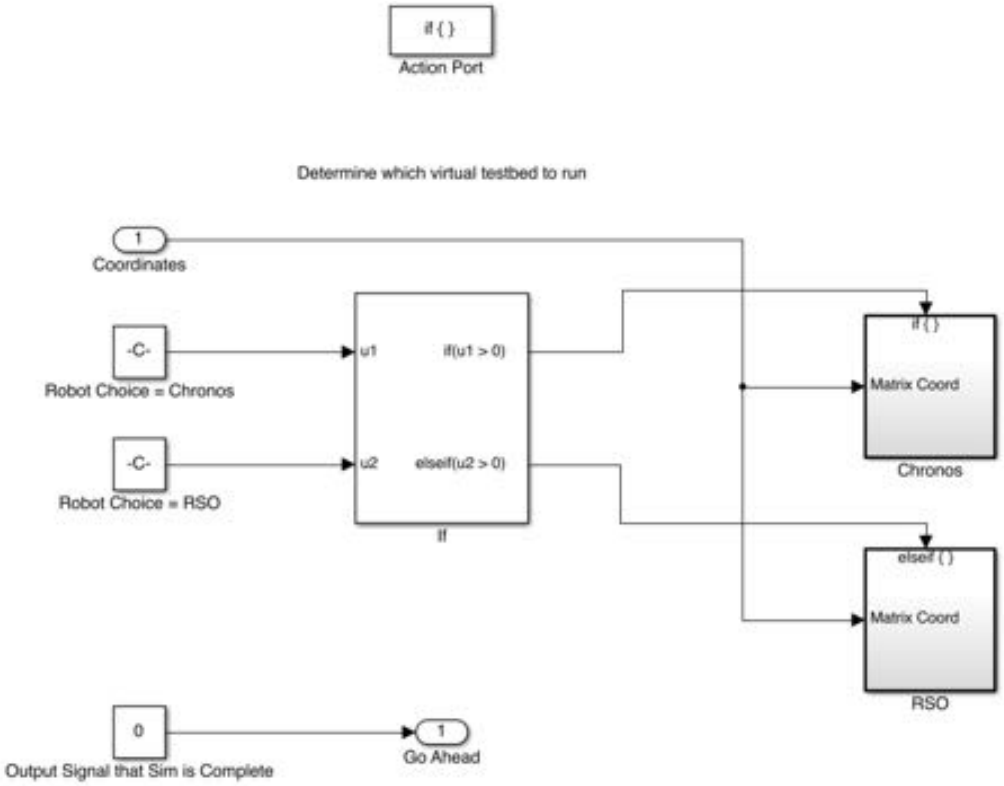


Figure C.5. Simulink block diagram allowing for the SimMechanics model to be variable

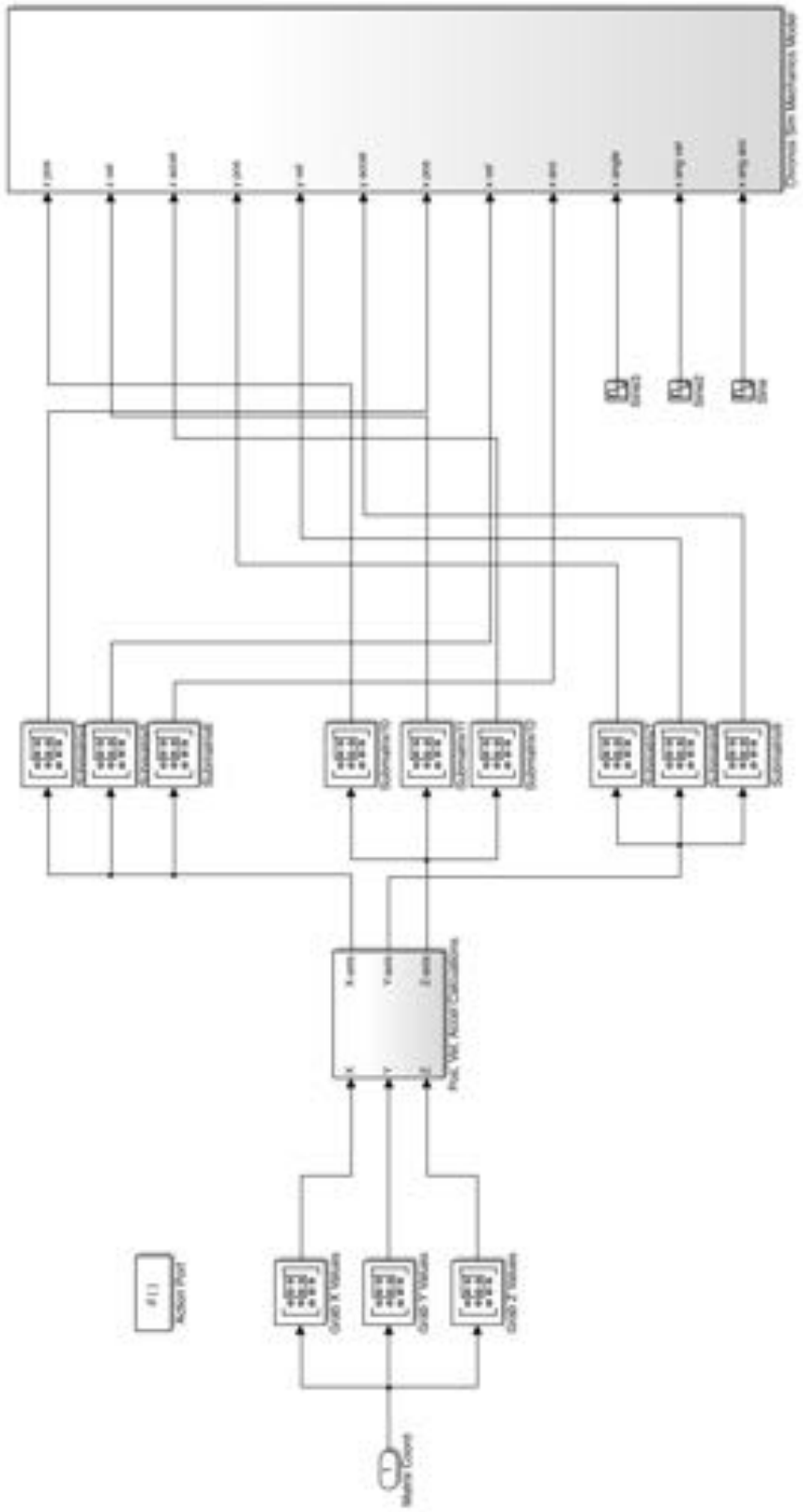


Figure C.6. Simulink block diagram showing scaled coordinated being sent to the SimMechanics model

