NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MECHATRONICS: THE DEVELOPMENT, ANALYSIS, AND GROUND-BASED DEMONSTRATIONS OF ROBOTIC SPACECRAFT HOPPING WITH A MANIPULATOR

by

Justin L. Komma

December 2018

Thesis Advisor:	Marcello Romano
Second Reader:	Josep Virgili-Llop (contractor)

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
MECHATRONICS: THE DEVELOPMENT, ANALYSIS, AND GROUND-BASED DEMONSTRATIONS OF ROBOTIC SPACECRAFT HOPPING WITH A MANIPULATOR

5. FUNDING NUMBERS

6. AUTHOR(S) Justin L. Komma

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

Astrobee is a robot designed by Intelligent Robotics Group at NASA Ames Research Center to operate inside of the International Space Station (ISS). The robot has a manipulator that is made up of various mechanical, electronic, and control systems. The designed purpose of the manipulator is to perch Astrobee in an effort to minimize power consumption. The study of grasping dynamics and hopping will lead to more efficient maneuvers that would not require propellant. Can the current Astrobee manipulator perform a propellantless maneuver by using its manipulator?

This thesis reports the construction, design, integration, and testing of a robotic manipulator. A replica model of NASA's Astrobee manipulator, with 3 degrees of freedom (3-DOF), was constructed at the Spacecraft Research Laboratory (SRL) of Naval Postgraduate School (NPS) using commercial off-the-shelf (COTS) avionics. The control principle of the manipulator was correspondingly developed. Using the Python scripts, the user can easily interact and control the manipulator. Purposely developed test beds enabled to measure the maximum linear force required to remove the manipulator from a perched rail and determine the gripper slip angle of the manipulator from a three-dimensional (3D) printed ISS rail. We found that Astrobee's manipulator can perform propellantless maneuvers by tossing itself from one ISS rail to another ISS rail.

14. SUBJECT TERMS spacecraft, robotics, dynamics, multi-body mechanics, manipulators, and mechatronics	**15. NUMBER OF PAGES** 123
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

MECHATRONICS: THE DEVELOPMENT, ANALYSIS, AND GROUND-BASED DEMONSTRATIONS OF ROBOTIC SPACECRAFT HOPPING WITH A MANIPULATOR

Justin L. Komma
Lieutenant, United States Navy
BSEE, University of North Florida, 2010

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ASTRONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2018

Approved by: Marcello Romano
Advisor

Josep Virgili-Llop
Second Reader

Garth V. Hobson
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Astrobee is a robot designed by Intelligent Robotics Group at NASA Ames Research Center to operate inside of the International Space Station (ISS). The robot has a manipulator that is made up of various mechanical, electronic, and control systems. The designed purpose of the manipulator is to perch Astrobee in an effort to minimize power consumption. The study of grasping dynamics and hopping will lead to more efficient maneuvers that would not require propellant. Can the current Astrobee manipulator perform a propellantless maneuver by using its manipulator?

This thesis reports the construction, design, integration, and testing of a robotic manipulator. A replica model of NASA's Astrobee manipulator, with 3 degrees of freedom (3-DOF), was constructed at the Spacecraft Research Laboratory (SRL) of Naval Postgraduate School (NPS) using commercial off-the-shelf (COTS) avionics. The control principle of the manipulator was correspondingly developed. Using the Python scripts, the user can easily interact and control the manipulator. Purposely developed test beds enabled to measure the maximum linear force required to remove the manipulator from a perched rail and determine the gripper slip angle of the manipulator from a three-dimensional (3D) printed ISS rail. We found that Astrobee's manipulator can perform propellantless maneuvers by tossing itself from one ISS rail to another ISS rail.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CAD	computer aided design
CAM	computer aided modeling
CSA	Canadian Space Agency
COTS	commercial-off-the-shelf
DOF	degrees of freedom
ETS-VII	Engineering Test Satellite No. 7
FSS	Floating Spacecraft Simulator
ISS	International Space Station
IVA	inter-vehicle activity
JPL	Jet Propulsion Laboratory
LAN	local area network
MATLAB	Matric Laboratory
MBS	Mobile Remote Servicer Base System
MSS	Mobile Serving System
NASA	National Aeronautical and Space Administration
NASDA	National Space Development Agency of Japan
NPS	Naval Postgraduate School
POSEIDYN	Proximity Operation of Spacecraft: Experimental Hardware-In-the-Loop Dynamic Simulator
PUMA	programmable universal manipulator for assembly
SCARA	Selective Compliance Assembly Robot Arm
SPDM	Special Purpose Dexterous Manipulator
SRL	Spacecraft Robotics Laboratory
SSAG	Space Systems Academic Group
SSRMS	Space Station Remote Manipulator System
UDP	User Datagram Protocol

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

There are many people to thank when one completes a 27-month program covering multidisciplinary subjects like mechanical engineering, electronics, computer engineering, systems engineering, control engineering, telecommunications engineering, and robotics.

First, to Dr. Marcello Romano, my advisor: Thank you for your oversight and guidance throughout the thesis process. I appreciate all of the work you have done over the years to develop a state-of-the-art research facility in which all of us can become competent engineers that are capable of so much more that the average graduate student.

Second, to my second reader, Dr. Josep Virgili-Llop, who was always the smartest person in the room: I appreciate your patience not only with me but also with everyone that comes to you for guidance. You always have a way of describing the most complicated ideas so all of us can understand and apply your wisdom.

Third, I would like to thank everyone in the Space Systems Academic Group including my classmates in the Astronautical Engineering curriculum. Every day I was surrounded by people who wanted and demanded more from me even when I thought I had nothing left; I thank you.

Importantly, I would like to thank my family: without you, none of this would be worth it. I am sorry for all of the hours this has taken from you all and hope that the process will lead to a future of adventures we all will remember.

Most importantly, to my wife who enables all of what I do to come to fruition: You push me to be a better man at everything I do. Thank you.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MECHATRONICS AND MANIPULATORS

Mechatronics, the interdisciplinary study of control systems, electronic systems, mechanical systems, and computers (see Figure 1) is a rapidly growing field as the world continues to move towards production automation. *Robotics* and *automated controls* are the key buzzwords within the mechatronics field, and robotic arms are known as *manipulators*. The use of manipulators in space is not new, the ability to collect samples and conduct experiments that may otherwise be impossible by any other means is crucial. The problem with the utilization of spacecraft to gather scientific data is it requires some form of propellant to maneuver the spacecraft; however, this thesis considers *robotic hopping*, a propellantless maneuver where a manipulator can throw a spacecraft from one location to another, thus saving propellant. Incredibly expensive and a limited on-board resource, propellant restricts how much a spacecraft can maneuver. The contact dynamics of manipulators on spacecraft is an active research field and minimum data is available. Overall, the thesis examines whether one specific manipulator, the Astrobee manipulator, can conduct a hopping maneuver within the International Space System. The following sections detail the history of state-of-the-art manipulators.

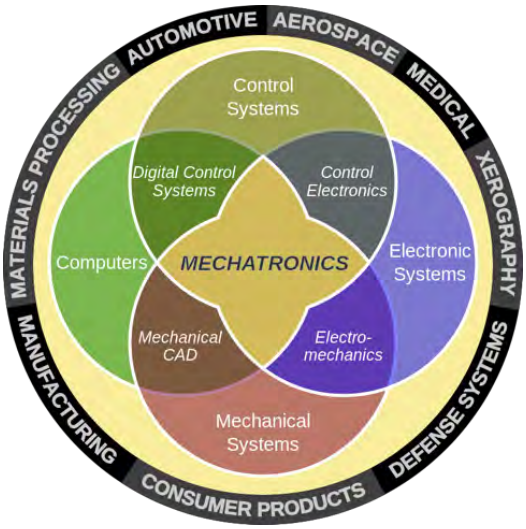

Figure 1. Mechatronics Multidisciplinary Engineering. Source: [1].

Mechatronics is a term that originated with Tetsura Mori in Japan. He was a senior engineer at Yaskawa Electric Corporation who specialized in building mechanical factory equipment. In 1969, he was working on mechanical systems that were slowly being integrated with electronics. He felt that an understanding of both disciplines was uniquely required to be successful at this type of manufacturing [2]. Currently, to be competent at mechatronic engineering, one would need to be proficient at electronic engineering, mechanical engineering, materials science, computer science, systems, control engineering, optical engineering, robotics, Computer Aided Design (CAD), Computer Aided Modeling (CAM), and programming languages [3].

1. The First Manipulators

The first robotic manipulator, called the Unimate was built and sold to General Motors. George Devol in 1954, designed what he called "A Programmed Article Transfer" [4], produced as the Unimate. Devol's patented idea was implemented during the industrial age with partnership of Joseph Engelberger, [5], [6], [7], (see Figure 2).

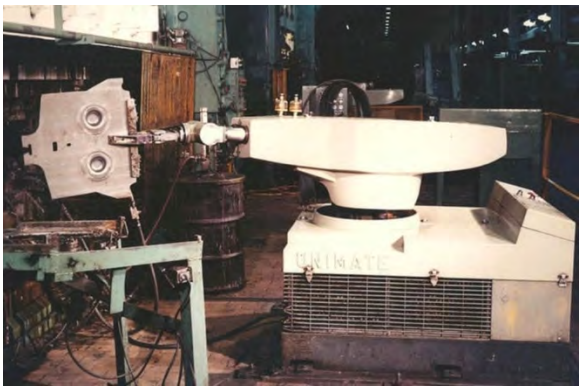

Figure 2. Unimate. Source: [8].

The first computer-controlled manipulator, the Rancho Arm, was acquired by Stanford University in 1963. This 6 degree of freedom (6-DOF) manipulator was developed at Rancho Los Amigos Hospital in Downey, California [5], [6], [7]. The concept of degrees of freedom in a manipulator is further discussed in Chapter II. This was the first

manipulator created in effort to assist handicapped individuals and duplicate normal arm function, (see Figure 3).

Figure 3. Rancho Arm. Source: [9].

The Rancho Arm led to the development of many manipulators, but one of particular importance is the programmable universal manipulator for assembly (PUMA). Victor Scheinman, while at Stanford University, developed the Stanford Arm, which was sold to Unimate, which developed the PUMA with assistance from General Motors [5], [6], [7], (see Figure 4), PUMA arms led to the development of first manipulators used in surgery [10].

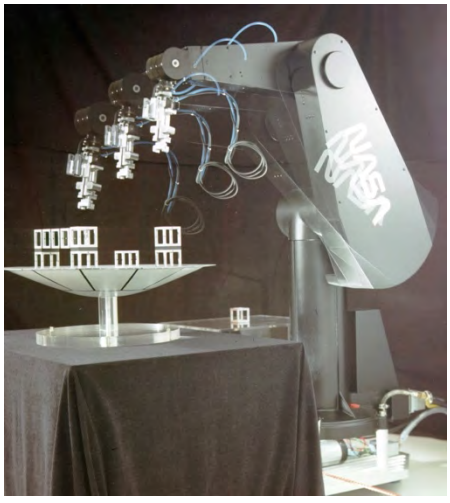

Figure 4. PUMA. Source: [11].

The first pick and place parts manipulator, the Selective Compliance Assembly Robot Arm (SCARA), developed in late 1970s by Hiroshi Makino at Yamanashi University in Japan. The simplicity of its design allowed for quick manipulator to move production items [5], [6], [7], (see Figure 5).

Figure 5. SCARA. Source: [12].

2. State of the Art Manipulators in Space

The use of manipulators has proven to be remarkably fruitful for terrestrial applications, and more recently have also found uses in space. The first manipulator to be used in space the Shuttle Remote Manipulator System (SRMS), or Canadarm, was designed by the Canadian Space Agency (CSA), it operated from 1981 until 2011. The Canadarm had a length of 15.2 m, a diameter of 38 cm, and a 6-DOF [13], (see Figure 6).

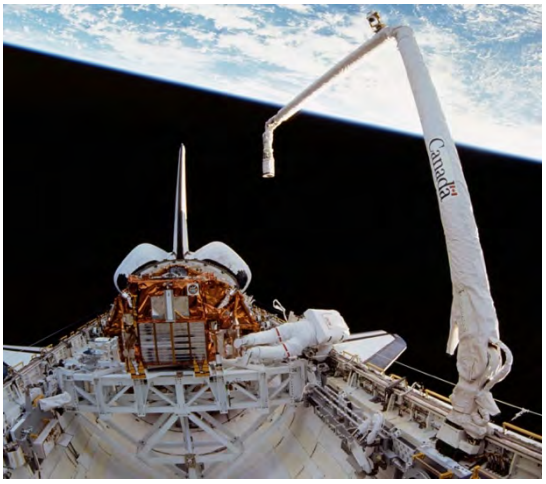

Figure 6. SRMS on the Space Shuttle. Source: [14].

The next generation Canadarm2, designed by CSA, has been in operation since 2001. The Canadarm2 is part of a larger system, the Mobile Serving System (MSS). The MSS composed of three components: Space Station Remote Manipulator System (SSRMS), known as Canadarm2, the Mobile Remote Servicer Base System (MBS), and the Special Purpose Dexterous Manipulator (SPDM). The SSRMS is a 17 m long manipulator with 7-DOF. Attached to the end of the SSRMS is the SPDM called Dextre, which is 3.7 m tall, each of its two arms are 3.5 m long, and each have 7-DOF [15], [16], (see Figure 7).

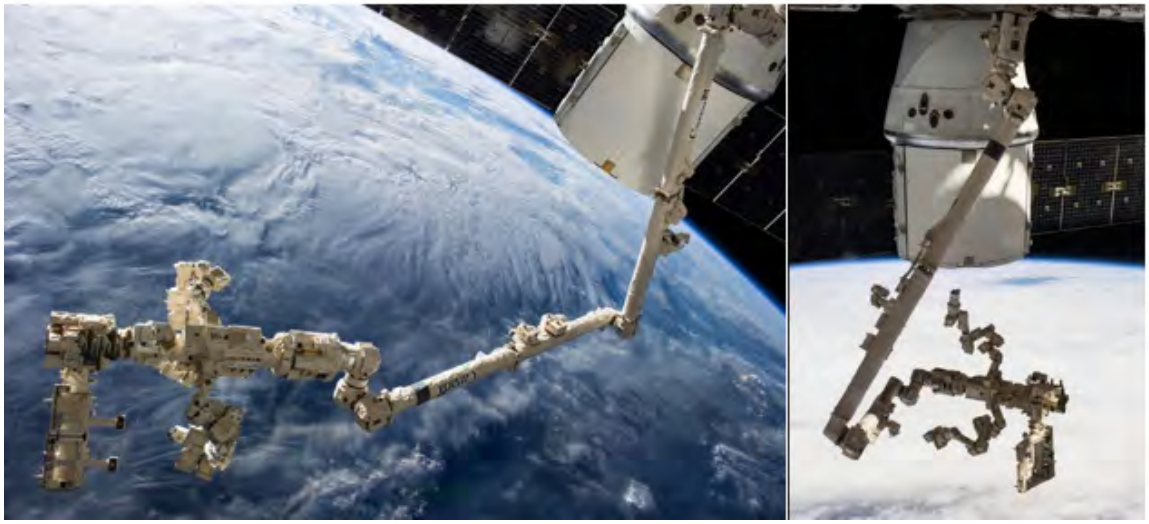

Figure 7. SSRMS, SPDM, and MBS. Adapted from [15], [16].

Another early example of the use of manipulators in space, the Engineering Test Satellite number 7 (ETS-VII) was designed by the National Space Development Agency of Japan (NASDA), (see Figure 8) [17]. The ETS-VII operated from 1997–2002 and was equipped with a 2 m long manipulator to grab a smaller satellite that was launch with it.

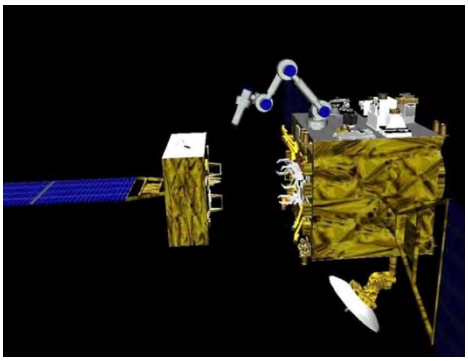

Figure 8. ETS-VII. Source: [18].

Robotic manipulators have also been used in planetary exploration, enabling fixed landers and rovers to collect and analyze ground samples. A whole set of Mars explorers have been equipped with manipulators, from the pair of long-lived Mars Exploration Rover Spirt and Opportunity, the Phoenix Mars Lander, the 1-ton Curiosity rover, to the recently landed InSight lander.

Curiosity, built by NASA's Jet Propulsion Laboratory (JPL) Robotics, is the most complex and capable manipulator ever sent to another planetary surface, enabling sample acquisition, processing, and delivery, as well as contact science operations. The manipulator is a 5-DOF manipulator supporting a 30 kg payload mounted at the end of the arm [19], (see Figure 9).

Figure 9. NASA's Curiosity with JPL Manipulator. Source: [19].

a. Manipulators at NPS Spacecraft Robotics Laboratory

Researchers at the Naval Postgraduate School (NPS) have access to a large granite table. This table and its related navigation and spacecraft simulation equipment is known as Proximity Operation of Spacecraft: Experimental hardware-In-the-loop Dynamic Simulator (POSEIDYN). This is a large square granite table is 16 m^2 in area that allows planar motion to simulate a zero-gravity environment for dynamic experimental testing of spacecraft dynamics [20]. The Spacecraft Robotics Laboratory (SRL) is on their fifth version of the Floating Spacecraft Simulator (FSS). These FSS ride on air bearings that allow the FSS to float on a thin film of air on top of the POSEIDYN granite. The idea of manipulators providing a method of maneuverability is not new for NPS. The dual manipulator was used by a previous NPS Master's thesis student, Andrew Bradstreet [21]. He explored the idea of hopping from one rail to another with the SRL test bed POSEIDYN. Manisat was the first to demonstrate a no propellant maneuver a spacecraft here at NPS [21]. This type of maneuver will be referred to as propellantless. For spacecraft to create a movement from one position to another without the use of propellant would be a propellantless maneuver, (see Figure 10) for the hopping maneuver.

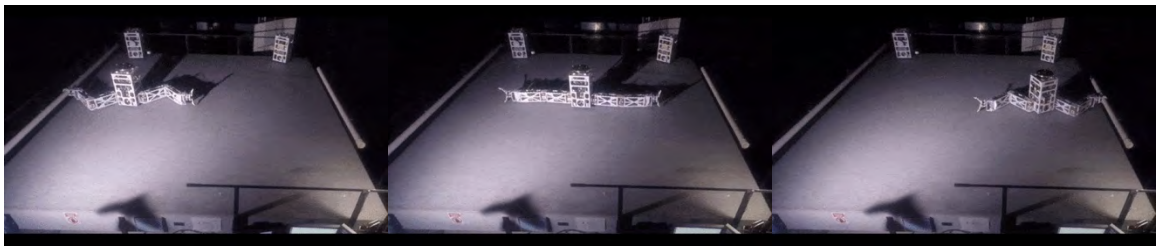

Figure 10. Manisat Hopping on POSEIDYN. Adapted from [21].

b. Astrobee Manipulator

In this thesis, I focus on one manipulator. Astrobee's manipulator is of current interest. Astrobee is a Free-Flying robot design to operate in the International Space Station (ISS). There will be three of these robots in the ISS. Astrobee is a cube 32 cm x 32 cm x 32 cm. It will fly autonomously throughout the ISS interior. Using a battery-operated impeller propulsion system. The purpose of Astrobee is to observe the astronauts, relay

live information back to the ground station at NASA and manipulate objects on the station. Each Astrobee has six cameras and one manipulator [22], (see Figure 11).

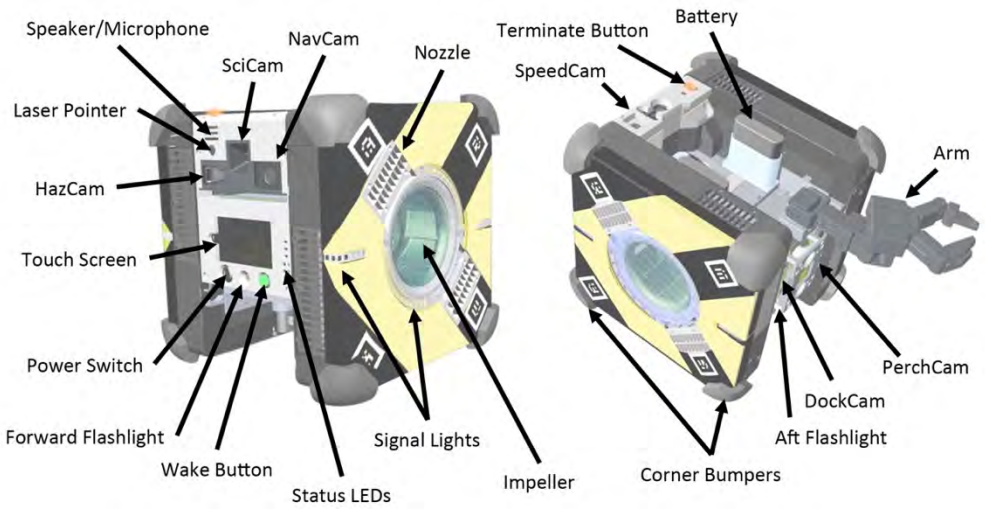

Figure 11. Astrobee. Source: [22].

Each Astrobee has a 3-DOF manipulator that allows it to perch on a rail so that it can save power and easily observe astronaut tasks. Astrobee's manipulator has a limited range of motion [23]. For Astrobee's pan range (see Figure 12), and for the tilt angles (see Figure 13).

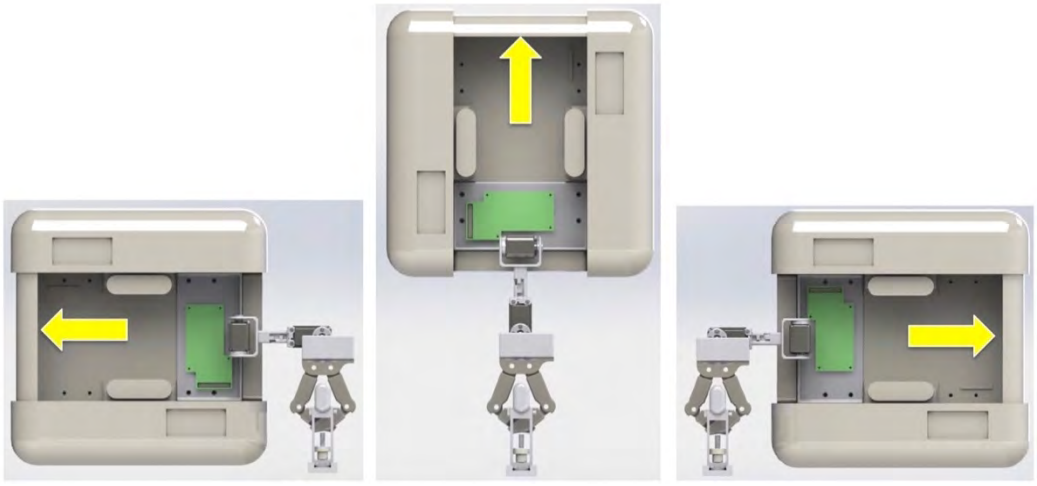

Figure 12. Astrobee Pan Range (-90° to 90°). Source: [23].

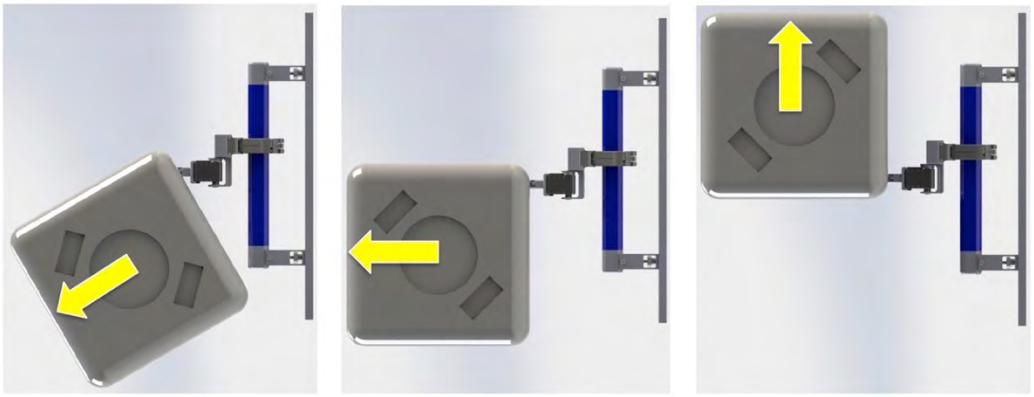

Figure 13. Astrobee Tilt Angles (-30° to 90°). Source: [23].

B. RESEARCH MOTIVATION

The Astrobee manipulator has potential to do more than the designed purpose of perching. Can the manipulator be used to toss Astrobee from one rail to another? To answer that question, we need: to conduct testing on a replica manipulator and to determine the gripper's release forces. The gripper must be able to hold the rail through the intended maneuver; if the force is too low, the maneuver would not be possible. If the gripper can hold, would the maneuver be repeatable, with which accuracy? This required the manipulator to be mounted in a way similar to how it currently is mounted on Astrobee. Once the manipulator is mounted, testing can be conducted on using the manipulator to hop or self-toss from the rail. This hopping maneuver determined that the release of the gripper is clean and that the maneuver can be repeated. NASA Ames and NPS agreed that this partnership would lead to useful data about Astrobee's capabilities.

C. RESEARCH OBJECTIVES

After conducting research on manipulators and their versatility in particular to space and throughout the DoD, building a manipulator was the next step. The design, development, and construction of a manipulator would be the first step to understand Astrobee's manipulator and its functionality.

Once built, the release force needed to be tested in a linear test bed. In order to achieve any type of propellantless maneuver, the maximum release force must be known.

The current simulated force is 0.5 N. This assumption has been made by the NPS SRL. Because no linear test bed was available, one was designed for this project. With the known release force, a verification of the simulation could be made.

Knowing that the hopping maneuver is possible, a demonstration is required to simulate the zero-gravity environment on the POSEIDYN test bed. In these experiments, the manipulator can be used to launch one of the FSS. This hypothesis provides a proof of concept for the hopping maneuver through a verification test.

After all of the data is gathered, an evaluation of the manipulator was conducted and the results were analyzed.

D. THESIS ORGANIZATION

This thesis is organized as follows: Chapter II describes the design, development, and construction of the NPS manipulator including DOF to better understand how the manipulator functions. Chapter III explains how the linear test bed was developed and the experimental results of the gripper release. Chapter IV explains the POSEIDYN test bed and the experimental results of the self-toss maneuver while Chapter V gives concluding remarks.

II. DEVELOPMENT OF THE MANIPULATOR

A. OVERVIEW

This chapter discusses how the manipulator was built. Astrobee's manipulator is quite unique. This manipulator is designed to perch Astrobee to a rail allowing it to observe the astronauts as they work in the International Space Station (ISS). With assistance from NASA, the development of an Astrobee manipulator replica for NPS began. The hardware and software used to create and control the manipulator is covered in this chapter. The end result was for NPS to conduct simulations, experiments, and demonstrations on this manipulator in an effort to prove that the simulated maneuvers are plausible.

A manipulator is a Multibody (MB) system which refers to a collection of bodies coupled by joints. The bodies are referred to as links and will be considered rigid bodies for this thesis. The MB systems are the bases of how a manipulator is constructed. A joint is connect to a link. Generally, joint 0 is the joint at which the base or link 0 is attached. From this point, link 0 will be only referred to as the base and joint 0 is not be discussed as it is only the method of how the base is attached to each of the test beds that are developed. The final link of the manipulator is called an end effector, link 2 would be the end effector. (See Figure 14) for an overview of the joints and links involved in this thesis.

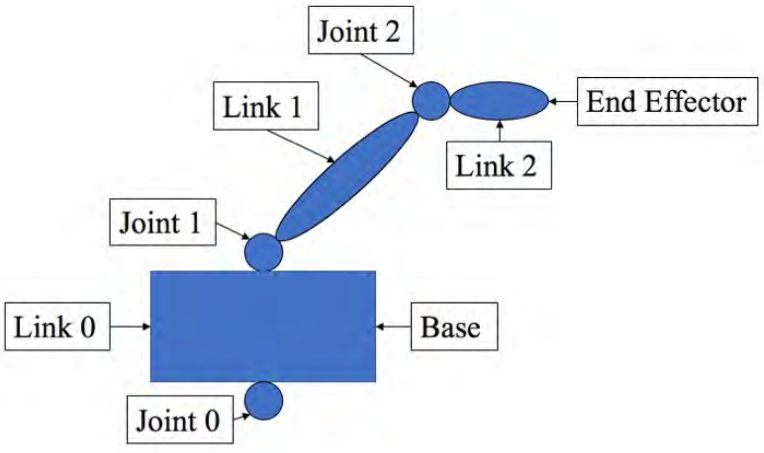

Figure 14. Robotic Manipulator Joints and Links

In actuality, this manipulator has 7 joints and 7 links, (see Figure 15). Joints 3–7 and links 3–7 are all controlled by one gripper motor. The joints and links all move in relation to the gripper motor as they are all connected by a tendon. The gripper motors speed and direction are controllable and open and close the gripper. The gripper motor is physically located inside of link 2. To remove any confusion, joints 3–7 and links 3–7 will no longer be discussed in this thesis. The end effector is known as the gripper and will be analyzed as one link, seen inside of the red circle, (see Figure 15).

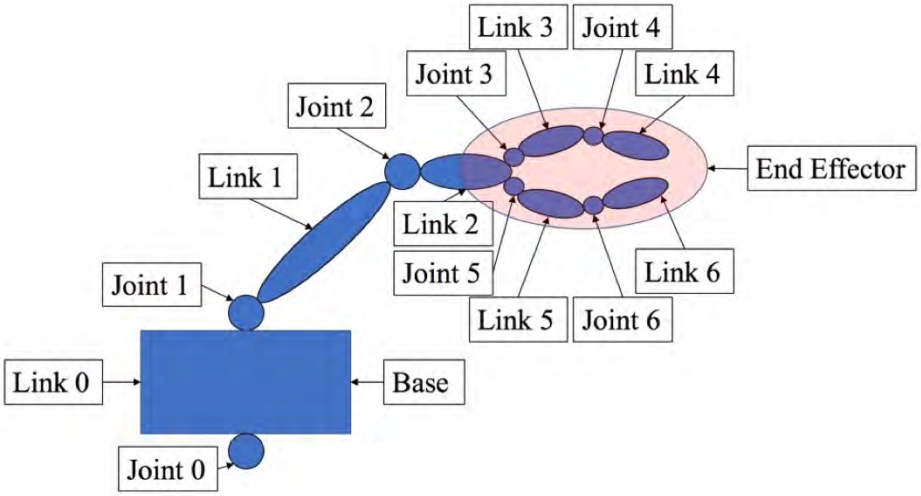

Figure 15. Robotic Manipulator Actual Joints and Links

The manipulator is a 3 Degree of Freedom (3-DOF) robotic arm. 3-DOF means that the device can angularly move at three different unique joints. Joint 1 and joint 2 have 180 degrees of available rotation. The rotation is accomplished by the joints and in this case revolute joints. The gripper also has 1-DOF which is associated to the gripper motor inside of link 2. Section D.2 discusses more detail on the functionality and interconnection of the gripper. Joint 1 allows rotation of all the attached links and joints to rotate about that axis in that plane defined at Joint 1, (see Figure 16). The red arrow displays the directions of rotation. Notice how the rest of the manipulator remains ridged as Joint 1 rotates about its axis of rotation.

