NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PLANAR ELECTROMECHANICAL ROBOTIC MANIPULATION SYSTEM TO ENABLE UNMANNED SPACECRAFT SERVICING (PERSEUS)

by

Ian A. Hardy

June 2021

Thesis Advisor:	Marcello Romano
Co-Advisor:	Jennifer Hudson
Second Reader:	Stephen Kwok-Choon, Spacecraft Robotics Laboratory

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 2021	3. REPORT TYPE AND DATES COVERED Master's thesis

4. TITLE AND SUBTITLE
PLANAR ELECTROMECHANICAL ROBOTIC MANIPULATION SYSTEM TO ENABLE UNMANNED SPACECRAFT SERVICING (PERSEUS)

5. FUNDING NUMBERS

6. AUTHOR(S) Ian A. Hardy

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)

Rising numbers of aging spacecraft and new missions demand the development of novel approaches to perform various tasks in orbit. Complex servicing and assembly missions have been successfully completed by human astronauts in the past. However, currently available human-rated vehicles are not capable of accessing all relevant orbital locations, nor are there enough assets available for human operators to service all essential payloads directly. If sufficient capability to perform simple tasks remotely could be provided via a robotic manipulator, it may be possible to meet the servicing needs of a far greater number of missions at a lower program cost and without requiring risky extravehicular activities.

The aim of this study is to develop a remote-operated robotic system capable of performing relevant tasks when mounted to a planar floating spacecraft simulator operating on an air bearing table. This system, known as PERSEUS, possesses three revolute joints to allow postural redundancy within a large workspace and a rapidly reconfigurable end-effector that enables simulation of various maneuvers for different planar orientations. When mounted to a simulated spacecraft, PERSEUS offers capability to simulate various grapple and hopping maneuvers representative of what may be required to inspect and service a host payload.

14. SUBJECT TERMS	15. NUMBER OF PAGES
orbital robotics, manipulator, autonomous systems	139
	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.

PLANAR ELECTROMECHANICAL ROBOTIC MANIPULATION SYSTEM TO ENABLE UNMANNED SPACECRAFT SERVICING (PERSEUS)

Ian A. Hardy
Ensign, United States Navy
BS, United States Naval Academy, 2020

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ASTRONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2021

Approved by: Marcello Romano
Advisor

Jennifer Hudson
Co-Advisor

Stephen Kwok-Choon
Second Reader

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

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Rising numbers of aging spacecraft and new missions demand the development of novel approaches to perform various tasks in orbit. Complex servicing and assembly missions have been successfully completed by human astronauts in the past. However, currently available human-rated vehicles are not capable of accessing all relevant orbital locations, nor are there enough assets available for human operators to service all essential payloads directly. If sufficient capability to perform simple tasks remotely could be provided via a robotic manipulator, it may be possible to meet the servicing needs of a far greater number of missions at a lower program cost and without requiring risky extravehicular activities.

The aim of this study is to develop a remote-operated robotic system capable of performing relevant tasks when mounted to a planar floating spacecraft simulator operating on an air bearing table. This system, known as PERSEUS, possesses three revolute joints to allow postural redundancy within a large workspace and a rapidly reconfigurable end-effector that enables simulation of various maneuvers for different planar orientations. When mounted to a simulated spacecraft, PERSEUS offers capability to simulate various grapple and hopping maneuvers representative of what may be required to inspect and service a host payload.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

COTS	Commercial-Off-The-Shelf
C++	C Object-Oriented Programming Language
DARPA	Defense Advanced Research Projects Agency
D-H	Denavit-Hartenberg Parameters
EEPROM	Electrically Erasable Programmable Read-Only Memory
FSS	Floating Spacecraft Simulator
HST	Hubble Space Telescope
IR	Infrared
ISAR	Intelligent Space Assembly Robot
ISS	International Space Station
IS-901	Intelsat 901
JST	Japan Solderless Terminal
MATLAB®	Matrix Laboratory by The MathWorks, Inc.
MEV-1	Mission Extension Vehicle 1
MSS	Mobile Servicing System
NASA	National Aeronautics and Space Administration
NPS	Naval Postgraduate School
PERSEUS	Planar Electromechanical Robotic Manipulation System to Enable Unmanned Spacecraft Servicing
POSEIDYN	Proximity Operation of Spacecraft Experimental Hardware-in-the-loop Dynamic Simulator
RS-485	Recommended Standard 485 for Serial Communications
SD	Secure Digital
SDK	Software Development Kit
SMPS	Switched-Mode Power Supply
SRL	Spacecraft Robotics Laboratory
SSRMS	Space Station Remote Manipulator System
STS	Space Transportation System
TORO	Toolset for Orbital Robotics©
USB	Universal Serial Bus

U2D2	USB to Dynamixel Interface
VDC	Volts Direct Current
WiFi	Wireless Fidelity

ACKNOWLEDGMENTS

For my late grandfather, Clyde Allan Dalton. You always pushed me to learn the most and be the best I could be. Memories of your humor during times of challenge carried me through this program. This is for you.

This study would not have been possible without the constant support of the NPS Spacecraft Robotics Laboratory team. Guidance from Professors Marcello Romano and Jennifer Hudson has been instrumental in developing this design and its operating concept. Their expertise was critical in framing a research topic that was simultaneously relevant, challenging, and interesting. Consistent mentorship and motivation from Dr. Stephen Kwok-Choon pushed me to solve problems unlike anything I previously tackled. His assistance has been, in a word, indispensable.

I would like to thank my mother and father, whose countless nights of toil and work brought me to this point. Their labors of love and words of encouragement reminded me to work hard and aim high. Their investments in books and building blocks sparked my curiosity, which continues to this day.

My thanks also go to Deborah Hefner Hansen. Her optimism and care have paved the way for me and many of her other students to pursue higher education and strive to serve the community.

Last, I want to give my thanks to God. His daily care has kept me moving forward when studies and demands have worn down my stamina. What constitutes a miracle is certainly up for debate these days, but completing this course of study during such an unusual year is more than sufficient for me.

> behold, thou hast made the heaven and the earth by thy great power and stretched out arm, and there is nothing too hard for thee
>
> —Jeremiah 32:17, KJV

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. ORBITAL ROBOTIC MANIPULATION SYSTEMS

Robotic manipulators, commonly referred to as "robotic arms" are utilized for a variety of engineering applications. Using sophisticated principles of mathematical modeling, these systems often provide greater precision and versatility than human operators can provide.

Terrestrial applications include industrial assembly, diagnostic inspection, and robotic-assisted medical applications. In these applications, robotic manipulators move heavy loads, assemble complex structures, and perform complex diagnostic and repair operations. These operations are often performed with a human in-the-loop but may also be performed autonomously. By developing operating models based on mathematical principles, these systems revolutionized a variety of fields ranging from assembly line-based manufacturing to precision robotic surgical systems [1]–[3]. Traditional practices of manufacturing, assembly, and medicine were directly limited by the locomotive precision and sensory input of human operators. Robotic methods allow for analytical calculation of solution trajectories limited by the characteristics of mechanical and electrical components.

Similar principles apply to the performance of servicing and assembly tasks on-orbit [4]. The orbital environment poses unique challenges for assembling and servicing sophisticated, expensive assets. Moving in constant freefall at great speed necessitates high precision as small errors may result in mission failure or loss of life. As in terrestrial applications, many of these complex orbital operations have historically been performed directly by human operators [5].

Missions to the Solar Maximum satellite [6], Hubble Space Telescope (HST) [7], and International Space Station (ISS) [8] serve as examples of orbital servicing. These missions utilized robotic manipulator capabilities to grapple to and capture satellite payloads in order to allow human crews to service and assemble various structures. A conceptual rendering of the grapple maneuver required to service Solar Maximum is given

in Figure 1. During STS-41C, astronaut crews were required to grapple to Solar Maximum in order to replace damaged attitude control and sensor equipment.

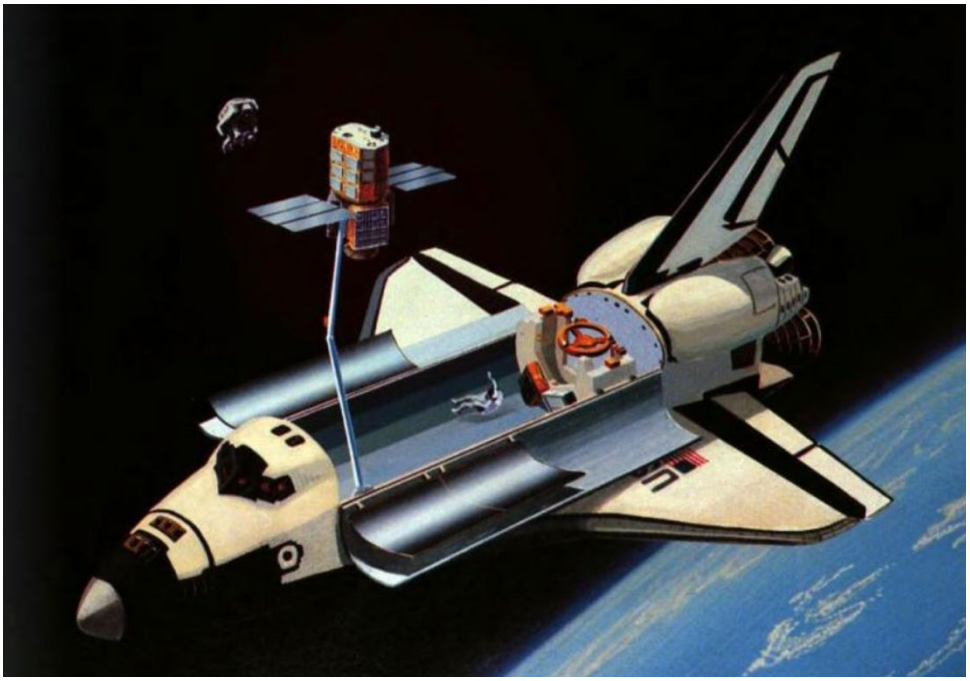


Figure 1. Solar Maximum Servicing Mission Artist's Concept. Source: [6]

For the HST servicing missions, use of a robotic manipulator aboard the Space Shuttle was essential to grapple to HST and allow crews to repair the spacecraft during extravehicular activities. One image of such a maneuver is given in Figure 2. This image depicts the use of the Space Shuttle robotic arm to deploy HST. Later, technical issues with spherical aberration in HST optics would require a series of five servicing missions. Throughout these missions, astronaut crews would perform numerous complex extravehicular activities to install instruments required to correct faults in Hubble's initial implementation [7].

Figure 2. Robotic Manipulator Use During Hubble Servicing Missions.
Source: [7]

In each of these cases, direct human servicing was required to operate robotic equipment to intervene by correcting errors and installing new equipment. These operations were vital to extend the capabilities of expensive space hardware whose useful life would have otherwise expired.

Human-in-the-loop robotic manipulators have been used extensively throughout the ISS program. Assembly was completed through the operation of the Mobile Servicing System (MSS) and Space Station Remote Manipulator System (SSRMS) to manipulate and install components ferried to the Station by the Orbiter [8]. These systems were also used to maneuver astronauts about the Station during extravehicular activities in order to access various components in need of servicing. The manipulators used for the Space Shuttle and ISS programs are highly capable but have been produced at costs far exceeding the budget of most missions. If sufficiently capable robotic systems could be developed to inspect and service host spacecraft without a human crew present, far greater numbers of missions could potentially be serviced. This would extend mission lifetimes of more assets and further decrease overall cost of providing capability for extended periods of time. Several

system concepts have been developed to evaluate techniques required to perform servicing tasks without the presence of a human crew. These include large systems such as: the Mission Extension Vehicle 1 (MEV-1) [9], DARPA Phoenix project [10], and CubeSat form factor concepts such as the United States Naval Academy's RSat spacecraft [11] and ISAR robotic testbed [12]. Remotely operated spacecraft have already been shown to possess capability to perform servicing tasks for real satellites in orbital environments. One such example is the successful rendezvous and docking of MEV-1 to the Intelsat-901 spacecraft as depicted in Figure 3 [9].

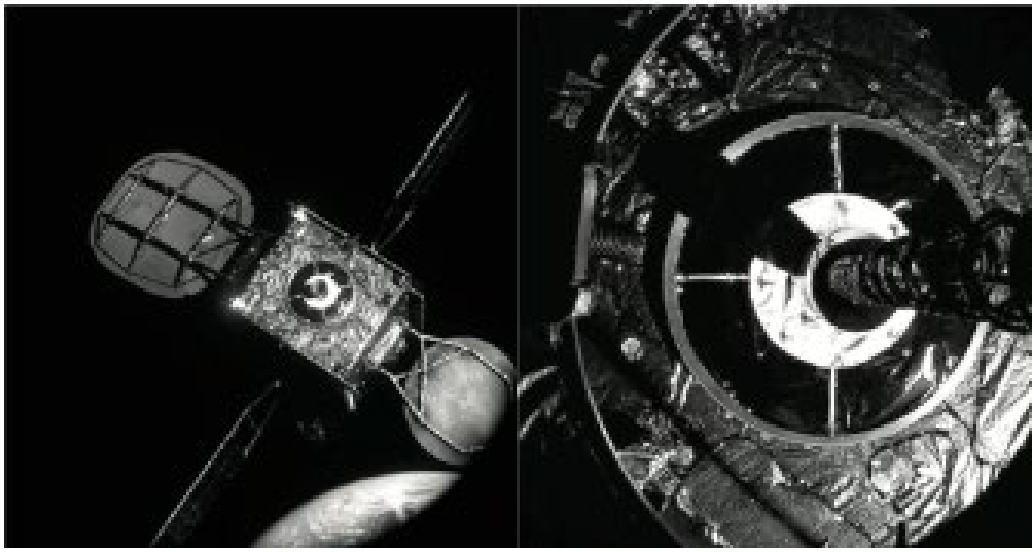


Figure 3. MEV-1 Grappling to IS-901 Satellite at GEO. Source: [9]

In order to extend mission life or assemble structural components, a manipulator with sufficiently complex workspace to allow a suitable range of orientations to inspect host spacecraft and reach relevant portions of host spacecraft topography to enable completion of required tasks [13]. Many spacecraft contain sensitive optics or large deployable surfaces that could be affected by exhaust from conventional maneuvering thrusters. Servicing spacecraft could protect these sensitive systems by maneuvering through the actuation of a robotic manipulator system instead of using thrusters. To explore the utility of such systems and develop the maneuvering techniques required for these operations, the Naval Postgraduate School (NPS) Spacecraft Robotics Laboratory (SRL)

has developed PERSEUS, a self-contained robotic manipulator used to augment the capability of existing Floating Spacecraft Simulators (FSS).

B. MANEUVER SIMULATION AND TESTING

1. Planar Floating Spacecraft Simulator

Simulation of orbital robotic operations is made difficult by the gravitational acceleration incident to a terrestrial laboratory setting. Nevertheless, maneuvers relevant to spacecraft servicing can be simulated through multiple methods in both terrestrial environments and in freefall conditions. NPS-SRL currently operates a suite of FSS units which simulate maneuvers in two axes through use of an air bearing table system known as the Proximity Operation of Spacecraft Experimental Hardware-in-the-loop Dynamic Simulator (POSEIDYN) [14].

True to its name, the FSS operates similarly to a generic on-orbit spacecraft with mass, size, and inertia comparable to a large CubeSat or other small satellite. FSS attitude determination is provided by use of a suite of Vicon® IR tracking cameras. Reaction wheels and compressed air thrusters can provide attitude control, with thrusters that can be used for propelling the unit. These thrusters and air bearing surfaces are supplied with compressed air from a composite gas cylinder and series of regulators. The entire system is controlled wirelessly from a test conductor terminal [14]. Position tracking and data collection are performed using a series of reflective infrared tags and a suite of tracking cameras. The FSS frame and structure are composed of additively manufactured polycarbonate, with access ports and mounting holes integrated into the frame for the addition of various test articles used for specific maneuvers. These are used to attach docking probes and cones, robotic manipulators, grapple fixtures, and other devices used to simulate various types of spacecraft and tasks relevant to servicing.

The POSEIDYN testbed provides FSS units with a 13’ x 13’ area to maneuver and conduct simulated operations. Its thickness is uniform across the area with a tolerance of ±0.0005.” This surface is bounded by metal rails to which a variety of surfaces and objects may be mounted for FSS interaction. POSEIDYN is enclosed in a room with reflectivity and lighting conditions controlled through a combination of paint and light sources that

simulate conditions on-orbit. For visualization purposes, an image of an FSS unit with reaction wheel assembly on the POSEIDYN table is included for the reader in Figure 4.

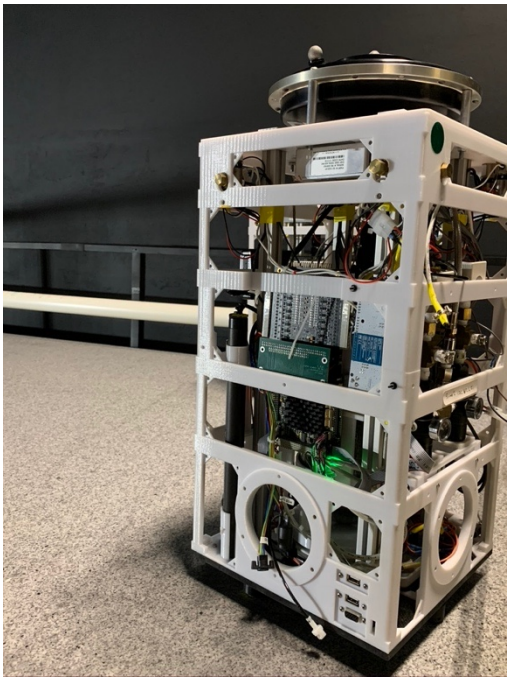


Figure 4. Image of FSS Unit on NPS POSEIDYN Table

When multiple FSS units are operated on the POSEIDYN table with associated tracking and lighting, a variety of complex maneuvers and operations may be simulated and modeled. These include proximity operations of small spacecraft such as grappling and docking. Images of a grappling maneuver and FSS docking hardware are provided in Figure 5.

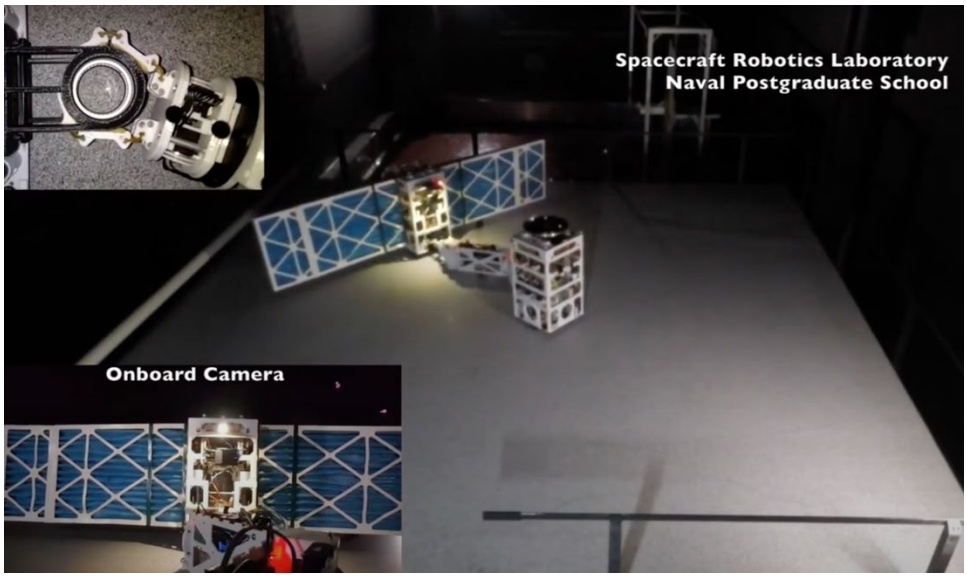


Figure 5. Spacecraft Proximity Operations Simulation with FSS. Source: [15]

Figure 5 represents the use of FSS to model rendezvous, grapple, and capture of a host spacecraft rotating at a constant rate. The capturing spacecraft FSS is augmented with a large planar robotic manipulator and reaction wheels for attitude control. Additional views provided from onboard cameras on both FSS units are also included to illustrate relative positioning of the spacecraft.

FSS units are also used to model rendezvous and docking of spacecraft using conventional probe-and-cone docking fixtures. Figure 6 depicts two FSS in close proximity on the NPS POSEIDYN table, with vehicles possessing the male and female docking features, respectively. These features and hardware can be used to model the final approach as spacecraft prepare for and complete the process of docking.

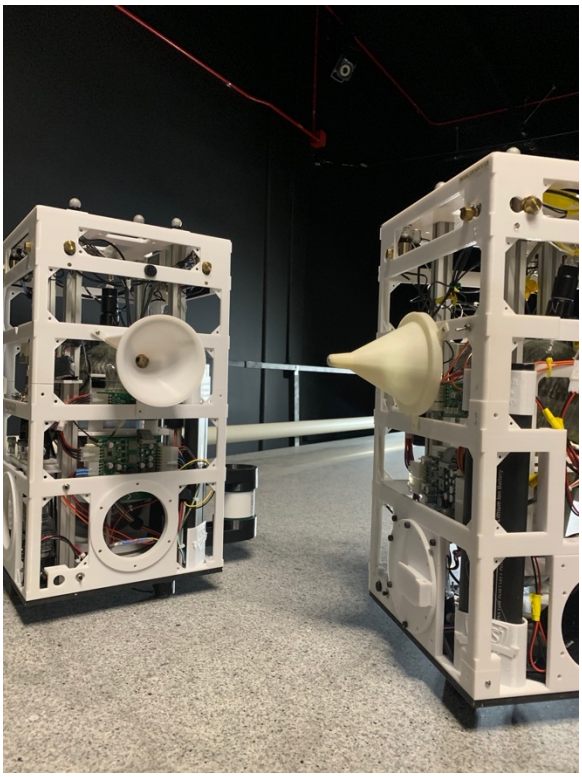


Figure 6. FSS Units Configured for Rendezvous and Docking Test

Previous experiments simulating orbital robotic maneuvers with FSS have demonstrated capability relevant to spacecraft servicing [16]. However, these have required servicing spacecraft to have similar size, weight, and power to host systems. Novel techniques with greater operational flexibility and lower size, weight, and power are necessary to develop viable concepts for spacecraft maneuver and servicing in a cost-effective manner.

2. Astrobee Microgravity Testing

Manipulator performance and spacecraft motion characteristics can also be evaluated using existing hardware on-orbit. NASA's Astrobee free-flyer possesses a robotic manipulator with two revolute joints and a gripper. It also possesses a suite of sensors including cameras, inertial measurement units, and a LIDAR to determine its own position and motion. An example block diagram of Astrobee's robotic and sensing systems is depicted in Figure 7 [17].

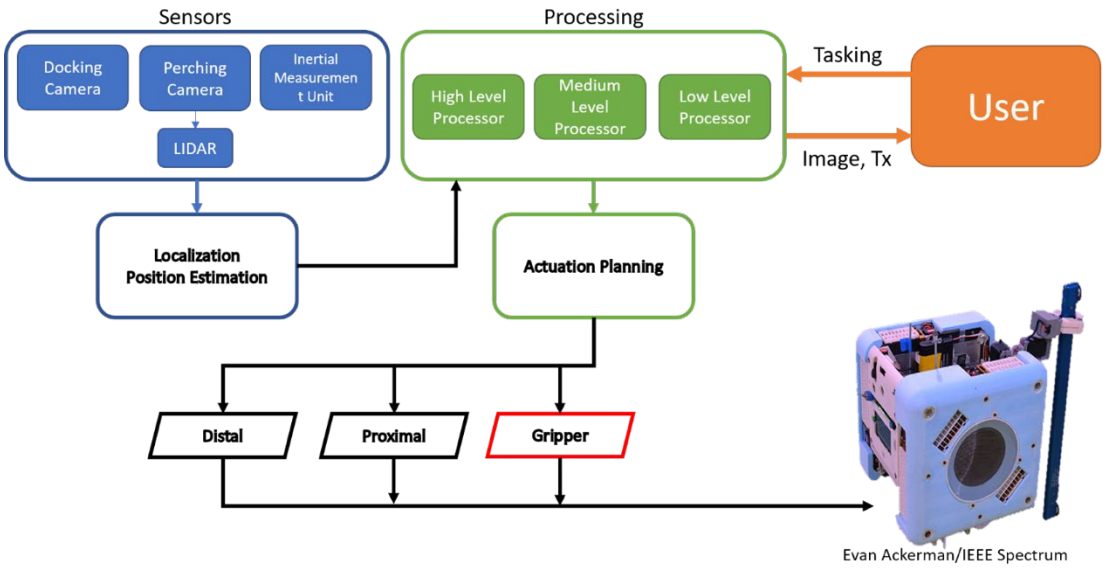


Figure 7. Astrobee Robotic System Block Diagram. Adapted from [17], [18]

Due to its robotic capability and ability to provide localization data, it is a prime candidate for testing and evaluation of the kinematics of actuator-driven spacecraft locomotion. Astrobee may be used to explore how a future servicing spacecraft could utilize a robotic manipulator for maneuvering in close proximity. In order to investigate these maneuvers using Astrobee, NPS-SRL and the NASA Ames Intelligent Robotics Group are conducting a campaign of experiments known as Astrobatics. The aim of Astrobatics experiments is to characterize the kinematics of manipulator-driven spacecraft locomotion in microgravity [19]. These experiments consisted of commanding Astrobee to perform a series of "self-toss" maneuvers by actuating the two revolute joints of the Astrobee end-effector to toss itself from a perched position on an ISS handrail. After actuating the revolute joints of the Astrobee manipulator, the gripper was commanded to open, allowing Astrobee to float through the laboratory spaces on an unconstrained trajectory. Examples of "self-toss" and perching maneuvers are depicted in Figures 8 and 9.

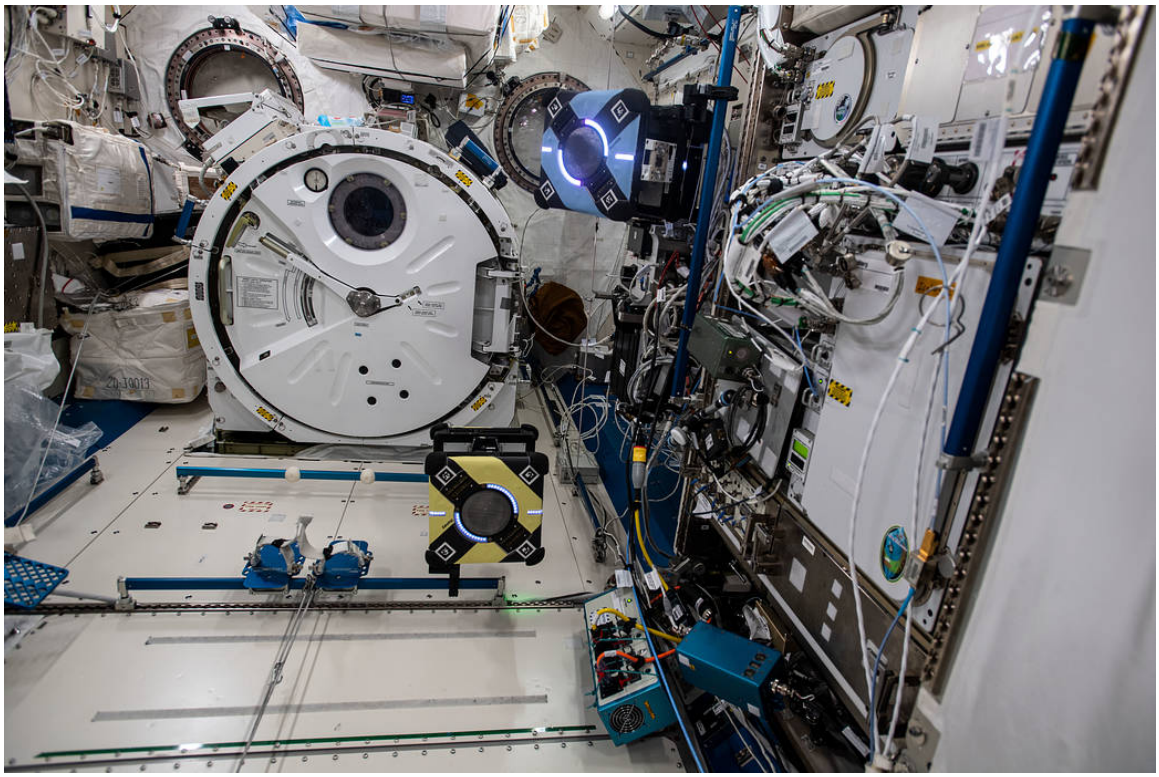


Figure 8. Astrobee Performing Self-Toss Maneuver in International Space Station Kibo Module. Source: [20]

Figure 8 shows two Astrobee robots, both perched on an ISS handrail using robotic manipulators. By actuating the distal and proximal joints of the manipulator, Astrobee can toss itself about the laboratory module, allowing sensing hardware to track the resulting motion in space. A close view of a perched Astrobee is shown in Figure 9.