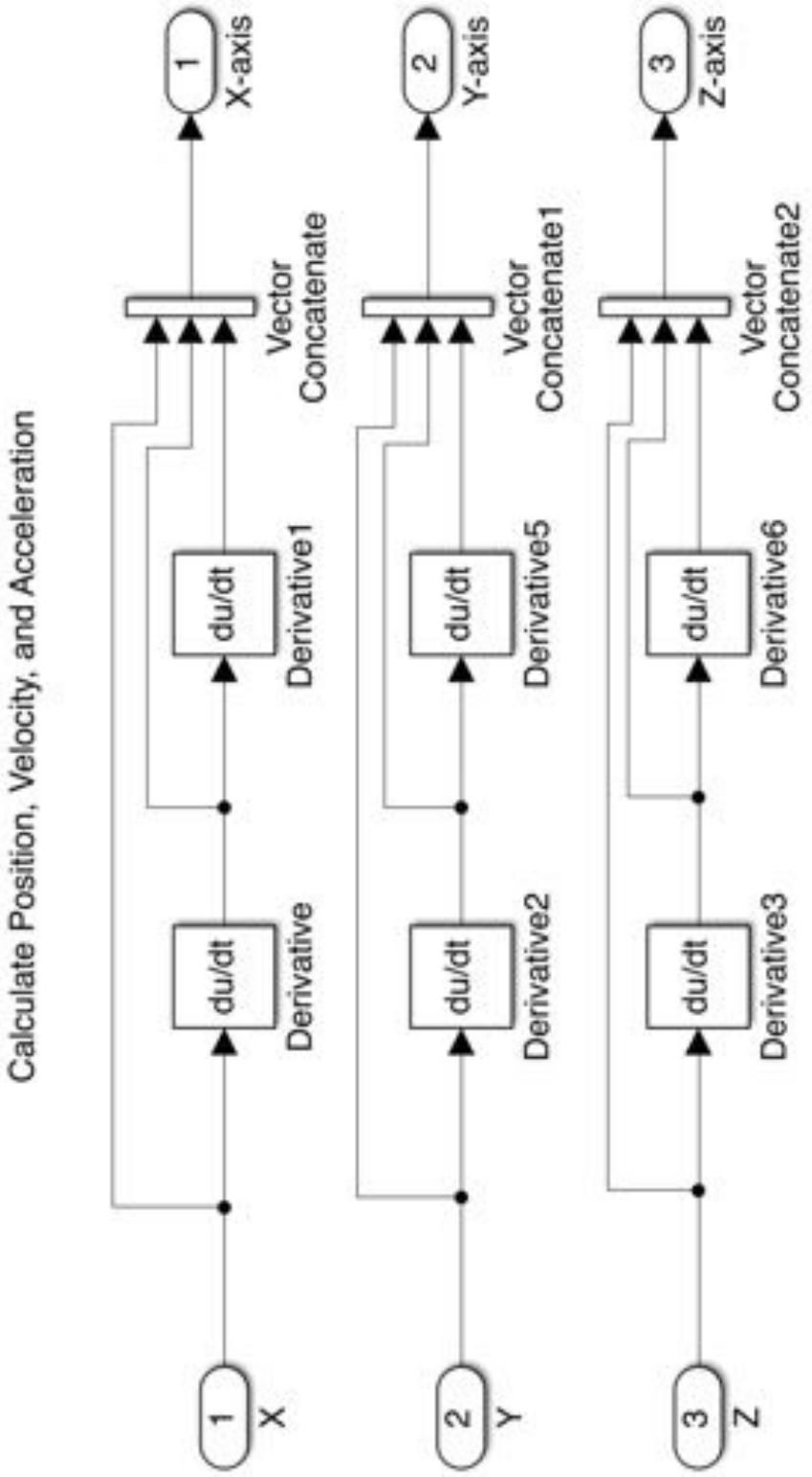


Figure C.7. Simulink block diagram calculating position, velocity, and acceleration to control the SimMechanics model