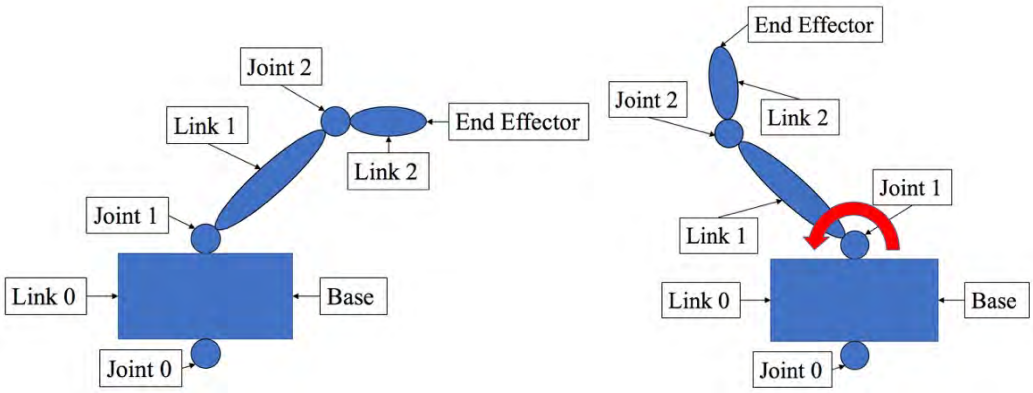

Figure 16. Joint 1 (1-DOF)

Joint 2 allows rotation of all the attached links and joints following it to rotate about this new axis in that plane defined at Joint 2, (see Figure 17). Unlike Joint 1 where the entire manipulator rotates, Joint 2 only allows the gripper to rotate.

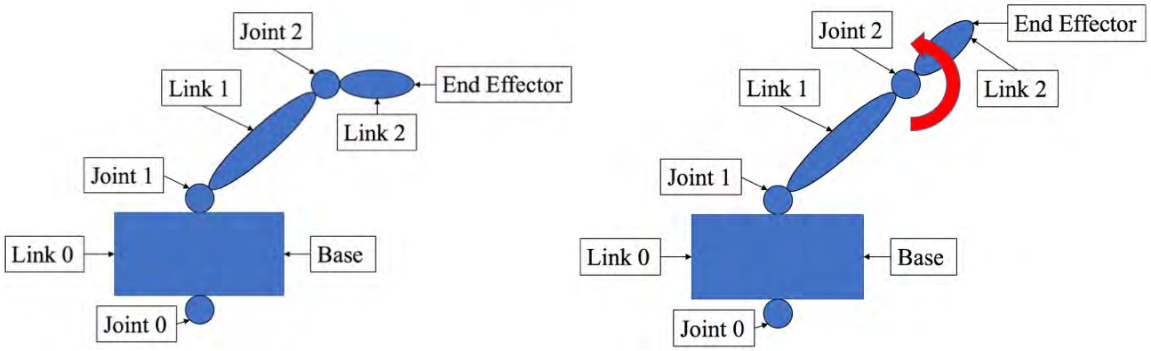

Figure 17. Joint 2 (1-DOF)

The axis of rotation of Joint 2 is perpendicular to Joint 1, (see Figure 17). This can be seen in more detail in actual manipulator, (see Figure 18). The third and final degree of freedom is the gripper. It can open and close given the manipulator's 3-DOF. From this point on, the three degrees of motion will be at Joint 1, Joint 2 and the Gripper.

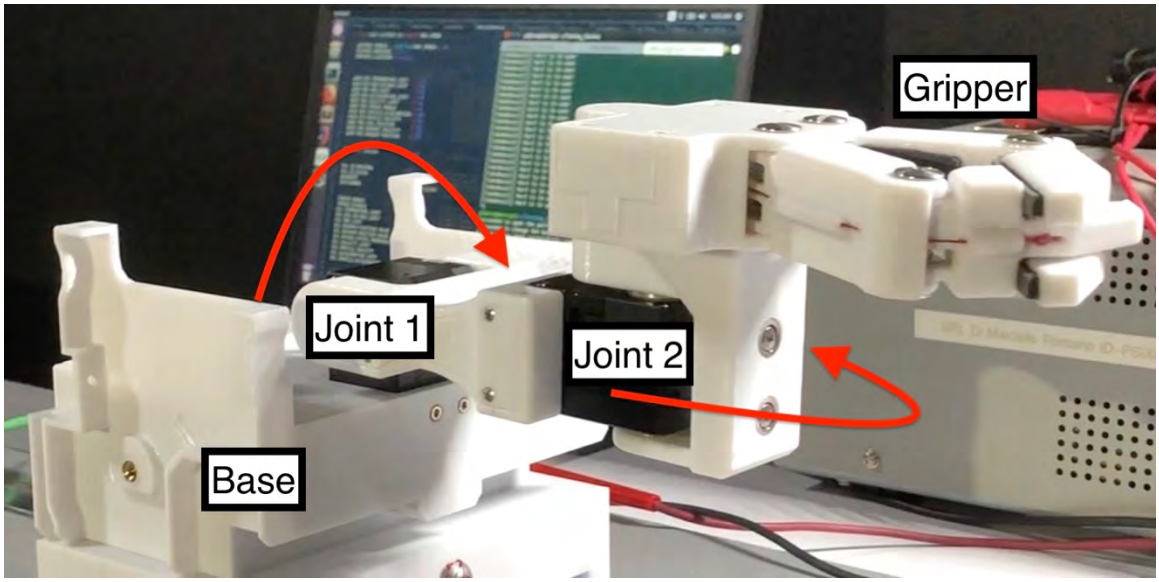

Figure 18. The Newly Developed NPS Manipulator Replica of the Astrobee Perching Arm

B. HARDWARE

To build the manipulator some hardware was acquired to provide control and functionality desired. The avionics on Astrobee are complex and expensive as it needs to survive the launch environment and the ISS environment. Custom made avionics allow the servos in Astrobee to be controlled with simple off the shelve items. The substitutions allowed for cost effective control required to operate the manipulator. The Raspberry Pi, Pololu motor driver, and an RS485 adapter allowed for custom avionics to control the servos and gripper motor, (see Figure 19). The following section discusses the individual hardware used in the building of the manipulator.

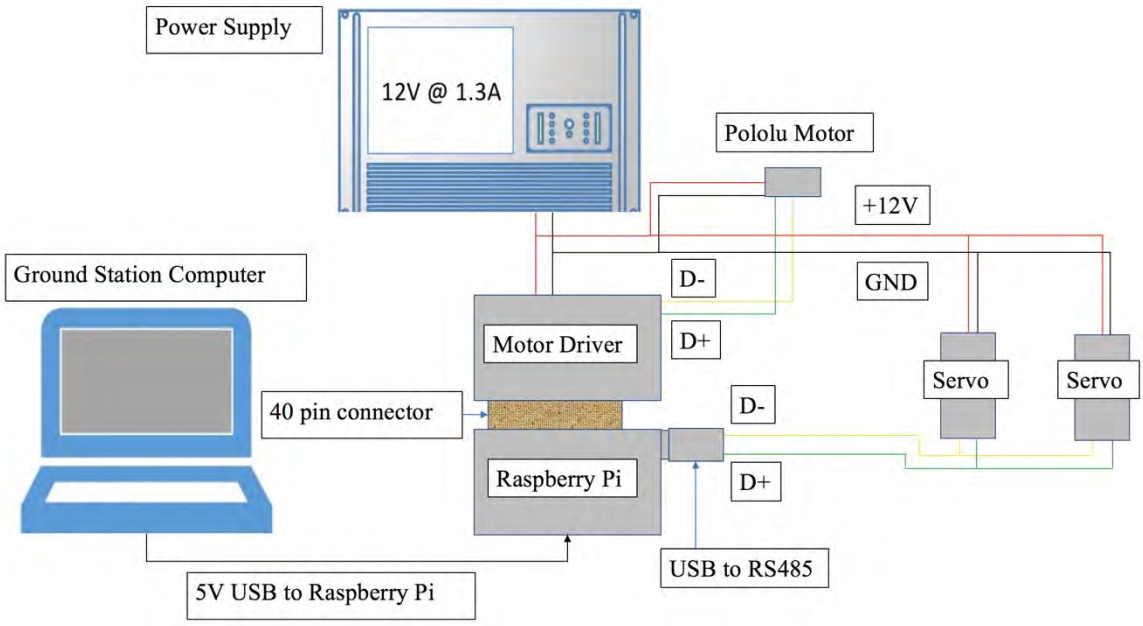

Figure 19. Wiring Diagram of Manipulator

1. Raspberry Pi

The Raspberry Pi 3 model B was chosen to be the controller for the manipulator. One key feature is the wireless Local Area Network (LAN) connectivity. The controller was formatted with the recommended Raspbian operating system to work with the Ubuntu operating system running on the Ground Station Computer (GSC). The Raspberry Pi was chosen because of the availability of free code and the available tutorials. There are many resources for connecting servos to a Raspberry Pi, making it a popular choice for off-the-shelf controllers. The Raspberry Pi can understand multiple coding languages, but the developers recommend Python [24]. All programing with the Raspberry Pi is in Python, (see Figure 20) for the Raspberry Pi 3 B.

Figure 20. Raspberry Pi 3 B. Source: [24].

The Raspberry Pi needed an RS485 USB adapter to create a serial communication path to the servo, (see Figure 21). The RS485 USB adapter can be connected to any of the four available USB ports on the Raspberry Pi. All that is needed once connected is to connect the servos to the D- and D+, further discussed in the commissioning of NPS's newest manipulator.

Figure 21. RS485 USB Adapter. Source: [25].

2. Motor Driver and Motor

The manipulator has a Pololu 12 V motor to control the opening and closing of the gripper. The Pololu 12 V motor will be referred to as the gripper motor, (see Figure 22 (left)). For the motor to function and be controlled it must be connected to a magnetic encoder, (see Figure 22 (right)).

Figure 22. Pololu Motor (Left) Motor and Magnetic Encoder (Right). Source: [26].

The encoder is soldered to the motor driver, and a magnet is attached to the smaller shaft outside of the encoder. The encoder is connected to the motor driver. The motor driver controls the bidirectional brushed DC motor. The motor driver was ordered partially assembled, (see Figure 23 (Left)). The motor driver is also from Pololu and is designed to stack on top of the Raspberry Pi [27], (see Figure 23 (right)).

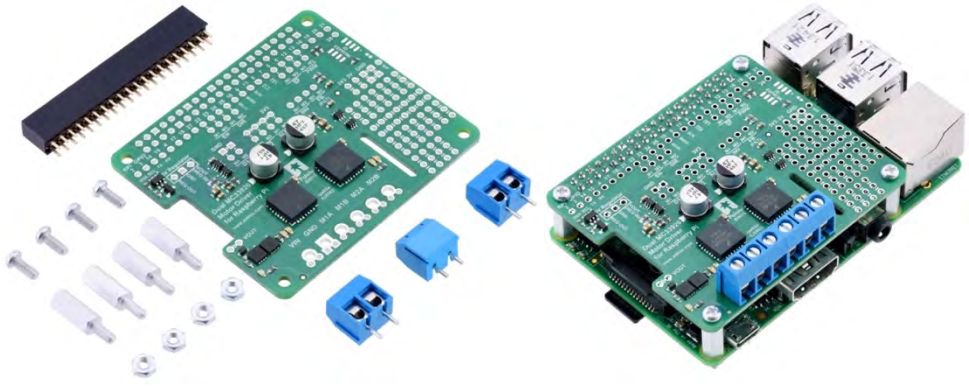

Figure 23. Pololu Motor Driver for Raspberry Pi. Source: [27].

3. Servo

The servos that are used in the Astrobee manipulator are the Dynamixel XH430-W210R. These are the same servos that are used in the NPS replica. Both servos are connected to the RS485 asynchronous serial communication port. This allows for the Raspberry Pi to control each of the servos independently [28], (see Figure 24).

Figure 24. Dynamixel XH430-W210R Servo. Source: [28].

4. Load Cell

To test all of the forces felt at the gripper a 6-axis load cell is utilized. The ATI nano43 sensor is used throughout the experiments. The sensor can read forces in the X, Y, and Z axes (up to 36N). The forces are referred to as F_x, F_y, and F_z. The sensor can read torques seen about the X, Y, and Z axes (up to 500N·mm) [29], referred to as T_x, T_y, and T_z from this point on, (see Figure 25) for the sensor.

Figure 25. ATI Nano43 Sensor. Source: [29].

C. SOFTWARE

1. Siemens NX12

To design parts to be 3D printed a Computer Aided Design (CAD) tool was selected. Siemens is the developer of NX12. Siemens has developed many extension free tutorials to help better understand some of the capabilities of their tools. NX is the software that has been used throughout the 591 curricula here at NPS. (See Figure 26) for examples of parts designed for this project in NX12.

Figure 26. Siemens NX12 CAD Software

Siemens software allows users to design parts, and save those parts as stereo lithography files or STL file that are easily understood by most 3D printers. STL is the current standard for saving files that contain data about shapes in 3 dimensions. “The first 3D printer was invented by Chuck Hull in 1987 at 3D systems. The same person was behind the STL file format” [30]. The file describes the surface geometry of the part so that it can be recreated by a 3D printer.

2. Python

The programing language recommended by Raspberry Pi developers is python. Python is an object-oriented, high-level programming language [31]. There is an extensive library that is supported free of charge for many operations systems. Python has been the programing language of choice for the 591 curricula at NPS and works well with Linux based systems such as Ubuntu. Ubuntu is the operating system used on the GSC.

3. MATLAB

To capture and analyze the data from the experiments Matrix Laboratory (MATLAB) was used. “MATLAB is a programming platform designed specifically for engineers and scientists. The heart of MATLAB is the MATLAB language, a matrix-based language allowing the most natural expression of computational mathematics” [32]. MATLAB is the tool of choice for the 591 curricula at NPS for analyzing data, performing numerical simulations, creating graphs, and plotting results.

D. COMMISSIONING OF THE MANIPULATOR

1. 3D printing

The first step in assembly was retrieving CAD files of the individual pieces. Each CAD file was imported to NX12 and then converted to STL files to be sent to the 3D printer. The NPS Space Systems Academic Group (SSAG) has access to several 3D printers. The model Stratasys Fortus 400mc 3D printer was utilized for all 3D printing in this thesis. The Stratasys Fortus 400mc 3D printer prints the desired pieces in polycarbonate which is known for its impact strength and durability [33]. The pieces to the NPS manipulator are intricate, (see Figure 27). Notice the build-up material on the 3D printed parts, (see Figure 27 (left)). This is used to strength the piece as it is printed until it can cool and harden. This build-up material must be broken off after it is printed, (see Figure 27 (right)).

Figure 27. NPS Manipulator 3D Printed

The gripper does have some unique aspects to the fingers of the gripper. There are a proximal and distal links. The proximal links are identical for each finger. The distal links are different. This is to allow the gripper to close over itself which creates a grasping quality to the gripper. One of the distal links has a single fingertip, the other has a double fingertip. They will be referred to as single distal and double distal links in this thesis. On the left side of the gripper is the double distal link, the single distal link is seen on the right, (see Figure 28).

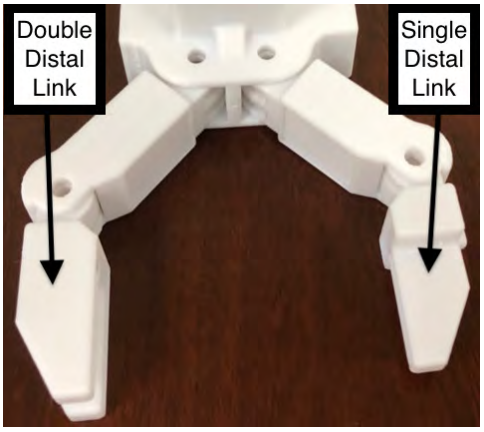

Figure 28. Gripper Distal Links

The final 3D printed manipulator, (see Figure 29).

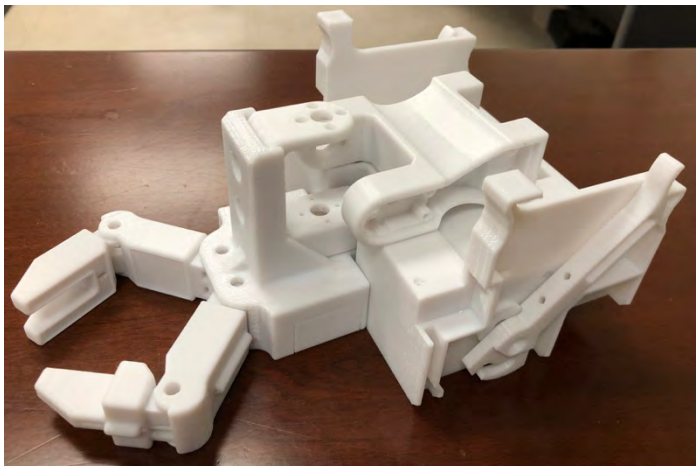

Figure 29. NPS 3D Printed Manipulator

2. Range of Motion

The gripper opens with the use of a gripper motor and tendon system through the gripper links. The gripper motor spins creating tension in the tendons which cause the grippers fingers to open. Each location of a joint in the gripper contains torsional springs. The torsional springs provide joint flexion to keep the gripper closed. The gripper closes using the same gripper motor and tendon system but now with assistance from the torsional springs. This is gripper's natural position. The tendons, seen in green, travel from the gripper motor through the proximal links and distal links to the fingertips, where the tendon is tied off. The torsional springs, seen in blue, are placed at the joints, (see Figure 30).

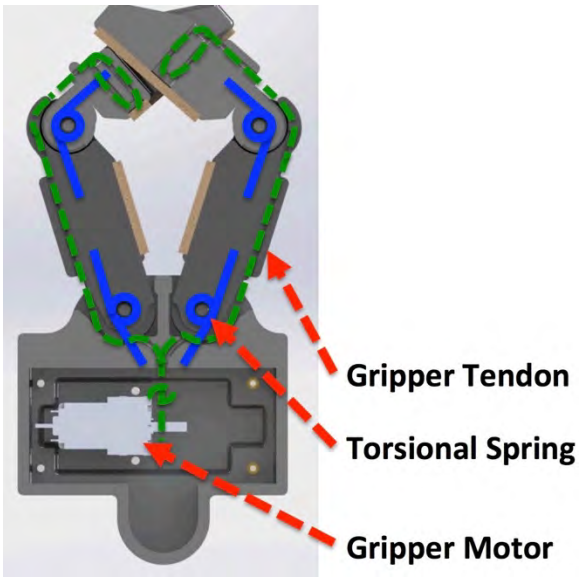

Figure 30. Gripper Tendon and Spring Locations. Source: [23].

For the NPS gripper motor there was a modification to the design. During initial testing and setup, it was found that the tendons would wrap around each other causing each finger to move in an unsymmetrical pattern. Sometimes they would work correctly, but every so often, it would bind up at the gripper motor shaft. One time it cut or broke the tendon. This was a time-consuming incident as the tendons needed to be completely rerouted again. This is not desirable; the placement of a screw and separator solved this issue. The placement of this screw and separator created a distance between the two

tendons allowing for two separate location to wind up the tendon. The unsymmetrical finger-opening pattern and tendon breaking issues were resolved with this modification, (see Figure 31). NASA Ames Astrobee has not had any issue of this type. NPS's manipulator may have had sharp edges on the shaft of the gripper motor from when the shaft was drilled to insert the tendon. The shaft was sanded smooth to remove any sharp edges, no future tendon issues would be found.

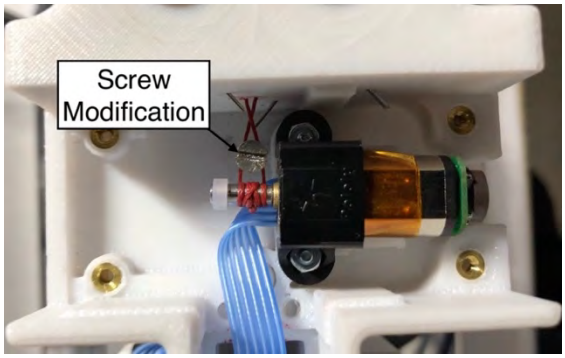

Figure 31. Gripper Motor Modification

Each of the proximal gripper links can open 45 degrees. This creates a 90-degree opening between the two-gripper fingers, (see Figure 32). When the gripper is fully open, the double and single distal links go just beyond parallel to one another.

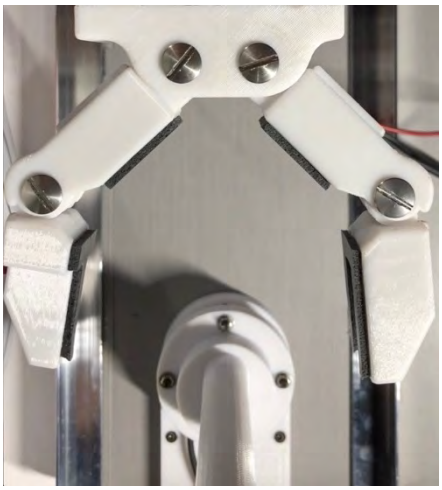

Figure 32. Gripper Opened

Joint 1 and Joint 2 has been restricted to 180 degrees of rotation to avoid any possible collisions with other components. Both joints have the ability to move a larger range of motion but are not required for this thesis. Joint 1 is set to -90 degrees or open, (See Figure 33 (left)). Joint 1 set at 0 degrees, which stands straight up or perpendicular to the open or closed position, (see Figure 33 (center)). Joint 1 is set to 90 degrees the manipulator is completely closed, this is called home position, (see Figure 33 (right)). This is the full range of motion of Joint 1, (see Figure 33).

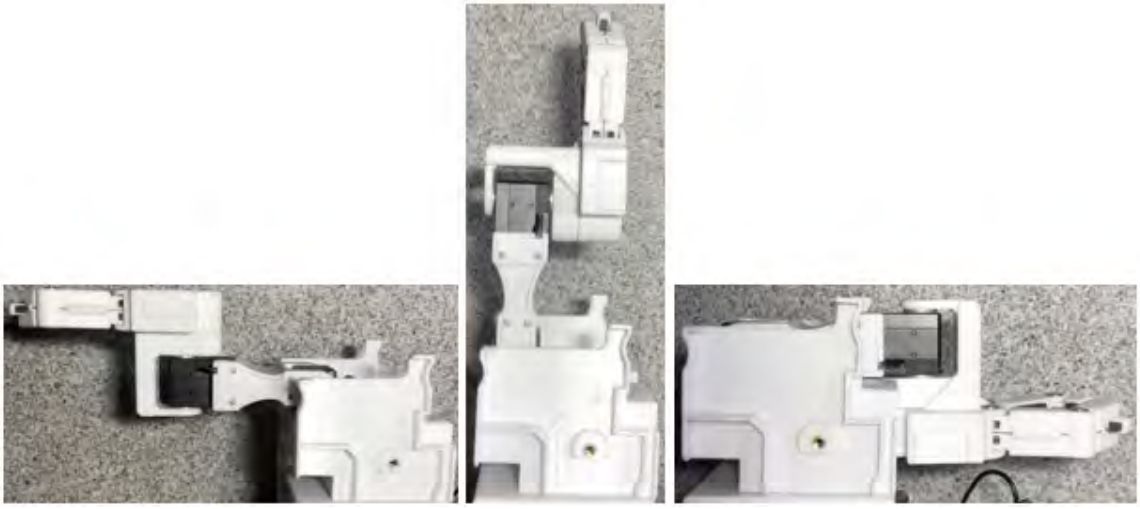

Figure 33. Range of Motion of Joint 1.

3. Operational Testing

a. Servos

Once the manipulator had been assembled the operational testing begun. Using an incremental approach in testing one item at a time would be energized and tested. First controlling of the servos needed to be tested. Starting with Joint 1, the addressing on each servo needed to be defined individually before the two servos could be connected in series. Joint 1, in the code proximal, was set to address 1. Using code that was provided by Dynamixel some modification was made to work for this manipulator. To make modification all that was needed was to add whatever control address from Dynamixel

product manual [34]. Once an address is created, setting changes can be made as required. The same process was used for Joint 2. The code created is called:

RA_Constants.py

b. Gripper Motor

The gripper motor was controlled form the same Python program as Joint 1 and Joint 2. The gripper only needed torque to be adjusted down and the gripper open and close times were set. The opening and closing time were set to approximately 2 seconds. The gripper setting was added to the RA_Constants.py file. A new file was created:

RA_Functions.py

This program sets up all the communication ports, sets up the servos, provides goal locations, reads the present position of the servos, starts the timer, and set the home position. All operations require the following that RA_Constants.py and RA_Functions.py are available, (see Figure 34). These two programs are imported to the final program, which performs whatever task is required of the manipulator, (see Appendix A. Python Code). Several programs have been created to use this manipulator but to narrow the code to two programs that were used heavily in this thesis are:

test_90.py - This program was used to test the linear forces.

self_toss.py - This program was used to test the self-toss maneuver.

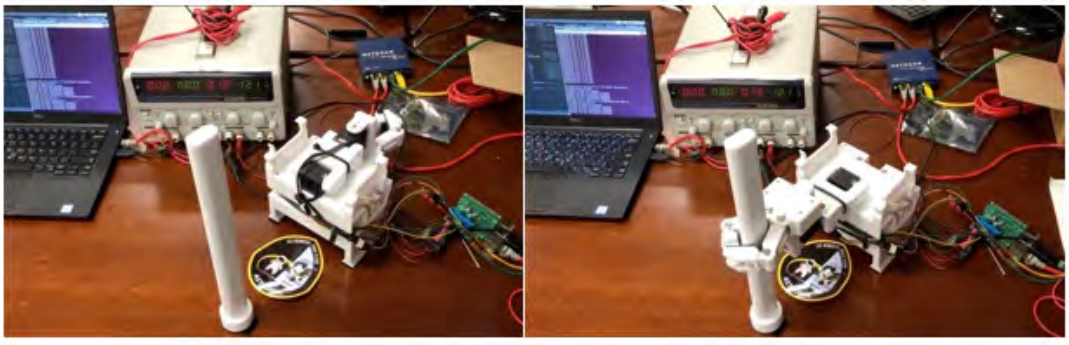

Figure 34. Manipulator Operational Testing

THIS PAGE INTENTIONALLY LEFT BLANK

III. GRIPPER FORCE TESTING

A. OVERVIEW

What are the max forces that can be applied before the gripper can no longer hold onto a rail to which it is perched to? The concern with testing the hopping maneuver is whether it could be conducted on the ISS. To measure this force a test setup analogous to a tensile strength test is used. The manipulator, with the gripper grasping the handrail is subjected to a controlled displacement, forcing the gripper to fail and release the handrail. The forces applied to the handrail are measured with a load cell and thus the max force that the gripper is capable to hold is empirically determined.

This chapter describes how the test bed was developed and the adapters that needed to be designed to hold the rail and sensor all while mounting them to the test bed. Then a procedure of how to set up the test bed and how the data was collected is set. Finally, conclusions are made on the results of the linear testing.

B. LINEAR EXPERIMENTS

1. Linear Test Bed Development

Testing of the gripper force required a test bed to be developed to control the motion of the manipulator along a single axis. A high-level view of what the linear test bed needs to achieve, (see Figure 35).

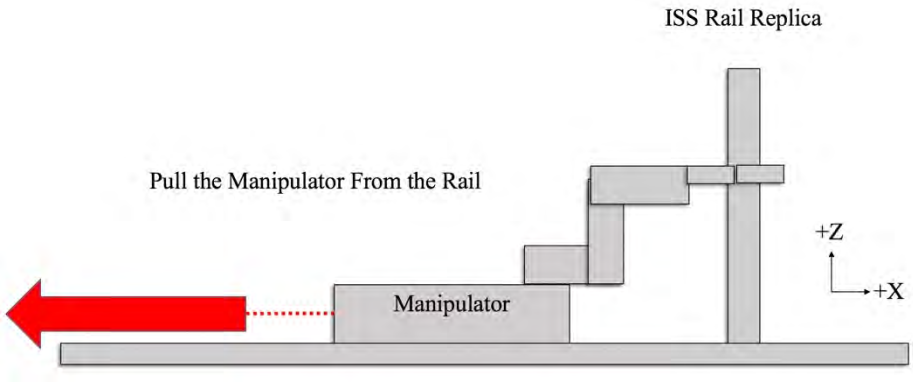

Figure 35. Linear Test Overview

The problem statement here is how to control a 3-DOF manipulator in a single axis with constant velocity and at a speed that would be experienced during the hopping maneuver. A device needed to be designed that would limit the manipulator to a single axis of motion. An adapter was designed to mount the manipulator and slide along two rails that restrict the base to only move along the ± X-axis. The adapter base sled and setup fundamental was developed, (see Figure 36). The design used a stepper motor mounted to a lead screw that would control the movement of the adapter base sled. Due to fiscal limitation during the design phase, a more resourceful test bed needed to be developed.

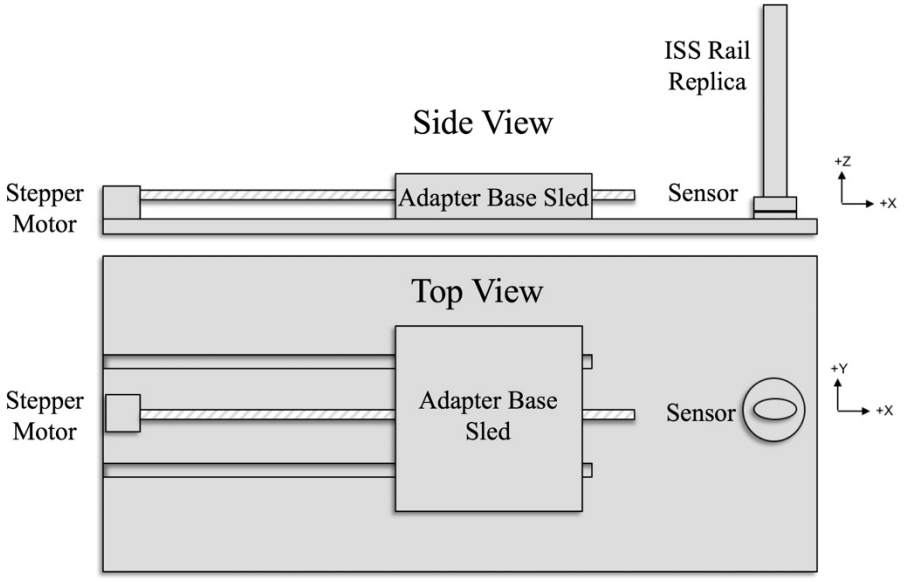

Figure 36. Initial Linear Test Bed Concept

Using the same concepts as the initial linear test bed, a wooden plate with two polished aluminum rails was mounted parallel to one another to create the same restricted motion in the Y-axis. To control the motion of the sled in the X-axis, an additional gripper motor was installed with a pulley attached to the sled. Thus, allowing the manipulator to be pulled with a constant velocity in the + X-axis. (See Figure 37), the long polish rail system allows the sled to slide along the X-axis, on the right is the sensor adapter mounting location, in the center the adapter base sled. The adapter base sled can be seen attached to the tendon cord which feeds through the pulley to the gripper motor. As the gripper motor

rotates, it winds up the tendon which pulls the sled via the pulley system. With the addition of the aluminum rails and a constant voltage of 2 V at 150 mA applied to the motor, a constant velocity in the X-axis can be reiterated.