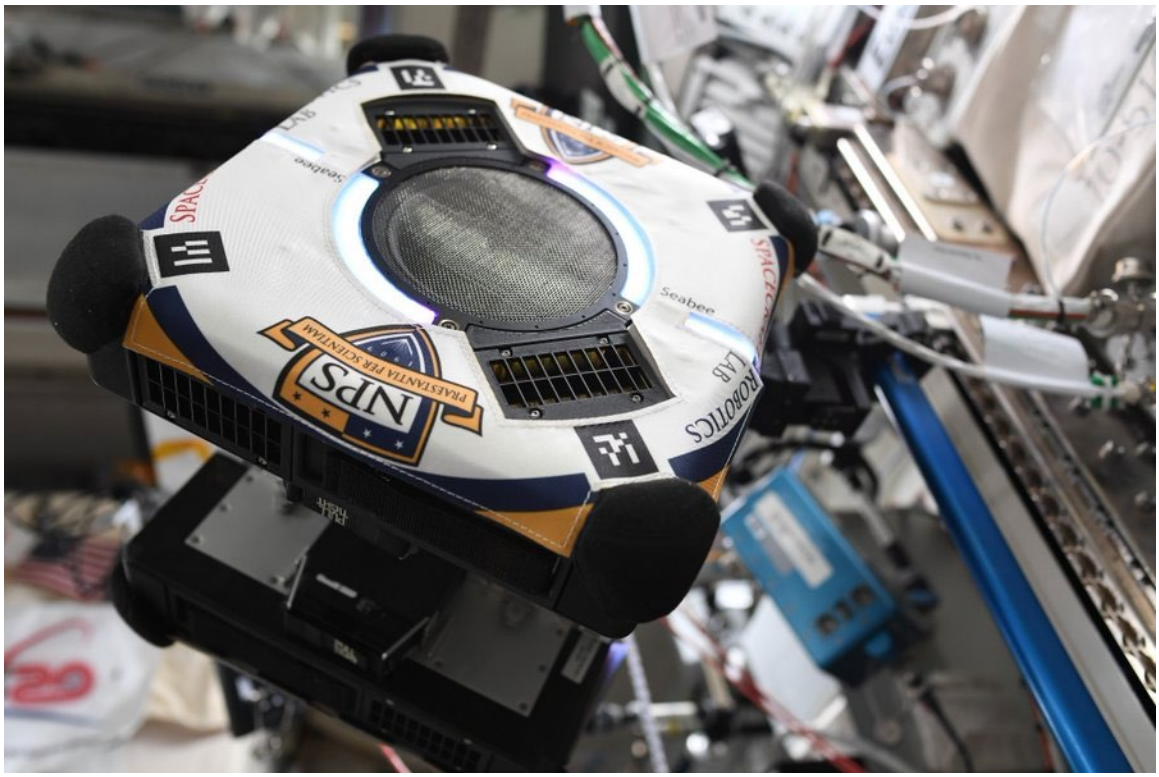


Figure 9. Astrobee Perched on ISS Handrail for Self-Toss. Source: [21]

These experiments gave insight into the general mechanics of maneuvering one spacecraft about a host spacecraft using a robotic manipulator instead of propellant. However, limitations of the joint space of the Astrobee manipulator require the development of an additional manipulator with greater workspace. By incrementing toward a manipulator with postural redundancy, a potentially wider range of candidate maneuvers could be evaluated. Future iterations of spacecraft robotic manipulators could one day be implemented on Astrobee or similar systems.

A system should be developed in a manner that could simulate maneuvers representative of those required by an unmanned servicing spacecraft operating in a microgravity environment. Previous systems have simulated microgravity in three axes, but it is not yet practical to test robotic proximity operations in this manner [22]. However, a series of planar maneuvers can be performed using a precision-machined granite air bearing table and a simulated spacecraft capable of operating in a two-axis simulated microgravity environment.

THIS PAGE INTENTIONALLY LEFT BLANK

II. SYSTEM OVERVIEW

A. SUMMARY

PERSEUS is a self-contained, four degree-of-freedom, three-link manipulator system designed to provide maneuver and grapple capability to Floating Spacecraft Simulator (FSS) units in the Naval Postgraduate School Spacecraft Robotics Laboratory. It is composed of a manipulator chain, end-effector, wireless communication and control equipment, and a structural housing enclosure. This system operates independently of FSS control techniques and does not require any sensor input or commanding from its host FSS. Instead, it acts as a self-contained unit and can simply be fastened to the FSS structure. Once attached, PERSEUS can then be operated wirelessly to execute joint-space trajectory commands to perform maneuvers. A description of the various components of the PERSEUS system will follow. A graphical rendering of the system assembly is given in Figure 10.

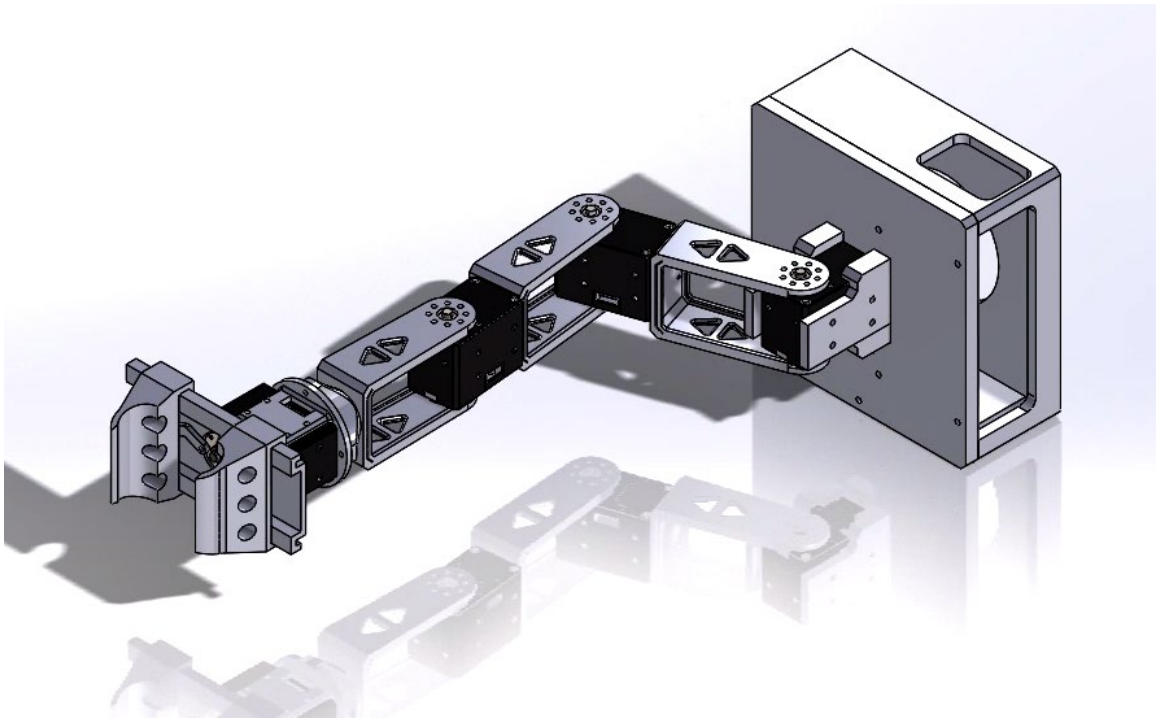


Figure 10. PERSEUS Manipulator Assembly Rendering

B. CAPABILITY

PERSEUS is used to augment a single FSS unit to provide capability for robotic manipulation and actuator-driven locomotion. Its manipulator chain provides a maximum reach of approximately forty centimeters outward the mounting surface and the ability to reach the port and starboard sides of the host unit. While a detailed description of the workspace and operating postures of the manipulator chain will be detailed later, it is important to note that the three revolute joints possessed by PERSEUS allow for redundant posturing within the manipulator workspace.

C. END-EFFECTOR

The PERSEUS end-effector is designed to interface with handrails mounted to the edge of the POSEIDYN table and with grapple fixtures placed on other relevant objects used on the testbed. A notable feature of this design is its ability to be rapidly reconfigured from horizontal to vertical orientations between experiments, requiring only the removal and replacement of four bolts. Left and right portions of the gripper deflect translationally through a slot track that ensures both halves are aligned properly. Opening and closing of the end-effector is accomplished by means of two bent-arm linkages attached to the actuator horn disk and gripper sections. Renderings of these components are included in Appendix A. The inner contact surfaces of the gripper are coated with Velcro® to reduce slipping on contact with rails and other objects. The outward opening of the gripper is slightly wider than the rear portion of the contact surface to allow smooth release after a continuous push from the manipulator.

D. ACTUATION

The manipulator chain is driven by a series of four identical Dynamixel XH430-W210-R actuators. These actuators combine motor, controller, and encoder in isolated units that receive power and command signals from a single connection. The actuators in the chain are connected serially off a single line to reduce cable routing through the structure. Motor commands from the user are converted from C++ code to the actuator communication protocol by an interface board that also provides power to the manipulator chain. Actuators in this system are configured for position control so the individual

manipulator postures may be commanded directly without requiring more complex torque control calculation on the part of the user.

E. INTEGRATION

The system is mated with the host FSS unit by fastening six bolts to the FSS mounting ring and PERSEUS electronics box inner mounting holes. Arduino® and interface boards are attached to the electronics box lid using standoffs and bolts. Actuators are secured to brackets present on the electronics box lid, link frames, and end-effector mounting frame by bolting into surface holes present on the actuator cases. Proximal and distal portions of the end-effector mounting frame are secured together by bolts, with the end-effector track mounting to the face of the fourth actuator in the chain. Bent links are attached to the fourth actuator horn disks and the proximal side of left and right gripper halves after being fed into the track. System input power is fed through the skeletonized frame of the electronics box. Hardline access to the Arduino® and interface boards is available through these same openings. Actuator cabling is fed through the lid of the electronics box and down the manipulator chain. Slack in cables is restrained with Velcro® to prevent tangling or pinching as the structure rotates.

THIS PAGE INTENTIONALLY LEFT BLANK

III. MATHEMATICAL MODEL DEVELOPMENT

A. FRAMES OF REFERENCE

Each joint in the manipulator chain retains a native reference frame which describes motion relative to each actuator. The reference frames of these joints are related to neighboring reference frames by means of transformation matrices which account for differences in angular orientation in space. A reference frame can be related to any other reference frame within same three-dimensional space by matrix multiplication by such a transformation matrix as shown in Figure 11 [23].

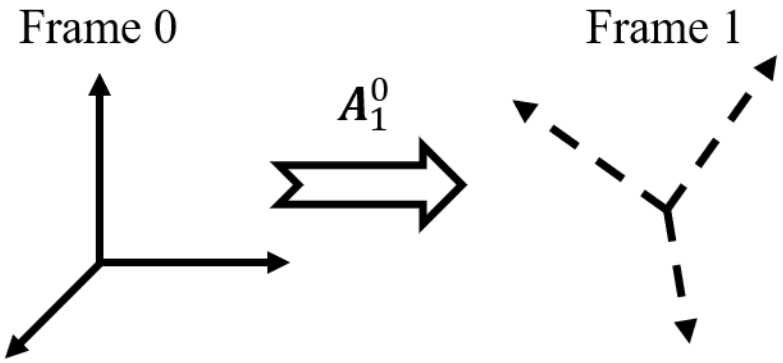


Figure 11. Graphical Representation of a Matrix Transformation

These transformation matrices are constructed by combining the spatial rotations required when moving sequentially from one joint frame to the next. Through this method, it is possible to relate end-effector position in its native frame to the manipulator base frame. This enables the determination of joint orientations required to place an end-effector in a defined base frame position when executing a command or completing a task. Due to the dimensionality of matrices constructed by this method, these transformations can be determined by means of matrix multiplication. An example of such a transformation matrix described by Siciliano et al. is given in Equation 1 [23].

$$\boldsymbol{A}_1^0 = \begin{bmatrix} \boldsymbol{R}_1^0 & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix} \tag{1}$$

where $\boldsymbol{R}_1^0$ represents an angular rotation about an axis required to transition from Frame 0 to Frame 1, and $\vec{o}_1^0$ represents the position vector relating position of the origin of Frame 1 to that of Frame 0. Similar matrices can be constructed for successive transformations to the final frame in the manipulator chain. These are then multiplied in series, resulting in a complete transformation from the base frame to the end-effector frame.

$$\vec{q} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \tag{2}$$

$$\boldsymbol{A}_e^b(\vec{q}) = \boldsymbol{A}_0^b(q_1)\boldsymbol{A}_1^0(q_2)\boldsymbol{A}_2^1(q_3)\boldsymbol{A}_{EE}^2 \tag{3}$$

$$\boldsymbol{A}_e^b(\vec{q}) = \begin{bmatrix} \hat{n}_e^b(\vec{q}) & \hat{s}_e^b(\vec{q}) & \hat{a}_e^b(\vec{q}) & \vec{p}_e^b(\vec{q}) \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

where $\hat{n}$, $\hat{s}$, and $\hat{a}$ are the unit vectors defining the local reference frame of the end-effector and $\vec{p}$ is the position vector relating the origin of the end-effector frame to the origin of the base frame for the manipulator chain.

B. DENAVIT-HARTENBERG (D-H) PARAMETERS

Design and construction of the physical structure of a manipulator system are meaningless without a mathematical model of system operation. Such a model would allow for the analytical calculation of the different states for each joint required to perform maneuvers and thereby allow a controller to command those states. Denavit and Hartenberg developed a method of cataloguing the relevant parameters required to relate the positions of each joint and end-effector in a manipulator chain to a common frame of reference [23]–[26]. These D-H parameters account for the length of each link, the type and limits of joint

actuation, and the orientation of the reference frame of each joint. The D-H parameters for the PERSEUS manipulator are given in Table 1.

Table 1. Denavit-Hartenberg Parameters for PERSEUS Manipulator

Link	$\mathbf{l_i}$	$\boldsymbol{\alpha_i}$	$\mathbf{d_i}$	$\boldsymbol{\theta_i}$
1	0.1m	0	0	θ_1
2	0.1m	0	0	θ_2
3	0.1m	0	0	θ_3
EE	0.1m	$0, \frac{\pi^*}{2}$	-	θ_4^{**}

*Reconfiguration of the End-Effector (EE) from vertical to horizontal orientations introduces a reference frame offset angle of $\frac{\pi}{2}$.

**End-Effector actuator angular deflection affects only the opening and closing of the gripper.

Using the D-H parameters for a given system, the transformation matrix for the end-effector of an actual manipulator may be determined. This allows the computation of joint orientations that allow access to specific areas within the manipulator workspace based on the geometry of the arm and the location of the target. A simple graphical representation of a three-link planar manipulator chain is provided in Figure 12. Note that for a planar manipulator only $\hat{s}$ and $\hat{a}$ are meaningful parameters, as $\hat{n}$ is equal to the zero vector. The planar nature of the PERSEUS manipulator results in these local frames containing meaningful information only in x- and y-directions.

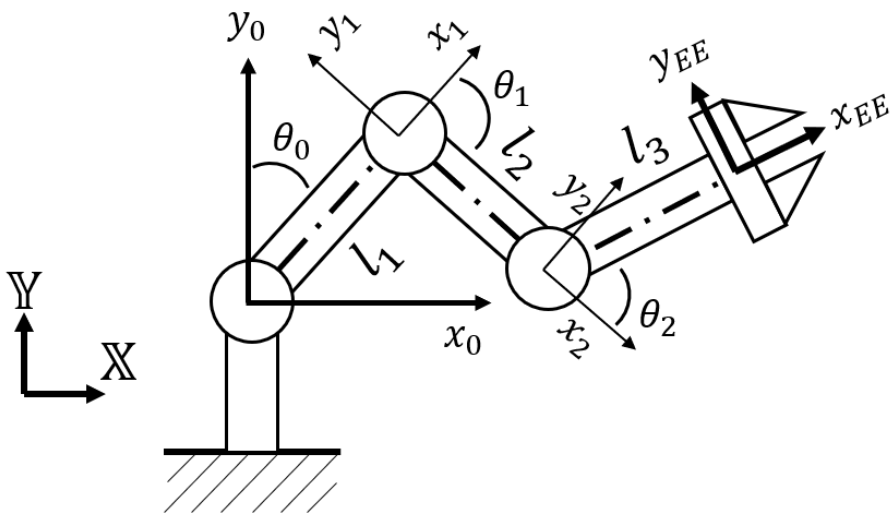


Figure 12. Graphical Representation of Three-Link Planar Manipulator

C. END-EFFECTOR POSITION

As described previously, the PERSEUS manipulator joints only deflect rotationally, with a range of motion from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ radians. Since PERSEUS operates only in one plane, the rotational axes of each joint are always aligned in parallel fashion. This results in a condition where the position of the end-effector is a function of only the constant length of each link and the angular orientation of each actuator in the chain. As such, the position of the end-effector in both the X- and Y-directions is given by the following equations:

$$x_{EE} = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) + l_{EE} \cos(\theta_1 + \theta_2 + \theta_3) \quad (5)$$

$$y_{EE} = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) + l_{EE} \sin(\theta_1 + \theta_2 + \theta_3) \quad (6)$$

where l_i corresponds to the length of each link in the manipulator chain and θ_i represents the angular orientation of each actuator at a given time. Joint orientations were limited according to constraints resulting from physical design of the links in the manipulator chain.

D. WORKSPACE

Given the parameters and equations listed previously, it is then possible to determine all possible positions of the end-effector by mapping the workspace of the manipulator according to possible joint angle commands. Using Equations 5 and 6, possible end-effector positions were calculated by iterating each joint angle with an angular resolution of 0.15 radians and accounting for keep-out zones resulting from physical system geometry. This workspace was developed using a technique described by [27] and is depicted in Figure 13.

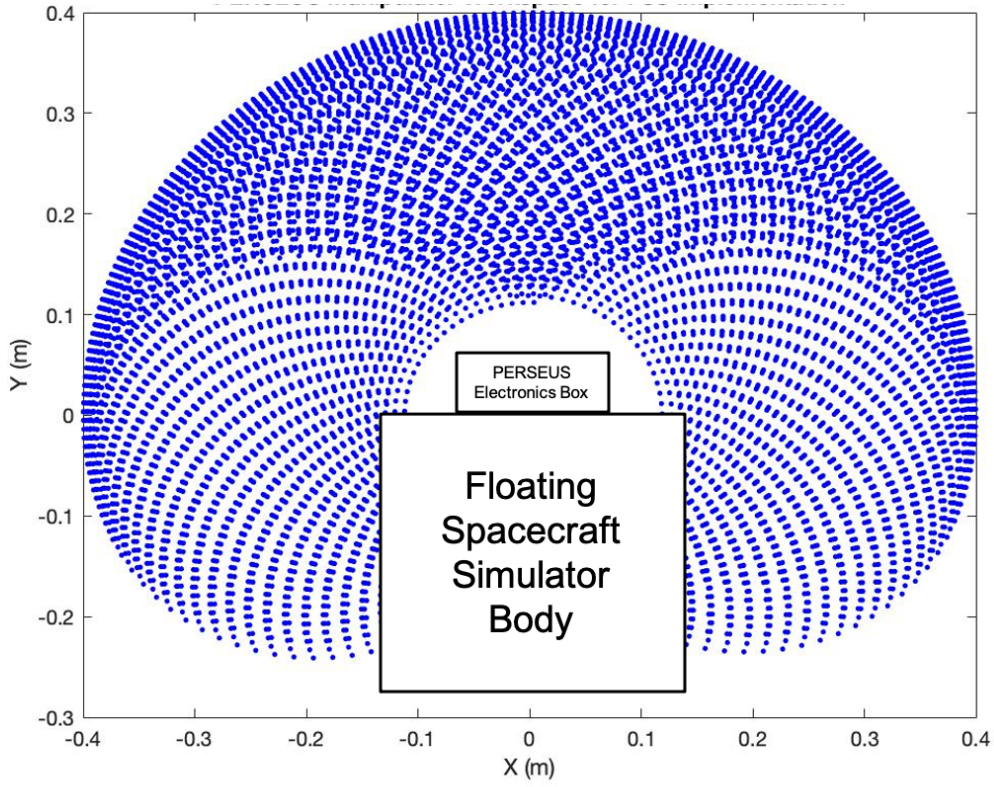


Figure 13. PERSEUS Manipulator Workspace. Adapted from [27]

THIS PAGE INTENTIONALLY LEFT BLANK

IV. KINEMATIC SIMULATION USING TORO

A. OVERVIEW

Representative predictions of system motion during maneuvers performed on an FSS using PERSEUS were desired in order to appropriately evaluate performance during physical system testing. These were obtained by using the Toolset for Orbital Robotics© (TORO) [28]. This software takes into account various rigid multibody system parameters including size, mass, inertia, and degrees of freedom to develop equations of motion to describe system motion. Actual system parameters are then fed into these equations of motion to develop trajectories for each body state and its respective first time-derivative. In this method, it is possible to simulate total body motion resulting from the actuation of one or more joints in a multibody system. By applying size, mass, and inertia data for PERSEUS and FSS units, it is possible to develop predictions as to system behavior during maneuvers evaluated in this study.

B. SCENARIO PARAMETERS

The scenario modeled in TORO© is the use of PERSEUS to perform a planar push maneuver, where the three revolute joints are actuated in such a manner that the end-effector translates outward from the spacecraft body in a straight line normal to the face to which the manipulator is mounted. The maneuver is simulated using the parameters listed in Tables 2 and 3. Degrees of freedom for TORO© simulation listed in Table 3 include position in X- and Y-directions, rotation of the plane, and joint angles for each revolute joint in the manipulator chain. First derivatives with respect to time of these parameters are also plotted. Moments of inertia for bodies about their respective principal axes normal to the plane were calculated using a MATLAB® function implementing Equation 7 over a rectilinear body with length, width, and mass as inputs.

$$I_{TOT} = \sum_{i}^{n} m_i \, r_i^2 \tag{7}$$

Table 2. TORO Simulation Physical Parameters

Body	Length (m)	Mass (kg)	Moment of Inertia (kg-m^2)
Base	0.27	10.18	0.2538
Link 1	0.1	0.14	4.06×10^{-4}
Link 2	0.1	0.14	4.06×10^{-4}
Link 3	0.2	0.14	4.06×10^{-4}
End Mass	0.27	9.88	0.2527

While the masses and moments of inertia for Link 3 and the End Mass are listed separately, for the simulated scenario they are assumed to be one rigid body since an ideal push maneuver would require contact between the two segments to be aligned. Since the angular displacement between these two bodies should be zero, they are instead treated as one rigid body for purposes of estimation. Simulation consisted of segmenting a push maneuver into two phases: a push phase and a coast phase. During the push phase, the spacecraft body acts as the manipulator base with a large mass mounted to the end of the final link. Based on defined initial angular states and velocities given in Table 3, a solution is propagated based on parameters similar to testing performed using the actual manipulator. The final state of this push phase is then fed into the propagator as the initial state of a coast phase where no actuation takes place and where no external forces are present.

TORO© operates by first symbolically computing equations of motion that describe the whole system. These equation of motion parameters are then utilized by a separated script to propagate solution trajectories using a fourth- or fifth-order Runge-Kutta method via the MATLAB® function ode45 [29]. For the sake of brevity, these equations of motion are included as part of Appendix D.

Table 3. Degrees of Freedom and State Parameters for TORO Simulation

Degree of Freedom	Parameter	Initial Value	Initial Time Derivative	Intermediate Value
1	X	0	0	X_{f_1}
2	Y	0	0	Y_{f_1}
3	Plane Rotation	0	0	φ_{f_1}
4	θ_1	$\frac{\pi}{4}$	$-\frac{\pi}{8}$	≈ 0
5	θ_2	$-\frac{\pi}{2}$	$\frac{\pi}{4}$	≈ 0
6	θ_3	$\frac{\pi}{4}$	$-\frac{\pi}{8}$	≈ 0

THIS PAGE INTENTIONALLY LEFT BLANK

V. MANIPULATOR CONTROL

A. JOINT-SPACE POSITION CONTROL

The PERSEUS manipulator is designed for direct control of the orientation of each actuator. Rather than commanding torques or angular velocities for each actuator, the user is able to provide commands for specific manipulator postures by varying the angular position of each revolute joint. This in turn allows for complex maneuvers and posture trajectories to be represented as the combination of a series of scripted, segmented postures. Given an appropriate mathematical model of the manipulator and an understanding of the workspace, maneuvers may be designed to allow the end-effector to reach a target position or follow a defined path. The transition from initial to final end-effector states is then modeled by selecting a number of points along that path and solving for appropriate joint angles based on Equations 5 and 6. This method is used to produce commands for two maneuvers: a so-called "Push Maneuver" where PERSEUS uses its actuators to cause a linear positional deflection of the host FSS unit normal to a perching rail, and a "Swing Maneuver" where the actuators are used to perform a self-toss by rotating away from a perching rail. Related methods of developing methods have been previously developed by Safbom [30].

B. PUSH MANEUVER

Effective use of a robotic manipulator for locomotion about a host spacecraft requires capability to induce translational relative motion. This maneuver will evaluate the kinematic behavior of an FSS unit attempting to distance itself from a perched position. Since the PERSEUS manipulator is capable of placing its end-effector in a given target position using multiple postures, it is possible to move the end-effector outward from the spacecraft body while following a straight line. According to Newton's Third Law, if this line of action crosses the center of mass of the system, the spacecraft will move away from its initial position along the line of action in the opposite direction. In this manner, the robotic manipulator may be used to allow a servicing spacecraft to push itself away from a

surface without inducing a moment about the body. An example of this type of maneuver is given in Figure 14.

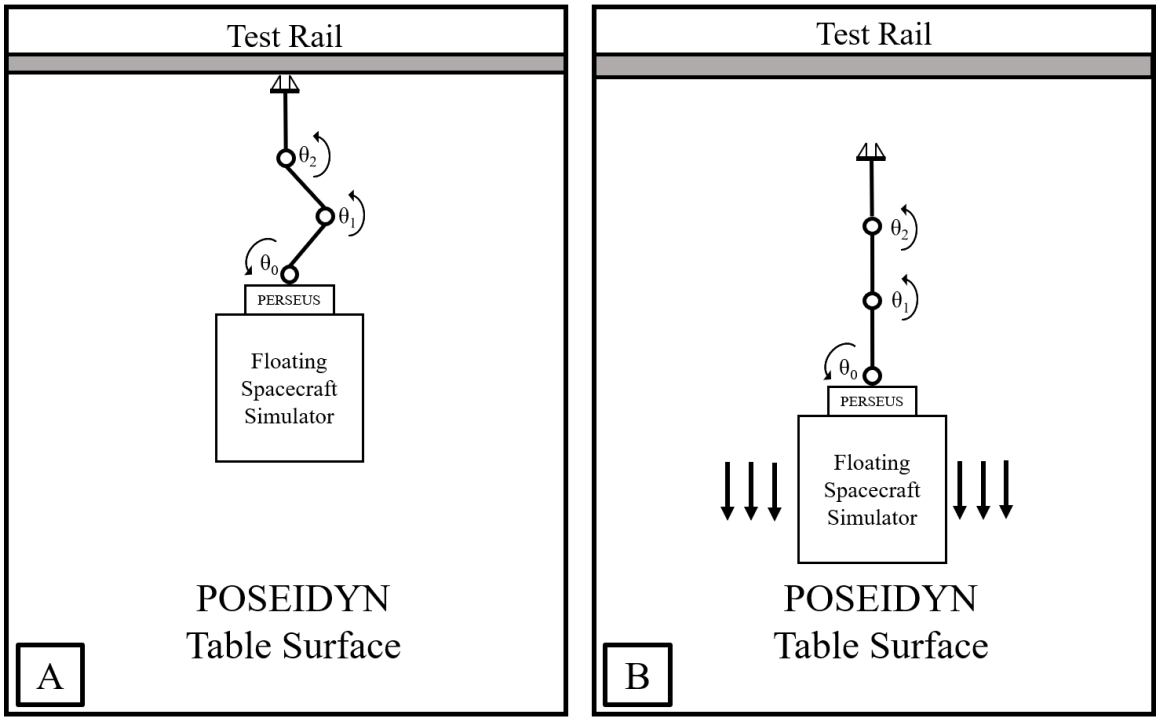


Figure 14. PERSEUS Push Maneuver Concept

C. SWING MANEUVER

Similar to translational motion requirements, servicing systems must be capable of changing angular orientation relative to the host spacecraft. This may be accomplished by perching the manipulator on a rail and actuating the joints to induce a rotation rate. By releasing the end-effector from the rail after this rotation has been established, the motion induced by the manipulator will then cause the FSS to separate from the rail with some rotational velocity. This will allow the FSS to swing from one perched orientation to another position atop the POSEIDYN table and arrive with a different angular orientation. Such a maneuver is depicted in Figure 15.

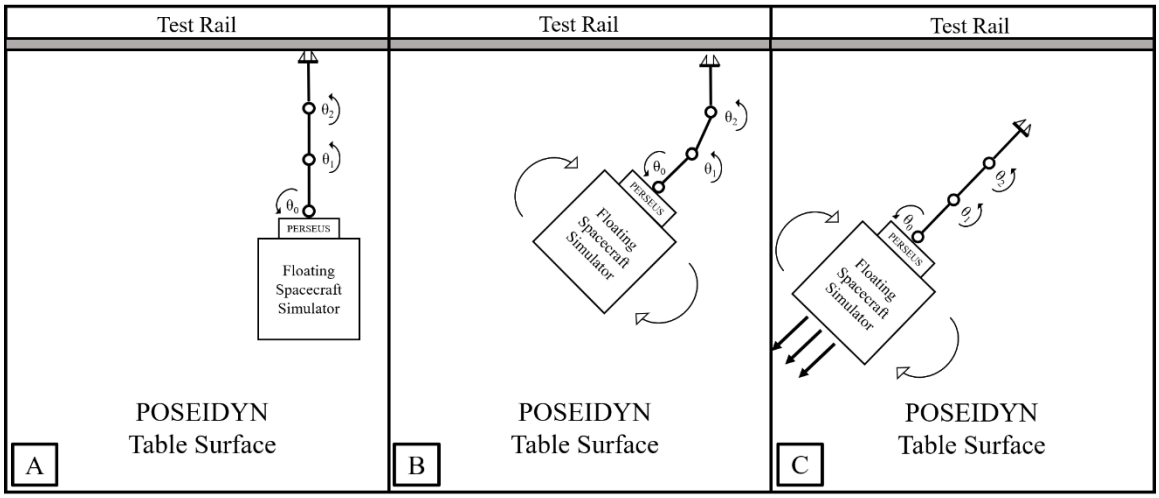


Figure 15. PERSEUS Swing Maneuver Concept

THIS PAGE INTENTIONALLY LEFT BLANK

VI. DETAILED SYSTEM DESIGN

A. ACTUATORS

1. Description

The PERSEUS manipulator is driven by a series of four identical Dynamixel XH430-W210-R actuators produced by Robotis. These actuators, shown in Figure 16, contain a motor, driver, encoder, and gearing within a closed case and are capable of operating via position, velocity, or current (torque) control. Data and 12VDC input power are provided by a four-pin Japan Solderless Terminal (JST) connector. The presence of two JST ports on each case allows multiple actuators to be connected in series for complex manipulator chains.