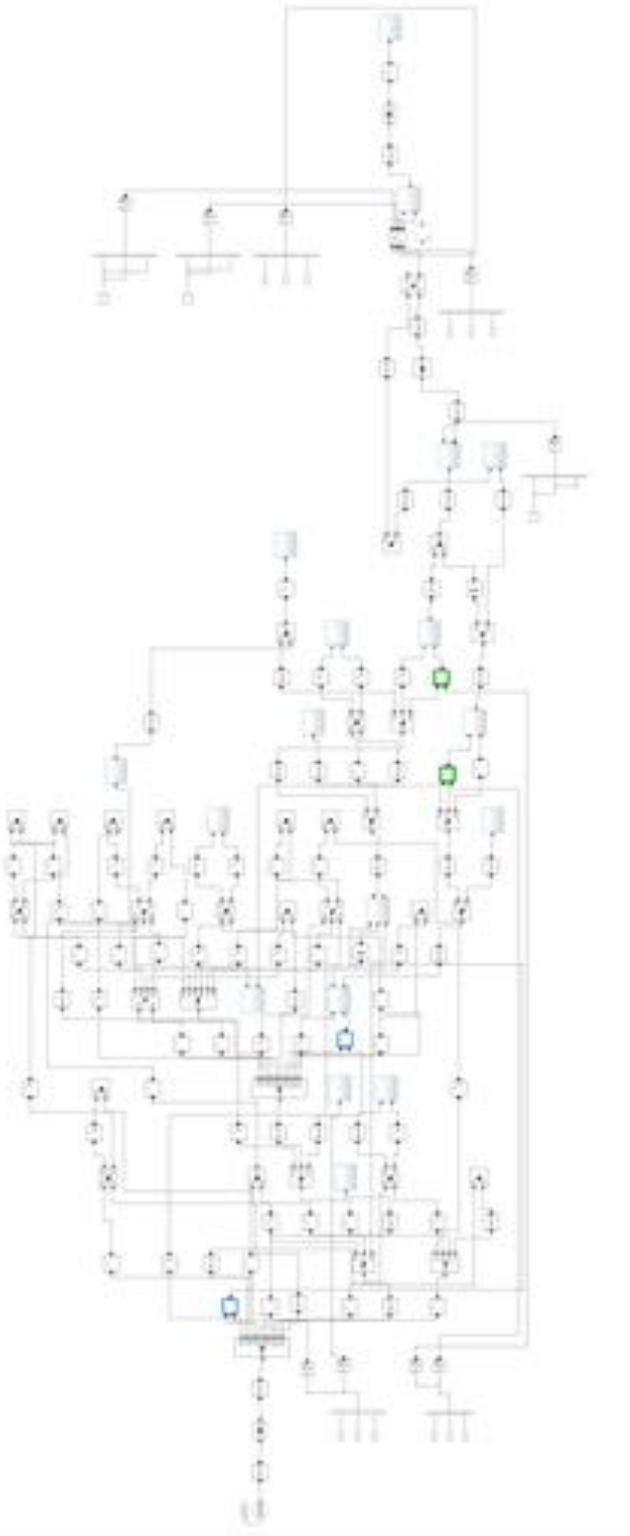


Figure C.8. The SimMechanics model of the Chronos testing apparatus

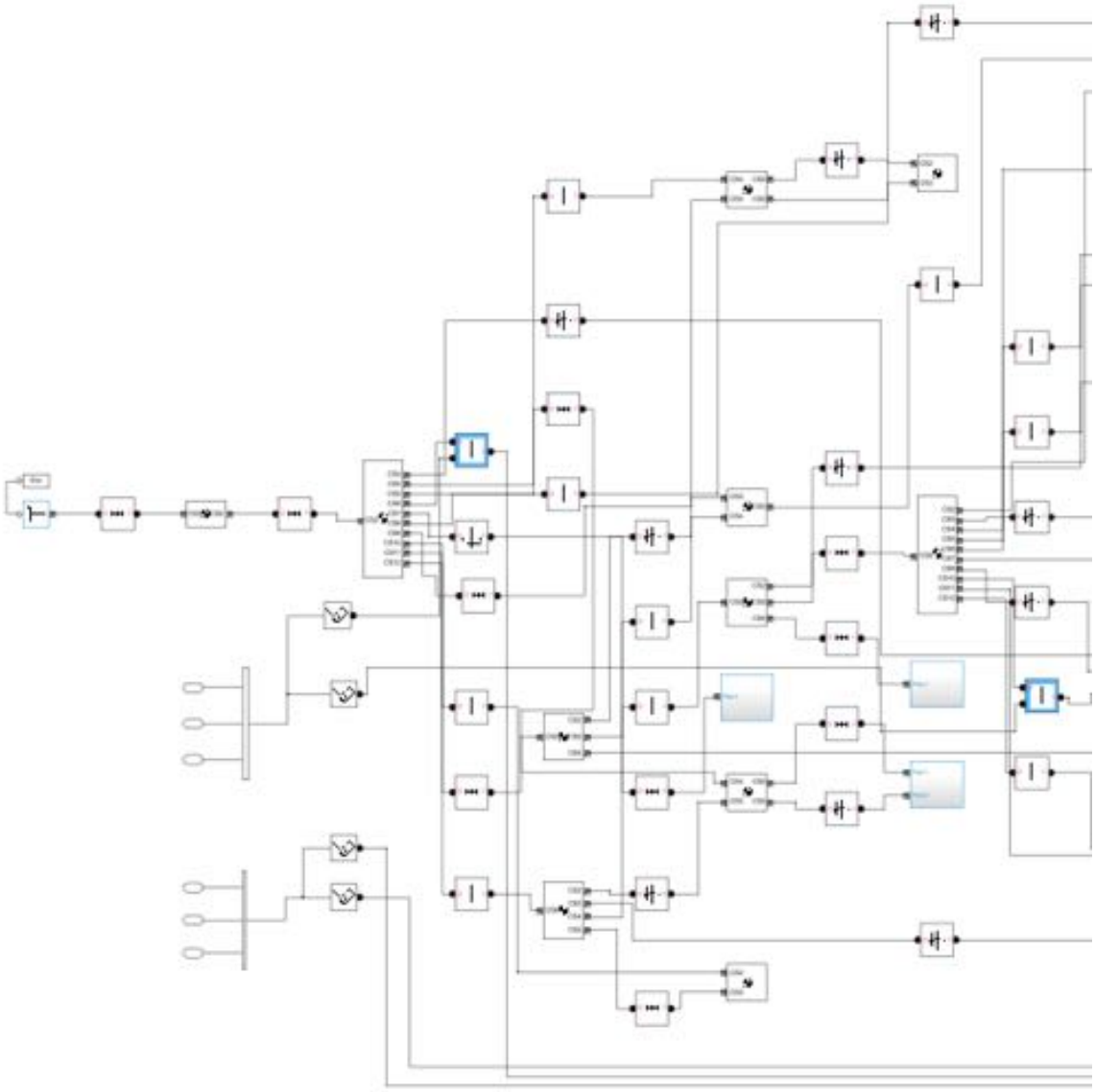


Figure C.9. The first section of the broken down SimMechanics model of the Chronos testing apparatus

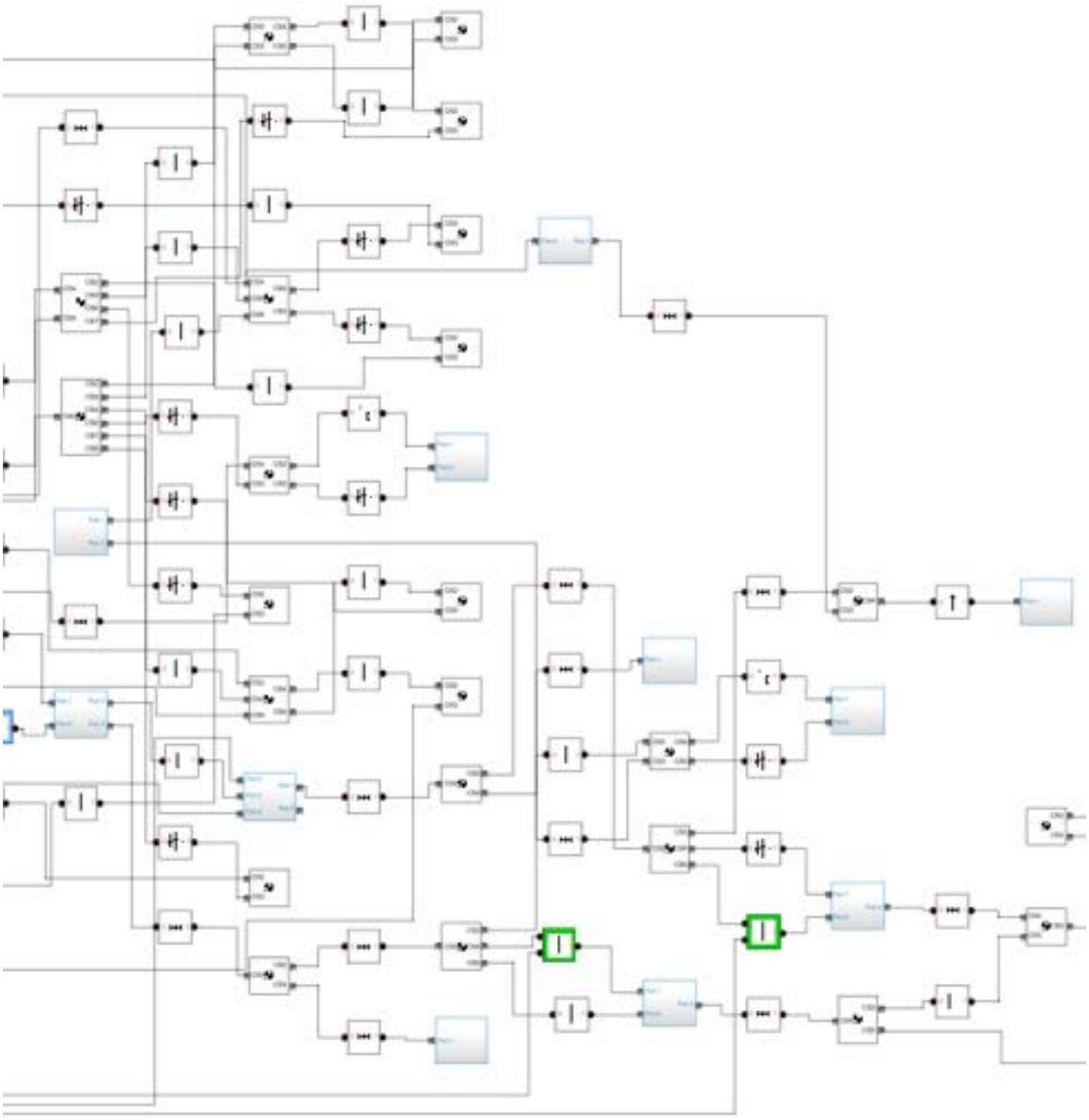


Figure C.10. The second section of the broken down SimMechanics model of the Chronos testing apparatus

