ABSTRACT

Title of Document: DESIGN AND EVALUATION OF END-EFFECTORS FOR AUTONOMOUS SAMPLING

Craig Michael Lewandowski, M.S., 2008

Directed By: Associate Professor David L. Akin
Department of Aerospace Engineering

Autonomous underwater vehicles are becoming increasingly prevalent, and their emergence will allow for the execution of previously unfeasible underwater missions. These missions include seeking naval mines, navigation and mapping of ocean features, and sampling on the ocean floor at extreme depths. One method to achieve this latter objective involves the attachment of a robotic manipulator to an underwater vehicle and use of the manipulator to collect specimens and deposit them in containers. This thesis focuses on the design and testing of an end-effector to be used on such a manipulator. End-effectors previously utilized in underwater robotics were evaluated during the conceptualization of the selected tool design. These evaluations in conjunction with manipulator interface requirements were used to produce the end-effector design that was constructed and subsequently tested. In addition, sample containers were designed and fabricated, and kinematics software used to determine sample container position, orientation, and quantity was developed.

DESIGN AND EVALUATION OF END-EFFECTORS FOR AUTONOMOUS SAMPLING

By

Craig Michael Lewandowski

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2008

Advisory Committee:
Associate Professor David L. Akin, Chair
Assistant Professor Raymond J. Sedwick
Adjunct Professor Craig R. Carignan

Acknowledgements

First and foremost, I would like to thank Dr. David Akin for the opportunity to work at the Space Systems Laboratory. The facility is truly unique, and I can say with certainty that I could not have had the same graduate experience at any other institution. Moreover, his insight and direction throughout these past two years has been greatly appreciated. Similarly, I would like to thank Dr. Sedwick and Dr. Carignan for serving on my committee and contributing their ideas and support.

Additionally, I would like to thank my team of undergraduate research assistants, Brandon, Courtney, and Jordan, who worked with me even when things did not go completely as planned. Also, the contributions made by all of the SSL undergraduate and graduate students and the time given by Stephen Roderick were all very much valued. Likewise, the assistance of Walt Smith and Mike Perna was especially appreciated as their answers to many of my questions helped make this research possible. Finally, I would like to thank my office companion Barrett, without whom my time at the University of Maryland would not have been nearly as enjoyable.

This research was conducted at the Space Systems Laboratory, part of the Aerospace Engineering Department of the A. James Clark School of Engineering at the University of Maryland, College Park. It was supported through the Institute for Dexterous Space Robotics.

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

AVATAR	Autonomous Vision Application for Target Acquisition and Ranging
ASTEP	Astrobiology Science and Technology for Exploring Planets
AUV	Autonomous Underwater Vehicle
CAD	Computer-Aided Design
CSSF	Canadian Scientific Submersible Facility
DH	Denavit-Hartenberg
DOF	Degrees of Freedom
EMF	Electromotive Force
EVA	Extravehicular Activity
FBD	Free Body Diagram
FOS	Factor of Safety
FOV	Field of View
GUI	Graphical User Interface
JAGUAR	Just Another Great Underwater Autonomous Robot
ISS	International Space Station
NASA	National Aeronautics and Space Administration
NBRF	Neutral Buoyancy Research Facility
NBV	Neutral Buoyancy Vehicle
OCP	Overcurrent Protection
OTCM	Orbital Replacement Unit / Tool Changeout Mechanism
PJM	Parallel Jaw Mechanism
ROV	Remotely Operated Vehicle
SAMURAI	Subsea Arctic Manipulator for Underwater Retrieval and Autonomous Intervention
SHCS	Socket Head Cap Screw
SSL	Space Systems Laboratory
TERPS	Tool for EVA or Robotic Planetary Sampling
UMD	University of Maryland
WHOI	Woods Hole Oceanographic Institute

A	Amperes
cm	Centimeter
in	Inches
ft	Feet
g	Gravitational Acceleration
h	Maximum Cam Follower Displacement
kg	Kilograms
ksi	Kilopounds per Square Inch
lb	Pounds (force)
lb_m	Pounds (mass)
m	Meters
mm	Millimeter

MPa	Megapascals
N	Newtons
psi	Pounds per Square Inch
rad	Radians
rpm	Revolutions per Minute
V	Volts
x	Cartesian Distance from Spiral Plate Center to Cam Groove
y	Cam Follower Vertical Displacement
y'	Cam Follower Velocity
y''	Cam Follower Acceleration
α_j	Angle of Joint Rotation
β	Cam Displacement Angle
θ	Angle of Joint Rotation

Chapter 1
Introduction

The scientific community has been employing submersibles for decades in a variety of applications including inspection of subsea structures, recovery operations, and sample collection. In general, these submersibles have been remotely operated vehicles (ROVs), enabled by receiving power and commands from a surface vessel through an umbilical. However, operation at extreme depths and in challenging environments such as an ice field presents serious complications for tethered ROVs.

The emergence of autonomous underwater vehicles (AUVs) has made such difficult missions possible. As AUVs are not constrained by an umbilical, they have considerably more operational freedom. As the field continues to develop, robotic manipulators mounted to AUVs are expected to have an increasingly prominent role [1]. In order for such manipulators to be truly effective, they will need to have end-effectors capable of achieving demanding tasks while operating in harsh environments.

This thesis focuses on the development of an end-effector capable of reliably collecting samples from the seafloor. The design of the storage containers to be mounted to an AUV and house the collected specimens is also presented. This end-effector is actuated with a roll degree of freedom (DOF) on an existing 6-DOF manipulator, and the kinematic effects of this actuation method are also investigated.

1.1 Motivation

The field of astrobiology addresses the issues of how life began and evolved, whether or not life exists elsewhere in the universe, and the future of life on Earth and beyond [2]. With its primary objective of space exploration, the National Aeronautics and Space Administration (NASA) has become increasingly active in this field in recent years. To increase its astrobiology involvement, NASA formed the Astrobiology Science and Technology for Exploring Planets (ASTEP) program.

ASTEP funds research efforts on Earth designed to ascertain the best methods to be used to search for life elsewhere in the Solar System. These places include Europa and Enceladus, geologically-active moons of Jupiter and Saturn, respectively, which are believed to be composed of liquid below the surface. These remote moons are characterized by harsh environmental conditions, and to prepare for such missions, the ASTEP research efforts are carried out in terrestrial analogs.

Similarly extreme conditions exist in many locations on Earth, and one of the most interesting is the Gakkel Ridge. The Gakkel Ridge, located beneath the Arctic ice cap as shown in Figure 1.1, represents the slowest spreading ridge on the planet and extends to a depth of 5500 m. This particular location is of considerable interest as evidence obtained from hydrocasts has suggested the presence of a hydrothermal vent field. These deep volcanic vents are biologically rich environments that thrive despite the absence of sunlight. Potential sampling targets include tubeworms and shrimp, as depicted in Figure 1.2.

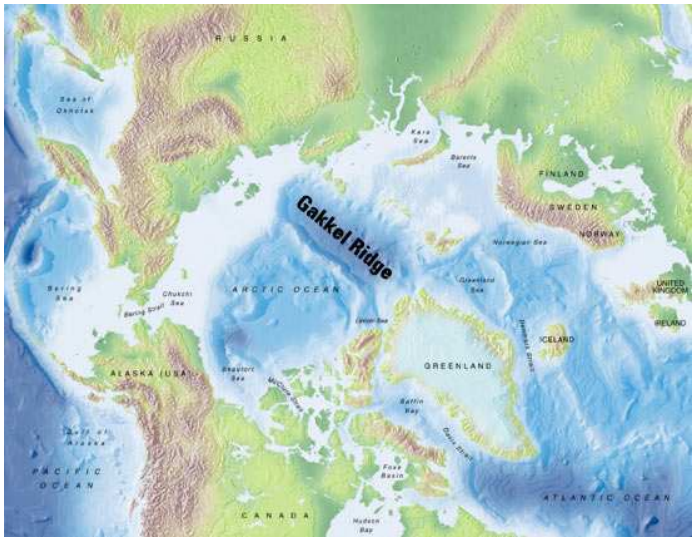


Figure 1.1: Map showing location of Gakkel Ridge (from [3]).


Figure1.2: Potential sampling targets: tubeworms (left)
and shrimp (right) (from [4]).

In an effort to develop a vehicle capable of autonomously observing and sampling biological specimens in such an environment, the Space Systems Laboratory (SSL) at the University of Maryland (UMD) joined with the Woods Hole Oceanographic Institute (WHOI). WHOI was tasked with the design and fabrication of the underwater vehicle, which evolved into JAGUAR, Just Another Great Underwater Autonomous Robot.

In parallel, the SSL applied its robotics experience towards the development of a 6-DOF robotic manipulator. SAMURAI, the Subsea Arctic Manipulator for

Underwater Retrieval and Autonomous Interventions, pictured in Figure 1.3, would mount to the WHOI vehicle and serve as the device to physically collect the specimens, as depicted in Figure 1.4. The SAMURAI/JAGUAR system would submerge to a depth of 6000 m where it is to spend up to 36 hours exploring and collecting samples.

Figure 1.3: SAMURAI manipulator.

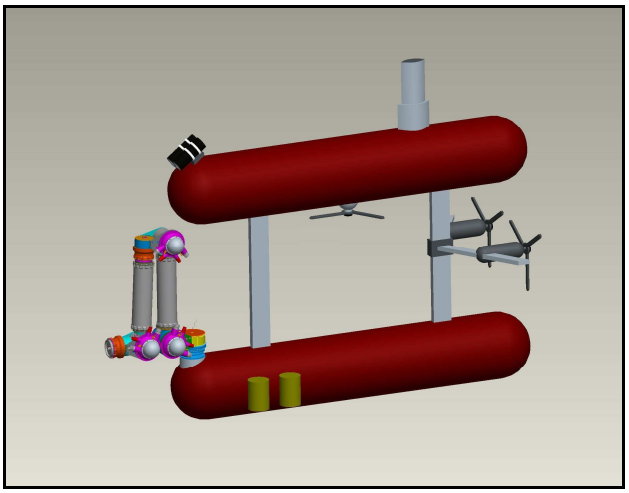


Figure 1.4: CAD model showing SAMURAI manipulator mounted on JAGUAR.

While several top-level trade studies of end-effector concepts had been performed by SSL personnel, resource constraints limited the potential end-effector to a simple scoop. In effect, a trowel would have been mounted to the end of the manipulator, and this would have resulted in significant limitations. Realizing these inadequacies, the SSL elected to develop an end-effector that would provide additional capabilities but not necessitate the complete fabrication of an additional joint.

1.2 Problem Statement

The primary objective of this research is to design and fabricate an end-effector which uses the hand roll degree of freedom of the SAMURAI manipulator for actuation. The hand roll joint is the terminal joint as indicated in Figure 1.5. The end-effector is to be capable of retrieving a sand dollar from the seafloor and depositing it in a sample container. While sand dollars are not necessarily the sampling objective, they are representative of the approximate size of the actual desired targets. The ability to collect seafloor sediment and additional biological specimens such as tubeworms was desired but not required for this particular effort.

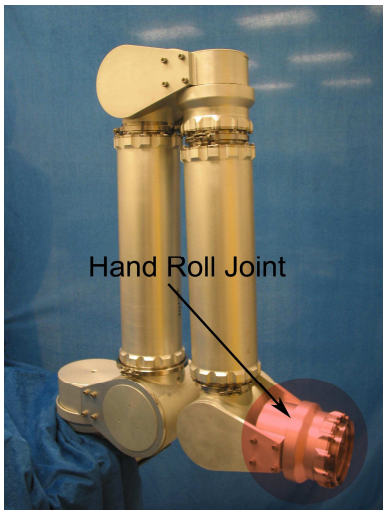


Figure 1.5: SAMURAI with hand roll joint specified.

A subset of objectives includes developing a modular design so that various components can be replaced easily if an alternative mission requires a different design. Moreover, the end-effector should be easy to assemble and disassemble so that it could be exchanged for a different instrument on the surface without risk of complications. Given a limited budget, the fabrication process needed to be straightforward and the materials relatively inexpensive.

The secondary research objectives included design and construction of a sample container. Possible locations for these containers on the JAGUAR vehicle are to be determined, and the effects of eliminating the hand roll DOF from the manipulator kinematics are to be examined.

1.3 Approach

This thesis discusses the design and testing of the end-effector, the sample containers, and the development of software to determine sample container location and quantity. The research began with the design of the end-effector, which constituted the primary research focus. Sample containers were fabricated to incorporate the end-effector geometry. Having established the sample container design, it was possible to create the kinematics software used to determine the number of possible containers and their locations.

1.4 Thesis Structure

Chapter 2 focuses on the function of an end-effector, how the devices are employed on current submersibles, and the tools that have been previously used by the SSL. Chapter 3 overviews the mechanical properties of the SAMURAI manipulator and the interface that the end-effector must accommodate. Chapter 4 discusses the design of the end-effector, focusing on the function of the significant components. Chapter 5 presents the theoretical end-effector performance in addition to results obtained through physical testing. Chapter 6 covers the design and testing of the sample containers, and Chapter 7 overviews the kinematics software employed

to determine sample container location and quantity. Chapter 8 presents conclusions and outlines future tasks pertaining to end-effector development.

Chapter 2
End-Effector Background and Previous Work

In order to design an end-effector, it is important to understand the role of this robotic component. End-effectors are used to accomplish a wide variety of objectives on the surface and can similarly be used to achieve multiple underwater tasks. This chapter focuses on the generic role of these robotic tools and discusses end-effectors that have been employed in underwater missions. Additionally, relevant devices previously developed and currently in use at the SSL are also detailed.

2.1 End-Effector Background

An end-effector is a functional unit associated with the interaction of a robotic system with the environment or with a given object [5]. It is so-named as when it is affixed to a serial manipulator, it is placed on the distal *end* and is the part of the robotic arm that has a direct *effect* on the workspace.

Types of end-effectors vary widely as they may be used for diverse applications. The term “end-effector” can refer to a welding torch, a vacuum pump, grippers or any other tool that is attached to a robotic manipulator. Figure 2.1 shows a robot containing a welding torch end-effector being used to manufacture an automotive exhaust system. Figure 2.2 is a depiction of the Orbital Replacement Unit / Tool Changeout Mechanism (OTCM). The OTCM is a gripper end-effector designed for the Dextre robotic system aboard the International Space Station (ISS).

Figure 2.1: Welding torch end-effector (from [6]).

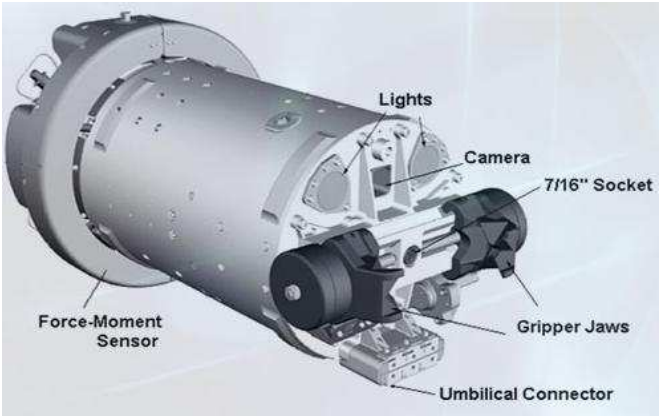


Figure 2.2: Dextre OTCM end-effector (from [7]).

2.2 Existing Underwater End-Effectors

Just as an end-effector is chosen to suit its application on the surface, the same is true of selecting an end-effector for an underwater application. Applications of underwater robots include seafloor mapping, water mine search and disposal, underwater structure inspection and maintenance, and geological sampling [8]. Surveying and/or inspection missions effectively entail mounting cameras to a submersible and do not require robotic manipulators.

However, for tasks pertaining to structure construction or maintenance, manipulators are essential. Figure 2.3 shows a commercially-available ROV manufactured by Oceaneering. The vehicle is designed for activities related to the drilling and production of oil and gas.

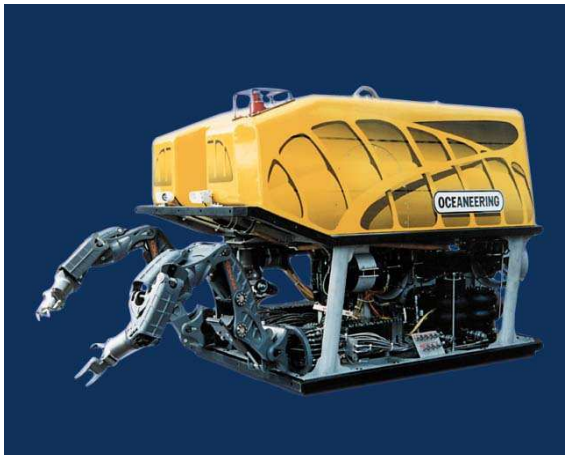


Figure 2.3: Oceaneering Magnum ROV (from [9]).

Figure 2.4: Schilling Orion Manipulator (from [10]).

In Figure 2.3, it is apparent that the ROV contains two manipulators, and Figure 2.4 is an enhanced view of one of these manipulators, which are developed by Schilling Robotics. One manipulator is used to grapple the subsea structure while the other manipulator performs the desired task. The end-effector in Figure 2.4 consists of parallel jaws actuated by a 4-bar linkage.

The continued strong demand for oil juxtaposed with the dwindling shallow-water reserves has been a significant driving factor in the technological progress of underwater robotics [11]; however, end-effectors have also been developed for the purpose of sample collection. Figure 2.5 shows one of the manipulators on the WHOI Alvin submersible reaching for a “black smoker” on the East Pacific Rise. The end-effector is characterized by a gripper design and functions similarly to the Schilling device previously shown in Figure 2.4.

Figure 2.5: WHOI Alvin end-effector reaching for black smoker chimney (from [12]).

The claw design has proven to be one of the most effective underwater sampling tools. In a WHOI survey, marine biologists were asked about their current and future needs regarding deep submergence sampling. The survey produced the following data regarding tool selection:

Table 2.1: Survey Responses to Preferred Manipulator Tools (from [13])

Device	Used Most Often (%)	Importance for Future Research (%)
Vacuum Sampler	74	70
Manipulator Claw	63	61
Sediment Push Core	59	36
Nets and Scoops	56	28
Bioboxes	56	54
In-Situ Sensors	40	57
Faunal Samplers	37	30

The survey demonstrates that not only do the marine biologists rely heavily on the claw, but they also hypothesize that these grippers would be important for their future research. Only the vacuum sampler proved to be more popular, which is

unsurprising given that it represents the best tool for collecting sediment and is comparatively easy to operate. While a gripper may require tremendous accuracy to securely grasp a target, a vacuum sampler does not need the same precision. The disadvantages to a vacuum sampler are a limited sample size and elevated power requirements. This latter disadvantage is of little consequence on an ROV where power is being supplied via an umbilical, but on an AUV, the power is stored in batteries and is therefore more limited.

2.3 Previously Considered End-Effector Concepts for SAMURAI

Just as WHOI surveyed marine biologists to assist in the selection of their manipulator tools, the SSL also contacted oceanic researchers to generate potential concepts for the SAMURAI end-effector [14]. The inquiries were made to a wide range of scientists and engineers and related to the sampling of a large variety of underwater specimens, and the potential targets were not limited to solid objects.

The investigation did include the vacuum sampler that had been popular in the WHOI survey, but the high power draw was noted by the interviewees. Additional disadvantages included potential cross-contamination of samples, the possibility of damaging soft specimens, and the fact that the vacuum could become plugged.

Additional devices considered included the Bushmaster and the Mussel Pot. The Bushmaster, developed at Pennsylvania State University, consists of netting attached to flexible ribbing that opens and closes through the actuation of hydraulic pumps. The Mussel Pot, developed by Pennsylvania State University and the College of William and Mary, functions similarly to the Bushmaster. However, it is different

in that it is a rigid container which is sealed by either pulling a drawstring or shutting an iris. Both of these devices are shown in Figure 2.6.

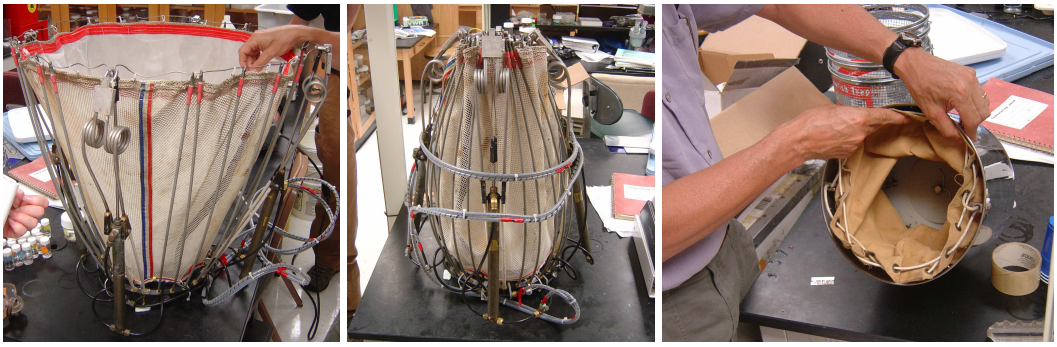


Figure 2.6: Bushmaster with basket open (left), Bushmaster with basket closed (middle), mussel pot with drawstring (right) (from [14]).

These designs are capable of collecting large quantities of diverse samples; however, interfacing with the SAMURAI manipulator would be a serious challenge. Modifying the actuators would be difficult as well, especially in the case of the Bushmaster with its hydraulic pumps.

Concepts highlighted by the interviewed researchers which could more easily interface with SAMURAI include the PacMan Scoop and the previously-detailed claw design. The PacMan scoop, featured in Figure 2.7 and in use at the Canadian Scientific Submersible Facility (CSSF), is an end-effector consisting of two semi-cylinders which open and close like its namesake videogame character.

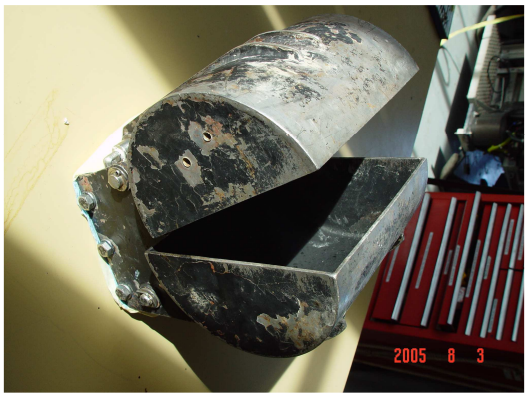


Figure 2.7: CSSF PacMan Sampler (from [14])

There appeared to be a consensus amongst the researchers that the claw was the most utilized device when collecting rocks in hydrothermal vent fields. Additionally, they noted that claws could be used to acquire tube worms by grabbing the worms and subsequently twisting, whereas the Bushmaster or a vacuum sampler would experience difficulty collecting these specimens. Thus, while claw-like grippers may not be able to collect samples on the same scale as these other devices, grippers are nevertheless regarded as very useful general purpose tools in underwater sampling.

2.4 SSL End-Effectors

The SSL has been developing complex robotics for years and has produced many different end-effector designs. Some of them, such as a bolt driver, were designed for space assembly tasks that would not coincide with a sampling end-effector. However, the design concepts behind several SSL tools could be applied to sample collection.

2.4.1 TERPS Planetary Sampler

In 2007, the SSL developed the Tool for EVA or Robotic Planetary Sampling (TERPS), a sampling tool depicted in Figure 2.8. It was designed primarily to assist an astronaut in the collection of rocks or soil on a planetary surface. Torque output by the motor is transferred to the jaws through worm gears.

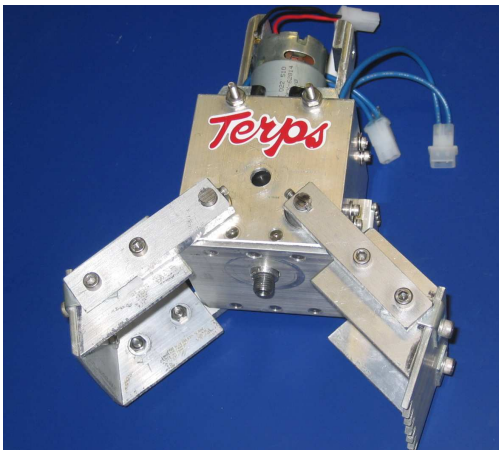


Figure 2.8: TERPS sampling tool (from [15]).

The TERPS sampler may be capable of grabbing a sand dollar, but the jaws would have to be enlarged to adequately fit the object. Additionally, at 100% efficiency, the torque output by the TERPS motor/gearbox is 25.5 N-m, which is approximately 10.5 times less than the 267 N-m output by the SAMURAI hand motor/harmonic drive combination. Either the structural components of the TERPS sampler would have to be enhanced or critical soft stops would be required in the SAMURAI controls to prevent the motor from damaging the end-effector.

2.4.2 Ranger Flight Development Manipulator Parallel Jaw Mechanism

The SSL Ranger project was funded by NASA as part of the Space Telerobotics Program. The objective was to design a robotic system capable of servicing the Hubble Space Telescope, though the developed manipulators can be used for other tasks. One of the end-effectors created that would be most capable of achieving the task of collecting a sand dollar would be the parallel jaw mechanism (PJM), shown in Figure 2.9. In the figure, the curvature in the grippers allows the

end-effector to grasp cylindrical beams, but this geometry could be altered to better coincide with underwater sampling objectives.

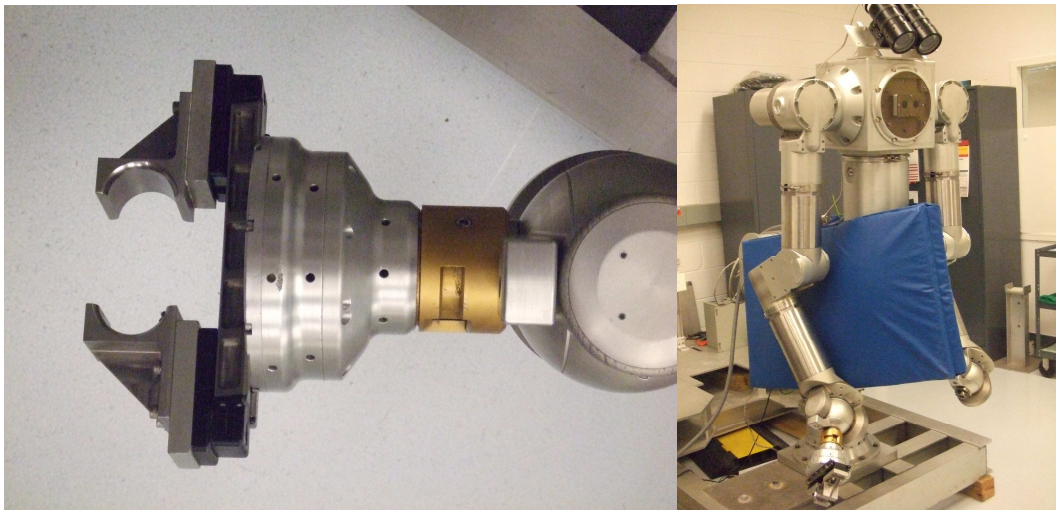


Figure 2.9: PJM end-effector on Ranger manipulator.

The PJM operates by rotating an anodized aluminum spiral plate containing two milled grooves. Rollers are inserted into the grooves, and their motion is confined to one direction by guide rails positioned on either side. As the plate is rotated by a motor, one roller is translated in one direction while the other moves in the opposite direction, producing either an opening or a closing of the jaws. Thus, the spiral plate is acting as a cam disk. These components can be viewed in Figure 2.10, which contains images of the PJM partially disassembled.

Figure 2.10: Ranger Spiral Plate (left) and Associated Track Rollers (right)

While the Ranger PJM is still used on a regular basis, several design iterations were required to reach this operational state. Early iterations occasionally produced binding of the rollers in the grooves, which in turn damaged the rollers and broke the end-effector. All mechanical power is transmitted from the motor to the jaws via the two rollers, which are ½ in stainless steel rollers with a dynamic load capacity of 544 lbs each, according to manufacturer specifications. The structural limitations of the rollers in turn limit the capabilities of the end-effector.

In addition, the guide blocks which move along the rails are held in place by heavily-lubricated Delrin sliders, as shown in Figure 2.11. Delrin would have been selected as the material for these parts as it would create less friction than metal pieces. Even so, plastic sliders are comparatively fragile and do not lead to a robust design.

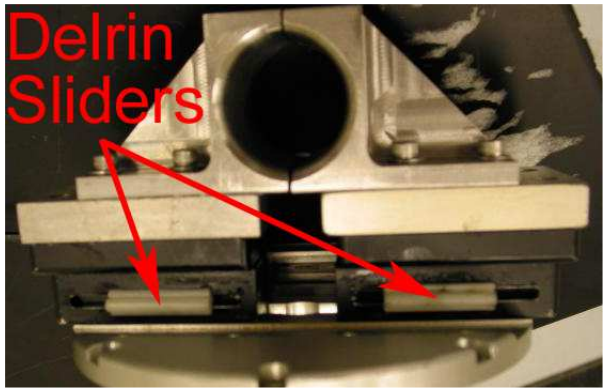


Figure 2.11: PJM Delrin sliders.

There are advantages to the concept, however, and these include a relatively simple design with few moving parts. The PJM is devoid of complicated gear trains, and any operational complications are immediately diagnosable. Moreover, if a

different force or speed profile is desired, the spiral plate can easily be replaced without affecting the rest of the end-effector.

2.4.3 Ranger Neutral Buoyancy Vehicle II Parallel Jaw Mechanism

A similar end-effector was created for use on the Ranger Neutral Buoyancy Vehicle (NBV). This device is also a parallel jaw mechanism that uses a spiral plate to create unidirectional jaw motion. The end-effector is shown in Figure 2.12, which also contains an image of the end-effector in use while attached to Ranger in the SSL Neutral Buoyancy Research Facility (NBRF).

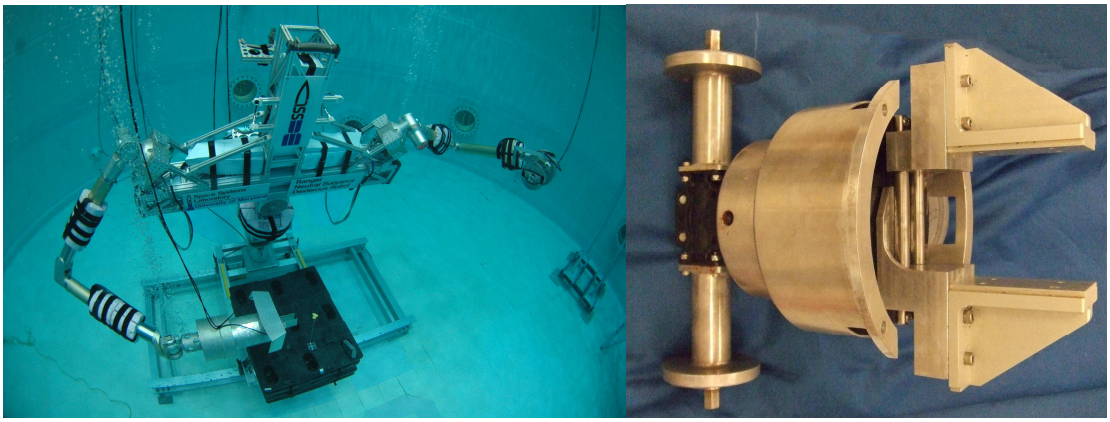


Figure 2.12: PJM (right) and Ranger operating with PJM (left) (from [16]).

In contrast to the PJM fabricated for the Ranger Flight Development Manipulator, this end-effector has flat jaws that can be used for more generic tasks. Additionally, as can be seen in the figure, where the former concept employed Delrin sliders, this end-effector uses aluminum rods to constrain the motion path.

2.5 Summary

A considerable variety of end-effectors are utilized in robotics applications throughout engineering and manufacturing disciplines; however, the number of

concepts decreases markedly when examining the potential tools for underwater tasks. Many submersibles equipped with manipulators use claw-shaped end-effectors to grasp targets. While other sampling systems are in use, they are either not as popular amongst the marine biologists or would present significant challenges in terms of interfacing with SAMURAI and/or JAGUAR.

The SSL has developed end-effectors and sampling systems in the past, but all of these concepts would require significant design modifications in order to attach to the SAMURAI hand roll joint. Moreover, some of these devices necessitated several design iterations in order to reach functional status. These lessons, along with the recommendations of the underwater sampling community, were considered in the selection of the final end-effector concept.

Chapter 3
SAMURAI & End-Effector Interface Overview

One of the most significant challenges associated with designing this end-effector was the SAMURAI interface. This chapter presents relevant details pertaining to the SAMURAI manipulator and more specifically, the hand roll joint.

3.1 SAMURAI Manipulator Mechanical Design Overview

The SAMURAI arm is a 6-DOF dexterous robotic manipulator with motors housed in three joint pairs (shoulder, elbow, and wrist). The joint pairs are labeled in Figure 3.1.

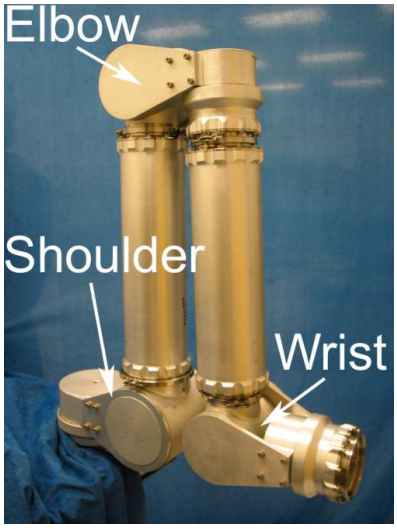


Figure 3.1: SAMURAI manipulator joint pairs.