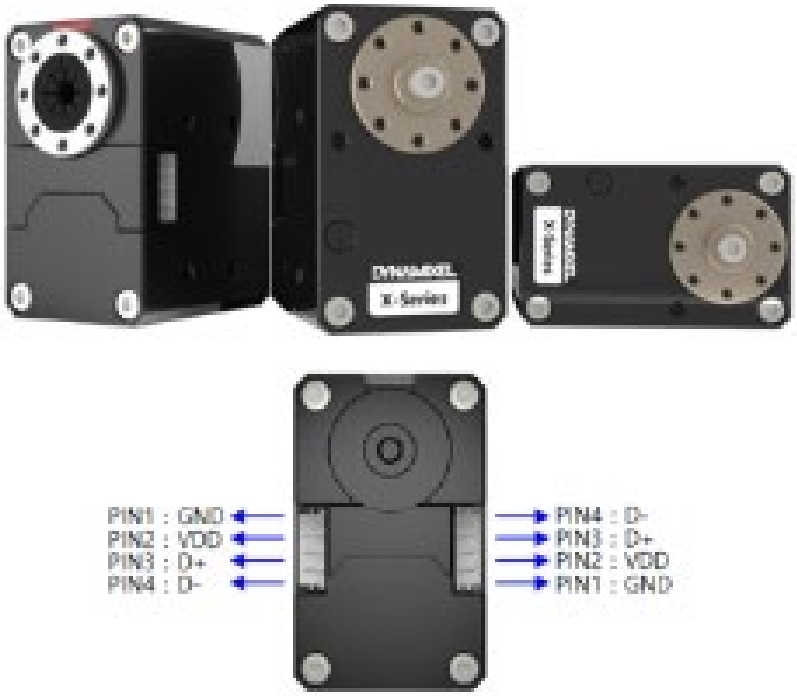


Figure 16. Dynamixel X-Series Actuators. Source:[31]

Mechanical design of the actuators provides for continuous, smooth angular deflection with little backlash. Numerous tapped holes are located about the exterior of the

case to allow for fastening to bases or manipulator link components. It is also possible to pass cables through the interior of the case in order to reduce cable tangling or pinching during operation. Detailed actuator specifications are included in Appendix B.

B. STRUCTURE

1. Description

PERSEUS is composed of two main structures: an electronics box and the manipulator chain. The electronics box is used for interfacing with a host FSS, securing essential power handling and computing hardware, and allowing access to data and power ports. This enables quick system installation and reprogramming as needed. The manipulator chain is composed of three links, with each link being formed by fastening an open hinge bracket to mounting holes on actuator cases. An end-effector assembly is formed by mounting a grooved track along the front face of the fourth actuator. Two gripper halves are inserted into this track and connected to the motor horn by means of two bent linkages. These linkages allow the gripper to open and close along the track by driving the fourth actuator between angles of 0 and $\frac{\pi}{2}$. This end-effector assembly is mounted to a two-piece interface structure that allows the end-effector to be reconfigured from vertical to horizontal orientation by removing a replacing three bolts. An expanded view of the system assembly is provided in Figure 17.

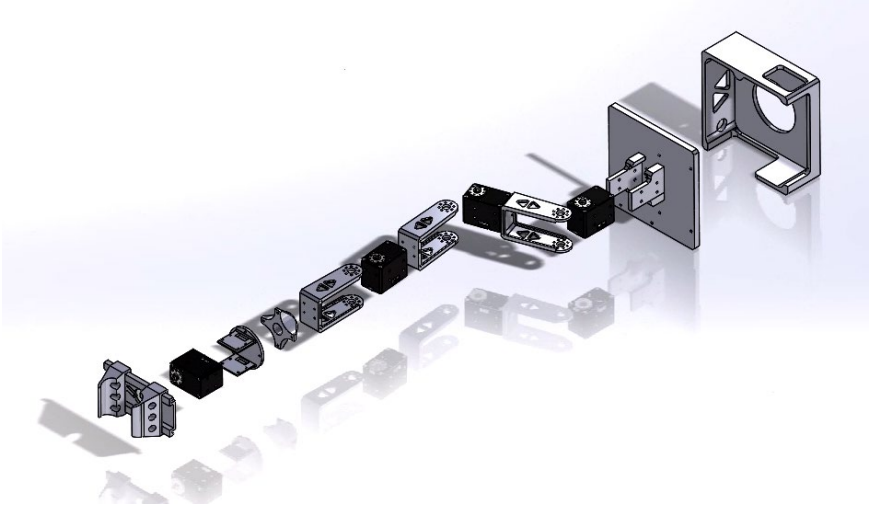


Figure 17. PERSEUS Expanded View

Together, these components form a manipulator with a 40 centimeter reach whose control and power handling electronics fit within a $14 \times 14 \times 5$ centimeter box. As PERSEUS does not rely on the host FSS for control or data handling capability, this can be installed and removed quickly by a single test conductor.

2. Limitations

Structural design of the PERSEUS manipulator has the following limitations:

- Radial loading on actuator horns shall not exceed 40 N
- Axial loading along motor shafts shall not exceed 20 N
- To prevent cable disconnection, actuator positions must not exceed

$$-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$$

In order to ensure operability of the manipulator, structural loads under 1 G were calculated and compared to published torque specifications. According to the manufacturer, each actuator is capable of developing a maximum of 2.6 N-m of torque. Masses of link and end-effector assemblies were measured and catalogued for calculation. Loads due to structural mass and orientation were determined using the following equation:

$$\boldsymbol{\tau}_{Max} = \sum_{i=0}^{3} \mathrm{M}_{str_i} g_0 l_i \tag{8}$$

where torque was represented as the sum of the products of mass, distance from anchor, and gravitational acceleration for each link. These data are shown in Table 4. For added safety margin, link masses were assumed to be lumped masses at the maximum physical distance from the mounting point of the manipulator chain.

Table 4. PERSEUS Manipulator Loading Under 1 G

Link	Mass	Distance from Anchor	Load
1	140g	0.1m	0.137N
2	140g	0.2m	0.275N
3	140g	0.4m	0.549N
Total	420g	-	0.961N

Using this method, it was determined that loads due to the mass of the structure under standard gravity were 37.0% of the published manufacturer maximum torque specification for the zeroth actuator in the manipulator chain. This indicates that the actuators are capable of moving the links of the manipulator throughout its workspace. It is also observed that inertial loading is well within the 40 N and 20 N radial and axial load limitations.

C. FASTENING

To withstand loads resulting from structural mass and manipulator operation, components are fastened with bolts. Mounting holes in polycarbonate components were designed such that bolts would self-tap during fastening. The following fasteners were used in the assembly:

- M2 x 8mm hex socket cap bolts → Actuator horns, hinge brackets, gripper linkages

- M2.5 x 6mm hex socket cap bolts → Actuator case mounting to structure
- M2.5 x 10mm hex socket cap bolts → Manipulator mounting to electronics box
- M4 x 10mm hex socket cap bolts → FSS interface fastening
- Nylon spacers → as needed to ensure level seating

D. ELECTRICAL DESIGN AND NETWORKING

PERSEUS operates using Commercial-Off-The-Shelf (COTS) components and open-source software. The following electrical components are used in the system:

- Arduino® Due Microcontroller (x1)
- Arduino® WiFi Shield (x1)
- 5VDC 1A Barrel Jack Power Source (x1)
- Dynamixel U2D2 Power Hub Board (x1)
- Dynamixel XH430-V210-R Actuator (x4)
- 12VDC 5A Barrel Jack Power Source (x1)
- Dynamixel RS-485 4-Pin Robot Cable, 180mm (x5)

A diagram of power and data connections is provided in Figure 18.

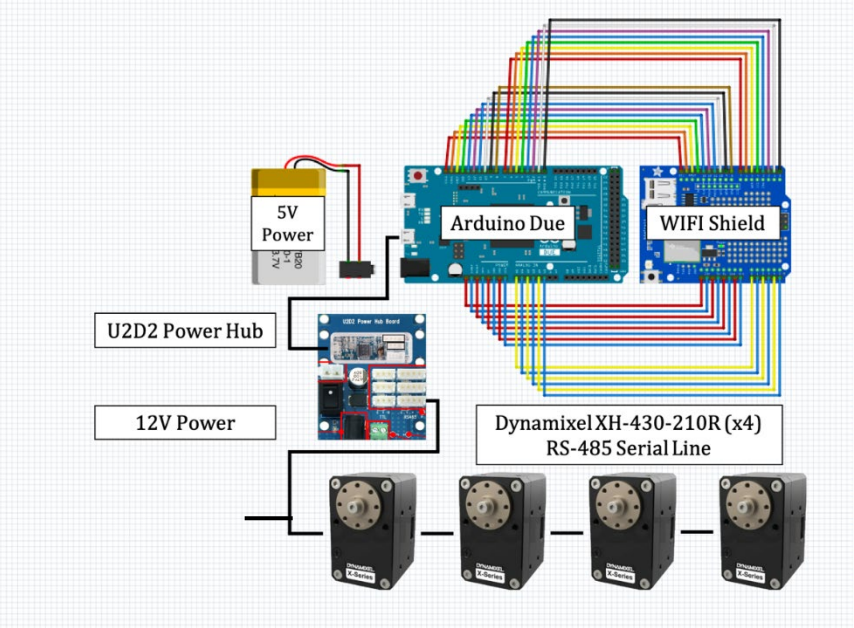


Figure 18. Wiring Diagram. Adapted from [31]–[34]

An Arduino® Due board serves as the system onboard computer. This microcontroller reads actuator position trajectory data from a text file stored on an SD card and transmits commands to the manipulator chain. Pairing with an Arduino® WiFi shield allows new actuator position trajectories to be loaded and saved to the SD card over-the-air in order to execute various maneuvers. Trajectories may also be loaded to the system over a direct connection to a master control terminal. These commands are passed over micro-USB by the onboard computer to a U2D2 communications interface unit that converts integer position values into commands recognized by the computing hardware built into each actuator.

The Power Hub Board provides necessary power handling to convert 12VDC input power into the proper conditions for actuator control. This delivers commands and receives feedback in the form of actuator state and error data. Connections between the U2D2 communications unit and each successive actuator are made using Dynamixel RS-485 4-pin Robot Cables.

Actuators are connected in such a manner that commands, feedback, and power may be sent and received over a single line of RS-485 cables. Signals are processed using

onboard control hardware that allows for position, velocity, current, and other specifications to be assigned using an internal Electrically Erasable Programmable Read-Only Memory (EEPROM). An exhaustive explanation of actuator internal computational capability is not provided here. However, it is sufficient to note that actuator control is performed by reading and writing specified commands to EEPROM addresses corresponding to desired operating parameters.

PERSEUS is designed such that it is compatible with multiple power inputs options. 5VDC power for the onboard computer may be provided over a barrel jack during benchtop testing or using a battery for testing on POSEIDYN. 12VDC power may be provided by either a Molex connector, barrel jack, or Switched-Mode-Power-Supply (SMPS) DC connector. This allows PERSEUS actuators to be powered by tapping FSS onboard power or by implementing a separate battery pack.

The system is designed to be operated over-the-air by transmitting trajectory text files to the onboard computer via WiFi over a local network. In this case, a master test computer opens a communications link with the onboard computer via the WiFi shield, reads command values, and delivers them to the U2D2 communications unit in order to drive the actuators. This may also be performed using a direct cable connection, so long as sufficient slack is present in the cable to prevent interference with motion during testing. A network diagram representing the flow of data during testing is provided in Figure 19.

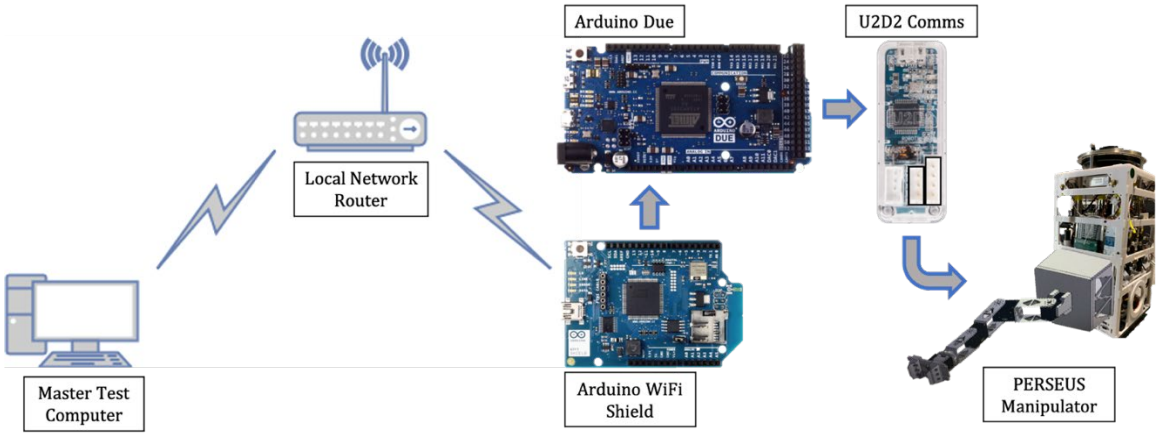


Figure 19. PERSEUS Testing System Network. Adapted from [32]–[34]

E. PROGRAMMING AND OPERATION

System maneuver execution must be performed safely and accurately. Namely, the manipulator chain actuators must be driven in such a manner that the end-effector reaches a target without either impacting its host FSS unit or exceeding angles and angular velocities that could cause cable disconnection or damage to the various link segments and structural components. Further, the system must perform specified operations using an onboard computer with limited memory. In order to ensure safe, accurate operation while minimizing demands on onboard computing hardware, system control is performed in a staged manner. This approach utilizes a higher-memory terminal to complete collision avoidance calculations and limit checks with streamlined, low-risk trajectories sent to onboard hardware for execution. The following subsections will describe the segments of this operating scheme.

1. Trajectory Generation and Verification

Joint space trajectories are generated from user-defined parameters passed to a MATLAB® script. This script queries input regarding the desired number of joint space state vectors, angles for the initial and final state of each joint, and desired end-effector behavior during the maneuver. From these inputs, the program generates a vector of orientation angles for each joint and converts these angles to integer values corresponding to a position count that can be interpreted by each actuator. These joint space orientation vectors are generated such that each actuator will drive linearly from initial to final state at a constant overall angular velocity.

A series of simple geometric calculations are then performed to determine whether or not the generated joint space trajectories are likely to cause a collision with the body of the host FSS unit or impact the manipulator physical structure. Due to the planar nature of manipulator design, this is performed in a straightforward manner by combining the physical size of each segment and the desired orientation. If any of the rigid bodies within the manipulator chain are deemed likely to cause a collision, an error message is displayed to the user indicating the suspected cause of a potential collision. In such cases, the program will not convert the trajectory to a format readable by the actuation program.

In like manner to collision avoidance calculations, trajectories are checked by the generating program to ensure that upper and lower bounds for angular orientation are upheld. To ensure the motion of the various links would not cause a cable to become dislodged, these limits were set at ±90 degrees, with zero degrees occupying the north position opposite the manufacturer label on the front face of each actuator. If a given candidate trajectory exceeds angular position bounds, an error message is displayed to the user identifying the joint or joints predicted to violate the limit.

In addition to these checks, the trajectory generation program determines the behavior of the end-effector throughout a maneuver. The program receives input on whether the end-effector will begin in the open or closed position, if it transitions from its initial open or closed state, and at what point during a maneuver this transition should occur. This is done by selecting a value between zero and one to indicate whether the end-effector opens or closes toward the beginning or end of an actuation sequence. A user may therefore use the program to generate a program where the end-effector grapples to an object or releases itself from a perched position, and at what time it does so.

If a candidate trajectory passes all checks, the trajectory is saved as a text file of user-defined name. This text file is then ready to be passed to computing hardware that may read the file and drive the actuator to the indicated states. A workflow diagram demonstrating the processes occurring in the generation of a maneuver control trajectory is given in Figure 20. A published version of the script is found in Appendix C.

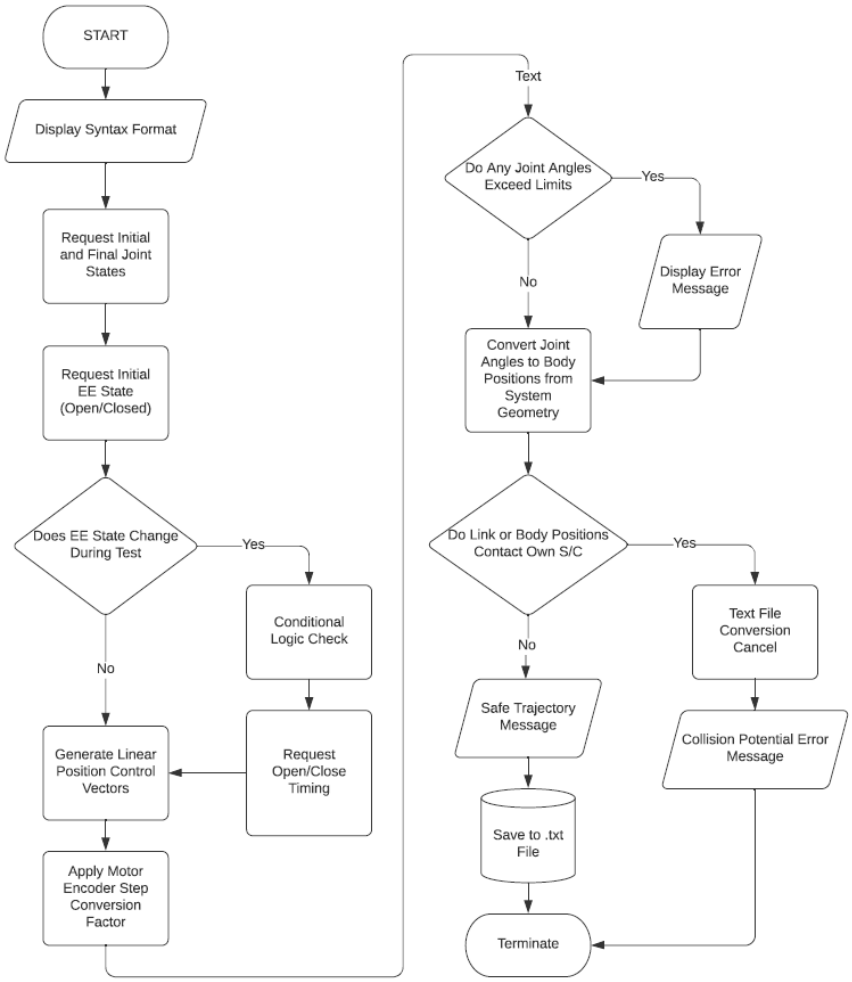


Figure 20. Trajectory Generation Program Workflow

2. Actuation Program

After a trajectory of joint angles required for a given maneuver is generated and saved as a text file it is then passed to the manipulator onboard computer for actuation. For the PERSEUS system, this program operates using a modified C++ version of the Dynamixel Software Development Kit (SDK) [35]. Actuation is performed by reading from and writing to an EEPROM present in each actuator case. Present state data from actuators may be read and stored in order to determine error. Commands may be sent to actuators by building and writing packets to EEPROM addresses corresponding to each desired parameter. Each actuator must be assigned an ID value to allow synchronous serial

command and feedback. With ID set for each actuator in the chain, data may be sent and received along a single RS-485 line.

Upon execution, the actuation program assigns parameters for byte length of commands and defines addresses to ensure commands are written to the proper EEPROM address. The trajectory test file is then read and converted from character format to integer values. Since C++ does not allow direct querying of integer values of array elements, trajectories are converted to a single vector of usable format. The program then proceeds by confirming an established communication link to each actuator, enabling motor torque, and reading the present state of each actuator. After receiving user authorization to proceed, joint space angular position vectors are written to addresses corresponding to goal positions. Present positions are then read and compared to goal position via a checksum for a defined error threshold. Once a given position within the overall maneuver trajectory set is achieved within checksum tolerance the next angular position vector is written to the goal position addresses. Actuation proceeds until the checksum condition is met, and the solution iteratively marches through the loaded trajectory until completion of the maneuver. Once the final angular position command is satisfied, the program terminates. A user may also terminate the program at any time in case of error. It is important to note that this position control configuration does not implement variation in angular velocity. Future versions of this actuation program may implement direct control of angular velocity or motor current corresponding to desired torque.

To illustrate the principles of operation behind the actuation program, a simplified flowchart is provided for the reader in Figure 21. A published version of this script is given in Appendix F.

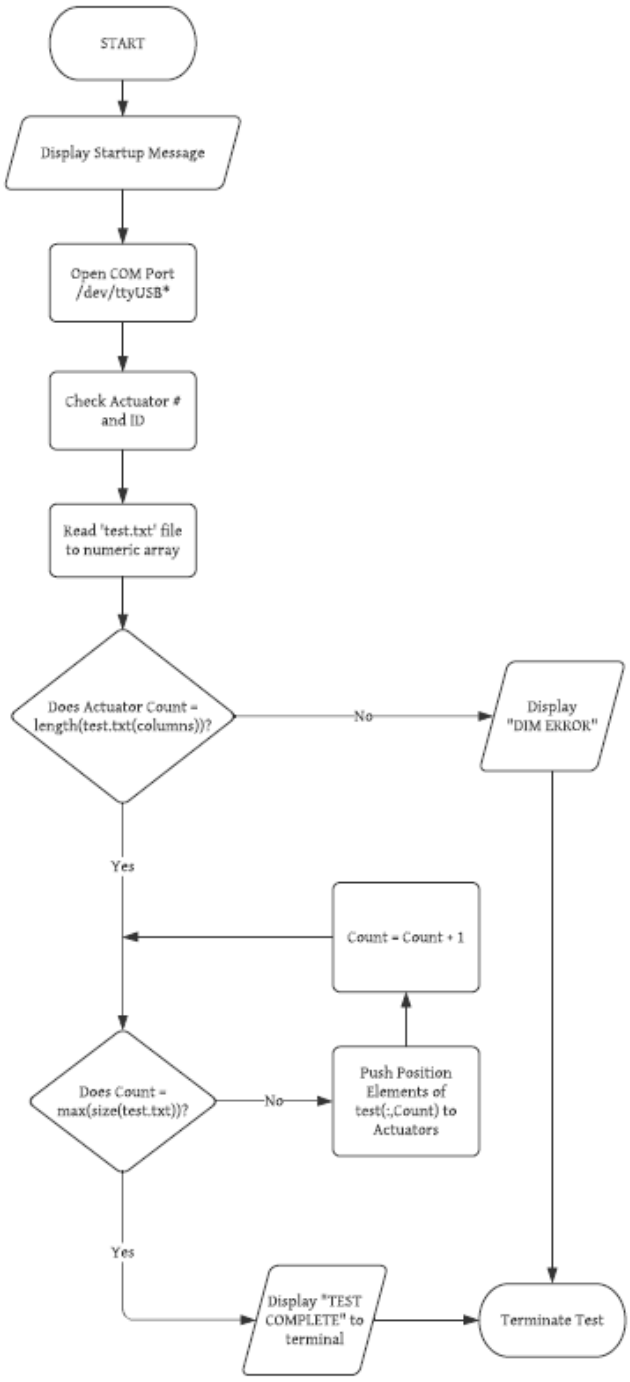


Figure 21. Actuation Program Flowchart

As depicted in Figure 21, the actuation program is designed to proceed iteratively through a candidate trajectory vector by reading elements corresponding to the number of

position command vectors desired. This streamlined approach reduces the demands on the onboard computer.

3. Operations and Testing

Testing and performance evaluation requires coordination of various hardware. Required systems include a master test computer, PERSEUS onboard computer, a data collection and recording terminal, and a localization broadcast computer. For testing to be completed appropriately, these systems are operated in the following manner:

- FSS hardware checks are performed to ensure batteries are charged, pneumatic cylinders are full, regulators and nozzles are functioning properly.
- PERSEUS hardware is checked to ensure fasteners are tight, ample power is supplied, and actuators move freely without abnormal resistance.
- Vicon® IR tracking system target resolution is confirmed.
- FSS localization data is broadcast appropriately to the data collection computer.
- Desired trajectories are loaded to the onboard computer or tethered test computer, as appropriate.

Once hardware checks are completed, test conductors proceed to establish communication link with PERSEUS via WiFi or USB. Once the trajectory text file is accessible to the actuation program, the actuation program proceeds to execute the commanded trajectory. Vicon® tracking cameras record position and orientation of various targets marked using IR-reflective tags. These data are broadcast over the air to a data collection computer which receives and records raw data via Simulink. All experimental data are recorded raw, and data processing is performed after the fact.

A simplified flowchart representative of the testing process is provided in Figure 22.

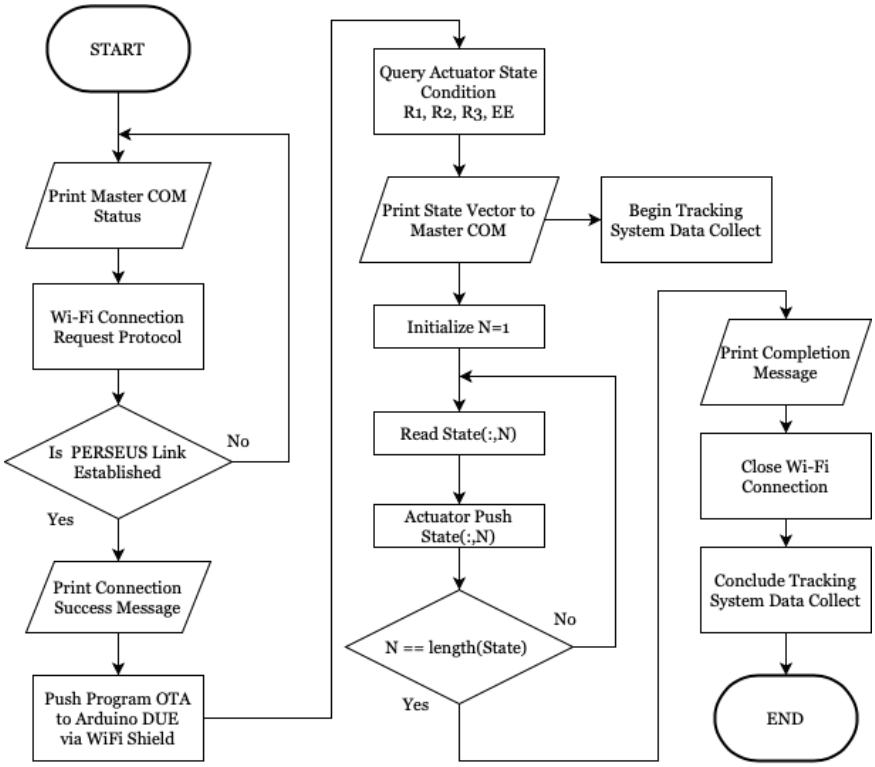


Figure 22. Testing Procedure Workflow

VII. EXPERIMENTATION

A. INTEGRATION

1. Component Preparation

Primary structural components were constructed of additively manufactured polycarbonate produced via fused deposition modeling. Prior to assembly, these parts were inspected to ensure minimal warping occurred during manufacturing. Support material was removed and bearing surfaces were sanded to allow smooth fit. Gripper components were sanded to reduce friction across bearing surfaces of the end-effector track. Through holes for electrical component mounting were measured to meet specifications from engineering drawings.

Actuators were visually inspected with no damage found. Idler horns on front and back faces of each actuator were installed to allow attachment of structural hinge brackets. A back idler horn was not installed on the end-effector joint to allow proper fitting within combined proximal and distal end-effector cradle assembly. The root actuator case was removed to allow through-case wire routing and replaced.

2. Wire Routing

Printed circuit boards for power handling, communications interface, and the onboard computer were mounted to the inside face of the electronics box lid using standoffs. This allowed easy access to benchtop power and USB connection for preliminary testing. For FSS testing, communications and power were routed through ports on the side and top of the PERSEUS electronics box.

3. Assembly

System assembly was conducted in the following manner:

1. Root actuator cables were attached via through-case mounting.
2. The root actuator was fastened to the electronics box lid using M2.5 bolts fed through the root cradle into holes on the bottom and sides of the case.

3. First and second links were constructed by bolting second and third actuators to hinge brackets.

4. The end-effector cradle was assembled by fixing proximal and distal halves using spacers and bolts. The fourth actuator was then fixed to this cradle by bolting the side of the case.

5. The end-effector was constructed by joining each half of the gripper to a motor head bent linkage using M2 bolts and nylon spacers. The track was then mounted to the fourth actuator using two M2.5 bolts placed in holes adjacent to the front motor idler horn. Linkage assemblies were then installed by sliding each half of the gripper down the track and fastening the linkage to the idler horn.

6. Electronics were then installed on the rear face of the electronics box lid.

Photographs of the complete system assembly are given in Figure 23.

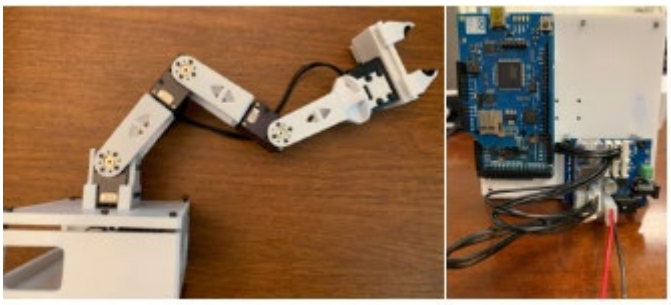


Figure 23. PERSEUS Manipulator Assembly

4. FSS Installation

In order to conduct testing aboard FSS units on the POSEIDYN table, the electronics box was first mated to the FSS interface ring and fastened using M4 bolts. Additional bolts were used to fasten FSS structural frame components to reduce structural vibration during manipulator operation. The PERSEUS manipulator chain was then fastened to the FSS by bolting the electronics box lid tightly using M3 bolts. Power and

data lines were passed through ports on the sides of the box. Images of this integration are given in Figure 24. Future iterations of this design will draw power directly from the FSS EPS and pass data over wireless connection, but for a proof-of-concept power was supplied from an external 12V source via a barrel jack and communications supported by USB cable.

Figure 24. PERSEUS Installation on FSS for Testing

B. BENCH TESTING

To verify proper system operation prior to testing, a series of tests were performed in two primary configurations. Hardware-in-the-loop maneuver simulation was performed by connecting actuators to a host computer and loading a series of test maneuver trajectories. These were then read and executed by the system to ensure anomalous behaviors were not present and that maneuvers with risk of potential damage were not performed. Once proper actuator operation was confirmed, the manipulator chain was assembled, and electrical components connected to a test conductor computer. The manipulator was then fastened to a large steel plate using clamps to reduce likelihood of motor torque rotating the mounting platform. This was done in such a manner that the

components were protected from electrical contact with the mounting platform. A series of maneuvers were then performed to simulate FSS operations, with performance recorded using a video camera. This bench testing configuration is shown in Figure 25.