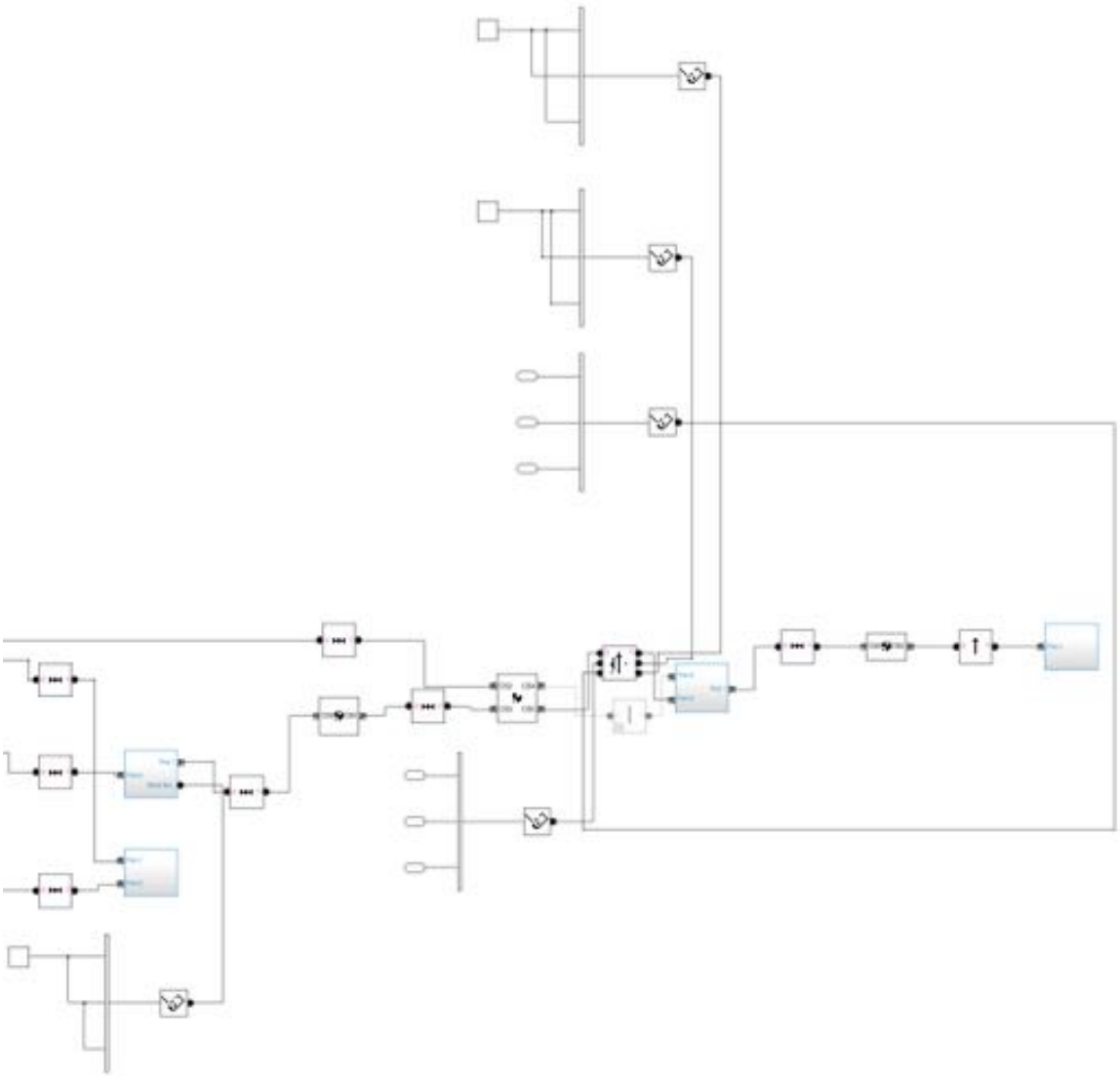


Figure C.11. The third section of the broken down SimMechanics model of the Chronos testing apparatus

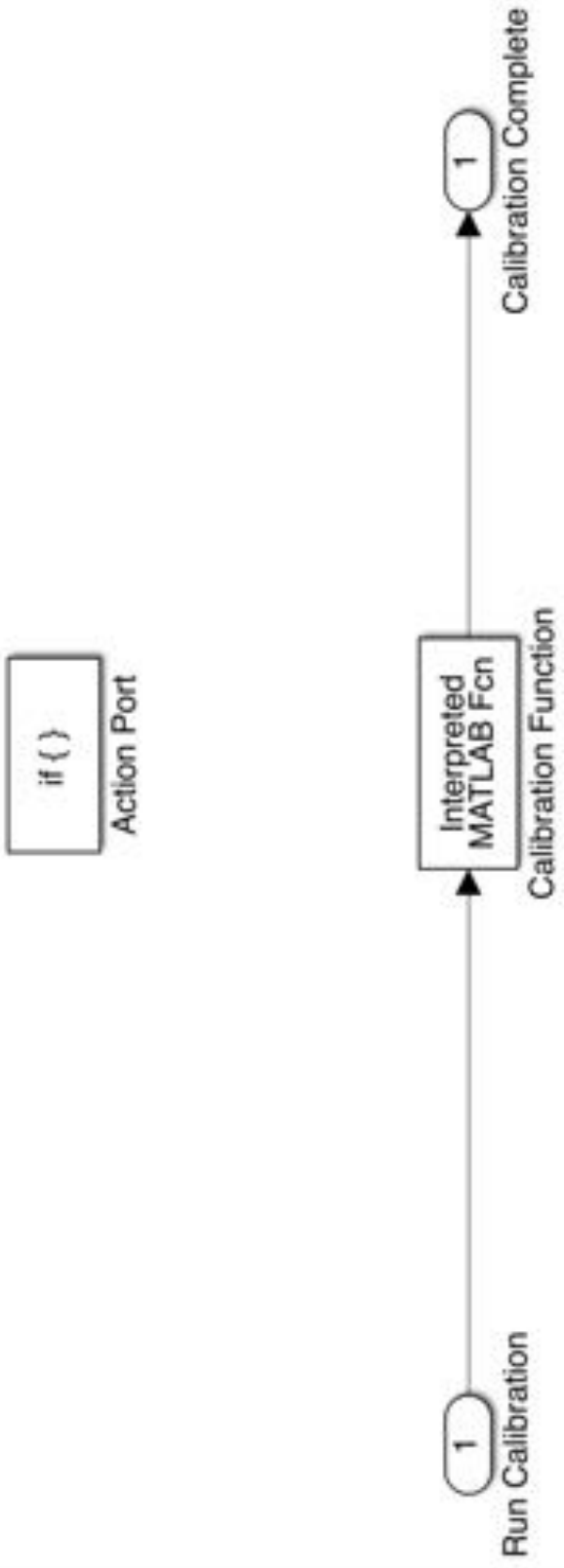


Figure C.12. Simulink block diagram of the matlab function for calibrating the testbed