The shoulder consists of a yaw joint and a pitch joint. Both joints use KollMorgen RBE-03010-B-00 brushless DC motors, which produce a continuous stall torque of 2.21 lb-ft (3.00 N-m) when supplied with 5.9 A of current.

The elbow is comprised of a pitch joint and a roll joint powered by RBE-02111-A-00 brushless DC motors. They generate a continuous stall torque of 1.23 lb-ft (1.67 N-m) at a current of 5.74 A. The wrist uses identical motors to the elbow and also has a pitch-roll configuration. The wrist roll joint is more accurately referred to as a hand roll joint[1], and the final configuration is yaw-pitch-pitch-roll-pitch-roll (Y-P-P-R-P-R). Table 3.1 summarizes the SAMURAI joints. The yaw and roll joints contain hard stops, while the manipulator itself constrains the pitch joint motion.

Table 3.1: SAMURAI joint summary.

Joint Number	Location	Motion
1	Shoulder	Yaw
2	Shoulder	Pitch
3	Elbow	Pitch
4	Elbow	Roll
5	Wrist	Pitch
6	Wrist	Roll

The torque output of all the joints is amplified through harmonic drives. Harmonic drives produce high gear ratios more efficiently and more compactly than alternative systems such as planetary gears. The devices have three main components: a wave generator, a flex spline, and a circular spline. Generic harmonic drive components can be viewed in Figure 3.2. The wave generator is an elliptical disk that rotates inside the cup-like flex spline. The walls of the flex spline are thin, making them flexible. Teeth on the external side of the flex spline fit with teeth in the circular spline, but there are two fewer teeth on the former component. Thus, as the wave generator rotates, it produces a small shift in the flex spine as it slowly moves

[1] Manipulator joints are referred to by their analogous human arm components. Because the human wrist is not capable of producing a rolling motion, Joint 6 is referred to as the "hand roll" joint. This matches the naming convention used previously by the SSL on the Ranger DXM manipulator.

about the circular spline. Flex spline motion is much slower but produces a much higher torque. All of the SAMURAI harmonic drives have gear ratios of 160:1 and increase the output torque substantially.

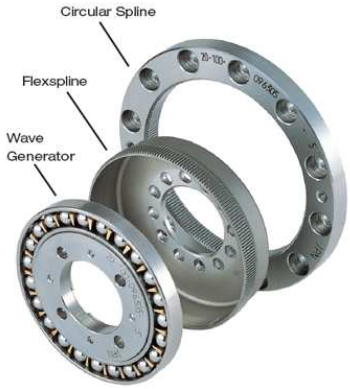


Figure 3.2: Harmonic drive components (from [17]).

Using the harmonic drives in combination with the motors, the manipulator is capable of generating a tool tip force of 32 lb (142 N) in 1-g and 71 lb (316 N) in neutral buoyancy [18]. The difference is attributable to the fact that in neutral buoyancy, the arm does not have to lift its own weight.

The SAMURAI joint pairs are separated by two cylindrical links. The links are attached to the joints via Marman bands. Marman bands, also called "Marman clamps" or "Marman rings," contain V-shaped wedges that pinch two flanges together. Figure 3.3 shows one of the SAMURAI Marman bands.

Figure 3.3: SAMURAI Marman band.

The manipulator joints will be filled with mineral oil to compensate for the considerable pressure 6000 m below sea level. The design calls for only the hemispherical electronics housings to be filled with air. Power, data, and oil are supplied from the JAGUAR AUV to each of the joints via daisy-chained cabling running external to the arm.

The manipulator links and joint housings are mainly fabricated from Aluminum 6061-T6. The flanges, Marman bands, and electronics housings are machined from Titanium 6-4, while the majority of the fasteners on the arm are made from A-286, a high-strength corrosion-resistant superalloy.

These material selections contribute to very high mechanical robustness in the manipulator. In each of the joints, torque is transferred from the motors to the harmonic drives through stainless steel motor keys. Discussions with the lead mechanical designer indicated that the design called for these to be the weakest mechanical components [19]. This corresponds to a minimum SAMURAI mechanical factor of safety of approximately 40. SAMURAI was designed to be dragged behind a ship through an ice field without loss of function, regardless of the number of collisions. The end-effector was created with this in mind.

3.2 Details of the SAMURAI Interface

In terms of end-effector construction, the design needed to take the structural interface of the joint into account. It was critical that the end-effector easily attach to SAMURAI, without significant alterations to the manipulator. Additionally, given that the joint motor constitutes the end-effector actuator, the motor properties needed to be considered as they would affect structural limits and tool performance.

3.2.1 Structural Interface

A close-up view of the hand roll joint is shown in Figure 3.4. As can be observed in the image, there is a titanium cover at the end of the joint. The threaded hole on the front of the plate is required during the oil-filling process. Mineral oil is to fill the joints after passing through hose barbs attached to the penetrator plates which will house all of the electrical connectors. In Figure 3.5, hose barbs inserted into a penetrator plate can be observed. At another location on each of the joint pairs, there is an additional hole. A vacuum pump will be attached to this hole during the filling process to ensure that potentially hazardous air bubbles are removed. The hole will then be plugged. This affects the end-effector only in that there must be sufficient clearance for the plug.

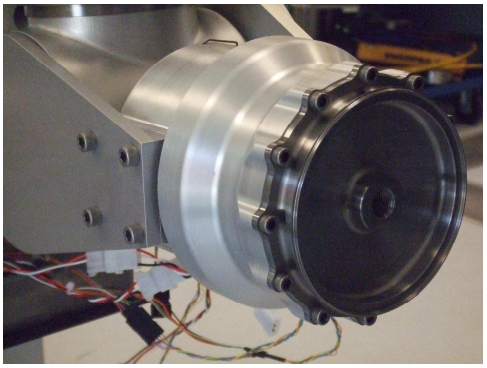


Figure 3.4: SAMURAI hand roll joint.

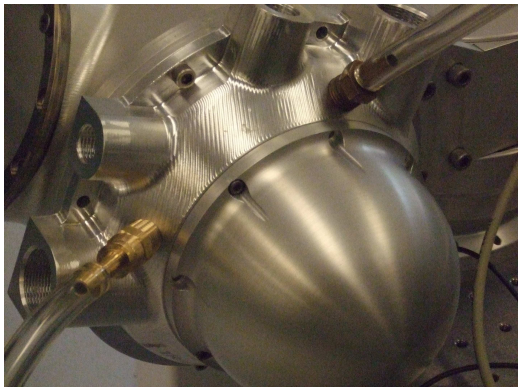


Figure 3.5: Oil hose barbs inserted in penetrator plate.

The outer rim of this titanium plate has a male flange identical to those used to connect the links to the joints. Its corresponding female counterpart had been previously engineered and was to be incorporated into the end-effector should such an attachment plan be selected. Inclusion of this part adds to the modularity of the manipulator and reduces the need for additional design work. A dimensioned drawing for the female flange and all additional end-effector components are featured in Appendix A.

Many of the end-effector concepts investigated would require that some elements be held stationary relative to the joint housing. While a vacuum sampler or a push core could mount directly to the end of the roll joint, almost all end-effector designs involving moving jaws would require some set fixtures. Certain gearing connections, guide rails for a PJM, a track for a rack and pinion, and other such items would need to be fixed to the joint. Drilling additional holes into the housing was highly undesirable for reasons pertaining to structural integrity and sealing. Attempts to use current holes would be made before removing additional material from the joint housing. Two fasteners have been removed in Figure 3.6, showing the existing holes.

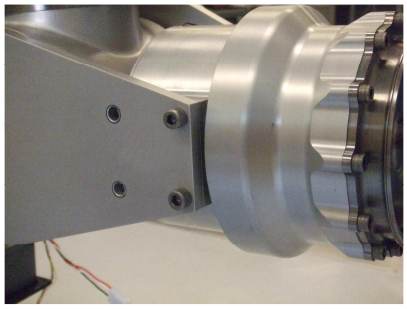


Figure 3.6: Existing holes for fasteners in the joint housing.

3.2.2 Hand Roll Joint Motor Properties

As previously stated, the motor internal to this joint is a KollMorgen RBE-02111-A-00 DC brushless motor (complete specifications in Appendix A). The motor is capable of operating at continuous stall torque at a current of 5.71 A; however, the motor driver electronics boards limit the continuous current delivered to the motors to 5 A. While the motor driver boards allow for a peak current of 10 A, elimination of the heat generated by the electronics is a serious issue, and considerable efforts will be made to avoid any current spikes. As such, 5 A is taken to be the maximum motor current. The supplied voltage is 28 V, rendering the maximum electrical motor power 140 W.

Given a maximum current, I_{max}, and obtaining the torque constant, K_T, from the specifications sheet, (3-1) can be used to compute the maximum expected motor torque, T_{max}.

$$T_{\max} = I_{\max} \cdot K_T \tag{3-1}$$

To determine the torque output by the joint, T_{roll}, the maximum motor torque is multiplied by N, the harmonic drive gear ratio, and η_{HD}, the harmonic drive efficiency. This relation is shown in (3-2).

$$T_{roll} = \eta_{HD} \cdot N \cdot T_{\max} \tag{3-2}$$

The numeric values for these variables are summarized in Table 3.2.

Table 3.2: Values relating to hand roll joint output torque.

Variable	Description	Value (English)	Value (SI)
T_{sens}	Torque Sensitivity	0.225 lb-ft/A	0.305 N-m/A
I_{max}	Maximum Current	5.0 A	5.0 A
T_{max}	Maximum Motor Torque	1.13 ft-lb	1.53 N-m
N	Gear Ratio	160	160
η_{HD}	Efficiency	0.80	0.80
T_{roll}	Output Joint Torque	144 ft-lb	195 N-m

The motor will generate a back electromotive force (EMF) that will cancel voltage at a rate of 31.9 V/krpm. While the motor specification sheet lists the speed at rated power as 4242 rpm, the actual maximum motor speed is approximately 880 rpm due to the supply voltage of 28 V. In regard to the initial end-effector performance predictions, it is assumed that the tool is operating with maximum joint torque. This presumes no structural limitations and provides the absolute end-effector performance limit.

3.3 Summary

The SAMURAI joints contain brushless DC motors and harmonic drives, providing significant torques throughout the arm. Material selections and various components are incorporated in the design to produce a mechanically robust manipulator.

Any modifications to the SAMURAI structure were to be avoided; however, there are multiple features that can be utilized for end-effector attachment. The male flange positioned on the end of the joint and fastener holes already located in the housing represent possible interfacing points. Motor performance was evaluated, and the maximum theoretical torque expected to be delivered to the end-effector is 144 ft-lb (195 N-m).

Chapter 4
End-Effector Design

In this chapter, candidate end-effector designs are compared, and the most capable device is determined. The manner in which torque will be transferred from the joint through the end-effector is discussed, and the rationales for significant end-effector components are overviewed.

4.1 Concept Selection

Although the vacuum sampler had been viewed the most favorably in the WHOI sampling tool survey, there would be serious complications in using it on the SAMURAI/JAGUAR system. The significant power draw would be a considerable obstacle for any AUV, but for a mission at 6000 m beneath the surface with a 36-hour duration, this is especially problematic. Moreover, a sand dollar would be too large and too brittle for many vacuum sampling systems currently in use. For these reasons, it was discounted as a possibility.

A push-core end-effector was seriously considered as it would basically entail mounting a cylinder on the end of the hand roll joint and actuating a lid with a spring mechanism. If the mission had entailed a single sample, this would have worked well as the end-effector would have doubled as the container. However, as multiple targets were desired, the push core would either need to open and release the sample into a container or SAMURAI would need to exchange end-effectors during the mission. The complexity associated with either of these designs was undesirable, and the push core was not selected.

Net-based concepts, such as the Bushmaster, were also considered; however, as in the case of the push-core, it would be difficult to collect multiple samples. While the netting could be opened and closed to possibly obtain several samples, cross-contamination would then be an issue. Furthermore, the large surface area of such a device would greatly increase drag as the AUV travels through the water. In addition, interfacing with SAMURAI would be extremely challenging, and this was not amongst the most popular devices in the research surveys.

A gripper end-effector was selected due to its successful implementation on previous undersea sampling missions and popularity amongst marine biologists. A design with actual claws composing the tool tip was considered, but jaws were ultimately used for reasons that will be detailed in Section 4.4.5. Although the interface design between the gripper end-effector and the manipulator was nontrivial, it was less complicated than some of the netted concepts. It would allow for acquisition of a sand dollar and could collect multiple samples and deposit them in separate containers.

A summary of the concept capabilities is included in Table 4.1. Only the gripper mechanism is capable of achieving all of the objectives.

Table 4.1: End-effector concept capabilities.

Capability	Vacuum Sampler	Push Core	Actuated Nets	Grippers
Sand Dollar Collection	Yes	Yes	Yes	Yes
Multiple Sample Collection	FER	No	FER	Yes
Cross-Contamination Avoidance	FER	Yes	No	Yes
Low Power Draw	No	Yes	Yes	Yes
Low Drag	Yes	Yes	No	Yes

Yes – Highly likely or certain to demonstrate the capability.
FER – Further Evaluation Required. Capability may or may not be possible depending on design configuration.
No – Highly unlikely if not impossible to demonstrate the capability.

4.2 SAMURAI Attachment

For the gripper end-effector to have moving jaws, it would be necessary to create stationary points relative to the rotating hand roll joint. An object devoid of these fixtures would simply spin along with the wrist. As had been shown in Chapter 3, there are fastener holes in the housing that connect the final joint to the manipulator. Stationary points are produced by using longer fasteners and attaching mounting braces to the joint housings using these same holes. These side mounts can be visualized in Figure 4.1.

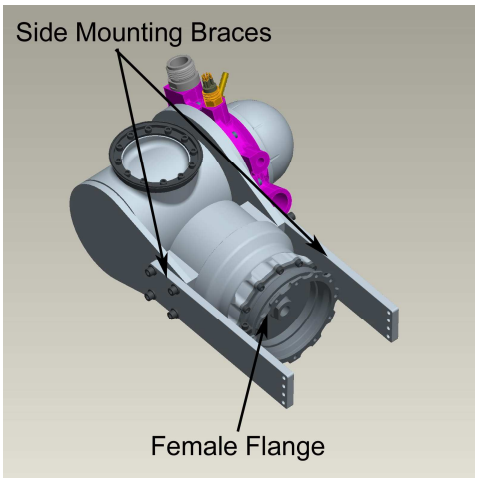


Figure 4.1: CAD model of wrist joint pair showing flange and braces.

In Chapter 3 it was also mentioned that the titanium plate on the front of the hand roll joint contains a male flange. This flange was created on the plate specifically for the attachment of whatever tool would be placed on the end of the arm. Correspondingly, the end-effector contains the female mate, and the two flanges are joined with a Marman band. This can also be observed in Figure 4.1.

4.3 Torque Transmission Method

Having established fixed mounts and an attachment to the rotating joint, it was necessary to determine how to transmit torque from the joint to the jaws. Some of the grippers used for underwater sampling employ 4-bar linkages activated by linear actuators. Other concepts use actuation schemes more analogous to the hand roll joint. These include a worm-gear driven linkage, such as the one pictured in Figure 4.2.

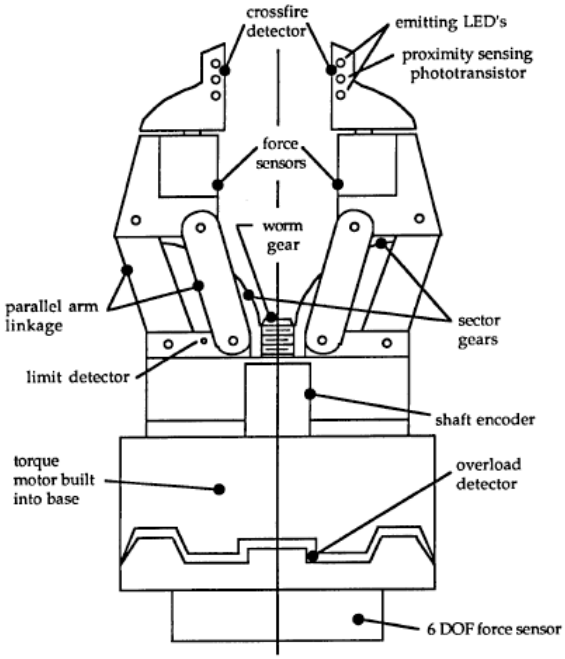


Figure 4.2: Parallel jaw gripper concept (from [21]).

Another potential worm gear concept for the SAMURAI end-effector was inspired by the TERPS sampler as the orientation of its motor and jaws is identical to the orientation in the SAMURAI joint. This early concept is displayed in Figure 4.3.

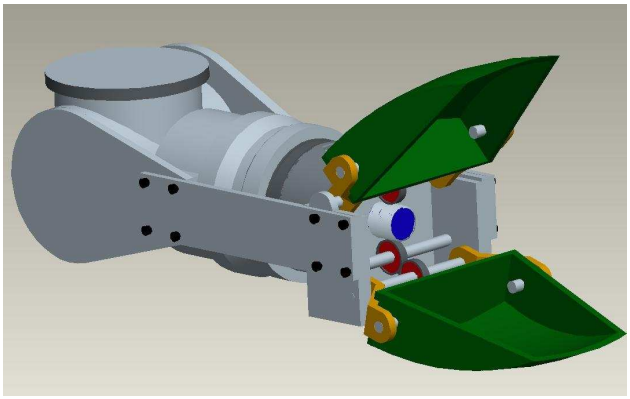


Figure 4.3: End-effector worm gear concept.

In this configuration, an adapter connects the female flange to a worm gear, colored blue in the figure. Rotation of the worm gear induces motion in spur gears mounted on shafts, which are supported by the mounting braces. Links connect the

shafts to the jaws. A 4-bar linkage could be used to augment the jaw mechanical advantage.

Other possibilities involved bevel gears or a rack and pinion, but all of these concepts use some form of gearing, which is a drawback. Gear trains can result in substantial efficiency losses as evidenced by the TERPS sampler, which demonstrated a mechanical efficiency of 15% [15]. Additionally, the relatively high torques being output by the harmonic drive could cause gear teeth to skip or break. If either of these events transpired, the end-effector would likely become inoperable. On the surface, the broken gear can be replaced quickly and the device returned to an operational state; however, over the course of 36 hours underwater, a single tooth could compromise the entire mission.

Based on these concerns, the cam disk concept previously used by the SSL in the Ranger PJM was selected. The spiral plate has been a successful development, though several design iterations were required to reach this state[2]. It offers the additional advantage of having no complicated gearing. The primary disadvantage is that all torque is transferred through two relatively small track rollers. However, difficulties could be avoided by designing an appropriate cam path and selecting a sufficiently strong roller.

4.4 End-Effector Component Design

The end-effector was composed of several significant components. These elements will be discussed individually. The respective functions will be presented in the order that the components attach to the end-effector, beginning with the flange

[2] Note on terminology. The terms “spiral plate” and “cam disk” are used interchangeably. The component of interest is both a plate and a disk that contains spiral grooves, which function as cams.

connector and concluding with the jaws. Loading and safety factors are presented in Chapter 5.

Ninety-six fasteners bind the end-effector components together, and they are all 10-32 socket head cap screws (SHCS), though lengths vary. The same socket head was selected to simplify assembly and disassembly procedures.

Unless otherwise stated, Aluminum 6061-T6 was selected as the material for all end-effector components due to its relatively high strength and low mass. Moreover, this is the material comprising the majority of the manipulator. Parts are made from commercially available stock (1/4 in, 3/8 in, and ½ in thickness) wherever possible. These thicknesses correspond to 0.635 cm, .9525 cm, and 1.27 cm in metric units. The total prototype mass 9.09 lb_m (4.12 kg).

As previously stated, dimensioned drawings for all end-effector components are contained in Appendix A. In addition, spreadsheets listing all end-effector parts and associated quantities, materials, volumes, costs, and masses are included in Appendix A.

4.4.1 Flange-Cam Disk Adapter

An adapter is used to connect the flange to the cam disk. The component is indicated in Figure 4.4. As the adapter must connect to the flange in the same manner as the SAMURAI link tubes join to their respective flanges, the same insert pattern was used in the outermost ring. An inner ring of 12 holes allows for attachment of the cam disk. A view of a link-flange interface is featured in Figure 4.5.

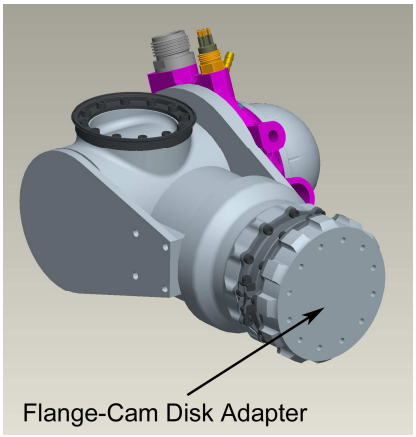


Figure 4.4: CAD model of wrist joint pair featuring adapter.

Figure 4.5: Manipulator link-flange interface.

The objective of this research focused on the development of a prototype, and modularity was incorporated into the design where it was possible. The adapter is included to allow for easier change-out of different cam disks, which may be machined to produce different profiles or be composed of different materials. If a solitary cam disk is to be used, the adapter could be eliminated and the flange could mate directly to the disk, reducing length and mass.

4.4.2 Spiral Plate

The spiral plate is critical to the transmission of torque from the roll joint to the parallel jaws. It attaches to the flange adapter with 12 fasteners on the lower surface. The upper surface contains two grooves, one for each of the track rollers which drive each of the respective jaws. The spiral plate can be seen in Figure 4.6.

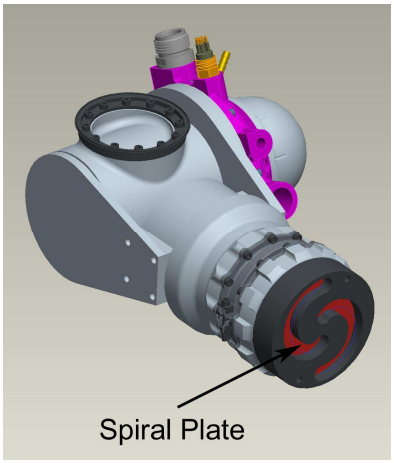


Figure 4.6: CAD model featuring cam disk.

Close inspection of the figure reveals two circular holes on the upper surface. These were included at the request of the machinist so that the part could easily be mounted in the mill.

Although cams are generally made from steel, iron, or other hard metals, the prototype spiral plate is to be fabricated from aluminum. The Ranger plates are made from aluminum, although those have been anodized to increase wear resistance. Should long-term testing indicate that increased hardness is necessary, the disk may be anodized or machined from stainless steel. Anodizing the plate would also alleviate concerns of sand entering the disk grooves and the small grains damaging the tracks.

When initially selecting profiles for the spiral plate grooves, the design sought a general pathway described in the *Cam Design Handbook* characterized by a high opening acceleration and closing deceleration [22]. The associated generic equations are contained in Figure 4.6 where y represents the vertical displacement of the cam follower, h is the total displacement, β is the cam angle associated with the displacement, θ is the angle of rotation, and y' and y'' are the velocity and

acceleration, respectively. This concept was selected so that high accelerations would negate any stiction effects and ensure that the jaws open and close smoothly and invariably.

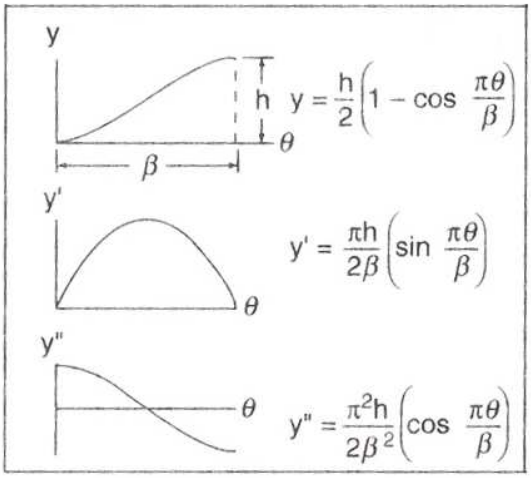


Figure 4.7: Selected generic cam profile for spiral plate (from [22]).

Such a profile will produce high opening and closing forces. While this may seem like a design attribute as the capability to break off a portion of a targeted specimen would be desirable, it is very likely that the jaws will inevitably close upon an extremely rigid object. In an instance where an object is to be fractured, it is critical that excessive force is not transmitted through the cam followers[3], two relatively weak end-effector components. As each jaw is associated with only one cam follower, any damage to this follower will render the end-effector inoperable.

Consequently, the profile was adjusted so that the force capability would continually decrease as the jaws closed. As the jaws proceed to shut, the likelihood that they will close upon an object simultaneously increases. By decreasing the force at which the motor will stall, the cam followers are increasingly protected.

[3] Note on terminology. The terms "cam follower" and "track roller" are used interchangeably. The components of interest are rollers designed to travel along a track. As that track is being generated by the cam disk, the track rollers are simultaneously cam followers.

In order to model the curves using CAD software and to achieve the desired forces, the position profile displayed in Figure 4.7 was modified to make it dependent solely on β and h, as shown in Equation 4-1. β and h are constants and are set to 155° (2.71 rad) and 2.25 in (5.715 cm), respectively.

$$r = \left(\frac{h}{2}\right) \cdot \left[1 + \cos\left(\beta - \frac{\pi}{2}\right)\right] \tag{4-1}$$

Whereas the generic position profile is a function of θ directly, the CAD package generates variables by defining t as a unitless parameter varying from 0 to 1. This requires that the rotation variables be made into functions of t, and this is done in Equation 4-2.

$$\theta = f(t) = t \cdot \beta \tag{4-2}$$

To model the grooves using Cartesian coordinates, r and θ are converted to x and y using Equations 4-3 and 4-4.

$$x = r \cdot \cos\left(\theta - \frac{\pi}{2}\right) \tag{4-3}$$

$$y = r \cdot \sin\left(\theta - \frac{\pi}{2}\right) + offset \tag{4-4}$$

The term *offset* in Equation 4-4 creates space between the groove and the center of the spiral plate. This offset ensures separation between the grooves, and the value is set to 0.45 in (1.143 cm). To obtain the equations for the second groove, Equations 4-3 and 4-4 are simply multiplied by -1.

The theoretical end-effector performance using these profiles will be discussed in Chapter 5.

4.4.3 Track Rollers

Stainless steel track rollers follow the motion path created by the spiral plate grooves. Their motion is constrained to one dimension by guide rails which will be discussed in Section 4.4.5. The track roller used in the end-effector is shown in Figure 4.8.

Figure 4.8: Stainless track roller (from [23]).

To avoid the complications experienced in early iterations of the Ranger PJM, a larger, stronger roller was chosen. Selected properties of the Ranger and SAMURAI rollers are compared in Table 4.2.

Table 4.2: Selected Ranger and SAMURAI roller properties.

Detail	Ranger	SAMURAI
Manufacturer	Carter Bearings	McGill
Outer Diameter	½ in (1.27 cm)	5/8 in (1.59 cm)
Dynamic Load Capacity	544 lb (2420 N)	955 lb (4250 N)

The track rollers are oriented in the spiral plate grooves as shown in Figure 4.9.

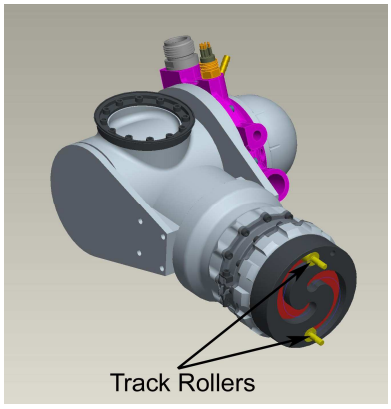


Figure 4.9: CAD model indicating track rollers.

4.4.4 Guide Blocks and Rails

Guide rails serve as structural fixtures to constrain track roller motion. The track rollers screw into the rear surface of guide blocks which travel along the rails. The guide blocks and rails are shown in Figure 4.10.

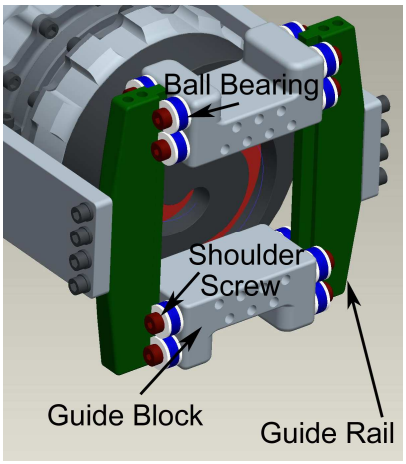


Figure 4.10: CAD model highlighting guide rails and guide blocks.

Each guide block incorporates eight stainless steel ball bearings to provide smooth motion along the rails. In Figure 4.10, these bearings are colored blue. Each of the bearings is characterized by a 332-lb (1477-N) dynamic load capacity and connects to the guide block by spinning on the shaft of a 1/4-in (0.635-cm) diameter shoulder screw. Initial designs used one row of four bearings, but a second row was added to increase stability and reduce binding concerns.

To ensure that the ball bearings are not pinched between the shoulder screw head and the guide block, 1/16-in (0.159-cm) thick Delrin thrust bearings are positioned on each side of the ball bearings. In the CAD model, the thrust bearings are shown in white. An exploded view of the shoulder screw assembly is shown in Figure 4.11.

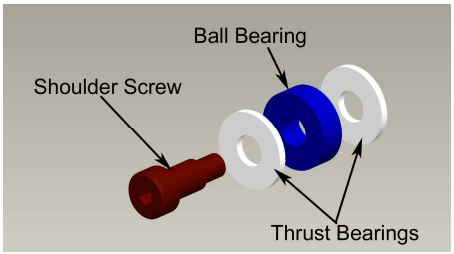


Figure 4.11: Exploded CAD model of shoulder screw assembly.

In Figure 4.10, there are seven holes on each of the guide block faces. These represent the holes for the jaw attachment fasteners.

4.4.5 Jaw Design

Scoop like jaws were selected over the true claws and other geometries employed on some submersibles in an effort to combine the capabilities of claws with those of a scoop. While the primary objective was sand dollar collection, the ability to acquire tube worms and sediment was also desirable.

In addition, consideration was given to how the end-effector would transfer the sample from the seafloor to the containers. To locate samples, the SAMURAI system uses two stereo cameras rated to 6000 m positioned on the upper cylinder of the AUV. The Autonomous Vision Application for Target Acquisition and Ranging (AVATAR) uses its pair of high-resolution cameras located in housings and image processing software to identify desired targets and determine their respective locations. However, the field of view (FOV) is limited, as shown in Figure 4.12. As the manipulator approaches the sample containers, it moves out of the AVATAR FOV. If a sample fell during the transfer, it would go unrecognized until the JAGUAR returned to the surface. Thus, a method to securely grasp the sample until releasing it into a container would be beneficial.

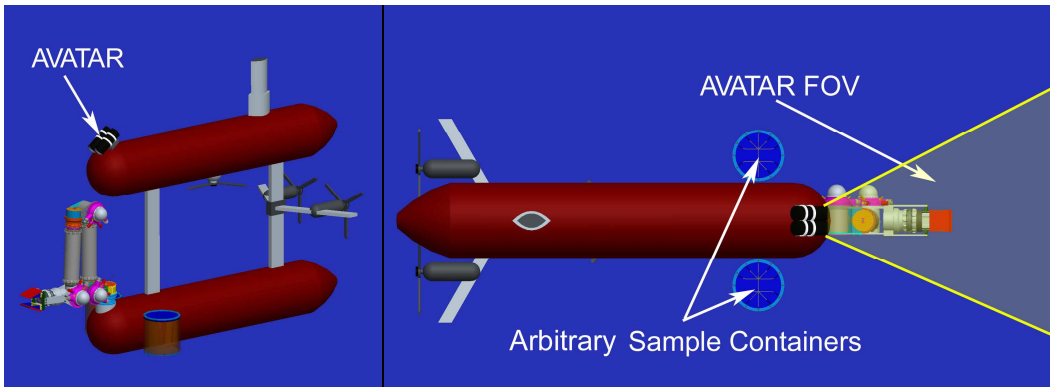


Figure 4.12: CAD model of JAGUAR including AVATAR (left) and AVATAR FOV (right).

Capabilities of the various grippers are compared in Table 4.3, which illustrates that the jaws are the superior option for this application. A trowel would be unable to break off tube worms while claws could not carry sediment. Samples could be lost in both of these devices during transfer to the containers. The PacMan scoop is capable of grasping all the desired samples; however, the sample containers would need to be excessively large to accommodate the device when it opens. It was not selected for this reason.

Table 4.3: Gripper capability comparison.

Capability	Jaws	Trowel Scoop	Claws	PacMan Scoop
Sand Dollar Collection	Yes	Yes	Yes	Yes
Tube Worm Collection	Yes	No	Yes	Yes
Sediment Collection	Yes	Yes	No	Yes
Secure Transfer to Container	Yes	No	No	Yes
Simple Container Insertion	Yes	Yes	Yes	No

Yes – Highly likely or certain to demonstrate the capability.
No – Highly unlikely if not impossible to demonstrate the capability.

Jaw dimensions were determined based on sand dollar sizes. George and Boone (2003) examined sand dollar populations in the state of Georgia from 1998 to 2002 and found their diameters to range from 50 mm to 110 mm with a mode of

60 mm (2.36 in) to 70 mm (2.75 in) [24]. Using this mode, the average sand dollar diameter was estimated to be 63.5 mm (2.5 in), and the jaw width and length were designed to accommodate sand dollars of this size and those of up to 25.4 mm (1 in) larger diameters. Thus, the jaw width and length were set to 88.9 mm (3.5 in).