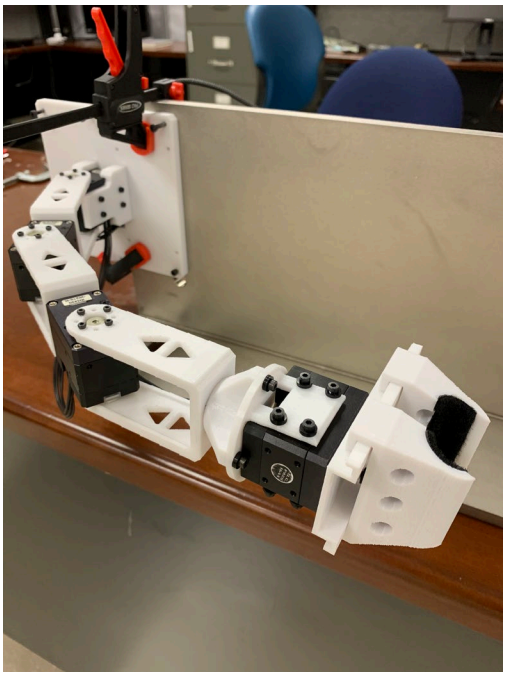


Figure 25. Bench Testing Configuration

C. PUSH MANEUVER

1. Setup

PERSEUS was integrated into a host FSS unit via the method described previously. The IR tracking system was activated, and identification of various system rigid bodies was confirmed. Due to close proximity of mounting positions for IR tracking tags, the manipulator chain could only be resolved into two rigid bodies attached to a rigid body representing the electronics box. Tests were conducted by ensuring all relevant vehicles were at rest and aligned for proper manipulator contact. Since power and data were provided using external cables, test conductors fed additional cable to the system during motion to provide slack and reduce external torque introduced by the tethered configuration.

2. Push Off Fixed Rail

To represent a planar spacecraft hopping maneuver from a body of much greater size and mass, a PERSEUS-equipped FSS was placed adjacent to a rail along the perimeter of the POSEIDYN table. The manipulator was then retracted toward the spacecraft body and the end-effector aligned with the rail. Once contact between the end-effector and the rail was confirmed, test conductors released the FSS unit to float freely on the table. After localization of IR-tagged rigid bodies and a countdown, a command was sent via the U2D2 communications board to execute the maneuver. While in motion, test conductors observed the FSS unit visually and paid out additional cable. Upon reaching the end of the cable, data collection was halted, and results were recorded. Post-processing of data was performed using MATLAB® to characterize translational and rotational motion of the body.

3. Push Off Static Simulated Spacecraft

The next series of push maneuver experiments replaced a fixed perimeter rail with a static, unpowered FSS unit in the interior area of the POSEIDYN table. Similar procedures were followed for setting initial orientations as for tests involving a fixed rail. However, rather than inducing motion from the perimeter toward the center of the table, the PERSEUS-equipped FSS was oriented to move parallel to the outer rail at approximately one-half meter inside the perimeter. This allowed test conductors greater control of cables and allowed longer distance duration maneuvers to be conducted for the same cable length while minimizing torque resulting from the cable. Similar to fixed rail experiments, post-processing was performed to show motion throughout the actuation and coast phases of the maneuver.

4. Push Off Floating Spacecraft Simulator

Lastly, tests were conducted to examine motion following a push maneuver conducted between two powered FSS units. By powering both units and providing air to the hover pads, these tests aimed to more accurately model system dynamics during an actual maneuver in microgravity. These tests were conducted similarly to fixed rail and static spacecraft tests. However, additional care was taken to ensure initial relative

translational and rotation motion of both units was near-zero. One test conductor followed the PERSEUS-equipped FSS, while another stood ready to receive the passive unit and prevent it from contacting perimeter rail structures or other hardware. Overall body motion data were processed and plotted as in previous experiments. These data were also used to determine any motion of the center of mass of the combined system to elucidate the motion influence of external torques resulting from non-zero friction along the table bearing surface, air drag acting on each body, and the use of cables for power and data connections.

VIII. RESULTS AND DISCUSSION

A. BENCH TESTING

1. Single Motor Actuation

Actuation of a single motor over RS-485 was successful. Position control configuration enabled the user to define orientations in terms of motor position counts. The program then continued to build and send packets for target position. As long as the actuator exceeded the ten-count position error threshold, the motor continued to drive. Once this was completed, additional states could be commanded in like manner.

2. Multi-Actuator Synchronous Actuation

Similar to single motor actuation, position trajectories corresponding to multiple motors were read and converted to command state data packets. These were sent synchronously to actuators, and no appreciable time delay was observed between actuation of motors in the chain. However, it was important to note that each actuator stopped rotating after the error threshold was met for each iterative step of the maneuver and started again from zero angular velocity at the next step. This indicated that spin-up and spin-down torques would be generated at the beginning and end of each step. While this effect was not pronounced when actuators were not coupled to manipulator links, the torque developed by each motor was sufficient to cause jitter after assembly.

3. Physical Arm Bench Testing

After successful completion of hardware-in-the-loop simulation, the manipulator chain was assembled for bench testing as described previously. Trajectories were then generated in MATLAB® for the push and swing maneuvers described in Manipulator Control. These were then read by the actuation program and executed iteratively as described previously. Parameters for each of these maneuvers, including initial and final state and end-effector behavior, are given in Table 5.

Table 5. Maneuver Trajectory State Parameters

Maneuver Type	State Command Count	θ_{1_0}	θ_{1_f}	θ_{2_0}	θ_{2_f}	θ_{3_0}	θ_{3_f}
Push	11	45°	0°	-90°	0°	45°	0°
Swing	11	0°	30°	0°	30°	0°	30°

During bench testing, manipulator motion was recorded on video and post-processed to determine timing of various phases of operation. Still images were captured from video and are used here in Figure 26 to depict motion throughout the maneuver. It is important to note that the end-effector opened 0.17 seconds after the maneuver began, simulating release from a perched position on a rail.

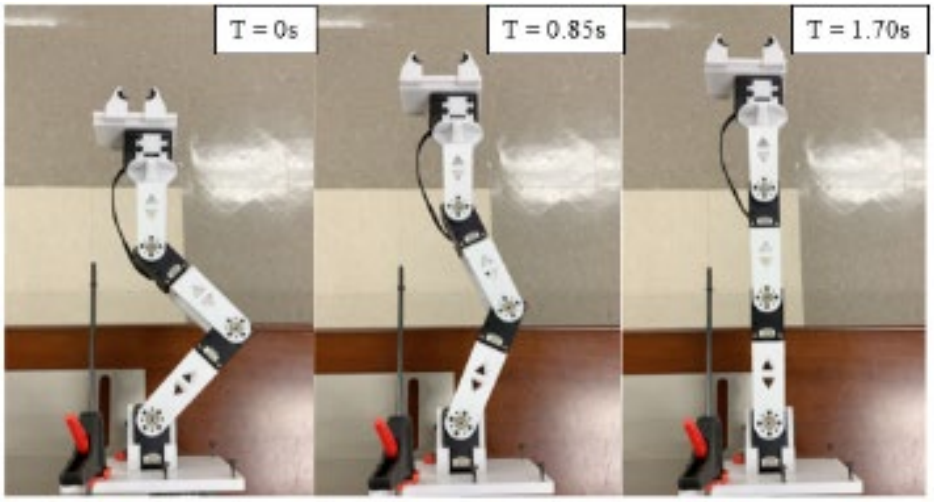


Figure 26. Push Maneuver Bench Test

A similar technique was used to demonstrate manipulator motion during a simulated swing maneuver. The manipulator was capable of driving each revolute joint actuator from a neutral position to a 30° orientation in 1.5 seconds. This is shown in Figure 27.

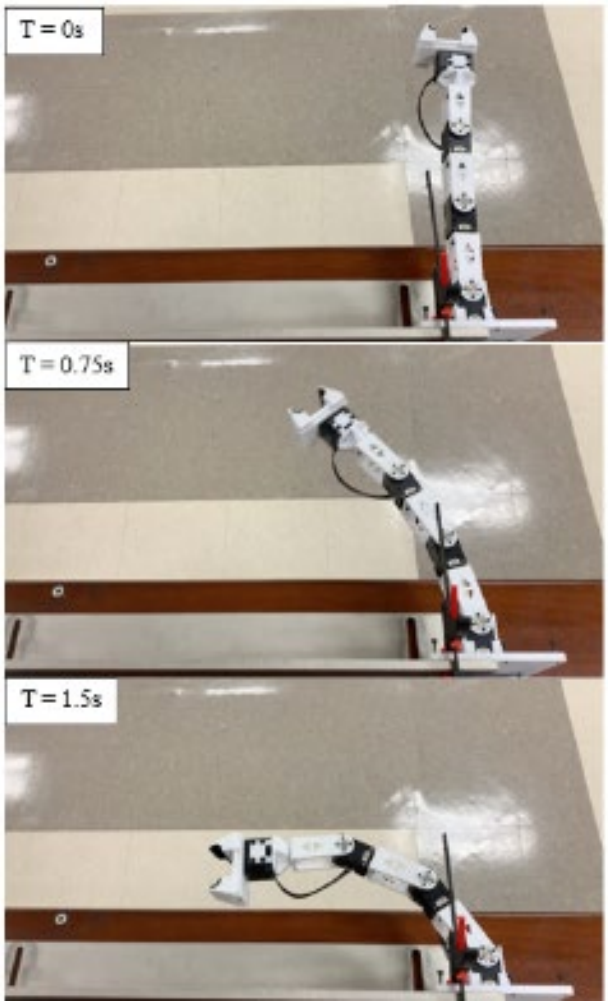


Figure 27. Swing Maneuver Bench Test

It is important to note that bench testing of the physical manipulator showed a more pronounced effect in terms of spin-up and spin-down torques due to iterative position stepping through the commanded vectors for each trajectory. This is due to the torque at each actuator head being transmitted through the manipulator links, causing a greater moment arm within the system for each link. This was partially mitigated by the low backlash characteristics of each actuator, with final resulting motion following the defined path appropriately but with noticeable jitter.

This jitter is not compatible with a system intended for use in an orbital environment as this effect would be more pronounced in microgravity. Future versions of this manipulator should provide a control solution that allows each trajectory waypoint to

be passed smoothly along a given path to reduce the number of times each motor spins up and down. However, for the purpose of demonstrating the concept of position control for planar maneuvers in a two-axis simulated microgravity environment, this may be deemed a sufficient first step.

B. POSEIDYN FSS PUSH MANEUVER TESTING

1. Physical System Data

Results of simulated spacecraft testing of the push maneuver were collected and processed according to methods described previously. Due to constraints on power and pressurized air systems aboard FSS units, a limited number of test runs could be conducted. Of these, multiple runs exhibited poor localization and loss of target tracking. Data from such runs were discarded. Experimental results were tabulated and plotted to demonstrate system behavior. These data are included in Table 6 and Figures 28 through 33. Vehicle 1 refers to the PERSEUS-equipped FSS, while Vehicle 2 refers to the passive FSS used during testing. For rail experiments, a second vehicle was not used. For dynamic push maneuver experiments involving multiple FSS units, motion of the center of mass of the combined two-FSS system was also recorded. Body rotations for FSS units in dynamic push experiments were not plotted.

As shown in Table 6, loss of localization and system complications resulted in two runs of each maneuver type that produced usable data. These issues may be resolved by improving the experimental setup by performing all commanding over-the-air, allowing the interior of the POSEIDYN table to be used and thereby avoiding the boundaries of the tracking system area of regard. However, the data collected still represent actual motion produced by means of actuating a robotic manipulator.

Table 6. Push Maneuver Experimental Results

Maneuver Type	V1Δ*X* (m)	V1Δ*Y* (m)	V2Δ*X* (m)	V2Δ*Y* (m)	CoM Δ*X*** (m)	CoM Δ*Y*** (m)	Body Rotation (deg)	Δ*T* (s)
Fixed Rail Run 1*	0.083	0.300	N/A	N/A	N/A	N/A	31.2	25.0
Fixed Rail Run 2*	0.199	0.466	N/A	N/A	N/A	N/A	61.2	14.7
Static FSS Run 1*	0.661	0.152	N/A	N/A	N/A	N/A	318.4	21.0
Static FSS Run 2*	0.668	0.153	N/A	N/A	N/A	N/A	347.3	25.0
Floating FSS Run 1	0.436	0.136	0.446	0.087	0.043	0.055	N/A	21.0
Floating FSS Run 2	0.111	0.130	0.496	0.075	0.054	0.024	N/A	17.0

*Vehicle 2 motion was only relevant for cases with two floating spacecraft.

**Center of mass motion measurements were used to evaluate experiments where the system center of mass should have remained stationary.

For push maneuvers performed from a static fixed rail, total displacements of 0.31 and 0.51 meters were achieved as shown in Figures 28 and 29. Maneuvers conducted by pushing off a static simulated spacecraft are given in Figures 30 and 31, with total displacements of 0.68 and 0.69 meters. Dynamic push maneuvers between two FSS units produced displacements of 0.17 and 0.46 meters as depicted in Figure 32 and 33. Large deviations between results of maneuvers were observed to result primarily from misalignments in initial mounting, where the PERSEUS-equipped FSS began to rotate between being released by the test conductor and the beginning of the actuation sequence. Tests runs which produced greater displacements were observed to result from initial release conditions where almost no body rotation was present, and where the end-effector was flush and normal to the surface off of which the push maneuver would be conducted. The presence of nonzero external torques resulting from cable mounting appeared to influence motion as well. This was most apparent in dynamic push maneuvers, where the system center of mass was shown to move 5.9 and 7.0 centimeters during the two runs.

This was most likely the result of friction across the table surface and torque resulting from cabling. Although the POSEIDYN table was cleaned prior to testing, these tests were not performed in clean room conditions. It is likely that debris was present on the surface. While test conductors also attempted to provide sufficient cable slack during maneuvers, this may have allowed translational and angular momentum exchange along the line as the cable may have pushed or pulled the FSS.

Assuming that center of mass displacement occurred as the result of constant acceleration from initial position at zero velocity to the final position, Newton's Second Law would predict an external force acting on the system between roughly 40 to 70 millinewtons.

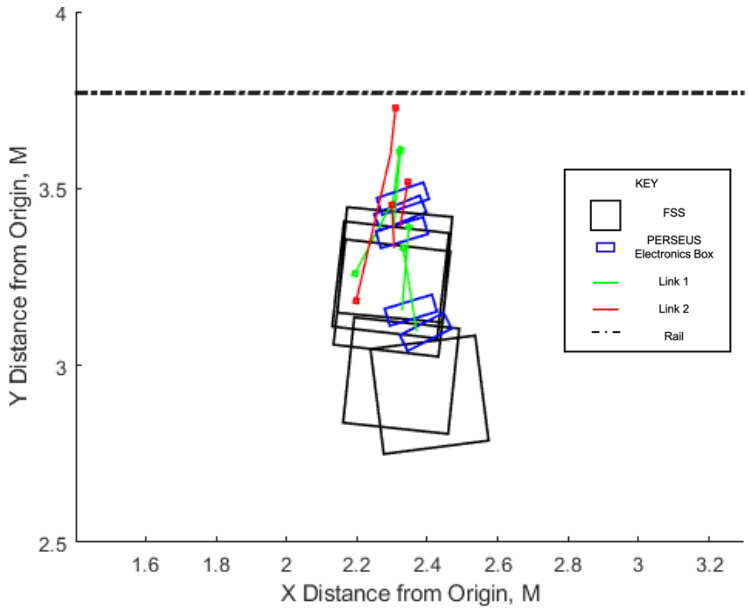


Figure 28. Push Maneuver from Fixed Rail, First Run

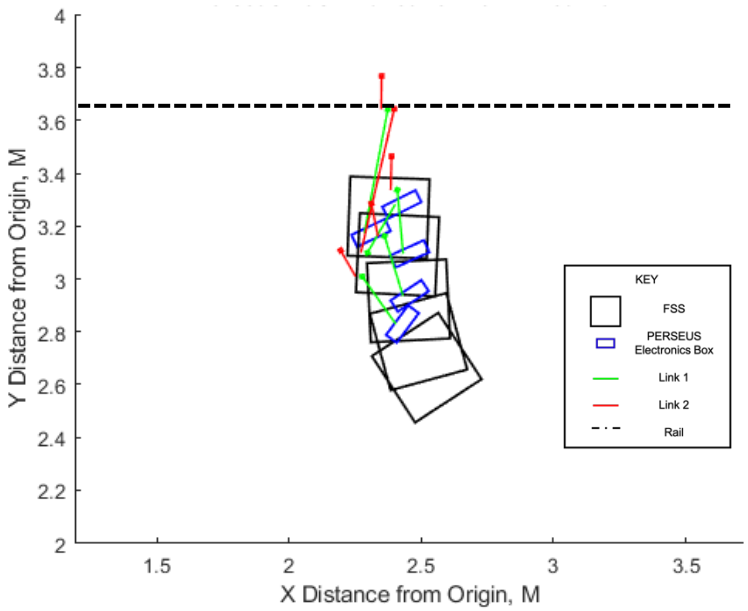


Figure 29. Push Maneuver from Fixed Rail, Second Run

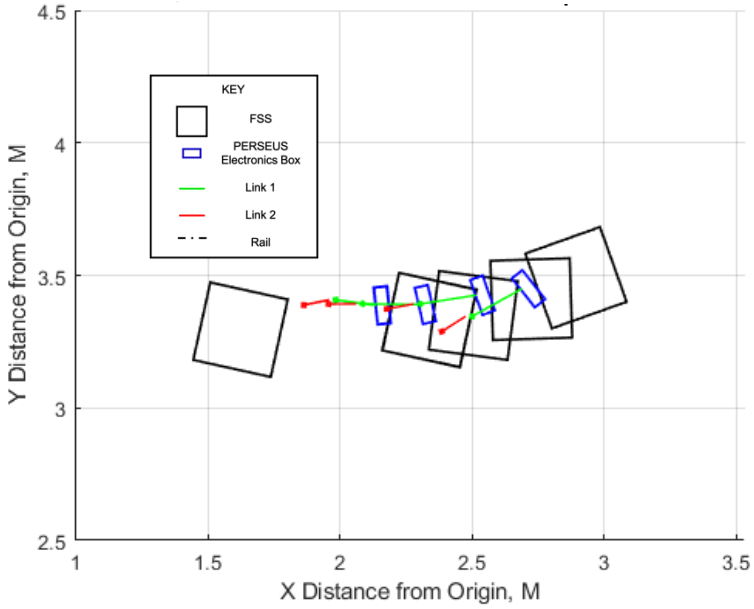


Figure 30. Push Maneuver from Static Simulated Spacecraft, First Run

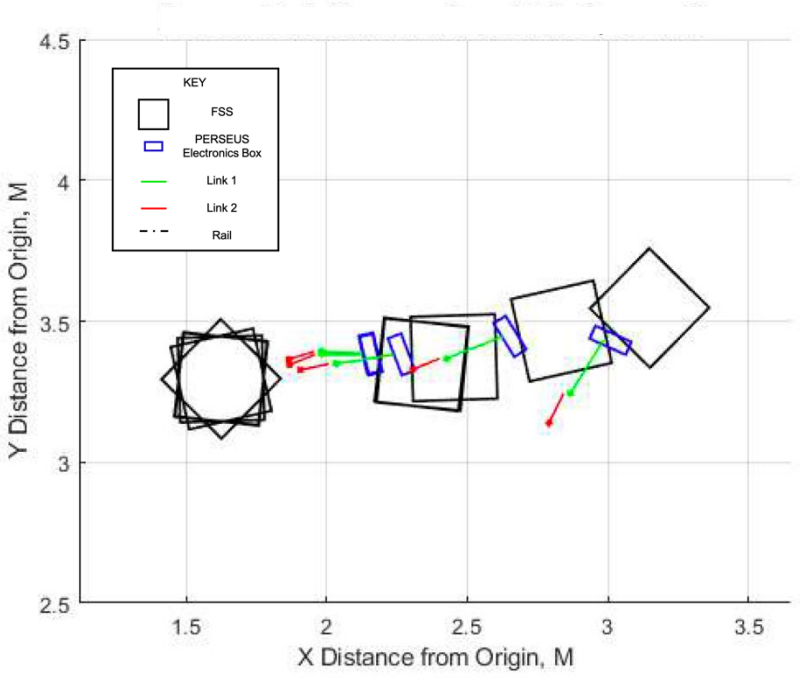


Figure 31. Push Maneuver from Static Simulated Spacecraft, Second Run

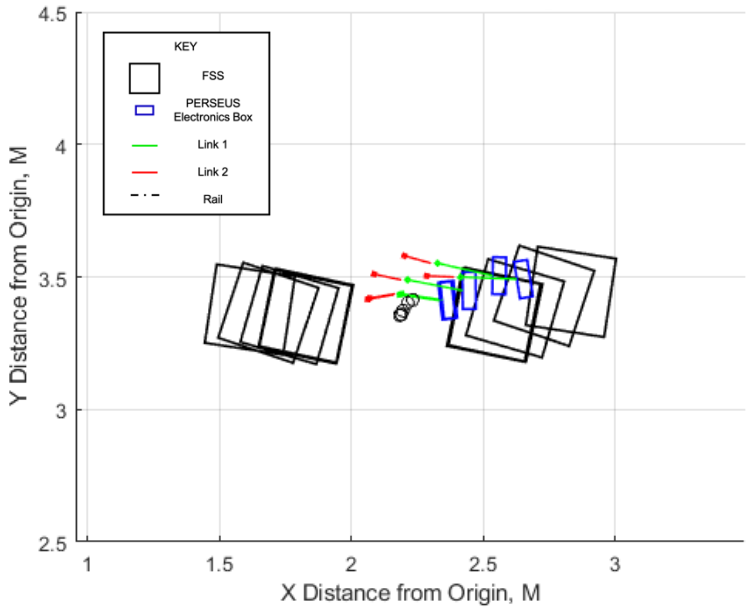


Figure 32. Push Maneuver Between Two Floating Spacecraft, First Run

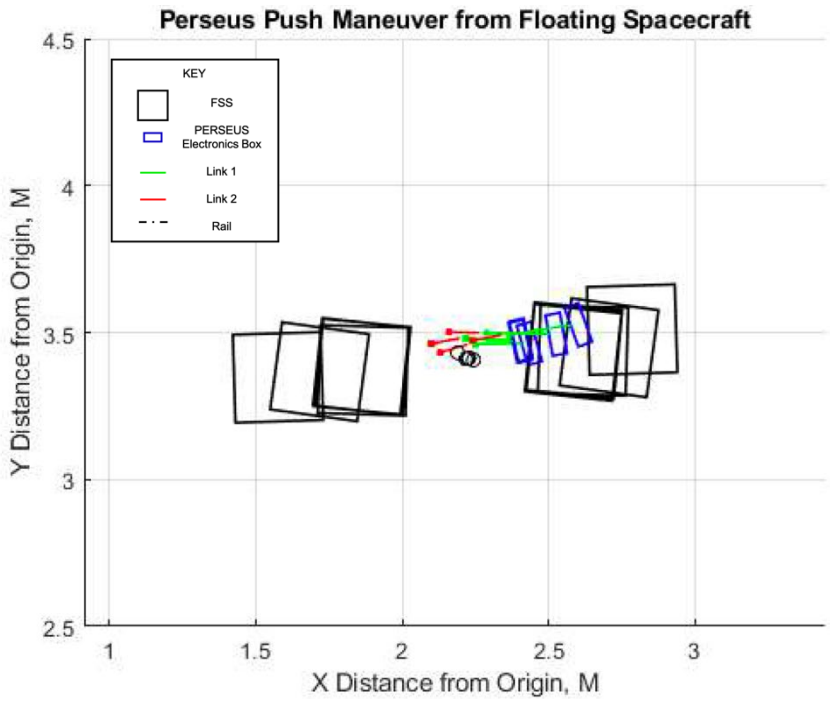


Figure 33. Push Maneuver Between Two Floating Spacecraft, Second Run

2. TORO Simulation

Using the segmented approach described previously, predicted trajectories were generated for the various parameters relating to the system degrees of freedom. TORO© simulation of a push maneuver from a static simulated spacecraft resulted in the following data listed in Table 7. Center of mass motion during these maneuvers, along with joint rotation data are given in Figures 34 through 37.

Table 7. TORO Simulation of Push Maneuver from Static Simulated Spacecraft

Maneuver Phase	CoM ΔX (m)	CoM ΔY (m)	$R1_0$ (deg)	$R1_f$ (deg)	$R2_0$ (deg)	$R2_f$ (deg)	$R3_0$ (deg)	$R3_f$ (deg)	ΔT (s)
Actuation*	0.062	0.016	45	4.47	-90	13.1	45	9.72	2.0
Coast**	0.168	0.008	0	0	0	0	0	0	18.0
Total	0.230	0.024	-	-	-	-	-	-	20.0

*Joint rotation model for actuation phase did not take backlash characteristics into account.

**For the coast phase, joint angles of zero were assumed for each joint with zero angular velocity.

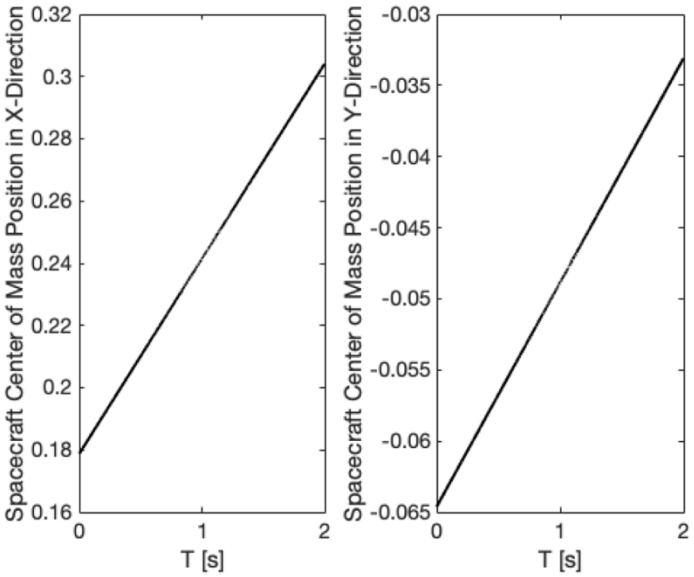
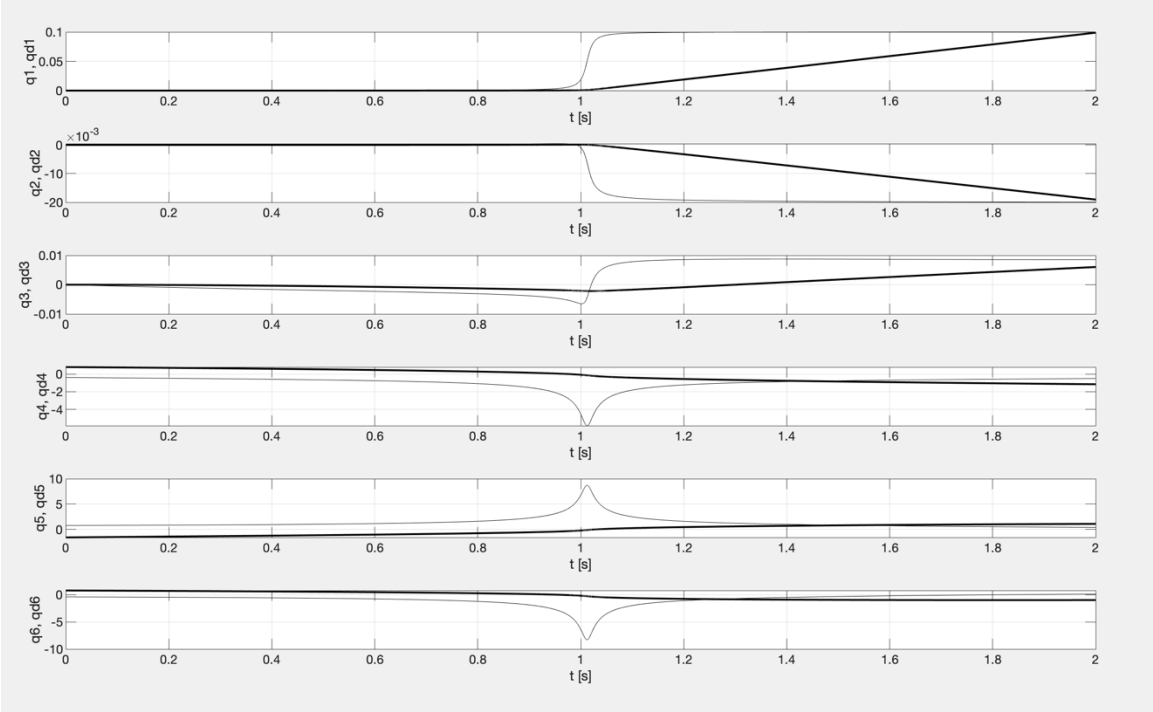


Figure 34. Simulated System Displacement During Manipulator Actuation



*Q1, Q2: Body motion in X and Y (m). Q3: Rotation of the plane (rad). Q4-Q6: Orientation of joints R1, R2, and R3 (rad).