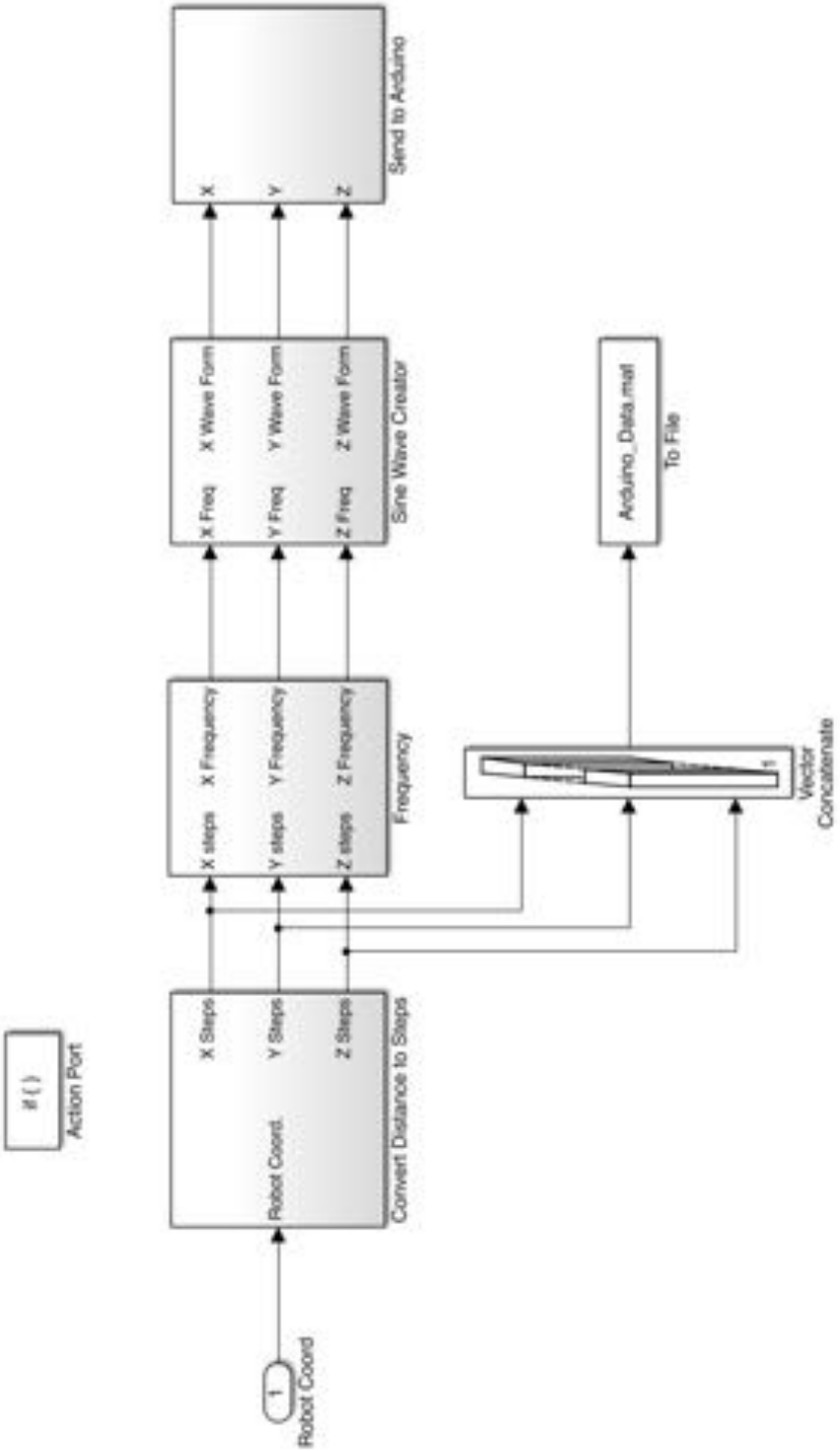


Figure C.13. Simulink block diagram that makes up the calculations for the testing apparatus

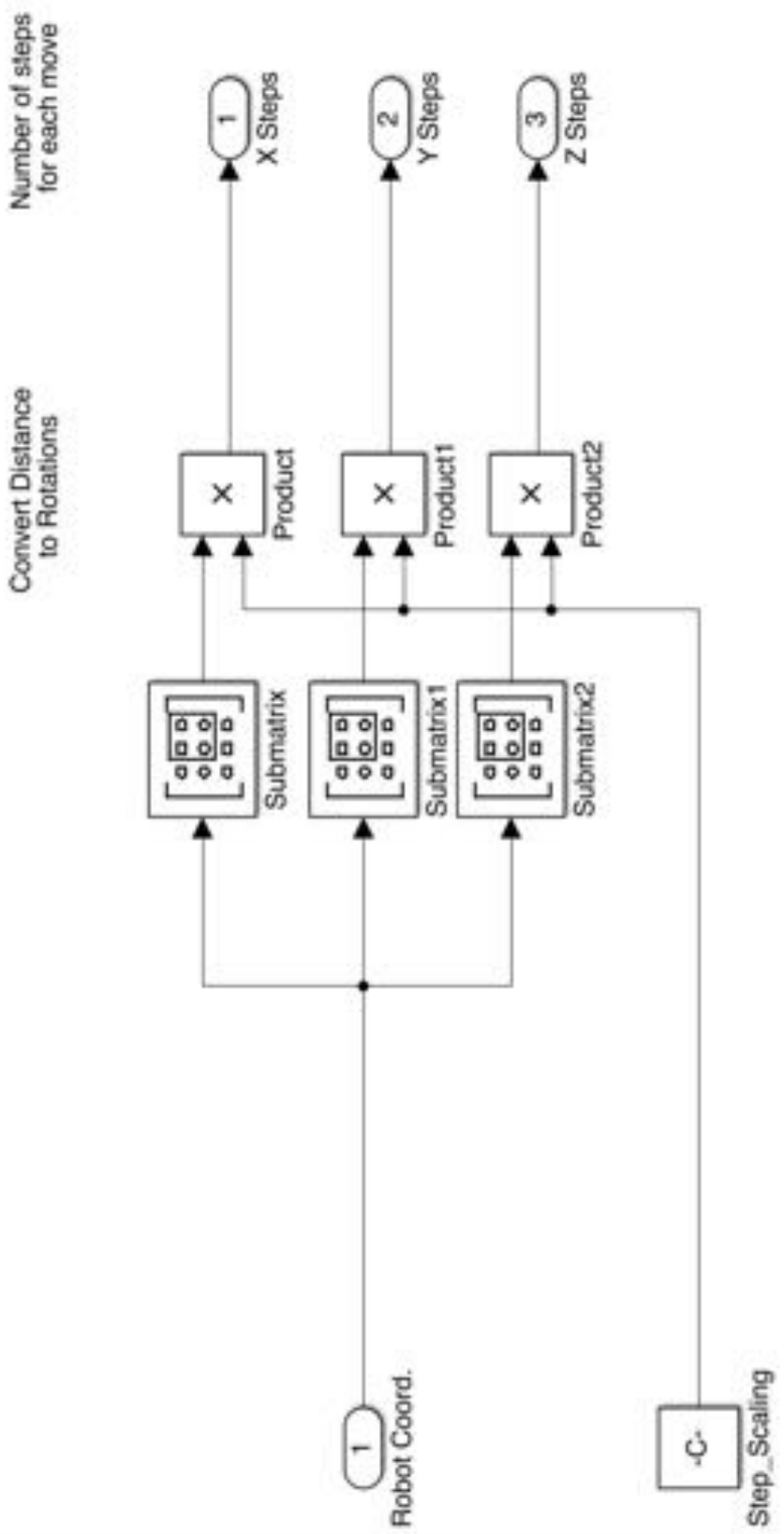


Figure C.14. Simulink block diagram converting the positions to steps on the motors