Triangular geometry was used to ease machining, and the wedge height was set at 31.75 mm (1.25 in). The basic dimensions can be seen in the model in Figure 4.13. The individual jaw internal volume is 126 cm^3 (7.66 in^3), rendering a total internal volume of 251 cm^3 (15.3 in^3).

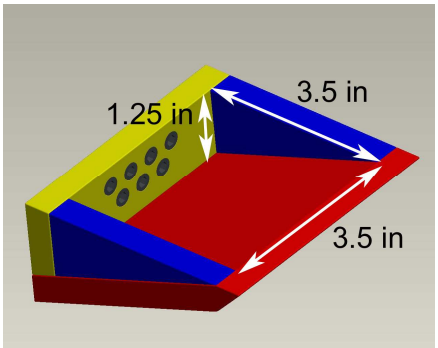


Figure 4.13: CAD model of jaw showing key dimensions.

The jaws are attached to the rest of the end-effector in the manner shown in Figure 4.14.

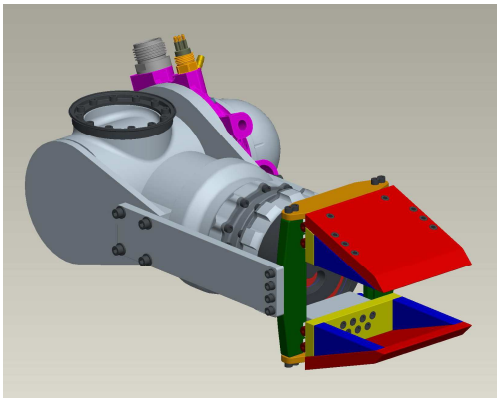


Figure 4.14: CAD model of wrist joint pair and complete end-dffector.

Fasteners used in the jaw are countersunk to avoid any entanglements with the samples. The jaw is an assembly of several pieces to allow for access to the guide block attachment fasteners. Additionally, the assembly allows for easier exchange of components. The prototype top plate, which has a smooth front edge, could be exchanged for one with a serrated edge or a blade depending on mission requirements.

Figure 4.15 shows the actual end-effector attached to the SAMURAI hand roll joint.

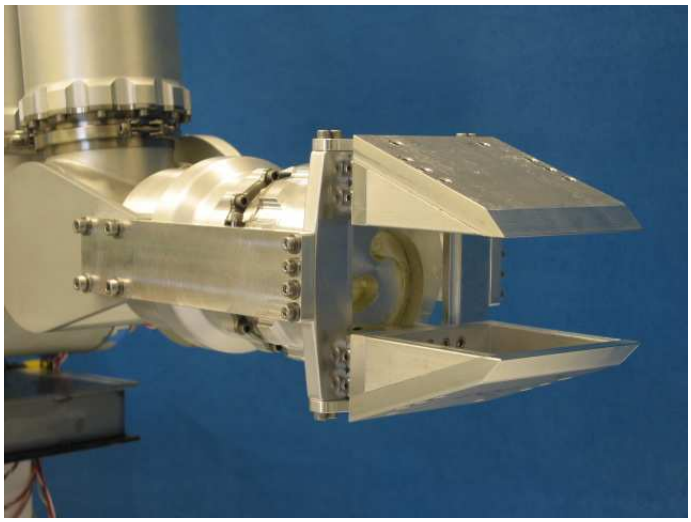


Figure 4.15: Fabricated end-effector.

4.5 Summary

The gripper concept was selected for the end-effector, and jaws are utilized as the grippers themselves. These selections constitute the geometry most capable of achieving mission objectives. It was determined that the end-effector would attach to the hand roll joint with both a flange and side mounting braces attached through existing fastener holes.

A spiral plate was selected for the transmission after deeming the complexity and risk associated with gearing systems to be undesirable. General overviews of the design rationale for the spiral plate and other significant end-effector components were presented. The performance of these components is evaluated in Chapter 5.

Chapter 5

End-Effector Performance: Theory and Test Results

This chapter discusses the theoretical performance of the end-effector, focusing in particular on the jaw opening distance and closing force. Theoretical calculations are used to establish anticipated end-effector performance, and structural analyses are completed to determine the force range in which the end-effector can safely operate. Closing force was measured for various current settings, and the physical results are compared to predicted values.

5.1 Theoretical Jaw Performance

Both the jaw opening distance and the closing force, two significant parameters, are dictated by the groove profiles in the spiral plate. Profile generation was discussed in Chapter 4, and depictions of the grooves created using CAD software and MATLAB are juxtaposed in Figure 5.1. The equations generating the MATLAB curves are identical in form to (4.1 – 4.4). The software used to generate this and all subsequent cam disk plots is featured in Appendix B.

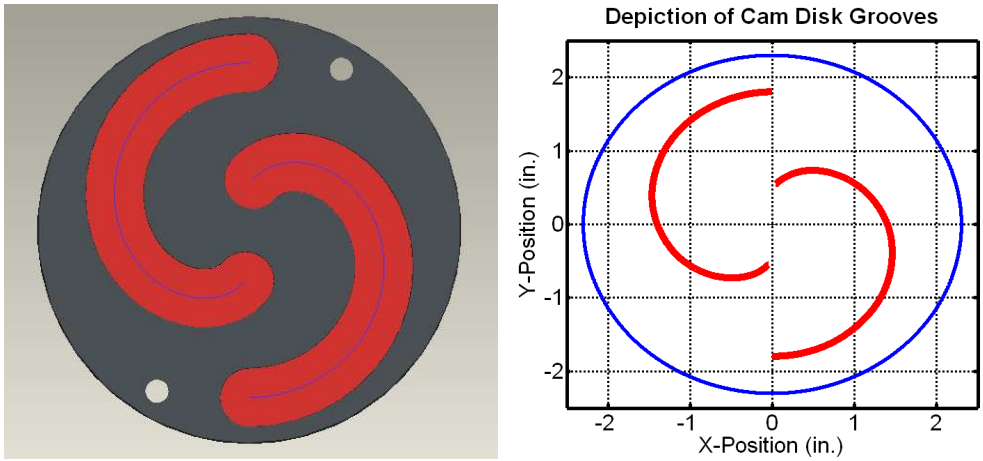


Figure 5.1: Cam disk representations in CAD (left) and MATLAB (right).

These groove profiles constitute the basis for the subsequent analysis.

5.1.1 Jaw Opening Distance

In order to calculate the jaw opening distance, it is necessary to determine the locations of the track rollers in the cam disk grooves. To accomplish this, the coordinate frame is rotated from the cam disk frame (x_1, y_1) to the global frame (x_2, y_2), where y_2 represents the cam follower location. These frames can be seen in Figure 5.2.

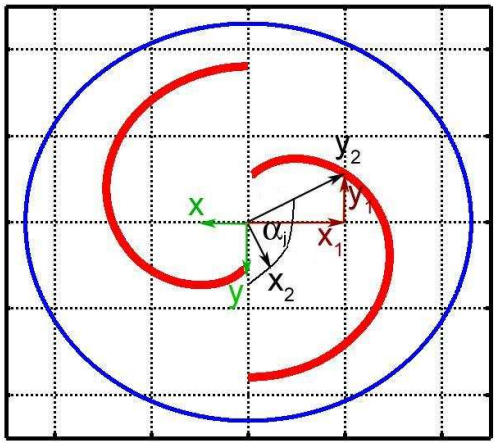


Figure 5.2: Cam disk groove profiles with superimposed coordinate frames.

In the image, α_j is the angle of joint rotation[4], which is not identical to the angle θ used to generate the groove profiles. The dissimilarity is attributable to the 0.45 in (1.14 cm) offset from the plate center. As a result of the offset, θ varies from 0° to 155° whereas $\alpha_{j,max}$ extends to 175.3°. Actual joint rotation is limited to 173.6° to ensure that the jaws contact before the track followers reach the end of the grooves.

To determine track roller location y_2, a rotation matrix is applied as shown in (5-1). The values of x_1 and y_1 were determined when the groove profile was

[4] Subscript "j" refers to the joint and is included to distinguish the α referring to joint rotation to the α used in Chapter 7 in the kinematics analysis.

converted to Cartesian coordinates using Equations 4-3 and 4-4. The π terms are included to properly orient the base frame (*x*, *y*), shown in green in Figure 5.2.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\alpha_j - \pi) & \sin(\alpha_j - \pi) \\ -\sin(\alpha_j - \pi) & \cos(\alpha_j - \pi) \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{5-1}$$

Equation 5-2 is employed to solve for the follower displacement, y_2.

$$y_2 = -x1 \cdot \sin(\alpha_j - \pi) + y_1 \cdot \cos(\alpha_j - \pi) \tag{5-2}$$

Follower displacement is plotted against the joint angle of rotation, α_j, in Figure 5.3.

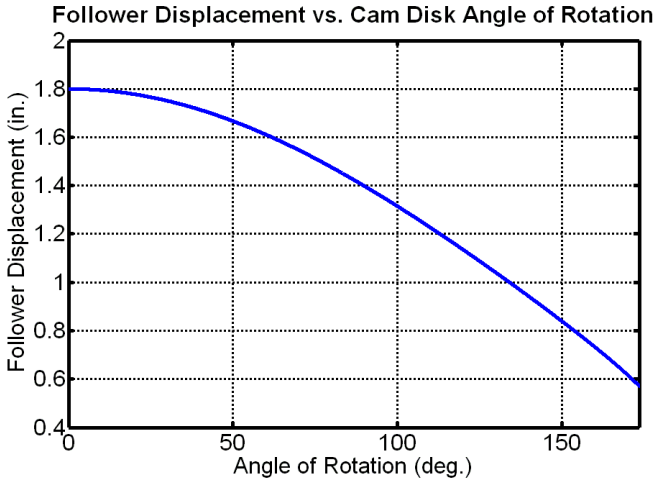


Figure 5.3: Plot of follower displacement vs. rotation angle.

Assuming the center of the cam disk represents the origin of the coordinate frame, Figure 5.3 shows that when the jaws are completely open, the upper jaw track roller is at a position of y_{open} = 1.80 in (4.57 cm). After a smooth descent, the jaws shut completely when the roller reaches y_{closed} = 0.572 in (1.453 cm). The total displacement (Δy_{tot}) is calculated by taking the difference using Equation 5-3.

$$\Delta y_{tot} = y_{open} - y_{closed} \tag{5-3}$$

The displacement of one jaw is 1.23 in (3.12 cm), resulting in a maximum opening distance of 2.46 in (6.25 cm) for both jaws.

If larger samples are desired, this opening distance could be increased with a larger cam disk with elongated grooves. In this design, the cam disk diameter was set equal to that of the SAMURAI links and yields an opening sufficient for collection of sand dollars, tube worms, and sediment.

5.1.2 Jaw Force Analysis

Jaw force is determined by equating the work done by the cam disk to the work done on the follower:

$$T_{roll} \cdot \Delta\alpha_j \cdot \eta_{roller} = F_{out} \cdot \Delta y \tag{5-4}$$

where T_{roll} is the torque input to the cam disk, $\Delta\alpha_j$ is the change in angle of rotation, Δy is the vertical displacement of a cam follower, and F_{out} is the total force being output by both jaws combined. The efficiency of the track rollers is represented by η_{roller}.

Solving for F_{out} in (5-4) gives:

$$F_{out} = \frac{T_{roll} \cdot \Delta\alpha_j \cdot \eta_{roller}}{\Delta y} \tag{5-5}$$

where T_{roll} is the constant torque associated with maximum current and was previously determined to be 144 lb-ft (195 N-m), while η_{roller} is estimated to be 0.90 based on established values [25]. This leaves the output force as a function of Δy and $\Delta\alpha_j$, but Δy can be set equal to the position of the cam follower. Since this was made to be a function of $\Delta\alpha_j$ in (5-2), F_{out} is a function of the angle of joint rotation exclusively. The force acting on the track rollers is plotted against angle of rotation in Figure 5.4. When the jaws close, the force output to the track rollers is 892 lb (3968 N).

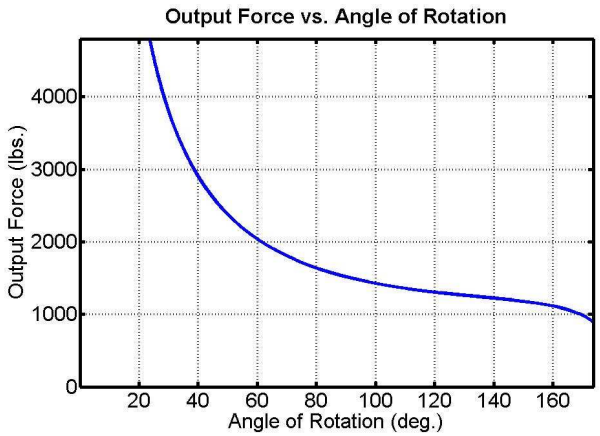


Figure 5.4: Plot of track roller force vs. rotation angle.

This force acting on the track rollers is not identical to the jaw closing force. The track roller force is a vector acting normal to the groove surface, whereas the closing force is represented by its vertical component due the vertical motion constraints imposed on the rollers. The pressure angle (γ) is defined as the angle between this normal vector and the instantaneous direction of motion of the cam follower. Figure 5.5 illustrates the relationship between these terms.

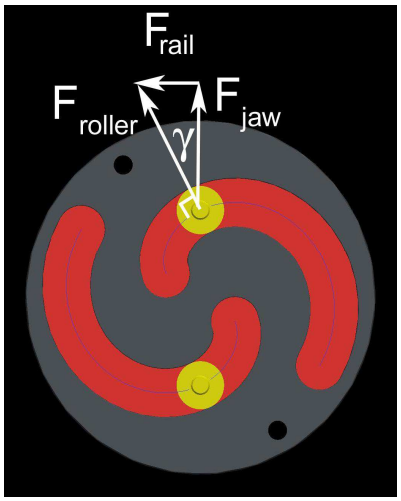


Figure 5.5: CAD model of cam disk including pressure angle parameters.

To calculate the pressure angle from known quantities, the MATLAB software takes two adjacent points on the cam disk curves and determines the slope of a line joining them. This basic relation is shown in (5-6).

$$m = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \tag{5-6}$$

By calculating the negative reciprocal of this line, the slope of the perpendicular line is determined:

$$m_{perp} = -\frac{1}{m} \tag{5-7}$$

The pressure angle is produced by calculating the arctangent of this slope, as shown in (5-8). The angle of rotation must be subtracted from the pressure angle to account for the rotation of the disk.

$$\gamma = a\tan\left(m_{perp}\right) - \alpha_j \tag{5-8}$$

Pressure angle is plotted as a function of rotation angle in Figure 5.6.

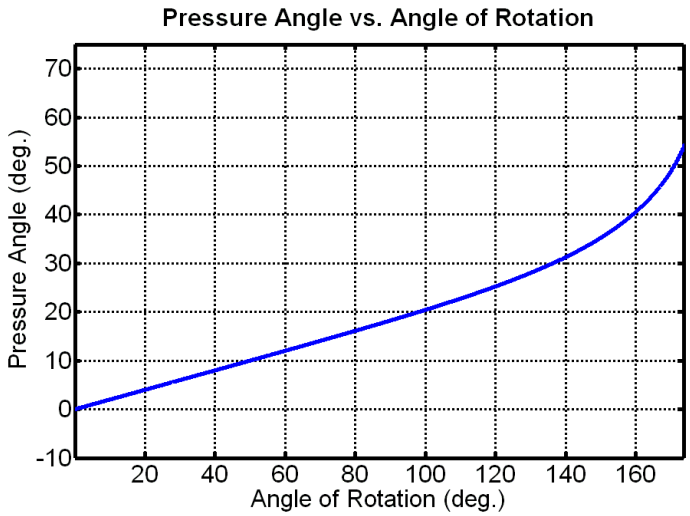


Figure 5.6: Plot of pressure angle vs. cam disk rotation angle.

The jaw closing force, F_{jaw}, and the horizontal force pushing against the guide rails, F_{rail}, are determined by the trigonometric relations in (5-9) and (5-10). F_{rail}

represents the total amount of horizontal force produced, though each rail only receives half of this total value. Equations 5-9 and 5-10 contain an additional term, $\eta_{bearing}$, which represents the efficiency of the guide block ball bearings. The parameter is estimated to be 0.90 based on established values [25].

$$F_{rail} = \eta_{bearing} \cdot F_{roller} \cdot \sin(\gamma) \tag{5-9}$$

$$F_{jaw} = \eta_{bearing} \cdot F_{roller} \cdot \cos(\gamma) \tag{5-10}$$

Using these equations, jaw closing force is plotted as a function of angle of rotation in Figure 5.7. The force acting against the guide rails and the total force (F_{roller}) are also included in the plot. These profiles are associated with higher forces than the track roller curve in Figure 5.4 because in this case, the forces from the upper and lower jaws are being combined to produce the total closing force.

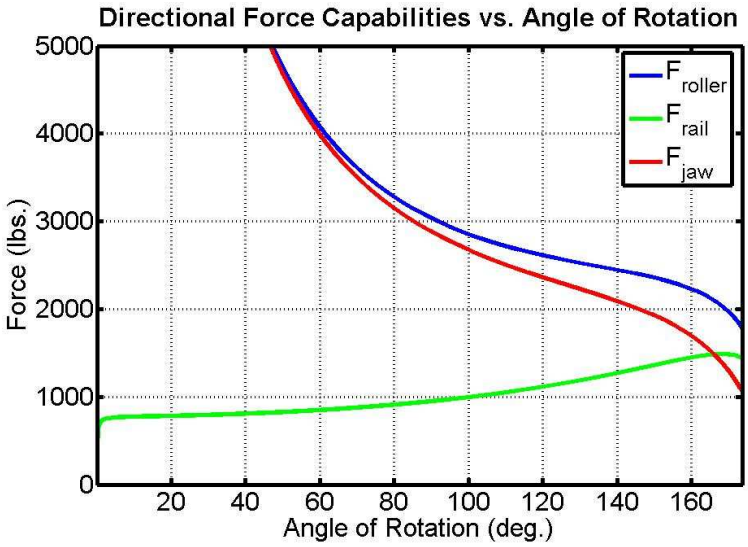


Figure 5.7: Plot of jaw closing force vs. angle of rotation.

The numeric values associated with the jaw-closed position in Figure 5.8 are listed in Table 5.1.

Table 5.1: Force values at the jaw-closed position (α_j = 173.56°).

Parameter	Value (lb)	Value (N)
F_{roller}	1784	7936
F_{rail}	1441	6410
F_{jaw}	1052	4680

Thus, disregarding structural considerations, the end-effector jaws will theoretically be able to close upon an object with a minimum force of 1052 lb (4680 N). This value would be more than adequate for achieving the end-effector sampling objectives.

5.1.3 Additional Performance Metrics

Profiles of jaw velocity and acceleration were generated as well. These plots appear in Appendix C and show the velocity to be continusiously decreasing while the acceleration curve remains relatively flat until the end of the profile, where it abruptly decreases. CAD analysis features were also used to generate plots of position, velocity, and acceleration. The corresponding data points and those from MATLAB were exported to Excel and were subsequently plotted together. Comparison plots were produced to verify the analysis and are also included in Appendix C. The comparison plots show that the data from the two different sources are identical.

5.2 Structural Calculations

The plot of the force output to the track rollers (Figure 5.4) revealed a substantial force at all angles of rotation. The minimum force in the profile is 892 lb (3968 N), nearly as high as the 955-lb (4248-N) dynamic load capacity of the track

rollers. This plot implies that overcurrent protection (OCP) soft stops will be necessary if a 5-A current is to be input to the hand roll joint.

As discussed in Section 4.4.2 and shown in Figure 5.4, the force capability is designed to continuously decrease to ensure the structural integrity of the track rollers. The factor of safety (FOS) of these devices is determined by dividing the 955-lb (4248-N) dynamic load capacity of the rollers by the force profile corresponding to a hypothetical input current of 1.0 A. Figure 5.8 shows the safety factor continuously increasing until the jaws close at 173.56°, where the FOS is 5.4.

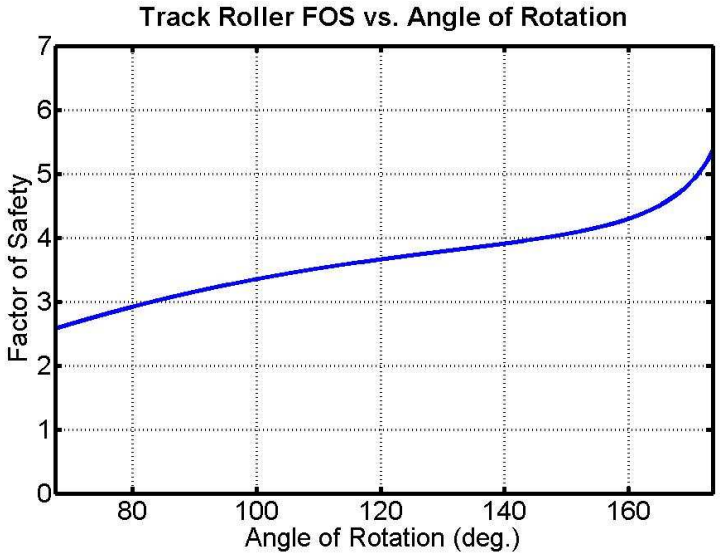


Figure 5.8: Plot of track roller FOS vs. rotation angle.

Although the track roller structural capabilities definitely require evaluation, the guide block ball bearings constitute the most probable failure mode in the current end-effector configuration. When the jaws clamp down on an object, the forces from the roller and the grasped sample induce reaction forces in the guide block bearings. The free body diagram (FBD) for the jaw assembly is shown in Figure 5.9.

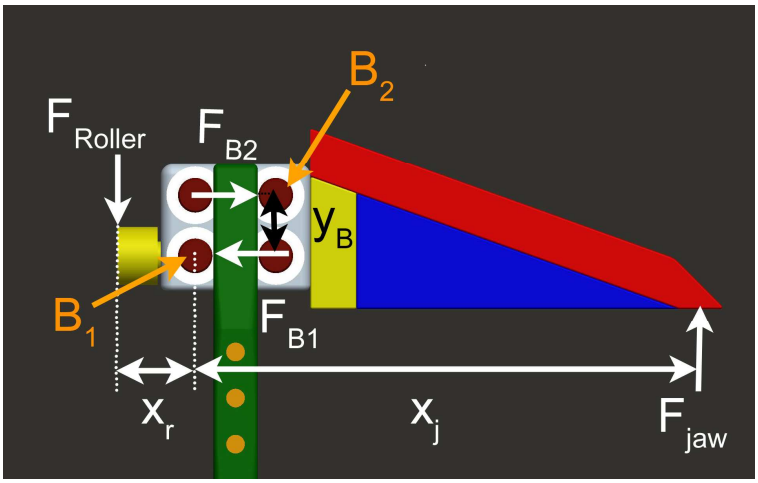


Figure 5.9: Jaw assembly free body diagram.

where B_1 and B_2 indicate the two nearside bearings counteracting the applied force, and F_{B1} and F_{B2} represent the bearing reaction forces. To determine the maximum jaw closing force, the moments will be summed about B_1. Dimensions y_B, x_j, and x_r, are the distances from the B_1 axis to B_2, F_{jaw}, and F_{Roller}, respectively. Equation 5-11 represents the sum of the forces in the x-direction, while (5-12) is the force balance for the y-components. The moments around Bearing 1 are then summed in (5-13).

$$\sum F_x = 0 = F_{B2} - F_{B1} \tag{5-11}$$

$$\sum F_y = 0 = F_{jaw} - F_{roller} \tag{5-12}$$

$$\sum M_{B1} = 0 = (F_{jaw}) \cdot (x_j) + (F_{roller}) \cdot (x_r) - (F_{B2}) \cdot (y_B) \tag{5-13}$$

Combining these three relations yields Equation 5-14.

$$F_{jaw} = \frac{(F_{B2}) \cdot (y_B)}{(x_j + x_r)} \tag{5-14}$$

Values for these variables are contained in Table 5.2. F_{B2} is set to 332 lb (1477 N), the load capacity of the ball bearings.

Table 5.2: Jaw assembly force analysis values.

Variable	Value (English)	Value (SI)
F_{B2}	332 lb	1477 N
y_B	0.655 in	1.66 cm
x_j	5.00 in	12.7 cm
x_r	.844 in	2.14 cm

Solving for F_{jaw} produces a force of 37.2 lb (165 N), which increases to 149 lb (663 N) when accounting for set of bearings on the opposite side of the guide block as well as those on the lower jaw. Thus, to maintain an FOS of approximately 2, the jaw closing force should be near 75 lb (334 N). Although this quantity is significantly less than the 1052 lb (4680 N) the cam disk is theoretically capable of generating, this value is substantial nevertheless. This limit will certainly allow the end-effector to grasp sand dollars and will provide the ability to fracture most desired specimens should it become necessary. If a higher closing force is desired, more ball bearings could be incorporated into the guide block design with minor alterations to the end-effector.

Additional structural analyses are presented in Appendix D. These analyses demonstrate that some of the end effector components have FOS values greater than 100; however, the values corresponding to some of the fasteners are approximately 10, which are comparatively low when juxtaposed with those on SAMURAI. In part, the high mechanical SAMURAI FOS values are produced by using A-286 superalloy fasteners. A-286 has a yield strength of 102 ksi (703 MPa), over three times greater than the 31.2 ksi (215 MPa) yield strength of the 18-8 stainless steel fasteners used in the end-effector. These superalloy fasteners are expensive and are not included in the

prototype for this reason; however, their inclusion in the design would increase some of the FOS values.

The structural analysis reveals that with OCP settings, the end-effector will be capable of operating without serious risk of structural failure, even without premium-grade fasteners. As previously mentioned, incorporation of different materials, a larger cam disk and rollers, or more conservative soft stops could further decrease structural risk.

5.3 Test Setup

During the performance of this end-effector research, the SAMURAI electronics were being developed in parallel. It follows that it is not currently possible to actuate the manipulator with SAMURAI electronics boards. However, spare Ranger electronics boards and test software could be used to power individual joint motors. With these boards, the joint will rotate at constant velocity, and the power supply will increase current up to the user-specified OCP limit to produce the additional torque needed to counter any resistance. By powering the hand roll joint with this method, it is possible to test end-effector functionality.

With SAMURAI in a state of limited functionality, specimens cannot be collected at present, but jaw closing force can be measured. To do this, the jaws were shut around one side of a scissors device while a digital force sensor was mounted at the other end. The sensor contains a peak value feature that allows it to record the highest registered force, which occurs when the OCP is triggered. The test setup is pictured in Figure 5.10.

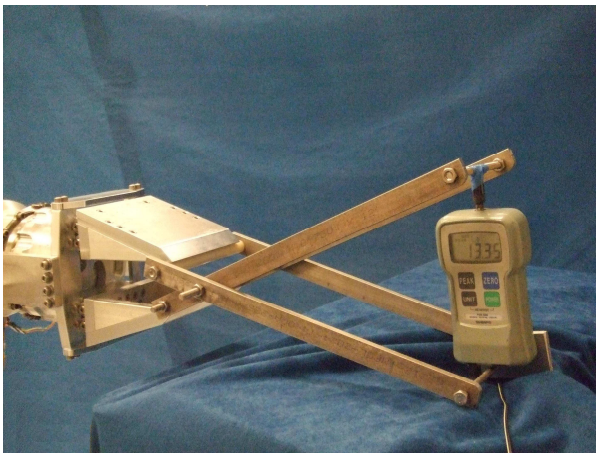


Figure 5.10: Force measurement test setup.

The distance between the jaws and the central scissors axis is 2 in (5.08 cm), while 10 in (25.4 cm) separate the center rod from the force sensor. This produces a moment arm that reduces the measured jaw closing force by a factor of five, as shown in (5-15). This scissors rig geometry was selected to ensure that the measurements would remain within the force sensor range, which is 0 to 50 lb (0 to 222 N).

$$F_{actual} = 5 \cdot F_{measured} \tag{5-15}$$

The scissors rig is placed at the end of the jaws, creating a moment around around the center of the guide block which will magnify the recorded force value. Referring to the distances in the free body diagram shown in Figure 5.9, the theoretical force is magnified by 5.92 (x_j/x_r = 5 in/0.844 in) to account for the rig placement.

The jaws clamp down on the scissor rig when they are 1.40 in (3.56 cm) apart, which corresponds to a rotation angle of 142°. While the force is expected to change at different rotation angles, this selection allowed for the simplest test setup and was deemed sufficient for establishing general end-effector performance.

The jaws were closed on the scissor rig for various motor current settings starting at 0.75 A and proceeding to 0.79 A in 0.01-A increments. An ammeter was connected in series with the power lines running to the motor, allowing for direct monitoring of the motor current. Increasing the current was expected to linearly increase the closing force. Five measurements were recorded at each current setting, and all data were subsequently plotted together.

In addition, a test was performed to determine if the jaws would bind under asymmetric loading. To perform this test, a rubber block 1.25 in (3.18 cm) in height was placed on one side of the lower jaw. The jaws were closed until the rubber was compressed 0.50 in (1.27 cm). The end effector was subsequently opened to check for binding.

5.4 Test Results

When the power supply to the hand roll motor is activated, the control electronics draw 0.67 A at 20 V when the joint is stationary. An additional 0.07 A are required to rotate the motor with no applied load. Thus, in the analysis, 0.74 A were subtracted from the input currents to determine force output. Figure 5.11 shows the actual data points with a superimposed linear trendline along with the predicted force values.

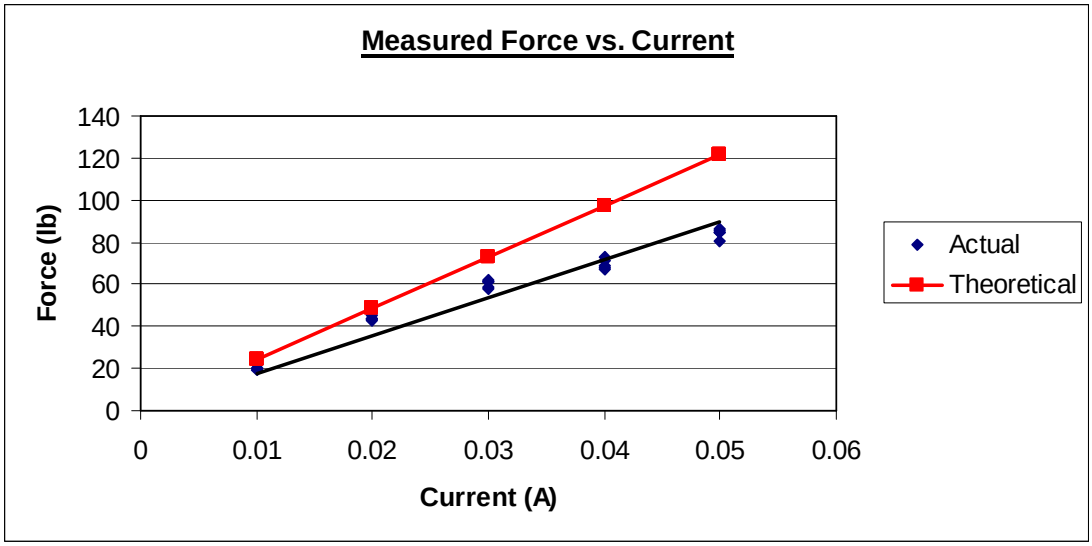


Figure 5.11: Plot of current vs. output force.

While the actual data are close to the theoretical values, there is a difference in slope. This could be caused by inaccuracies in the various efficiency estimates made in the theoretical model. A complete data table containing the values recorded during the testing is contained in Appendix D.

At 0.05 A, the force generated by the end effector is approximately 85 lb (378 N). This is comparable to the capability of the human hand, which has been shown to generate an average grasping force of 66 lb (284 N) and 102 lb (454 N) for women and men, respectively [26]. Thus, while the end effector cannot replicate the dexterity of the human hand, it is capble of replicating its strength.

Figure 5.11 does show that data recorded for the five different current settings are consistent. As expected, the data are characterized by a strong linear trend, which is evidenced by an R^2 value of 0.97. The registered force values demonstrate that the jaws are capable of functioning while applying relatively high closing forces.

To test asymmetric load conditions, the rubber stack was placed in the middle of one of the jaw side plates and subsequently compressed 0.50 in (1.27 cm). The test can be observed in Figure 5.12.

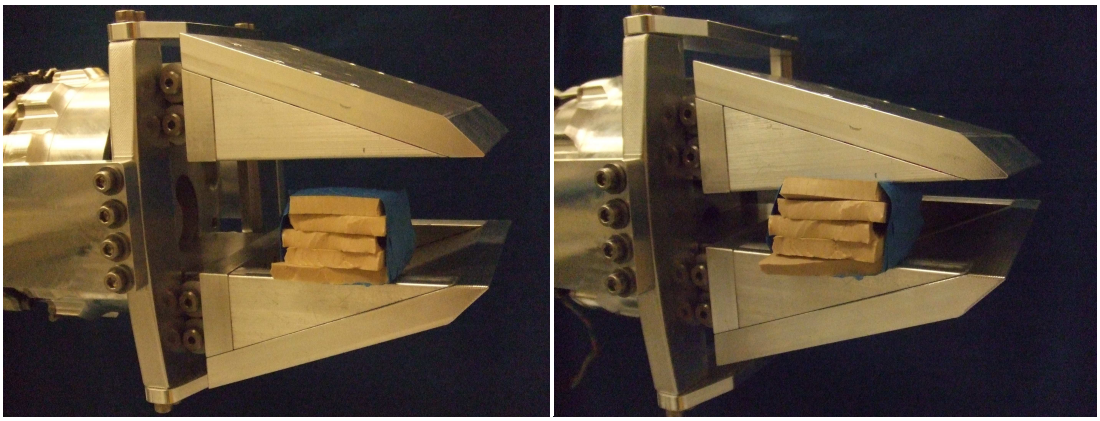


Figure 5.12: Assumetric load test with rubber stack before testing (left) and during testing (right).