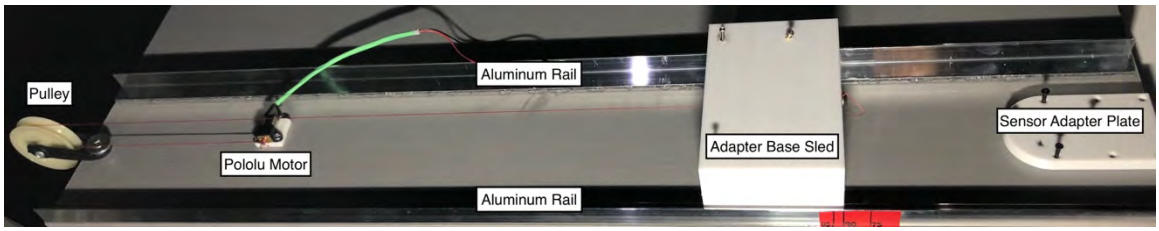

Figure 37. Linear Test Bed Rail System

2. Mounting the Load Cell

Mounting the load cell sensor took some design work to create two sensor adapter plates and one rail adapter. The rail adapter needed to securely hold the rail throughout all testing. Using the NX12 software to design the adapters allowed for a polycarbonate material to be 3D printed. A12 mm deep hole was designed so that the rail could fit inside the hole and mount to the rail adapter. Also, a large screw was placed through the bottom surface of the rail adapter to keep the rail in place. The ISS rail which is 30.5 cm long and fits into the rail adapter, rail (see Figure 38 (top)), rail adapter (see Figure 38 (bottom)).

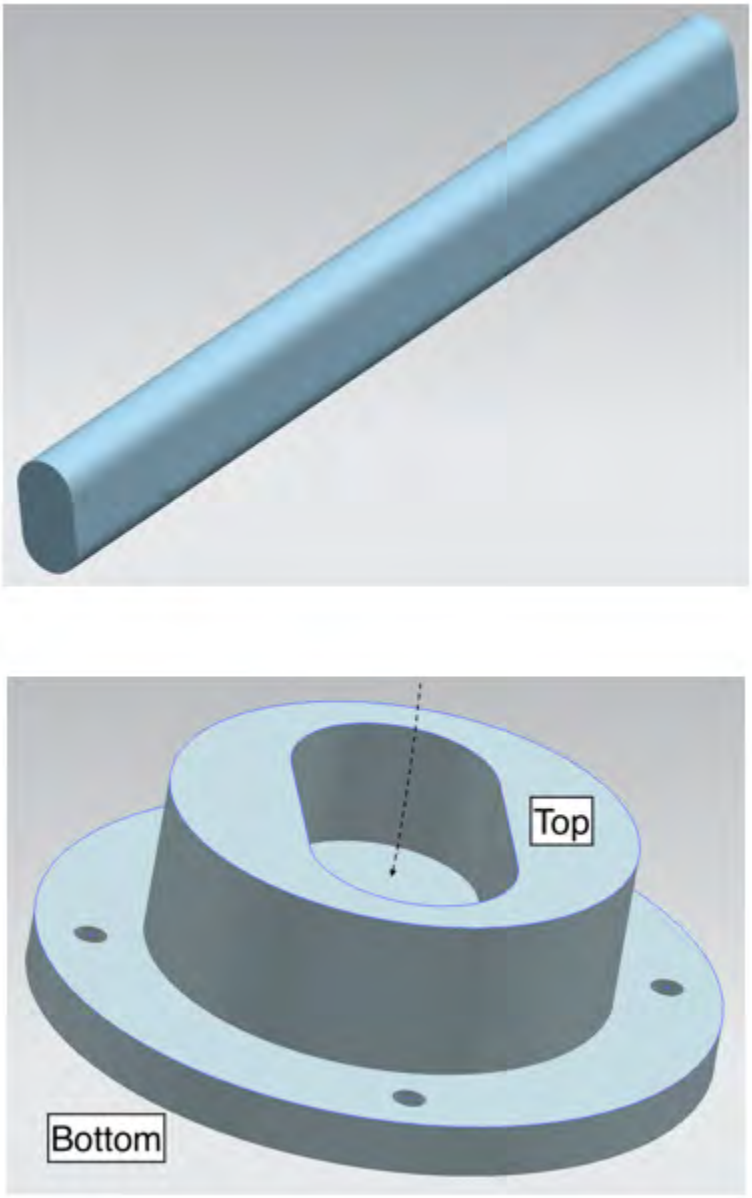

Figure 38. NX12 Rail (top) / Rail Adapter (bottom)

The sensor requires two adapters. One adapter is on the tool side of the sensor and the other is on the mounting side of the sensor. The tool side adapter to the sensor, (see Figure 39 (top)). The mounting side adapter for the sensor, (see Figure 39 (bottom)).

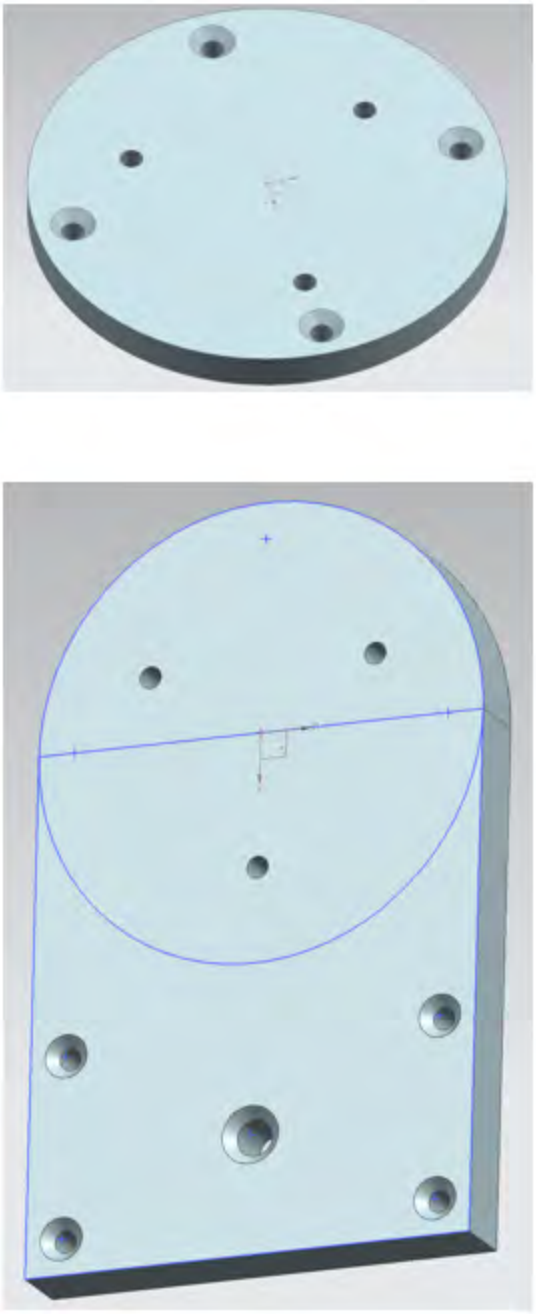

Figure 39. NX12 Sensor Adapters. Tool Side Adapter (top) / Mounting Side Adapter (bottom)

Once all of the adapters were 3D printed, the sensor could be mounted to the test bed. The overall view of the sensor mounted to the adapters, (see Figure 40). On the top of this figure, you can see the rail adapter holding the ISS rail replica. Below that is the tool

side adapter that mounts to the tool side of the sensor and the rail adapter. The next layer down, the nano43 sensor is seen with the umbilical cord coming out of it. Finally, the mounting side adapter is mounted to the mounting side of the sensor and the bottom plate of the test bed.

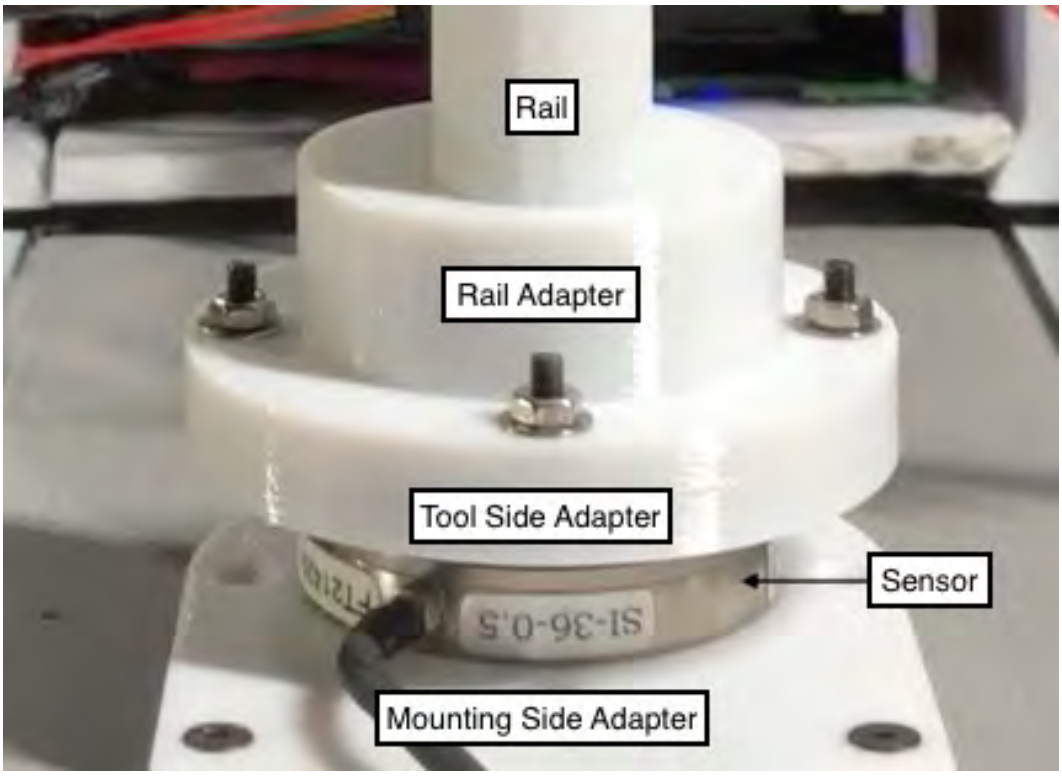

Figure 40. Sensor Adapters Mounted to Sensor and Rail

3. Test Bed Assembly

To put the test bed together the following diagram can be used to recreate the interconnecting elements of the test bed. (See Figure 41) for the command and control of the test bed. The ground station computer controls the manipulator and supplies 5 V via the USB cable to the Raspberry Pi 3. The two servos are powered by a power supply regulated to 12 V at 1.3 A. Another power supply powers the gripper motor to 2 V at 150 mA. The sensor is attached to the data acquisition device (1001 DAQ) connected to a PCI port on the rear of the computer tower. The computer recorded the data in matrix files in MATLAB.

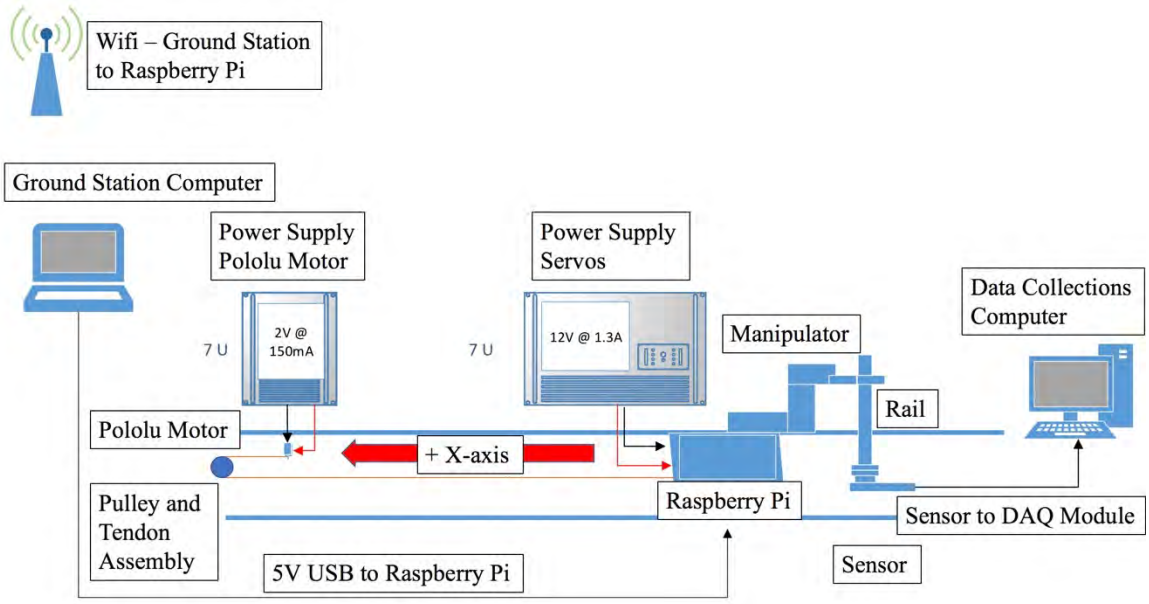

Figure 41. Command and Control of the Test Bed

The motion of the manipulator in the + X-axis is approximately 75 mm. To initiate the +X direction, the power supply to the gripper motor must be energized. When power is applied, the motor spins and slowly wind up the tendon. Thus, the pulley system begins to pull the sled holding the Raspberry Pi, motor driver, and manipulator at a constant velocity. The gripper releases from the rail over a period of 12 seconds. The gripper needs to be removed slowly for a reason: by minimizing the transient and dynamic effects on the manipulator, a static loading effect is analyzed. The intended hopping maneuver is expected to be slow and controlled. The slow transition between grip surfaces could reveal critical information that may be useful to future real-world testing in the ISS. (See Figure 42) for the actual test bed that was used for this experiment.

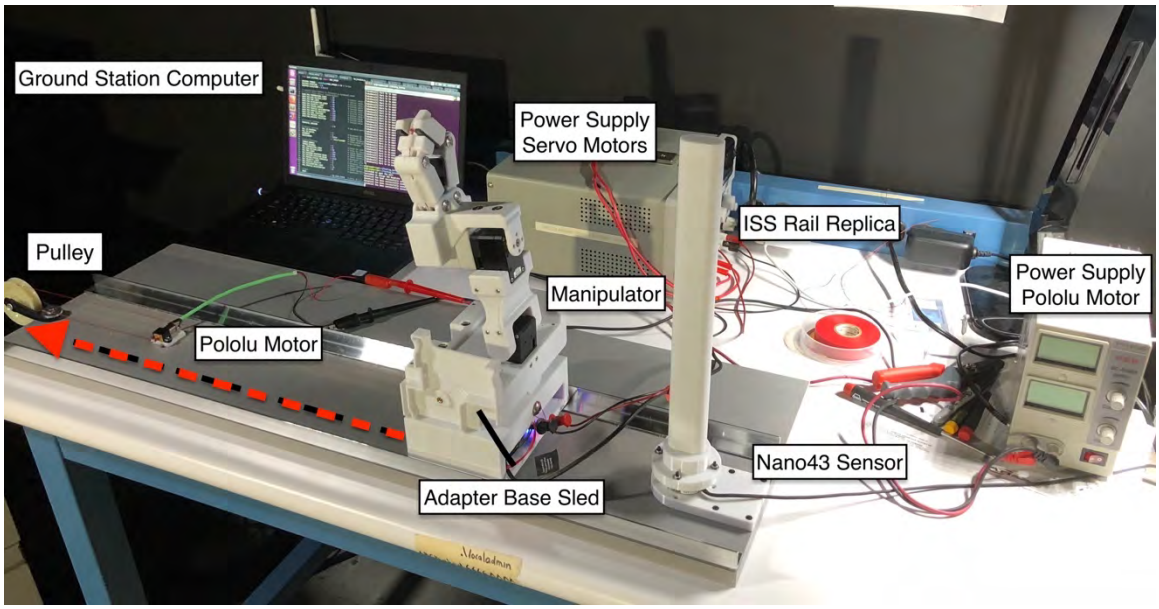

Figure 42. Actual Test Bed

The red arrow shows the direction of travel of the adapter base sled towards the pulley, (see Figure 42). After the data is collected through the sensor, it is processed at the data collection computer.

C. PROCEDURE

1. Setting Up and Testing the Test Bed

Before the test bed can be energized, confirm that the power supplies are outputting the required voltage and current so no damage may happen to the equipment or test bed. The servo power supply should be set to 12 V at 1.3 A, while the gripper motor supply is set to 2 V at 150 mA. Once the power levels have been set and confirmed, ensure that the power supplies power on the required devices.

To begin the experiment, the Ground Station Computer which contains the python script to control the manipulator and communicates to the Raspberry Pi must be turn on. A terminal window is used to remotely login into Raspberry Pi via the Wi-Fi router. A static IP address is assigned in the router settings. For this test, 192.168.0.105 was assigned to the Raspberry Pi. Once logged on as root into the Raspberry Pi the python file needs to be loaded. The following command was used to load the file:

scp test_90.py pi@192.168.0.105

To run the test, the file test_90.py was developed and used, (see Appendix A. Python Code). Once logged into the Raspberry Pi and the script is loaded into memory all that is needed is to run the script. In Linux, to run the python script the following command was used:

sudo python test_90.py

The code opens the gripper and places it on the rail and then closes the gripper throughout the test for 30 seconds. Once the timer has completed the gripper opens and return to home position when complete.

The manipulator and gripper can now be commanded via the onboard Raspberry Pi and capture data from the sensor. The data is saved as a .mat file for future analysis.

2. Data Collection

Once the manipulator has been set up, the test bed is ready to begin collecting data, (see Appendix B. MATLAB Code). The first step in collecting samples is to run the MATLAB file named:

Test_loading_Justin.m

Once the file is run, the bias will be set, a read out will display “Bias Set.” Which is seen in the command window. The commanded window will display “Paused waiting for manipulator…10 seconds.” This is to provide time to move to the ground station computer and run the python script:

sudo python test_90.py

This script moved the manipulator and gripper into place for the data collections. Next, the power supply of the gripper motor should be prepared to be energized. After the “Reading forces…” is displayed in the MATLAB command window. When the sensor is reading, energize the gripper motor to pull the gripper from the rail. The removal of the gripper takes approximately 12 seconds and the sensor recorded data for 15 seconds to capture all the data for that test sample.

D. FORCE RESULTS

Every data sample collected contains 1200 points of data over all 6 axes of the sensor. The first three columns of data are the forces F_x, F_y, and F_z. The next three columns are the torques T_x, T_y, and T_z. There were a total of 15 samples collected and analyzed. The individual results (see Appendix C. Linear Test Results). The 15 samples were plotted together and the mean of those samples was found, (see Figure 43). The individual samples can be seen in gray with the mean in black. To remove the disparity of the samples not always being started and sampled at the exact same time, the MATLAB code shifts all the data to the first point where the force goes above 0.2 N. This lined up the data to provide a much more expressive mean line.

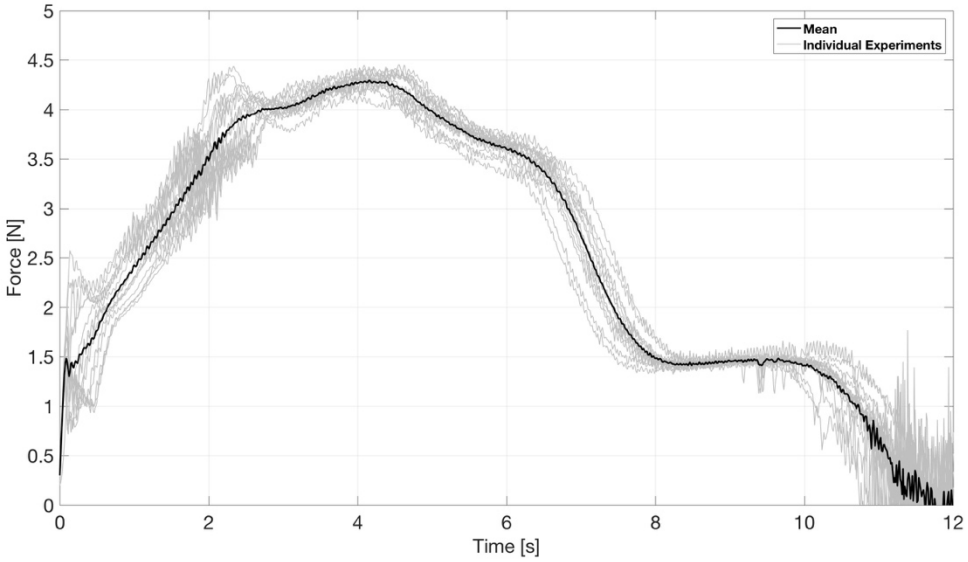

Figure 43. 15 Samples on Linear Test Bed

(From Figure 43), it can be seen when following the mean line from 0 - 2.3 seconds that the force increases in a quasi-linear way. During this time, the gripper is building force as it is pulled from the rail. The proximal and distal gripper links are pulled open increasing the torque on the torsion springs in those joints. There is also contact dynamics from the

foam pads on all four surfaces of the gripper's links. One pad for each of the two proximal links and one on each of the two distal links, (see Figure 44).

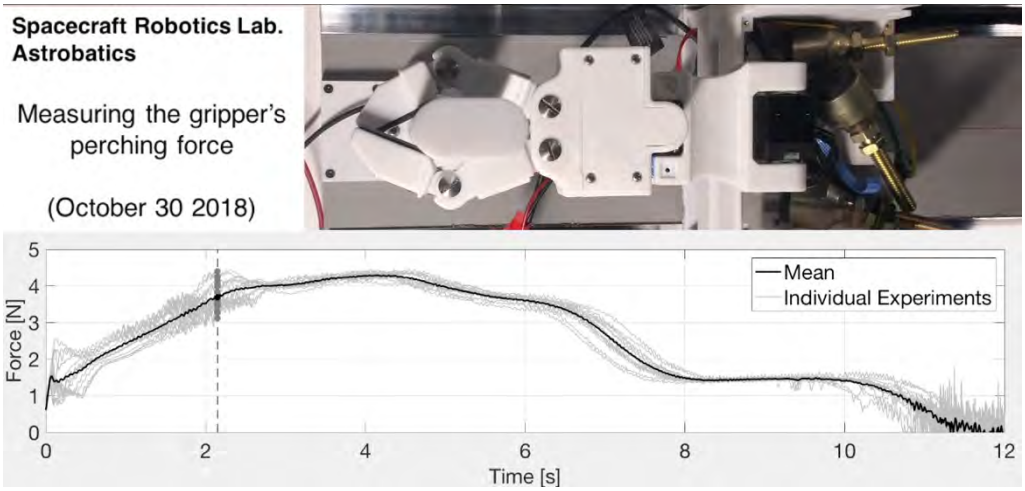

Figure 44. Gripper Building Force Linearly

During the testing, from 2.3 - 4.3 seconds, the springs and foam pads exert a maximum combined force on the rail as the proximal link begins to side off at 4.295 N, (see Figure 45) for max force.

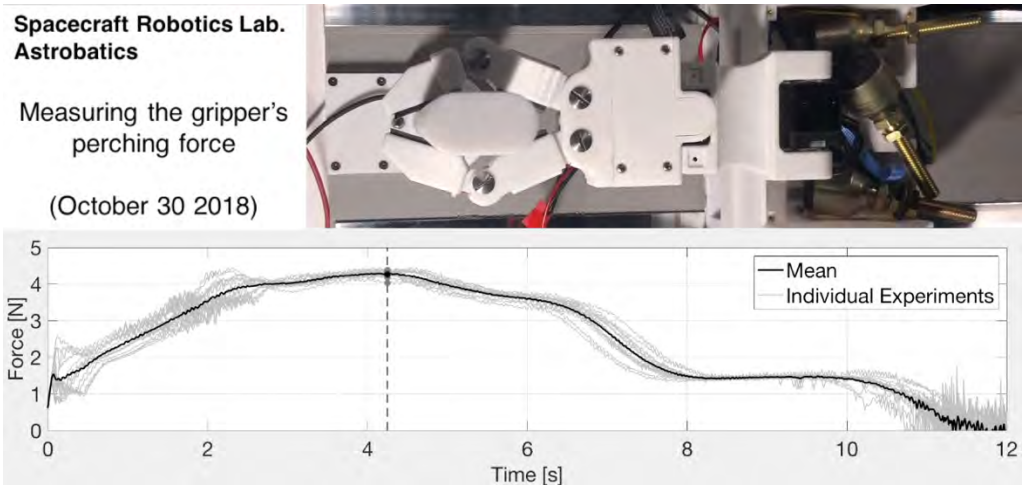

Figure 45. Gripper Max Force Achieved

The testing from 4.3 - 6.3 seconds, the proximal link pads are sliding off until they are removed at 6.3 seconds, (see Figure 46) for the proximal pads release.

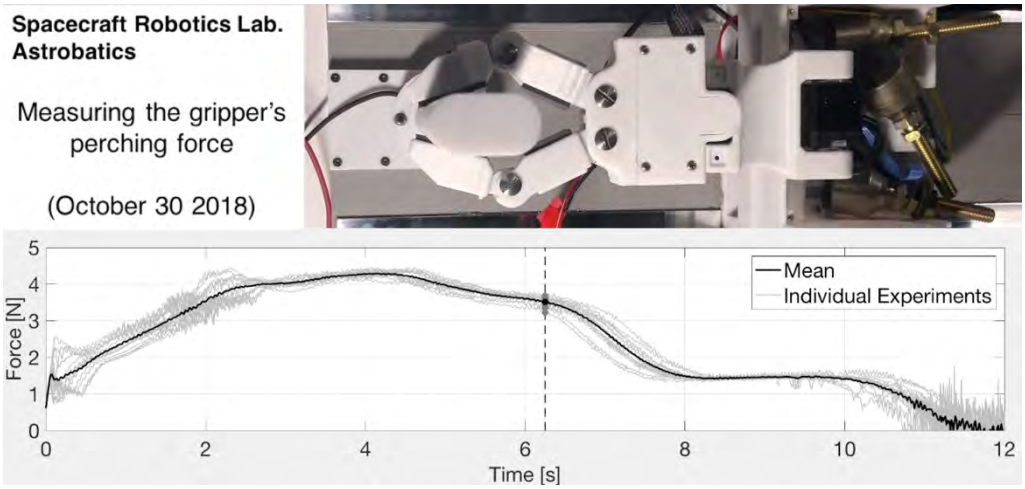

Figure 46. Gripper Proximal Pads Are Removed

At this time, the force rapidly drops to 1.5 N and the distal pads are the only thing in contact with the rail from 6.3–8 seconds. At first, the entire surface area of the distal link is felt and decreases rapidly as the surface area in contact with the rail is reduced, (see Figure 47) for the transition from distal link pad to distal link fingertip.

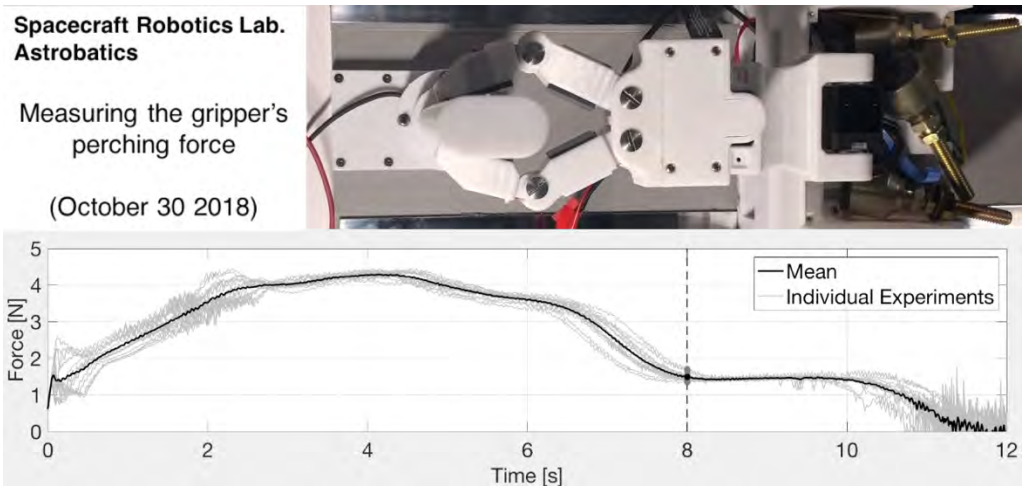

Figure 47. Gripper Distal Contact

The testing from 8 -10 seconds once the gripper transitioned to a distal link fingertip grip it holds a constant 1.5 N, (see Figure 48) for the fingertip release near the end of holding the rail.

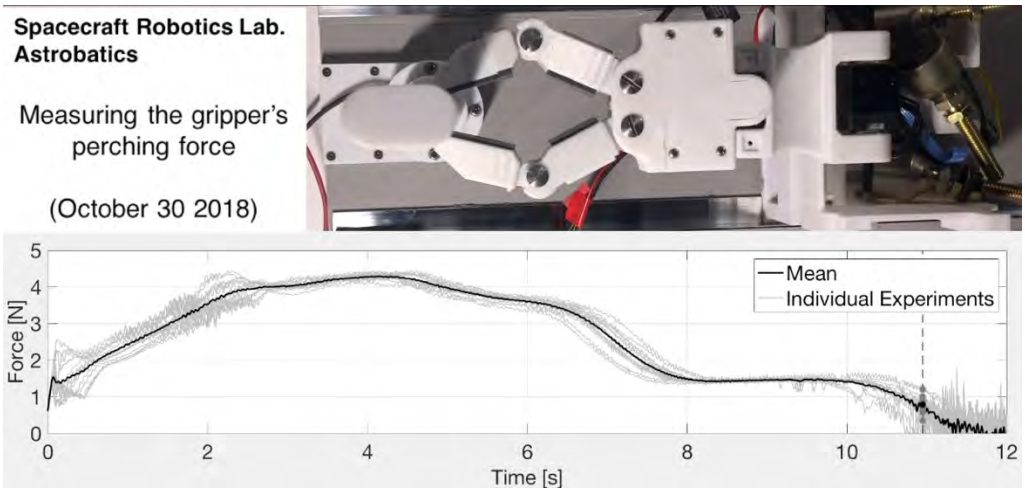

Figure 48. Gripper Fingertip Grip Final

The testing after 10 seconds the single distal link begins to slip off the rail followed by the double distal link at 11 seconds, (see Figure 49) for the single distal release.

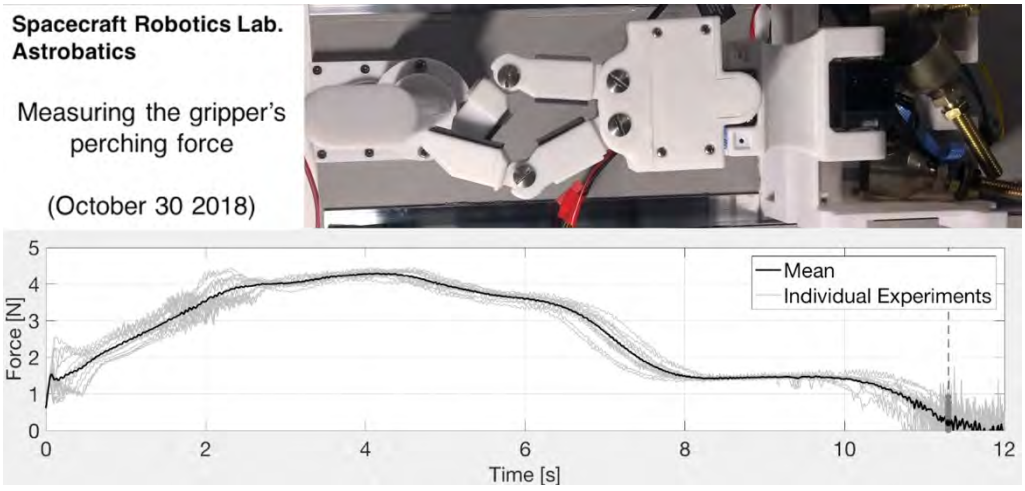

Figure 49. Gripper Single Distal Link Release

The vibrations that are seen beyond 10 seconds are from the double distal link remaining in contact for a fraction of a second longer than the single distal link, (see Figure 50).