Figure 35. Simulated Motion of System Degrees of Freedom During Manipulator Actuation*

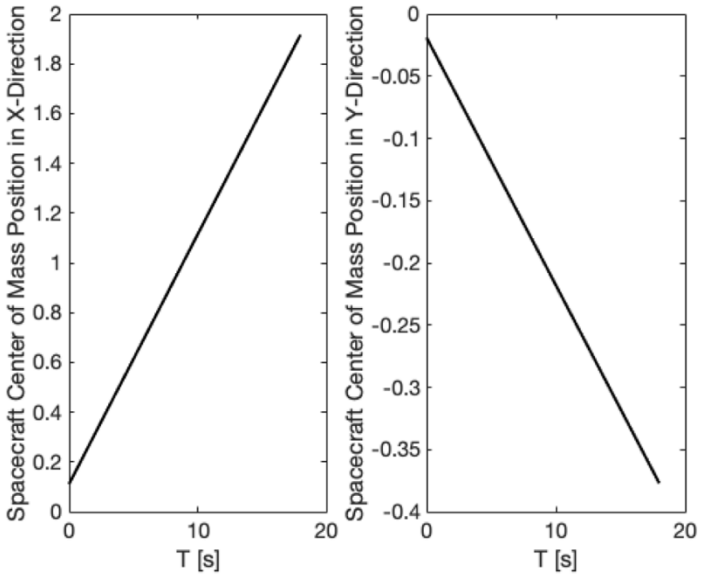
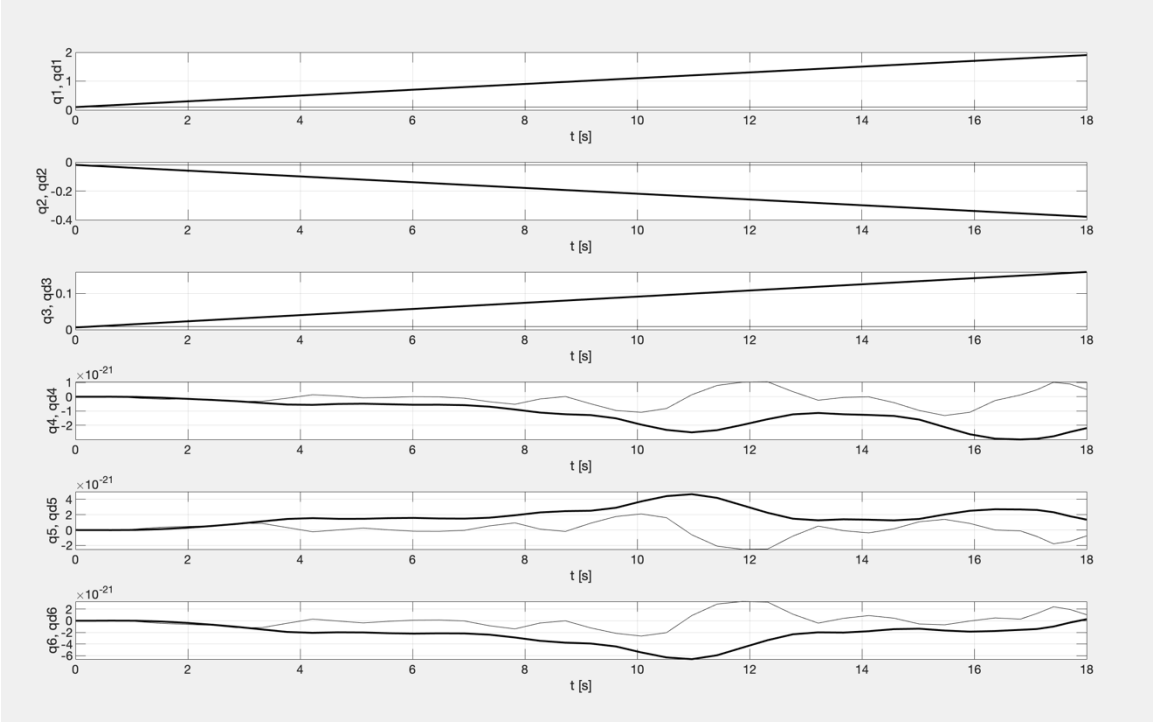


Figure 36. Simulated System Displacement During Coast



*Q1, Q2: Body motion in X and Y (m). Q3: Rotation of the plane (rad). Q4-Q6: Orientation of joints R1, R2, and R3 (rad).

Figure 37. Simulated Motion of Degrees of Freedom During Coast*

As shown in Figures 34-37, TORO© Simulation predicted total spacecraft motion of 6.4 centimeters during the actuation of the manipulator chain and 16.8 centimeters during the coast phase for a total of 23.2 centimeters during the maneuver. Motion during actuation was roughly comparable to the 5.9-centimeter change in end-effector distance from the root during actuation. However, overall predicted system motion was significantly lower than the experimental results which demonstrated system motion of nearly 70 centimeters.

This discrepancy is likely the result of differences between the operating assumptions of motion in TORO© when compared to the motion constraints of the physical manipulator design. TORO© software assumes smooth motion of joints from initial to final state, as numerical propagation techniques do not perform well with non-smooth conditions. PERSEUS manipulator control caused each actuator to begin and conclude each segment of maneuver trajectories with zero velocity. This corresponded to the

introduction of a spin-up or spin-down torque no less than eleven times throughout each push maneuver. Manipulator motion is caused entirely by torque produced at the motor shaft being transmitted through the body of the system. Therefore, it is not unreasonable to suspect that the production of additional torque at each iterative step along a maneuver trajectory resulted in greater motion than predicted. Since cable torque was shown to cause a displacement roughly six or seven centimeters greater than ideal conditions, actual system motion resulting from manipulator actuation for a push maneuver from a static simulated spacecraft was likely closer to 60 centimeters.

THIS PAGE INTENTIONALLY LEFT BLANK

IX. CONCLUSION

A. SYSTEM DESIGN

PERSEUS represented a successful first step toward the implementation of sophisticated robotic manipulation systems aboard NPS FSS units. The combination of additively manufactured structure, actuation system with documented open-source software, and use of self-contained communications and onboard computing hardware allowed PERSEUS to be rapidly integrated aboard existing systems without requiring modifications. Postural redundancy within a planar workspace also allowed the actual demonstration of maneuvers that could not be performed by previous systems with fewer degrees of freedom. The structure was observed to withstand both inertial loads and loads imparted to the system through robotic actuation.

B. CONCEPT OF OPERATIONS

The self-contained PERSEUS design architecture allowed the manipulator to be installed by a single technician in approximately fifteen minutes without requiring any modifications to the host FSS unit. Operation of the data collection system and both FSS units was carried out with the direct involvement of only two test conductors. Modifications to the operational procedures for PERSEUS will not likely decrease the number of test conductors required to perform experiments. However, process improvements that eliminate the need for physical cable tethering to the system and provide a means of controlling the system via wireless means could significantly improve the quality of experimental data. By streamlining the process of uploading programs, the need to recompile and execute for each arm maneuver could be eliminated, allowing test conductors to improve initial state controls to prevent the introduction of external forces and torques to the system.

C. PERFORMANCE

In its first implementation, PERSEUS was able to successfully perform a series of push maneuvers starting from a variety of initial conditions. This is considered successful.

However, a system fault experienced when attempting swing maneuver testing aboard POSEIDYN precluded the operational data collection of such maneuvers. Correction of this issue is of high importance and includes both improving the EPS interface between PERSEUS and FSS onboard power, as well as tightening of tolerances for opening and closing the end-effector to prevent crossing threshold values between minimum and maximum position indices for the end-effector actuator.

Overall, the system was observed to follow defined trajectories successfully with minimal deviation from user-defined end-effector position constraints. These small deviations resulted primarily as a result of the small, but nonzero backlash characteristics of each actuator combined with the angular momentum of rotating manipulator segments of significant length. These may be reduced by adjusting the default angular velocity used by the control software, and by eliminating unnecessary structural mass to reduce moments of inertia. Iterative joint space position control introduced significant vibratory motion to the manipulator chain. This is due primarily to the segmented zero velocity requirement resulting from sequential position control.

D. SUMMARY

PERSEUS was capable of performing a maneuver similar to that which could be used to separate two spacecraft from a docked position. This holds promise for examining future methods of performing proximity operations and servicing tasks using purely electromechanical means. The combination of software and hardware developed as part of the PERSEUS system provide an open framework for the iterative improvement and refinement of such techniques.

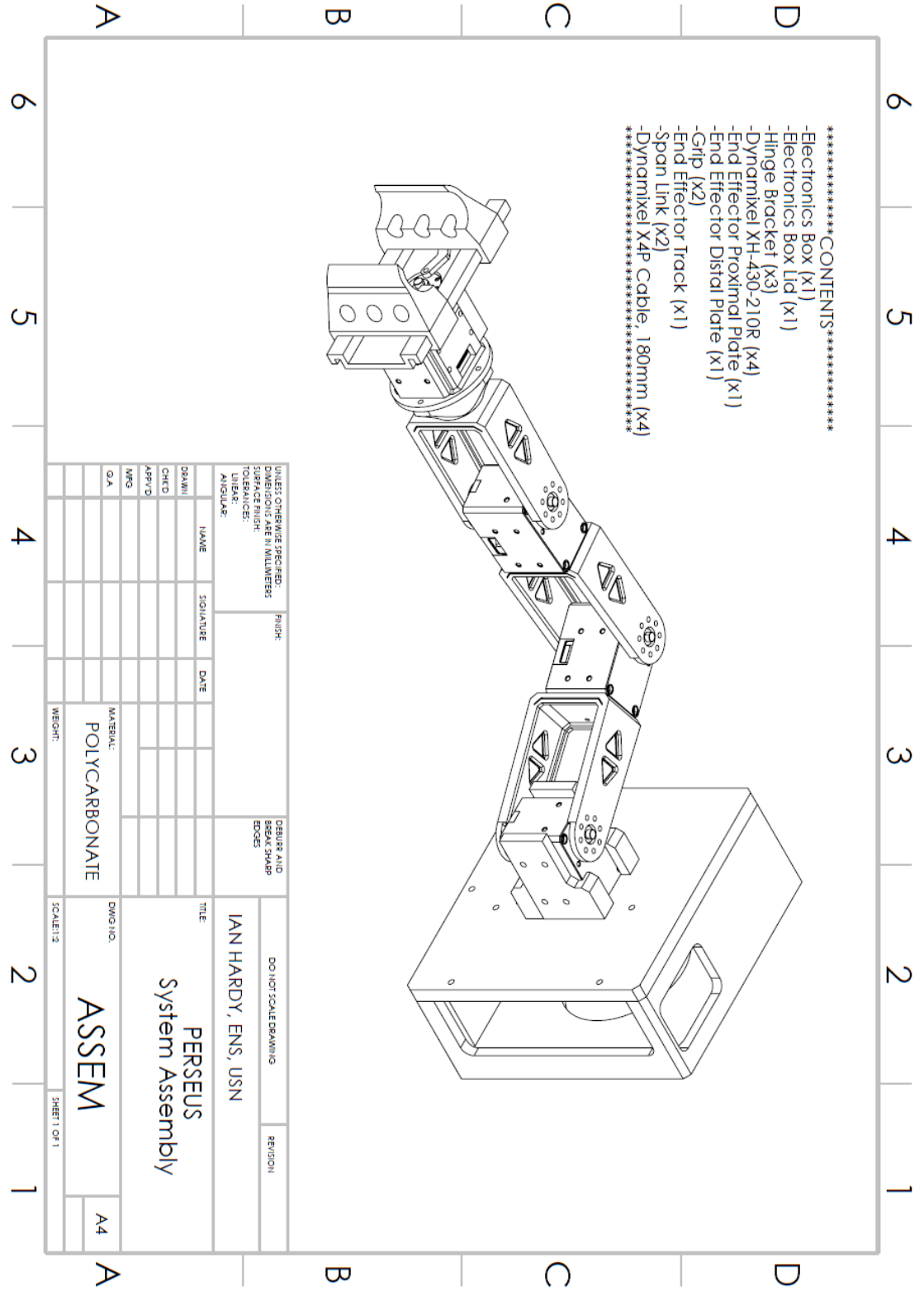
E. FUTURE WORK

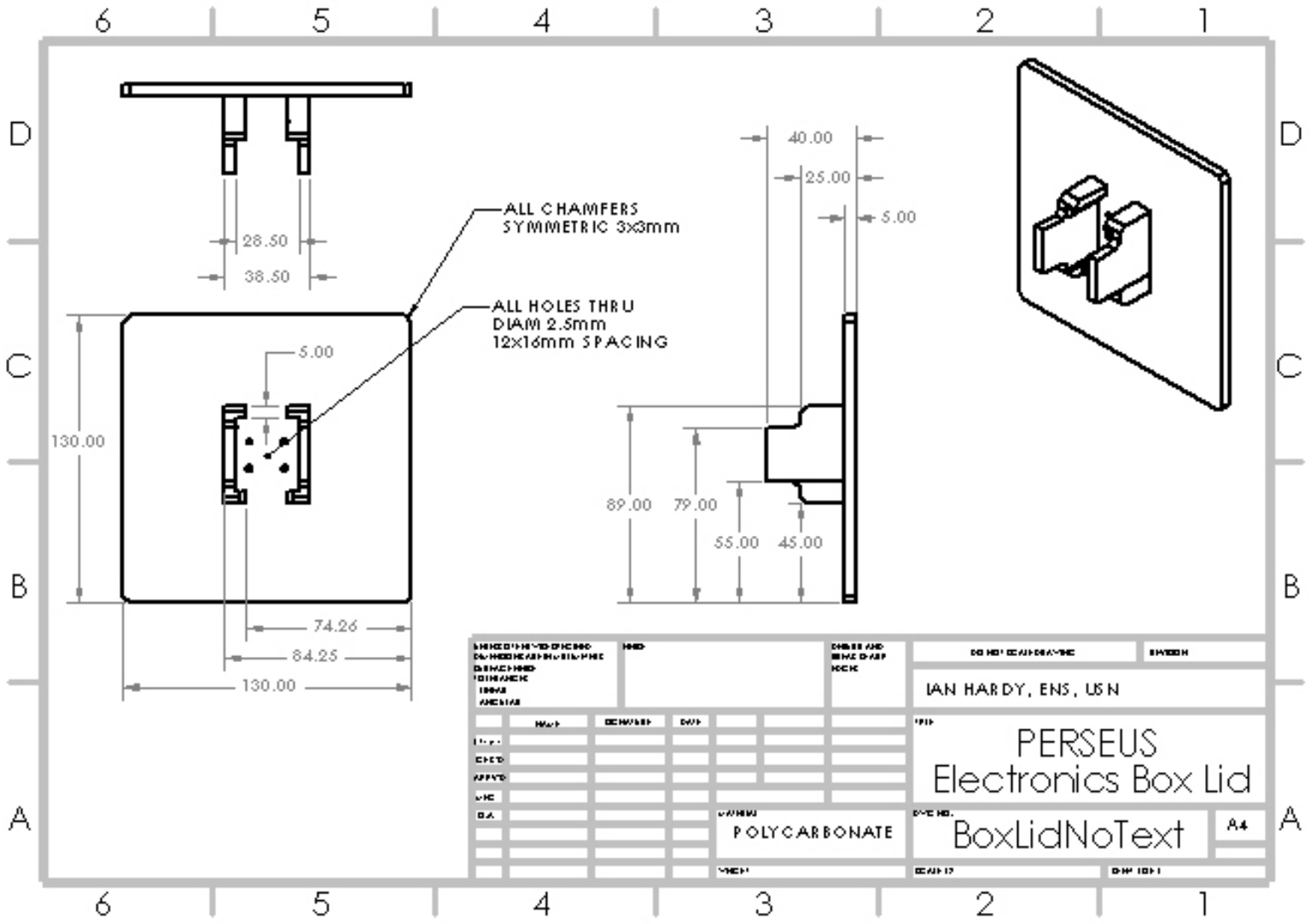
Multiple elements of system design should be refined in future versions to provide users greater flexibility, performance, and ease of use. These include enhanced access to power switches and communications interface ports, implementation of the OTA programming concept, and a restructuring of manipulator control code in order to reduce spin-up and spin-down torques resulting from current techniques used to segment maneuvers. Future iterations of PERSEUS could reduce this loading by introducing a state-

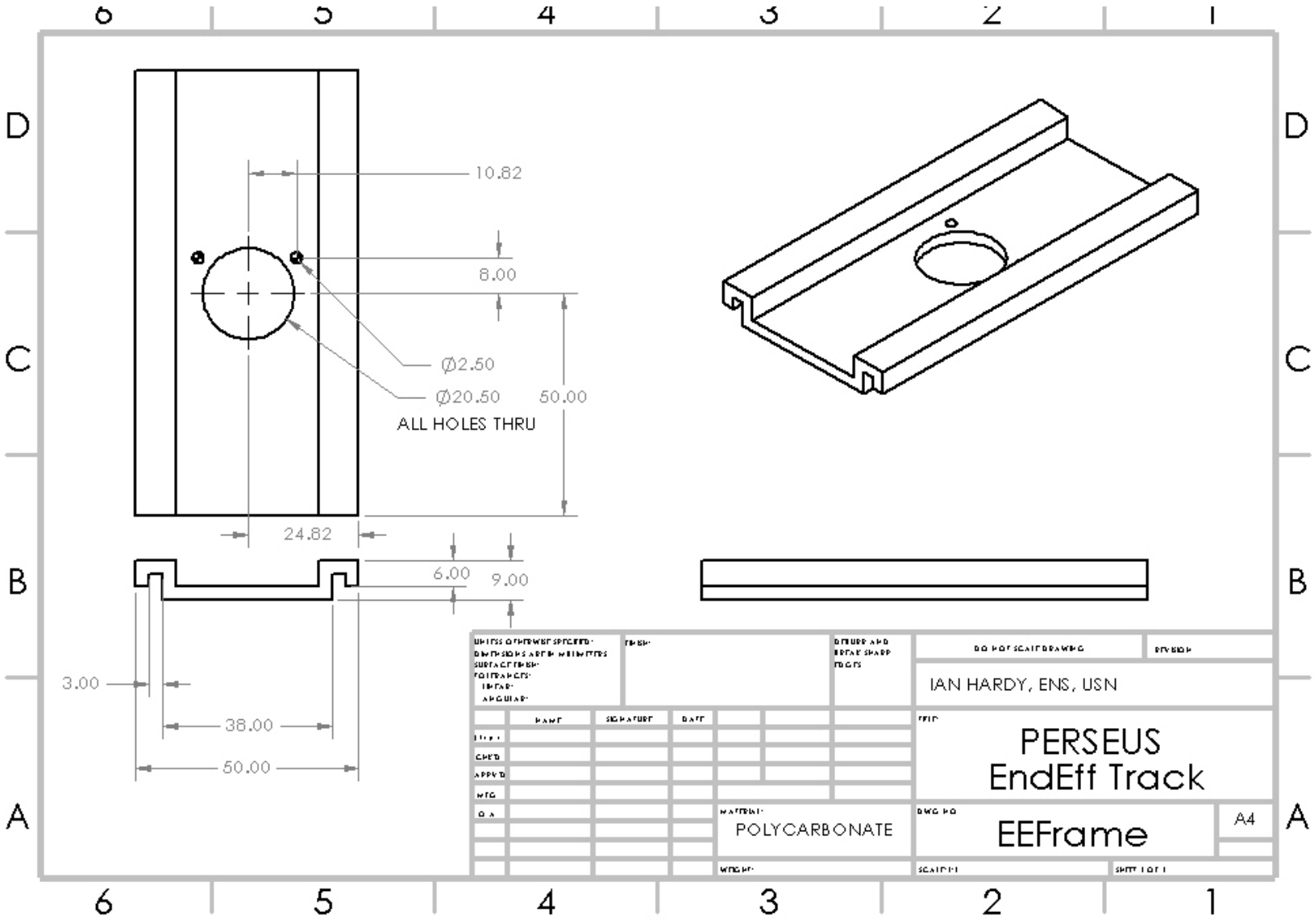
space control method allowing for non-zero angular velocity for the various actuators in between actuation steps. Vibratory motion during actuation could be reduced by reducing unnecessary component mass. Operational processes may be improved to allow more precise control of initial conditions by the same number of test conductors as used in this iteration. Tolerances for end-effector motion should be adjusted to prevent potential damage or misalignment that could result from crossing from motor position 0 to motor position 4095. Adjustments to trajectory generation software could also be used to account for these issues. By improving upon the existing design in these areas, PERSEUS could provide additional capability and perform a wider variety of proximity operations robotic maneuvers accurately.

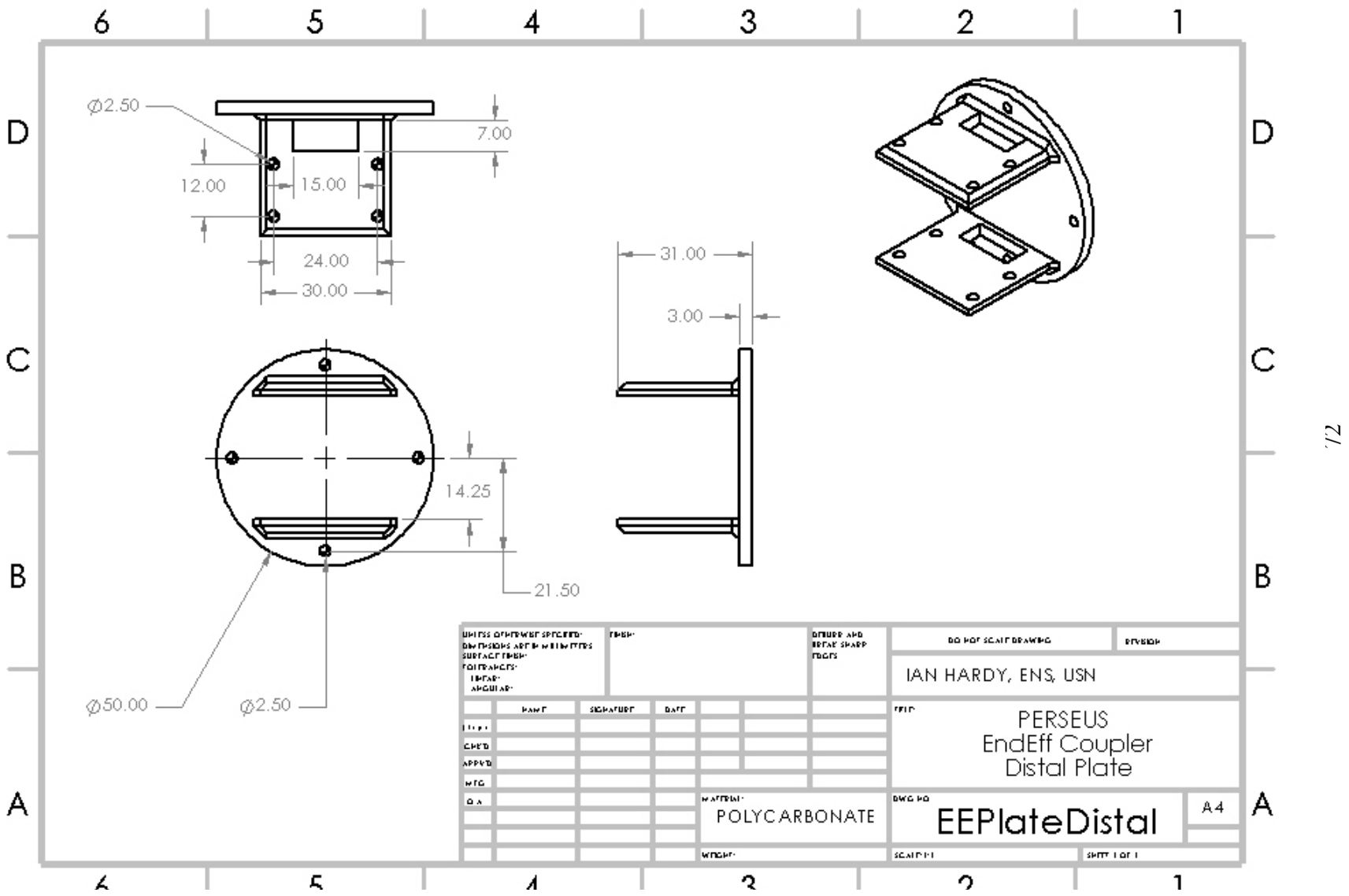
THIS PAGE INTENTIONALLY LEFT BLANK

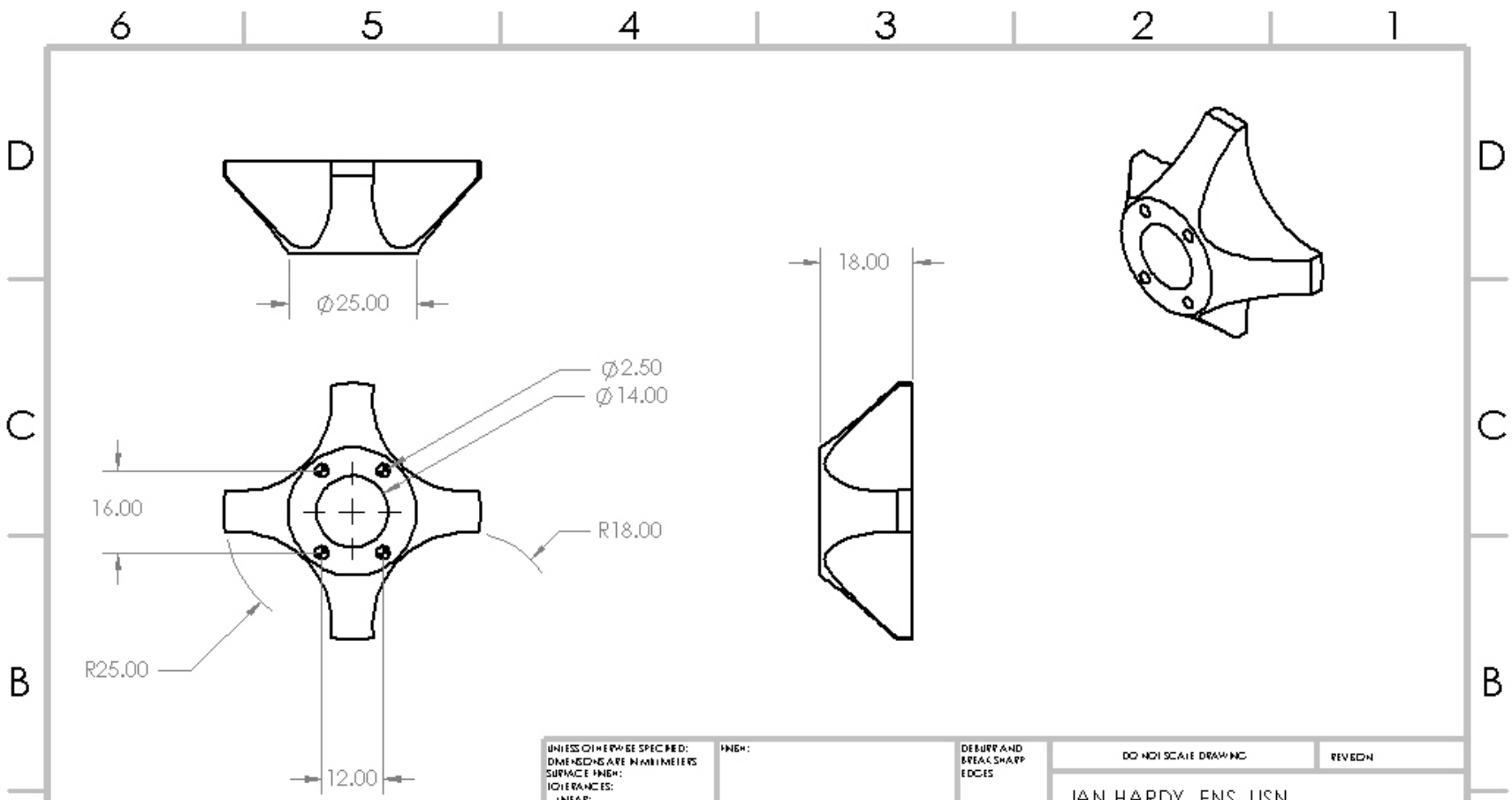
APPENDIX A. PERSEUS CAD DESIGN




ALL CHAMFERS
SYMMETRIC 3x3mm
ALL HOLES THRU
DIAM 2.5mm
12x16mm SPACING
28.50
38.50
5.00
130.00
74.26
84.25
130.00
40.00
25.00
5.00
89.00
79.00
55.00
45.00
IAN HARDY, ENS, USN
PERSEUS
Electronics Box Lid
POLYCARBONATE
BoxLidNoText
A4


10.82
8.00
Ø2.50
Ø20.50
50.00
ALL HOLES THRU
24.82
6.00
9.00
3.00
38.00
50.00
DO NOT SCALE DRAWING
IAN HARDY, ENS, USN
PERSEUS
EndEff Track
POLYCARBONATE
EEFrame
A4


6
5
4
3
2
1
D
C
B
A
⌀2.50
7.00
12.00
15.00
24.00
30.00
31.00
3.00
14.25
21.50
⌀50.00
⌀2.50
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
IAN HARDY, ENS, USN
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
PERSEUS
EndEff Coupler
Distal Plate
MATERIAL:
POLYCARBONATE
DWG NO.
EEPlateDistal
A4
WEIGHT:
SCALE: 1:1
SHEET 1 OF 1



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND BREAK SHARP EDGES

DO NOT SCALE DRAWING | REVISION

IAN HARDY, ENS, USN

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A.			