Based on empirical data obtained with the same force sensor used in the closing force test setup, approximately 40 lb (178 N) of force are required to compress the block by this amount. After completion of the test, the jaws showed no evidence of binding and remained fully operational. The test was repeated five times, and the end effector retained functionality during each of these tests.

5.5 Summary

The end-effector was found to have a maximum jaw opening of 2.46 in (6.25 cm) and a possible closing force of 1052 lb (4680 N) if structural considerations are disregarded. The guide block ball bearings were determined to be the weakest structural end-effector components, limiting the jaw closing force to 75 lb (334 N) with an FOS of 2.

Jaw closing force was tested using a force sensor mounted on a scissor rig. Measured values were plotted against varying input currents, and the results were compared to theoretical predictions. Recorded data indicated a strong linear current-force relationship, and relatively high closing forces (85 lb, 378 N) were observed with no disruption to end-effector functionality. This functionality was maintained even in cases of asymmetric loading.

Chapter 6
Sample Container Design and Testing

As the objectives of the end-effector entail the collection of multiple samples, the various specimens are to be stored in several separate containers. These storage units must interface with the end-effector geometry. This chapter is devoted to sample container design and testing.

6.1 Sample Container Design

The sample container engineering objective was to produce a design that is functional but as simple as possible. The containers were to be entirely passive; however, some sort of cover was necessary to ensure sample retention. Some target specimens are neutrally buoyant; thus, a method to scoop samples from the jaws was necessary.

Before determining the container cover, the primary geometry needed to be established. Cylinders were selected for structural and hydrodynamic reasons, and PVC was chosen as the cylinder material due to its relatively low density. The low density generates buoyancy, which reduces weight concerns.

The diagonal distance across the end-effector face is 8.011 in (20.348 cm), as shown in Figure 6.1. Thus, the inner diameter of the PVC cylinder must be larger than this dimension to allow for insertion.

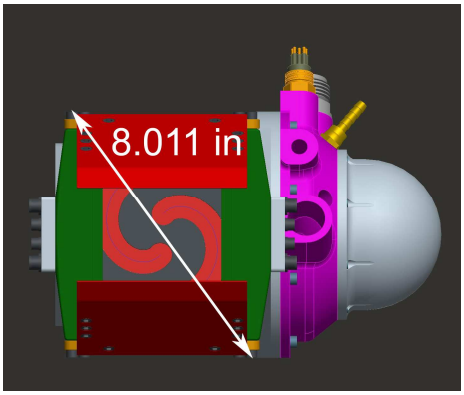


Figure 6.1: CAD model highlighting end-effector diagonal dimension.

After examining commercially available PVC cylinders, a 10-in (25.4 cm) nominal diameter, Schedule 80, cylinder was selected. To accommodate the end-effector length, the cylinder height is 10.88 in (27.64 cm). The frontal projection area for one container is 117 in^2 (755 cm^2), creating a drag profile area of 234 in^2 (1510 cm^2) assuming containers are to be positioned on either side of JAGUAR. Eight ¼ in (0.635 cm) holes are drilled into the container sides to allow for pressure equalization during depth changes. The container base plate was set as 3/8 in (0.953 cm) thick PVC sheet as it was readily available at the SSL. Parameters relating to sample container geometry are compiled in Table 6.1.

Table 6.1: Sample container geometry parameters.

Parameter	Value (English)	Value (SI)
Material	Schedule 80 PVC	Schedule 80 PVC
Outer Diameter	10.75 in	27.31 cm
Inner Diameter	9.49 in	24.10 cm
Height	10.88 in	27.64 cm
Projected Area (side)	234 in^2	1510 cm^2

Rubber was chosen for the lid material, which is to function similarly to a garbage disposal to trap the specimen. Specifically, natural gum rubber was selected as it is virtually neutrally buoyant and is resistant to abrasions, tears, and impacts.

Cuts are made into a rubber sheet to allow for end-effector insertion. When the end-effector is extracted, the rubber flaps ride up with the end-effector before folding out into the jaw, ensuring that the samples are retained in the container. A second, smaller rubber ring is positioned beneath the primary ring to offer structural support. A PVC ring is used to fasten the rubber sheets to the cylinder. The various sample container components can be seen in the views in Figures 6.2 and 6.3.

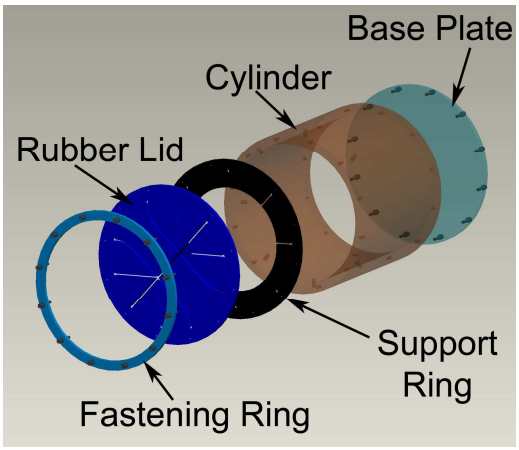


Figure 6.2: Exploded sample container CAD model.

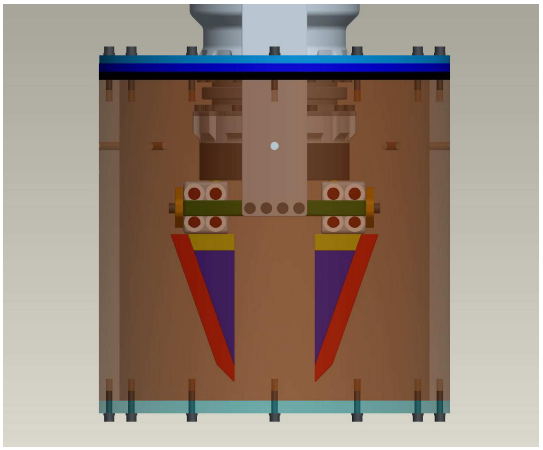


Figure 6.3: CAD model of end-effector insertion.

Since weight is a concern, calculations were performed to determine the sample container wet weight. This evaluation is contained in Appendix E. The sample containers were determined to be 3.47 lb (15.44 N) negatively buoyant, but the evaluation also determined that this weight could be negated by incorporating 202 in^3 (3310 cm^3) of syntactic foam into the design. Figure 6.4 is a reproduction of Figure 6.3; however, the PVC sample container has been hidden and a potential 202 in^3 (3310 cm^3) syntactic foam configuration is displayed.

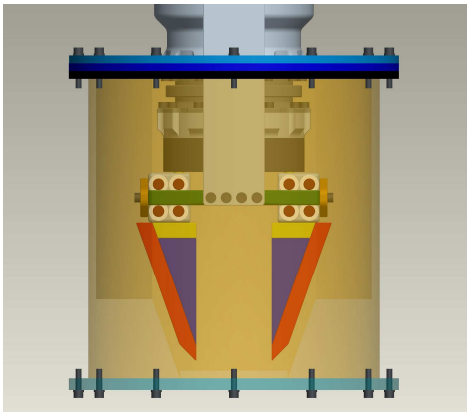


Figure 6.4: CAD Model of Container Internal Syntactic Foam

While it would not be necessary to employ this specific configuration, it demonstrates that there is sufficient internal volume for enough foam to render the containers neutrally buoyant. It may be possible to compensate for the sample container weight by incorporating the foam external to the sample containers, but that is a determination that will be made by WHOI personnel.

6.2 Sample Container Testing

The manufactured sample container can be seen in Figure 6.5.

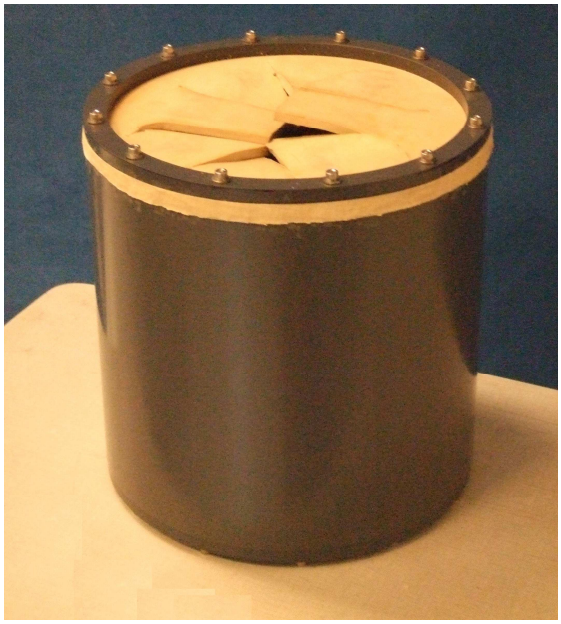


Figure 6.5: Sample container prototype.

To determine whether the flaps would fold as desired during end-effector insertion and extraction, a sample container was constructed and placed on a wheeled table of adjustable height. The table was adjusted as necessary, pushed into the stationary manipulator, and subsequently removed. This basic setup can be seen in Figure 6.6.

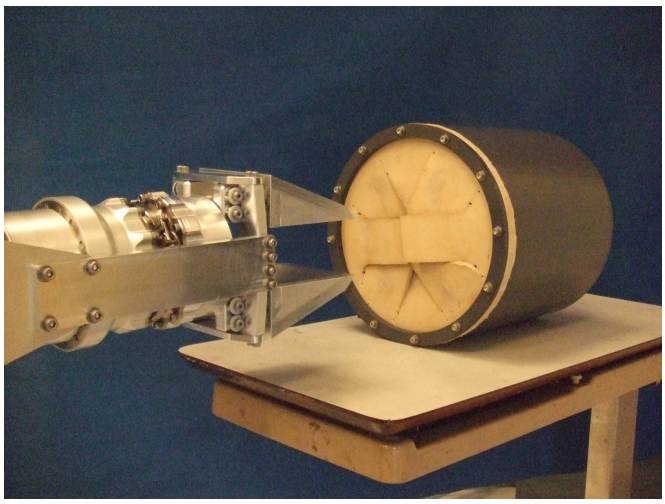


Figure 6.6: Sample container insertion/extraction test setup.

To test the effectiveness of the rubber flaps, a plastic egg was placed inside of the open jaws, and the jaws were subsequently closed. The table with the sample container was pushed over the closed end-effector, and the jaws were opened. When the sample container was pulled away from the manipulator, the rubber flaps folded out as designed, and the egg remained inside the sample container. This sequence can be observed in Figure 6.7. The final image in sequence shows the container positioned vertically with the flaps pulled back to reveal the egg inside.

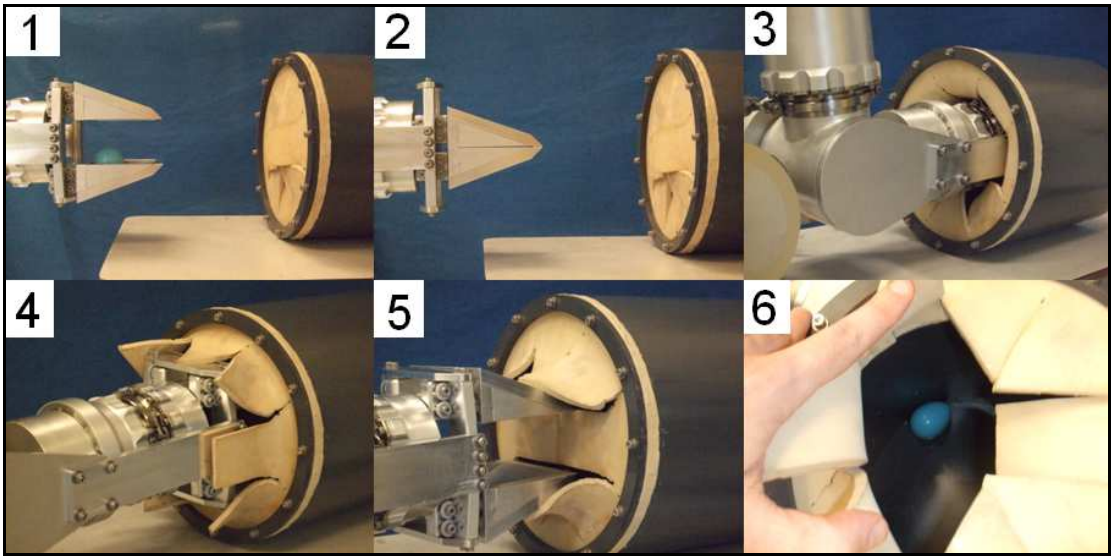


Figure 6.7: Sample container function demonstration.

6.3 Summary

The sample containers to be used in the SAMURAI collection system are to be composed of Schedule 80 PVC cylinders with a 10-in (15.4-cm) nominal diameter. Rubber lids with flaps cut into them represent the upper container covers. The flap geometry is such that they fold out to ensure that samples contained in the jaws remain in the sample containers.

Physical testing of the containers in conjunction with the end-effector was performed by demonstrating an insertion and extraction procedure. Testing demonstrated that the flaps function as designed and will ensure that even neutrally buoyant specimens are retained inside of the sample container.

Chapter 7
Kinematics

Having determined the sample container geometry, it was necessary to establish the maximum possible quantity and corresponding locations for the storage units on the JAGUAR vehicle. These determinations are dependent upon the SAMURAI kinematics and work envelope. Thus, basic SAMURAI inverse kinematics software composed by Carignan [2006] was used as a skeleton to develop a robust program to make the necessary determinations.

This chapter presents the forward and inverse kinematics used for the sample container evaluations. Details of a graphical user interface (GUI) used to produce easy access to a plethora of kinematics options are also presented. The SAMURAI range of motion is outlined, and sample container quantity and corresponding locations are suggested.

7.1 Effects of Removing the Hand Roll Degree of Freedom

Removing a degree of freedom from a manipulator will degenerate the kinematics, and this is certainly true of the effective elimination of the SAMURAI hand roll joint. The largest consequence for sampling is the limitation on sample orientation. The end-effector is designed to collect a sand dollar positioned flat on the ocean floor, and this can be accomplished with or without the hand roll degree of freedom. However, in the hypothetical case in which the sand dollar is propped up against a rock towards vertical, if the hand roll joint is accessible, the grippers would be rotated to match the sample orientation. By removing this DOF, it will be much

more challenging and perhaps impossible to acquire the sample. The end-effector could potentially be oriented as necessary through a combination of Joints 2, 3, 4, and 5, but in this case, what was a trivial exercise in kinematics is now a much more difficult problem.

Additionally, when depositing the samples in their respective containers, if the hand roll joint is in use, the grippers can be rotated to match the container orientation. Thus, container orientation when mounted to JAGUAR is not particularly important. However, when this DOF is removed, there is a solitary configuration in which the vector normal to the storage unit lid will match the axis of the hand roll joint. In this latter case, the attachment to the AUV must be made with precision.

7.2 SAMURAI Kinematics Software

To determine sample container position and orientation for the manipulator in a 5-DOF configuration, kinematics software was developed. Significant enhancements were made to the existing framework of both forward and inverse kinematics calculations.

7.2.1 Description of Pre-Existing Software

During manipulator development, multiple programs in Mathematica and C were computed to characterize the SAMURAI kinematics [27]. The inverse kinematics program was the one applicable to the sample container issue and was selected as a starting point for the subsequent kinematics development.

In its present configuration, the code combines the known Denavit-Hartenberg (DH) parameters with programmed joint angles to generate the relevant position

vectors, rotation matrices, and tool position using forward kinematics. These results are then used in an inverse kinematics solution, which demonstrates code functionality by outputting joint angles identical to the originally-programmed values.

7.2.2 MATLAB Kinematics GUI Overview

This program was converted into MATLAB and subsequently expanded to incorporate a wide range of features. The program files comprising the enhanced software are contained in Appendix G along with an overview of the function of each file.

A GUI was constructed to allow the user to select options and insert values without modifying the program itself. The GUI is shown in Figure 7.1.

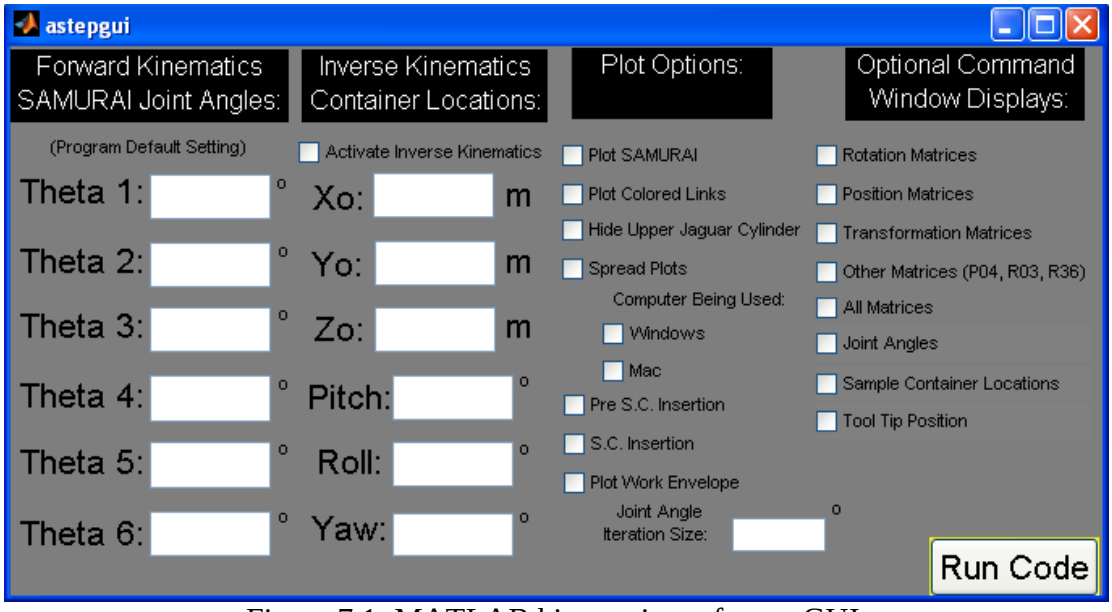


Figure 7.1: MATLAB kinematics software GUI.

7.2.3 Forward Kinematics

Each SAMURAI joint has a coordinate frame with its respective z-axis oriented to coincide with the axis of rotation. Frames 1 through 6 correspond to the

six joints. In addition, Frame 0 represents the global frame and is positioned beneath the center of the shoulder yaw joint (Joint 1). Frame T is located at the tool tip at the forward end of the manipulator. The SAMURAI coordinate frames can be seen in Figure 7.2. The figure shows the manipulator in the configuration in which all joint angles are set to 0°.

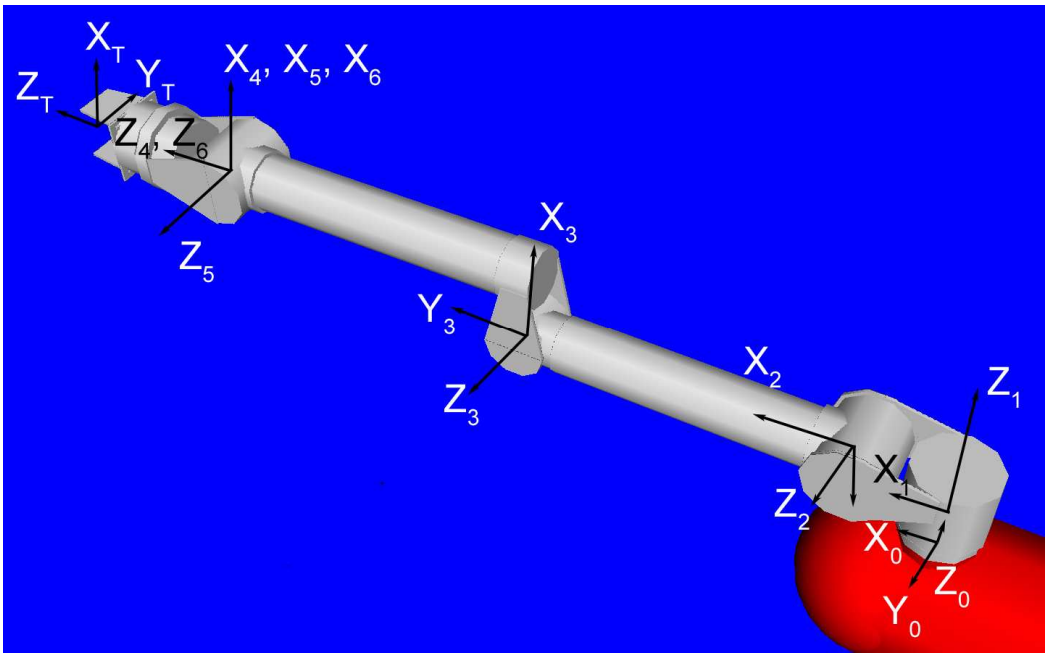


Figure 7.2: SAMURAI kinematics coordinate frames.

The DH parameters that determine the manipulator kinematics are provided in Table 7.1. While the hand roll joint angle (θ_6) would ordinarily be a variable, it has been set to 0° in Table 7.1 because it is being used to power the end-effector and will have no kinematic impact.

Table 7.1: SAMURAI DH parameters.

i	α_{i-1} (deg.)	a_{i-1} (m)	d_i (m)	θ (deg.)
1	0	0	0.108	θ_1
2	-90°	0.152	0	θ_2
3	0	0.610	0	θ_3 - 90°
4	-90°	0.114	0.610	θ_4
5	90°	0	0	θ_5
6	-90°	0	0	0
T	0	0	0.441	0

The kinematics software sets the α_{i-1}, a_{i-1}, and d_i DH parameters to their constant values and creates an array of joint angles based on the user specifications input through the GUI. The software uses these values to create transformation matrices. The form of a generic transformation matrix is featured in Equation 7-1[5] [28]. In this equation, ${}^{i-1}_{i}T$ represents the transformation matrix used to locate Frame i relative to Frame i-1.

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7-1} $$

The 4x4 homogenous transformation matrix is composed of a 3x3 rotation matrix in the upper left corner. Additionally, it contains a position vector whose respective components are featured in the first three elements of the fourth column.

The transformation matrices representing each of the coordinate frames relative to Frame 0 are found by beginning with the base frame and multiplying the matrices sequentially, as shown in Equation 7-2.

$$ {}^{0}_{N}T = {}^{0}_{1}T \cdot {}^{1}_{2}T \cdot {}^{2}_{3}T \cdot \ldots {}^{N-1}_{N}T \tag{7-2} $$

[5] The letters “c” and “s” constitute the shorthand notation for the sine and cosine functions.

After the joint angles are input to the forward kinematics, the program computes the local transformation matrices. These matrices are then cascaded as shown in (7-2) to produce transformation matrices relative to Frame 0. Global frame positions, represented by the vector ${}^{0}P_{N}$, are determined by isolating the position components in ${}^{0}_{N}T$. This procedure is outlined in Equation 7-3.

$$ {}^{0}P_{N} = \left({}^{0}x_{N},\ {}^{0}y_{N},\ {}^{0}z_{N} \right) = \left({}^{0}_{N}T(1,4),\ {}^{0}_{N}T(2,4),\ {}^{0}_{N}T(3,4) \right) \tag{7-3} $$

The program then generates the SAMURAI links by taking two successive positions and using three-dimensional plotting functions to generate lines between the points. The process is repeated until reaching the tool frame at which point the entire manipulator has been produced.

The SAMURAI representation is superimposed on a model of JAGUAR, which was created using a collection of MATLAB plotting tools. In the kinematics analysis, it is assumed that SAMURAI will mount to JAGUAR on the upper surface of the plane where the AUV hemispherical cap joins with the cylinder. The model also includes sample containers, which are set to default locations when using forward kinematics. An image of the GUI output showing the manipulator in its stowed configuration is shown in Figure 7.3. An isometric view and three two-dimensional views show the manipulator from all perspectives while the software repositions the plots around the GUI.

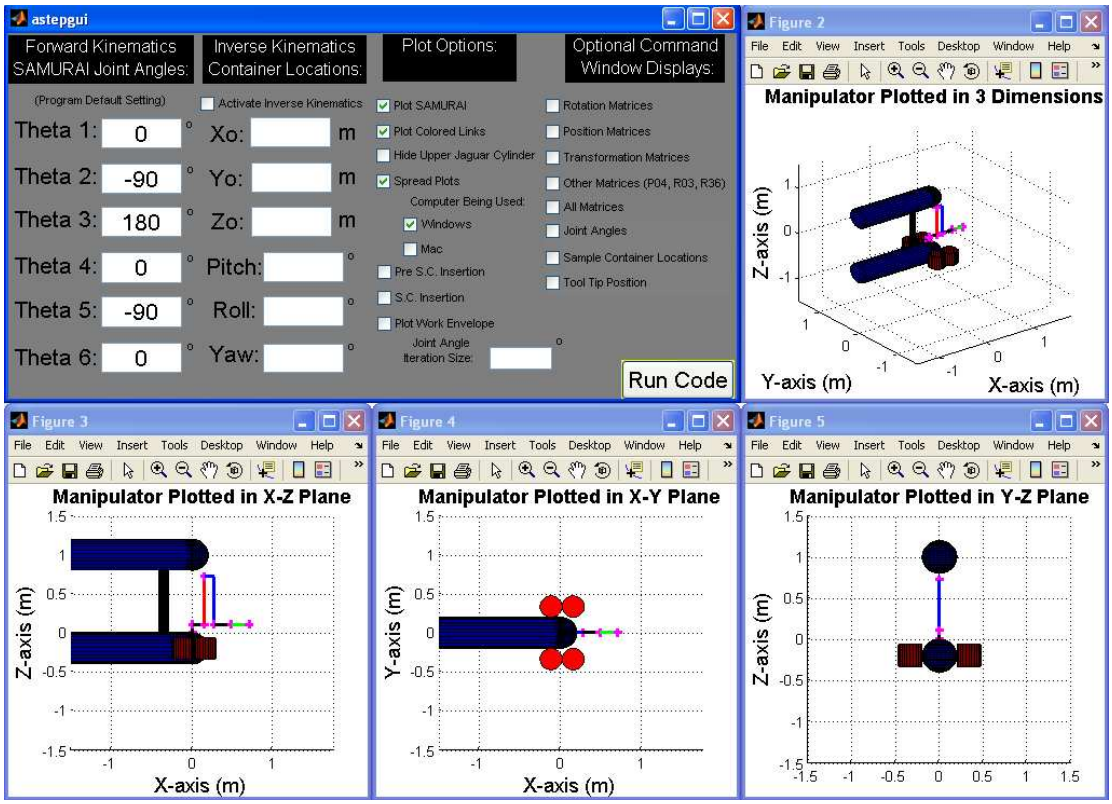


Figure 7.3: Kinematics GUI with plots of SAMURAI in stowed configuration.

All joint angles, position vectors, and rotation matrices are automatically saved to a data file in the working directory. Every time the code is compiled, the previously saved file is overwritten. A screenshot of the data file corresponding to arbitrary joint angles is shown in Figure 7.4.

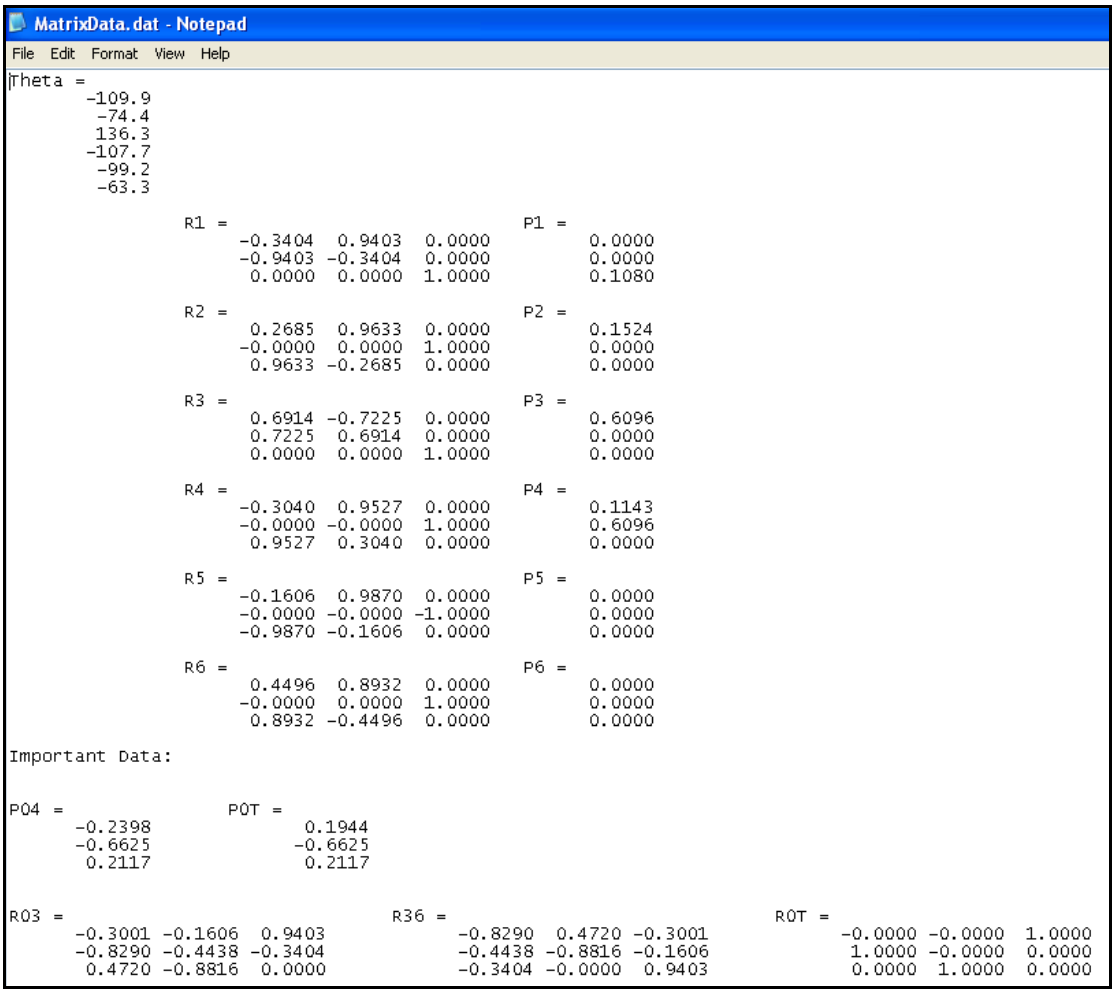


Figure 7.4: Screenshot of automatically-generated kinematics data file.

The user also has the option of displaying these parameters in the MATLAB command window. If only one or two parameters are of interest, the command window displays are ideal; however, the data file eliminates the need for excessive scrolling and instantly provides significant manipulator data in a cohesive format. Additionally, the data file can easily be imported to Excel and/or other programs if further analysis is desired.

7.2.4 Inverse Kinematics

Inverse kinematics is employed to determine joint angles based on user-input sample container position and orientation. The solution follows the procedure outlined by Carignan [27], and a detailed description of that process is contained in Appendix F. With the sample container inputs, the software generates the corresponding joint angles, which are then used to plot the manipulator with the forward kinematics approach previously described. An image of the GUI output showing the manipulator immediately prior to sample container insertion is shown in Figure 7.5.

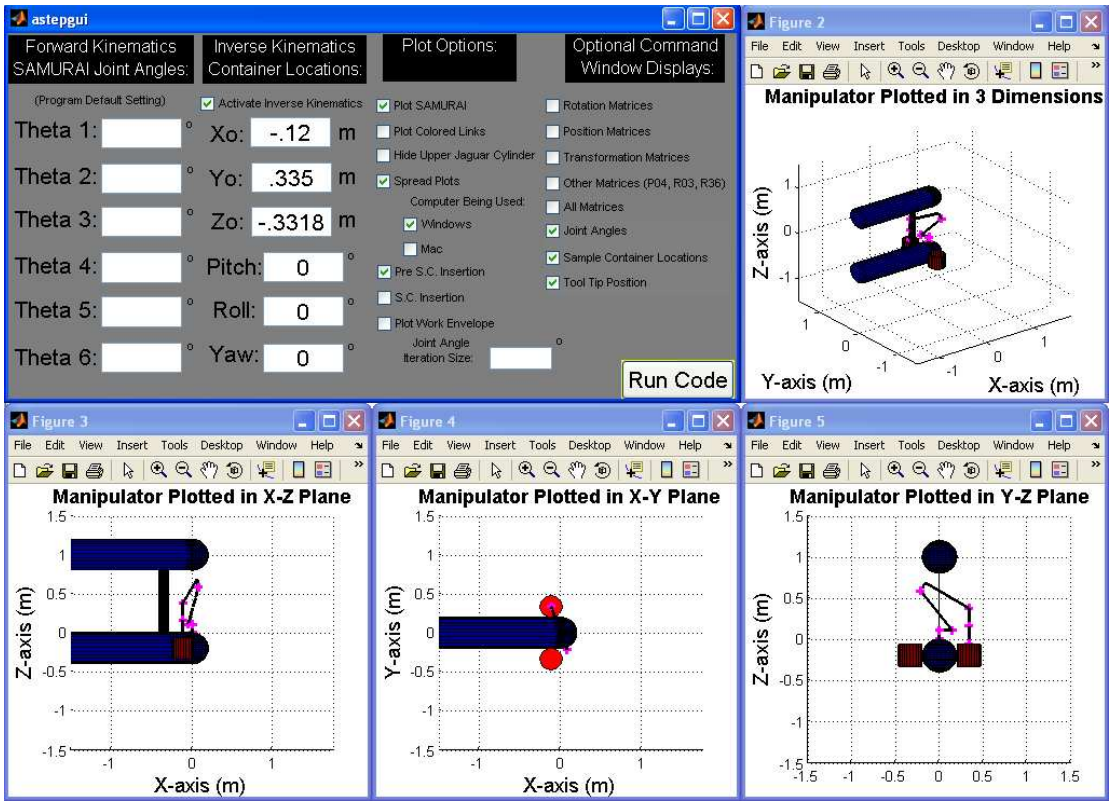


Figure 7.5: GUI with SAMURAI in sample container pre-insertion orientation.