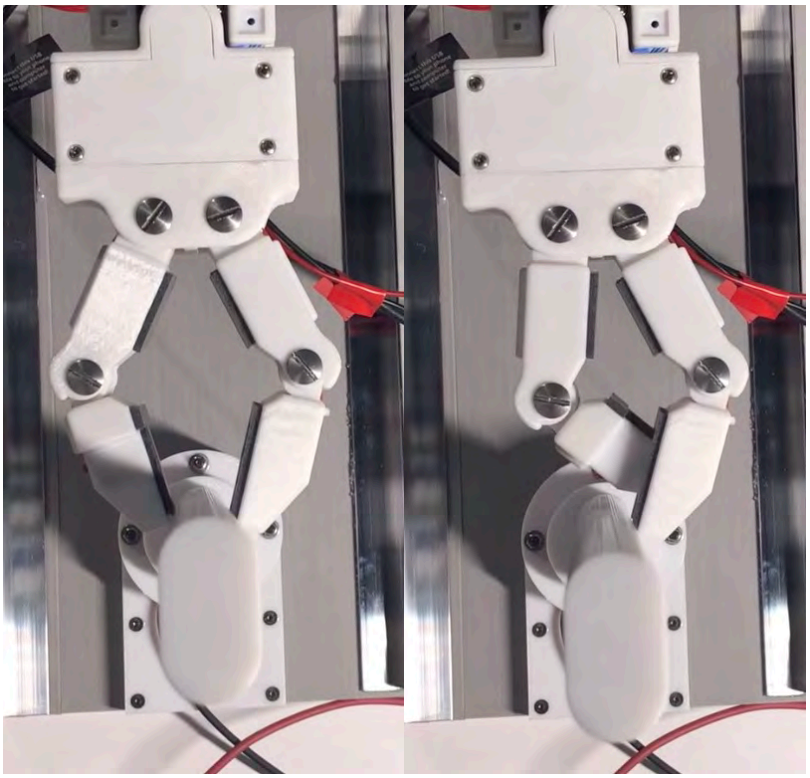

Figure 50. Gripper with Distal Links in Contact with the Rail

E. CONCLUSIONS

The linear testing is how the forces were determined that were needed to release the gripper from the handrail. The question was answered, “What are the max forces that can be applied before the gripper can no longer hold onto a rail to which it is perched to?” Through the analysis, it was found that the maximum mean grip force that can be imposed to the rail is 4.295 N. The maximum force felt from all 15 sample experiments was 4.453 N. To perform the propellantless maneuver an assumption of 0.5 N was made and applied to the simulations [35]. Alsup’s assumptions were conservative and her maneuver has an order of magnitude larger capability. Further analysis should be conducted on an actual

ISS rail. The 3D printed rail in the test may provide different release forces than what will be felt on the actual ISS handrails.

A secondary observation from this experiment is that the distal fingers do not release at the same time when removed from the rail. This induced vibrations into the rail when they sequentially slide off the rail, first the single distal link followed by the double distal link. Originally, it was thought that both of the distal links would act in a mirrored structure to the rail. Even though this data point may help other research, it does not affect the planed propellantless maneuver, since the gripper will self-toss away from the rail. The timing of the gripper's links opening and clearing the rail are not a concern as the self-toss maneuver is demonstration in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SELF-TOSS TESTING

A. OVERVIEW

The chapter answers the research question, “Can a clean release from the rail be achieved during a self-tossing maneuver of Astrobee?” The timing to open the gripper is crucial. For the propellantless maneuver to be successful it must be determined if the gripper opens at the expected time. For the self-toss to be successful, it must be conducted without interference from the gripper’s links. Any contact on the rail after the release has been made will induce unpredicted results. Testing the release of the gripper required a test bed. Utilizing the POSEIDYN test bed and the FSS is how the experiments are performed, (see Figure 51). The manipulator was mounted to the FSS and the ISS rail was mounted to the test bed. Code was written to have the manipulator toss the FSS in a planar motion so that the manipulator would be in a simulated zero gravity environment. A procedure of how to set up the test bed and how the data was collected is covered in this chapter.

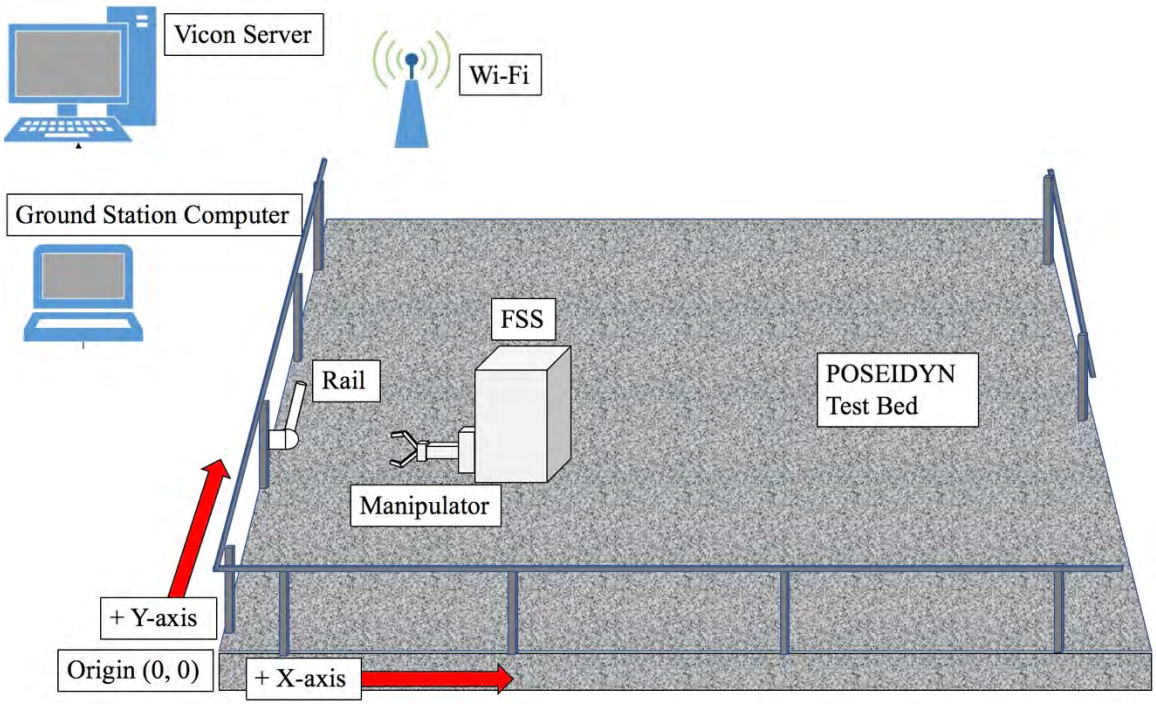

Figure 51. POSEIDYN Test Bed

This chapter will also answer, “Is the maneuver repeatable?” How well can the manipulator recreate the intended motion is of importance. When a maneuver is commanded, it will be expected to release at a precise time and self-toss in a predetermined direction. Also, while conducting this experiment some other interesting results about the gripper were found and are discussed in this chapter.

B. SELF-TOSS EXPERIMENTS

1. Hardware

The SRL group at Professor Romano’s lab has developed the POSEIDYN test bed that was discussed in chapter one. This simulator can simulate a planar motion that represents the weightlessness condition that would be felt on Astrobee throughout the maneuver. The manipulator must be mounted parallel to the granite floor so that the intended maneuver can be replicated as close as possible. There are three FSS in the SRL. FSS # 2 was selected to adapt the manipulator to, (see Figure 52). To do this the adapter needed to hold the manipulator, the motor drivers and the Raspberry Pi. Also, power would need to be provided from the FSS #2. The power requirements to run the servos and gripper motor are 12 V at 1.3 A and 5 V USB to power the Raspberry Pi.

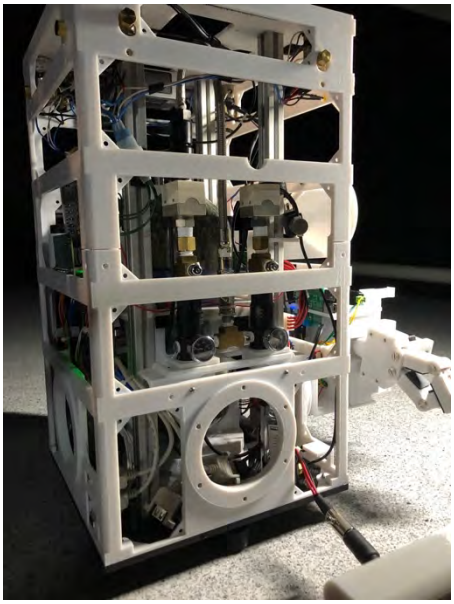

Figure 52. SRL FSS

2. Integration with Fourth Generation Floating Spacecraft Simulator

The FSS has been utilized for past experiments and currently has a mounting location to mount the current manipulator, (see Figure 52) and (see Figure 53). At the bottom edge of the FSS, a circular mounting location can be seen which is similar to the mounting location of the manipulator. The adapter needed to match the mounting holes that were available on the spacecraft.

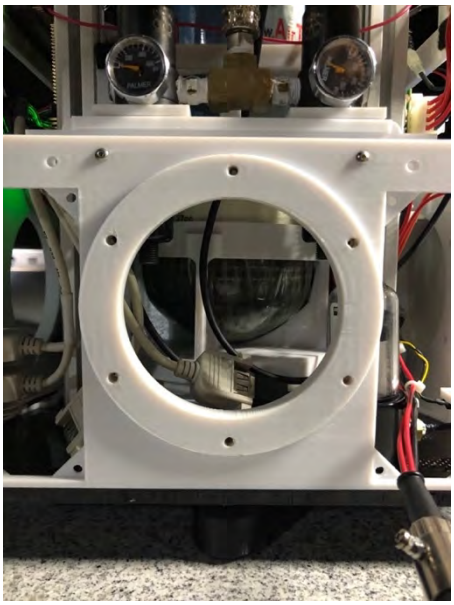

Figure 53. FSS Adapter Location

One previous Master's student [21], designed an adapter for his manipulator to mount to the 4th generation FSS, (see Figure 54). This is one adapter with two views from the left and from the right. The plate has 5 screw holes that allow the adapter to be firmly attached to the FSS. This adapter provided the ability to mount another adapter to it by sliding it over the gray portion of the figure.

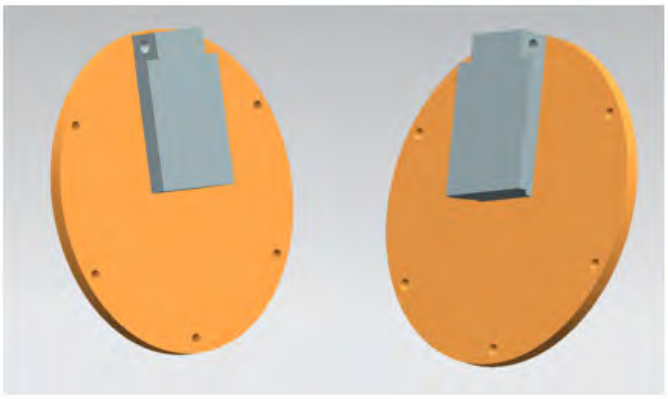

Figure 54. Previous FSS Adapter Plate

The manipulator side adapter was modified to hold the current smaller manipulator assembly. The design was a slide on style that would make it easy to attach and remove the assembly in the future. The sliding connection point was too close in tolerance to be removed after it was place on the mount. To make the mount easier to remove a Dremel was used to remove some of the 3D material. The adapter holds the manipulator assembly and can easily be removed for troubleshooting and future adaptability. (See Figure 55) for the new adapter to the FSS. The light blue portion, (see Figure 55) slides over the gray portion, (see Figure 54). As seen in the previous figures, this is one adapter with two views. The adapter has two separate compartments. The closest compartment to the adapter slide plate is where the Raspberry Pi assembly is placed. The open compartment farthest from the adapter slide plate is where the manipulator is mounted.

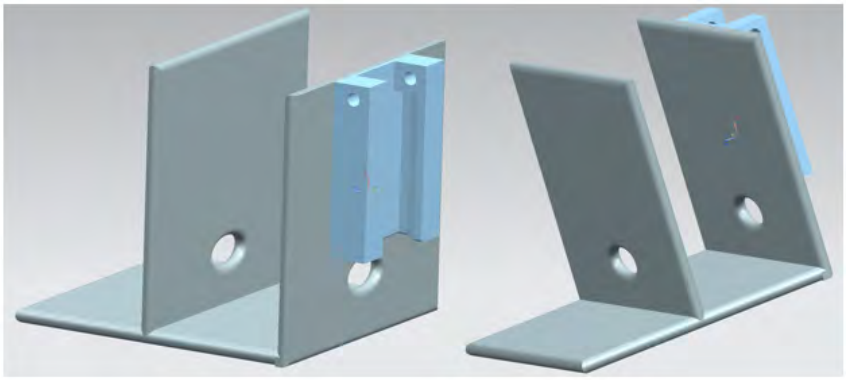

Figure 55. Manipulator Assembly Adapter to FSS

The adapter is designed to hold the manipulator out away from the FSS to allow a minimum unobstructed 180 degrees of motions. The manipulator needed to have freedom of movement in both the XY plane and the distal joint servo needed to also have a minimum unobstructed 180 degrees of motion. This led to the final solution, (see Figure 56). This figure displays the mounting from the left and right sides of the manipulator. The only requirement the manipulator needed from the FSS is the 5 V USB input and 12 V to power the servos and gripper motor. Luckily the FSS already had both of these power sources available and only a harness needed to be made to connect the 12 V power supply to the manipulator.

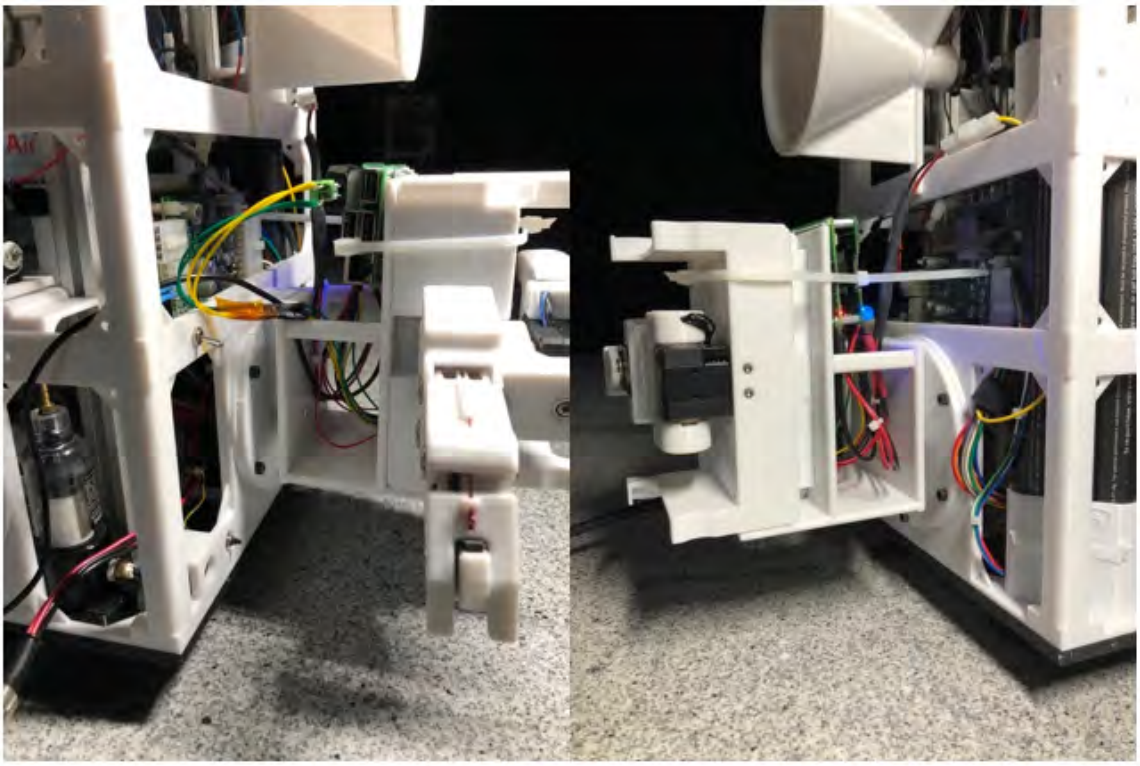

Figure 56. Manipulator Mounted to FSS

3. Rail Integration with POSEIDYN

The next step is to mount ISS rail replica to the POSEIDYN test bed. The rail needs to be mounted parallel to the granite table so that the maneuver simulates the environment

that Astrobee will be in while on the ISS. The adapter to mount to the linear test bed was originally design to meet the mounting needs of both the linear test bed and POSEIDYN test bed, (see Figure 57). The rail is mounted to the test bed with four M3 x 40 mm bolts. To reduce the chance of the rail loosing throughout the testing a large M4 x 50 mm bolt with lock washers and a nut were placed through the center of the mount.

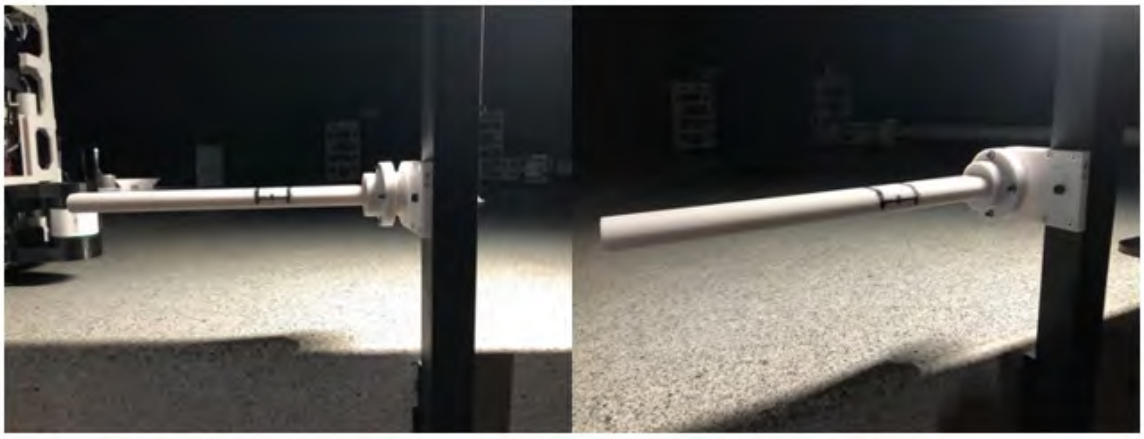

Figure 57. ISS Rail Mounting Location

4. Vicon Motion Capture

Once the manipulator has been mounted onto the FSS and the rail has been mounted to the POSEIDYN test bed, a new positioning system was utilized to track the position of the FSS on the test bed. The motion caption system in called Vicon and it uses 10 infrared cameras, a server, and proprietary software to activate the system to sense the location of small reflective spheres mounted to the FSS. The infrared cameras sense the light that reflects off the spherical markers and sends this telemetry data back to the Vicon server to be processed and to determine precise position and orientation of the FSS, (see Figure 58)

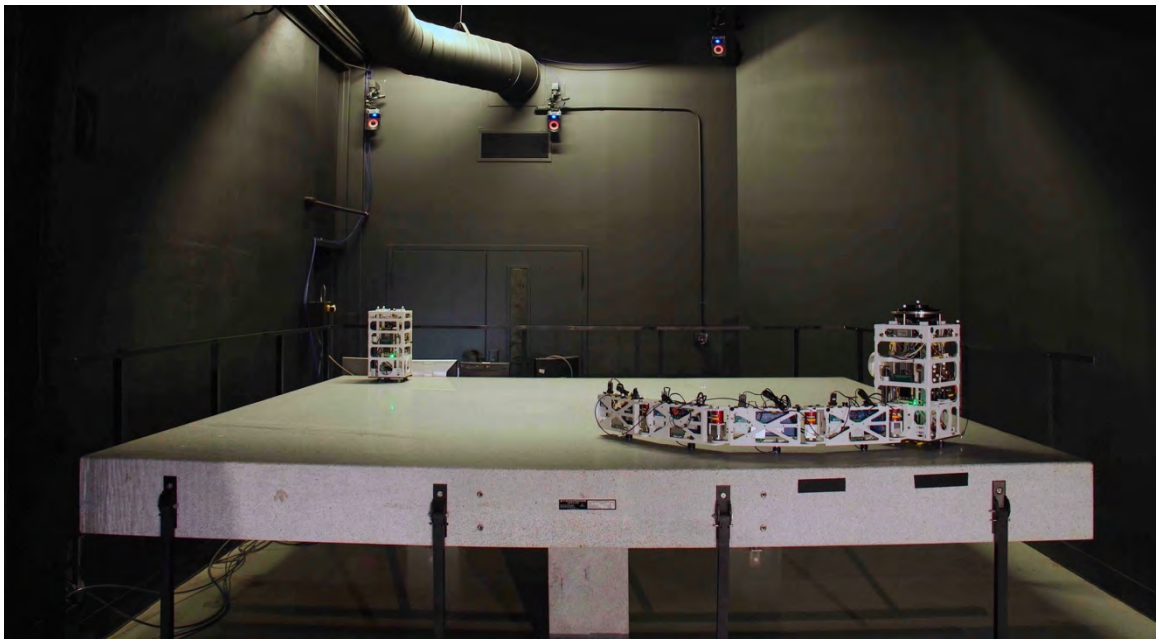

Figure 58. Vicon and POSEIDYN Test Bed. Source: [20].

C. PROCEDURE

1. Setting Up the Test Bed

Once the previous hardware section adapters have been mounted the FSS is ready to test its functionality. Before beginning any operational testing, the FSS have their own functional testing that needs to be performed. Each FSS has a large tank that can hold up 20,684 kPa (3000 psi) of compressed air, to maintain the FSS simulated weightlessness on the granite table. The air tank provides air pressure to three small air bearings that slowly allow the air to escape causing an air cushioning affect that allows the FSS to hover. The tanks need to be at or near the 20,684 kPa (3000 psi) before initializing any equipment. Next, the FSS has two large battery packs and a power conditioning and distribution system that provide the 24V needed to power the FSS hardware. There is an umbilical cord that slowly charges the battery packs to ensure they are at maximum capacity. The unit needs to be fully charged before beginning any operational testing.

After the FSS's functionality is validated, power can be supplied to the FSS by pressing the small button on the lower portion of the FSS. A valve is open to allow air from

the tank to enter the air bearing system. Finally, the FSS air bearing switch is activated to energize the air bearing system, (see Figure 59).

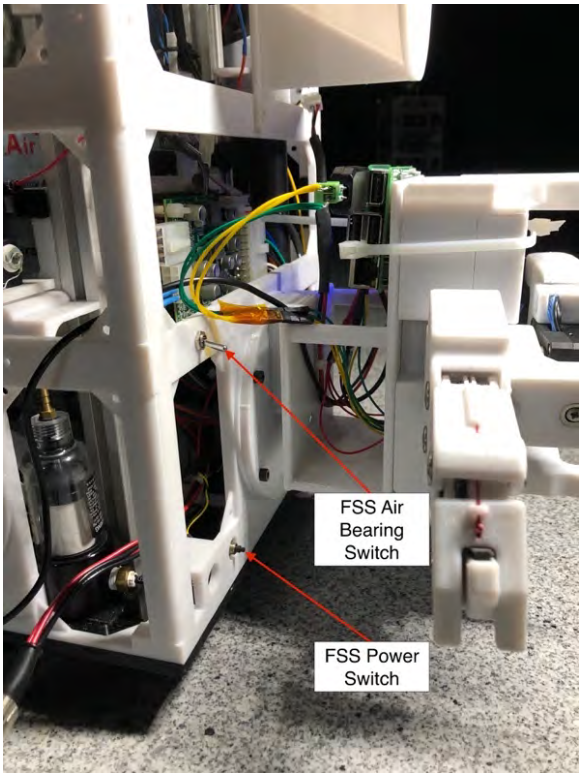

Figure 59. FSS Power and Air Bearing Switches

At this point the FSS is on and floating. As in the previous linear force tests, it is required to remote login to the Raspberry Pi to command and up load new python scripts to control the manipulator. The new script that has been created is called self_toss.py. This script commanded the manipulator to rock itself back to a set angle and then pull itself back to a set angle and release. This simulated the self-toss motion that will be utilize in the ISS testing. The angles that were chosen through many iterations that provided a good resemblance to the self-toss simulation conducted by Master's student [28] were 80 degrees and 20 degrees. Due to the fact that the mass of the FSS was not the same as what was expected to the actual Astrobee these numbers should be taken as only examples and not applied as actual possible angles of release for the future ISS testing.

Originally, the forces that were induced while preforming this maneuver were going to be sampled and analyzed but it was during this phase of functionality testing of the equipment that the nano43 sensor was found to be providing faulty data. The sensor when reset with a zero bias would continue to read forces that were not being felt at the sensor. Complete removal of the sensor and extensive troubleshooting led to the conclusion that the G4 resistance is not within tolerance and the sensor needs to be sent to be recalibrated, (see Figure 60). The sensor recalibration is a lengthy process and a new method to analyze the motion of the manipulator needed to be devised.

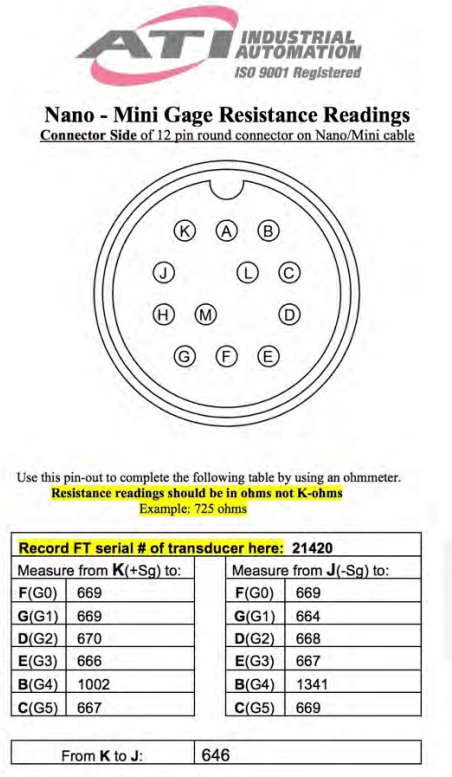

Figure 60. ATI Sensor Analysis. Source: [34].

The load cell was no longer available. The forces on the rail may be important but, the best way to prove that the maneuver would be able to be performed would be a demonstration. A deeper look into the position of the manipulator relative to the FSS may provide more useful data to analysis and come to a determination of the feasibility of the

ISS self-toss maneuver. To do this, a new method of data transfer was needed to transfer the positioning and orientation information from the Vicon and to the servos on the manipulator to the ground station computer, (see Figure 61) for and overview of the POSEIDYN test bed.

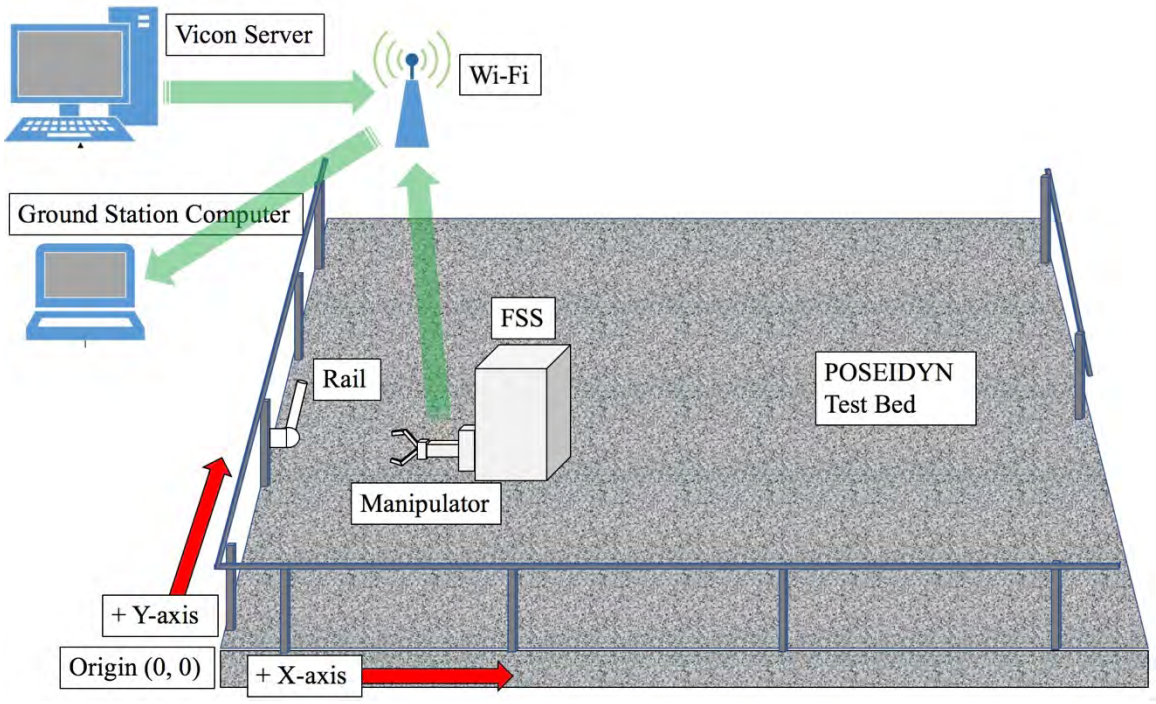

Figure 61. POSEIDYN Test Bed

User Datagram Protocol (UDP) communication protocol was the chosen solution to this problem, the green arrows represent the communication paths, (see Figure 61). A new script needed to be created for the Raspberry Pi to transmit UDP data about the position of its servos via the Wi-Fi to the ground station computer. Using the previously developed self-toss code the UDP transmit was added to report its current servo position of both joints 1 and joints 2. UDP is a method to transmit communication data with messages, called datagrams, that are sent from the transmitting user to destination IP address and port number. This method is suitable to the small environment that is being utilize and not much interference was involved in this situation. Interference could cause loss of packet data and which creates data loss for the analysis. The NPS Vicon system is

already setup to send data in this method. All that was required was to set the new IP address and port needed to receive the telemetry data correlating to the FSS. Both the Vicon and the Raspberry Pi where setup to transmit UDP data via Wi-Fi to the ground station computer at 192.168.0.104. The Raspberry Pi utilizes port 25010 to transmit data and the Vicon utilized port 25020. A MATLAB script was developed to modified and capture the UDP data. The script was designed to receive the angles of joint 1 and joint 2 from the Raspberry Pi as well as the FSS orientation and position in the XY plane. Data is received with the FSS's exact position and orientation while the servos provide the ordered position of each joint at the same time.