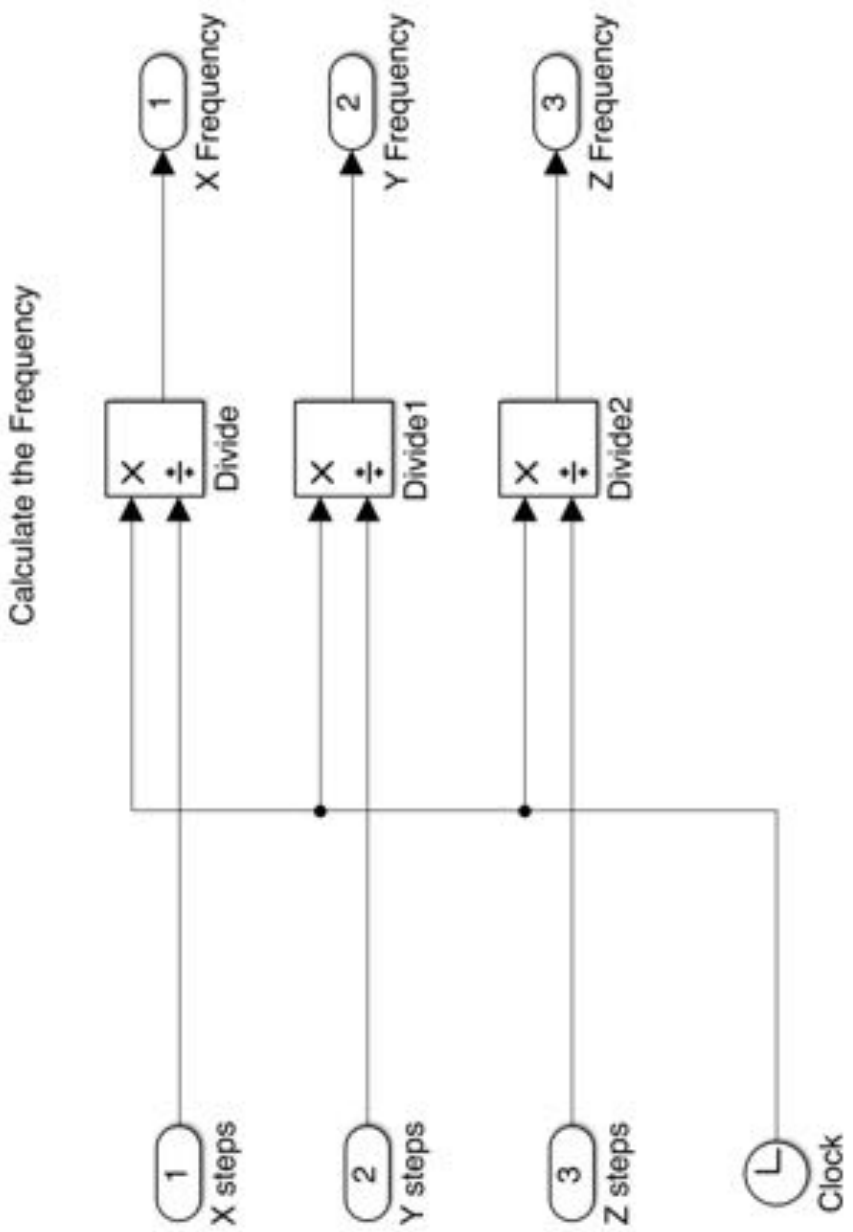


Figure C.15. Simulink block diagram calculating frequency for the sine wave design

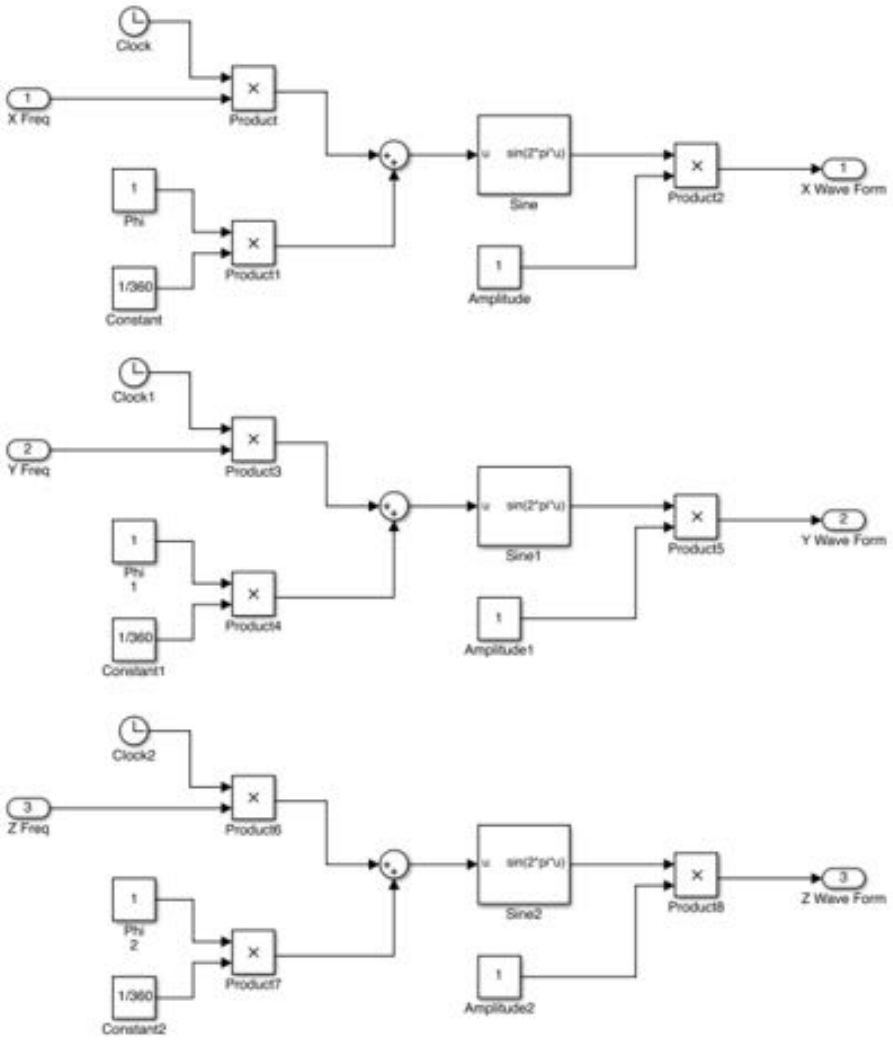


Figure C.16. Simulink block diagram designing a sine wave for each movement

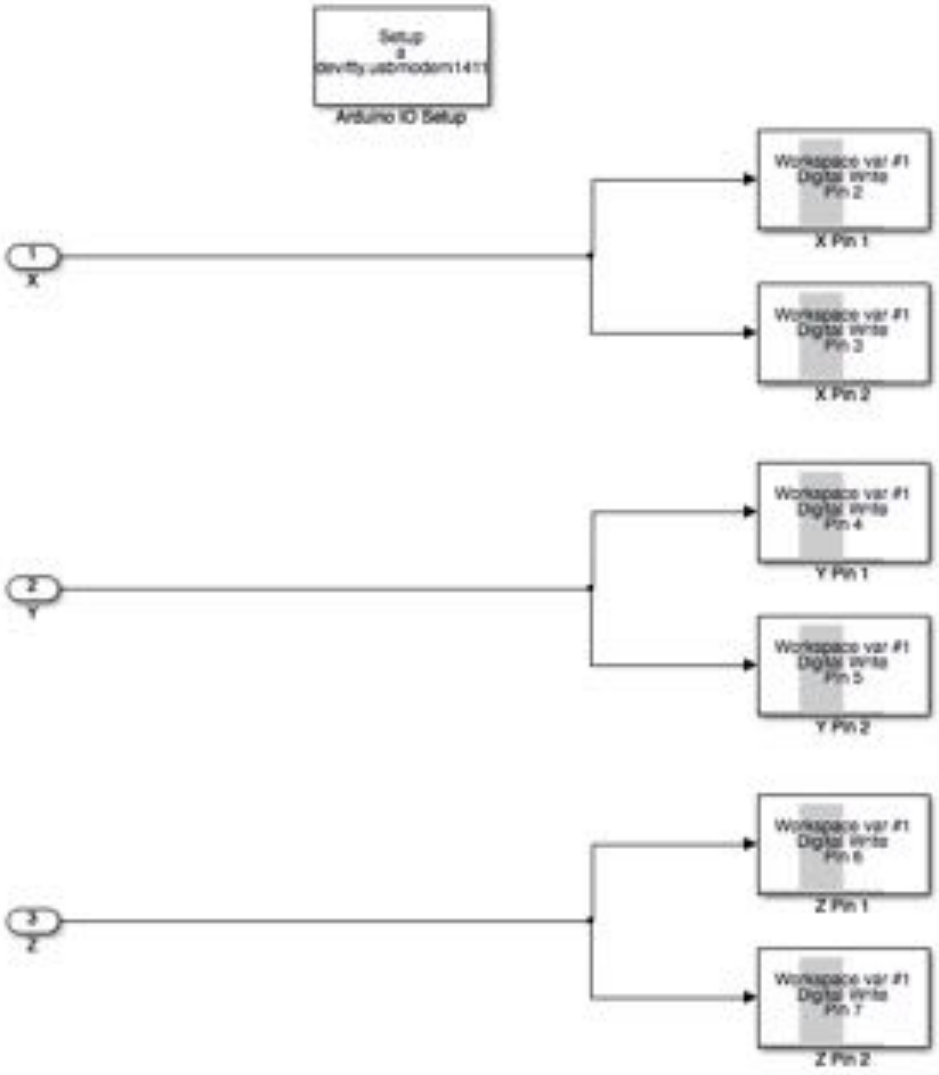


Figure C.17. Simulink block diagram sending the calculated sine wave to the Arduino pins

D. Accuracy Results

The following tables show the data from the accuracy testing of Chronos. The data displayed here is the average value for each measurement point over a series of 10 repeated tests.