Thus, the software allows the user to select any sample container location and orientation and will output the joint angles required to insert the end-effector into the

container and a visualization of the manipulator in that configuration. This is especially useful in that the storage unit mounting points have not been determined. If the sample containers need to be mounted in a certain manner, it will be easy to evaluate whether or not that that will generate a feasible set of joint angles for the SAMURAI manipulator. In addition, it is possible to establish potential sample container quantity by using the software to generate the workspace and placing containers inside of it. However, this is dependent upon joint ranges of motion, which are addressed in Section 7.3.

7.2.5 Kinematics Software Limitations

While the software provides many new capabilities, it does have limitations. In its current state, the program does not compute joint rates or Jacobians, nor does it actively monitor for singularities or sample container locations outside of the workspace. If the user requests an impossible sample container location or orientation corresponding to a singularity, the software will not execute as it is unable to construct the geometry. While this indirectly informs the user that the inputs are problematic, an improved system for monitoring for impossible configurations should be added to the code, and this constitutes a possible area of future work.

7.3 SAMURAI Range of Motion

To determine how many sample containers could be used on Jaguar, it is necessary to establish how many units could be contained within the SAMURAI workspace. Determination of the workspace requires values for each of the joint ranges of motion. Internal hard stops were known to limit the motion of the shoulder

yaw, elbow roll, and hand roll joints to 220°, 540°, and 540° respectively. The three pitch joints have external hard stops represented by the manipulator itself. The ranges of motion for these joints were determined by recording images of the manipulator at range-of-motion extremes. Figure 7.6 is an example of one of these images.

Figure 7.6: Image of the elbow pitch joint at maximum rotation.

A protractor was then used to measure the extreme angles and establish motion ranges. A detailed description of the range of motion determination procedure is included in Appendix G. The joint ranges are shown in Table 7.2.

Table 7.2: SAMURAI joint ranges of motion.

Joint Number	Range of Motion
1 (Shoulder Yaw)	220°
2 (Shoulder Pitch)	225°
3 (Elbow Pitch)	210°
4 (Elbow Roll)	540°
5 (Wrist Pitch)	215°
6 (Hand Roll)	540°

With the joint ranges of motion known, the SAMURAI work envelope can be plotted by using the forward kinematics software and iterating through the joint motion ranges. The kinematics GUI has an option for plotting the work envelope and

allows the user to specify the resolution in degree increments. Only half of the symmetric envelope is displayed to reduce compilation time and to allow for better visibility, as shown in Figure 7.7. In Figure 7.7, the plots have been generated using 2° increments.

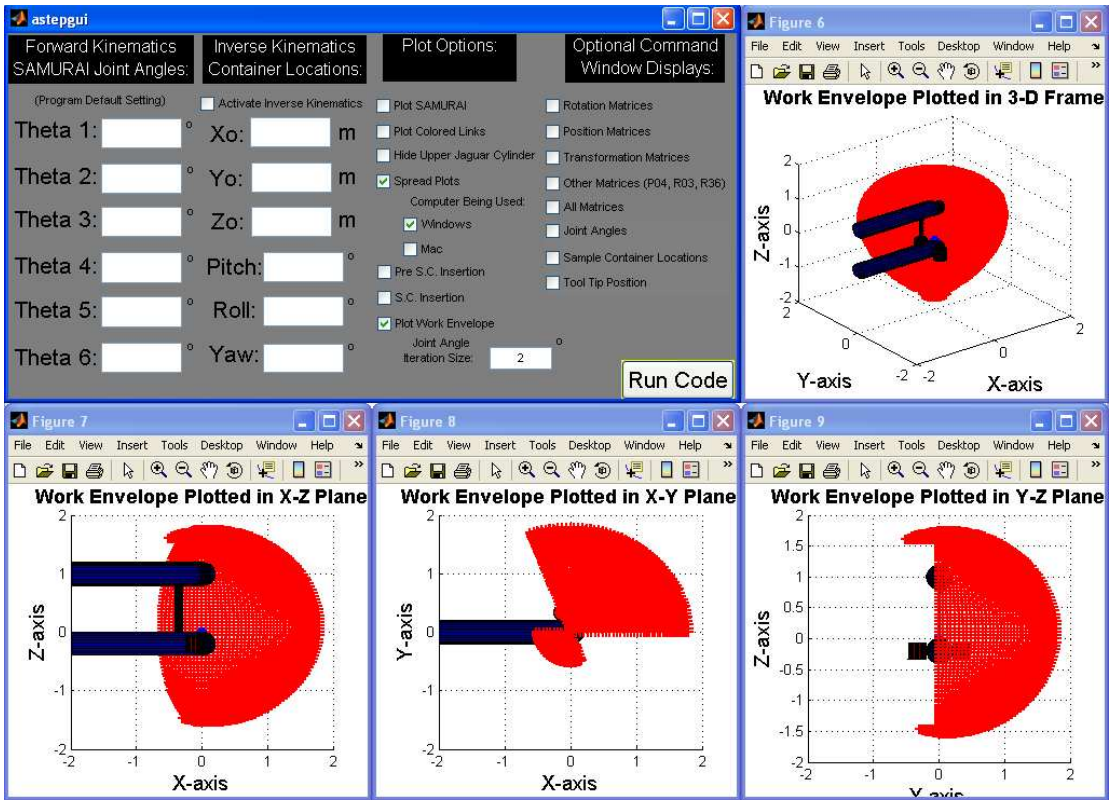


Figure 7.7: SAMURAI work envelope plotted in MATLAB.

7.4 Sample Container Quantity and Location

To maximize sample container quantity, the containers are placed at the end of the workspace on either side of the JAGUAR AUV. As shown in Figure 7.7, the SAMURAI workspace does not extend far down the length of JAGUAR. This is attributable to the shoulder yaw joint hard stop, which limits the total range of motion to 220°. The hard stop is included in the design to ensure that SAMURAI would not inadvertently damage aft JAGUAR components.

Two additional containers can be added forward of the initial two. The resulting configuration is modeled in Figure 7.8. These latter two extend beyond the front of JAGUAR, but attachment to the AUV and/or the other sample containers should be possible.

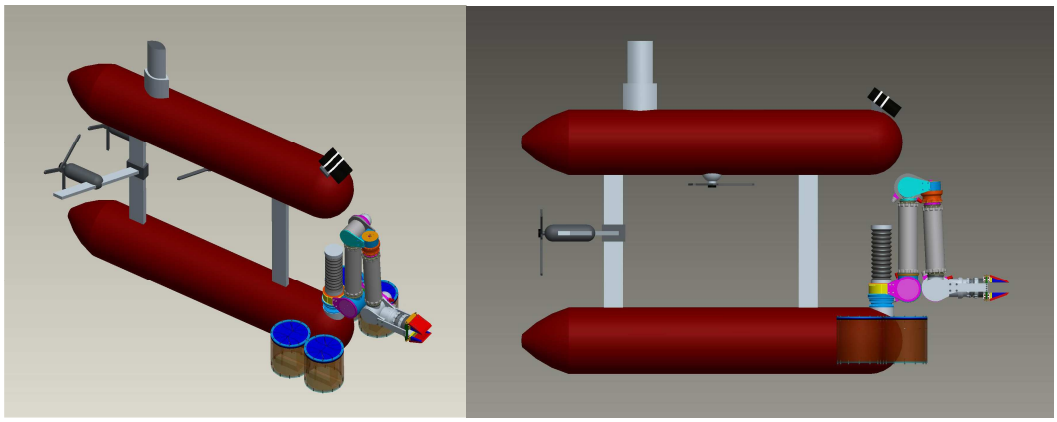


Figure 7.8: Isometric (left) and 2-D (right) views of JAGUAR/SAMURAI CAD models with sample containers.

The sample container locations are listed in Table 7.3. These coordinates correspond to the center of the container base plates and are relative to Frame 0, which once again, is located on the underside of the SAMURAI shoulder yaw joint.

Table 7.3: Sample container coordinates.

Container	X-Coordinate (in)	Y-Coordinate (in)	Z-Coordinate (in)
1	-4.72	13.19	-13.06
2	-4.72	-13.19	-13.06
3	6.03	13.19	-13.06
4	6.03	-13.19	-13.06

The joint angles corresponding to these locations are listed in Table 7.4. Values both immediately before sample container insertion and after sample container insertion are listed.

Table 7.4: Sample container pre- and post-insertion joint angles.

	Container 1		Container 2	
Joint	Pre-Insertion (°)	Post-Insertion (°)	Pre-Insertion (°)	Post-Insertion (°)
1	109.6	109.6	-109.6	-109.6
2	-128.7	-83.8	-128.7	-83.8
3	158.9	171.6	158.9	171.6
4	180	180	180	180
5	-59.7	-2.1	-59.7	-2.1
6	-160.4	-160.4	-19.6	-19.6
	Container 3		Container 4	
Joint	Pre-Insertion (°)	Post-Insertion (°)	Pre-Insertion (°)	Post-Insertion (°)
1	65.4	65.4	-65.4	-65.4
2	-126.6	-82.9	-126.6	-82.9
3	158.1	170.4	158.1	170.4
4	180	180	180	180
5	-58.4	-2.5	-58.4	-2.5
6	155.4	155.4	24.6	24.6

The θ_6 values vary in Table 7.4, and this is because in each of these cases, the sample container yaw, pitch, and roll values were all set to zero. Thus, the θ_6 values in Table 7.4 are actually representative of the amount by which the sample containers themselves must be rotated in order to align with the end-effector.

If WHOI determines that JAGUAR is able to operate with additional profile drag, several more sample containers could be positioned outside of the current two rows. Though, it is expected that the addition of just SAMURAI and two rows of sample containers will have very adverse effects on the hydrodynamic properties of the AUV which could restrict system performance and mission parameters.

7.6 Summary

Existing SAMURAI kinematics software was modified to create a program that accepts a multitude of user inputs. The developed program applies forward and

inverse kinematics to generate data files, create plots for visualization, and establish the SAMURAI work envelope.

The software was utilized to determine sample container locations and quantity. Four containers can reasonably mount to JAGUAR within the SAMURAI workspace, possibly more if the AUV will be able to perform with additional profile drag.

Chapter 8
Conclusion and Future Work

This thesis presented the design of an end-effector to be used in conjunction with autonomous underwater sampling missions. End-effectors currently in use on submersibles, the preferences of the marine biology community, and previous SSL projects were considered during concept selection. The chosen jaw concept and corresponding sample container were designed, constructed, and tested. In addition, kinematics software was developed to ascertain sample container location and quantity. The result is a sampling system which achieves all design objectives.

8.1 Conclusions

Although robotic devices are used in myriad of applications on the surface, complex problems still need to be solved in order render the attachment of manipulators to AUVs routine [29]. The SAMURAI/JAGUAR sampling system aims to perform innovative research by collecting biological specimens in extreme environments.

The end-effector represents the device that will physically collect the samples and deposit them in containers. A gripper design was selected due to successful past implementation and popularity amongst marine biologists. Jaws constitute the grippers themselves to allow for potential sample diversity.

The existing titanium flange and fastener holes in the hand roll joint housing were selected as end-effector attachment points to avoid any modifications to the

SAMURAI structure. Side mounting braces fixed to the fastener holes allow for the relative motion necessary to actuate the jaws.

A cam disk was selected as the method of actuation due to past SSL success with the design and concerns that gear teeth could potentially skip or fracture. With the exception of the cam disk, the rest of the end-effector is designed to be both easy to machine and easy to construct. The fasteners are nearly entirely 10-32 screws, which render the assembly and disassembly procedures simple.

End-effector performance was found to be limited by the structural constraints of the guide block bearings. Nevertheless, the jaws were found to be capable of safely outputting a closing force of 75 lb, and this was verified with physical testing.

A sample container to be used in conjunction with the end-effector was designed and tested as well. PVC was selected as the container material due to its favorable buoyancy properties, and the lid was made out of rubber to render the unit as simple as possible. Testing demonstrated that the lid flaps fold out effectively to remove specimens from the end-effector jaws and retain them in the container.

It was determined that four sample containers should be used on JAGUAR, though more storage units could be included if additional profile drag is permissible. Kinematics software was enhanced to ascertain locations for the containers as well as the corresponding manipulator joint angles.

This work has produced both functional hardware and software. The end-effector operates as designed, opening and closing smoothly and interfacing with the sample container. Although further testing and design enhancements will likely be

necessary, the device could be employed on a submersible and successfully retrieve samples.

8.2 Future Work

The actual utility of this or any other SAMURAI end-effector will only be established when electronics to control the entire manipulator have been developed. When the electronics are operational, the end-effector should be used to simulate the collection of actual samples. Sand dollars and other potential specimens should be placed in sand and the entire arm used to grasp the objects and place them in the sample containers. The SSL is currently developing a structure to replicate the JAGUAR geometry. By attaching sample containers to the JAGUAR mockup, the effectiveness of the end-effector-sample container combination should be evaluated through testing.

After a more extensive test plan is used to assess end-effector performance, the prototype design should be finalized. The cam disk should be modified to interface with the titanium flange directly, eliminating the need for the adapter. This would reduce the end-effector mass by 0.71 lb_m (0.32 kg) and length by 0.814 in (2.068 cm). The cam disk should also be anodized to increase wear resistance.

Parts used to reduce cost in the current design should be replaced with high-grade components. The steel track rollers should be replaced with stainless steel, and the 18-8 stainless fasteners should be replaced by A-286 parts. This latter alteration will increase the FOS of several components and eliminate any confusion with fasteners used elsewhere on the manipulator.

Different jaw plates could be developed to increase the ability to break or grip samples. For example, by decreasing the surface area of the jaw surface, the pressure exerted on a sample could be increased substantially. In the concept shown in Figure 8.1, the flat plate edges have been replaced by a raised edge on the upper jaw and a mating edge on the lower jaw. The material removed on the lower jaw allows the jaws to shut completely.

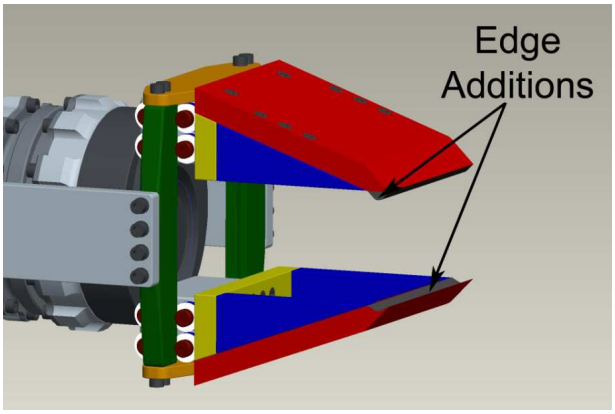


Figure 8.1: CAD model of jaw plate with raised edge concept.

Concepts using features such as this, serrated edges, and other geometries could be developed and tested without extensive effort. These designs could also be incorporated into the jaw side plates should such a configuration allow for easier acquisition of a sampling target.

WHOI will need to be consulted regarding several end-effector-related tasks. Additional work needs to be performed to attach the sample containers to JAGUAR. WHOI will be responsible for the information pertaining to the necessary modifications that will allow the containers to mount to the AUV structure. WHOI will also need to verify that the system will remain operational even with the

additional drag produced by SAMURAI and the sample containers. The quantity of syntactic foam required to offset component mass also needs to be established.

The sample container software should be modified to include checks for impossible container locations and arm configurations. The program could be further enhanced to incorporate trajectories and possibly animations. These trajectories could be used for the manipulator path planning that will be necessary when the electronics development has been completed and the manipulator is fully operational.

Appendix A

End-Effector and Sample Container Hardware

Table A.1: List of end-effector and sample container CAD drawings.

END EFFECTOR		
Drawing #	Drawing Type	Description
FD20-0070	Part	Female Flange (End Effector Attachment Flange)
EE-0001e	Part	Flange/Cam Disk Adapter
EE-0003	Part	Cam Disk
EE-0004	Part	Track Roller/Cam Follower
EE-0012	Part	Ball Bearing for Block Slider
EE-0014	Part	Custom Guide Block
EE-0015	Part	Custom Guide Rail
EE-0015b	Part	Custom Guide Rail - Penetrator Plate Side
EE-0016	Part	Delrin Thrust Bearings
EE-0017	Part	Jaw Side Plate
EE-0018	Part	Side Mounting Brace
EE-0018b	Part	Side Mounting Brace (penetrator plate side)
EE-0019	Part	Jaw Top Plate
EE-0020	Part	Jaw Back Plate
EE-0022	Part	Guide Rail Connector
AYEE-0005	Assembly	Custom Guide Block Assembly
AYEE-0006	Assembly	Shoulder Screw, Ball Bearing, Thruster Bearing Combination
AYEE-0007	Assembly	Wrist Joint Pair/End Effector Assembly
AYEE-0008	Assembly	Custom Jaw (Top, Back, Side Plates, and Fasteners)
SAMPLE CONTAINER		
Drawing #	Drawing Type	Description
EE-0026	Part	Sample Container Cylinder
EE-0027	Part	Sample Container Base Plate
EE-0028	Part	Sample Container Top Fastening Ring
AYEE-0013	Assembly	Complete Sample Container

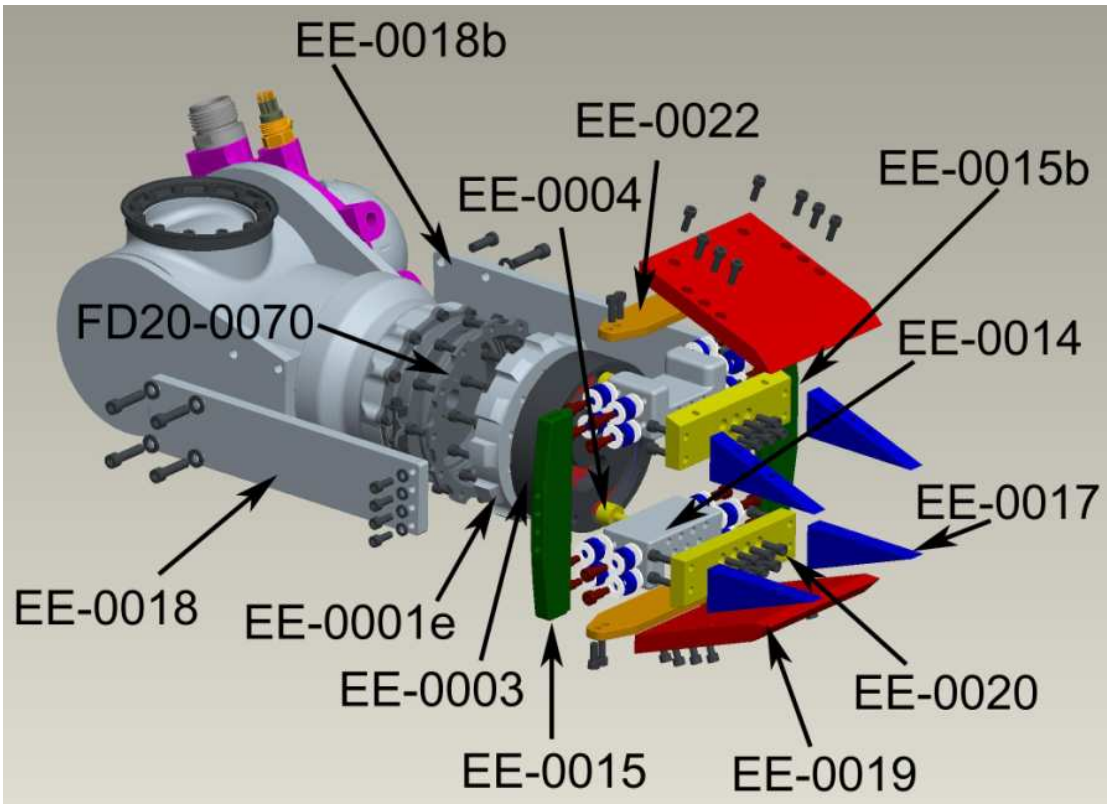
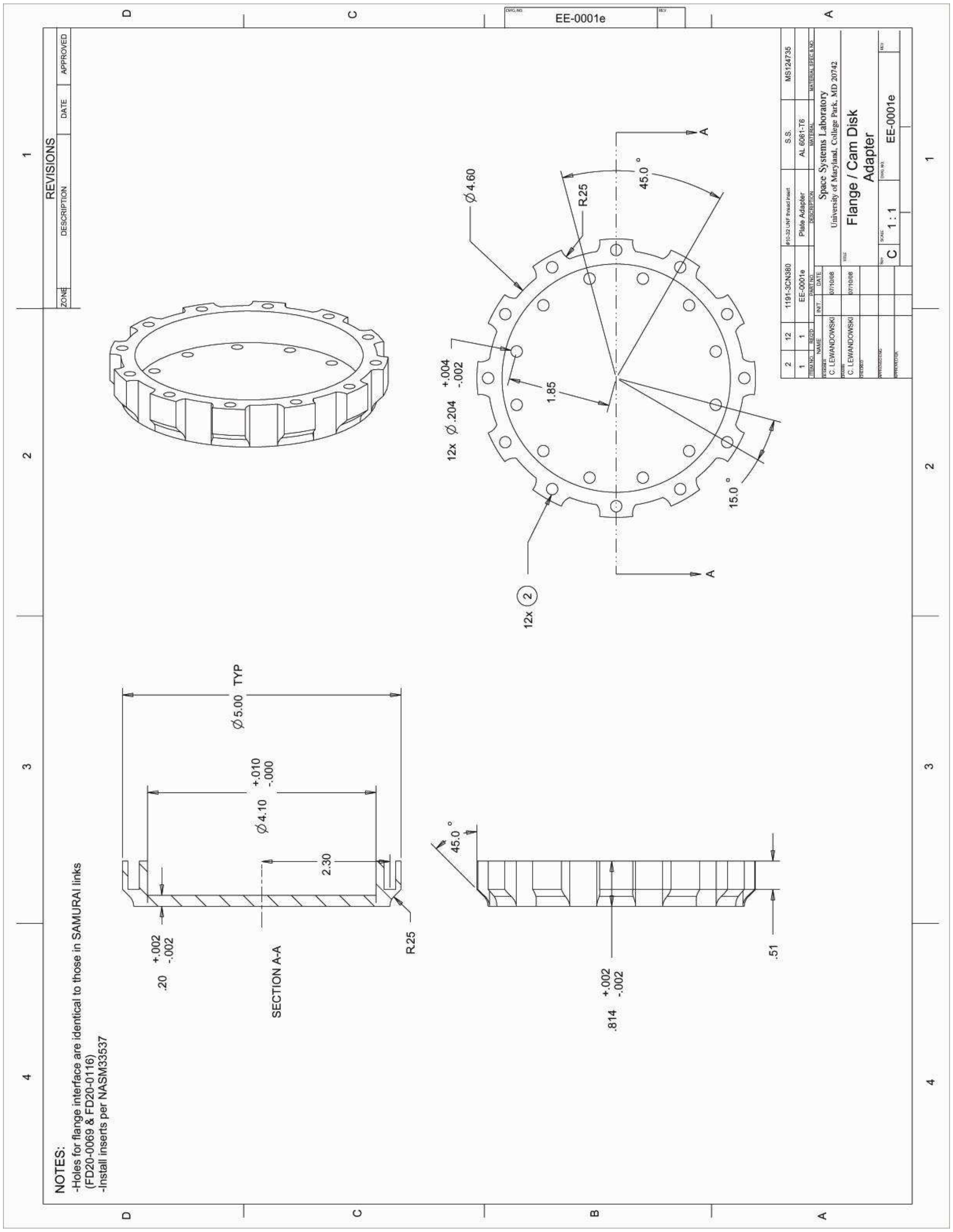


Figure A.1: Exploded view of end-effector CAD model with component numbers.


NOTES:
-Holes for flange interface are identical to those in SAMURAI links
(FD20-0069 & FD20-0116)
-Install inserts per NASM33537
REVISIONS
SECTION A-A
Ø5.00 TYP
Ø4.10
2.30
R.25
.20
.814
.51
45.0 °
Ø4.60
1.85
12x Ø.204
15.0 °
12x
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Flange / Cam Disk
Adapter
EE-0001e
C. LEWANDOWSKI
AL 6061-T6
Plate Adapter
1191-3CN380
MS124735
1 : 1

NOTES:

-Scripts for Groove Construction:
(t varies from 0 to 1)
Script for Spiral Groove 1:
h = 2.25; offset = .45; beta = 155;
r = (h/2)*(1 + cos(beta * t))
theta = t * beta
x = r * cos(theta - 90)
y = r * sin(theta - 90) + offset
z = 0

Script for Spiral Groove 2:
h = 2.25; offset = .45; beta = 155;
r = (h/2)*(1 + cos(beta * t))
theta = t * beta
x = -r * cos(theta - 90)
y = -r * sin(theta - 90) - offset
z = 0

Points Located in Groove 1:

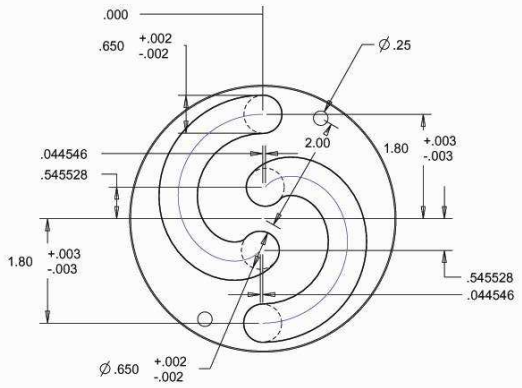
Pt.	X	Y	Z
1	0	-1.8000	0
2	0.5903	-1.6787	0
3	1.0760	-1.3408	0
4	1.3777	-0.8574	0
5	1.4596	-0.3261	0
6	1.3360	0.1538	0
7	1.0646	0.5057	0
8	0.7283	0.6937	0
9	0.4111	0.7273	0
10	0.1750	0.6549	0
11	0.0445	0.5455	0

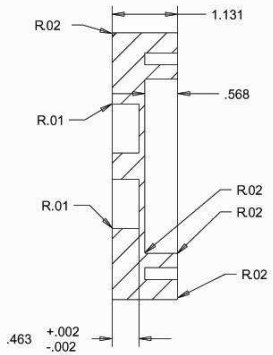
Points Located in Groove 2:

Pt.	X	Y	Z
1	0	1.8000	0
2	-0.5903	1.6787	0
3	-1.0760	1.3408	0
4	-1.3777	0.8574	0
5	-1.4596	0.3261	0
6	-1.3360	-0.1538	0
7	-1.0646	-0.5057	0
8	-0.7283	-0.6937	0
9	-0.4111	-0.7273	0
10	-0.1750	-0.6549	0
11	-0.0445	-0.5455	0

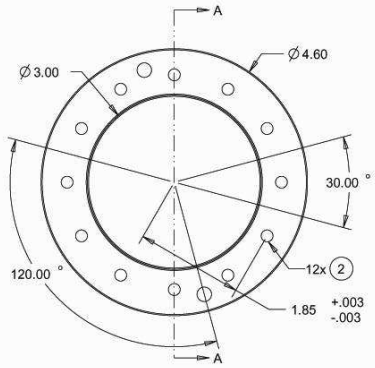
ADDITIONAL NOTES:

-Install inserts per NASM33537
-Dimensions ".545528" & ."044546" are included in top-left view only as a reference to the point tables





SECTION A-A



REVISIONS

ZONE	DESCRIPTION	DATE	APPROVED

ITEM NO.	REQ'D	PART NO.	DESCRIPTION	MATERIAL	MATERIAL SPEC & NO.
2	12	1191-3CN285	#10-32 UNF Broad Insert	S.S.	MS124695
1	1	EE-0003	Spiral Plate	TBD	

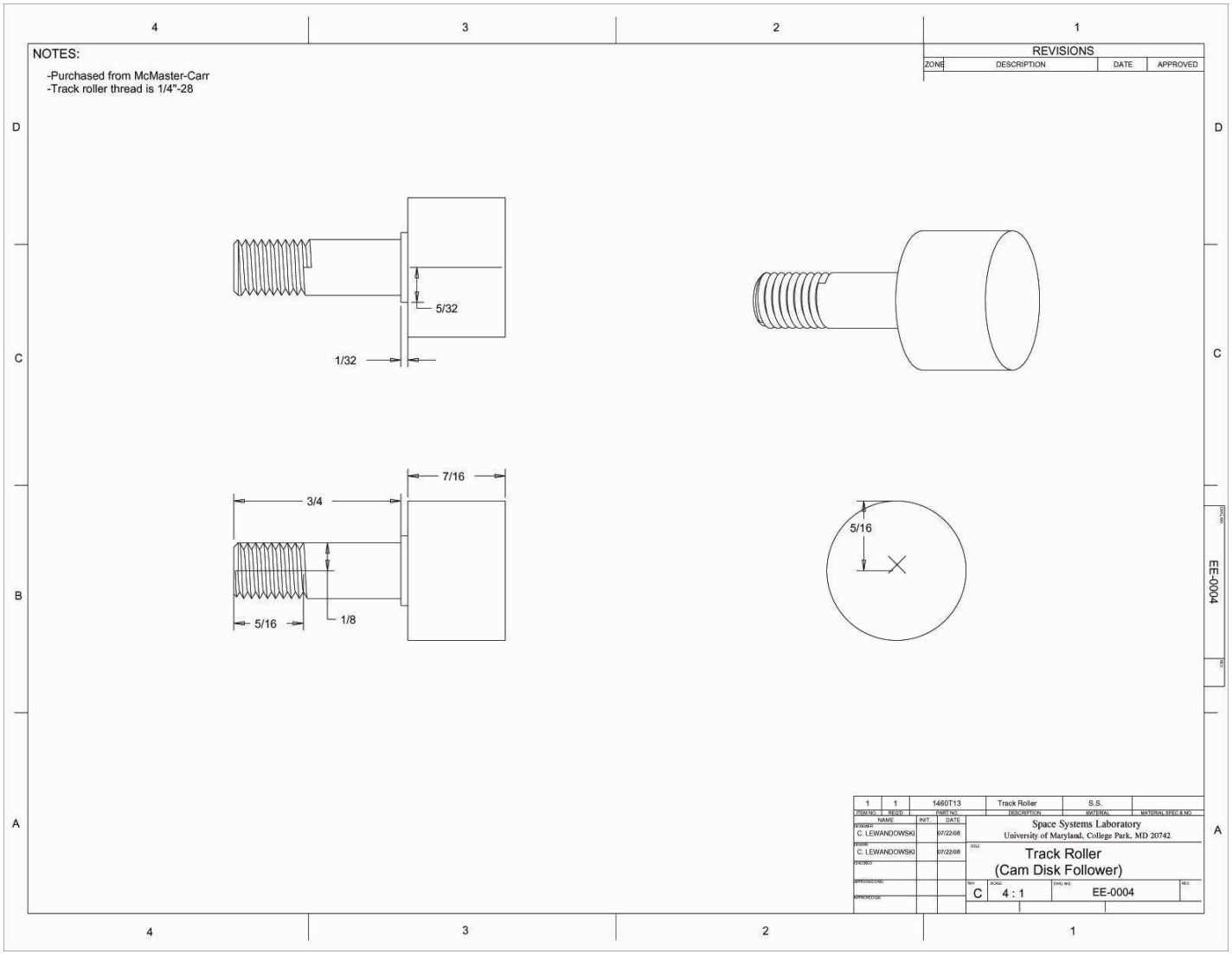
	NAME	INIT.	DATE
DESIGNED	C. LEWANDOWSKI		7/20/08
DRAWN	C. LEWANDOWSKI		7/20/08
CHECKED			
APPROVED [ENG]			
APPROVED [QA]			