Once the FSS is functional and the Raspberry Pi and Vicon systems are reporting their data, the latest version of the self-toss python program needs to be loaded to the Raspberry Pi with the following command from root:

scp self_toss.py pi@192.168.0.105

The Raspberry Pi is ready to start the self-toss script. To run the script the following command was entered:

sudo python self_toss.py

The manipulator begins to open and move to start angle of 80 degrees. Once the servo reads 80 degrees, the gripper closes onto the rail at the predetermined launch location, (see Figure 62). The gripper is place perpendicular to the rail between the two black lines. This allows for a common launch location to be used throughout the testing.

Figure 62. Gripper Starting Location

Once the gripper was place on the rail a waiting period of 30 seconds was given to the system to stabilize. The manipulator can allow the FSS to twist back and forth while attached to the rail. The 30 seconds was provided to securely fasten the manipulator to the rail and remove any unwanted motion it may have before retrieving data from the test. This was an indication, that there is a bit of play or that the grasp is not very firm. This then manifest itself in terms of a slip angle during the self-toss maneuver. The gripper would allow the FSS to rock back and forth even while the gripper is closed. This was an unknown phenomenon of the gripper that needed to be analyzed. Before the applied time, the gripper was considered to be fixed once on the rail but instead the gripper had a slip angle that was consistently seen before motion was perceived in the FSS. The slip angle has become a new data point that must be captured.

Once the self-toss command is entered and the gripper is applied to the chosen location on the rail the manipulator stabilized the spacecraft during the 30 second waiting period. The manipulator begins the self-toss maneuver by pulling the spacecraft forward until joint 1 reads the set release angle. 20 degrees was set to be the release angle. The gripper is commanded to open releasing the spacecraft from the rail. The commanded motion launches the spacecraft to the right of the rail. To launch the spacecraft to the left, the code would need to be changed. The start angle would need to be set to -80 degrees and the release angle set to -20 degrees. While the manipulator is in the wait phase of 30 seconds and the FSS is stabilized the MATLAB program can be ran to receive the UDP data from both the Raspberry Pi and the Vicon. The Vicon system has a MATLAB program that needs to only be ran one time. The program continued to send updated telemetry until it is stopped. The data was be saved as a .mat file on the ground station computer to be analyzed later. Once all of these steps are completed consistently, data can be collected.

2. Data Collection

Once the FSS is functional and the Raspberry Pi is updated with the latest self-toss script the collection process begins. First, Vicon must be transmitting UDP telemetry data to the ground station computer. Second, command the self-toss program to begin from the ground station computer. Once the manipulator is applied to the correct location on the rail

and has stabilized, the MATLAB program on the Ground Station Computer can be executed to begin collecting data from the UDP data ports. A total of 20 individual sample runs were conducted on the manipulator. 10 samples were of the manipulator launching the FSS to the left and 10 samples were of the manipulator launching the FSS to the right.

The data collection process for these samples was lengthy. The motion of the maneuver is very slow. To recreate the SRL ISS hopping maneuver. The motion duration coupled with the 30-second waiting period would take each sample approximately two and a half minutes to gather the data. Resetting the test bed would take another two minutes. Thus, it would only allow for approximately 5 samples to be taken until the tank needed to be refilled to allow consistent results to be taken. As the tank pressure drops, the pressure applied to the air bearings decreases. This slowly allows more resistance to be felt by the FSS against the granite surface. To maintain consistent data, it is recommended not to exceed approximately 20 continuous minutes of usage. The tank slowly leak air while the air tank valve is open. To avoid unneeded loss of air, the valve needs to be closed when not actively in use. Extending the time between air pressure tank refills.

D. RELEASE RESULTS

This experiment received two separate data samples via UDP to the Ground Station Computer. Every sample collected contained 1635 points of data in respect to the position of joint 1 and position of joint 2 form the Raspberry Pi. Also, 1635 points of data in respect to position in X, position in Y, and orientation from Vicon. The data was correlated and truncated to focus on the release of the manipulator. Due to the lengthy sampling time, 10 self-toss samples were collected in each direction from the rail. To record the results of the manipulator closing, the command motion of joint 1 was -80 degrees for the start angle and -20 degrees for the release angle, generating a self-toss to the left of the rail. To record the results of the manipulator opening, the command motion of joint 1 was 80 degrees for the start angle and 20 degrees for the release angle, generating a self-toss to the right of the rail.

There were a total of 20 sample experiments collected and analyzed. The individual results can be seen in (Appendix D. Self-Toss Results). The 10 samples were plotted

together and the mean of those samples was found, (see Figure 63). The individual samples can be seen in gray with the mean in black. These are the results found when preforming the self-toss with the manipulator opening launching the FSS to the right. The manipulator remains at 80 degrees until commanded at time = 0 seconds, which is the first vertical dashed line, to begin the self-toss maneuver. The joint begins to transition to the command 20-degree position which takes a mean time of 5.5 seconds to complete, displayed by the second vertical dashed line. The gripper opens and the FSS is released from the rail. The code continues to move the manipulator to avoid possible collisions with the rail and subsequently returns to the command 20-degree position and remains there until the completion of the sample at 20 seconds. These results are as commanded and anticipated.

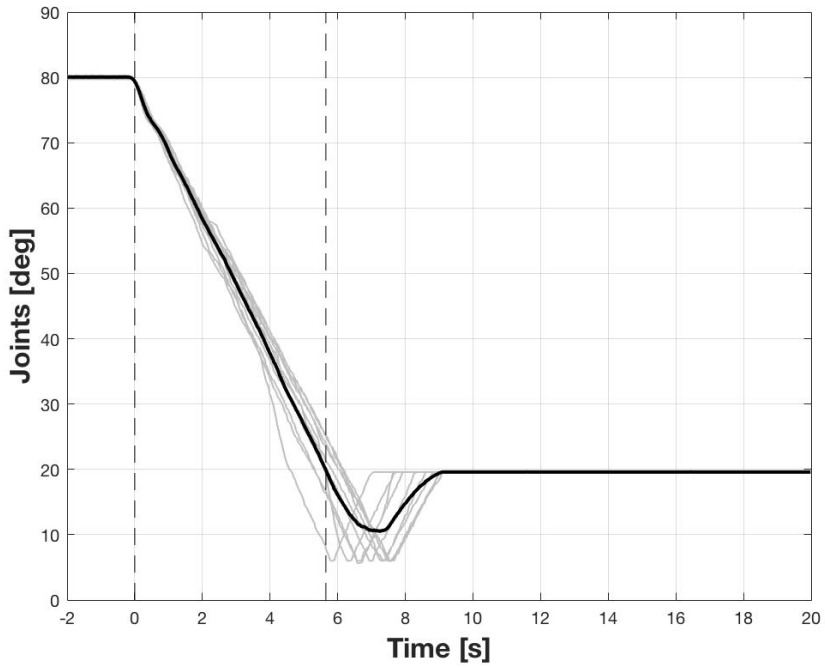

Figure 63. Joints Position over Time, Manipulator Opening

The position of the FSS was analyzed to see if the maneuver was repeatable, (see Figure 64). The POSEIDYN granite table is 4 m x 4 m. The testing of the self-toss only used a small portion of that table available. The origin of the reference frame for POSEIDYN is the bottom left corner, (see Figure 64) and (see Figure 65). The FSS started

near the center of the left side of the table near the edge. The maneuver was small as it moved approximately 0.6 m the -Y direction and 0.5 m in the +X direction. The maneuver is the self-toss to the right. The + represents the starting position of the FSS and the ° represents the point of release. Once the FSS is released, it continues on a linear path.

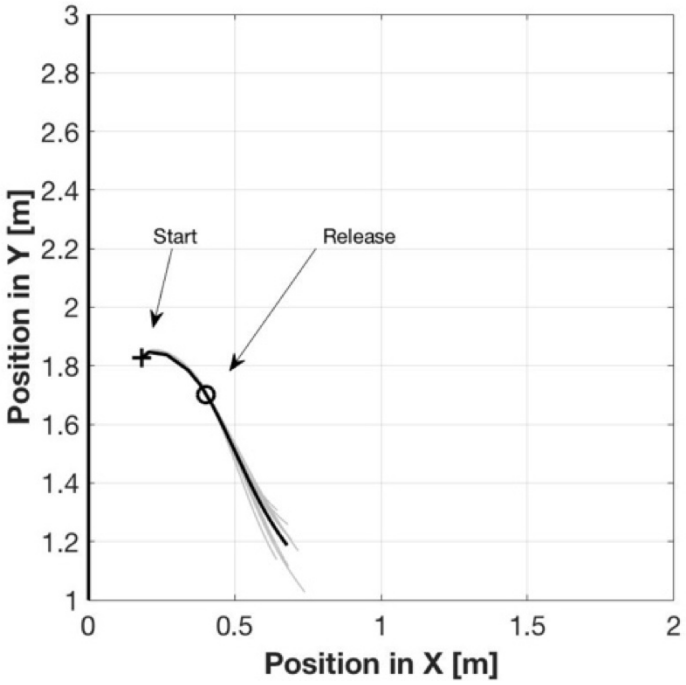

Figure 64. Self-Toss Right (Opening) of FSS on POSEIDYN

When the maneuver was preformed to the left or closing the manipulator the FSS was tossed a larger distance. The maneuver was larger as it moved approximately 1.2 m the +Y direction and 0.5 m in the +X direction, (see Figure 65) for self-toss to the left.

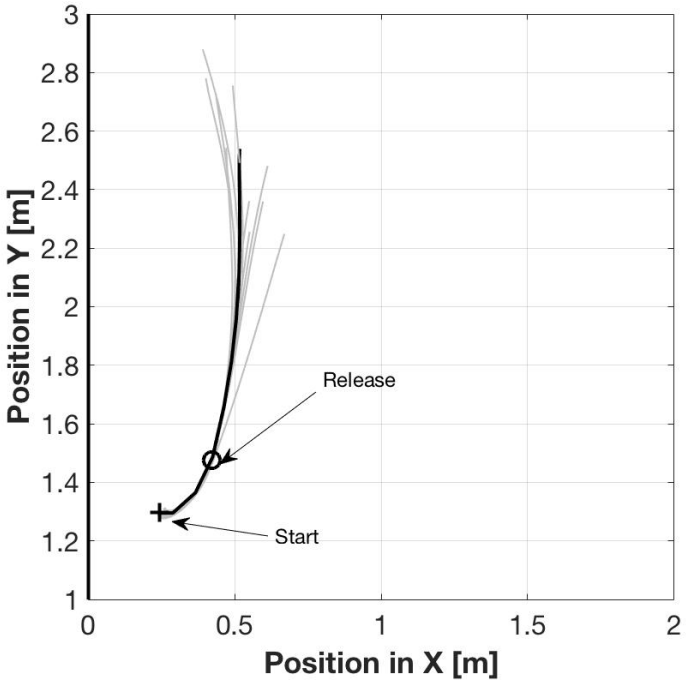

Figure 65. Self-Toss Left (Closing) of FSS on POSEIDYN

The orientation of the FSS was analyzed. As before, the 10 samples were plotted together and the mean of those samples was found, (see Figure 66). The individual samples can be seen in gray with the mean in black. The orientation was expected to change linear rate after release. The 10 samples all changed linearly but not with the same linear rate. Further analysis showed that at the beginning of the maneuver, the vertical line on the left represents the start of the maneuver, the FSS did not immediately start changing in orientation as the FSS was tossed. Adversely the joints immediately began to change in position, (see Figure 63). A new question was established, “What was causing this anomaly?”

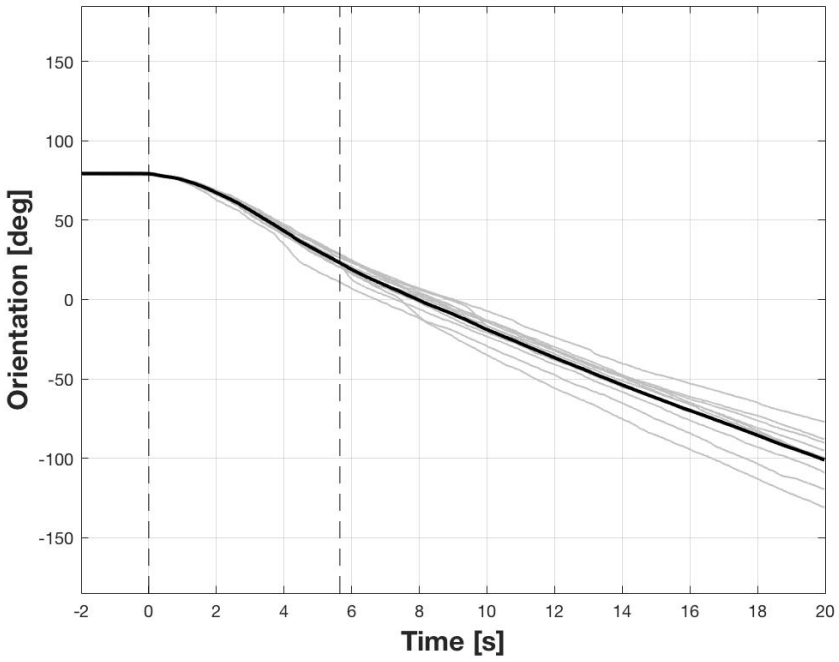

Figure 66. Orientation of FSS over Time, Manipulator Opening

After reviewing the videos and the data, it was noticed that the gripper slips before it creates a force imposed to the rail which starts the motion in the FSS. (See Figure 67), for photos that characterize the maneuver over time. Starting on row one column one [1, 1] the manipulator is seen at the start angle of -80 degrees. The next photo row 1 column 2, [1, 2] displays the slip angle. The slip angle is discussed more later. The next row starting [2, 1] shows the maneuver moving the FSS to the left. In the same row, [2, 2] the manipulator has now reached the command position or -20 degrees and the distal link in the gripper are still attached to the rail. In the next row, [3, 1] is the moment of release. The distal link in the gripper is now open. In the same row, [3, 2] the FSS is free-floating as it leaves the rail. In the next row, [4, 1] the FSS is one a linear path away from the rail. In the same row, [4, 2] the FSS is clearing the data samples area.

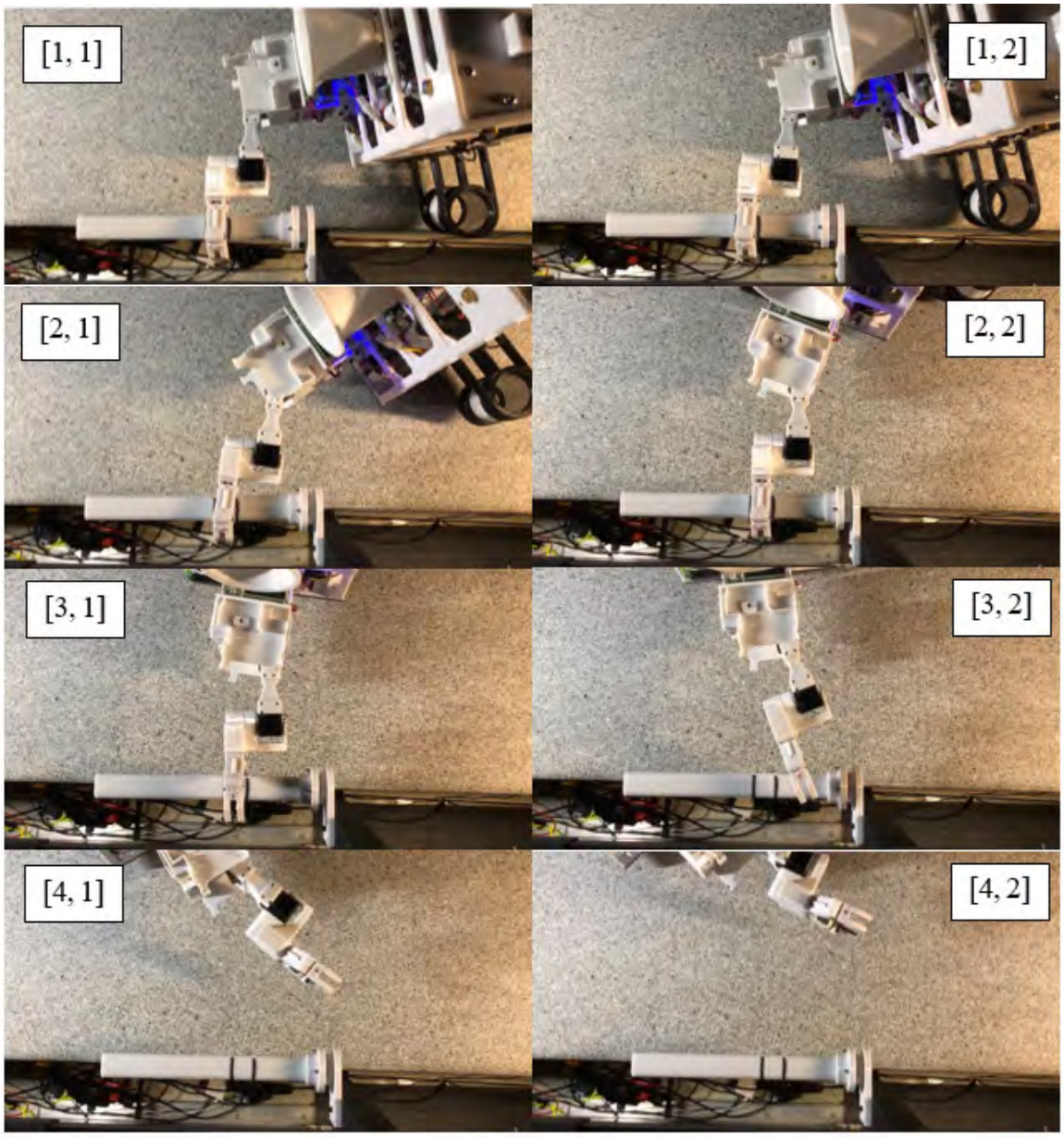

Figure 67. Release to Left, Manipulator Closing

(See Figure 68) for gripper slip angle, this is the angle at which the manipulator slips before it begins to self-toss the FSS. The maneuver starts at an angle of -80 degrees, (see Figure 68 (top)). When the manipulator is commanded to move to -20 degrees, the manipulator does move as commanded, (see Figure 68 (bottom)). Notice how the FSS has only slightly rotated and not yet moved away from the side of the POSEIDYN test bed.

(See Figure 68 (bottom)), the gripper at this time, was expected to have remained perpendicular to the rail, (see Figure 68 (top)). A slip in the gripper was observed.

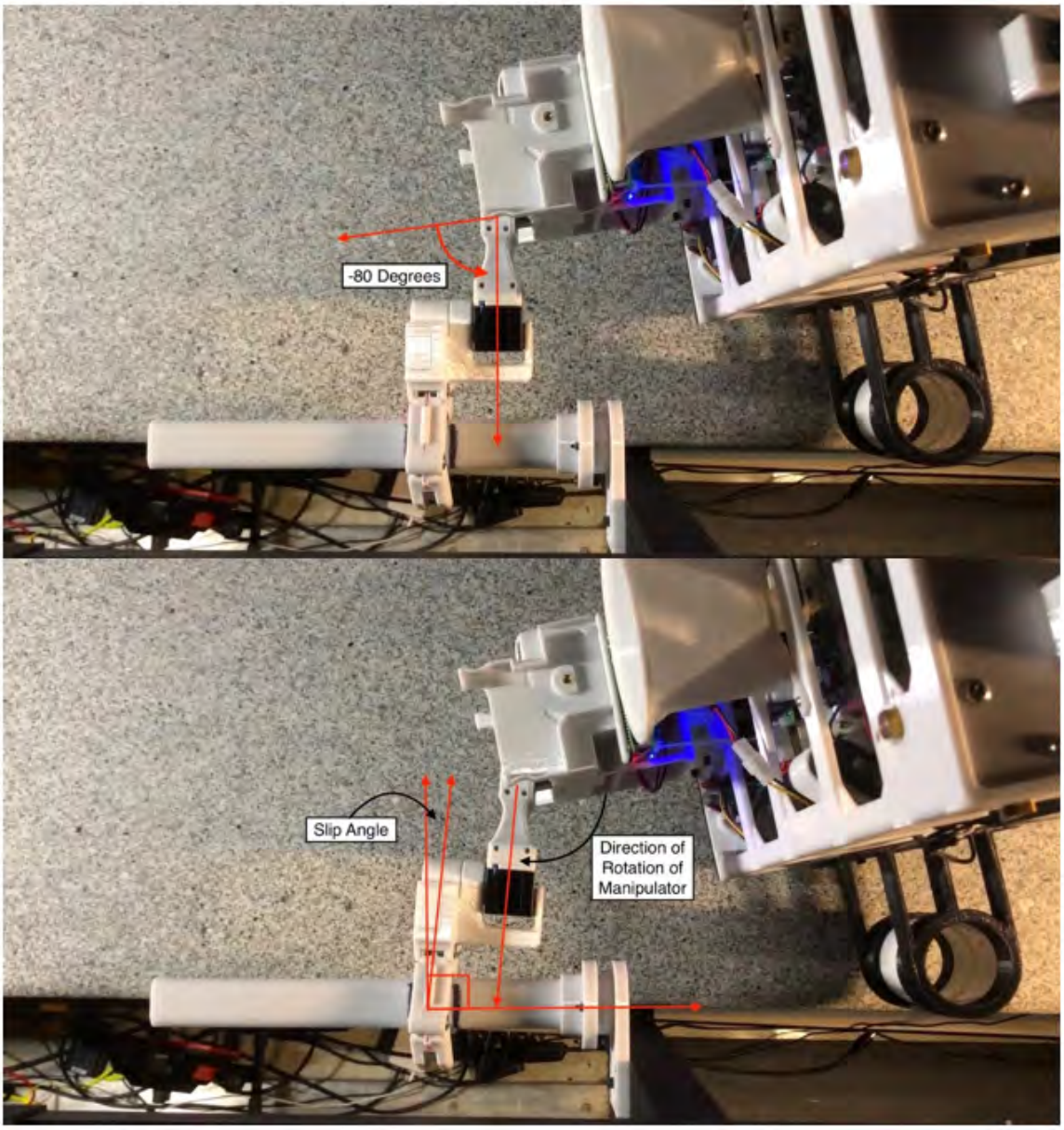

Figure 68. Gripper Slip Angle Demonstration

Deeper analysis found that the max slip angle on the opening manipulator, self-toss right, was 10.8 degrees. The max slip angle on the closing manipulator, self-toss left, was

13.1 degrees. As before the individual samples can be seen in gray with the mean in black, (see Figure 69) for slip angle results with a self-toss to the right.

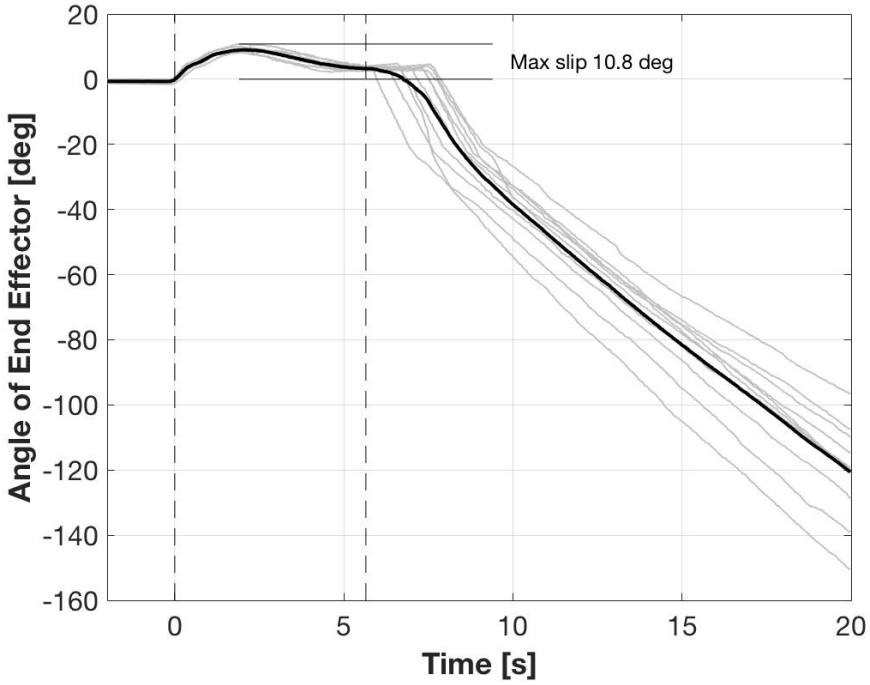

Figure 69. Gripper Slip Angle, Self-Toss Right

E. CONCLUSIONS

The second goal was to determine if the gripper would have a clean separation after the self-toss maneuver has been performed. The experiments demonstrated that the release of the gripper is clean and is not a concern for the maneuver. Not one time through all of the testing did the gripper come in contact with the rail or even the larger mounting adapter.

The maneuver was found to be repeatable only when the gripper slip angle was considered. If the gripper slip angle is not considered for the maneuver the success of the maneuver would be coincidence and not a calculated controlled maneuver.

V. CONCLUSION

A. SUMMARY OF WORK

Robotic hopping is a unique idea and can be utilized in many ways in space. Through the extension hardware development process coupled with multiple software language developments, the NPS manipulator executed a controlled robotic hopping maneuver. Through this process, original data was acquired. This thesis demonstrated that the current Astrobee manipulator can perform a propellantless maneuver by using its manipulator.

To the research question "What are the max forces that can be applied before the gripper can no longer hold onto a rail to which it is perched to?" the answer 4.453 N. The linear testbed determined what are the required forces to release the gripper from the handrail. Through the analysis, it was found that the maximum mean grip force that can be imposed to the rail is 4.295 N. The maximum force felt from all 15 samples was 4.453 N. In conducting this experiment, it was found that the distal fingers do not release at the same time when removed from the rail.

To the research question "Can a clean release from the rail be achieved during a self-tossing maneuver of Astrobee?" the answer is yes. The self-toss experiments conducted empirically demonstrate that the release of the gripper is indeed clean and not a concern for the maneuver. In conducting this experiment, it was found that the gripper has a slip angle. This maximum slip angle on the opening manipulator, self-toss right, was 10.8 degrees. The max slip angle on the closing manipulator, self-toss left, was 13.1 degrees. The conclusion was made that the maneuver is plausible and repeatable only when the gripper slip angle is considered.

B. LIST OF ACCOMPLISHMENTS

(1) NPS manipulator was designed, developed, and constructed.

Control software was developed to control the manipulator

(2) A linear test bed was designed and constructed.

Control software was developed to control the manipulator on the linear test bed.

(3) Experiments were conducted on the linear test bed.

Software was developed to collect the forces effects felt on the ISS rail. Results were collected.

(4) The NPS manipulator was adapted to the POSEIDYN test bed.

Modification to the existing test bed and FSS were made to allow the manipulator to function while on the FSS.

(5) Self-Toss experiments were conducted on POSEIDYN test bed.

Software was developed to collect UDP data from the Vicon positioning systems and UDP data from the Raspberry Pi about the position of the joints in the manipulator. Results were collected.

(6) Data was analyzed and conclusions were made.

MATLAB code was developed to plot and analyze the data from both of the experiments. Video were analyzed and data correlated to tell the story.

C. FUTURE WORK

The next step is to preform testing on the NPS manipulator on Astrobee. This would validate the results found on the NPS POSEIDYN test bed. Currently NASA Ames is building an air bearing simulator for Astrobee. An alternate power source would be needed to supply the power required to operate the manipulator. Using the control built in the thesis the manipulator could self-toss the actual Astrobee and provide data relative to the mass of Astrobee instead of the NPS FSS. This would allow practice of the intended hopping maneuver on the ISS.

Another avenue would be to acquire a new force sensor, with wireless capability, and retest the NPS manipulator on the linear test bed. Then borrow the Astrobee manipulator from NASA Ames and test the gripper on the linear test bed. This would

validate the results found from the testing of the NPS manipulator. Once the manipulator is validated then extension experiment could be conducted on POSEIDYN with two ISS rails and practice the hopping maneuver here at NPS. If the sensor had not failed this would have been the next step for this thesis.

D. RESEARCH SIGNIFICANCE

Robotic hopping is a potentially effective and efficient method of motion. Anytime an object can move without the use of propellant is adventurous. This research will lead to more extensive hopping maneuvers of manipulators for many purposes. In the ISS, man-hours are of the most expensive resource and are tremendously limited. The more workloads that robotics can remove from the astronaut's daily requirements, the more time the astronauts would have available conducting the needed human interactive tasks. This would lead to hopping maneuvers exterior to the ISS and remove astronauts from the risk of injury while providing maintenance and repairs to the ISS. The possibility of a maneuver that is beyond the limits of its propulsions system has an even larger impact to hopping maneuvers.