TITLE:
PERSEUS
EndEff Coupler
Proximal Plate

MATERIAL:
POLYCARBONATE

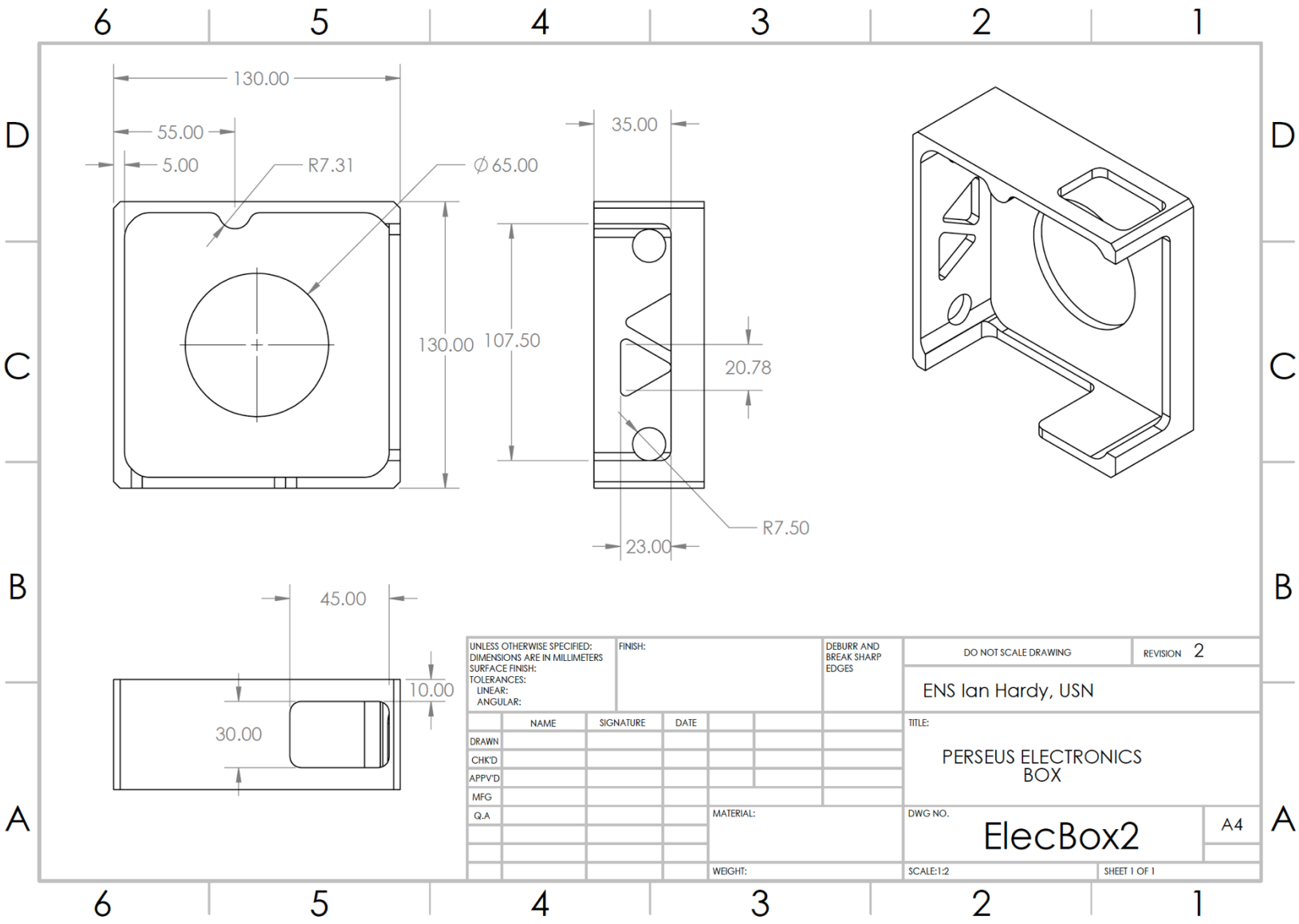
DWG NO.
EEPlateProximal

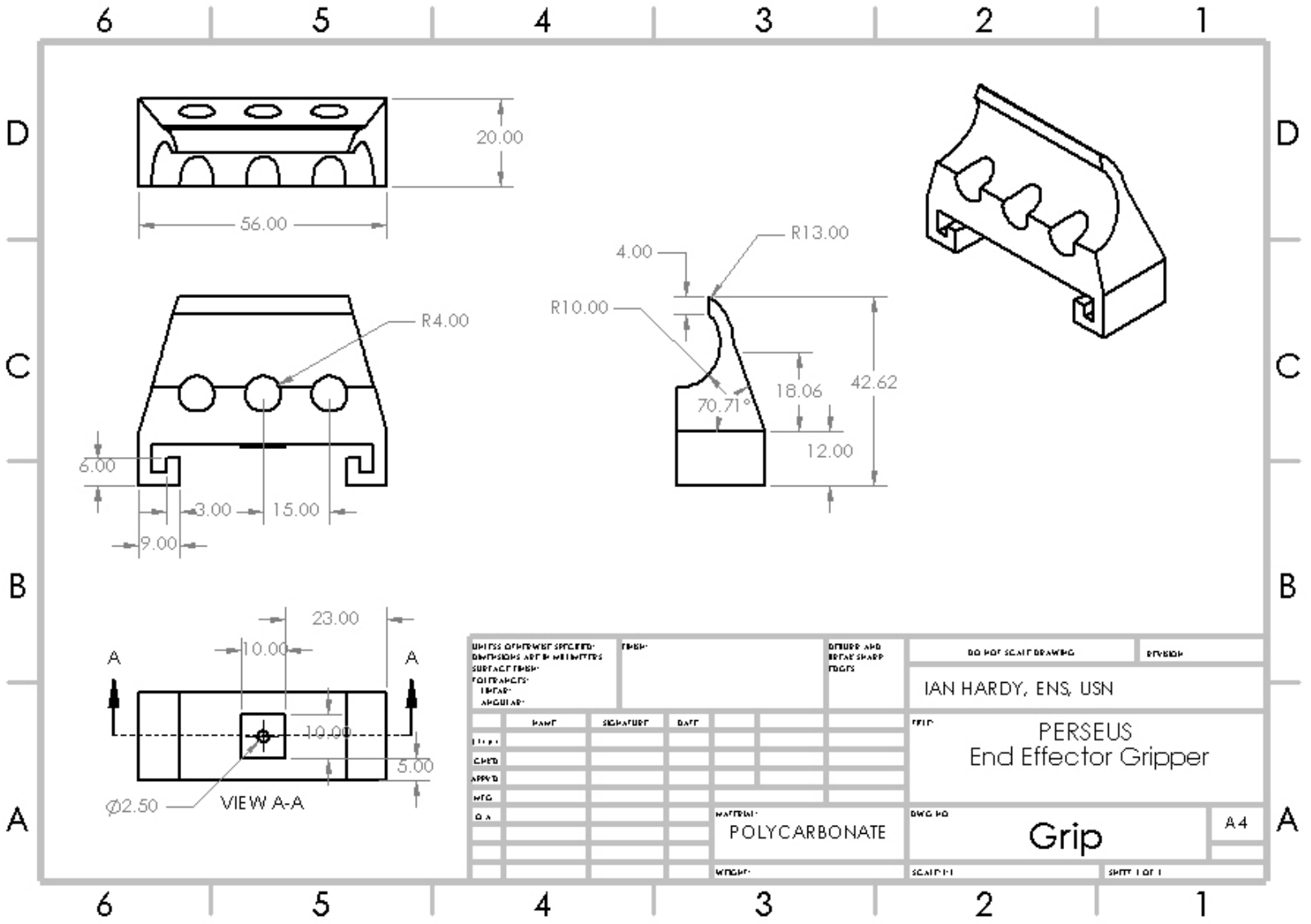
A4

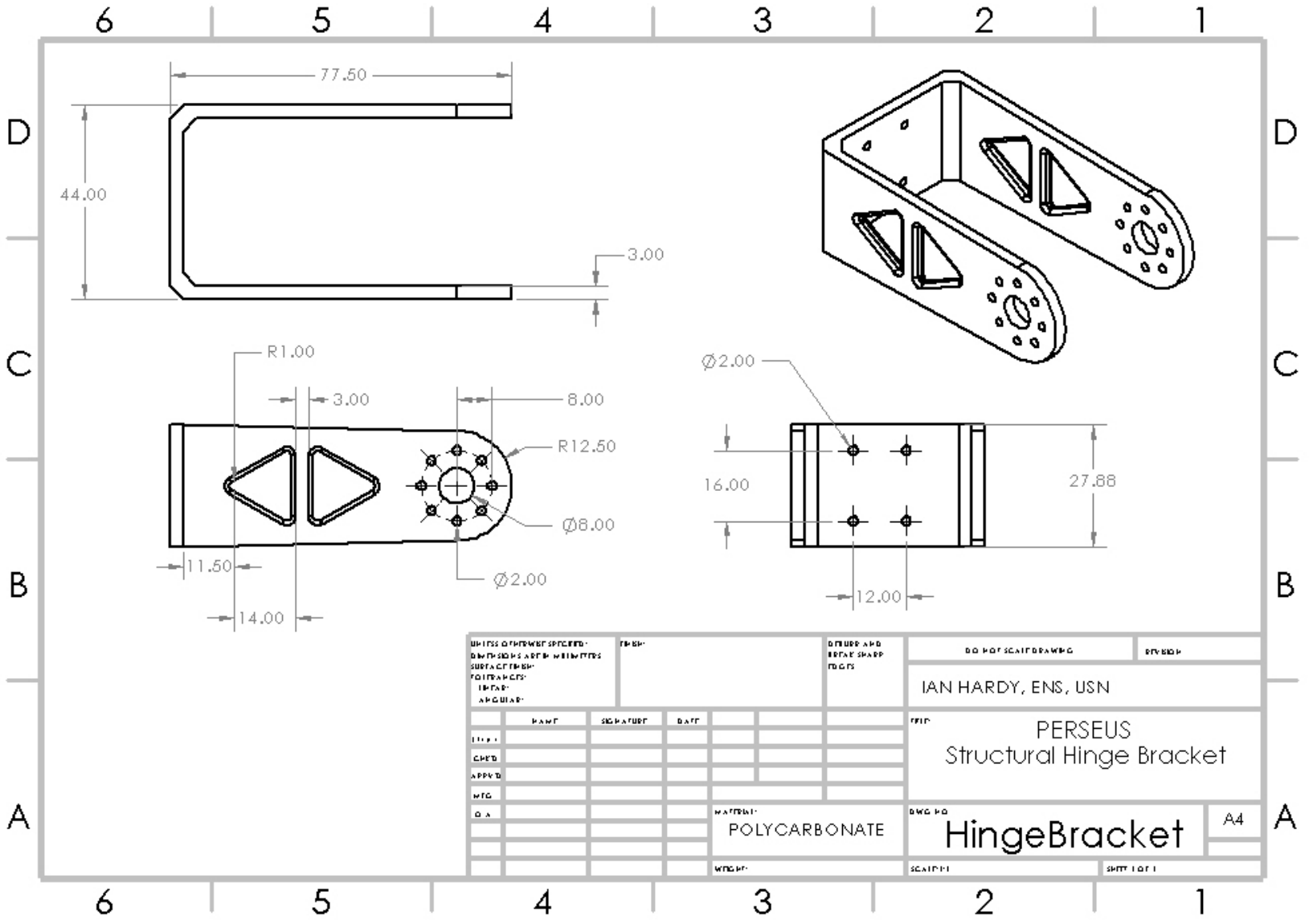
WEIGHT:

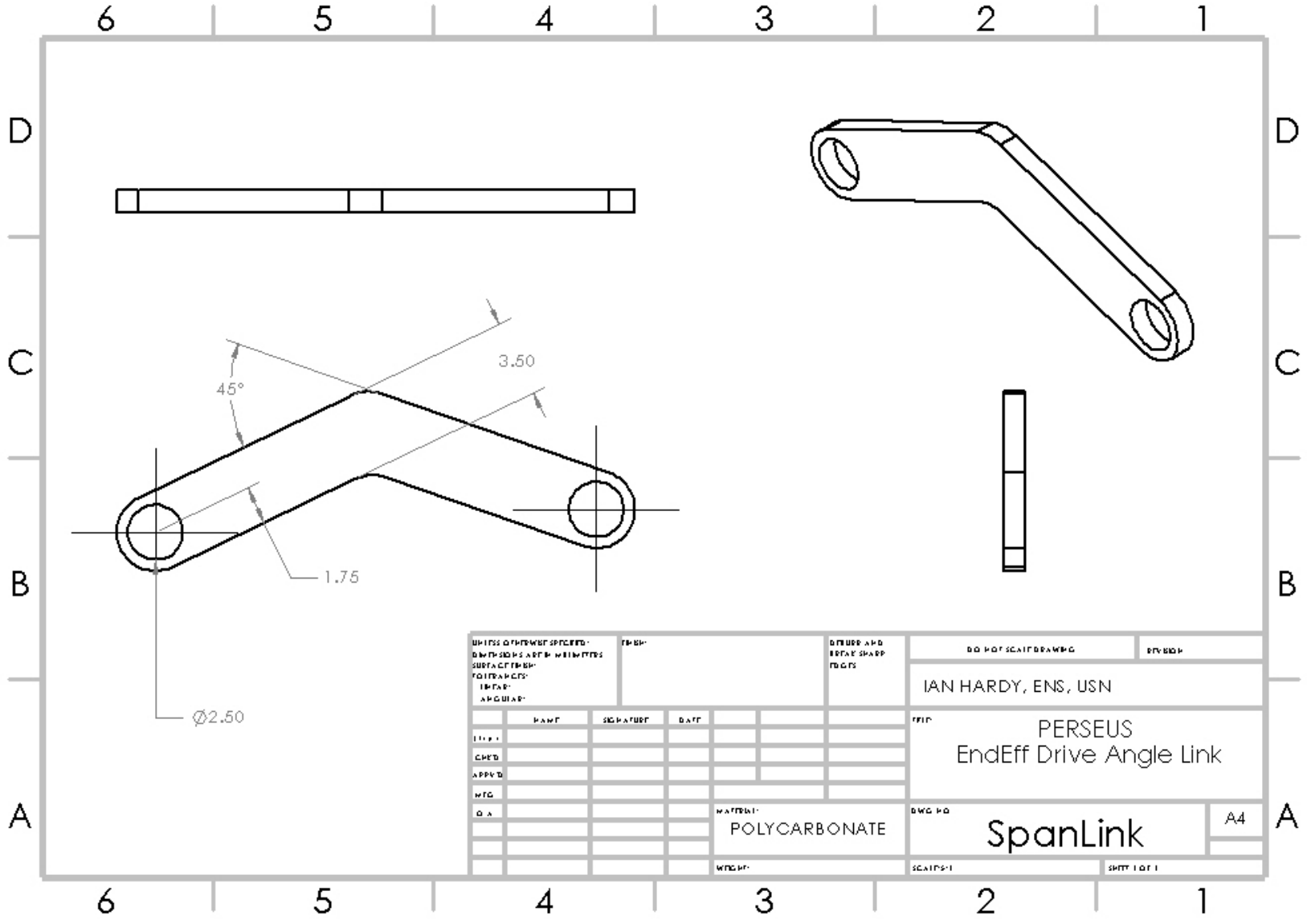
SCALE: 1:1

SHEET 1 OF 1


130.00
55.00
5.00
R7.31
∅65.00
130.00
107.50
35.00
20.78
R7.50
23.00
45.00
10.00
30.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION 2
ENS Ian Hardy, USN
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
PERSEUS ELECTRONICS BOX
DWG NO.
ElecBox2
A4
SCALE:1:2
SHEET 1 OF 1


6
5
4
3
2
1
D
C
B
A
20.00
56.00
R4.00
6.00
3.00
15.00
9.00
R13.00
4.00
R10.00
18.06
42.62
70.71°
12.00
23.00
10.00
A
A
10.00
5.00
Ø2.50
VIEW A-A
IAN HARDY, ENS, USN
PERSEUS
End Effector Gripper
POLYCARBONATE
Grip
A4


77.50
44.00
3.00
R1.00
3.00
8.00
R12.50
Ø8.00
Ø2.00
11.50
14.00
Ø2.00
16.00
27.88
12.00
IAN HARDY, ENS, USN
PERSEUS
Structural Hinge Bracket
POLYCARBONATE
HingeBracket
A4


6
5
4
3
2
1
D
C
B
A
45°
3.50
1.75
Ø2.50
IAN HARDY, ENS, USN
PERSEUS
EndEff Drive Angle Link
POLYCARBONATE
SpanLink
A4

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. DYNAMIXEL XH430-W210-R SPECIFICATIONS

MCU	ARM CORTEX-M3 (72 [MHz], 32Bit)
Position Sensor	Contactless absolute encoder (12Bit, 360 [°]) Maker : ams(www.ams.com), Part No : AS5045
Motor	Coreless(Maxon)
Baud Rate	9,600 [bps] ~ 4.5 [Mbps]
Control Algorithm	PID control
Resolution	4096 [pulse/rev]
Backlash	15 [arcmin] (0.25 [°])
Operating Modes	Current Control Mode Velcoity Control Mode Position Control Mode (0 ~ 360 [°]) Extended Position Control Mode (Multi-turn) Current-based Position Control Mode PWM Control Mode (Voltage Control Mode)
Weight	82 [g]
Dimensions (W x H x D)	28.5 x 46.5 x 34 [mm]
Gear Ratio	212.6 : 1
Stall Torque	2.2 [N.m] (at 11.1 [V], 1.2 [A]) **2.5 [N.m] (at 12.0 [V], 1.3 [A])** 3.1 [N.m] (at 14.8 [V], 1.5 [A])
No Load Speed	46 [rev/min] (at 11.1 [V]) **50 [rev/min] (at 12.0 [V])** 62 [rev/min] (at 14.8 [V])
Radial Load	40 [N] (10 [mm] away from the horn)
Axial Load	20 [N]
Operating Temperature	-5 ~ +80 [°C]
Input Voltage	10.0 ~ 14.8 [V] (**Recommended : 12.0 [V]**)
Command Signal	Digital Packet
Physical Connection	RS485 / TTL Multidrop Bus TTL Half Duplex Asynchronous Serial Communication with 8bit, 1stop, No Parity RS485 Asynchronous Serial Communication with 8bit, 1stop, No Parity
ID	253 ID (0 ~ 252)
Feedback	Position, Velocity, Current, Realtime tick, Trajectory, Temperature, Input Voltage, etc
Case Material	Metal (Front, Middle), Engineering Plastic (Back)
Gear Material	Full Metal Gear
Standby Current	40 [mA]

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. TRAJECTORY GENERATION CODE

PERSEUStest2txt

Written by: ENS Ian Hardy Naval Postgraduate School Spacecraft Robotics Laboratory JUNE 2021 "Ad Astra Per Aspera"

Creates Joint-Space Position Array for Test Maneuvers for PERSEUS Robotic Manipulator

Contents

Setup and Code Hygiene

```
clc
clear all
format compact
```

User Input

Requires exact adherence to requested input syntax from prompts. Deviation from syntax will cause error.

```
disp('CAUTION: Failure to follow requested input syntax will cause error')

StateVecs = input('Please Select Desired # of State Segments for Maneuver: (INT) ');

R0initial = input('Please Enter Desired Initial R0 Joint Angle in Degrees: ');
R0final = input('Please Enter Desired Final R0 Joint Angle in Degrees: ');

R1initial = input('Please Enter Desired Initial R1 Joint Angle in Degrees: ');
R1final = input('Please Enter Desired Final R1 Joint Angle in Degrees: ');

R2initial = input('Please Enter Desired Initial R2 Joint Angle in Degrees: ');
R2final = input('Please Enter Desired Final R2 Joint Angle in Degrees: ');

EEinitial = (input('Please Enter Desired INITIAL End-Effector State (O/C): ','s'));
EEchange = (input('Does End-Effector State Change? (Y/N) ','s'));
EEisopen = strcmp(EEinitial,'O');
EEcYes = strcmp(EEchange,'Y');
EEcNo = strcmp(EEchange,'N');


if EEcNo == 1
    if EEisopen == 1
        EEinitial = 0;
        EEfinal = 0;
        EEtraj = linspace(EEinitial,EEfinal,StateVecs);
    else
        EEinitial = 1;
        EEfinal = 1;
        EEtraj = linspace(EEinitial,EEfinal,StateVecs);
    end


else
    if EEisopen == 1
```

```
        EEfinal = 1; %'C'
    else EEfinal = 0; %'V'
    end

    EEtran = input('At What Point During the Maneuver Does EE open/close? (0.0 - 1.0) ');
    TranPoint = round(EEtran * StateVecs);
    EEtraj = NaN(1,StateVecs);
    EEtraj(TranPoint+1:end) = EEfinal;
    EEtraj(1:TranPoint) = -EEfinal + 1;
end
```

```
CAUTION: Failure to follow requested input syntax will cause error


Error using input
Cannot call INPUT from EVALC.
Error in PERSEUStest2txt (line 22)
StateVecs = input('Please Select Desired # of State Segments for Maneuver: (INT) ');
```

Linear Joint-Space Trajcetory Generation

All vectors are linearly-spaced to reduce mid-trajectory spinup/spindown torque. Transient torque condition still present at initial and final points

```
R0traj = linspace(R0initial,R0final,StateVecs);
R1traj = linspace(R1initial,R1final,StateVecs);
R2traj = linspace(R2initial,R2final,StateVecs);
```

Angle to Position Conversion

Uses conversion factor for Dynamixel XH-430 gearing

```
numPos = 4096;
flip180 = fix(numPos/2);
PosperDeg = numPos / 360;

EEcloseAngle = 90; % 90 degree angle orientation closes EE. Subject to design change

R0trajPos = fix((PosperDeg * R0traj)+flip180);
R1trajPos = fix((PosperDeg * R1traj)+flip180);
R2trajPos = fix((PosperDeg * R2traj)+flip180);
EEtrajPos = fix(PosperDeg * EEtraj * EEcloseAngle) + 1;

c0 = R0trajPos < 0;
c1 = R1trajPos < 0;
c2 = R2trajPos < 0;
cE = EEtrajPos < 0;

R0trajPos(c0) = R0trajPos(c0) + numPos;
R1trajPos(c1) = R1trajPos(c1) + numPos;
R2trajPos(c2) = R2trajPos(c2) + numPos;
EEtrajPos(cE) = EEtrajPos(cE) + numPos;
```

Link and Body Positions

Link 2 includes end-effector due to fixed geometry

```
RootOffsetX = 0;
RootOffsetY = 0.04; %cm, offset of R0 from mounting face

L0 = 0.1; % 10cm
L1 = 0.1; % 10cm
L2 = 0.2; % 20cm

L0posX = RootOffsetX + (L0 * sind(R0traj));
```

```
L0posY = RootOffsetY + (L0 * cosd(R0traj));

L1posX = L0posX + (L1 * sind(R0traj + R1traj));
L1posY = L0posY + (L1 * cosd(R0traj + R1traj));

L2posX = L1posX + (L2 * sind(R0traj + R1traj + R2traj));
L2posY = L1posY + (L2 * cosd(R0traj + R1traj + R2traj));
```

Angle Limit Check

NOTE: Current limits are due to cable routing configuration imposed by Dynamixel XH-430 case design. Updates to routing may change these limits.

```
R0limitLo = -90;
R0limitHi = 90;

R1limitLo = -90;
R1limitHi = 90;

R2limitLo = -90;
R2limitHi = 90;

disp(':::::CHECKING JOINT ANGLE BOUNDS:::::')

if all(R0traj >= R0limitLo) && all(R0traj <= R0limitHi)
    disp('R0 Joint Space Trajectory Appears Within Bounds')
elseif any(R0traj < R0limitLo)
    disp('R0 CCW Rotation Exceeds Bound')
else
    disp('R0 CW Rotation Exceeds Bound')
end

if all(R1traj >= R1limitLo) && all(R1traj <= R1limitHi)
    disp('R1 Joint Space Trajectory Appears Within Bounds')
elseif any(R1traj < R1limitLo)
    disp('R1 CCW Rotation Exceeds Bound')
else
    disp('R1 CW Rotation Exceeds Bound')
end

if all(R2traj >= R2limitLo) && all(R2traj <= R2limitHi)
    disp('R2 Joint Space Trajectory Appears Within Bounds')
elseif any(R2traj < R2limitLo)
    disp('R2 CCW Rotation Exceeds Bound')
else
    disp('R2 CW Rotation Exceeds Bound')
end

disp(':::::JOINT ANGLE BOUNDS CHECK COMPLETE:::::')
```

Physical Position Check

Compares positions of link bodies and end-effector to determine if a candidate trajectory would contact own spacecraft body

```
Body.xlimR = 0.15;
Body.xlimL = -0.15;
Body.ylimHi = 0;
Body.ylimLo = -0.3;

disp(':::::CHECKING LINK AND END-EFFECTOR POSITION FOR POTENTIAL COLLISION:::::')

posWarn = 0;

if any(L0posX <= Body.xlimR) && any(L0posX >= Body.xlimL) && any(L0posY <= Body.ylimHi) && any(L0posY >= Body.ylimLo)
    disp('WARNING: L0 COLLISION PREDICTED')
    posWarn = posWarn + 1;
    else disp('NO COLLISION PREDICTED FOR L0')
end
```

```
if any(L1posX <= Body.xlimR) && any(L1posX >= Body.xlimL) && any(L1posY <= Body.ylimHi) && any(L1posY >= Body.ylimLo)
    disp('WARNING: L1 COLLISION PREDICTED')
    posWarn = posWarn + 1;
    else disp('NO COLLISION PREDICTED FOR L1')
end

if any(L2posX <= Body.xlimR) && any(L2posX >= Body.xlimL) && any(L2posY <= Body.ylimHi) && any(L2posY >= Body.ylimLo)
    disp('WARNING: END-EFFECTOR COLLISION PREDICTED')
    posWarn = posWarn + 1;
    else disp('NO COLLISION PREDICTED FOR END-EFFECTOR')
end

if posWarn == 0
    disp('NO COLLISION PREDICTED WITH OWN S/C BODY')
end

disp('::::::POTENTIAL COLLISION CHECK COMPLETE::::::')
```

Append Each Row Trajectory with Rotation Direction Data

Clockwise equals zero per dynamixel software development kit *IMPORTANT NOTE: Index elements for final state command follow this pattern: 1-0-1-0. This will cause the arm to "fold in" at the last rotation in order to reduce probability of collision Be sure to examine txt file carefully before execution of C++ script

```
R0index = ones(size(R0traj));
R1index = zeros(size(R1traj));
R2index = ones(size(R2traj));
EEindex = zeros(size(EEtraj));

for indexcounter = 1:(StateVecs-1)
    if (R0traj(indexcounter+1) < R0traj(indexcounter))
        R0index(indexcounter) = 1; % CCW rotation
    else R0index(indexcounter) = 0; % CW rotation
    end

    if (R1traj(indexcounter+1) < R1traj(indexcounter))
        R1index(indexcounter) = 1; % CCW rotation
    else R1index(indexcounter) = 0; % CW Rotation
    end

    if (R2traj(indexcounter+1) < R2traj(indexcounter))
        R2index(indexcounter) = 1; % CCW rotation
    else R2index(indexcounter) = 0; % CW rotation
    end

    if (EEtraj(indexcounter+1) < EEtraj(indexcounter))
        EEindex(indexcounter) = 1; % CCW rotation
    else EEindex(indexcounter) = 0; % CW Rotation
    end

end
```

Concatenate Trajectories and Save to .txt File

```
myFile = input("Select Trajectory File Name (e.g. 'File1.txt'): ","s");

if posWarn ==0
    disp('No Warnings Detected. Saving to column-oriented .txt file...')
    testArray = cat(2,R0trajPos,R1trajPos,R2trajPos,EEtrajPos);
    %testArray = cat(1,R0trajPos,R0index,R1trajPos,R1index,R2trajPos,R2index,EEtrajPos,EEindex);
    writematrix(testArray,myFile,'Delimiter','space')
    disp(testArray)
else
    disp('Trajectory Not Saved Due to Warning')
end
```

APPENDIX D. TORO EQUATIONS OF MOTION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (C) Dr. Marcello Romano, 2020-08-19
% This function is part of TORO: a Tool Set for Orbital ROBotics
%
% Writing the symbolic EoM for a System made of
% an UNGROUNDED SYSTEM (UG), CONSISTING OF
% A Rigid-base Spacecraft and
% A Rigid-link Robotic Manipulator with two joints with rotary actuator: convention for coordinates IS as in figure
%

clear all
clc

%publishing:
% publish('SCRIPT1_EOM_UG_Planar_SC_R2.m','pdf')

%declare t as a symbolic variable, it will be used for "time"
syms t real

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% USER INPUT BEGIN
%%%%%%%%%%%%%%%%%

%DECLARING SYMBOLIC VARIABLES USED
%
%WARNING: IMPORTANT, USE the following symbols in your inputs
%
%q1,q2,...qn for generalized coordinates
%qd1,qd2,...qdn for generalized velocities
%
%m0,m1,...mn-1 for the masses of the body
%I0, I1...In-1 for the inertia matrices at the Center of Mass of each link
%projected in the Link coordinate systems

%USER INPUT: Name of the system: an integrandum function SystemName.m is eventually created
SystemName = 'PERSEUS_SC_R3'

%USER INPUT: %Declare symbolic variables needed in the input
%THIS IS NEEDED FOR THE FUNCTION EoM2Odefun.m called later below
syms l0 l1 l2 l3 real %constant parameters: length of the base side (considered of being of square shape) and link 1
syms m0 m1 m2 m3 real % constant masses of base (m0) and link (m1)
syms ic0 ic1 ic2 ic3 real %principal moment of inertia of base (ic0) and link (ic1) about the axis normal to the plane of motion
syms X1 X2 X3 X4 X5 X6 real %Generalized NON-Conservative Forces associated to the coordinates
par = {l0 l1 l2 l3 m0 m1 m2 m3 ic0 ic1 ic2 ic3}; %always use this order of listing the parameters

syms q1 q2 q3 q4 q5 q6 real % gen. coordinates variable (function of time)
% x, y, plane, the1, the2, the3
%EXPLANATION OF THE COORDINATES (see figure)
%let us call x0 and y0 the coordinates of \vect{r}_0 in \vectorbase{N}
%q1=x0
%q2=y0
%q3=theta0
%q4=theta1
%q5=theta2


%USER INPUT: number of degrees of freedom
%(it is assumed that the number of generalized coordinates is equal to the
%number of degrees of freedom (minimal system representation), i.e. constrains among the qi are NOT
%supported)
numdof=6;
tic
%USER INPUT:
%DEVELOPING EXPRESSION FOR POSITION VECTOR COORDINATES OF the Center of Mass of each body IN INERTIAL BASE N

%position vector r0 in coordinates in vector base N
N_r0 = [q1;q2];


C_N_L0=[cos(q3),-sin(q3);sin(q3), cos(q3)]; %DCM from base (L0) to N


%Let us call r1 the position vector of c1 relative to the origin of the
%vector base N (ON),
%J1c1: the position vector of c1 relative to J1 in vector base L0
L0_J1c1=l1/2*[cos(q4); sin(q4)];
L0_c01=l0/2*[1; 0];

%Position vector r1=Oc1 in coordinates in N
N_r1 = N_r0 + C_N_L0 *(L0_c01+L0_J1c1);

%theta2=q5
L1_J2c2=l2/2*[cos(q5); sin(q5)];
L1_c12=l1/2*[1; 0];

%theta1=q5
C_L0_L1= [cos(q4),-sin(q4);sin(q4), cos(q4)];
C_N_L1=C_N_L0*C_L0_L1;
```

```
N_r2 =N_r1+ C_N_L1 *(L1_c12+L1_J2c2);


%Attempting for the last link now:
L2_J3c3= l3/2*[cos(q6); sin(q6)];
L2_c23= l2/2*[1; 0];

C_L0_L2= [cos(q5),-sin(q5);sin(q5), cos(q5)];
C_N_L2= C_N_L0*C_L0_L2;

N_r3=N_r2+C_N_L2 *(L2_c23+L2_J3c3);

% Final Base (massive object at end of chain)
% L3_J4c4 = l4/2*[cos(q6); sin(q6)];
% L3_c34 = l3/2*[1; 0];
%
% C_L0_L3= [cos(q6),-sin(q6);sin(q6), cos(q6)];
% C_N_L3= C_N_L0*C_L0_L3;
%
% N_r4=N_r3+C_N_L3*(L3_c34+L3_J4c4);

NPositionVectors={N_r0,N_r1, N_r2, N_r3};

% Now for final mass fixed at end

%USER INPUT:
verbose = 1; %[either 0 or 1] if 1 is selected, the expression of K, V, and of H, C and Tau are printed on matlab screen.