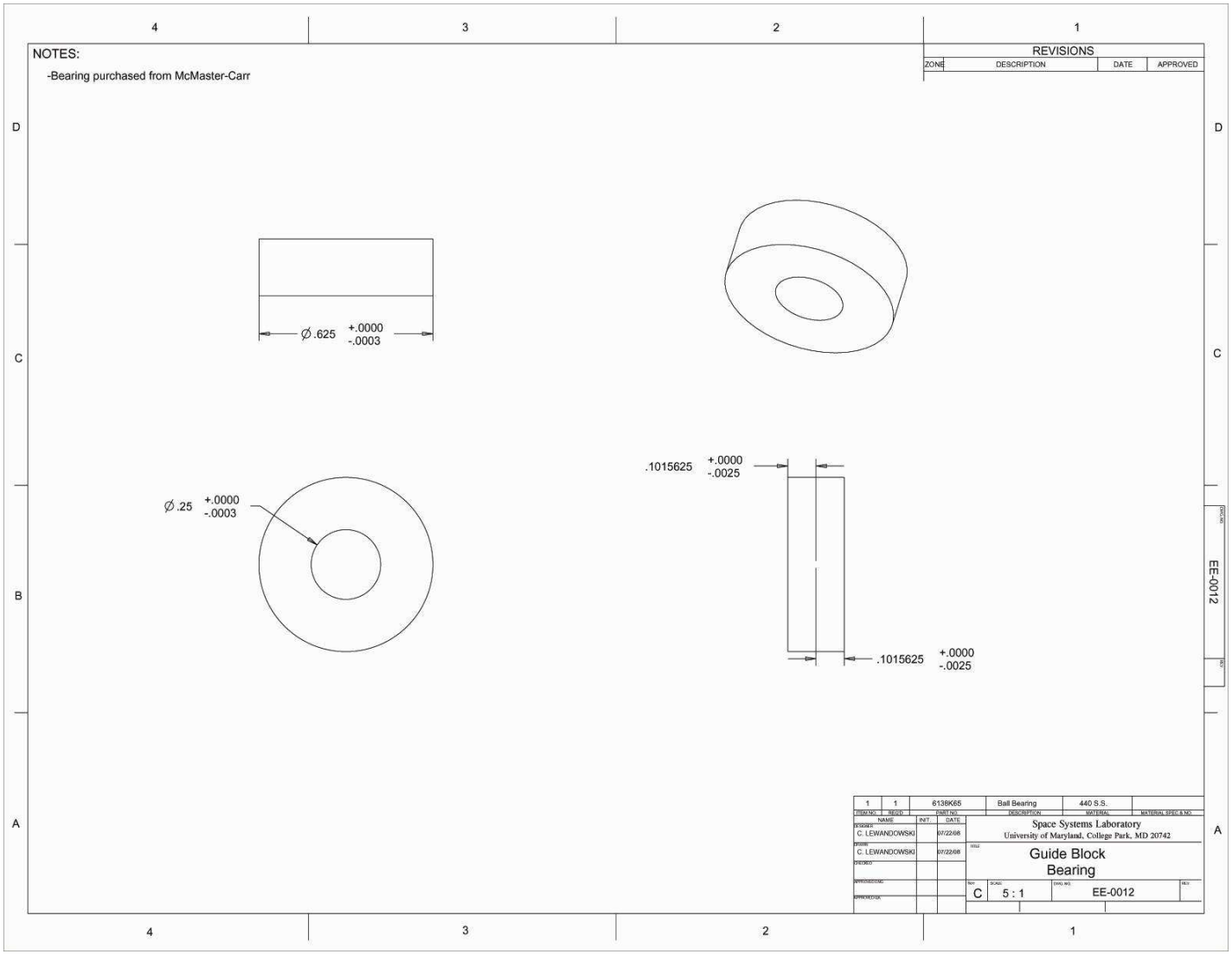
Space Systems Laboratory
University of Maryland, College Park, MD 20742

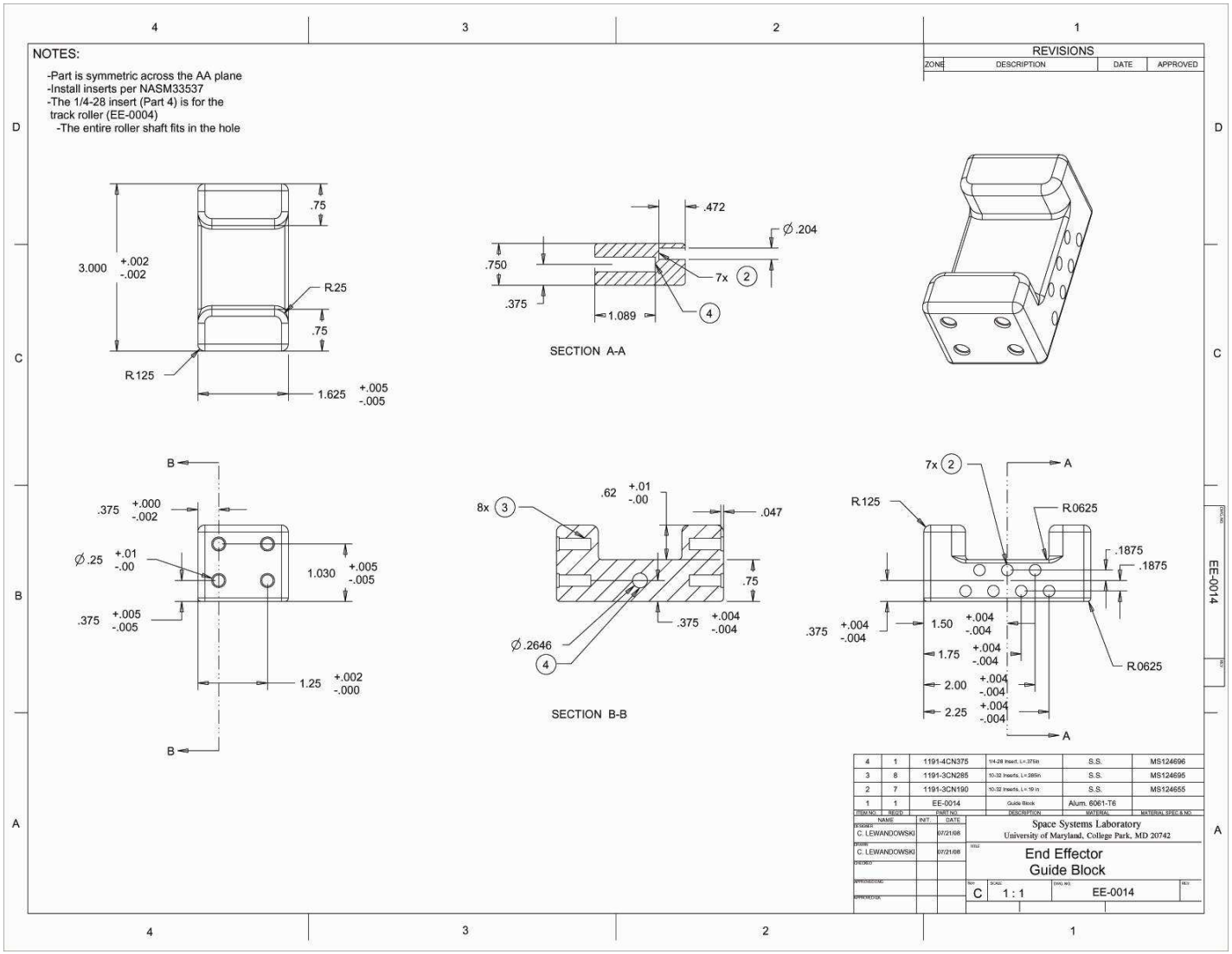
TITLE: Cam Disk/ Spiral Plate

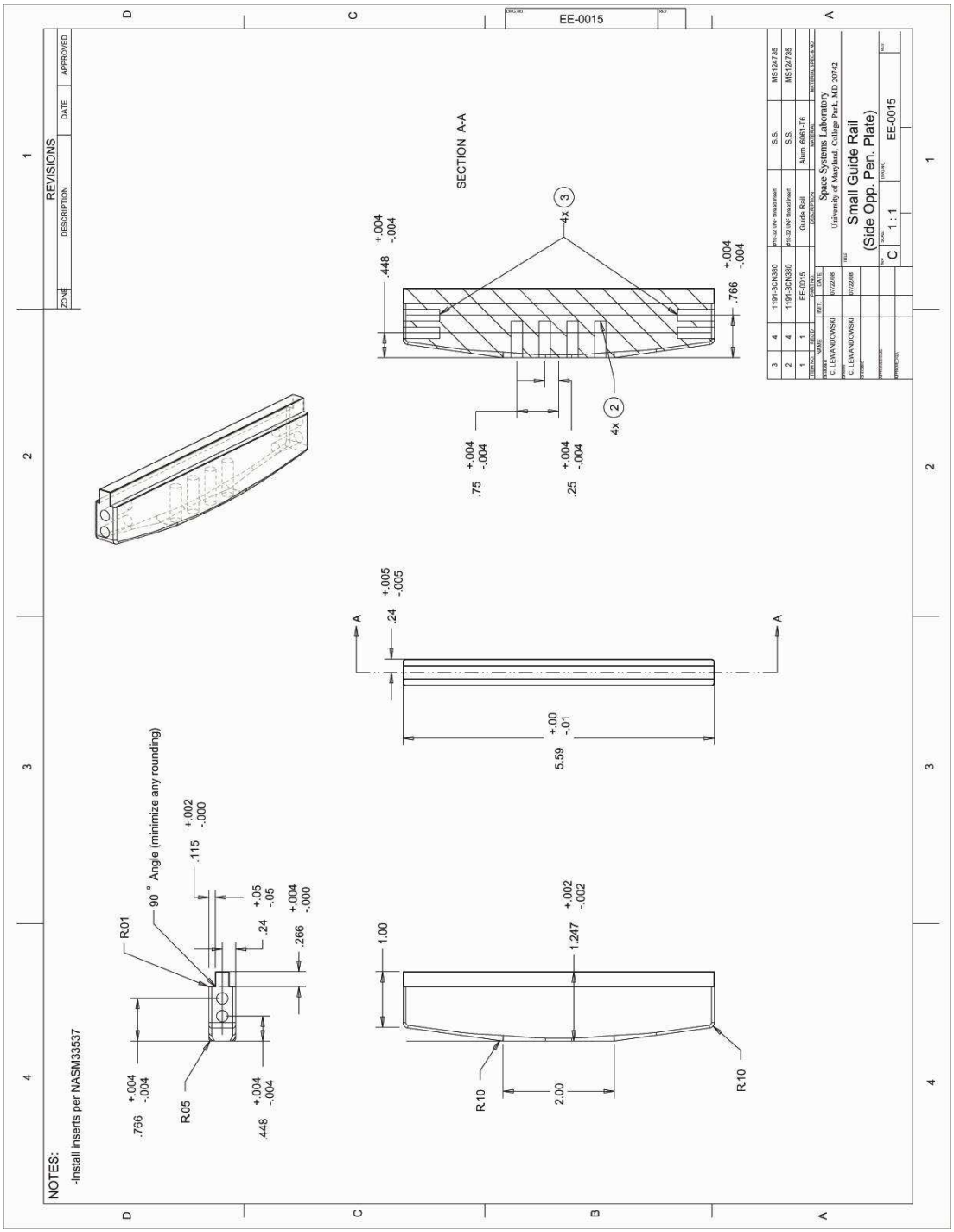
SIZE	SCALE	DWG. NO.	REV
C	.9 : 1	EE-0003	

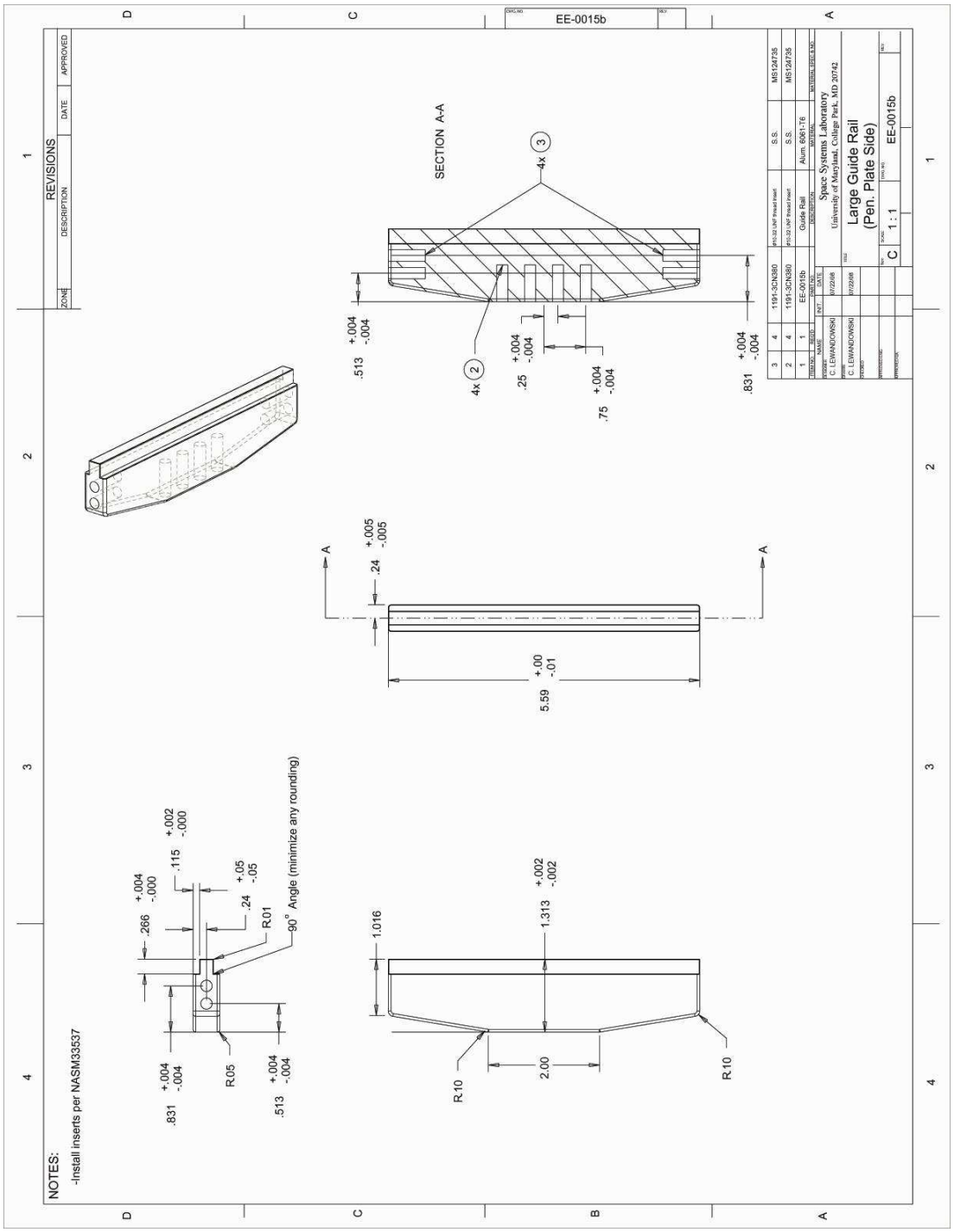
EE-0003

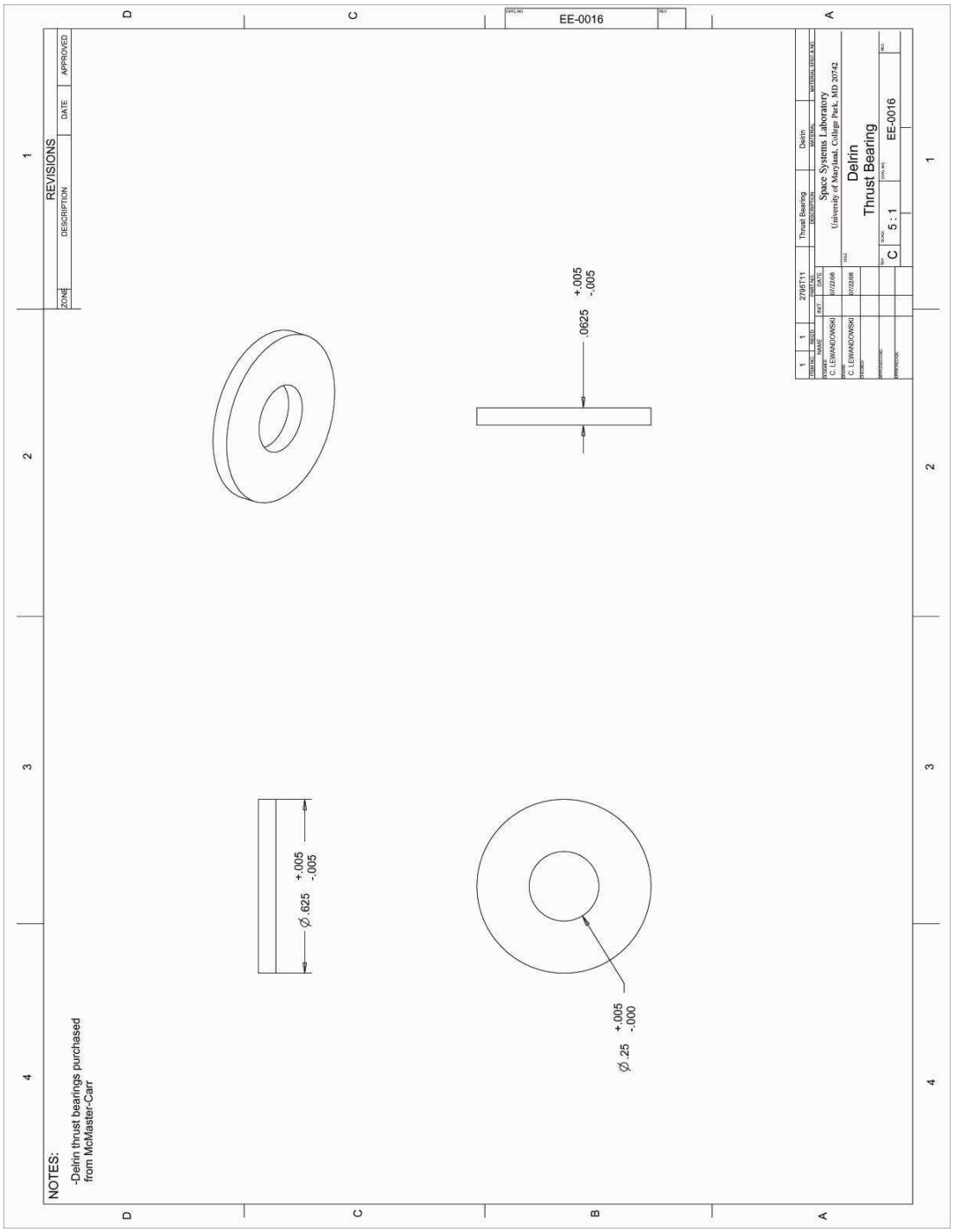

NOTES:
-Purchased from McMaster-Carr
-Track roller thread is 1/4"-28
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
5/32
1/32
7/16
3/4
5/16
1/8
5/16
EE-0004
1
1
1460T13
Track Roller
S.S.
ITEM NO.
REQ'D
PART NO.
DESCRIPTION
MATERIAL
MATERIAL SPEC & NO.
NAME
INIT.
DATE
C. LEWANDOWSKI
07/22/08
C. LEWANDOWSKI
07/22/08
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Track Roller
(Cam Disk Follower)
C
4 : 1
EE-0004

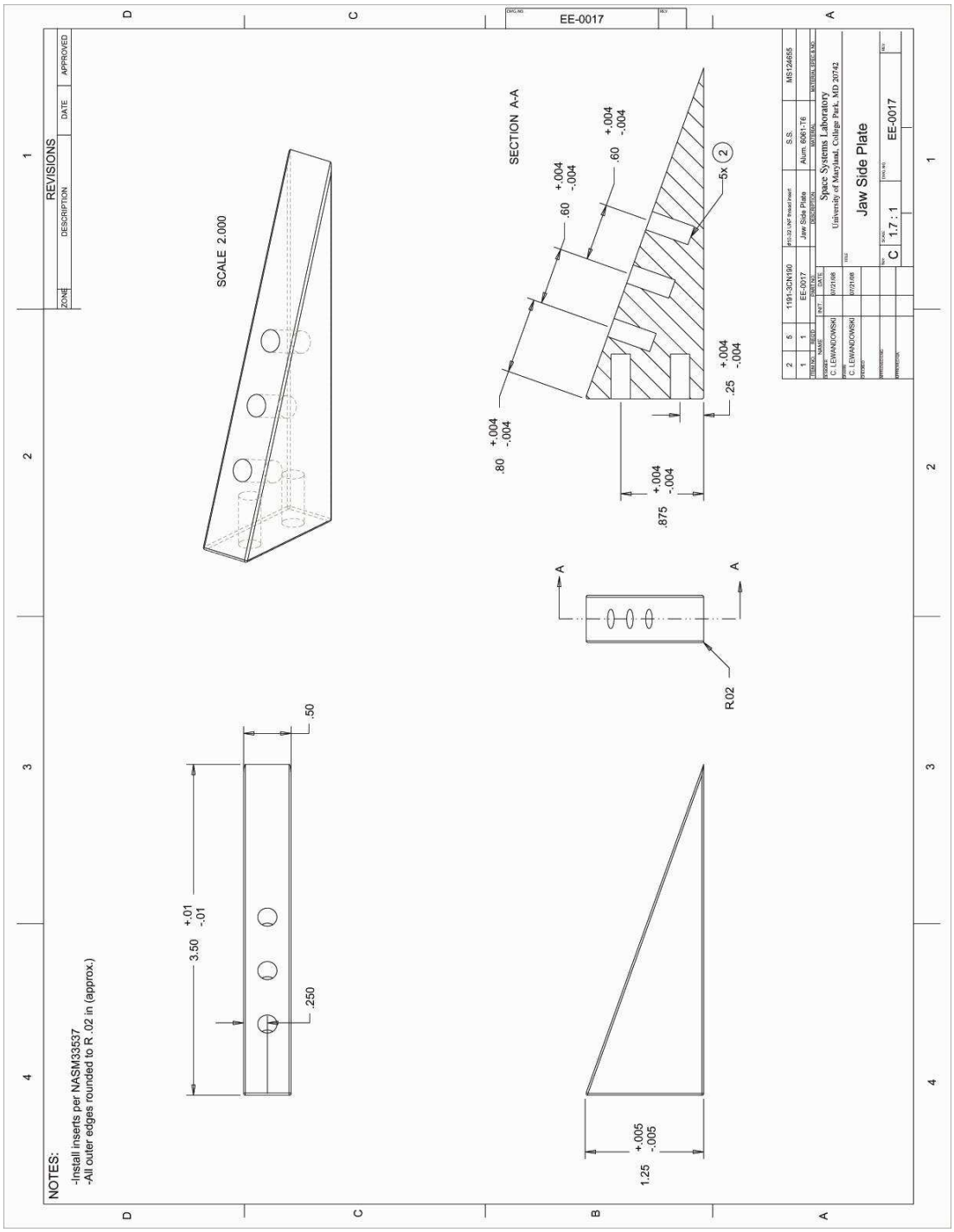

NOTES:
-Bearing purchased from McMaster-Carr
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
Ø .625 +.0000 -.0003
Ø .25 +.0000 -.0003
.1015625 +.0000 -.0025
.1015625 +.0000 -.0025
1
1
6138K65
Ball Bearing
440 S.S.
ITEM NO.
REQ'D
PART NO.
DESCRIPTION
MATERIAL
MATERIAL SPEC & NO.
NAME
INIT.
DATE
C. LEWANDOWSKI
07/22/08
C. LEWANDOWSKI
07/22/08
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Guide Block
Bearing
C
5 : 1
EE-0012
EE-0012

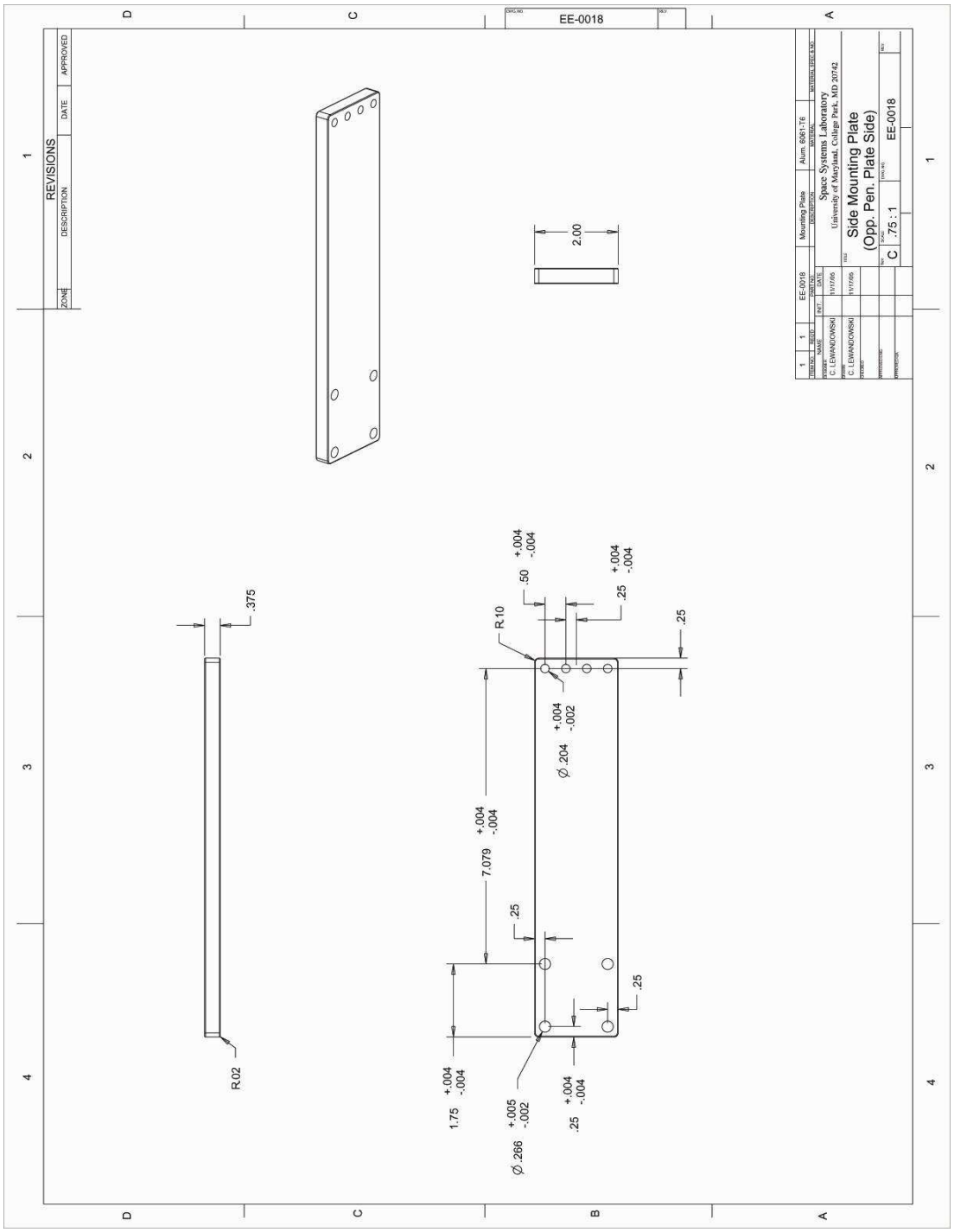

NOTES:
-Part is symmetric across the AA plane
-Install inserts per NASM33537
-The 1/4-28 insert (Part 4) is for the track roller (EE-0004)
-The entire roller shaft fits in the hole
REVISIONS
ZONE DESCRIPTION DATE APPROVED
3.000 +.002 -.002
.75
R.25
.75
R.125
1.625 +.005 -.005
.472
Ø.204
.750
7x 2
.375
1.089
4
SECTION A-A
B
.375 +.000 -.002
Ø.25 +.01 -.00
1.030 +.005 -.005
.375 +.005 -.005
1.25 +.002 -.000
B
8x 3
.62 +.01 -.00
.047
.75
.375 +.004 -.004
Ø.2646
4
SECTION B-B
7x 2
A
R.125
R.0625
.1875
.1875
.375 +.004 -.004
1.50 +.004 -.004
1.75 +.004 -.004
2.00 +.004 -.004
2.25 +.004 -.004
R.0625
A
4 1 1191-4CN375 1/4-28 Insert, L=.375in S.S. MS124696
3 8 1191-3CN285 10-32 Inserts, L=.285in S.S. MS124695
2 7 1191-3CN190 10-32 Inserts, L= 10 in S.S. MS124555
1 1 EE-0014 Guide Block Alum. 6061-T6
ITEM NO. REQ'D PART NO. DESCRIPTION MATERIAL MATERIAL SPEC & NO.
DESIGNED C. LEWANDOWSKI 07/21/08
DRAWN C. LEWANDOWSKI 07/21/08
CHECKED
APPROVED QA
APPROVED MFG
Space Systems Laboratory
University of Maryland, College Park, MD 20742
TITLE
End Effector
Guide Block
SIZE C SCALE 1 : 1 DWG. NO. EE-0014 REV
EE-0014

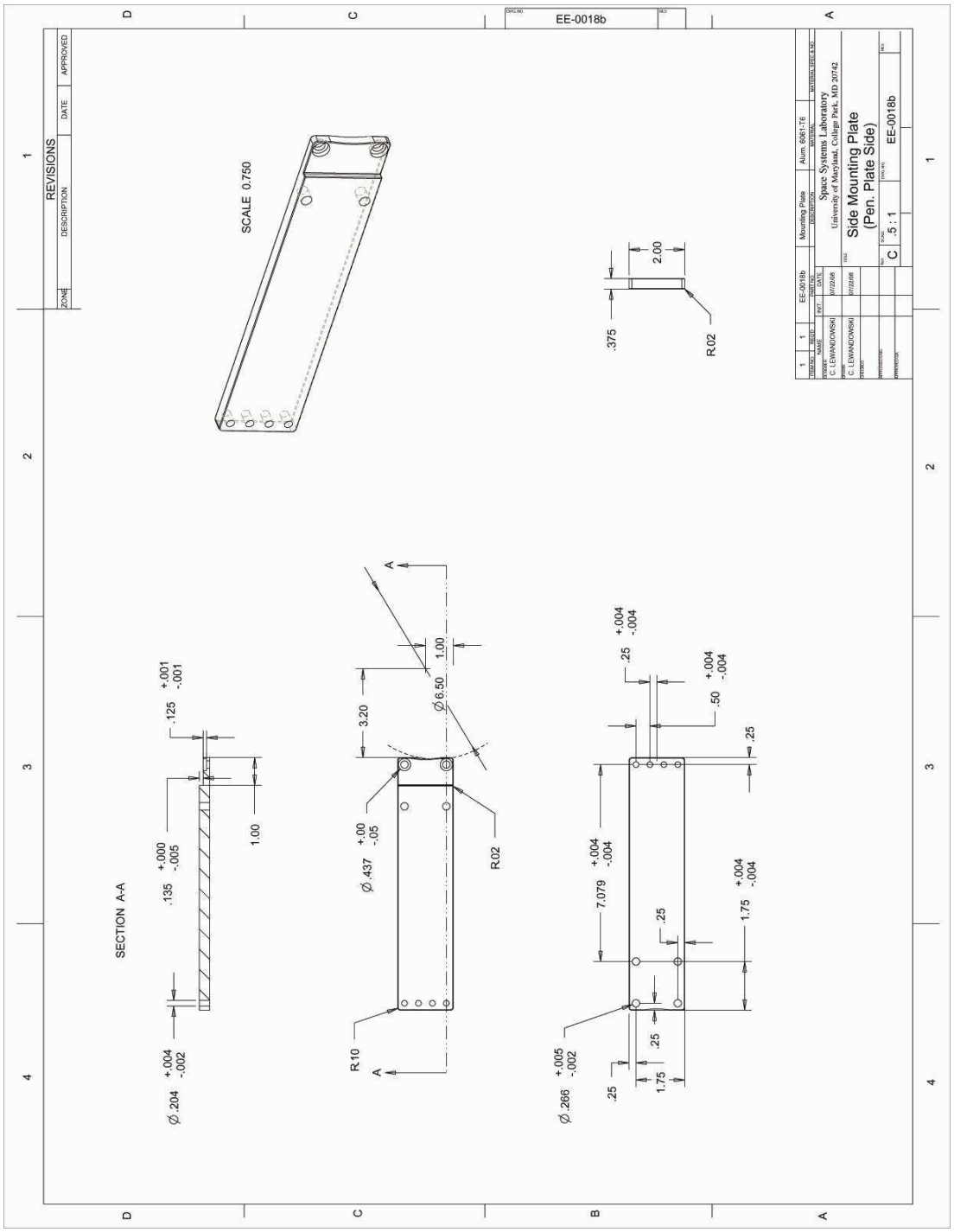

EE-0015
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
SECTION A-A
4x
4x
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Small Guide Rail
(Side Opp. Pen. Plate)
EE-0015
C
1 : 1
Guide Rail
Alum. 6061-T6
1191-3CN380
EE-0015
MS124738
NOTES:
-Install inserts per NASM33537
90 ° Angle (minimize any rounding)
R.01
R.05
R.10
R.10
5.59
2.00
1.247
1.00
.766
.448
.75
.25
.24
.115
.266