With the new NPS manipulator and controls available, more thesis student can explore the range of opportunity that mechatronics can provide to solving the problems that DoD, and other U.S. agencies have not yet solved.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. PYTHON CODE

```
#RA_Constants.py
from dual_mc33926_rpi import MAX_SPEED

GRIPPER_TORQUE      = int(float(MAX_SPEED)/2.0) # Torque
GRIPPER_OPENTIME    = 2.0
GRIPPER_CLOSETIME = 2.0/2.5

# Control table address is different in Dynamixel model
# Control table address (XH430-W210-R)
ADDR_PRO_TEMPERATURE_LIMIT       = 31          # Initial value 80, NASA going to 80
ADDR_PRO_MIN_VOLTAGE_LIMIT       = 34          # Initial value 95, NASA going to 100
ADDR_PRO_PWM_LIMIT               = 36          # Initial value 885, NASA going to 400
ADDR_PRO_ACCELERATION_LIMIT      = 40          # Initial value 32767, NASA going to 10
ADDR_PRO_VELOCITY_LIMIT          = 44          # Initial value 360, NASA going to 5
ADDR_PRO_TORQUE_ENABLE           = 64          # Initial value 0
ADDR_PRO_PROFILE_ACCELERATION = 108           # Initial value 0
ADDR_PRO_PROFILE_VELOCITY        = 112         # Initial value 0
ADDR_PRO_GOAL_POSITION           = 116         # Initial value -
ADDR_PRO_PRESENT_VELOCITY        = 128         # Initial value -
ADDR_PRO_PRESENT_POSITION        = 132         # Initial value -

# Protocol version
PROTOCOL_VERSION            = 2.0          # See which protocol version is used in the Dynamixel

# Default setting
DXL_ID_PROXIMAL             = 1            # Dynamixel ID : 1
DXL_ID_DISTAL               = 2            # Dynamixel ID : 2
BAUDRATE                    = 57600        # (NASA) Dynamixel default baudrate : 57600
DEVICENAME                  = '/dev/ttyUSB0'  # Check which port is being used on your controller
                # ex) Windows: "COM1"   Linux: "/dev/ttyUSB0" Mac: "/dev/tty.usbserial-*"
# Servo Settings
TORQUE_ENABLE               = 1            # Value for enabling the torque
TORQUE_DISABLE              = 0            # Value for disabling the torque
DXL_TEMPERATURE_LIMIT     = 8 # (NASA) Dynamixel temperature confirmed to default setting
DXL_MIN_VOLTAGE_LIMIT     = 100 # (NASA) Dynamixel temperature confirmed to default setting
DXL_PWM_LIMIT             = 400 # (NASA) Dynamixel temperature confirmed to default setting
DXL_MINIMUM_POSITION_VALUE    = 975 # Dynamixel will rotate between this value
DXL_MAXIMUM_POSITION_VALUE    = 3125 # and this value (note that the Dynamixel would
                                        # not move when the position value is out of
                                        # movable range. Check e-manual about the range of
                                        # the Dynamixel you use.)
DXL_MOVING_STATUS_THRESHOLD = 5 # Dynamixel moving status threshold in degrees
                                    # from ordered
DXL_VELOCITY_LIMIT              = 5   # (NASA) Dynamixel velocity limit (50 NPS)
DXL_PROFILE_VELOCITY            = 5   # Dynamixel ????, cannot exceed velocity limit (50 NPS)
DXL_ACCELERATION_LIMIT      = 10  # (NASA) Dynamixel acceleration limit
DXL_PROFILE_ACCELERATION = 0    # Dynamixel ????, cannot exceed acceleration limit (NPS 10)

# Bias
BIAS    = [0,2048,2048]   # Servo [0,1,2] only using 1-Proximal and 2-Distal
```

```
# Conversion constants
DEG_PER_COUNT  = 360.0/4096.0    # Deg/step = 0.0878, 4096 steps in 1 turn or 360 deg

#UDP constants
GS_IP = “192.168.0.104”
GS_UDP_Port = 25010

#Frequency
FREQ = 20;

##########################################################################################

#RA_Functions.py
from dynamixel_sdk import *                  # Uses Dynamixel SDK library
from dual_mc33926_rpi import motors
import RA_Constants as RAC
import os
import time
import socket
import struct

if os.name == ‘nt’:
   import msvcrt
   def getch():
      return msvcrt.getch().decode()
else:
   import sys, tty, termios
   fd = sys.stdin.fileno()
   old_settings = termios.tcgetattr(fd)
   def getch():
      try:
         tty.setraw(sys.stdin.fileno())
         ch = sys.stdin.read(1)
      finally:
         termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
      return ch


def InitializeComms():

            # Initialize PortHandler instance
            # Set the port path
            # Get methods and members of PortHandlerLinux or PortHandlerWindows
            portHandler = PortHandler(RAC.DEVICENAME)

            # Initialize PacketHandler instance
            # Set the protocol version
            # Get methods and members of Protocol1PacketHandler or Protocol2PacketHandler
            packetHandler = PacketHandler(RAC.PROTOCOL_VERSION)

            # Open port
            if portHandler.openPort():
               print(“Succeeded to open the port”)
            else:
               print(“Failed to open the port”)
```

```
            print("Press any key to terminate...")
            getch()
            quit()

        # Set port baudrate
        if portHandler.setBaudRate(RAC.BAUDRATE):
            print("Succeeded to change the baudrate")
        else:
            print("Failed to change the baudrate")
            print("Press any key to terminate...")
            getch()
            quit()

        return portHandler, packetHandler


def SetUpServo(portHandler,packetHandler,DXL_ID):
        # Disable Dynamixel Torque
        dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler, DXL_ID,
RAC.ADDR_PRO_TORQUE_ENABLE, RAC.TORQUE_DISABLE)
        if dxl_comm_result != COMM_SUCCESS:
            print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
        elif dxl_error != 0:
            print("%s" % packetHandler.getRxPacketError(dxl_error))
        else:
            print("Dynamixel has been successfully connected")

        # Write velocity
        dxl_comm_result, dxl_error = packetHandler.write4ByteTxRx(portHandler, DXL_ID,
RAC.ADDR_PRO_VELOCITY_LIMIT, RAC.DXL_VELOCITY_LIMIT)
        if dxl_comm_result != COMM_SUCCESS:
            print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
        elif dxl_error != 0:
            print("%s" % packetHandler.getRxPacketError(dxl_error))

        # Write acceleration
        dxl_comm_result, dxl_error = packetHandler.write4ByteTxRx(portHandler, DXL_ID,
RAC.ADDR_PRO_ACCELERATION_LIMIT, RAC.DXL_ACCELERATION_LIMIT)
        if dxl_comm_result != COMM_SUCCESS:
            print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
        elif dxl_error != 0:
            print("%s" % packetHandler.getRxPacketError(dxl_error))

        # Write profile velocity
        dxl_comm_result, dxl_error = packetHandler.write4ByteTxRx(portHandler, DXL_ID,
RAC.ADDR_PRO_PROFILE_VELOCITY, RAC.DXL_PROFILE_VELOCITY)
        if dxl_comm_result != COMM_SUCCESS:
            print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
        elif dxl_error != 0:
            print("%s" % packetHandler.getRxPacketError(dxl_error))

        # Enable Dynamixel Torque
        dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler, DXL_ID,
RAC.ADDR_PRO_TORQUE_ENABLE, RAC.TORQUE_ENABLE)
        if dxl_comm_result != COMM_SUCCESS:
```

```
            print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
        elif dxl_error != 0:
            print("%s" % packetHandler.getRxPacketError(dxl_error))
        else:
            print("Dynamixel has been successfully connected")

def MoveIDTo(portHandler,packetHandler,DXL_ID,Goal_Pos):
        # Write goal position
    dxl_comm_result, dxl_error = packetHandler.write4ByteTxRx(portHandler, DXL_ID,
RAC.ADDR_PRO_GOAL_POSITION,int(round(Goal_Pos/RAC.DEG_PER_COUNT))+RAC.BIAS[DX
L_ID])
    if dxl_comm_result != COMM_SUCCESS:
        print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
    elif dxl_error != 0:
        print("%s" % packetHandler.getRxPacketError(dxl_error))

def GetPosition(portHandler,packetHandler,DXL_ID):
        # Read present position
    dxl_present_position, dxl_comm_result, dxl_error = packetHandler.read4ByteTxRx(portHandler,
DXL_ID, RAC.ADDR_PRO_PRESENT_POSITION)
    if dxl_comm_result != COMM_SUCCESS:
        print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
    elif dxl_error != 0:
        print("%s" % packetHandler.getRxPacketError(dxl_error))

    return float(dxl_present_position-RAC.BIAS[DXL_ID])*RAC.DEG_PER_COUNT

# print("[ID:%03d] GoalPos:%03d  PresPos:%03d" % (DXL_ID_DISTAL, dxl_goal_position[index],
dxl_present_position))


def SendingSleep(arm,sleep_time):
        #Loop start time
        start_loop_time = time.time()
        t = 0;
        while t<sleep_time:
                start_time = time.time()
                pos=arm.GetPos()
                arm.sock.sendto(struct.pack('%sf' % len(pos), *pos), (RAC.GS_IP,
RAC.GS_UDP_Port))
                current_time=time.time()
                if ((current_time - start_time)<(1.0/RAC.FREQ)):
                        time.sleep(1.0/RAC.FREQ-(current_time - start_time))
                else:
                        print('Not meeting deadline')
                t=time.time()-start_loop_time;

class arm:

        def __init__(self):

                #Parameters
                self.gripper_state = 0
                self.moving_status_threshold = RAC.DXL_MOVING_STATUS_THRESHOLD
```

```
        # Initialize serial comms port
                self.portHandler,self.packetHandler=InitializeComms()

                #Set Up the servos
                SetUpServo(self.portHandler,self.packetHandler,RAC.DXL_ID_PROXIMAL)
                SetUpServo(self.portHandler,self.packetHandler,RAC.DXL_ID_DISTAL)

                # Set Up gipper motor to zero
                motors.enable()
                motors.setSpeeds(0, 0)

                #Configure socket
                self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        # Methods
        def Move(self,Goal_Proximal,Goal_Distal,wait=False):

        MoveIDTo(self.portHandler,self.packetHandler,RAC.DXL_ID_PROXIMAL,Goal_Proximal)
                MoveIDTo(self.portHandler,self.packetHandler,RAC.DXL_ID_DISTAL,Goal_Distal)
                if wait:
                        while True:
                                pos=self.GetPos()
                                if (abs(pos[0] - Goal_Proximal) < self.moving_status_threshold) &
(abs(pos[1] - Goal_Distal) < self.moving_status_threshold):
                                        break

        def GetPos(self):

        Proximal_Pos=GetPosition(self.portHandler,self.packetHandler,RAC.DXL_ID_PROXIMAL)
                Distal_Pos=GetPosition(self.portHandler,self.packetHandler,RAC.DXL_ID_DISTAL)
                return Proximal_Pos, Distal_Pos

        def Fold(self):
                self.Move(0,0,True)
                self.Close_Gripper()
                self.Move(90,0,True)

        def Open_Gripper(self):
                if self.gripper_state == 0:
                        motors.motor1.setSpeed(RAC.GRIPPER_TORQUE)
                        #time.sleep(RAC.GRIPPER_OPENTIME)
                        SendingSleep(self,RAC.GRIPPER_OPENTIME)
                        motors.setSpeeds(0, 0)
                        self.gripper_state = 1
                else:
                        print “Gripper is Already Open”


        def Close_Gripper(self):
                if self.gripper_state == 1:
                        motors.motor1.setSpeed(-RAC.GRIPPER_TORQUE)
                        #time.sleep(RAC.GRIPPER_CLOSETIME)
                        SendingSleep(self,RAC.GRIPPER_CLOSETIME)
                        motors.setSpeeds(0, 0)
                        self.gripper_state = 0
```

```
                else:
                        print “Gripper is Already Closed”

        # Clearing Objects, Resetting
        def __del__(self):
                self.Fold()
                time.sleep(2)

                # Disable Dynamixel Torque of Proximal
                dxl_comm_result, dxl_error = self.packetHandler.write1ByteTxRx(self.portHandler,
RAC.DXL_ID_PROXIMAL, RAC.ADDR_PRO_TORQUE_ENABLE, RAC.TORQUE_DISABLE)
                if dxl_comm_result != COMM_SUCCESS:
                        print(“%s” % self.packetHandler.getTxRxResult(dxl_comm_result))
                elif dxl_error != 0:
                        print(“%s” % self.packetHandler.getRxPacketError(dxl_error))

                # Disable Dynamixel Torque of Distal
                dxl_comm_result, dxl_error = self.packetHandler.write1ByteTxRx(self.portHandler,
RAC.DXL_ID_DISTAL, RAC.ADDR_PRO_TORQUE_ENABLE, RAC.TORQUE_DISABLE)
                if dxl_comm_result != COMM_SUCCESS:
                        print(“%s” % self.packetHandler.getTxRxResult(dxl_comm_result))
                elif dxl_error != 0:
                        print(“%s” % self.packetHandler.getRxPacketError(dxl_error))

                # Close port
                self.portHandler.closePort()

                # Stop the motors, even if there is an exception
                # or the user presses Ctrl+C to kill the process.
                motors.setSpeeds(0, 0)
                motors.disable()

##############################################################################################

#test_90.py
import RA_Constants as RAC
import RA_Functions as RAF
import time

arm= RAF.arm()
arm.Move(0,0,True)
arm.Open_Gripper()
arm.Move(-90,0,True)
arm.Close_Gripper()
time.sleep(30)
arm.Open_Gripper()
arm.Move(0,0,True)

##############################################################################################

#self_toss.py
import RA_Constants as RAC
import RA_Functions as RAF
import time
import struct
```

```
def sign(x):
	return 1–2*int(x<=0)


#Start angle
# To release right, angle must be between 90 to 0
# To release left, angle must be between -90 to 0
S_angle=80 # -60 max -110
# 80

#Release
#
R_angle=20 # -6
# 20

#Self-toss direction
direction = sign(R_angle-S_angle)
print direction

#Initialize
arm = RAF.arm()

#Perch
arm.Open_Gripper()
arm.Move(S_angle,0,True)
arm.Close_Gripper()

#Toss
RAF.SendingSleep(arm,30)

arm.Move(R_angle+20*direction,0,False)

while True:

	start_time = time.time()
	pos=arm.GetPos()
	arm.sock.sendto(struct.pack('%sf' % len(pos), *pos), (RAC.GS_IP, RAC.GS_UDP_Port))


	if (pos[0]-R_angle)*direction>0:
		arm.Open_Gripper()
		break

	current_time=time.time();
	if ((current_time - start_time)<(1.0/RAC.FREQ)):
		time.sleep(1.0/RAC.FREQ-(current_time - start_time))
	else:
		print('Not meeting deadline')



arm.Move(R_angle,0,False)
RAF.SendingSleep(arm,60)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. MATLAB CODE

```
%Test_loading_Justin.m
%Reads load cell data

%% Clean and clear
clear
clc
close all

%% Read sensor data
Sensor_Param

%% Discover Analog Input Devices
% To discover a device that supports analog input subsystems, click the
% name of the device in the list in the Command window, or access the
% device in the array returned by |daq.getDevices| command. This example
% uses a NI 9201 device with ID 'cDAQ1Mod4'. This is a 8 channel analog
% input module and is in slot 4 of Chassis 'cDAQ1'.
device = daq.getDevices;

%% Create a Session and Add an Analog Input Channel
% Create a session, and use the |addAnalogInputChannel| function to add two
% analog input channels from this device to the session.
s = daq.createSession('ni');
% addAnalogInputChannel(s,'cDAQ1Mod4', 0, 'Voltage');
% addAnalogInputChannel(s,'cDAQ1Mod4', 1, 'Voltage');
addAnalogInputChannel(s,'Dev1', 0, 'Voltage');
addAnalogInputChannel(s,'Dev1', 1, 'Voltage');
addAnalogInputChannel(s,'Dev1', 2, 'Voltage');
addAnalogInputChannel(s,'Dev1', 3, 'Voltage');
addAnalogInputChannel(s,'Dev1', 4, 'Voltage');
addAnalogInputChannel(s,'Dev1', 5, 'Voltage');
%% Set Session Rate
% By default the session is configured for 1000 scans/second.
% Change the scan rate to acquire at 8000 scans / second.
s.Rate = 80;

%% Set bias
Gb = (s.inputSingleScan*M')';
disp('Bias Set.')

%% Acquire data for a Specified Duration

%Pause to attached gripper
disp('...Paused waiting for manipulator...')
pause(10)

%Adqusistion time
s.DurationInSeconds = 15;

%Adquire data
disp('Reading forces')
```

```
[raw_data,time] = s.startForeground;

%Process data
data = raw_data*M'-ones(s.Rate*s.DurationInSeconds,1)*Gb';
force   = data(:,1:3);   % [N]
torque = data(:,4:6);   % [Nm]

%% Plot output
close all

figure(1)
plot(time,force);
xlabel('Time (secs)');
ylabel('Force [N]')
legend({'Fx','Fy','Fz'})