%USER INPUT:
%POTENTIAL ENERGY OF THE SYSTEM (up to an addittive constant)
%the first motor is at the joint 1 and the second motor is at joint 2
V= 0; %flat space hypothesis


%%%%%%%%%%%%%%%
% USER INPUT END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% WARNING:
% DO NOT MODIFY ANY CODE UNDERNEATH, UNLESS THERE ARE ERRORS
% OF EXECUTION AND YOU ARE SURE THAT THE INPUT ABOVE HAS BEEN PREPARED
% ACCORDINT TO THE INSTRUCTIONS

thisfilepath= fileparts(matlab.desktop.editor.getActiveFilename);

numberofbodies=size(NPositionVectors,2);

%masses = sym('m', [numberofbodies,1], 'real') %create array of masses=[m1; m2;...mn]
for i=1:numberofbodies%(numberofbodies+1):(2*numberofbodies)
    masses(i,1) = par{numberofbodies+i};
    inertias(i,1) = par{2*numberofbodies+i};
end

K = PlanarChainUGMB2KE(NPositionVectors, verbose, masses,inertias)

[EoM, H, C, G] =EOM_Lagrange(K,V, numdof, verbose)
%H, C, G
%H: n-by-n symmetric positive definite Mass Matrix of the system
%C: n-by-n Centrifugual-Coriolis Matrix of the system
%G: n-by-1 matrix of generalized CONSERVATIVE forces acting on the system
%
%H,C,G allows to write the equations of motion in the canonical form:
%H*qdd+C*qd = G + GenForces (added later when building the odefun)

SCM=PlanarChainUGMB2SCM(NPositionVectors, verbose, masses, inertias);

GenerateFunctionsUG(K, V, par, SystemName, thisfilepath,H,C,G, SCM,numdof);
time_to_obtain_functionsCGHKSCMV=toc
%if automatically_create_odefun
%    GenerateOdefun_Using_Ode2VectField(EoM, K, V, par, SystemName, thisfilepath, H,C,G);
%end
```

```
SystemName =
    'PERSEUS_SC_R3'
TransKineticEnergy =
(m0*(qd1^2 + qd2^2))/2
RotKineticEnergy =
(ic0*qd3^2)/2
TransKineticEnergy =
(m1*((qd2 + qd3*cos(q3)*(l0/2 + (l1*cos(q4))/2) - (l1*qd3*sin(q3)*sin(q4))/2 - (l1*qd4*sin(q3)*sin(q4))/2 + (l1*qd4*cos(q3)*cos(q4))/2)^2 + (qd3*sin(q
RotKineticEnergy =
(ic1*(qd3 + qd4)^2)/2 + (ic0*qd3^2)/2
TransKineticEnergy =
(m2*(((l1/2 + (l2*cos(q5))/2)*(qd3*cos(q3)*sin(q4) + qd3*cos(q4)*sin(q3) + qd4*cos(q3)*sin(q4) + qd4*cos(q4)*sin(q3)) - qd1 + (l2*sin(q5)*(qd3*cos(q3)
RotKineticEnergy =
(ic1*(qd3 + qd4)^2)/2 + (ic2*(qd4 + qd5)^2)/2 + (ic0*qd3^2)/2
TransKineticEnergy =
```

```
(m2*(((l1/2 + (l2*cos(q5))/2)*(qd3*cos(q3)*sin(q4) + qd3*cos(q4)*sin(q3) + qd4*cos(q3)*sin(q4) + qd4*cos(q4)*sin(q3)) - qdl + (l2*sin(q5)*(qd3*cos(q3)
RotKineticEnergy =
(ic1*(qd3 + qd4)^2)/2 + (ic2*(qd4 + qd5)^2)/2 + (ic3*(qd5 + qd6)^2)/2 + (ic0*qd3^2)/2
```

The kinetic energy computed is

$$
\begin{aligned}
&\left(m2\,\left(\left(\#12\,\#22 - qd1 + \#13 + \#15 + \#20 + \#17 + \#7 + \#6 + \#5\right)^2\right.\right. \\
&\quad \left.\left. + \left(qd2 + \#12\,\#21 - \#14 + \#16 - \#4 - \#3 + \#19 - \#18 + \#8\right)^2\right)\right)/2 \\
&\quad + \frac{ic1\,(qd3 + qd4)^2}{2} + \frac{ic2\,(qd4 + qd5)^2}{2} + \frac{ic3\,(qd5 + qd6)^2}{2} \\
&\quad + \left(\left|m3\right|\left|qd2 + \#12\,\#21 + \#11\,\#1 - \#14 - \frac{l3\,\sin(q6)\,\#2}{2} + \#16 - \#4 - \#3\right|\right. \\
&\quad \left. + \#19 + \frac{l3\,qd6\,\cos(q6)\,\#9}{2} - \#18 - \frac{l3\,qd6\,\sin(q6)\,\#10}{2} + \#8\right|^2 \\
&\quad + \left(\left|\#12\,\#22 - qd1 + \#11\,\#2 + \#13 + \frac{l3\,\sin(q6)\,\#1}{2} + \#15 + \#20\right|\right. \\
&\quad \left. + \frac{l3\,qd6\,\cos(q6)\,\#10}{2} + \#17 + \frac{l3\,qd6\,\sin(q6)\,\#9}{2} + \#7 + \#6 + \#5\right|^2 \\
&\quad + \frac{ic0\,qd3^2}{2} \\
&\quad + \frac{ml\,\left(\left(qd2 + \#16 - \#4 - \#3 + \#8\right)^2 + \left(\#15 - qd1 + \#7 + \#6 + \#5\right)^2\right)}{2} \\
&\quad + \frac{m0\,\left(qdl^2 + qd2^2\right)}{2}
\end{aligned}
$$

where

$$
\begin{aligned}
\#1 &= qd3\,\cos(q3)\,\cos(q5) + qd5\,\cos(q3)\,\cos(q5) - qd3\,\sin(q3)\,\sin(q5) - qd5\,\sin(q3)\,\sin(q5) \\
\#2 &= qd3\,\cos(q3)\,\sin(q5) + qd3\,\cos(q5)\,\sin(q3) + qd5\,\cos(q3)\,\sin(q5) + qd5\,\cos(q5)\,\sin(q3) \\
\#3 &= \frac{l1\,qd4\,\sin(q3)\,\sin(q4)}{2} \\
\#4 &= \frac{l1\,qd3\,\sin(q3)\,\sin(q4)}{2} \\
\#5 &= \frac{l1\,qd4\,\cos(q4)\,\sin(q3)}{2} \\
\#6 &= \frac{l1\,qd4\,\cos(q3)\,\sin(q4)}{2} \\
\#7 &= \frac{l1\,qd3\,\cos(q3)\,\sin(q4)}{2} \\
\#8 &= \frac{l1\,qd4\,\cos(q3)\,\cos(q4)}{2} \\
\#9 &= \cos(q3)\,\cos(q5) - \sin(q3)\,\sin(q5) \\
\#10 &= \cos(q3)\,\sin(q5) + \cos(q5)\,\sin(q3) \\
\#11 &= \frac{l2}{2} + \frac{l3\,\cos(q6)}{2}
\end{aligned}
$$

```
          2        2

         l1   l2 cos(q5)
#12 == -- + ----------
         2        2

         l2 sin(q5) #21
#13 == --------------
                2

         l2 sin(q5) #22
#14 == --------------
                2

#15 == qd3 sin(q3) #23

#16 == qd3 cos(q3) #23

         l2 qd5 sin(q5) #24
#17 == -------------------
                2

         l2 qd5 sin(q5) #25
#18 == -------------------
                2

         l2 qd5 cos(q5) #24
#19 == -------------------
                2

         l2 qd5 cos(q5) #25
#20 == -------------------
                2

#21 == qd3 cos(q3) cos(q4) + qd4 cos(q3) cos(q4) - qd3

   sin(q3) sin(q4) - qd4 sin(q3) sin(q4)

#22 == qd3 cos(q3) sin(q4) + qd3 cos(q4) sin(q3) + qd4

   cos(q3) sin(q4) + qd4 cos(q4) sin(q3)

         l0   l1 cos(q4)
#23 == -- + ----------
         2        2

#24 == cos(q3) cos(q4) - sin(q3) sin(q4)

#25 == cos(q3) sin(q4) + cos(q4) sin(q3)
```

```
K =
(m2*((2*qd2 + l0*qd3*cos(q3) + l2*qd3*cos(q3 + q4 + q5) + l2*qd4*cos(q3 + q4 + q5) + l2*qd5*cos(q3 + q4 + q5) + 2*l1*qd3*cos(q3 + q4) + 2*l1*qd4*cos(q
```

The system Mass matrix computed by the function TORO_Lagrange_ConservativeSys is

```
--
|
|  [m0 + m1 + m2 + m3, 0, #4, #8, - #36 - #35 - #21 - #22, -#21],
--

  [0, m0 + m1 + m2 + m3, #3, #7, #31 + #30 + #19 + #20, #19],

  --                      2         2              2
  |                    l0  m1    l0  m2         l0  m3
  |  #4, #3, ic0 + ic1 + ------ + ------ + #11 + ------
  --                     4         4              4

       2         2                          #28          #17   #16   #29   #14
   + l1  m2 + l1  m3 + #23 + #15 + #41 + ----- + #40 + ----- + --- + --- + ---
                                          2             2     2     2     2

                                                                  --
                #18               #43          #42                |
  + #13 + #12 + --- + #25 + #24 + --- + #27 + ---, #1, #2, #6  |,
                 2                 2            2                --

                                    2         2
  [#8, #7, #1, ic1 + ic2 + #11 + l1  m2 + l1  m3 + #23 + #26 + #25 + #24,

  #5, #9],

  --
  |  - #36 - #35 - #21 - #22, #31 + #30 + #19 + #20, #2,
  --
```

```
                                      #28   #43   #42      --
  #5, ic2 + ic3 + #23 + #15 + #41 + --- + --- + ---, #10  |,
                                       2     2     2       --

  --                                 2       -- --
  |                           m3 l3           |  |
  |  -#21, #19, #6, #9, #10, ------ + ic3  |  |
  --                            4          -- --
```

where

```
                        2        2                   #28   #40   #17   #16
  #1 == ic1 + #11 + l1  m2 + l1  m3 + #23 + #26 + --- + --- + --- + ---
                                                     4     2     4     4

       #14   #13   #12                #27   #42
    + --- + --- + --- + #25 + #24 + --- + ---
      4     2     2                   2     4

                           #28   #40   #17   #16   #29   #18   #25   #24
  #2 == #23 + #15 + #41 + --- + --- + --- + --- + --- + --- + --- + ---
                            2     2     4     4     4     4     2     2

       #43   #27   #42
    + --- + --- + ---
      2     2     2

       10 m1 cos(q3)   10 m2 cos(q3)   10 m3 cos(q3)
  #3 == ------------- + ------------- + ------------- + #31 + #30 + #19
             2               2               2

    + #32 + #34 + #33 + #20

         10 m1 sin(q3)   10 m2 sin(q3)   10 m3 sin(q3)
  #4 == - ------------- - ------------- - ------------- - #36 - #35 - #21
                2               2               2

    - #37 - #39 - #38 - #22

                           #28   #40   #25   #24   #27   #42
  #5 == ic2 + #23 + #26 + --- + --- + --- + --- + --- + ---
                            4     2     2     2     2     4

              #40   #29   #43   #42
  #6 == #41 + --- + --- + --- + ---
               2     4     4     4

  #7 == #31 + #30 + #32 + #34 + #33

  #8 == - #36 - #35 - #37 - #39 - #38

       #40   #42
  #9 == --- + ---
        2     4

                     #43   #42
  #10 == ic3 + #41 + --- + ---
                      4     4

          2
         l1  m1
  #11 == ------
           4

  #12 == l0 l1 m3 cos(q4)

  #13 == l0 l1 m2 cos(q4)

  #14 == l0 l1 m1 cos(q4)

          2
         l2  m3
  #15 == ------
           2

  #16 == l0 l2 m3 cos(q4 + q5)

  #17 == l0 l2 m2 cos(q4 + q5)

  #18 == l0 l2 m3 cos(q5)

         l3 m3 cos(q3 + q5 + q6)
  #19 == ------------------------
                    2

         l2 m3 cos(q3 + q5)
  #20 == -------------------
                   2

         l3 m3 sin(q3 + q5 + q6)
```

```
   #21 == ------------------------
                     2

          l2 m3 sin(q3 + q5)
   #22 == ------------------
                    2

            2
          l2  m2
   #23 == ------
            4

   #24 == l1 l2 m3 cos(q5)

   #25 == l1 l2 m2 cos(q5)

            2
          l2  m3
   #26 == ------
            4

   #27 == l1 l2 m3 cos(q4 - q5)

           2
   #28 == l2  m3 cos(q4)

   #29 == l0 l3 m3 cos(q5 + q6)

          l2 m3 #44
   #30 == ----------
            2

          l2 m2 #44
   #31 == ----------
            2

          l1 m1 cos(q3 + q4)
   #32 == ------------------
                    2

   #33 == l1 m3 cos(q3 + q4)

   #34 == l1 m2 cos(q3 + q4)

          l2 m3 #45
   #35 == ----------
            2

          l2 m2 #45
   #36 == ----------
            2

          l1 m1 sin(q3 + q4)
   #37 == ------------------
                    2

   #38 == l1 m3 sin(q3 + q4)

   #39 == l1 m2 sin(q3 + q4)

   #40 == l1 l3 m3 cos(q5 - q4 + q6)

            2
          l3  m3
   #41 == ------
            4

   #42 == l2 l3 m3 cos(q4 - q6)

   #43 == l2 l3 m3 cos(q6)

   #44 == cos(q3 + q4 + q5)

   #45 == sin(q3 + q4 + q5)


The system Centrifugal-Coriolis matrix computed by the function TORO-Lagrange_ConservativeSys is

-- --                      / l0 m1 cos(q3)   l0 m2 cos(q3)   l0 m3 cos(q3)
|  |  0, 0, - qd4 #6 - qd3 | ------------- + ------------- + -------------
-- --                      \       2               2               2

                                                  \
   + #31 + #30 + #33 + #20 + #22 + #21 + #32 | - #12 - #9,
                                                  /

  - qd3 #6 - qd4 #6 - qd5 (#31 + #30), - qd4 (#31 + #30)

                                          l3 m3 qd3 #39   l3 m3 qd5 #39
```

```
  - qd3 (#31 + #30 + #33 + #32) - #12 - #9, - -------------- - --------------
                                                      2               2

       --
  - #9  |],
       --

  --                    / 10 m1 sin(q3)   10 m2 sin(q3)   10 m3 sin(q3)
  |  0, 0, - #13 - qd3 | -------------- + -------------- + -------------- + #35
  --                    \       2               2               2

                                    \
   + #34 + #37 + #23 + #25 + #24 + #36 | - qd4 #7 - #8,
                                    /

  - qd3 #7 - qd4 #7 - qd5 (#35 + #34), - qd3 (#35 + #34 + #37 + #36) - #13

                          l3 m3 qd3 #41   l3 m3 qd5 #41      --
   - qd4 (#35 + #34) - #8, - -------------- - -------------- - #8  |,
                                2               2              --

  [0, 0, - qd4 #3 - qd5 #4 - #10, - qd5 #5 - qd3 #3 - qd4 #3,

  - qd4 #5 - qd3 #4 - qd5 #4 - #10, - #11 - qd5 (#27 + #28 + #29 - #26)

   - #10],

  [0, 0, qd5 #2 - qd6 (#27 - #26) + qd3 #3, -qd5 (#19 + #18),

  qd3 #2 - qd4 (#19 + #18) - qd6 (#27 - #26) + qd5 #2,

  - qd3 (#27 - #26) - qd5 (#27 - #26) - qd6 (#27 - #26)],

  [0, 0, qd3 #4 - qd6 (#29 - #26) - qd4 #2, - qd3 #2 - qd4 #2 - qd5 #1,

  - qd4 #1 - qd6 (#29 - #26), - qd3 (#29 - #26) - qd5 (#29 - #26)

   - qd6 (#29 - #26)],

  --
  |  0, 0, qd4 (#27 - #26) + qd5 (#29 - #26) + #11,
  --

                                      l2 l3 m3 qd5 sin(q4 - q6)
  qd3 (#27 - #26) + qd4 (#27 - #26) - -------------------------,
                                                4

                                      l2 l3 m3 qd4 sin(q4 - q6)    -- --
  qd3 (#29 - #26) + qd5 (#29 - #26) - -------------------------, 0  |  |
                                                4                  -- --

where

                    2
         m3 sin(q4) l2    l3 m3 sin(q4 - q6) l2
  #1 == -------------- + ----------------------
                4                    4

  #2 == #17 - #27 - #19 - #18 + #14 + #26

                                 10 l1 m1 sin(q4)   10 l1 m2 sin(q4)
  #3 == #17 - #27 + #16 + #15 + ----------------- + -----------------
                                        4                  2

        10 l1 m3 sin(q4)
     + ----------------- + #14 + #26
                2

                                 10 l2 m3 sin(q5)
  #4 == #27 + #16 + #15 + #28 + ----------------- + #19 + #18 - #14
                                        4

  #5 == #17 + #16 + #15 + #19 + #18 + #26

  #6 == #31 + #30 + #20 + #22 + #21

  #7 == #35 + #34 + #23 + #25 + #24

        l3 m3 qd6 #41
  #8 == --------------
                2

        l3 m3 qd6 #39
  #9 == --------------
                2

  #10 == qd6 (#27 + #28 + #29 - #26)

  #11 == qd3 (#27 + #28 + #29 - #26)
```

```
#12 == qd5 (#31 + #30 + #33 + #32)

#13 == qd5 (#35 + #34 + #37 + #36)

       l1 l2 m3 sin(q4 - q5)
#14 == ---------------------
                  2

       l0 l2 m3 sin(q4 + q5)
#15 == ---------------------
                  4

       l0 l2 m2 sin(q4 + q5)
#16 == ---------------------
                  4

         2
       l2  m3 sin(q4)
#17 == --------------
              4

       l1 l2 m3 sin(q5)
#18 == ----------------
                  2

       l1 l2 m2 sin(q5)
#19 == ----------------
                  2

       l1 m1 cos(q3 + q4)
#20 == ------------------
                  2

#21 == l1 m3 cos(q3 + q4)

#22 == l1 m2 cos(q3 + q4)

       l1 m1 sin(q3 + q4)
#23 == ------------------
                  2

#24 == l1 m3 sin(q3 + q4)

#25 == l1 m2 sin(q3 + q4)

       l2 l3 m3 sin(q4 - q6)
#26 == ---------------------
                  4

       l1 l3 m3 sin(q5 - q4 + q6)
#27 == --------------------------
                     2

       l0 l3 m3 sin(q5 + q6)
#28 == ---------------------
                  4

       l2 l3 m3 sin(q6)
#29 == ----------------
                  4

       l2 m3 #38
#30 == ---------
             2

       l2 m2 #38
#31 == ---------
             2

       l2 m3 cos(q3 + q5)
#32 == ------------------
                  2

       l3 m3 #39
#33 == ---------
             2

       l2 m3 #40
#34 == ---------
             2

       l2 m2 #40
#35 == ---------
             2

       l2 m3 sin(q3 + q5)
#36 == ------------------
                  2

       l3 m3 #41
```

```
   #37 == ---------
                2

   #38 == cos(q3 + q4 + q5)

   #39 == cos(q3 + q5 + q6)

   #40 == sin(q3 + q4 + q5)

   #41 == sin(q3 + q5 + q6)
```

The system Centrifugal-Coriolis matrix computed by the function TORO-Lagrange_ConservativeSys is

```
/ 0 \
|   |
| 0 |
|   |
| 0 |
|   |
| 0 |
|   |
| 0 |
|   |
\ 0 /
```

EoM =

```
ic0*diff(q3(t), t, t) + icl*diff(q3(t), t, t) + icl*diff(q4(t), t, t) + (l0^2*ml*diff(q3(t), t, t))/4 + (l0^2*m2*diff(q3(t), t, t))/4 + (l1^2*ml*diff(
```

```
H =
[
[
[- (l0*ml*sin(q3))/2 - (l0*m2*sin(q3))/2 - (l0*m3*sin(q3))/2 - (l2*m2*sin(q3 + q4 + q5))/2 - (l2*m3*sin(q3 + q4 + q5))/2 - (l3*m3*sin(q3 + q5 + q6))/2
[                                                                                                                   - (l2*m2*sin(q3 + q4 + q5))/2 - (l
[                                                                                                                             - (l2*m2*sin(q3 + q4 +
[
C =
[0, 0,                                          - qd4*((l2*m2*cos(q3 + q4 + q5))/2 + (l2*m3*cos(q3 + q4 + q5))/2 + (l1*ml*cos(q3 + q4))/2 + l1*m2*c
[0, 0,                                          - qd5*((l2*m2*sin(q3 + q4 + q5))/2 + (l2*m3*sin(q3 + q4 + q5))/2 + (l3*m3*sin(q3 + q5 + q6))/2 + (l
[0, 0, - qd4*((l2^2*m3*sin(q4))/4 - (l1*l3*m3*sin(q5 - q4 + q6))/2 + (l0*l2*m2*sin(q4 + q5))/4 + (l0*l2*m3*sin(q4 + q5))/4 + (l0*l1*ml*sin(q4))/4 + (l
[0, 0,                                                                                                                    qd5*((l2^2*m3*sin(q4))/4 - (l1*l3
[0, 0,                                                                                                                                                      q
[0, 0,
G =
0
0
0
0
0
0
```

The system CoM (NPositionVector_SCM) computed by the function TORO-PlanarChainUGMB2SCM is

```
[[(2 m0 q1 + 2 ml q1 + 2 m2 q1 + 2 m3 q1 + l0 ml cos(q3) + l0 m2

   cos(q3) + l0 m3 cos(q3) + l2 m2 #3 + l2 m3 #3 + l3 m3

   cos(q3 + q5 + q6) + l1 ml cos(q3 + q4) + 2 l1 m2 cos(q3 + q4) + 2 l1 m3

   cos(q3 + q4) + l2 m3 cos(q3 + q5))/#1],

  [(2 m0 q2 + 2 ml q2 + 2 m2 q2 + 2 m3 q2 + l0 ml sin(q3) + l0 m2

   sin(q3) + l0 m3 sin(q3) + l2 m2 #2 + l2 m3 #2 + l3 m3

   sin(q3 + q5 + q6) + l1 ml sin(q3 + q4) + 2 l1 m2 sin(q3 + q4) + 2 l1 m3

   sin(q3 + q4) + l2 m3 sin(q3 + q5))/#1]]

where

   #1 == (m0 + ml + m2 + m3) 2

   #2 == sin(q3 + q4 + q5)

   #3 == cos(q3 + q4 + q5)
```

THIS PAGE INTENTIONALLY LEFT BLANK

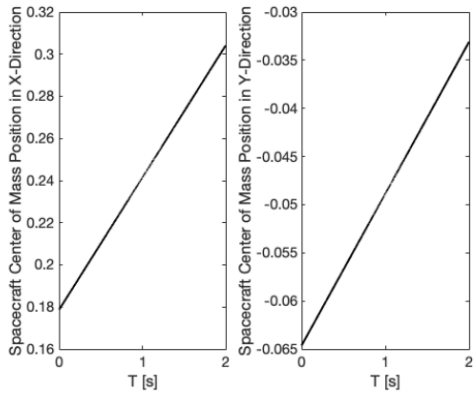
APPENDIX E. TORO TRAJECTORY PROPAGATION CODE

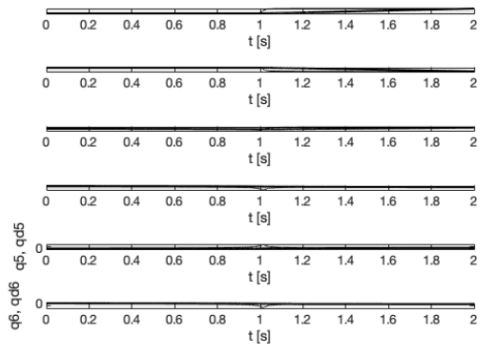
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (C) Dr. Marcello Romano, 2020-05-06
% Modified on 2020-08-20
% This function is part of TORO: Tool Set for Orbital RObotics
%
% EXAMPLE 1: Numerical experiments of motion
%
% IMPORTANT: first you need to execute SCRIPT 1
%
% This script performs numerical simulation by using the automatically
% generated H, C, G -but odefun is not automatically generated!- (this is referred to as method2)
%
% This method should be advantageous for large dof system where the
% function odeToVectorField get stuck
%
% Of course in those cases the Articulated Body Method approach might be a better
% option
clear all

close all

%publishing:
% publish('SCRIPT2_SIM_Method2_UG_Planar_SC_R2.m','pdf')
%********************
% START USER INPUT
%********************

%USERINPUT:
%this system name: you can give any name here

sysname='PERSEUS_SC_R3';

%USERINPUT:
%here the name used in the SymbolEoM file need to be used
sysclass='PERSEUS_SC_R3'

%USERINPUT:
%HERE GO THE PARAMETERS required in TOROGen_sysclass_odefun.m after Y

%l1,l2,m1,m2,ic1,ic2,X1,X2,X3,X4

%n in front stands for numerical value     MAKE SURE TO ADD THE NEW STUFF IN
nl0 = 0.27; %m
nl1 = 0.1; %m
nl2 = 0.1; %m
nl3 = 0.2 + (0.27/2); %m
%ng= 9.8; %m/s^2 masses go here
nm0 = 9.882 + 0.3; %kg
nm1 = 0.140 ; %kg
nm2 = 0.140 ; %kg
nm3 = 0.140 + 9.882; %kg
%nmr1 = 5; %kg
%nmr2 = nmr1; %kg
nic0 = 0.2527 + PrMOI(0.14,0.05,0.2); %kg m^2 principle moment of ineria of body about axis normal to the plane
nic1 = PrMOI(0.1,0.04,nm1); %kg m^2
nic2 = PrMOI(0.1,0.04,nm1); %kg m^2
nic3 = PrMOI(0.1,0.04,nm1)+0.2527; %kg m^2

nX1 = 0; %Nm Center of Mass
nX2 = 0; %Nm Center of Mass
nX3 = 0; %N
nX4 = 0.75; %N
nX5 = 0.75; %N
nX6 = 0.75; %N



%[Xi1;Xi2] = @GeneralizedTorques(l1 l2 m1 m2 mr1 mr2 ic1 ic2 ir1 ir2 kr1 kr2)

%USERINPUT
%report the name of the parameters above in the following row matrix
%WARNING: the value of the parameters need to be specified in the same
%order they are appearing in the arguments of TOROGen_sysclass_odefun.m
%after Y AND in the functions H, C, G
sysparameters=[nl0 nl1 nl2 nl3 nm0 nm1 nm2 nm3 nic0 nic1 nic2 nic3]   %EDITED THIS

%USER INPUT: final time [s]
finaltime=2; % maneuver to match PERSEUS experiment push

%USER INPUT: INITIAL CONDITIONS
% WARNING: they need to be in the following order:
% [q1(0), qd1(0), q2(0), qd2(0),...,qn(0), qdn(0)]
%let us call x0 and y0 the coordinates of \vect{r}_0 in \vectorbase{N}
%q1=x0
%q2=y0
%q3=theta0
%q4=theta1
%Y0      = [0 0 0 0 0 0.4 0 1]; %rad
```

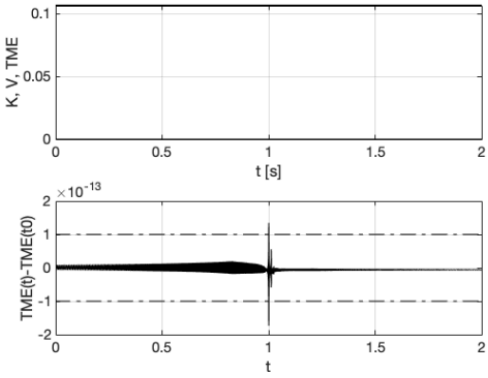
```
% x0 xd0 y0 yd0 theta0 thetad0 theta1 thetad1 - are d's derivatives?
%Y0      = [1 0, 1 0, 0 0., 0 0.1]; %rad
%Y0      = [1 -2, 1 2, 1.4 3, 2 -1]; %rad
% Y0      = [1 2 2 3 4 5 6 7 4 3 2 1]; this one was used %EDITED THIS TOO

% SYNTAX [x0 xd0 y0 yd0 plane_spin plane_updn th0 thd0 th1 thd1 th2 thd2 xb0 xbd0 yb0 ybd0]


Y0 = [0 0 0 0 0 0 (pi/4) (-pi/8) (-pi/2) (pi/4) (pi/4) (-pi/8)]; % initial state and velocity to complete coast in 1.7 sec
%Y0      = [1 0.1, 1 0.2, 0 0.3, 0 0.4]; %rad
%Y0      = [6 7 4 5 1 2 2 3];

numberofbodies = 3;

tspan   = [0 finaltime];

%TOROGen_FS_PSC_R1_odefun(0,Y0,nl0, nl1, nm0, nm1, nic0, nic1, nX1, nX2, nX3, nX4)


%USER INPUT: Rel and Abs Tolerance for the ODE integrator
integratorTOL=1e-13;


%whichodefun=1; %1: use odefun via Hinv, 2: use odefun generated using ode2vectorfield

%********************
% END USER INPUT
%********************


% WARNING:
% DO NOT MODIFY ANY CODE UNDERNEATH, UNLESS THERE ARE ERRORS
% OF EXECUTION AND YOU ARE SURE THAT THE INPUT ABOVE HAS BEEN PREPARED
% ACCORDING TO THE INSTRUCTIONS
close all

syspar_arg = mat2cell(sysparameters,1,ones(1,numel(sysparameters)))

tspan   = [0 finaltime];
options = odeset('RelTol',integratorTOL,'AbsTol',integratorTOL);

% Use created .m file to solve DE
%[t, Y]  = ode45(eval(strcat('@TOROGen_',sysclass,'_odefun')),tspan,Y0,options,syspar_arg{:})
%[t, Y]  = ode45(eval(strcat('@TOROGen_',sysclass,'_odefun')),tspan,Y0,options,sysparameters_string)'
par_commaseparatedlist= sprintf('%.15f,' , sysparameters);
par_commaseparatedlist = par_commaseparatedlist(1:end-1);
tic
%if whichodefun==1
    [t, Y]  = ode45(@odefun_via_Hinv,tspan,Y0,options,par_commaseparatedlist,sysclass, @GenForceFun);
%else
%    eval(strcat('[t, Y]  = ode45(@TOROGen_UG_Planar_SC_R1_odefun_Ode2VF,tspan,Y0,options,',par_commaseparatedlist,',@GenForceFun)'));
%end

numdof= size(Y0,2)/2;

PlotFigures

time_to_run_simulation=toc
```

```
sysclass =
    'PERSEUS_SC_R3'
sysparameters =
  Columns 1 through 7
    0.2700    0.1000    0.1000    0.3350   10.1820    0.1400    0.1400
  Columns 8 through 12
   10.0220    0.2538    0.0004    0.0004    0.2531
syspar_arg =
  1×12 cell array
  Columns 1 through 5
    {[0.2700]}    {[0.1000]}    {[0.1000]}    {[0.3350]}    {[10.1820]}
  Columns 6 through 10
    {[0.1400]}    {[0.1400]}    {[10.0220]}    {[0.2538]}    {[4.0600e-04]}
  Columns 11 through 12
    {[4.0600e-04]}    {[0.2531]}


**********************
CAPTION OF FigurePERSEUS_SC_R3_NSysCOM:
Evolution in time of the coordinates of the system CoM on the N CCS
each subplot is showing a gen. coordinate in darker line and the corresponding generalized velocity in lighter line.
**********************



**********************
CAPTION OF FigurePERSEUS_SC_R3_qs_and_qds:
Evolution in time of the generalized coordinates and velocites
```

```
each subplot is showing a gen. coordinate in darker line and the corresponding generalized velocity in lighter line.
(Unit of measurement is left unspecified as it can be different for different systems)
************************


************************
CAPTION OF FigurePERSEUS_SC_R3_ENERGIES:
Evolution in time of the Energies
The first subplot is representing the Kinetic energy in full darker line, the potential energy in full lighter line and the total mech. energy in dash
The second subplot is representing in full line the Total Mechanical Energy variation relative to the initial one and in dash-dot line +/- the integra
(Unit of measurement of the Energies is [Nm = Joule] if IS units are used for the qs and qds)
************************


time_to_run_simulation =
    2.8940
```







Published with MATLAB® R2021a

APPENDIX F. C++ DYNAMIXEL ACTUATION CODE.