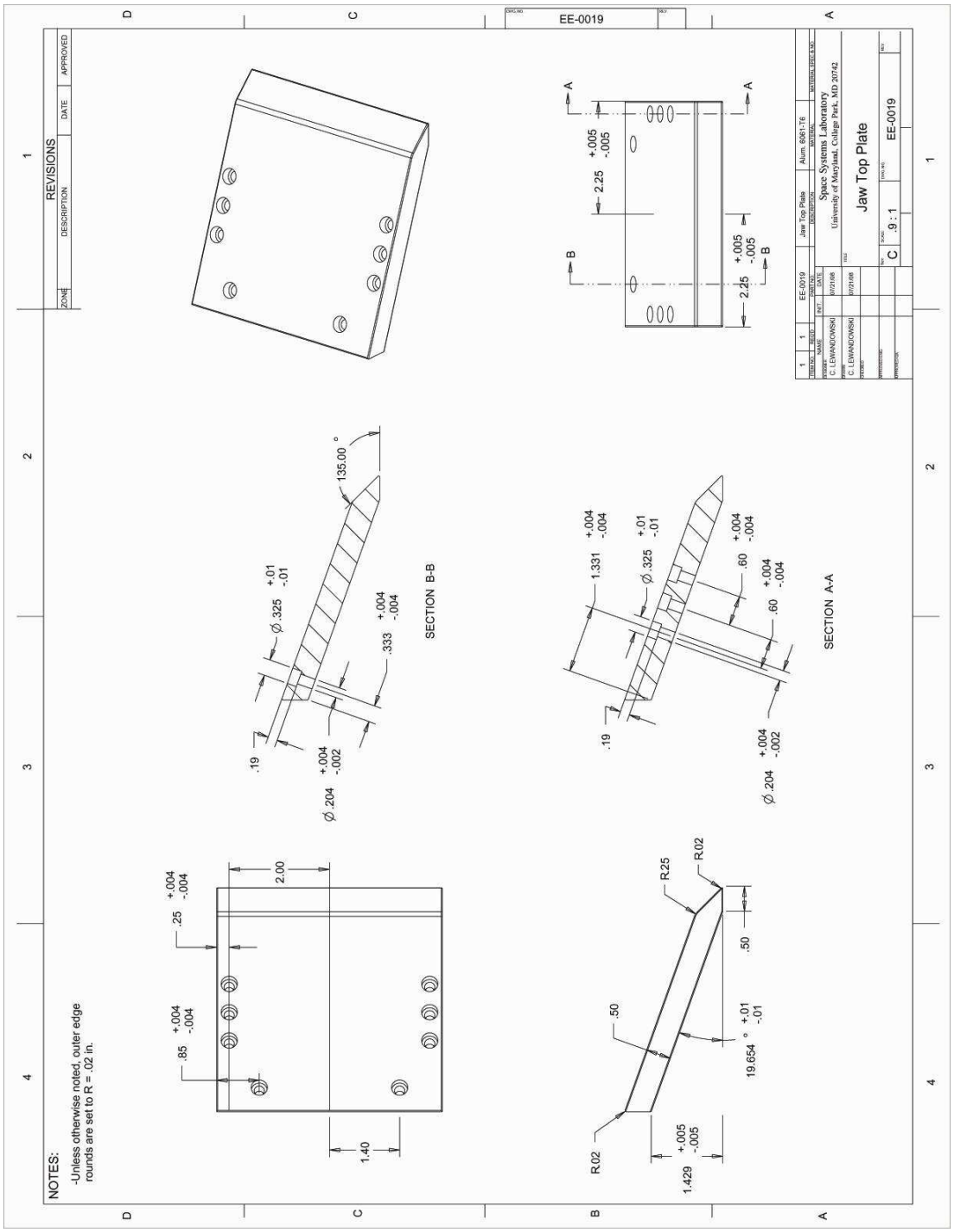

EE-0015b
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
SECTION A-A
4x 3
4x 2
.513 +.004 -.004
.25 +.004 -.004
.75 +.004 -.004
.831 +.004 -.004
1.016
1.313 +.002 -.002
2.00
R.10
R.10
5.59 +.00 -.01
.24 +.005 -.005
.266 +.004 -.000
.115 +.002 -.000
.24 +.05 -.05
R.01
90° Angle (minimize any rounding)
.831 +.004 -.004
R.05
.513 +.004 -.004
NOTES:
-Install inserts per NASM33537
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Large Guide Rail
(Pen. Plate Side)
SCALE: 1 : 1
SIZE: C
DWG NO: EE-0015b
REV:
1191-3CN380
MS124738
Alum. 6061-T6
EE-0015b
Guide Rail
C. LEWANDOWSKI
8/7/2018

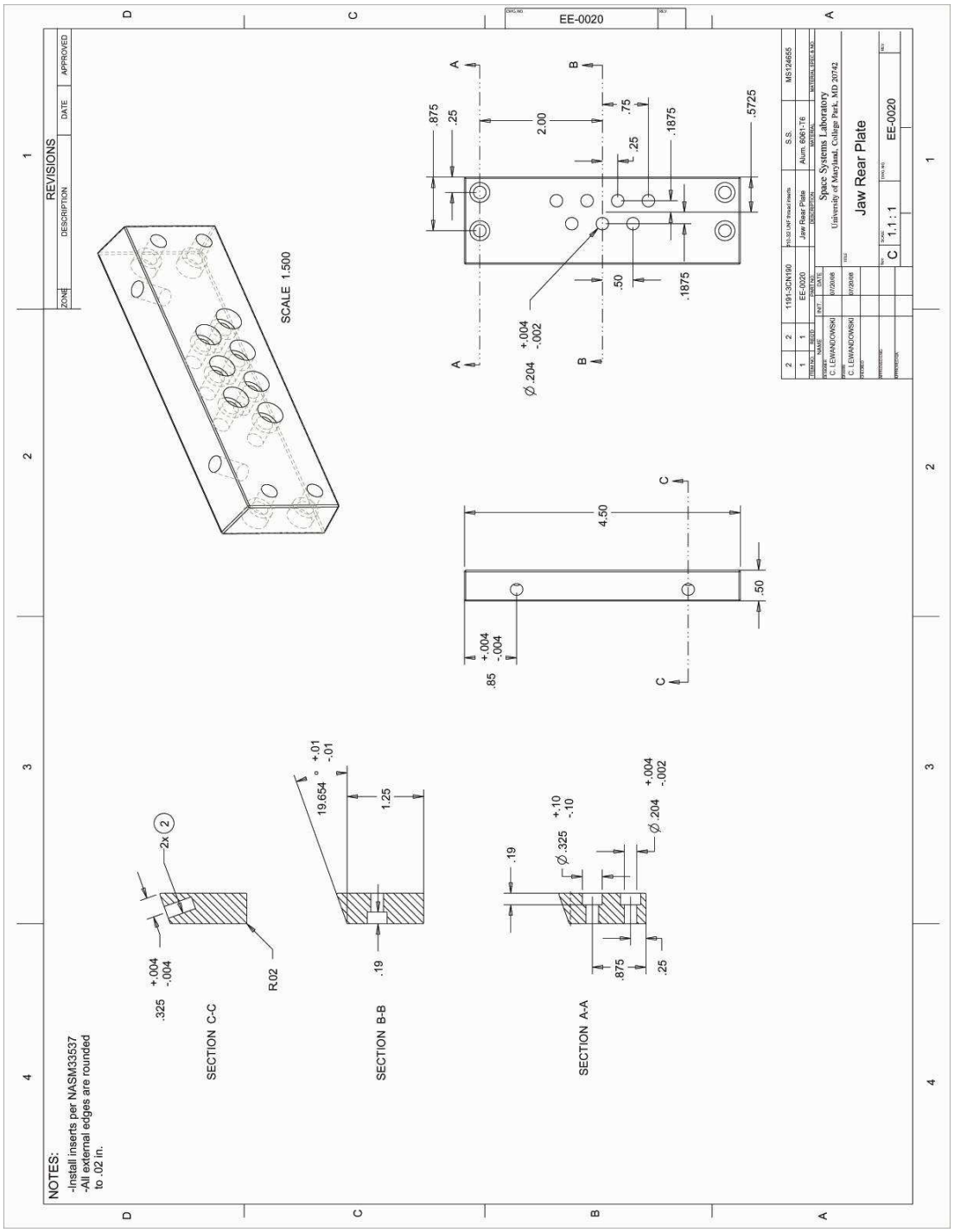

EE-0016
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
NOTES:
-Delrin thrust bearings purchased from McMaster-Carr
+.005 -.005
.0625
Ø .625 +.005 -.005
Ø .25 +.005 -.000
Thrust Bearing
Delrin
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Delrin
Thrust Bearing
C
5 : 1
EE-0016

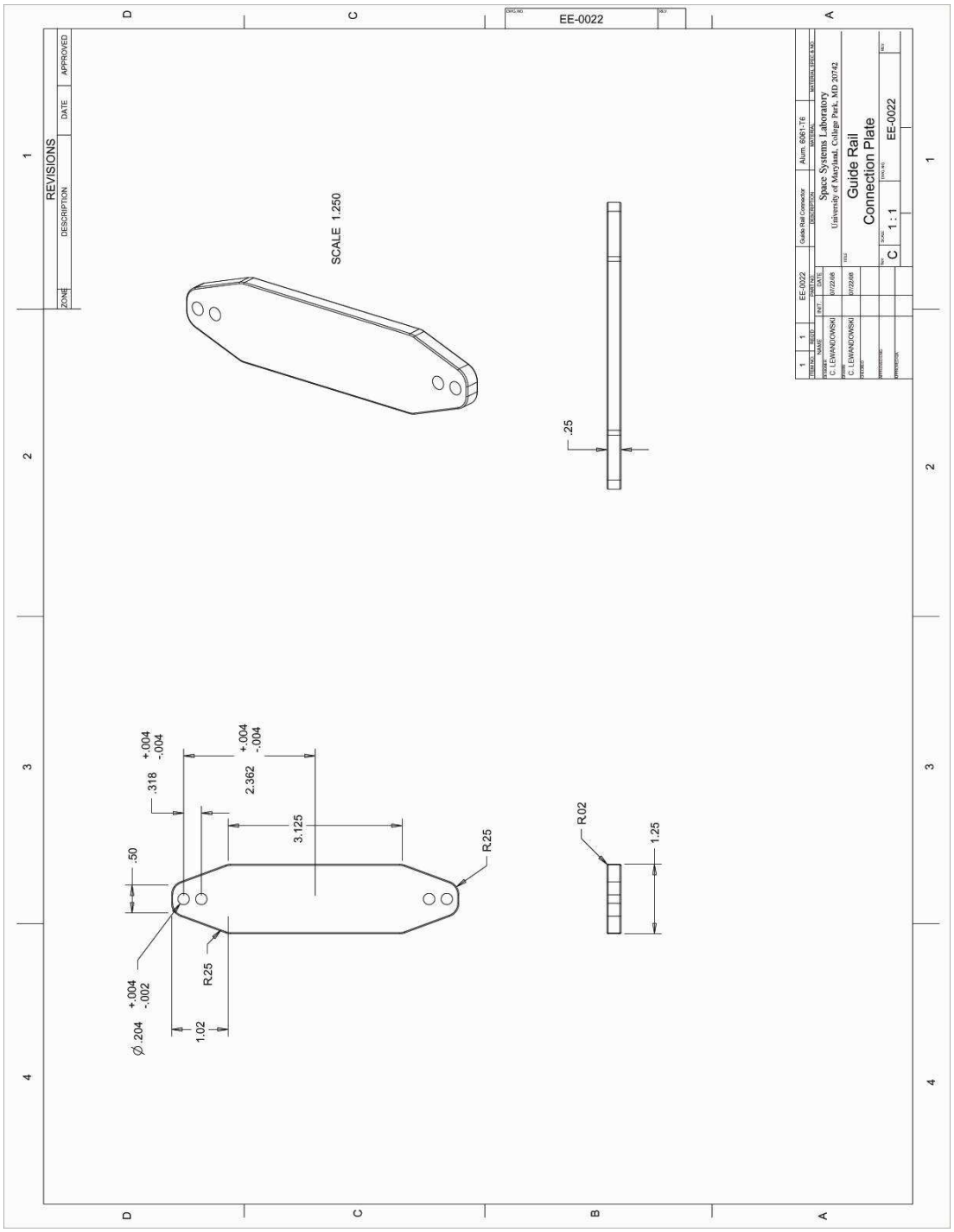

EE-0017
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
SCALE 2.000
SECTION A-A
5x
2
.60
.80
.875
.25
.50
3.50
.250
1.25
R.02
NOTES:
-Install inserts per NASM33537
-All outer edges rounded to R .02 in (approx.)
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Jaw Side Plate
EE-0017
C
1.7 : 1
MATERIAL
Alum. 6061-T6
MS124855
1191-3CN190
EE-0017
C. LEWANDOWSKI
8/7/2008
8/7/2008

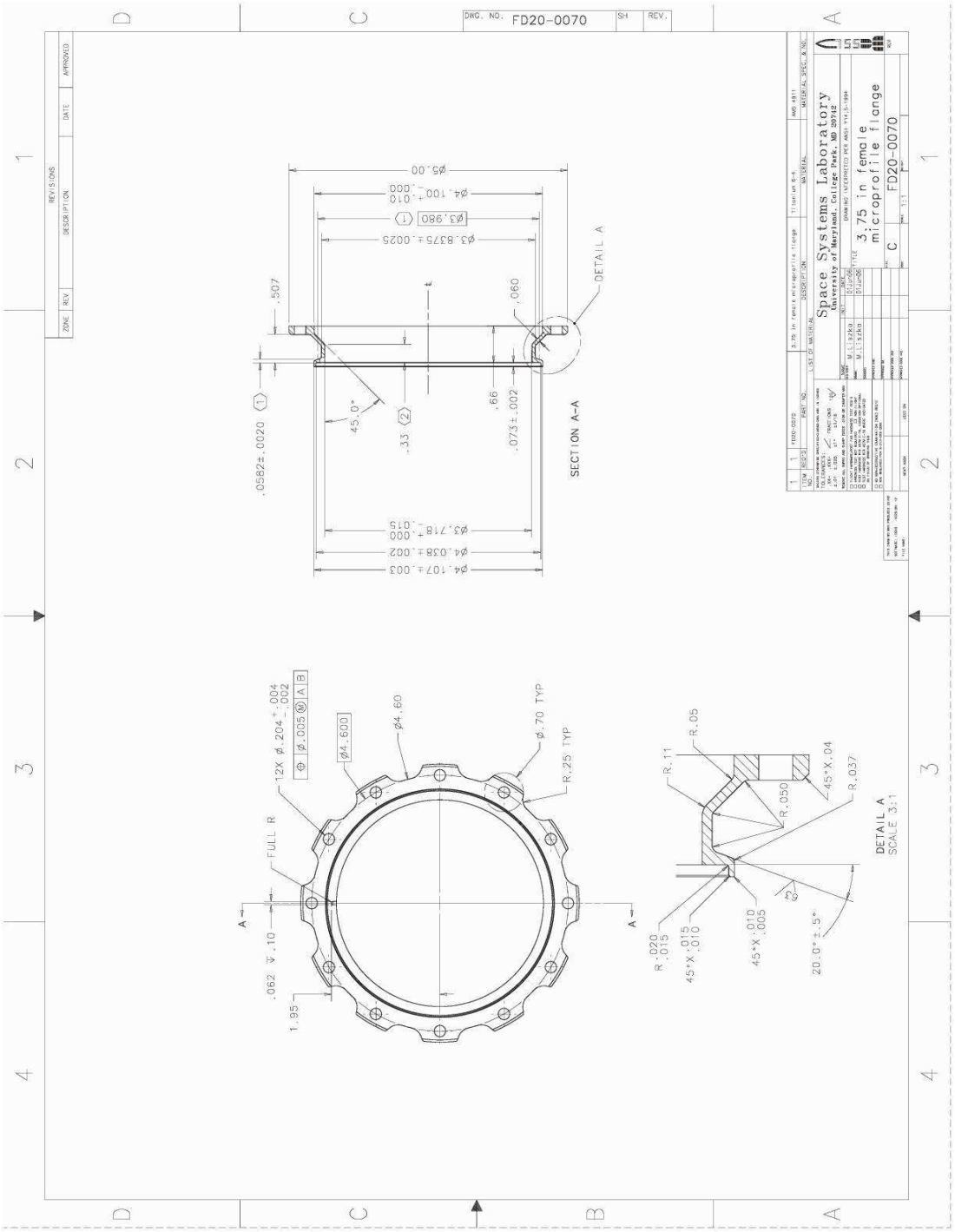

EE-0018
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
Mounting Plate
Alum. 6061-T6
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Side Mounting Plate
(Opp. Pen. Plate Side)
EE-0018
C
.75 : 1
2.00
.375
R.02
R.10
.50
.25
.25
7.079
.25
.25
1.75
.25
Ø .204
Ø .266

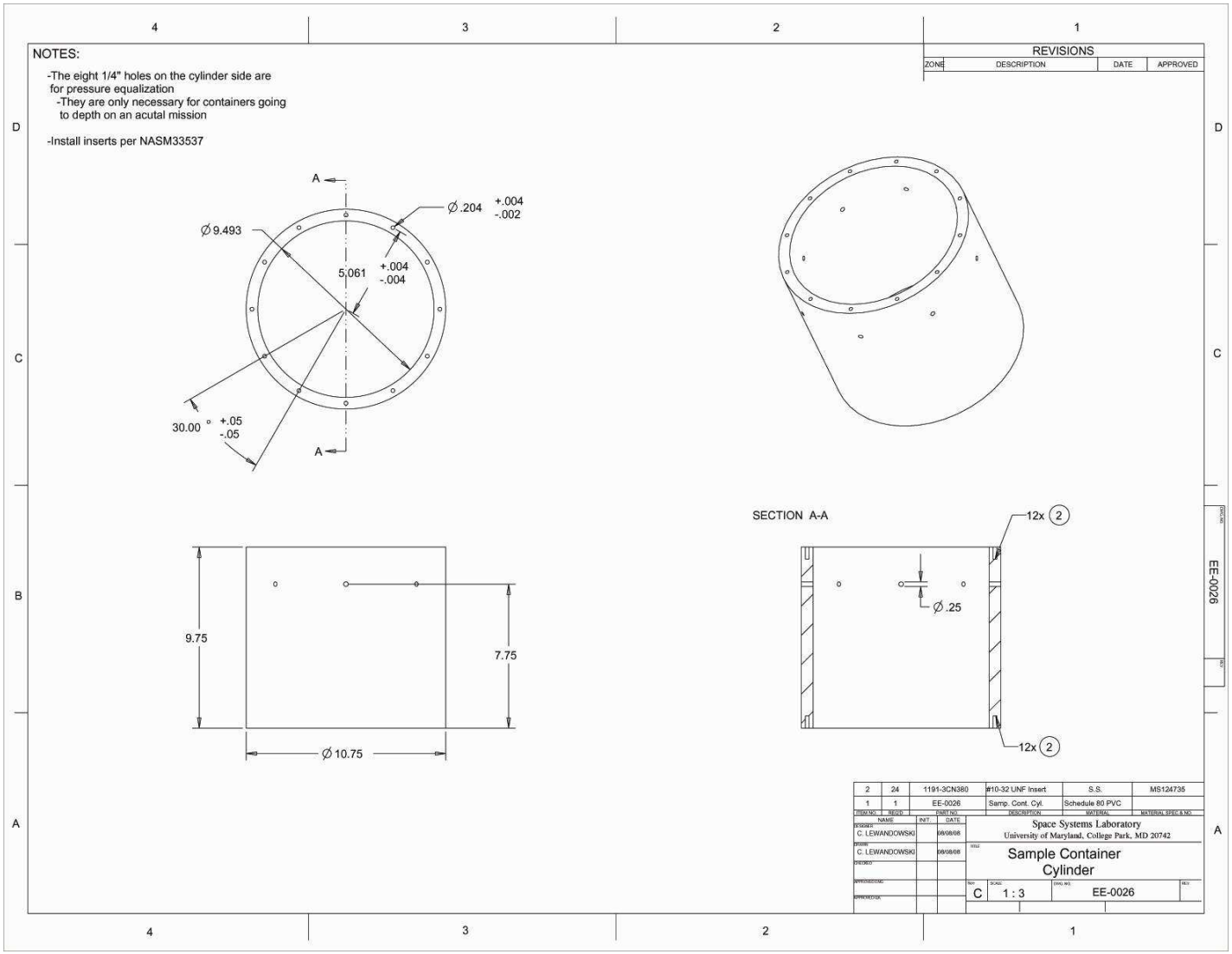

EE-0018b
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
SCALE 0.750
SECTION A-A
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Side Mounting Plate
(Pen. Plate Side)
Mounting Plate
Alum. 6061-T6
EE-0018b
C
.5 : 1
1 OF 1
R.02
2.00
.375
R10
R02
1.00
3.20
6.50
.437
.204
.266
.135
.125
7.079
1.75
.50
.25

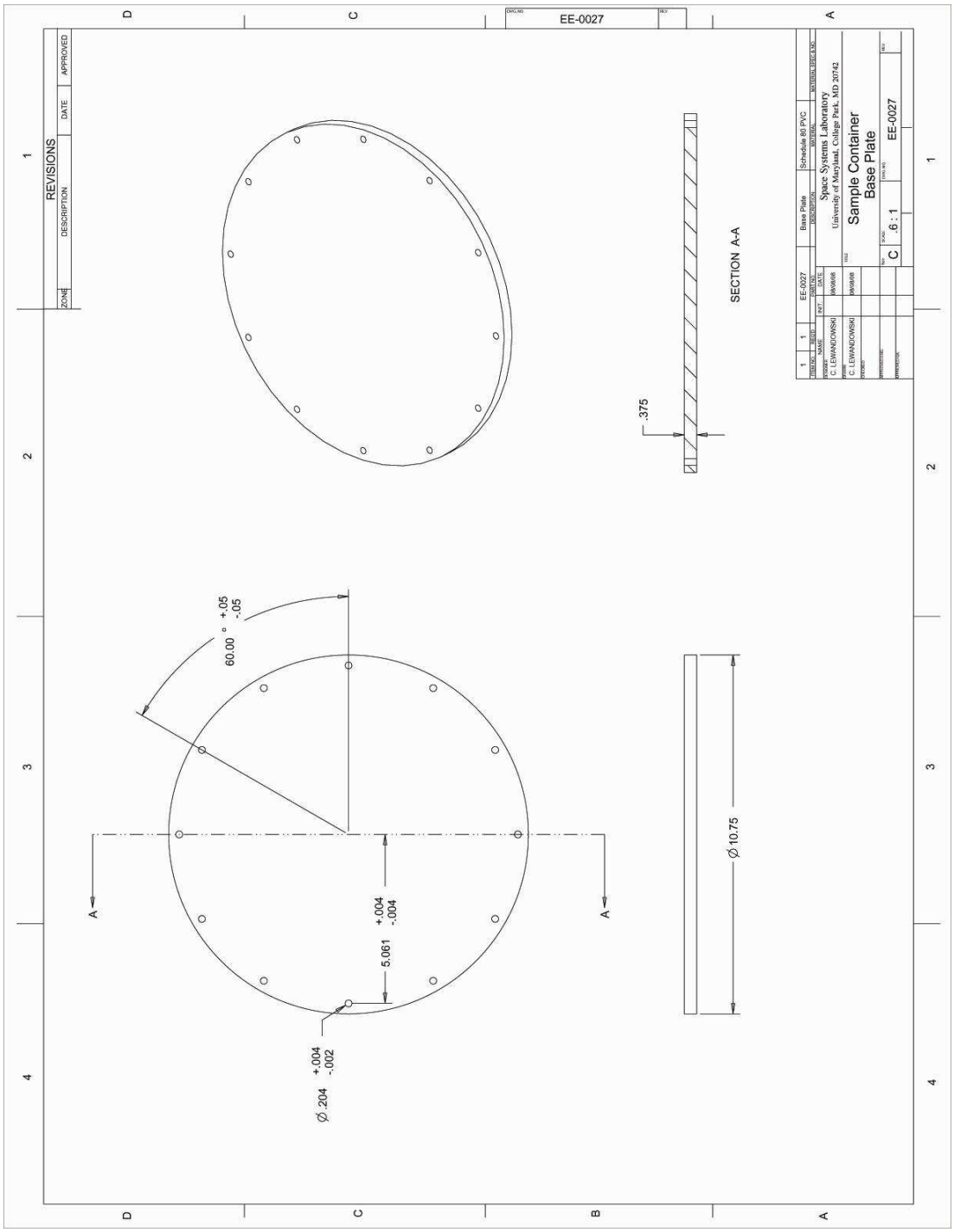

EE-0019
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
NOTES:
-Unless otherwise noted, outer edge rounds are set to R = .02 in.
SECTION B-B
SECTION A-A
Jaw Top Plate
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Alum. 6061-T6
C
.9 : 1
EE-0019


EE-0020
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
SCALE 1.500
NOTES:
-Install inserts per NASM33537
-All external edges are rounded to .02 in.
SECTION C-C
SECTION B-B
SECTION A-A
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Jaw Rear Plate
EE-0020
SCALE 1.1 : 1
SIZE C
MATERIAL
Alum. 6061-T6
S.S.
MS124655
1191-3CN190
EE-0020
Jaw Rear Plate


EE-0022
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
SCALE 1.250
.25
R.02
1.25
.318 +.004 -.004
2.362 +.004 -.004
3.125
.50
R25
R25
Ø.204 +.004 -.002
1.02
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Guide Rail
Connection Plate
EE-0022
C
1 : 1
Alum. 6061-T6
Guide Rail Connector


DWG. NO. FD20-0070
REV.
Space Systems Laboratory
University of Maryland, College Park, MD 20742
3.75 in female microprofile flange
FD20-0070
SECTION A-A
DETAIL A
DETAIL A
SCALE 3:1


NOTES:
-The eight 1/4" holes on the cylinder side are for pressure equalization
-They are only necessary for containers going to depth on an acutal mission
-Install inserts per NASM33537
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
Ø 9.493
Ø .204 +.004 -.002
5.061 +.004 -.004
30.00 ° +.05 -.05
A
A
9.75
7.75
Ø 10.75
SECTION A-A
12x 2
Ø .25
12x 2
2 24 1191-3CN380 #10-32 UNF Insert S.S. MS124735
1 1 EE-0026 Samp. Cont. Cyl. Schedule 80 PVC
ITEM NO. REQ'D PART NO. DESCRIPTION MATERIAL MATERIAL SPEC & NO.
NAME INIT. DATE
DESIGNED C. LEWANDOWSKI 09/08/08
DRAWN C. LEWANDOWSKI 09/08/08
CHECKED
APPROVED (QA)
APPROVED (IA)
Space Systems Laboratory
University of Maryland, College Park, MD 20742
TITLE
Sample Container
Cylinder
SIZE C
SCALE 1 : 3
DWG. NO. EE-0026
REV
EE-0026


EE-0027
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
SECTION A-A
.375
60.00 ° +.05 -.05
+.004 -.004
5.061
+.004 -.002
Ø .204
Ø 10.75
A
A
Base Plate
Schedule 80 PVC
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Sample Container
Base Plate
EE-0027
C
.6 : 1

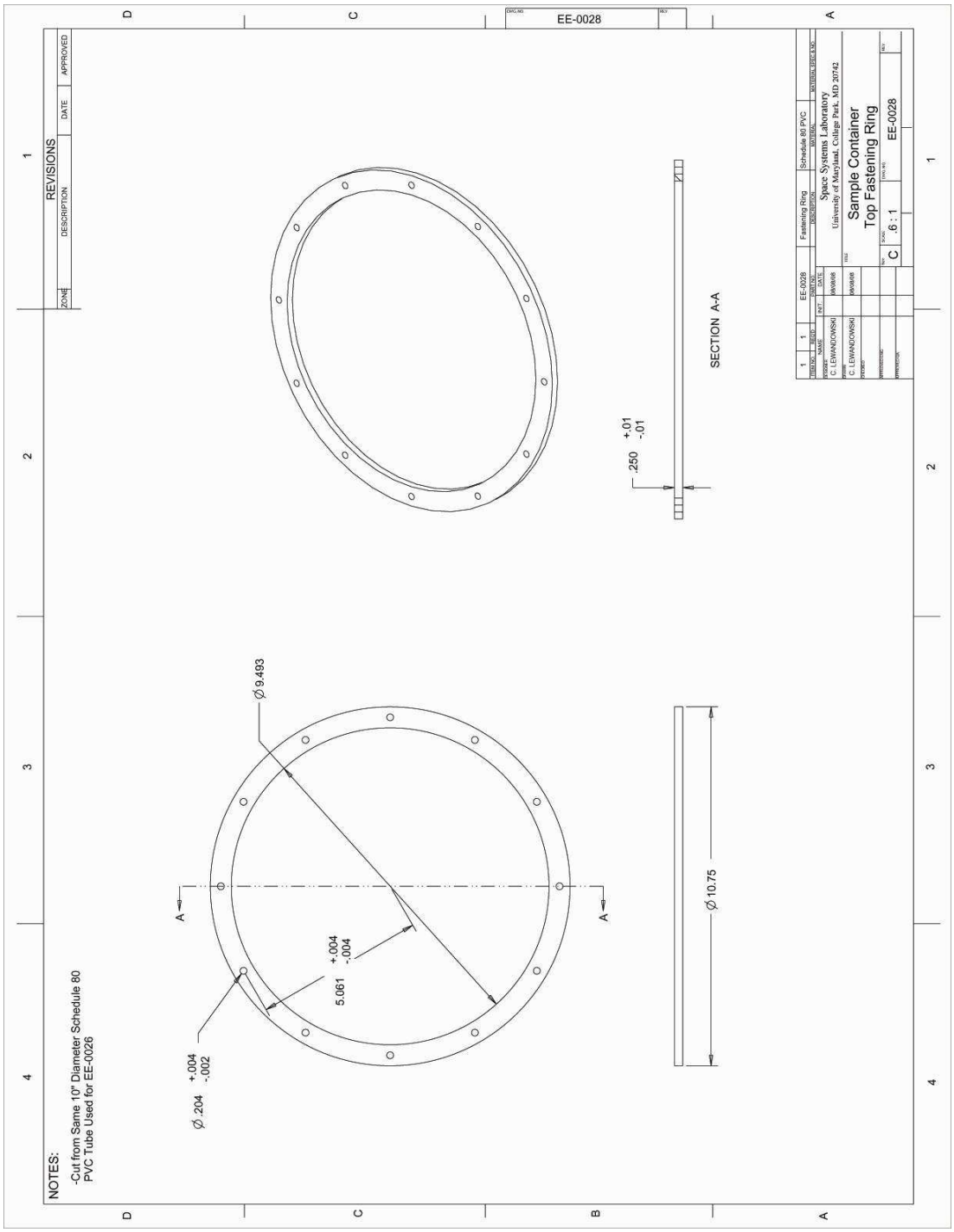

EE-0028
REVISIONS
ZONE
DESCRIPTION
DATE
APPROVED
NOTES:
-Cut from Same 10" Diameter Schedule 80
PVC Tube Used for EE-0026
SECTION A-A
Ø9.493
Ø10.75
5.061 +.004 -.004
Ø.204 +.004 -.002
.250 +.01 -.01
Fastening Ring
Schedule 80 PVC
Space Systems Laboratory
University of Maryland, College Park, MD 20742
Sample Container
Top Fastening Ring
EE-0028
SCALE: .6 : 1
SIZE: C

Table A.2: End-effector fasteners.

FASTENERS

Item	Screw Description	Quantity	Thread	Head	Length (in)
1	Screws Connecting Flange to Al Adapter	12	10-32	SHCS	0.375
2	Screws Connecting Al Adapter to Cam Disk	12	10-32	SHCS	0.5
3	Track Rollers to Join Disk to Guide Blocks	2	1/4-28	Roller	0.3125
4	Shoulder Screws to Hold Bearings to Guide Blocks	16	10-32	SHCS	.25
5	Jaw Screws to Connect Rear Plate to Guide Blks.	14	10-32	SHCS	0.5
6	Screws Connecting Jaw Rear Plate to Side Plate	8	10-32	SHCS	0.5
7	Screws Connecting Jaw Top to Side & Rear Plates	16	10-32	SHCS	0.5
8	Screws Bolting Connection Plate to Guide Rails	8	10-32	SHCS	0.5
9	Screws Bolting Side Mounts to Guide Rails	8	10-32	SHCS	0.5
10	Screws Connecting Side Mounts to Wrist Joint	8	1/4-20	SHCS	1
	Total Screw Quantity:	**104**			

Prototype

Item	Material	Part #	Cost	Ind. Cost ($)	Tot. Cost ($)
1	18-8 Stainless	92196A267	$8.94 per 100	0.0894	1.0728
2	18-8 Stainless	92196A269	$9.95 per 100	0.0995	1.194
3	Steel	1460T13	$13.62 Each	13.62	27.24
4	416 Stainless	93985A535	$2.81 Each	2.81	44.96
5	18-8 Stainless	92196A269	$9.95 per 100	0.0995	1.393
6	18-8 Stainless	92196A269	$9.95 per 100	0.0995	0.796
7	18-8 Stainless	92196A269	$9.95 per 100	0.0995	1.592
8	18-8 Stainless	92196A269	$9.95 per 100	0.0995	0.796
9	18-8 Stainless	92196A269	$9.95 per 100	0.0995	0.796
10	18-8 Stainless	92196A542	$12.83 per 50	0.2566	2.0528
				Total ($):	**81.89**

Final Design

Item	Material	Part #	Cost	Ind. Cost ($)	Tot. Cost ($)
1	A286 Super Alloy	92423A502	$2.07 Each	2.07	24.84
2	A286 Super Alloy	92423A505	$2.07 Each	2.07	24.84
3	440C Stainless	8043K74	$61.20 Each	61.2	122.4
4	416 Stainless	93985A535	$2.81 Each	2.81	44.96
5	A286 Super Alloy	92423A505	$2.07 Each	2.07	28.98
6	A286 Super Alloy	92423A505	$2.07 Each	2.07	16.56
7	A286 Super Alloy	92423A505	$2.07 Each	2.07	33.12
8	A286 Super Alloy	92423A505	$2.07 Each	2.07	16.56
9	A286 Super Alloy	92423A505	$2.07 Each	2.07	16.56
10	A286 Super Alloy	92423A539	$3.68 Each	3.68	29.44
				Total ($):	**358.26**

Notes:
-SHCS = Socket Head Cap Screw
-2 of the screws in Item 10 are to have their heads machined down so they can be countersunk under the penetrator plate
-These 2 screws may have to be switched to 3/4" length to fit (Part No. 92423A536)
-SSL stock has plenty of 18-8, 1" length screws with 1/4"-20 threads
-SSL stock has a lot of18-8, 1/2" length, 10-32 screws, but not excessive quantities
-SSL has few 18-8, 3/8" length, 10-32 screws. Count is presently at 18 (7/24/08)
-Source: www.mcmaster.com

Table A.3: End-effector Heli-Coil inserts.