figure(2)
plot(time,torque);
xlabel('Time (secs)');
ylabel('Torque [N mm]')
legend({'Tx','Ty','Tz'})
```

APPENDIX C. LINEAR TEST RESULTS

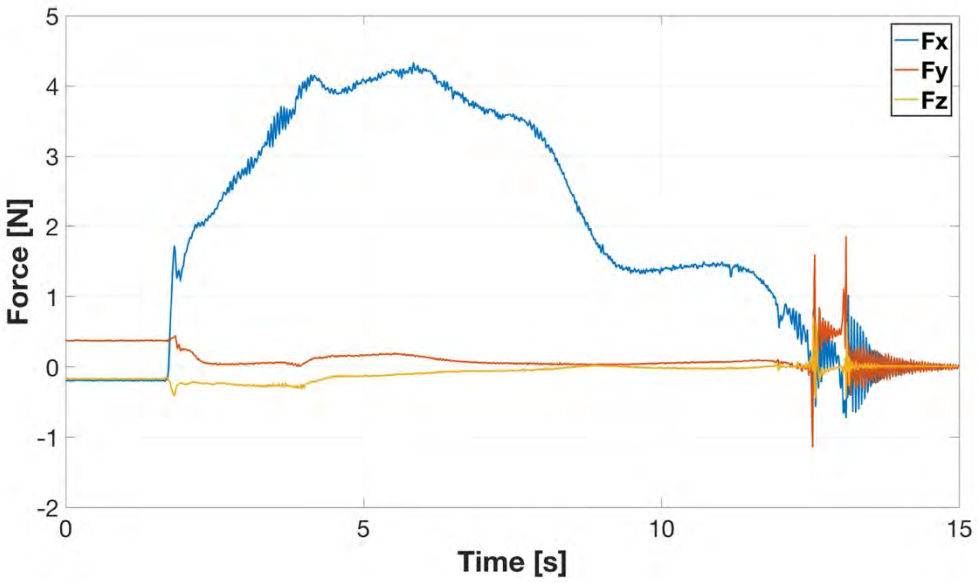

Figure 70. Linear Force Data Sample 1

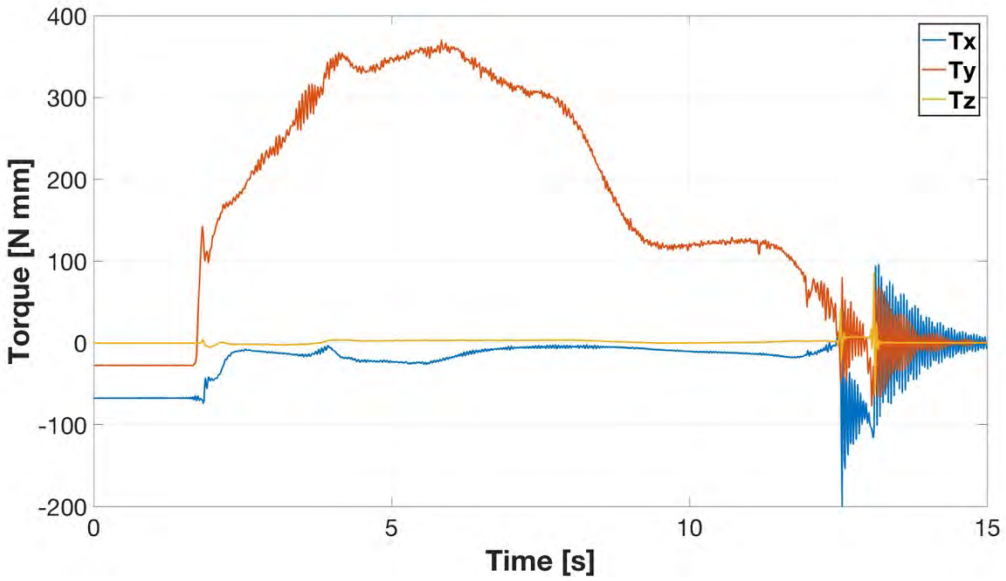

Figure 71. Linear Torque Data Sample 1

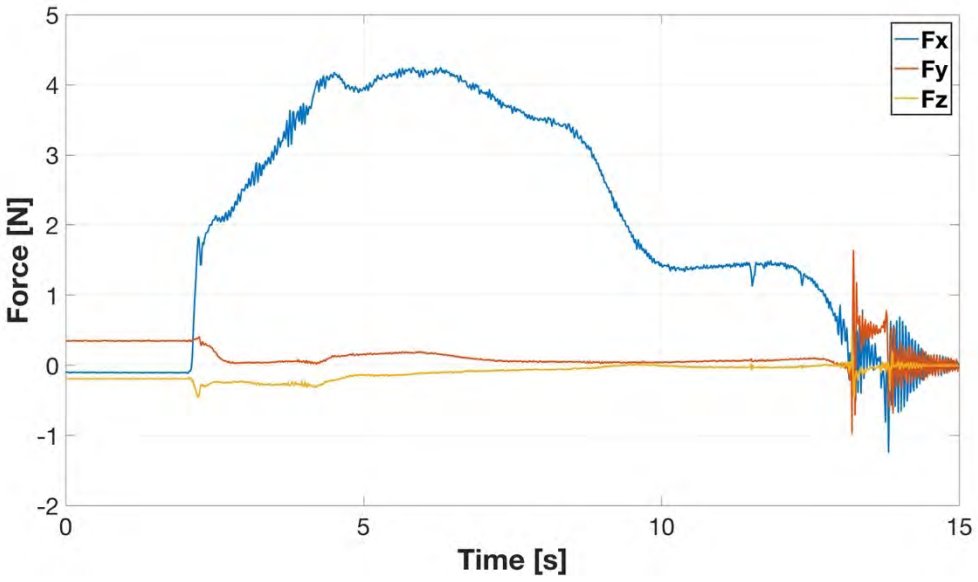

Figure 72. Linear Force Data Sample 2

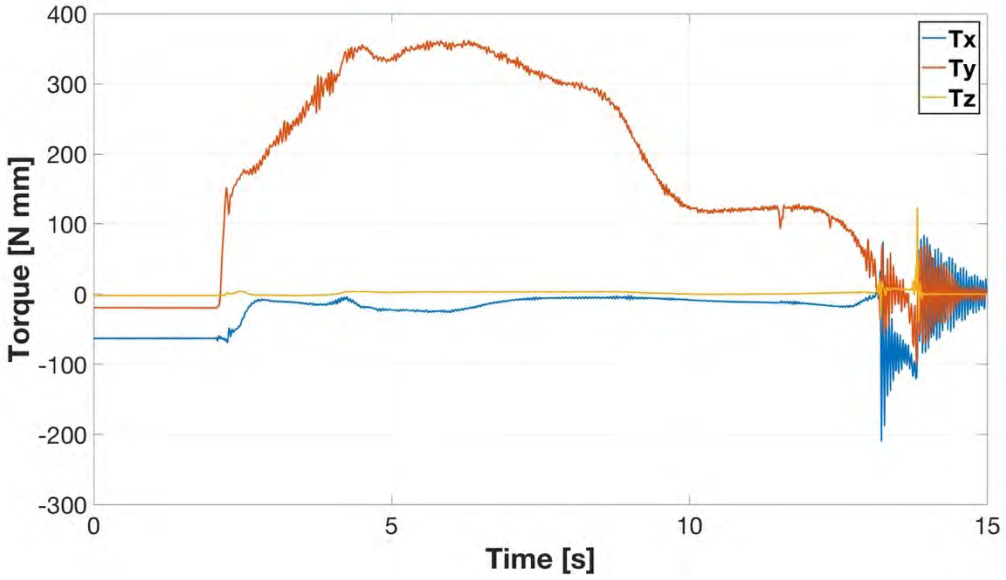

Figure 73. Linear Torque Data Sample 2

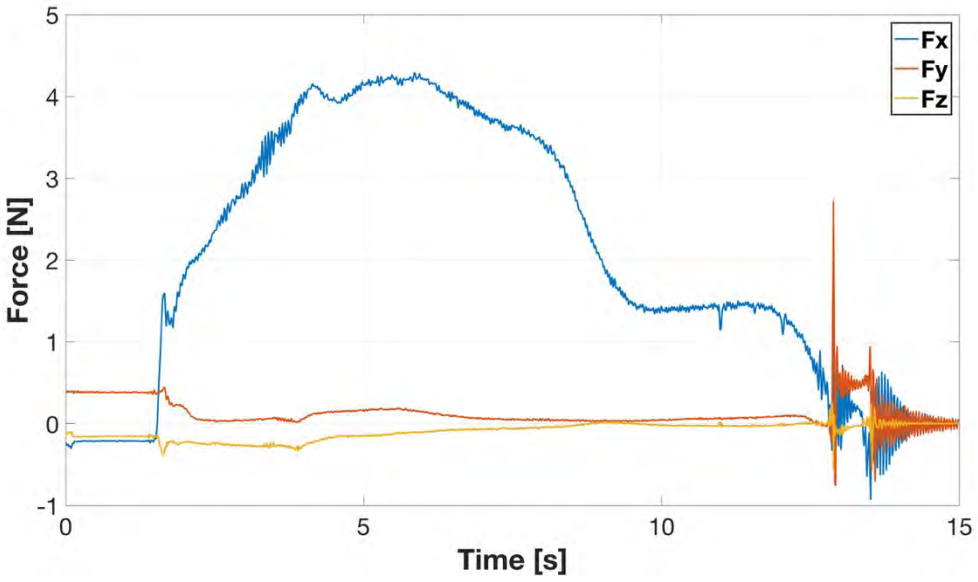

Figure 74. Linear Force Data Sample 3

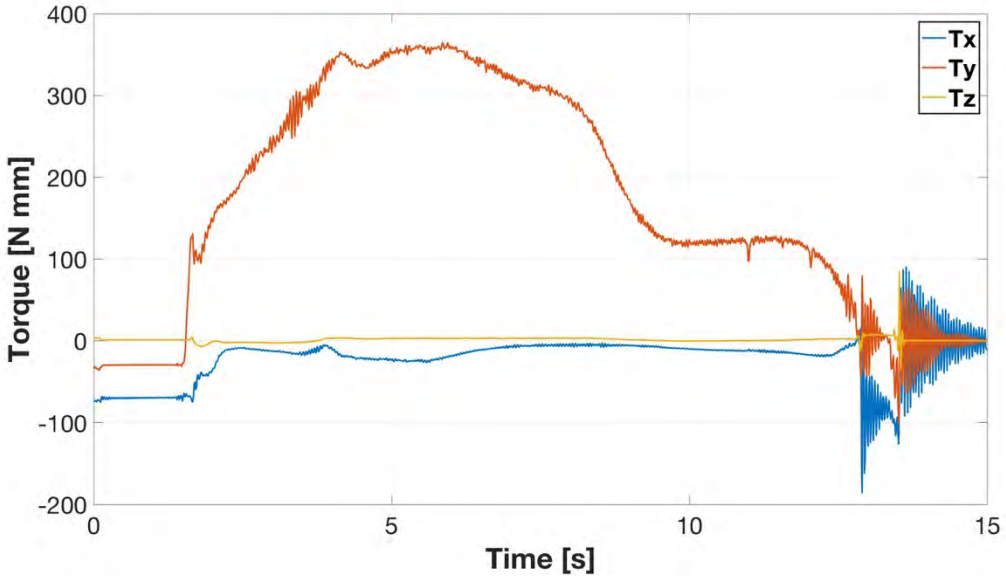

Figure 75. Linear Torque Data Sample 3

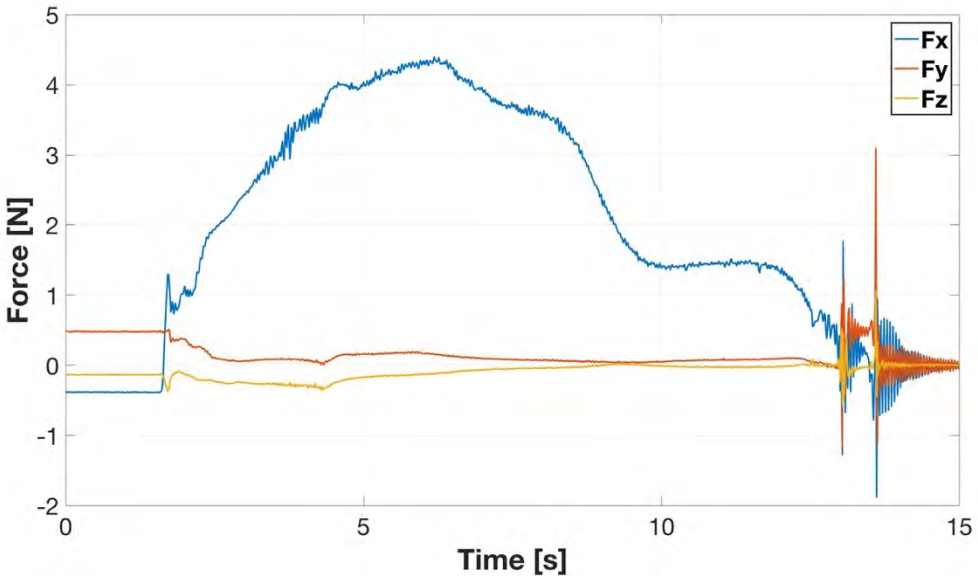

Figure 76. Linear Force Data Sample 4

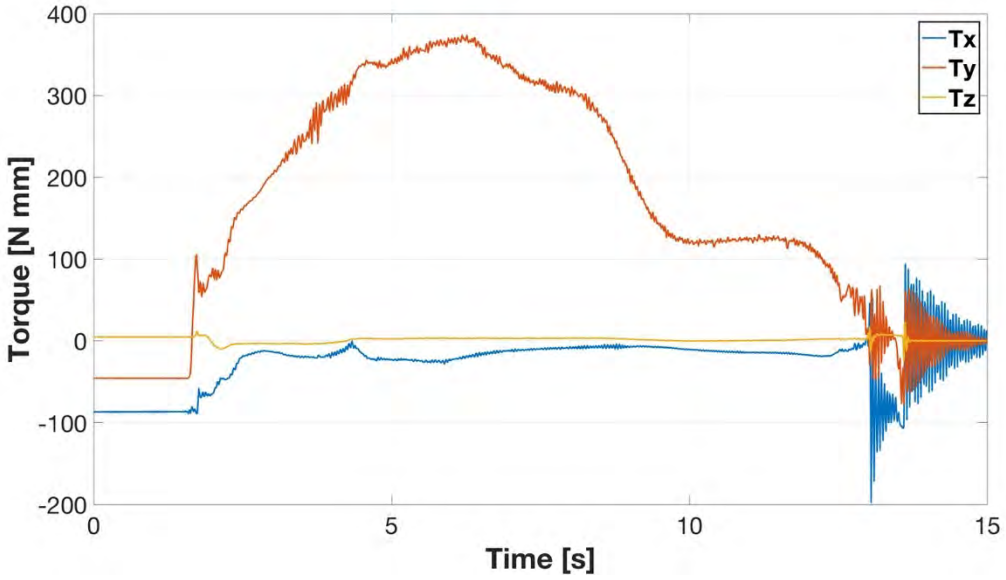

Figure 77. Linear Torque Data Sample 4

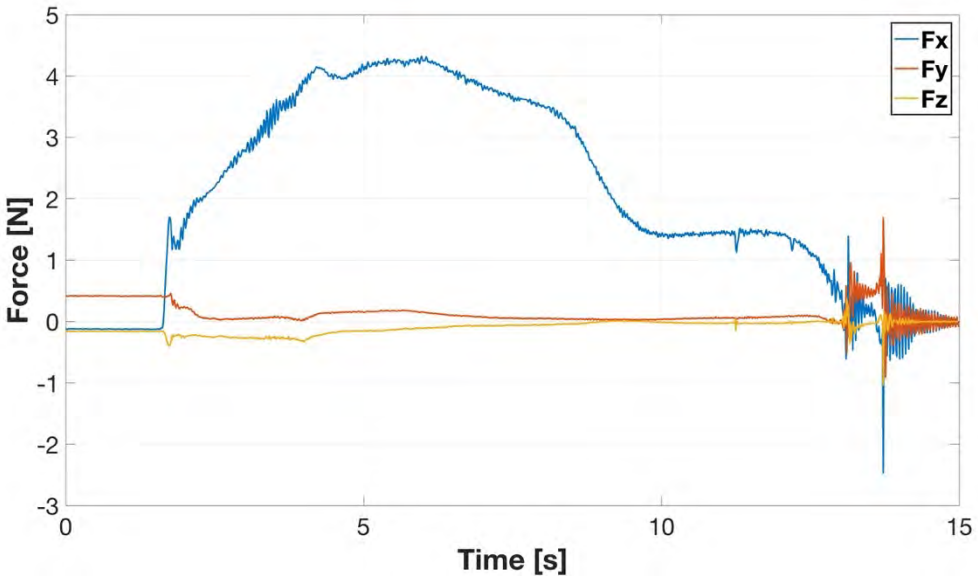

Figure 78. Linear Force Data Sample 5

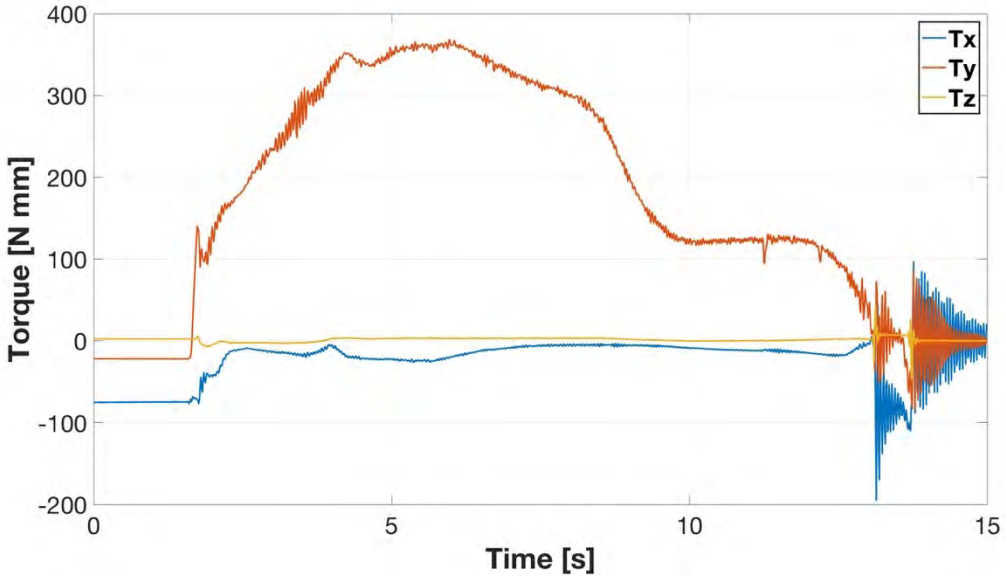

Figure 79. Linear Torque Data Sample 5

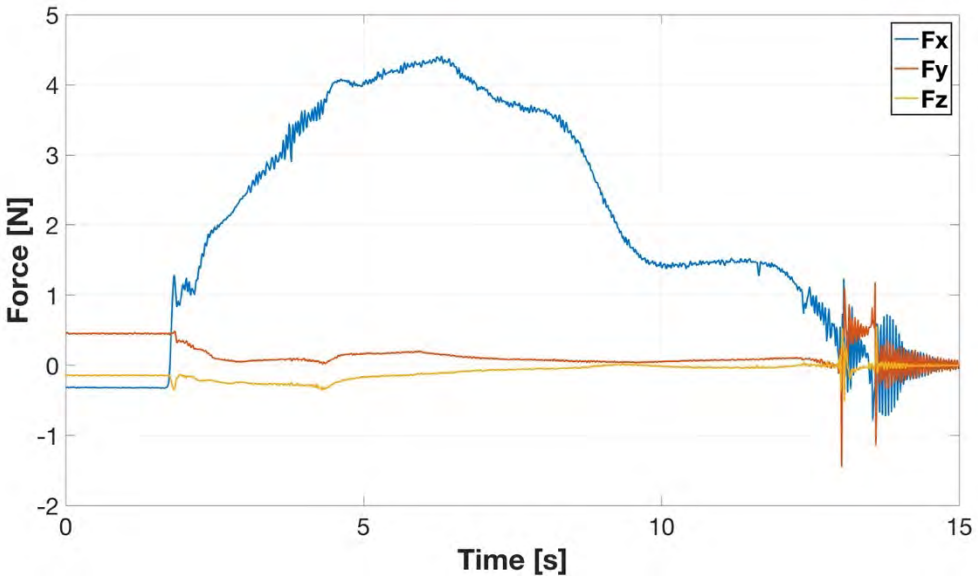

Figure 80. Linear Force Data Sample 6

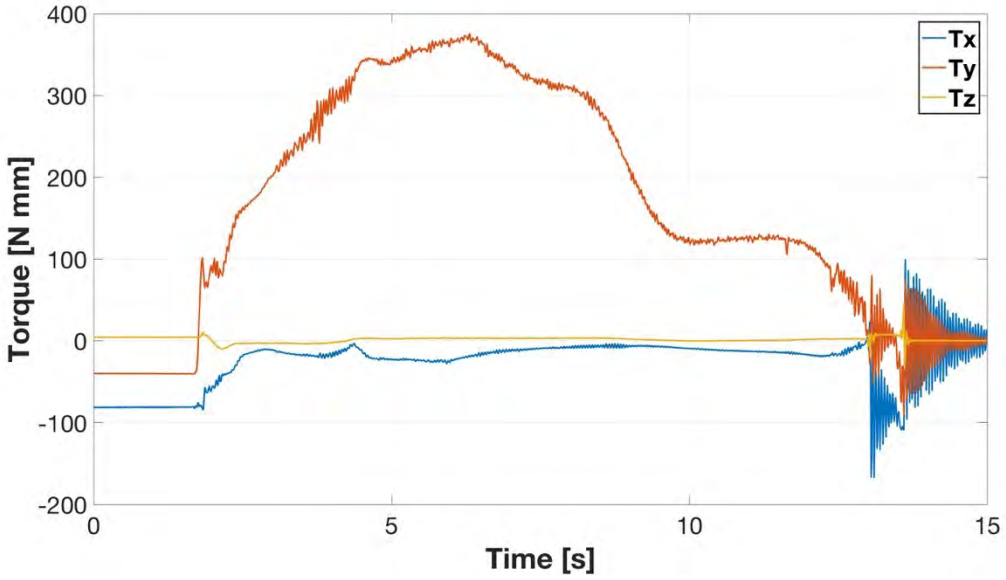

Figure 81. Linear Torque Data Sample 6

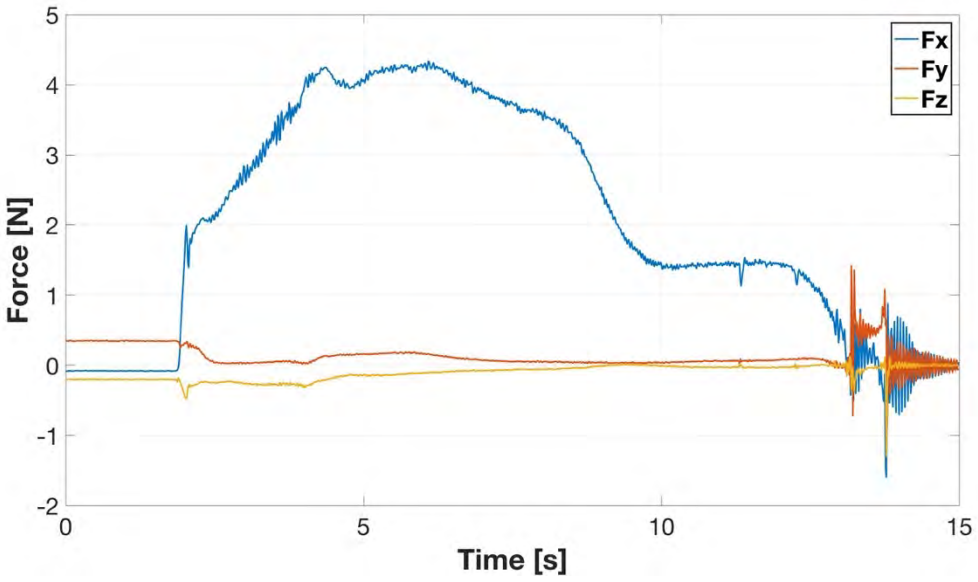

Figure 82. Linear Force Data Sample 7

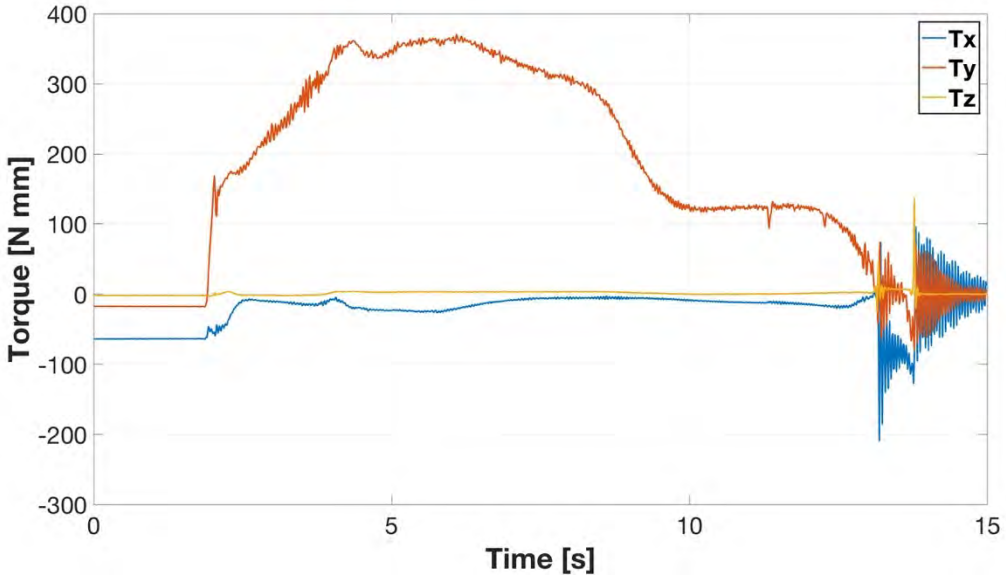

Figure 83. Linear Torque Data Sample 7

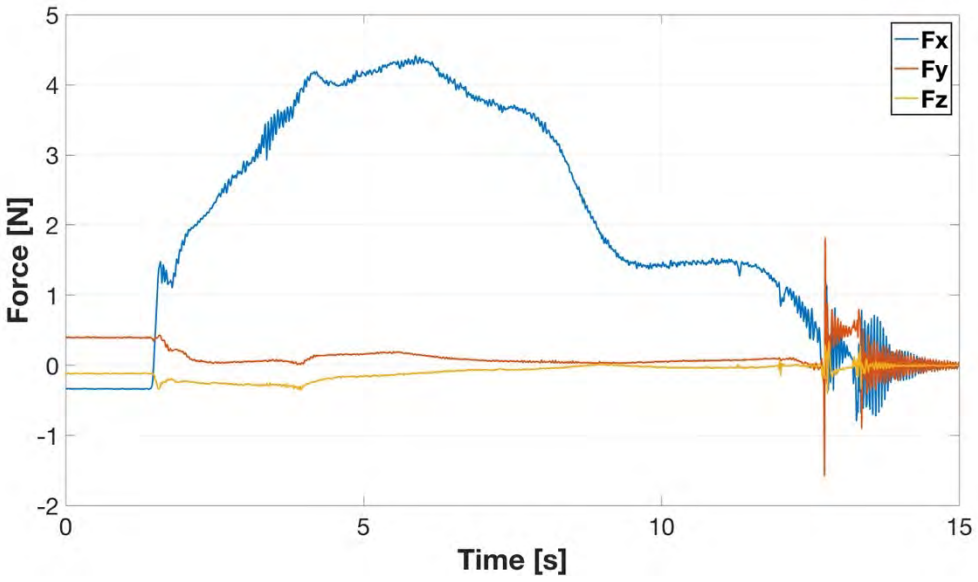

Figure 84. Linear Force Data Sample 8

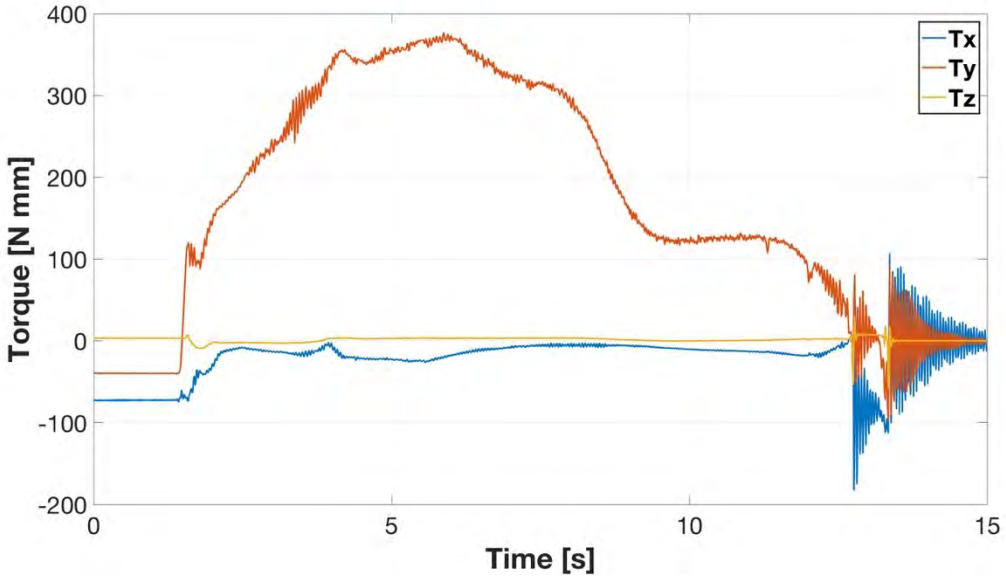

Figure 85. Linear Torque Data Sample 8

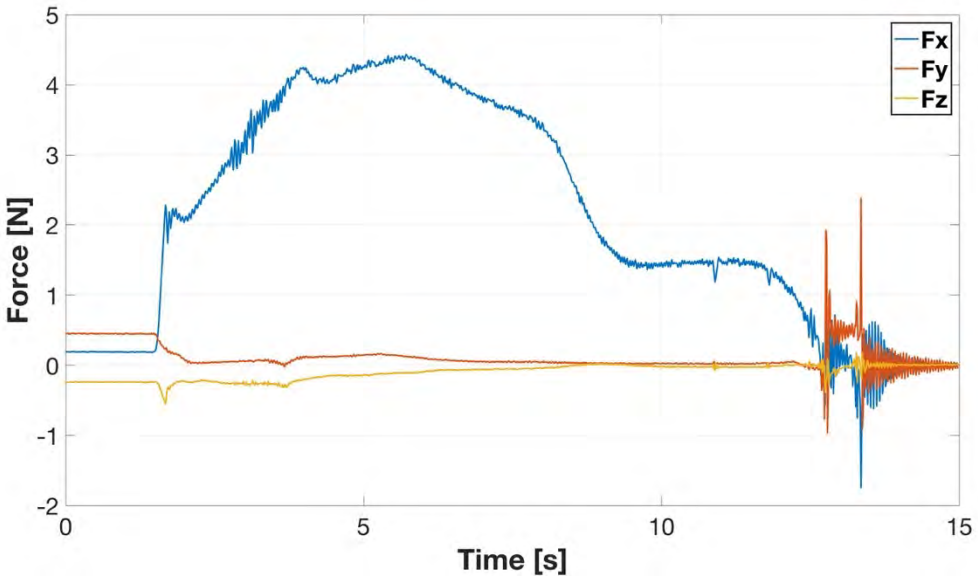

Figure 86. Linear Force Data Sample 9

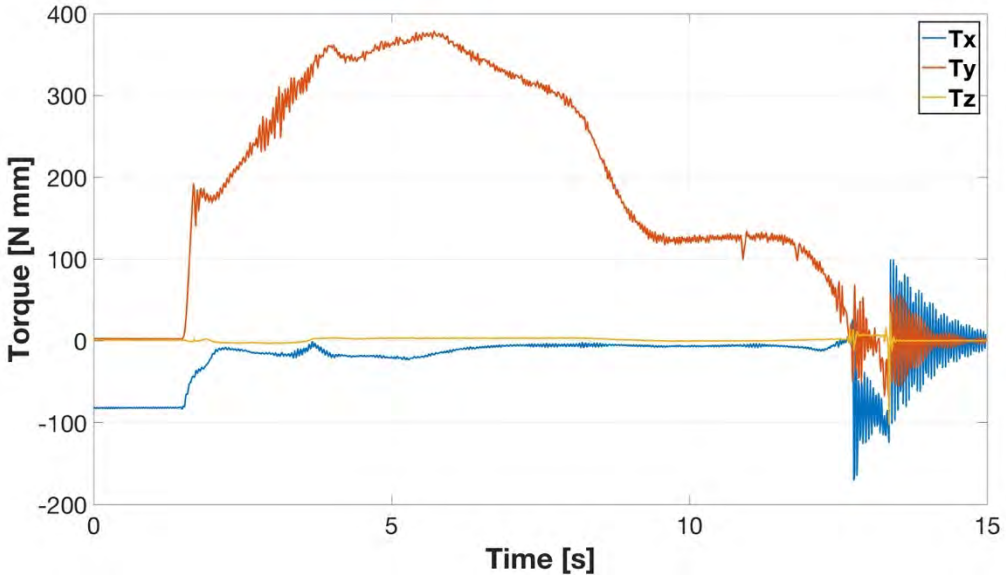

Figure 87. Linear Torque Data Sample 9

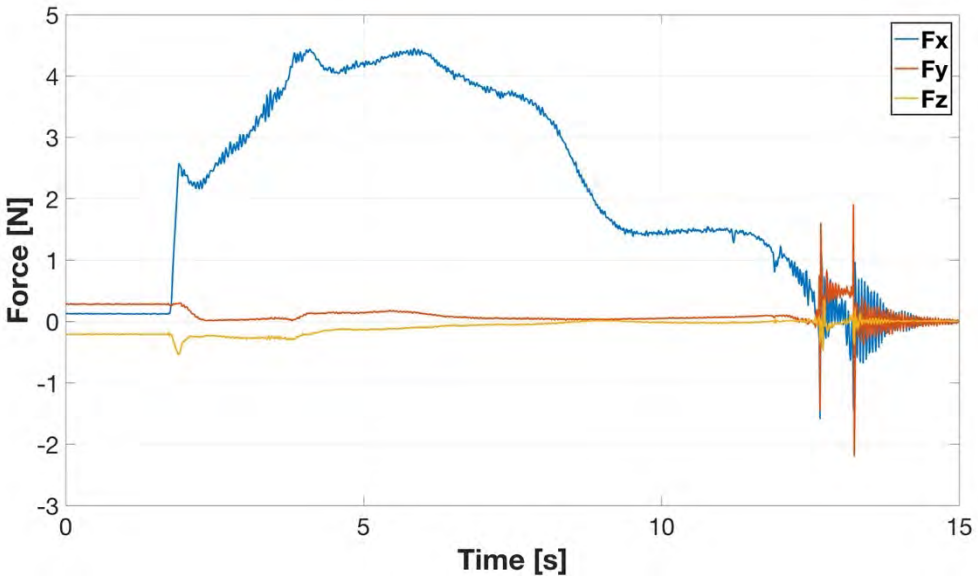

Figure 88. Linear Force Data Sample 10

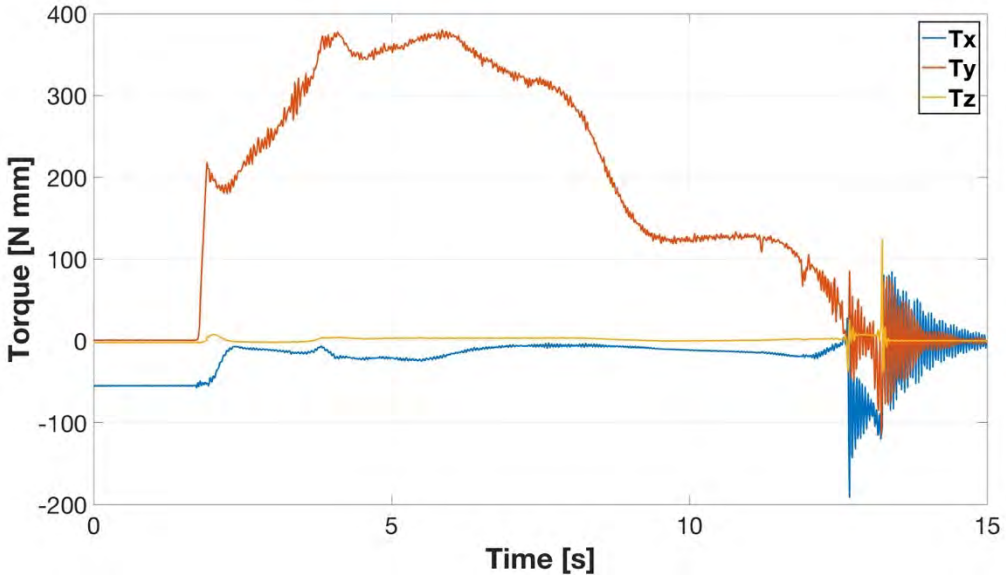

Figure 89. Linear Torque Data Sample 10

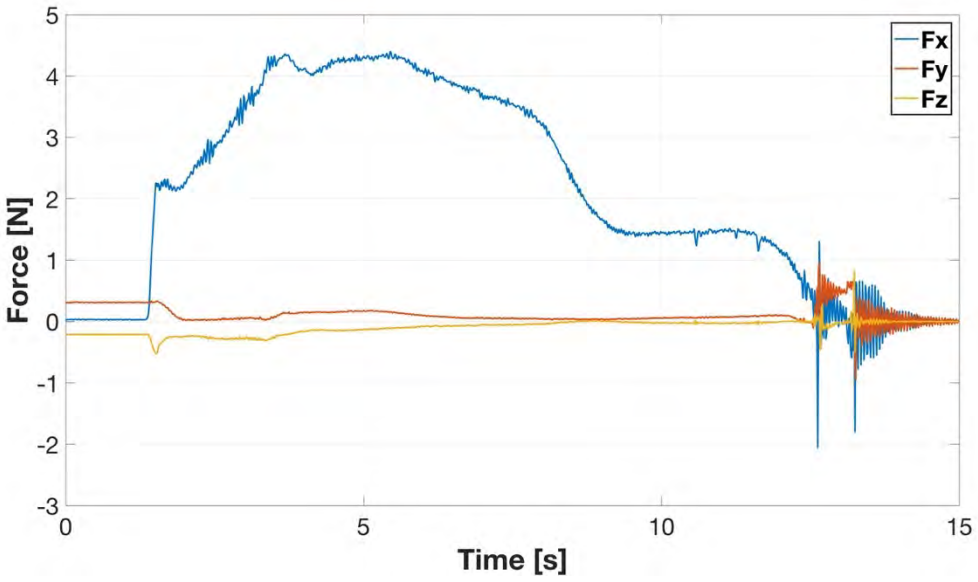

Figure 90. Linear Force Data Sample 11

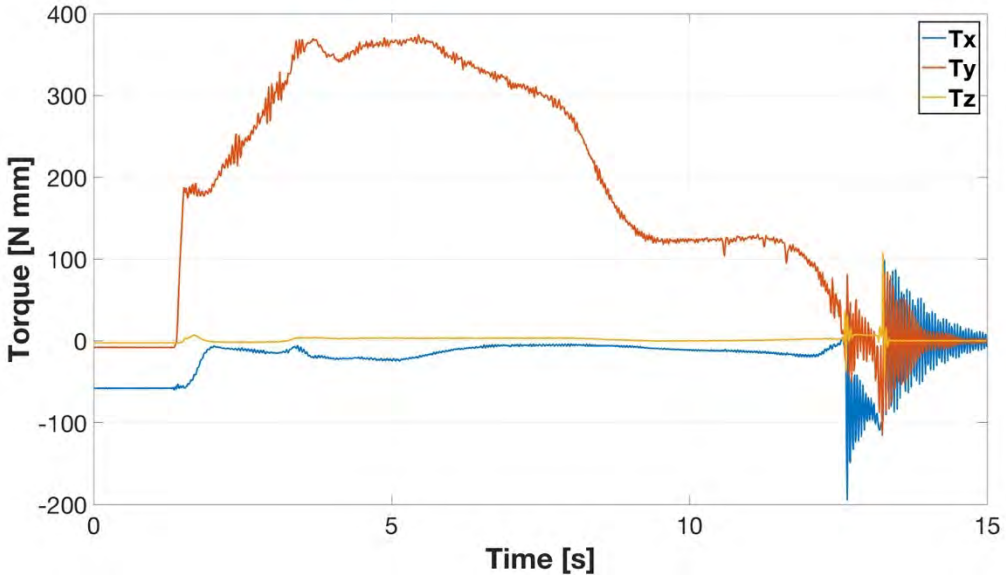

Figure 91. Linear Torque Data Sample 11

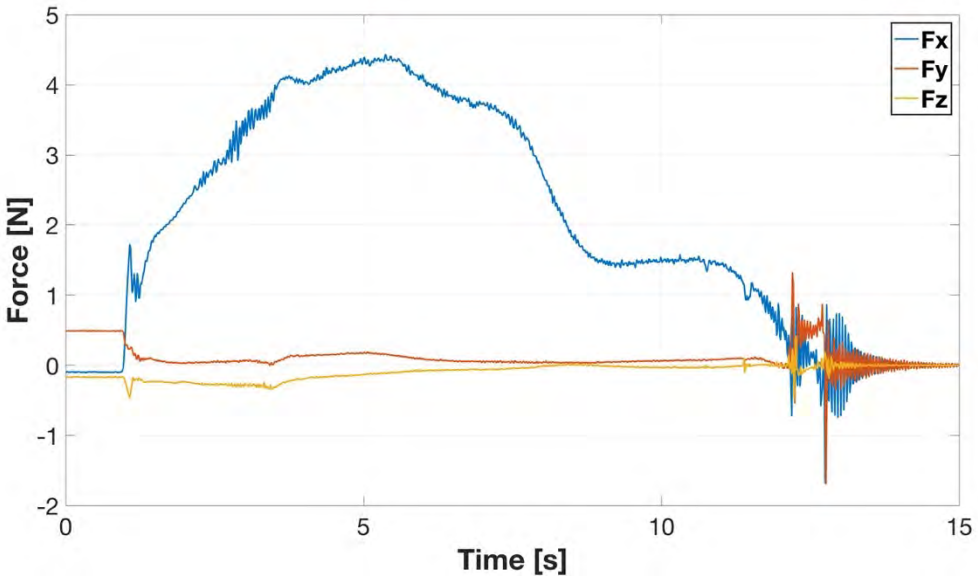

Figure 92. Linear Force Data Sample 12

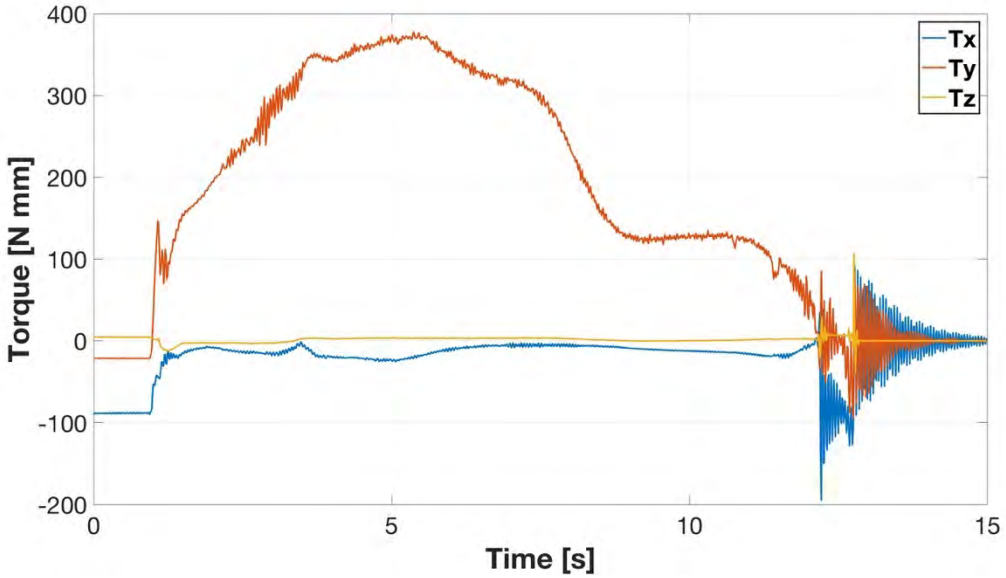

Figure 93. Linear Torque Data Sample 12

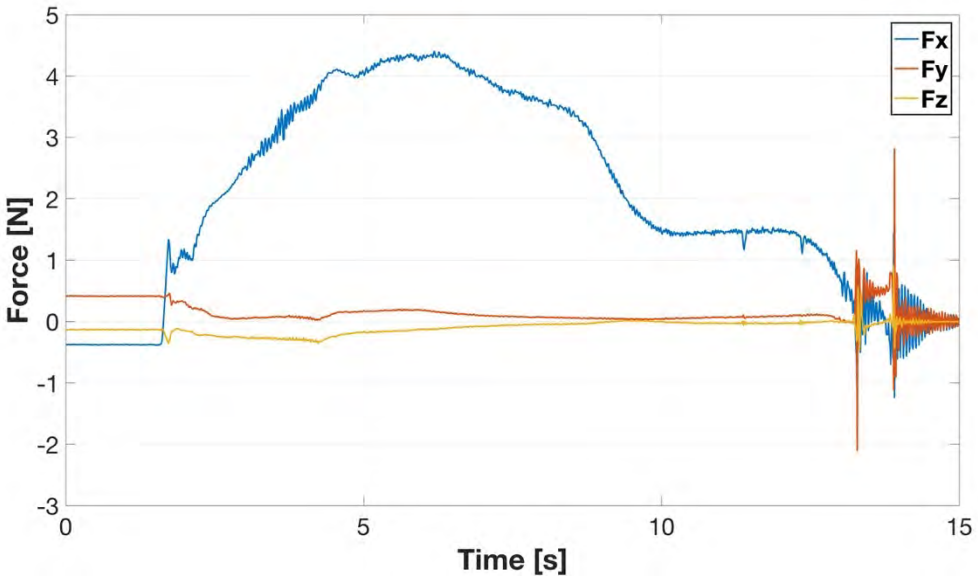

Figure 94. Linear Force Data Sample 13

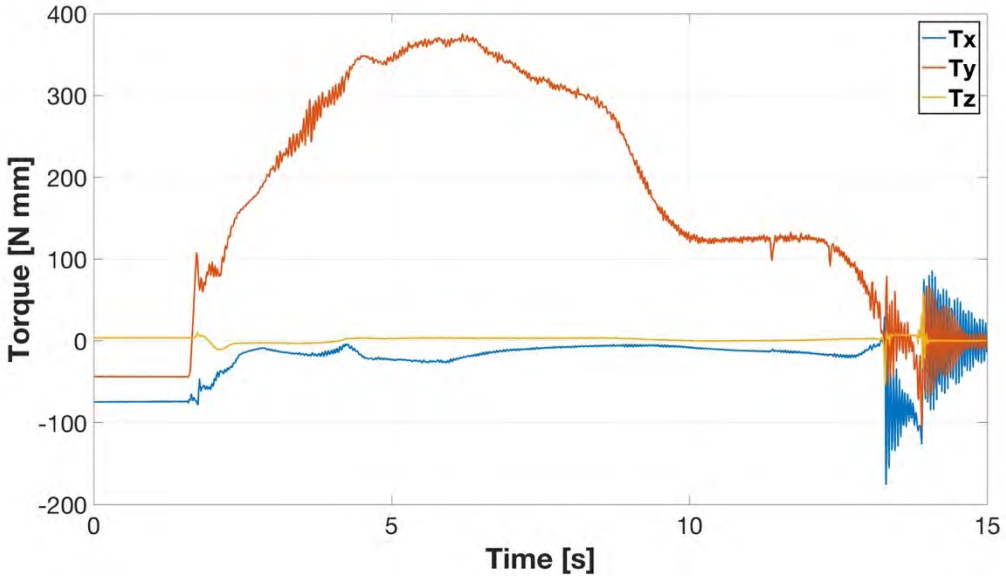

Figure 95. Linear Torque Data Sample 13

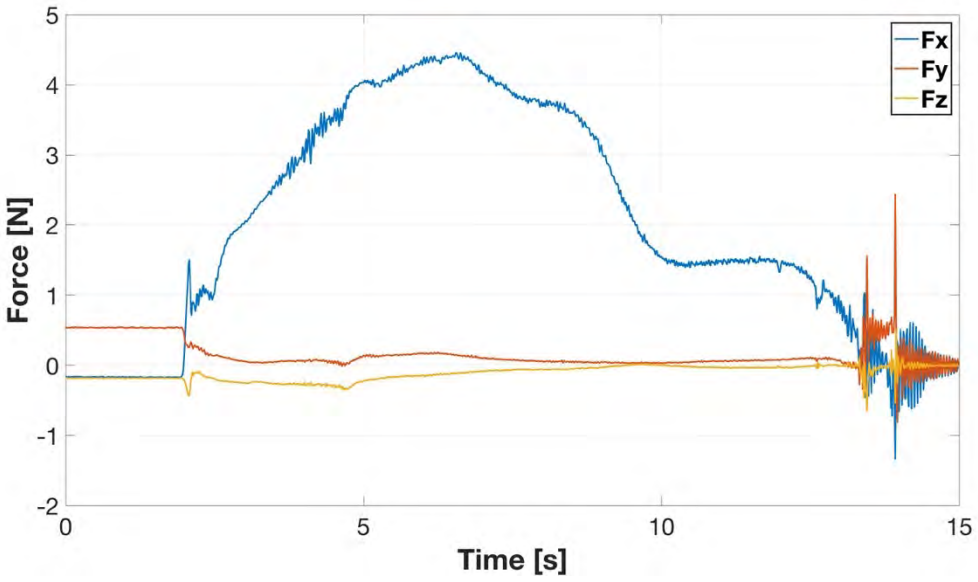

Figure 96. Linear Force Data Sample 14

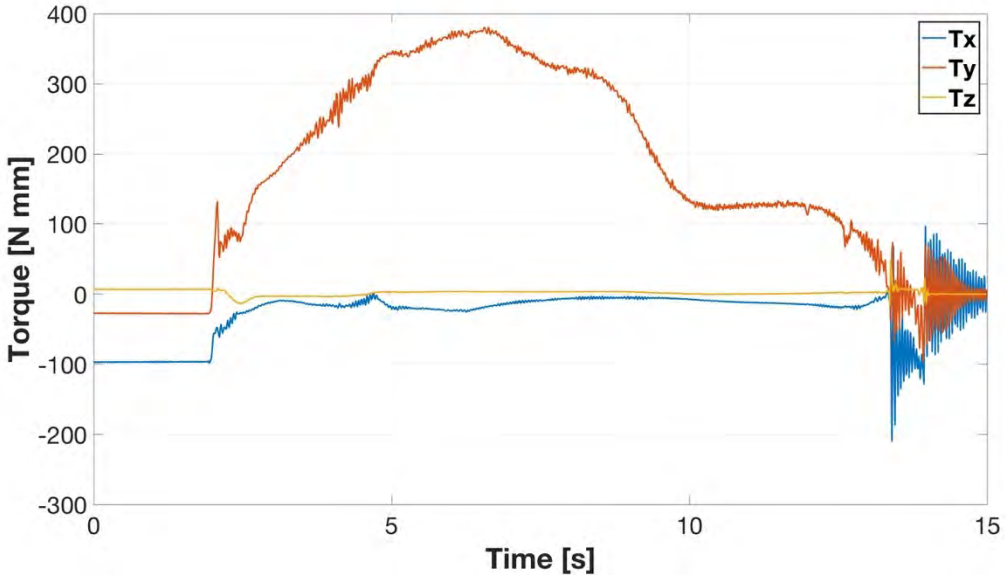

Figure 97. Linear Torque Data Sample 14

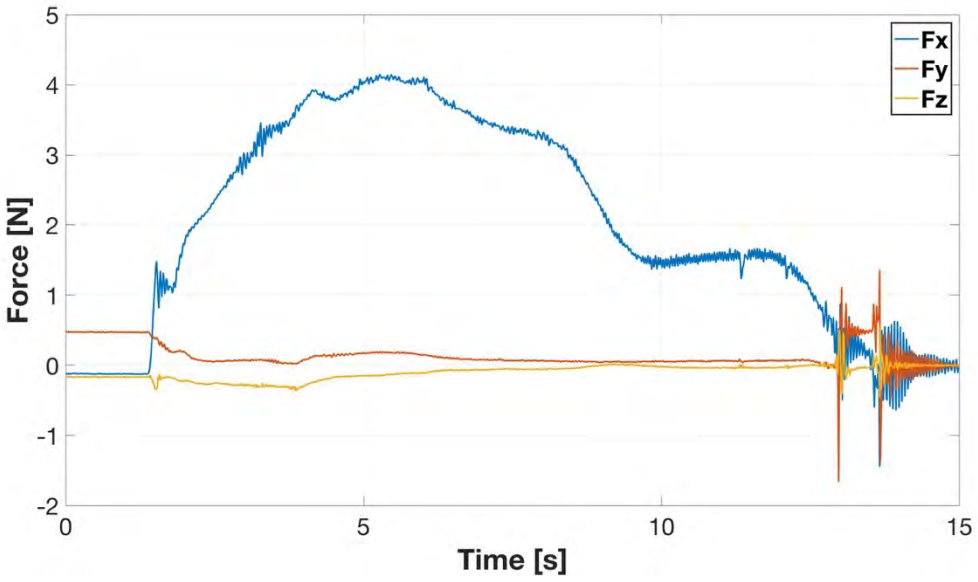

Figure 98. Linear Force Data Sample 15

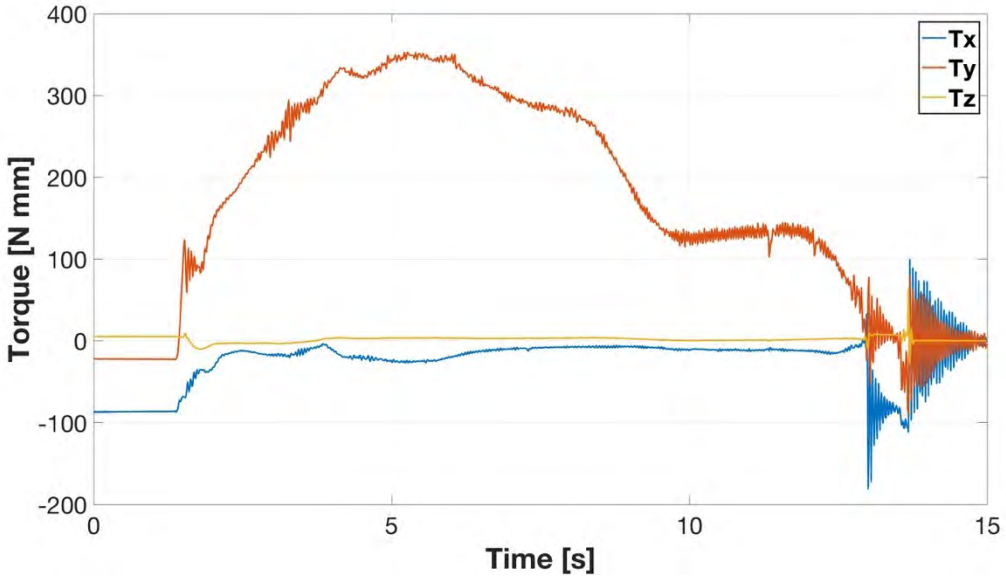

Figure 99. Linear Torque Data Sample 15

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. SELF-TOSS RESULTS

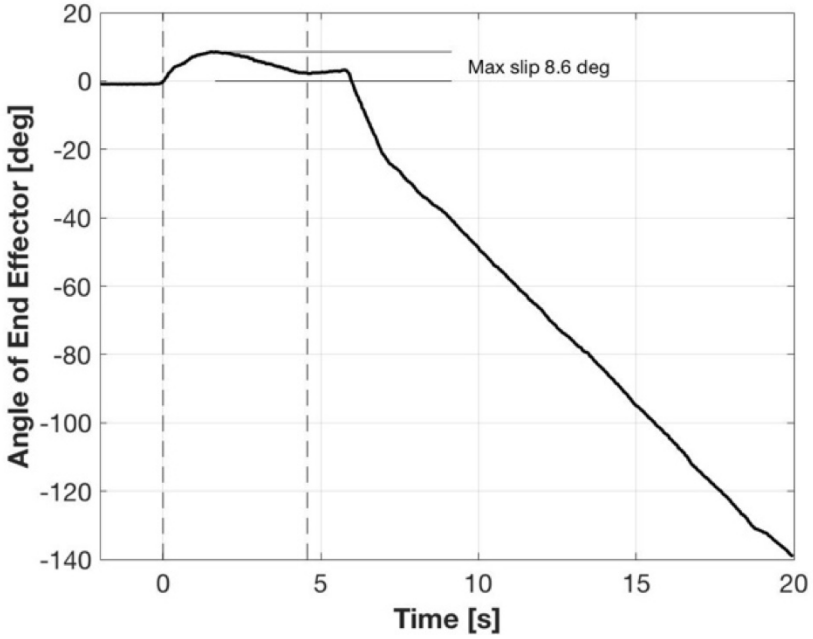

Figure 100. Self-Toss Slip Angle Sample 1

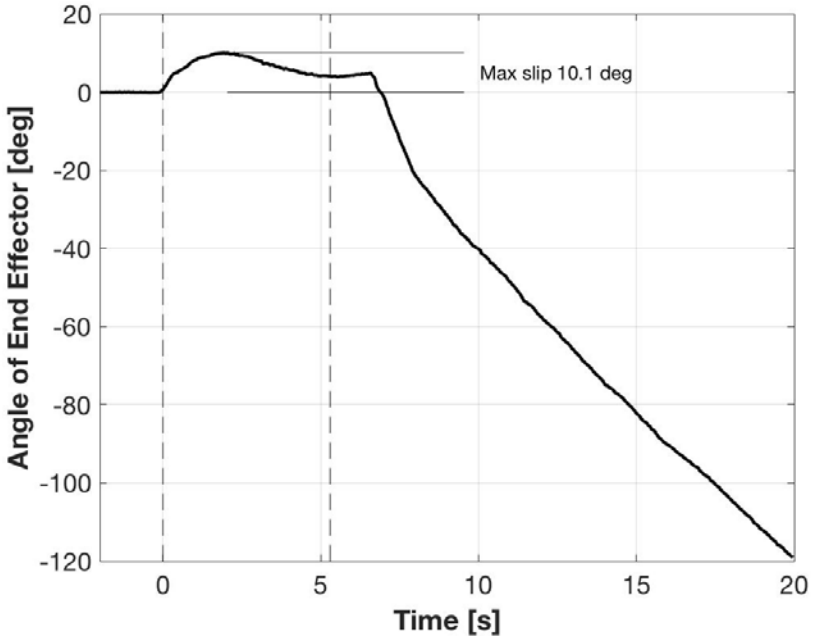

Figure 101. Self-Toss Slip Angle Sample 2

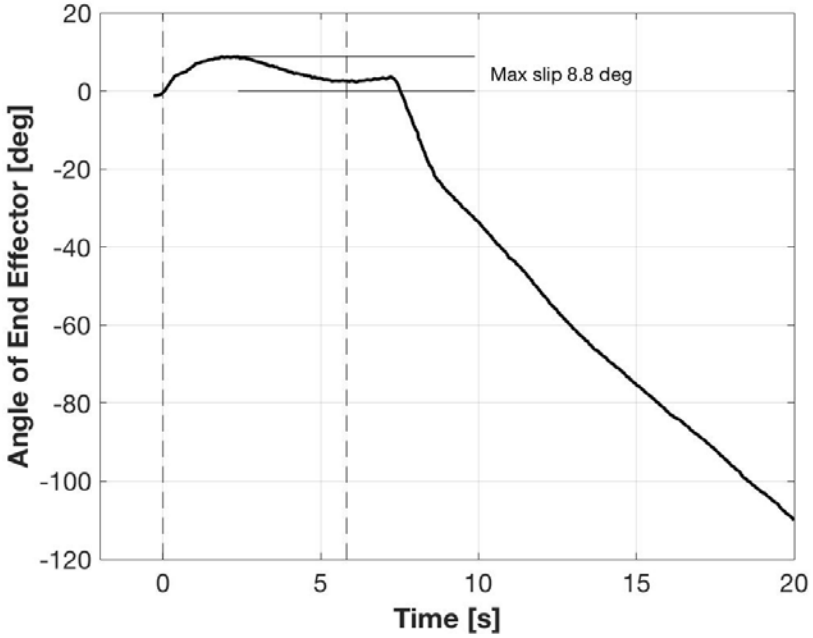

Figure 102. Self-Toss Slip Angle Sample 3

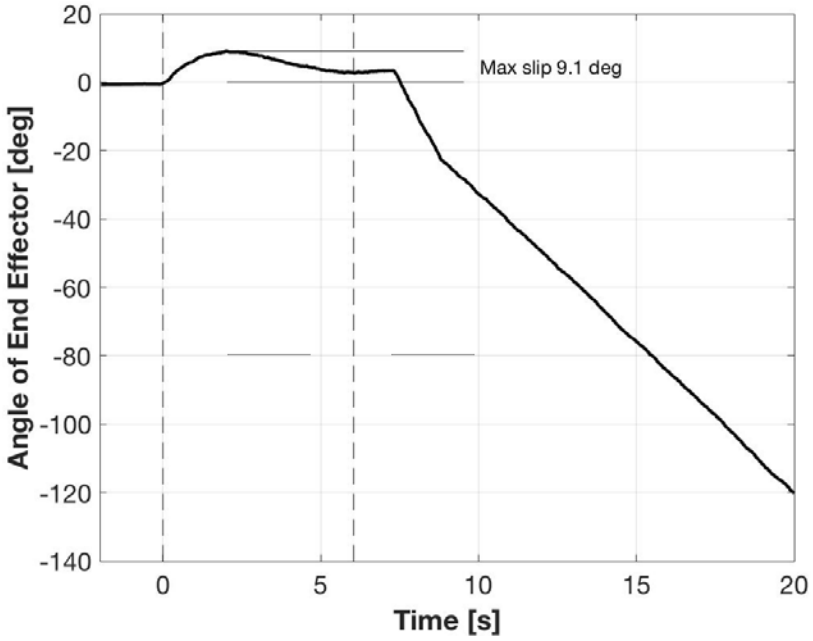

Figure 103. Self-Toss Slip Angle Sample 4

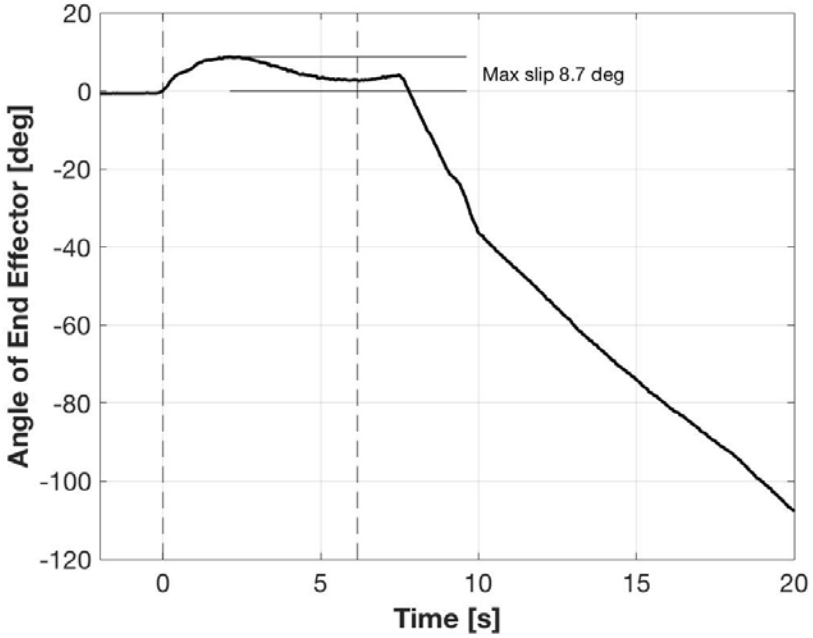

Figure 104. Self-Toss Slip Angle Sample 5

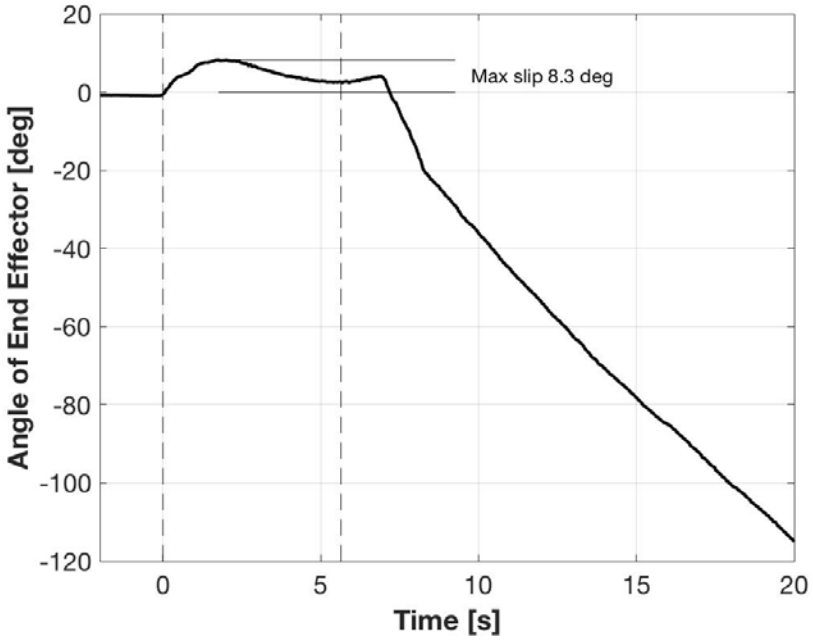

Figure 105. Self-Toss Slip Angle Sample 6

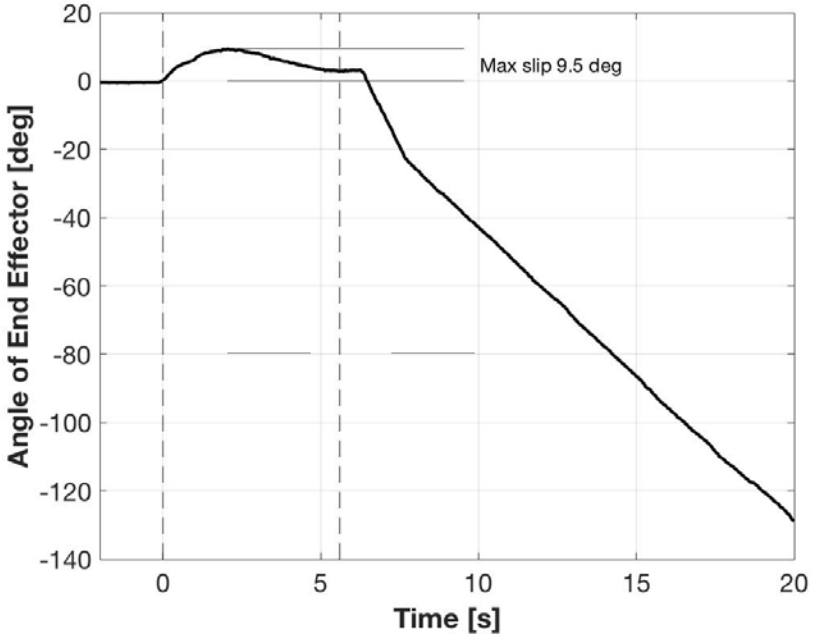

Figure 106. Self-Toss Slip Angle Sample 7

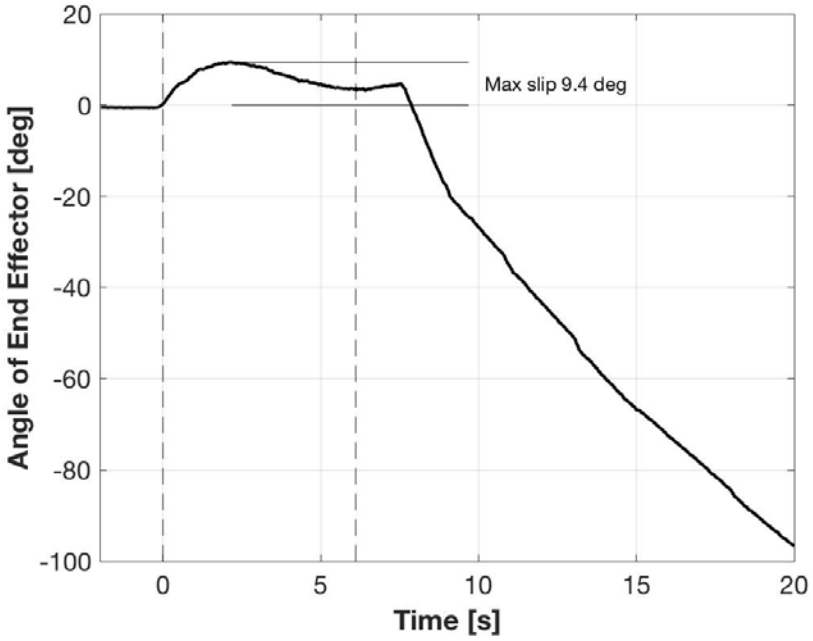

Figure 107. Self-Toss Slip Angle Sample 8

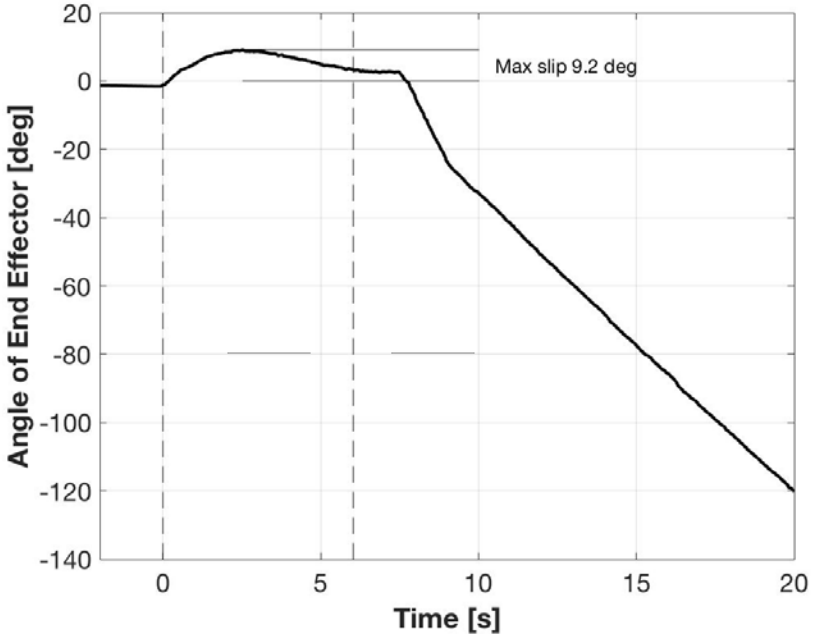

Figure 108. Self-Toss Slip Angle Sample 9

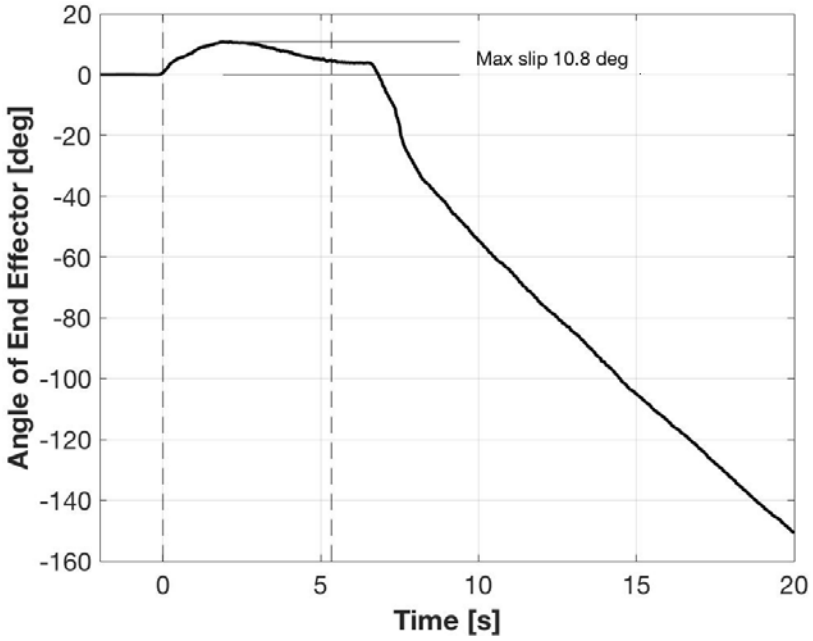

Figure 109. Self-Toss Slip Angle Sample 10

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Mechatronic Works LLC, “Mechatronics,” Accessed Nov. 28, 2018. [Online]. Available: http://mechatronicworks.com/index.php/engineering-services/mechatronics-engineering

[2] K. Ryan, “Mechatronics: A vertical perspective,” *Intech*, vol. 58, no. 1, pp. 22–25, Jan./Feb. 2011. [Online]. Available: login?url=https://search.proquest.com/docview/852729387?accountid=12702.

[3] Types of Engineering Degrees, “Mechatronics engineering degree programs,” Accessed Nov. 28, 2018. [Online]. Available: https://typesofengineeringdegrees.org/mechatronics-engineering/

[4] J. C. Devol Jr., “Programmed article transfer,” U.S. Patent 2988237A, Jun. 13, 1961. [Online]. Available: https://patents.google.com/patent/US2988237A/en

[5] J. R. Surg, “Evolution of robotic arms” Journal of robotic surgery vol. 1,2 (2007): 103–11. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4247431/.

[6] J. Wallén, “The history of the industrial robot,” Linköping, 2008. [Online]. Available: http://liu.diva-portal.org/smash/get/diva2:316930/FULLTEXT01.pdf.

[7] C. Woodford, “Robots,” Explain that stuff, May 9, 2018. [Online]. Available: https://www.explainthatstuff.com/robots.html

[8] L. Kelly, “Unimate: the first industrial robot and why it failed,” History 101, May 1, 2018. [Online]. Available: http://www.history101.com/unimate-first-industrial-robot/1/.

[9] J. Clark, “Rancho arm,” Computer History Museum, May 16, 2003. [Online]. Available: http://www.computerhistory.org/collections/catalog/102654000