Table D.1. The precision data for the Z-axis moving in the positive Z direction

Forward				
Total Distance Spanned [cm]	Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
4	4	4.9	3.9	2.5%
8	4	4	4	0.0%
12	4	4	4	0.0%
16	4	4	4	0.0%
20	4	3.9	3.9	2.5%
24	4	4	4	0.0%
28	4	4	4	0.0%
32	4	4	4	0.0%
36	4	4	4	0.0%
40	4	4	4	0.0%
44	4	3.9	3.9	2.5%
48	4	4	4	0.0%
52	4	4	4	0.0%
56	4	3.9	3.9	2.5%
60	4	4	4	0.0%
64	4	4.1	4.1	-2.5%
68	4	3.9	3.9	2.5%
72	4	4	4	0.0%
			Average:	0.6%

Table D.2. The precision data for the Z-axis moving in the negative Z direction

Reverse				
Total Distance Spanned [cm]	Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
4	4	4.9	3.9	2.5%
8	4	4	4	0.0%
12	4	3.9	3.9	2.5%
16	4	4	4	0.0%
20	4	4	4	0.0%
24	4	4	4	0.0%
28	4	4	4	0.0%
32	4	4	4	0.0%
36	4	4	4	0.0%
40	4	4	4	0.0%
44	4	4	4	0.0%
48	4	4	4	0.0%
52	4	4	4	0.0%
56	4	4	4	0.0%
60	4	3.9	3.9	2.5%
64	4	4	4	0.0%
68	4	3.9	3.9	2.5%
72	4	4.1	4.1	-2.5%
			Average:	0.4%

Table D.3. The precision data for the Y-axis moving in the upward Y direction

Up				
Total Distance Spanned [cm]	Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
2	2	9.5	1.9	5.0%
4	2	2	2	0.0%
6	2	2	2	0.0%
8	2	1.9	1.9	5.0%
10	2	2	2	0.0%
12	2	2.1	2.1	-5.0%
14	2	1.9	1.9	5.0%
16	2	2	2	0.0%
18	2	2.1	2.1	-5.0%
20	2	2.1	2.1	-5.0%
22	2	2.1	2.1	-5.0%
24	2	1.9	1.9	5.0%
26	2	1.8	1.8	10.0%
28	2	1.4	1.4	30.0%
30	2	2.1	2.1	-5.0%
32	2	2.2	2.2	-10.0%
34	2	1.7	1.7	15.0%
			Average:	2.4%

Table D.4. The precision data for the Y-axis moving in the downward Y direction

Down				
Total Distance Spanned [cm]	Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
2	2	1.9	1.9	5.0%
4	2	1.8	1.8	10.0%
6	2	2	2	0.0%
8	2	2	2	0.0%
10	2	1.9	1.9	5.0%
12	2	2.2	2.2	-10.0%
14	2	2.1	2.1	-5.0%
16	2	1.9	1.9	5.0%
18	2	1.8	1.8	10.0%
20	2	1.9	1.9	5.0%
22	2	2	2	0.0%
24	2	2.1	2.1	-5.0%
26	2	2	2	0.0%
28	2	1.8	1.8	10.0%
30	2	2.1	2.1	-5.0%
32	2	1.9	1.9	5.0%
34	2	1.9	1.9	5.0%
			Average:	2.1%

Table D.5. The precision data for the X-axis moving in the positive X direction

Right				
Total Distance Spanned [cm]	Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
4	4	5	4	0.0%
8	4	3.9	3.9	2.5%
12	4	4	4	0.0%
16	4	4	4	0.0%
20	4	4	4	0.0%
24	4	4	4	0.0%
28	4	3.9	3.9	2.5%
32	4	4	4	0.0%
36	4	4	4	0.0%
40	4	4	4	0.0%
44	4	4	4	0.0%
48	4	4	4	0.0%
52	4	3.9	3.9	2.5%
56	4	4	4	0.0%
60	4	4	4	0.0%
64	4	3.9	3.9	2.5%
68	4	4	4	0.0%
72	4	4	4	0.0%
			Average:	0.6%

Table D.6. The precision data for the X-axis moving in the negative X direction

Left				
Total Distance Spanned [cm]	Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
4	4	5	4	0.0%
8	4	9	4	0.0%
12	4	12.9	3.9	2.5%
16	4	16.9	4	0.0%
20	4	20.9	4	0.0%
24	4	24.9	4	0.0%
28	4	28.9	4	0.0%
32	4	32.9	4	0.0%
36	4	36.9	4	0.0%
40	4	40.8	3.9	2.5%
44	4	44.8	4	0.0%
48	4	48.8	4	0.0%
52	4	52.8	4	0.0%
56	4	56.8	4	0.0%
60	4	60.8	4	0.0%
64	4	64.8	4	0.0%
68	4	68.7	3.9	2.5%
72	4	72.7	4	0.0%
			Average:	0.4%

Table D.7. Z-Axis length accuracy data

Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
8	9	8	0.0%
12	13	12	0.0%
16	17.1	16.1	-0.6%
20	21	20	0.0%
24	24.9	23.9	0.4%
28	28.9	27.9	0.4%
32	33	32	0.0%
36	36.8	35.8	0.6%
40	40.9	39.9	0.3%
44	44.9	43.9	0.2%
48	49	48	0.0%
52	53	52	0.0%
56	56.9	55.9	0.2%
60	60.8	59.8	0.3%
64	65.4	64.4	-0.6%
68	68.8	67.8	0.3%
72	72.7	71.7	0.4%
		Average:	0.1%

Table D.8. Y-Axis length accuracy data

Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
8	15.2	7.6	5.0%
12	19.3	11.7	2.5%
16	23.5	15.9	0.6%
20	27.5	19.9	0.5%
24	31.4	23.8	0.8%
28	35.5	27.9	0.4%
32	38.7	31.1	2.8%
36	42.7	35.1	2.5%
		Average:	1.9%

Table D.9. X-Axis length accuracy data

Commanded Distance [cm]	To Wheel [cm]	Moved Distance [cm]	Percentage Error
8	9.1	8.1	-1.3%
12	13	12	0.0%
16	17.1	16.1	-0.6%
20	21	20	0.0%
24	25	24	0.0%
28	29	28	0.0%
32	32.9	31.9	0.3%
36	36.8	35.8	0.6%
40	40.8	39.8	0.5%
44	45	44	0.0%
48	48.9	47.9	0.2%
52	52.8	51.8	0.4%
56	57	56	0.0%
60	61	60	0.0%
64	64.8	63.8	0.3%
68	68.9	67.9	0.1%
72	72.6	71.6	0.6%
		Average:	0.1%

E. Design Iterations

In order to understand the basis for some of the design choices, it is important to discuss the iterations that the design went through first. This document discusses the different iterations of the testbed.

The original design, shown in Figure E.1, was to use a robotic arm mounted on a carriage. This carriage would be outfitted with a motor and follow along a linear or crescent shaped rail. The rail consists of two parallel tubes, ideally made of aluminum with copper or PVC serving as a substitute for prototypes. This iteration used the Dynamixel AX–18A Robot Actuator from Robotis as the robotic arm of choice. The Robotis arm has the six axis upgrade to add an additional two axes of movement to the base model.