The following code was adapted from [35].

```
// Modified by ENS I.A. Hardy, USN
// Adapted from Dynamixel SDK Code
// as written and published by
// Ryu Woon Jung
// As listed below


/*******************************************************************************
* Copyright 2017 ROBOTIS CO., LTD.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*******************************************************************************/

/* Author: Ryu Woon Jung (Leon) */

//
// *********     Bulk Read and Bulk Write Example      *********
//
//
// Available Dynamixel model on this example : All models using Protocol 2.0
// This example is tested with two Dynamixel PRO 54-200, and an USB2DYNAMIXEL
// Be sure that Dynamixel PRO properties are already set as %% ID : 1 and 2 /
 Baudnum : 1 (Baudrate : 57600)
//

#if defined(__linux__) || defined(__APPLE__)
#include <fcntl.h>
#include <termios.h>
#define STDIN_FILENO 0
#elif defined(_WIN32) || defined(_WIN64)
#include <conio.h>
#endif

#include <stdlib.h>
#include <stdio.h>

#include "dynamixel_sdk.h"                                  // Uses Dynamixel
 SDK library

#include <iostream>
```

```
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <bits/stdc++.h>
#include <typeinfo>

// Control table address
#define ADDR_XH430_TORQUE_ENABLE          64                 // Control table
 address is different in Dynamixel model
#define ADDR_XH430_LED_RED                65
#define ADDR_XH430_GOAL_POSITION          116
#define ADDR_XH430_PRESENT_POSITION       132

// Data Byte Length
#define LEN_XH430_LED_RED                 1
#define LEN_XH430_GOAL_POSITION           4
#define LEN_XH430_PRESENT_POSITION        4

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

// Default setting
#define DXL1_ID                         1                  // Dynamixel#1 ID:
 1
#define DXL2_ID                         2                  // Dynamixel#2 ID:
 2
#define DXL3_ID                         3                  // Dynamixel#1 ID:
 3
#define DXL4_ID                         4                  // Dynamixel#2 ID:
 4
#define BAUDRATE                        57600
#define DEVICENAME                      "/dev/ttyUSB0"      // Check which
 port is being used on your controller
                                                           // ex) Windows:
                                                    "COM1"   Linux:
                                                    "/dev/ttyUSB0" Mac:
                                                    "/dev/tty.usbserial-*"

#define TORQUE_ENABLE                   1                  // Value for
 enabling the torque
#define TORQUE_DISABLE                  0                  // Value for
 disabling the torque
#define DXL_MINIMUM_POSITION_VALUE      0             // Dynamixel will rotate
 between this value
#define DXL_MAXIMUM_POSITION_VALUE      4095              // 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.)
#define DXL_MOVING_STATUS_THRESHOLD     10                 // Dynamixel
 moving status threshold
```

```
#define ESC_ASCII_VALUE                 0x1b

int getch()
{
#if defined(__linux__) || defined(__APPLE__)
  struct termios oldt, newt;
  int ch;
  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  ch = getchar();
  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  return ch;
#elif defined(_WIN32) || defined(_WIN64)
  return _getch();
#endif
}

int kbhit(void)
{
#if defined(__linux__) || defined(__APPLE__)
  struct termios oldt, newt;
  int ch;
  int oldf;

  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
  fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

  ch = getchar();

  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  fcntl(STDIN_FILENO, F_SETFL, oldf);

  if (ch != EOF)
  {
    ungetc(ch, stdin);
    return 1;
  }

  return 0;
#elif defined(_WIN32) || defined(_WIN64)
  return _kbhit();
#endif
}
```

```
//int testarray[4][11];

using namespace std;

int main()
{
    // Initialize PortHandler instance
  // Set the port path
  // Get methods and members of PortHandlerLinux or PortHandlerWindows
  dynamixel::PortHandler *portHandler =
   dynamixel::PortHandler::getPortHandler(DEVICENAME);

  // Initialize PacketHandler instance
  // Set the protocol version
  // Get methods and members of Protocol1PacketHandler or
   Protocol2PacketHandler
  dynamixel::PacketHandler *packetHandler =
   dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);

  // Initialize GroupBulkWrite instance
  dynamixel::GroupBulkWrite groupBulkWrite(portHandler, packetHandler);

  // Initialize GroupBulkRead instance
  dynamixel::GroupBulkRead groupBulkRead(portHandler, packetHandler);

    // READ PERSEUS TEST FILE (.txt) format


  int dxl_goal_position[4][11];

  string motor1, motor2, motor3, motor4; // parameters we want vectors for
  vector<int>M1;
  vector<int>M2;
  vector<int>M3;
  vector<int>M4;



  FILE *fp;
   char c_vec;
   string target_array;
   int readcount = 0;



   fp = fopen("push2.txt","r");//("PERSEUStest.txt","r"); //////READS THE
    TEXT FILE TO A STRING OF DIGITS [1xn]
   while(1)
   {
      c_vec = fgetc(fp);
```

```
   if( feof(fp) )
   {
      break ;
   }

   target_array += c_vec;
   printf("%c", c_vec);
   readcount++;

}
fclose(fp);

cout << "CANDIDATE TRAJECTORY" << endl;
cout << target_array << endl;

 stringstream stream(target_array); ///////CONVERTS STRING OF DIGITS TO INT BY
  RECOGNIZING SEPARATION BY SPACE KEYSTROKE

 int target_as_num[44];
 int numcount = 0;



 while(stream){
     int n;
     stream>>n;
     target_as_num[numcount] = n;
     cout<<n<<endl;
     cout<<typeid(n).name() <<endl;
     numcount++;
 }
 //cout<< target_as_num <<endl;

     printf("%d\n",target_as_num[43]);
     cout<<typeid(target_as_num).name() <<endl;

     int motor1_goal[11];
     int motor2_goal[11];
     int motor3_goal[11];
     int motor4_goal[11];


     for(int vec_col_count = 0;vec_col_count <11;vec_col_count++)
     {
       motor1_goal[vec_col_count] = target_as_num[vec_col_count];
       motor2_goal[vec_col_count] = target_as_num[11+vec_col_count];
       motor3_goal[vec_col_count] = target_as_num[22+vec_col_count];
       motor4_goal[vec_col_count] = target_as_num[33+vec_col_count];

     }
```

```
      for (int i = 0; i < 11; i++)
      {
      std::cout << motor1_goal[i] << ' ';
      }
```

```
int dxl_comm_result = COMM_TX_FAIL;             // Communication result
bool dxl_addparam_result = false;               // addParam result
bool dxl_getdata_result = false;                // GetParam result


uint8_t dxl_error = 0;                          // Dynamixel error
uint8_t param_goal_position1[4];
uint8_t param_goal_position2[4];
uint8_t param_goal_position3[4];
uint8_t param_goal_position4[4];

int32_t dxl1_present_position = 0;              // Present position #1
int32_t dxl2_present_position = 0;              // Present position #2
int32_t dxl3_present_position = 0;              // Present position #1
int32_t dxl4_present_position = 0;              // Present position #2

int dxl1_check_pres;
int dxl2_check_pres;
int dxl3_check_pres;
int dxl4_check_pres;

// Open port
if (portHandler->openPort())
{
  printf("Succeeded to open the port!\n");
}
else
{
  printf("Failed to open the port!\n");
  printf("Press any key to terminate...\n");
  getch();
  return 0;
}

// Set port baudrate
if (portHandler->setBaudRate(BAUDRATE))
{
  printf("Succeeded to change the baudrate!\n");
}
```

```
else
{
  printf("Failed to change the baudrate!\n");
  printf("Press any key to terminate...\n");
  getch();
  return 0;
}


// Enable Dynamixel#1 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL1_ID,
 ADDR_XH430_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  printf("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
  printf("Dynamixel#%d has been successfully connected \n", DXL1_ID);
}

// Enable Dynamixel#2 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL2_ID,
 ADDR_XH430_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  printf("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
  printf("Dynamixel#%d has been successfully connected \n", DXL2_ID);
}


  // Enable Dynamixel#3 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL3_ID,
 ADDR_XH430_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
```

```
  printf("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
  printf("Dynamixel#%d has been successfully connected \n", DXL3_ID);
}

// Enable Dynamixel#4 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL4_ID,
 ADDR_XH430_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  printf("%s\n", packetHandler->getRxPacketError(dxl_error));
}
else
{
  printf("Dynamixel#%d has been successfully connected \n", DXL4_ID);
}


// Add parameter storage for Dynamixel#1 present position
dxl_addparam_result = groupBulkRead.addParam(DXL1_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);
if (dxl_addparam_result != true)
{
  fprintf(stderr, "[ID:%03d] grouBulkRead addparam failed", DXL1_ID);
  return 0;
}

// Add parameter storage for Dynamixel#2 present position
dxl_addparam_result = groupBulkRead.addParam(DXL2_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);
if (dxl_addparam_result != true)
{
  fprintf(stderr, "[ID:%03d] grouBulkRead addparam failed", DXL2_ID);
  return 0;
}

  // Add parameter storage for Dynamixel#3 present position
dxl_addparam_result = groupBulkRead.addParam(DXL3_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);
if (dxl_addparam_result != true)
{
  fprintf(stderr, "[ID:%03d] grouBulkRead addparam failed", DXL3_ID);
  return 0;
}
```

```
// Add parameter storage for Dynamixel#4 present position
dxl_addparam_result = groupBulkRead.addParam(DXL4_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);
if (dxl_addparam_result != true)
{
  fprintf(stderr, "[ID:%03d] grouBulkRead addparam failed", DXL4_ID);
  return 0;
}


  int colcount = 0;

  while(colcount <11)
{
  printf("Press any key to continue! (or press ESC to quit!)\n");
  if (getch() == ESC_ASCII_VALUE)
    break;

  for(colcount = 0; colcount < 11; colcount++) // iterates through columns
   of testarray-->dxl_goal_position
  {
    cout << "Column Number" << colcount << endl ;

    cout << "Actuator 1 Count" << motor1_goal[colcount] << endl;
    cout << "Actuator 2 Count" << motor2_goal[colcount] << endl;
    cout << "Actuator 3 Count" << motor3_goal[colcount] << endl;
    cout << "Actuator 4 Count" << motor4_goal[colcount] << endl;


                // Allocate goal position values into byte arrays for each
                 actuator
    param_goal_position1[0] = DXL_LOBYTE(DXL_LOWORD(motor1_goal[colcount]));
    param_goal_position1[1] = DXL_HIBYTE(DXL_LOWORD(motor1_goal[colcount]));
    param_goal_position1[2] = DXL_LOBYTE(DXL_HIWORD(motor1_goal[colcount]));
    param_goal_position1[3] = DXL_HIBYTE(DXL_HIWORD(motor1_goal[colcount]));

    param_goal_position2[0] = DXL_LOBYTE(DXL_LOWORD(motor2_goal[colcount]));
    param_goal_position2[1] = DXL_HIBYTE(DXL_LOWORD(motor2_goal[colcount]));
    param_goal_position2[2] = DXL_LOBYTE(DXL_HIWORD(motor2_goal[colcount]));
    param_goal_position2[3] = DXL_HIBYTE(DXL_HIWORD(motor2_goal[colcount]));

    param_goal_position3[0] = DXL_LOBYTE(DXL_LOWORD(motor3_goal[colcount]));
    param_goal_position3[1] = DXL_HIBYTE(DXL_LOWORD(motor3_goal[colcount]));
    param_goal_position3[2] = DXL_LOBYTE(DXL_HIWORD(motor3_goal[colcount]));
    param_goal_position3[3] = DXL_HIBYTE(DXL_HIWORD(motor3_goal[colcount]));

    param_goal_position4[0] = DXL_LOBYTE(DXL_LOWORD(motor4_goal[colcount]));
    param_goal_position4[1] = DXL_HIBYTE(DXL_LOWORD(motor4_goal[colcount]));
    param_goal_position4[2] = DXL_LOBYTE(DXL_HIWORD(motor4_goal[colcount]));
    param_goal_position4[3] = DXL_HIBYTE(DXL_HIWORD(motor4_goal[colcount]));
```

```
// Add parameter storage for Dynamixel#1 goal position
dxl_addparam_result = groupBulkWrite.addParam(DXL1_ID,
 ADDR_XH430_GOAL_POSITION, LEN_XH430_GOAL_POSITION,
 param_goal_position1);
if (dxl_addparam_result != true)
{
  fprintf(stderr, "[ID:%03d] groupBulkWrite addparam failed", DXL1_ID);
  return 0;
}

// Add parameter storage for Dynamixel#2 goal position
dxl_addparam_result = groupBulkWrite.addParam(DXL2_ID,
 ADDR_XH430_GOAL_POSITION, LEN_XH430_GOAL_POSITION,
 param_goal_position2);
if (dxl_addparam_result != true)
{
  fprintf(stderr, "[ID:%03d] groupBulkWrite addparam failed", DXL2_ID);
  return 0;
}

// Add parameter storage for Dynamixel#1 goal position
dxl_addparam_result = groupBulkWrite.addParam(DXL3_ID,
 ADDR_XH430_GOAL_POSITION, LEN_XH430_GOAL_POSITION,
 param_goal_position3);
if (dxl_addparam_result != true)
{
  fprintf(stderr, "[ID:%03d] groupBulkWrite addparam failed", DXL3_ID);
  return 0;
}

// Add parameter storage for Dynamixel#2 goal position
dxl_addparam_result = groupBulkWrite.addParam(DXL4_ID,
 ADDR_XH430_GOAL_POSITION, LEN_XH430_GOAL_POSITION,
 param_goal_position4);
if (dxl_addparam_result != true)
{
  fprintf(stderr, "[ID:%03d] groupBulkWrite addparam failed", DXL4_ID);
  return 0;
}


// IMPORTANT NOTE ********** --> txPacket refers to the goal
 position as the host control computer passes that value to the dxl over
 serial (portHandler)

// PART 2: txRxPacket refers to the received present position value as
 the result of query from host control to dxl via portHandler


// Bulkwrite goal position and LED value
```

```
dxl_comm_result = groupBulkWrite.txPacket();
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}

/// prints string of result from comm packet

// Clear bulkwrite parameter storage
groupBulkWrite.clearParam();

do
{
  // Bulkread present position and LED status /// NOTE TXRX packet
   notation indicates that the function is receiving input as read from
   dxls

  // define comm result as whatever exists in txrxpacket as found in
   groupBuldRead data structure
  // if successful, prints the result (present) position as that exists
   within dxl_comm_result
  // else, throws error message tied to whichever dxl cannot communicate
   properly

  dxl_comm_result = groupBulkRead.txRxPacket();
  if (dxl_comm_result != COMM_SUCCESS)
  {
    printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
  }
  else if (groupBulkRead.getError(DXL1_ID, &dxl_error))
  {
    printf("[ID:%03d] %s\n", DXL1_ID,
     packetHandler->getRxPacketError(dxl_error));
  }
  else if (groupBulkRead.getError(DXL2_ID, &dxl_error))
  {
    printf("[ID:%03d] %s\n", DXL2_ID,
     packetHandler->getRxPacketError(dxl_error));
  }

  else if (groupBulkRead.getError(DXL3_ID, &dxl_error))
  {
    printf("[ID:%03d] %s\n", DXL3_ID,
     packetHandler->getRxPacketError(dxl_error));
  }
  else if (groupBulkRead.getError(DXL4_ID, &dxl_error))
  {
    printf("[ID:%03d] %s\n", DXL4_ID,
     packetHandler->getRxPacketError(dxl_error));
  }
  // READS PRESENT POSITION OF ACTUATOR CHAIN
```

```
// the next section of dxl_getdata_result's is to check if the
 relevant data [ID, addr and len of present position] are available
// if unavailable for any dxl, error is thrown

// Check if groupbulkread data of Dynamixel#1 is available
dxl_getdata_result = groupBulkRead.isAvailable(DXL1_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);
if (dxl_getdata_result != true)
{
  fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL1_ID);
  return 0;
}

// Check if groupbulkread data of Dynamixel#2 is available
dxl_getdata_result = groupBulkRead.isAvailable(DXL2_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);
if (dxl_getdata_result != true)
{
  fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL2_ID);
  return 0;
}

////REPLICATING GROUPBULKREAD CHECK //////////

// Check if groupbulkread data of Dynamixel#3 is available
dxl_getdata_result = groupBulkRead.isAvailable(DXL3_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);
if (dxl_getdata_result != true)
{
  fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL3_ID);
  return 0;
}

// Check if groupbulkread data of Dynamixel#4 is available
dxl_getdata_result = groupBulkRead.isAvailable(DXL4_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);
if (dxl_getdata_result != true)
{
  fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL4_ID);
  return 0;
}

////CHECK REPLICATION COMPLETE////////////////////


// Get DXL1 present position value
dxl1_present_position = groupBulkRead.getData(DXL1_ID,
 ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);

// Get DXL2 present position value
```

```
        //dxl2_led_value_read = groupBulkRead.getData(DXL2_ID,
         ADDR_XH430_LED_RED, LEN_XH430_LED_RED);
        dxl2_present_position = groupBulkRead.getData(DXL2_ID,
         ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);

        ////REPLICATE POSITION GET//////

              // Get DXL3 present position value
        dxl3_present_position = groupBulkRead.getData(DXL3_ID,
         ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);

        // Get DXL4 present position value
        //dxl2_led_value_read = groupBulkRead.getData(DXL2_ID,
         ADDR_XH430_LED_RED, LEN_XH430_LED_RED);
        dxl4_present_position = groupBulkRead.getData(DXL4_ID,
         ADDR_XH430_PRESENT_POSITION, LEN_XH430_PRESENT_POSITION);

        ////////REPLICATE POSITION GET COMPLETE/////

        //printf("[ID:%03d] Present Position 1 : %d \t [ID:%03d] LED Value:
         %d\n", DXL1_ID, dxl1_present_position, DXL2_ID, dxl2_led_value_read);
        printf("[ID:%03d] Present Position 1 : %d \t [ID:%03d] Present
         Position 2: %d \t ", DXL1_ID, dxl1_present_position, DXL2_ID,
         dxl2_present_position);
        printf("[ID:%03d] Present Position 3 : %d \t [ID:%03d] Present
         Position 4: %d \t ", DXL3_ID, dxl3_present_position, DXL4_ID,
         dxl4_present_position);

        dxl1_check_pres = (int) dxl1_present_position;
        dxl2_check_pres = (int) dxl2_present_position;
        dxl3_check_pres = (int) dxl3_present_position;
        dxl4_check_pres = (int) dxl4_present_position;


        int happyguy = (abs(dxl_goal_position[0][colcount] - dxl1_check_pres));
        cout << "Checksum value" << happyguy << endl;



      }while((abs(motor1_goal[colcount] - dxl1_check_pres) >
       DXL_MOVING_STATUS_THRESHOLD)||(abs(motor2_goal[colcount] -
       dxl2_check_pres) >
       DXL_MOVING_STATUS_THRESHOLD)||(abs(motor3_goal[colcount] -
       dxl3_check_pres) >
       DXL_MOVING_STATUS_THRESHOLD)||(abs(motor4_goal[colcount] -
       dxl4_check_pres) > DXL_MOVING_STATUS_THRESHOLD));

    }
}
```

```
// UPON WHILE LOOP TERMINATION, PROGRAM PROCEEDS TO, disable torque CLOSE
 PORT AND EXIT CLEANLY

// Disable Dynamixel#1 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL1_ID,
 ADDR_XH430_TORQUE_ENABLE, TORQUE_DISABLE, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  printf("%s\n", packetHandler->getRxPacketError(dxl_error));
}

// Disable Dynamixel#2 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL2_ID,
 ADDR_XH430_TORQUE_ENABLE, TORQUE_DISABLE, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  printf("%s\n", packetHandler->getRxPacketError(dxl_error));
}

  // Disable Dynamixel#3 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL3_ID,
 ADDR_XH430_TORQUE_ENABLE, TORQUE_DISABLE, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  printf("%s\n", packetHandler->getRxPacketError(dxl_error));
}

// Disable Dynamixel#4 Torque
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL4_ID,
 ADDR_XH430_TORQUE_ENABLE, TORQUE_DISABLE, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS)
{
  printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
}
else if (dxl_error != 0)
{
  printf("%s\n", packetHandler->getRxPacketError(dxl_error));
}
```

```
  // Close port
  portHandler->closePort();

  return 0;
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] S. Kalan *et al.*, "History of Robotic Surgery," *J. Robot. Surg.*, vol. 4, no. 3, pp. 141–147, 2010, doi: 10.1007/s11701-010-0202-2.

[2] I. Iglesias, M. A. Sebastián, and J. E. Ares, "Overview of the State of Robotic Machining: Current Situation and Future Potential," *Procedia Eng.*, vol. 132, pp. 911–917, 2015, doi: https://doi.org/10.1016/j.proeng.2015.12.577.

[3] H. S. Cho, H. J. Warnecke, and D. G. Gweon, "Robotic Assembly: a Synthesizing Overview," *Robotica*, vol. 5, no. 2, pp. 153–165, 1987, doi: DOI: 10.1017/S0263574700015332.

[4] V. Y. Rutkovsky, I. N. Krutova, V. M. Sukhanov, and V. M. Glumov, "Graph Models of Orbital Assembly and Dynamics of a Large Space Structure," *IFAC Proc. Vol.*, vol. 37, no. 6, pp. 77–82, 2004, doi: 10.1016/s1474-6670(17)32153-5.

[5] R. M. Muller, "Assembly and servicing of a large telescope at the International Space Station," *IEEE Aerosp. Conf. Proc.*, vol. 7, pp. 3611–3619, 2002, doi: 10.1109/AERO.2002.1035337.

[6] T. McMahan and V. Neal, *Repairing Solar Max: The Solar Maximum Repair Mission*. Goddard Spaceflight Center, Greenbelt, MD: Office of Space Science and Applications, 1984.

[7] D. J. Shayler and D. M. Harland, *The Hubble Space Telescope: From Concept to Success*. Springer, 2015.

[8] P. Laryssa, E. Lindsay, and O. Layi, "International space station robotics: a comparative study of ERA, JEMRMS and MSS," *... Robot. ...*, pp. 1–8, 2002, [Online]. Available: http://robotics.estec.esa.int/ASTRA/Astra2002/Papers/astra2002_1.3-1.pdf.

[9] N. T. Redd, "Bringing Satellites Back from the Dead: Mission Extension Vehicles Give Defunct Spacecraft a New Lease on Life," *IEEE Spectr.*, vol. 57, no. 8, pp. 6–7, 2020, doi: 10.1109/MSPEC.2020.9150540.

[10] C. G. Henshaw, "The DARPA Phoenix Spacecraft Servicing Program : Overview and Plans for Risk Reduction," *Proc. 'i-SAIRAS 2014 - 12th Int. Symp. Artif. Intell. Robot. Autom. Sp.*, pp. 1–9, 2014, [Online]. Available: http://goo.gl/6j7xGO.

[11] J. M. Gregory, J. S. Kang, M. Sanders, and D. Wenberg, "Characterization of Semi-autonomous On-orbit Assembly CubeSat Constellation," 2019.

[12] D. Wenberg, A. Hardy, T. Lai, C. Wellins, and J. Kang, “Advancing On-Orbit Assembly With ISAR,” in *32nd Annual AIAA/USU Conference on Small Satellites*, 2018, pp. 1–8.

[13] E. Stoll *et al.*, “On-orbit servicing,” *IEEE Robot. Autom. Mag.*, vol. 16, no. 4, pp. 29–33, 2009, doi: 10.1109/MRA.2009.934819.

[14] R. Zappulla, J. Virgili-Llop, C. Zagaris, H. Park, A. Sharp, and M. Romano, “Floating spacecraft simulator test bed for the experimental testing of autonomous guidance, navigation, and control of spacecraft proximity maneuvers and operations,” *AIAA/AAS Astrodyn. Spec. Conf. 2016*, no. September, 2016, doi: 10.2514/6.2016-5268.

[15] J. Virgili-Llop, C. Zagaris, R. Zappulla, A. Bradstreet, and M. Romano, “Laboratory experiments on the capture of a tumbling object by a spacecraft-manipulator system using a convex-programming-based guidance,” *Adv. Astronaut. Sci.*, vol. 162, pp. 787–807, 2018.

[16] M. Romano, D. A. Friedman, and T. J. Shay, “Laboratory experimentation of autonomous spacecraft approach and docking to a collaborative target,” *Journal of Spacecraft and Rockets*, vol. 44, no. 1. American Institute of Aeronautics and Astronautics, Reston, Va. :, p. 164, 2007, doi: 10.2514/1.22092.

[17] T. Smith *et al.*, “Astrobee: A New Platform for Free-Flying Robotics on the ISS,” *Intell. Robot. Group, NASA Ames Res. Cent.*, 2016.

[18] E. Ackerman, “How NASA’s Astrobee Robot Is Bringing Useful Autonomy to the ISS,” *IEEE Spectrum*, 2017. https://spectrum.ieee.org/automaton/robotics/space-robots/how-nasa-astrobee-robot-is-bringing-useful-autonomy-to-the-iss (accessed May 05, 2021).

[19] M. C. S. 3rd C. Weston, Leonard, “Robots in Space!? NPS & NASA Team Up on ‘Astrobatics’ Project to Advance Spacecraft Robotics,” *All Hands Mag.*, no. June, p. 9, 2021.

[20] M. Garcia, “A pair of Astrobee robotic assistants are pictured flying around,” 2021. http://www.nasa.gov/image-feature/a-pair-of-astrobee-robotic-assistants-are-pictured-flying-around (accessed May 05, 2021).

[21] V. Glover, “NPS Astrobatics Session 1.” https://twitter.com/astrovicglover/status/1372678416646955013 (accessed May 05, 2021).

[22] H. Woo, O. Rico Perez, S. Chesi, and M. Romano, “CubeSat three axis simulator(CubeTAS),” *AIAA Model. Simul. Technol. Conf. 2011*, no. August, pp. 82–89, 2011, doi: 10.2514/6.2011-6271.

[23] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics Modelling, Planning and Control*. London: Springer-Verlag London Limited, 2010.

[24] P. I. Corke, "A Simple and Systematic Approach to Assigning Denavit-Hartenberg Parameters," *IEEE Trans. Robot.*, vol. 23, no. 3, pp. 590–594, 2007, doi: 10.1109/TRO.2007.896765.

[25] A. E. R. Jonge, "THE CORRELATION OF HINGED FOUR-BAR STRAIGHT-LINE MOTION DEVICES BY MEANS OF THE ROBERTS THEOREM AND A NEW PROOF OF THE LATTER," *Ann. N. Y. Acad. Sci.*, vol. 84, no. 3, pp. 77–145, 1960, doi: 10.1111/j.1749-6632.1960.tb42784.x.

[26] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley & Sons, Inc., 2006.

[27] P. Corke, "Robotics Toolbox." https://petercorke.com/toolboxes/robotics-toolbox/ (accessed May 07, 2021).

[28] M. Romano, "Tool Set for Orbital Robotics." 2021.

[29] The MathWorks Inc., "Solve nonstiff differential equations — medium order method - MATLAB ode45." https://www.mathworks.com/help/matlab/ref/ode45.html (accessed Jun. 15, 2021).

[30] C. Safbom, S. Kwok-Choon, and M. Romano, "DESIGN, TESTING, AND ANALYSIS OF SELF-TOSS HOPPING MANEUVERS OF ASTROBEE AT NPS AND NASA AMES RESEARCH CENTER," 2020.

[31] ROBOTIS, "XH430-W210-T/R," *ROBOTIS e-Manual*, 2021. https://emanual.robotis.com/docs/en/dxl/x/xh430-w210/ (accessed May 06, 2021).

[32] Robotis Co LTD., "U2D2." https://emanual.robotis.com/docs/en/parts/interface/u2d2/ (accessed Jun. 14, 2021).

[33] Arduino, "Getting started with the Arduino Due | Arduino," 2021. https://www.arduino.cc/en/Guide/ArduinoDue (accessed Jun. 14, 2021).

[34] Arduino, "Getting Started with the Arduino WiFi Shield | Arduino," 2021. https://www.arduino.cc/en/Guide/ArduinoWiFiShield (accessed Jun. 14, 2021).

[35] Robotis Co LTD. and R. W. Jung, "DYNAMIXEL SDK," 2017. https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/ (accessed Jun. 14, 2021).

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