[10] D. Samadi, “History and the future of robotic surgery,” Robotic Oncology, Accessed Nov 28, 2018. [Online]. Available: https://www.roboticoncology.com/history-of-robotic-surgery/

[11] NASA Ames, “Robotic arm,” March 23, 2008. [Online]. Available: https://www.nasa.gov/multimedia/imagegallery/image_feature_433.html

[12] J. Camillo, “What’s new with SCARA robots,” Assembly Mag, April 7, 2016. [Online]. Available: https://www.assemblymag.com/articles/93338-whats-new-with-scara-robots

[13] Canadian Space Agency, “The structure of canadarm,” Accessed November 28, 2018. [Online]. Available: http://www.asc-csa.gc.ca/eng/canadarm/description.asp

[14] Canadian Space Agency, “Canadarm made its space debut on the space shuttle columbia (STS-2) 36 years ago,” Twitter, November 13, 2017. [Online]. Available: https://twitter.com/csa_asc/status/930148479866941442

[15] Canadian Space Agency, “About canadarm2,” Accessed November 28, 2018. Available: [Online]. Available: http://asc-csa.gc.ca/eng/iss/canadarm2/about.asp

[16] Canadian Space Agency, “About Dextre,” November 28, 2018. [Online]. Available: [Online]. Available: http://asc-csa.gc.ca/eng/iss/dextre/about.asp

[17] JAXA, “Engineering Test Satellite VII “KIKU-7” (ETS-VII),” Accessed November 28, 2018. [Online]. Available: http://global.jaxa.jp/projects/sat/ets7/index.html

[18] Space Robotics Laboratory, “Japanese engineering test satellite (ETS-VII),” Tohoku University, Accessed November 28, 2018. [Online]. Available: https://www.researchgate.net/figure/Japanese-Engineering-Test-Satellite-ETS-VII-credit-Space-Robotics-Laboratory_fig4_259902832

[19] Jet Propulsion Laboratory, “Flight projects - mars science laboratory,” Acessed November 28, 2018. [Online]. Available: https://www-robotics.jpl.nasa.gov/projects/MSL.cfm?Project=3

[20] Wilde M., Ciarcia’ M., Grompone, A., Romano M., “Experimental characterization of inverse dynamics guidance and control in docking with a rotating target,” Journal of Guidance, Control, and Dynamics, Vol. 39, No. 6, pp. 1173–1187, 2016. [Online]. Available: https://www.researchgate.net/publication/303423862_/Experimental_Characterization_of_Inverse_Dynamics_Guidance_in_Docking_with_a_Rotating_Target

[21] A. Bradstreet, J. Virgili-Llop, and M. Romano, “Demonstration of a propellantless spacecraft hopping maneuver on a planar air bearing test bed,” 2018 AAS/AIAA Astrodynamics Specialist Conference, 2018. [Online]. Available: https://www.researchgate.net/publication/327560912_Demonstration_of_a_Propellantless_Spacecraft_Hopping_Maneuver_on_a_Planar_Air_Bearing_Test_Bed

[22] NASA Ames, “Astrobee guest science guide,” Accessed November 28, 2018. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/irg-ff029-astrobee-guest-science-guide.pdf

[23] Park, I. W., Smith, T., Sanchez, H., Wong, S. W., Piacenza, P., and Ciocarlie, M., “Developing a 3-DOF compliant perching arm for a free-flying robot on the International Space Station,” 2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), 2017, pp. 1135–1141.https://doi.org/10.1109/AIM.2017.8014171.

[24] Raspberry Pi, “Raspberry pi 3 model b,” Accessed November 28, 2018. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[25] Amazon, “USB to RS485 USB-485 Converter Adapter Support Win7 / 8 XP Vista Linux,” Accessed November 28, 2018. [Online]. Available: https://www.amazon.com/RS485-USB-485-Converter-Adapter-Support/dp/B06Y1JTGZX

[26] Pololu Robotics and Electronics, “298:1 Micro metal gearmotor HPCB 12V with extended motor shaft,” Accessed November 28, 2018. [Online]. Available: https://www.pololu.com/product/3056

[27] Pololu Robotics and Electronics, “Dual TB9051FTG motor driver for raspberry pi (partial kit),” Accessed November 28, 2018. [Online]. Available: https://www.pololu.com/product/2761

[28] Robotis, “DYNAMIXEL XH430-V210-R,” Accessed November 28, 2018. [Online]. Available: http://www.robotis.us/dynamixel-xh430-v210-r/

[29] ATI Industrial Automation, “ATI’s Force/Torque Sensors Help Stanford Researchers Cut Ties with Old Flight Testing Methods,” November 28, 2018. [Online]. Available: https://www.ati-ia.com/company/NewsArticle2.aspx?id=180829986

[30] D. Chakravorty, “4 Most Common 3D Printer File Formats of 2018,” All3DP, June 16, 2018. [Online]. Available: https://all3dp.com/3d-printing-file-formats/

[31] Python, “Quick & Easy to Learn,” November 28, 2018. [Online]. Available: https://www.python.org/

[32] MathWorks, “What is MATLAB,” November 28, 2018. [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html

[33] Airwolf 3D, “The guide to polycarbonate 3d printing,” November 28, 2018. [Online]. Available: https://airwolf3d.com/polycarbonate-3d-printing/

[34] Dynamixel, “ ROBOTIS e-Manual v1.31.30,” Accessed November 28, 2018. [Online]. Available: http://support.robotis.com/en/product/actuator/dynamixel_x/xh_series/xh430-w210_main.htm

[35] K. Alsup, “Robotic spacecraft hopping: Application, analysis, and demonstration,” M.S. thesis, Dept. of Astro. Eng, NPS, Monterey, CA, USA, 2018.

[36] G. Parnell, email, Nov. 2018.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California