In this design, many problems arise with the robotic arm. The largest issue encountered is the AX–18A servo motors used in the arm. Although many of the axes have two motors installed, there are many issues with the motors not being able to support any weight while performing simple maneuvers. The robotic arm is originally tested using the iPhone 5 as a stand-in for a payload. The iPhone 5 has a mass of 112 g. Besides the issues with the lifting capability of the servo motors, the use of a robotic arm posed interesting complications to modeling the spacecraft movement. Due to the way in which the robotic arm moved, the coordinate transformations necessary

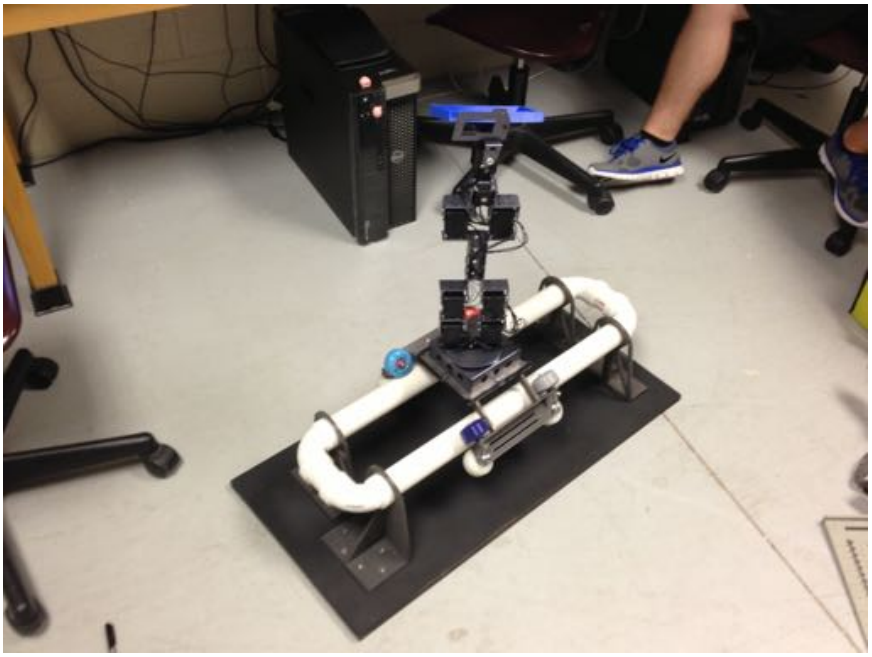


Figure E.1. The first iteration of the testbed design featuring a robotic arm mounted on a carriage

between the axis system used for spacecraft and that of the robotic arm would become complicated and it is expected that the coordinate frame would change with every maneuver since the arm would slip position considerably throughout each movement. For this reason, in the next iteration of the design, the robotic arm was rethought.

The next, and all following iterations, contain a coordinate rail system that is powered by motors. The coordinate rail system replaces the robotic arm. This solved the coordinate frame concerns. This design prevents those coordinate transformations from getting unnecessarily complicated. The coordinate rail system works on only the x, y, z, roll, tilt, yaw system. The advantage here is also that the coordinate system never changes depending on the starting position. The coordinate system in action can be seen in the second major iteration of the testbed in Figure E.2.

Figure E.2. The second iteration of the testbed design with a coordinate rail system

The second major iteration contains the new coordinate rail system along with the linear rail system from the original design. This iteration combined the more successful concepts from previous iterations. This particular iteration has one major issue, instability. With any weight, the structure would lean and sway. The cause of this motion is determined to be the height of the y-axis rail in comparison to both its

width and the loose connection at the bottom. Thus, the next and final design took in the lessons of the previous designs and added stability as a focus.

F. 3D Printing Test

One of the first studies that took place for this project is the practicality of using rapid prototyping for the manufacture of the components for Chronos. The largest area of worry is the capability of the Makerbot 2 to accurately model size with an emphasis on circular features. The system designed for this study is a motor mounting design for a roll, tilt, and pan system for Chronos. The particular component for this study is designed and modeled in Catia v5. The first version of the Computer-aided design (CAD) model follows the specifications given by the manufacturer of the motor. For the second iteration, the motors are in-house, thus true values for the dimensions are able to be incorporated into the designs as well as some features that are not in the manufacturer's drawings. In the third iteration, the focus is more towards accounting for the error from printing since the CAD model is accurate. In the fourth and final iteration, the last printing errors are accounted for. In each version, after the Catia model is improved, the file is set to 0.001 tessellations and exported as a STereoLithography (STL) file. This file is then imported into Autodesk Meshmixer for adjustments to the CAD file (this step does not occur in the 1st iteration), including the addition of supports and checking for faults in the STL file. Finally, the modified files are imported into the appropriate printer accompanying software. The software being used is Makerware for the Makerbot 2 and Catalyst EX for the uPrint. This

is the final step where the part is optimally oriented and rafting or supports can be added. In addition, changes to the print quality can be made.

The data is analyzed using both measurements and fit checks. Calipers are used to precisely measure the dimensions of the printed prototypes. These numbers are compared to the values of the original CAD model. A fit test is performed by trying the motor in each of the printed prototypes. Using the fit test, elements that need to be adjusted and accounted for can easily be spotted.

The data from each of the measurements taken from each of the models is shown Table F.1.

Table F.1. The numerical results of the measurements from each print job and corresponding CAD model

		Screw Radius [mm]	Center Radius [mm]	Mount Radius [mm]	Avg. Wall Thickness [mm]	Length [mm]	Width [mm]	Height [mm]
Test 1	**CAD**	2.5	0.75	1	2	46.3	46.3	36
	Printed	1.87	0.5	0.8	2.045	45.76	45.6	35.69
	% Diff.	25.2	33.33	20	2.25	1.166	1.512	0.861
Test 2	**CAD**	2.5	0.75	1	4	50.3	50.3	36
	Printed	2.36	0.5	0.8	4.013	49.93	49.93	35.7
	% Diff.	5.6	33.33	20	0.313	0.736	0.736	0.833
Test 3	**CAD**	2.5	0.75	1	4	51.67	51.67	51.45
	Printed	2.23	1.1	0.8	4.25	51.02	51.09	51.15
	% Diff.	18.8	46.67	20	6.25	1.258	1.123	0.583
Avg. % Diff.		13.87	37.78	17.33	2.938	1.265	1.123	0.759

The biggest finding from this study is the fact that circular objects print with a lower accuracy than their straight sided counterparts. The height of the part also demonstrates the most accurate print quality. The most overarching finding is that

overall the printed part is smaller than the original CAD model. This shrinkage is dependent on the feature geometry; however, with the exception of a couple of random features, each feature is generally smaller than the given dimensions. The average percentage of error is 22.99% for the holes and 1.52% for the generally straight features. This yielded an overall percent error of 10.72%. The software makes a noticeable difference in the print quality of the part. The addition of Autodesk Meshmixer allows for the fixing of the original STL file exported from Catia. The program fixes holes in the files and provides an improved surface smoothness. This fix is obvious between the first and second prints. The use of the Catalyst program for the uPrint allows for customization of the print setup of the file. It is possible to detail infill, support structures, and the tooling path directly.

A few conclusions are apparent from this study. First, there is a definite inaccuracy when using the Makerbot Replicator 2 for printing and typically the part is smaller than modeled. Second, for non-essential support holes it is better to model them larger so that screws still fit since washers can be used for a hole that is too large. The calculated average percent error of 10.72% closely matches with the described 10% shrinkage of the Makerbot Replicator 2. The prototypes which are modeled are actually usable parts for the purposes of this study. They provide the strength and functionality required to serve the appropriate function in the research supported by this study.