INSERTS

Item	Insert Description	Total Quantity	Thread	Drill Depth (in)	
1	For Screws Connecting Flange to Al Adapter	12	10-32	0.662	
2	For Screws Connecting Al Adapter to Cam Disk	12	10-32	0.568	
3	For Track Roller Attachment in Guide Block	2 (1 x 2 Blocks)	1/4-28	0.714	
4	For Shoulder Screws into Guide Block	16 (8 x 2 Blocks)	10-32	0.568	
5	For Jaw Screws into Guide Block	14 (7 x 2 Blocks)	10-32	0.472	
6	For Jaw Screws Connecting Side and Rear Plates	8 (2 x 4 Pieces)	10-32	0.472	
7	For Jaw Screws Connecting Top to Side & Rear Plates	16 (8 x 2 Pieces)	10-32	0.472	
8	For Screws Bolting Connection Plate to Guide Rails	8 (4 x 2 Pieces)	10-32	0.662	
9	For Screws Bolting Side Mounts to Guide Rails	8 (4 x 2 Pieces)	10-32	0.662	
10	For Screws Connecting Side Mounts to Wrist Joint	8 (4 x 2 Sides)	1/4-20	0.675	

Item	Thread Depth (in)	Insert Number	McMaster-Carr #	Cost	Ind. Cost ($)
1	0.38	1191-3CN380	91732A725	$5.29 per 10	0.529
2	0.285	1191-3CN285	91732A231	$4.10 per 10	0.41
3	0.375	1191-4CN375	91732A232	$4.10 per 10	0.41
4	0.285	1191-3CN285	91732A231	$4.10 per 10	0.41
5	0.19	1191-3CN190	91732A511	$2.92 per 10	0.292
6	0.19	1191-3CN190	91732A511	$2.92 per 10	0.292
7	0.19	1191-3CN190	91732A511	$2.92 per 10	0.292
8	0.38	1191-3CN380	91732A725	$5.29 per 10	0.529
9	0.38	1191-3CN380	91732A725	$5.29 per 10	0.529
10	0.25	1185-4CN250	91732A368	$3.13 per 10	0.313

Item	Sum Quantity	Thread	Drill Depth (in)	Thread Depth (in)	Insert Number
A	38	10-32	0.472	0.19	1191-3CN190
B	28	10-32	0.568	0.285	1191-3CN285
C	28	10-32	0.662	0.38	1191-3CN380
D	2	1/4-28	0.714	0.375	1191-4CN375
E	8	1/4-20	0.675	0.25	1185-4CN250
	Total Quantity: 104				

Item	McMaster-Carr #	Ind. Cost ($)	Tot. Cost ($)		
A	91732A511	0.292	11.096		
B	91732A231	0.41	11.48		
C	91732A725	0.529	14.812		
D	91732A232	0.41	0.82		
E	91732A368	0.313	2.504		
		Total Cost ($):	**40.71**		

Table A.4: End-effector washers.

WASHERS

Item	Washer Description	Quantity	ID (in)	OD (in)	Thick. (in)	Washer Size
A	Separate Screws from Al Adapter	12	0.195	0.354	.06 to .067	10
B	Separate Screws from Guide Rail Sides	8	0.195	0.354	.06 to .067	10
C	Separate Screws from Side Mount Origins	6	0.255	0.468	0.035	1/4"
D	Separate Screws from Side Mount Origins	6	0.265	0.5	.059 to .067	1/4"

Prototype & Final Design (no difference)

Item	Material	Part #	Cost	Ind. Cost ($)	Total Cost ($)	
A	18-8 Stainless	90945A741	$8.75 per 100	0.0875	1.05	
B	18-8 Stainless	90945A741	$8.75 per 100	0.0875	0.7	
C	18-8 Stainless	90945A760	$6.38 per 100	0.0638	0.3828	
D	18-8 Stainless	98017A660	$5.28 per 100	0.0528	0.3168	
				Total ($):	**2.13**	

Notes:
-SSL stock contains plenty of #10 washers.
-C will replace D if thinner washers are necessary.
-Source: http://www.mcmaster.com

Table A.5: End-effector bearings.

BEARINGS

Item	Description	Material	Total Quantity	ID (in.)	OD (in.)
1	Ball Bearings for Guide Block	416 Stainless	16	1/4	5/8
2	Thrust Bearings for Guide Block	Delrin	32	1/4	5/8

Item	Thickness (in.)	Part Number	Ind. Cost ($)	Total Cost ($)	
1	13/64	6138K65	6.03	96.48	
2	1/16	2795T11	0.91	29.12	
			Total ($):	**125.60**	

Notes:
-The 2 roller bearings are considered fasteners due to their threaded ends.
-The bearings are the same for both the prototype and the final design.
-Source: http://www.mcmaster.com

Table A.6: Sample container components.

Vendor	Part Number	Quantity	Description	Material
United States Plastic Corp.	26333	1	Main Cylinder, 10", Schedule 80	PVC
United States Plastic Corp.	45089	1	Base Plate, 3/8" Thick Sheet	PVC
United States Plastic Corp.	26333	1	Top Ring, 10", Schedule 80	PVC
McMaster-Carr	87145K85	1	1/4" Thick Rubber Support Layer	Natural Gum Rubber
McMaster-Carr	87145K85	1	1/4" Thick Rubber Cover Layer	Natural Gum Rubber
McMaster-Carr	90945A740	24	#10 Washer, .195" ID, .354" OD, t = .067"	18-8 Stainless
McMaster-Carr	92196A273	12	10-32 Thread Screw, 7/8" Length	18-8 Stainless
McMaster-Carr	92196A275	12	10-32 Thread Screw, 1-1/8" Length	18-8 Stainless

A.2 Track Roller Data

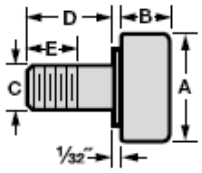
Track Roller Part Number: 8043K74

Corrosion-Resistant Extended-Life Track Rollers

Made entirely of Type 440C stainless steel, these track rollers offer excellent corrosion resistance. Quality construction gives them roughly twice the life of ordinary track rollers. They're sealed to block out contaminants and have needle bearings. Maximum temperature is 250° F.

Rollers with stud can be through-hole mounted or threaded directly into a housing or machine component (mounting nut not included). They have a hex head for easy fastening and one lubrication hole, unless noted.

Studless rollers mount onto a shaft or clevis (yoke) end linkage.



				With Stud					
Roller		Max.	Radial Load	Radial Load	Stud		Thread		
Dia. (A)	Wd. (B)	rpm @ No Load	Cap., lbs. (Dynamic)	Cap., lbs. (Static)	Dia. (C)	Lg. (D)	Size	Lg. (E)	Each
1/2"	3/8"	11,500	680	790	3/16"	5/8"	10-32	1/4"	8043K72♣ $61.58
5/8"	7/16"	9,200	955	1,215	1/4"	3/4"	1/4"-28	5/16"	8043K74♣ 61.20
3/4"	1/2"	6,400	1,660	2,065	3/8"	7/8"	3/8"-24	3/8"	8043K75 61.96
7/8"	1/2"	5,400	1,660	2,065	3/8"	7/8"	3/8"-24	3/8"	8043K76 62.72
1"	5/8"	4,800	2,225	3,060	7/16"	1"	7/16"-20	1/2"	8043K77 75.20
1 1/8"	5/8"	3,400	2,225	3,060	7/16"	1"	7/16"-20	1/2"	8043K78 77.06
1 1/4"	3/4"	3,100	3,930	4,250	1/2"	1 1/4"	1/2"-20	5/8"	8043K49 90.16
1 3/8"	3/4"	2,800	3,930	4,250	1/2"	1 1/4"	1/2"-20	5/8"	8043K52 94.18
1 1/2"	7/8"	2,500	4,840	5,640	5/8"	1 1/2"	5/8"-18	3/4"	8043K51 110.44
1 5/8"	7/8"	2,350	4,840	5,640	5/8"	1 1/2"	5/8"-18	3/4"	8043K56 109.56
1 3/4"	1"	2,200	6,385	7,920	3/4"	1 3/4"	3/4"-16	7/8"	8043K53 132.10
1 7/8"	1"	2,000	6,385	7,920	3/4"	1 3/4"	3/4"-16	7/8"	8043K57 133.81
2"	1 1/4"	1,400	8,090	10,570	7/8"	2"	7/8"-14	1"	8043K55 173.07

♣ Has no lubrication hole.

McMASTER-CARR

A.3 Hand Roll Motor Data Sheets

Hand Roll Motor Part Number: RBE-02111-A-00

RBE(H) Motor Series

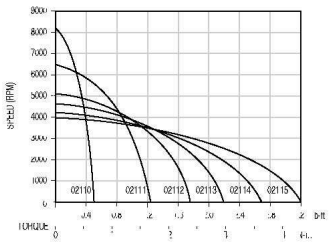
RBE(H) 02110 MOTOR SERIES PERFORMANCE DATA

Motor Parameters	Symbols	Units	02110	02111	02112	02113	02114	02115
Max Cont. Output Power	HP Rated	HP	0.323	0.672	0.761	0.854	0.944	1.07
at 25°C amb.	P Rated	Watts	241	501	568	637	704	796
Speed at Rated Power	N Rated	RPM	5300	4242	3500	3050	2770	2650
Max Mechanical Speed	N Max	RPM	12000	12000	12000	12000	12000	12000
Continuous Stall Torque	Tc	lb-ft	0.703	1.23	1.77	2.20	2.69	3.20
at 25°C amb.		N-m	0.952	1.67	2.40	2.99	3.64	4.33
Peak Torque	Tp	lb-ft	1.87	3.37	5.10	6.80	8.27	10.2
		N-m	2.55	4.57	6.92	9.22	11.2	13.8
Max Torque	Tsl	lb-ft	1.26	2.56	3.75	5.00	6.37	7.49
for Linear KT		N-m	1.72	3.47	5.08	6.78	8.64	10.2
Motor Constant	Tm	lb-ft/$\sqrt{W}$	0.102	0.175	0.243	0.293	0.345	0.394
		N-m/$\sqrt{W}$	0.139	0.237	0.329	0.396	0.467	0.534
Thermal Resistance*	Rth	°C/Watt	1.70	1.60	1.50	1.40	1.30	1.20
Viscous Damping	Fi	lb-ft/RPM	1.04E-05	2.36E-05	3.59E-05	4.82E-05	6.06E-05	7.29E-05
		N-m/RPM	1.41E-05	3.19E-05	4.87E-05	6.54E-05	8.21E-05	9.88E-05
Max Static Friction	Tf	lb-ft	0.026	0.052	0.077	0.10	0.13	0.15
		N-m	0.035	0.071	0.104	0.136	0.171	0.203
Max Cogging Torque	Tcog	lb-ft	0.016	0.039	0.061	0.082	0.104	0.125
Peak to Peak		N-m	0.022	0.053	0.083	0.111	0.141	0.169
Inertia	Jmf	lb-ft-sec^2	5.50E-05	9.70E-05	1.40E-04	1.74E-04	2.13E-04	2.66E-04
Frameless		Kg-m^2	7.46E-05	1.32E-04	1.90E-04	2.36E-04	2.89E-04	3.61E-04
Motor Weight	Wtf	lb	1.29	2.21	3.07	3.94	4.80	5.66
		Kg	0.585	1.00	1.41	1.77	2.18	2.59
Inertia	Jmh	lb-ft-sec^2	5.60E-05	1.10E-04	1.41E-04	1.75E-03	2.14E-04	2.62E-04
Housed		Kg-m^2	7.59E-05	1.49E-04	1.91E-04	2.37E-03	2.90E-04	3.55E-04
Motor Weight	Wth	lb	2.00	3.22	4.37	5.51	6.66	7.80
		Kg	0.907	1.46	2.00	2.50	3.04	3.54
No. of poles	P		12	12	12	12	12	12

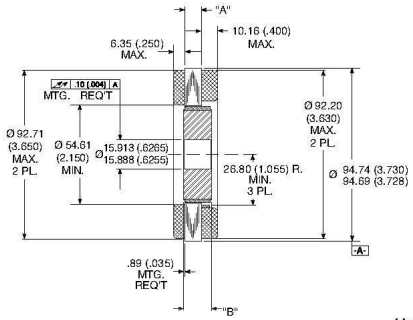
Winding Constants	Symbols	Units	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
Current at Cont. Torque	Ic	Amps	6.34	2.53	10.6	5.71	2.27	9.74	5.42	2.17	9.03	5.07	2.03	8.46	8.13	3.95	3.05	8.67	3.98	1.77
Current at Peak Torque	Ip	Amps	25.3	10.0	40.2	25.3	10.0	40.2	25.3	10.0	40.2	25.3	10.0	40.3	40.3	20.1	15.9	45.3	20.1	10.6
Torque Sensitivity	Kt	lb-ft/Amp	0.115	0.287	0.0690	0.225	0.566	0.132	0.341	0.851	0.204	0.454	1.14	0.272	0.347	0.714	0.925	0.386	0.840	1.89
		N-m/Amp	0.156	0.390	0.0935	0.305	0.768	0.179	0.462	1.15	0.277	0.62	1.54	0.37	0.471	0.968	1.24	0.523	1.14	2.56
Back EMF constant	Kb	V/KRPM	16.3	40.8	9.80	31.9	80.4	18.7	48.4	121	29.1	64.5	161	38.6	49.2	101	130	54.8	119	268
Motor Resistance	Rm	Ohms	1.27	8.05	0.479	1.66	10.6	0.611	1.97	12.5	0.743	2.40	15.2	0.904	1.01	4.17	6.83	0.961	4.74	23.2
Motor Inductance	Lm	mH	1.7	10	0.60	3.2	20	1.1	5.1	32	1.8	6.2	39	2.2	2.8	12	20	3.0	14	72

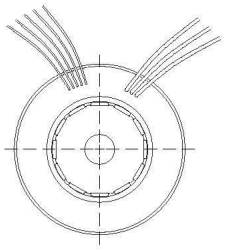
*Rth assumes a housed motor mounted to a 7" x 7.5" x 0.75" aluminum heatsink or equivalent

Continuous Duty Capability for 130°C Rise — RBE - 02110 Series



RBE-0211X-X00



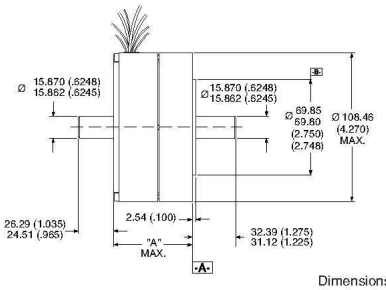


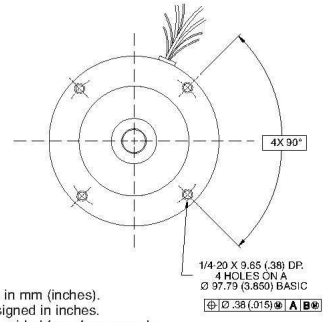
Dimensions in mm (inches).
Product designed in inches.
Metric conversions provided for reference only.

Notes:
1) For a C.W. rotation, as viewed from lead end, energize per excitation sequence table.
2) V-AB, V-BC and V-CA is back EMF of motor phases AB, BC and CA respectively, aligned with sensor output as shown for C.W. rotation only.
3) Mounting surface is between Ø 92.71 (3.650) and Ø 94.72 (3.729) on both sides.

MODEL NUMBER	RBE-02110	RBE-02111	RBE-02112	RBE-02113	RBE-02114	RBE-02115
"A" Dimension	8.89 (0.350)	19.05 (0.750)	28.58 (1.125)	38.1 (1.500)	47.63 (1.875)	57.15 (2.250)
"B" Dimension	15.24 (0.600)	25.4 (1.000)	34.93 (1.375)	44.45 (1.750)	53.98 (2.125)	63.5 (2.500)
Tolerance ± .010 on "A" Dimension.						

RBEH-0211X-00





Dimensions in mm (inches).
Product designed in inches.
Metric conversions provided for reference only.

Notes:
1) Shaft end play: with a 18 lb reversing load, the axial displacement shall be .013-.13 (.0005-.005).
2) For a C.C.W. rotation, as viewed from pilot end, energize per excitation sequence table.
3) V-AB, V-BC and V-CA is back EMF of motor phases AB, BC and CA respectively, aligned with sensor output as shown for C.C.W. rotation only.

MODEL NUMBER	RBEH-02110	RBEH-02111	RBEH-02112	RBEH-02113	RBEH-02114	RBEH-02115
"A" Dimension	57.86 (2.278)	68.02 (2.678)	77.55 (3.053)	87.07 (3.428)	96.60 (3.803)	106.12 (4.178)

RBE/RBEH LEADWIRE

Motor Leads: #18 AWG Teflon coated per MIL-W-22759/11, 3 leads, 152 (6.00) min lg. ea. 1-black, 1-red, 1-white.

Sensor Leads: #26 AWG type "ET" Teflon coated per MIL-W-16878, 5 leads, 152 (6.00) min lg. ea. 1-blue, 1-brown, 1-green, 1-orange, 1-yellow.

Appendix B

MATLAB Function for Evaluating Cam Disk Performance

```
function camdisk()

%This function solves the grooves in the cam disk and finds the associated dynamic properties

close all, clear all                    %Close all windows and clear variables from previous runs

time_plots = 0;                         %Set time_plots to 1 to activate figures plotting variables vs. time
excel = 0;                              %Set to 1 to send time and position data to Position Data.xls spreadsheet
offset = .45;                           %Declare an offset which serves as the dist. between the center...
                                        %of the circle and the start of the groove
h = 2.25;                               %Total follower displacement (inches)
                                        %It is the height one of the jaws opens
beta_deg = 155;                          %Cam angle for displacement "h" (degrees)
%alpha_max_deg = 175.33;                %Maximum cam disk rotation - found with trig. (deg)
alpha_max_deg = 173.56;                 %Actual max cam disk rotation - limited by jaws contacting (deg)
iter = .0005;                           %Iteration size of theta (rad)
cd_rad = 2.3;                           %Cam disk radius (inches)...
                                        %Diameter of flange/cam disk adapter is 4.60 inches
follower_lim = 955;                     %Cam follower dynamic radial load capacity (lbs)
T_in = 144;                             %Input torque, based on motor break-in values (inch-lbs), SAMURAI Boards
%T_in = 98.3;                           %Input torque, based on motor break-in values (inch-lbs), Ranger Boards
eta = .9;                               %Combined efficiency of track rollers and guide block bearings (90% each)
ang_vel_deg = 13.86;                    %Angular velocity of hand roll joint, found empirically (deg/s)
ang_vel = ang_vel_deg * pi/180;         %Angular velocity converted to (rad/s)
Lo = -10;                               %External load on the cam follower, value is neg. as force is down (lbs)

%Plot controls and variables:
   title_size = 26;                     %Set title, xlabel, and ylable font sizes
   axes_size = 24;                      %Set axes font size
   line_width = 4;                      %Set width of plotted lines
   mult = 2;                             %Line_width multiplier for the cam disk grooves
   fig_color = [1 1 1];                  %Sets background in figures to white
   set(0, 'DefaultFigureColor', fig_color, 'DefaultAxesLineWidth',...
      line_width, 'DefaultAxesFontSize', axes_size);        %Set figure color line width of axes, axes font size
%End plot controls and variables

beta = beta_deg * pi/180;               %Convert beta angle into radians (rad)

%Create a visual depiction of the curves to be inserted into the cam disk to drive the parallel jaws
%Step 1: Create the first groove in the cam disk (presently on right side of disk
theta_deg = 0;                          %Reset theta_deg to its initial position (deg)
x1_array = [];   y1_array = [];         %Initialize x and y arrays, which will contain individual values (in)
%Initialize rotation arrays - these arrays represent the x and y arrays put
%through a rotation matrix to track points in the cam-disk grooves:
x_rot_array = [];  y_rot_array = [];
%Initialize velocity, acceleration, and output force arrays:
vel_array = [];   accel_array = [];     F_out_array = [];
%Initialize variables used in velocity and acceleration calculations:
y_old = 0;  time_old = 0;   v_old = 0;
%Initialize variables used in pressure angle calculations:
y3 = 0; x3 = 0; gamma_save = -pi; gamma_array = [];
%Initialize rotation angle variables:
theta_deg_array = [];  alpha_deg_array = [];   alpha = 0;  alpha_deg = 0;
theta_deg_incr = iter*(180/pi);                 %Increment value for theta_deg (deg)
while theta_deg <= beta_deg;                    %As long as theta is within the beta limit...
   theta = theta_deg * pi/180;                  %Convert theta to radians (rad)
   radius = (h/2)*(1+cos((beta_deg*(pi/180))*theta/beta));     %Follower displacement (inches)

   x1 = radius * cos(theta-pi/2);               %X position of the location within the 1st groove (in)
   y1 = radius * sin(theta-pi/2)+offset;        %Y position of the location within the 1st groove (in)
   %Note: The pi/2 values in the trig. terms rotate the grooves for easier viewing
   x1_array = [x1_array, x1];                   %Add latest x position to the x_array (in)
```

```
   y1_array = [y1_array, y1];                    %Add latest y position to the y_array (in)

   l1 = sqrt(x1^2 + y1^2);                       %Calculate the length of the vector to the point in the groove
   if alpha_deg <= 89.99,                        %Case for which alpha <= 90 degrees
      alpha = asin(x1/l1);                       %Calculate the angle via inverse sine function
   else                                          %Case for which alpha >= 90 degrees
      alpha = asin(y1/l1) + pi/2;                %Calculate the angle via inverse sine function + 90 deg.
   end
   alpha_deg_old = alpha_deg;                          %Store value of alphs_deg in "old" variable (deg)
   alpha_deg = alpha * 180/pi;                         %Convert alpha to degrees (deg)
   delta_alpha_deg = alpha_deg - alpha_deg_old;        %Calculate change in alpha (deg)
   delta_alpha = delta_alpha_deg * pi/180;             %Convert change in alpha to radians (rad)
   alpha_deg_array = [alpha_deg_array, alpha_deg];     %Add alpha_deg to array
   theta_deg_array = [theta_deg_array, theta_deg];     %Add theta_deg to theta_deg_array

   y_rot = -x1*sin(alpha - pi) + y1*cos(alpha-pi);     %Y-coords after rotation alpha (in)
   x_rot = x1*cos(alpha-pi) + y1*sin(alpha-pi);        %X-coords after rotation alpha (in)
   y_rot_array = [y_rot_array, y_rot];                 %Add rotated y-coord to array
   x_rot_array = [x_rot_array, x_rot];                 %Add rotated x-coord to array

   %Velocity Determination:
   y_new = y_rot;                                %Set new y-position to the the rotated value (in)
   delta_y = y_new - y_old;                      %Calculate the change in position over the time step (in)
   y_old = y_rot;                                %Redefine "y_old" as the new position (in)
   time_new = alpha_deg / ang_vel_deg;           %Calculate the time at the given position (s)
   delta_time = time_new - time_old;             %Calculate the change in time from the previous time step (s)
   time_old = time_new;                          %Redefine "time_old" as the new time (s)
   vel = delta_y / delta_time;                   %Divide change in position by change in time to get velocity (in/s)
   vel_array = [vel_array, vel];                 %Put velocity term in velocity array

   %Acceleration Determination:
   v_new = vel;                                  %Set new velocity term to the calculated value (in/s)
   delta_v = v_new - v_old;                      %Determine change in velocity across the time step (in/s)
   v_old = v_new;                                %Redefine vel_old as the new velocity (in/s)
   accel = delta_v / delta_time;                 %Divide change in velocity by change in time to get accel. (in/s^2)
   accel_array = [accel_array, accel];           %Add acceleration term to the array

   %Force Determination:
   F_out = T_in * (delta_alpha)/delta_y;         %Individual output force (lbs)
   F_out_array = [F_out_array, F_out];           %Add force value to the output force array (lbs)

   %Pressure Angle Determination:
   x4 = x1;                                      %Set the new x (x4) equal to the new x1 (in)
   y4 = y1;                                      %Set the new y (y4) equal to the new y1 (in)
   slope = (y4 - y3)/(x4 - x3);                  %Calculate the slope based on changes in x and y
   x3 = x1;                                      %Set x3 to what is now the old x1 (in)
   y3 = y1;                                      %Set y3 to what is now the old y1 (in)
   slope_perp = -(1/slope);                      %Calculate the neg. repicrocal to find the perpendicular slope
   gamma = atan(slope_perp) + pi/2 - alpha;      %Let gamma equal to the pressure angle
                                                 %pi/2 accounts for the frame rotation
                                                 %-alpha is used to find the difference between the follower…
                                                 %direction and vector normal to the cam
   %Use if statements to correct for tangent calculations:
   if gamma >= gamma_save | alpha_deg <= 0.036,        %If gamma is increasing or if alpha is near zero
      gamma_array = [gamma_array, gamma];              %Add gamma term to array
   else
      gamma = gamma + pi;                              %If tangent dropped the pressure angle, flip the…
                                                       %direction back with pi addition
      gamma_array = [gamma_array, gamma];              %Add gamma term to array
   end
   gamma_save = gamma;                                  %Update gamma_save for comparison purposes

   theta_deg = theta_deg + theta_deg_incr;              %Increment theta_deg (deg)
end

%Create Figure 1 and plot the curve for the first groove in the cam disk:
figure(1),   plot(x1_array, y1_array,'r-', 'LineWidth', line_width*mult), hold on
%Step 2: Create the second groove in the cam disk
theta_deg = 0;                                  %Reset theta_deg to its initial position (deg)
x2_array = [];   y2_array = [];                 %Initialize x and y arrays, which will contain individual values (in)
```

```
while theta_deg <= beta_deg;                    %As long as theta is within the beta limit...
   theta = theta_deg * pi/180;                  %Convert theta to radians (rad)
   radius = (h/2)*(1+cos((beta_deg*(pi/180))*theta/beta));          %Follower displacement (inches)
   x2 = -radius * cos(theta-pi/2);              %X position of the location within the 1st groove (in)
   y2 = -radius * sin(theta-pi/2) - offset;     %Y position of the location within the 1st groove (in)
   %Note: The pi/2 values in the trig. terms rotate the grooves for easier viewing
   x2_array = [x2_array, x2];                   %Add latest x position to the x_array (in)
   y2_array = [y2_array, y2];                   %Add latest y position to the y_array (in)
   theta_deg = theta_deg + theta_deg_incr;              %Increment theta_deg (deg)
end
%Plot the curve for the second groove in the cam disk
plot(x2_array, y2_array, 'r-', 'LineWidth', line_width*mult), hold on

%Plot the four quadrants of the cam disk border (it's a circle)
x = 0       : iter : cd_rad;  y = sqrt(cd_rad^2-x.^2);   plot(x,y,'-', 'LineWidth', line_width), hold on
x = -cd_rad : iter :      0;  y = sqrt(cd_rad^2-x.^2);   plot(x,y,'-', 'LineWidth', line_width), hold on
x = -cd_rad : iter :      0;  y = -sqrt(cd_rad^2-x.^2);  plot(x,y,'-', 'LineWidth', line_width), hold on
x = 0       : iter : cd_rad;  y = -sqrt(cd_rad^2-x.^2);  plot(x,y,'-', 'LineWidth', line_width), hold on
x = 0;   y = 0;   plot(x,y,'k+')   %Plot point in center of disk

title('Depiction of Cam Disk Grooves', 'FontWeight', 'bold')
xlabel('X-Position (in.)'), ylabel('Y-Position (in.)')
grid on, axis([-2.5 2.5 -2.5 2.5])                  %Manually set axes for better plot viewing


%Generate plot of vertical position of track roller:
%Note: Plot of horizontal position is a constant at zero
figure(2), plot(alpha_deg_array, y_rot_array, '-', 'LineWidth', line_width), hold on
title('Follower Displacement vs. Cam Disk Angle of Rotation', 'FontWeight', 'bold')
xlabel('Angle of Rotation (deg.)'), ylabel('Follower Displacement (in.)')
grid on, axis([0 alpha_max_deg .4 2])               %Manually set axes for better plot viewing

time = alpha_deg_array / ang_vel_deg;               %Time corresponding to a given angle of rotation (s)...
[time_last_index] = find(time,1, 'last');           %Index of last value in the time array
time_last = time(time_last_index);                  %Last time value in array (s)


%Generage plot of velocity of track roller as a function of angle alpha
figure(3), plot(alpha_deg_array, vel_array, '-', 'LineWidth', line_width)
title('Jaw Velocity vs. Cam Disk Angle of Rotation', 'FontWeight', 'bold')    %Plot jaw velocity
xlabel('Angle of Rotation (deg.)'), ylabel('Jaw Velocity (in./deg.)')
grid on, axis([0 alpha_max_deg -.25 0])                 %Manually set axes for better plot viewing

%Generate plot of acceleration of track roller as a function of angle alpha
figure(4), plot(alpha_deg_array, accel_array, '-', 'LineWidth', line_width)          %Plot jaw acceleration
title('Jaw Acceleration vs. Cam Disk Angle of Rotation', 'FontWeight', 'bold')
xlabel('Angle of Rotation (deg.)'), ylabel('Jaw Acceleration (in./deg.^2)')
grid on, axis([.1 alpha_max_deg -.2 0])                 %Manually set axes for better plot viewing

%Generate plot of output force of track roller as a function of angle alpha
   %Note: F_out is multiplied by a minus sign because it's only the magnitude that matters here
   %F_out is divided by 2 because there are 2 jaws, each of which is receiving the same amount of work
%F_out_array = -eta * F_out_array/2;
F_out_array = -eta * F_out_array;
%Plot output force versus angle of rotation:
figure(5), plot(alpha_deg_array, F_out_array, '-', 'LineWidth', line_width)
title('Output Force vs. Angle of Rotation', 'FontWeight', 'bold')
xlabel('Angle of Rotation (deg.)'), ylabel('Output Force (lbs.)')
grid on, axis([.1 alpha_max_deg 0 8*10^2])              %Manually set axes for better plot viewing


%Plot output force versus angle of rotation along with cam follower capacity limits
figure(6), plot(alpha_deg_array, F_out_array, '-', 'LineWidth', line_width), hold on
follower_lim = alpha_deg_array .* 0 + follower_lim;     %Create follower limit array to speed up computation
plot(alpha_deg_array, follower_lim, 'r-', 'LineWidth', line_width)      %Add follower limit to the plot
title('Force vs. Angle of Rotation', 'FontWeight', 'bold')
xlabel('Angle of Rotation (deg.)'), ylabel('Force (lbs.)')
legend('Output Force Capacity', 'Cam Follower Structural Limit', 'Location', 'Best')        %Generate legend
grid on, axis([.1 alpha_max_deg 0 4000])                %Manually set axes for better plot viewing
```

```
%Output maximum and minimum forces to the Command Window
fprintf('\nMaximum Output Force: %.4g lbs\n', max(abs(F_out_array)))      %Output max force (lbs)
%Minimum force corrsponds to the asymptote at theta = 0, so min() can't be used
[M, N] = size(F_out_array);                                              %Find the size of the array
F_out_min = abs(F_out_array(N));                    %Find the output force corresponding ot the index & divide by 2
fprintf('Minimum Output Force: %.6g lbs\n', F_out_min)  %Output min force (lbs)

%Plot pressure angle versus angle of rotation:
gamma_deg_array = gamma_array * 180/pi;                  %Convert array of pressure angles to degrees (deg.)
figure(7), plot(alpha_deg_array, gamma_deg_array, '-', 'LineWidth', line_width)
title('Pressure Angle vs. Angle of Rotation', 'FontWeight', 'bold')
xlabel('Angle of Rotation (deg.)'), ylabel('Pressure Angle (deg.)')
grid on, axis([.1 alpha_max_deg -10 75])                 %Manually set axes for better plot viewing

%Generate plot of vector components of cam disk output force capacity:
%Note: F_out_array has already been switched to pos. & divided by 2 at this point
F_out_x_array = F_out_array .* sin(gamma_array);    %Determination of the force capability in the x-direction (lbs)
F_out_y_array = F_out_array .* cos(gamma_array);    %Determination of the force capability in the x-direction (lbs)
figure(8), plot(alpha_deg_array, F_out_array, '-', 'LineWidth', line_width), hold on
plot(alpha_deg_array, F_out_x_array, 'g-', 'LineWidth', line_width), hold on
plot(alpha_deg_array, F_out_y_array, 'r-', 'LineWidth', line_width)
title('Directional Force Capabilities vs. Angle of Rotation', 'FontWeight', 'bold')
xlabel('Angle of Rotation (deg.)'), ylabel('Force (lbs.)')
legend('F_r_o_l_l_e_r', 'F_r_a_i_l', 'F_j_a_w', 'Location', 'Best')
grid on, axis([.1 alpha_max_deg 0 5*10^2])                               %Manually set axes for better plot viewing

%Generate plot of vector components of cam disk output force based on external follower load:
F_Lo_array = -Lo./(cos(gamma_array));
F_Lo_x_array = F_Lo_array .* sin(gamma_array);
F_Lo_y_array = F_Lo_array .* cos(gamma_array);
figure(9), plot(alpha_deg_array, F_Lo_array, '-', 'LineWidth', line_width), hold on
plot(alpha_deg_array, F_Lo_x_array, 'g-', 'LineWidth', line_width), hold on
plot(alpha_deg_array, F_Lo_y_array, 'r-', 'LineWidth', line_width)
title('Directional Reaction Forces vs. Angle of Rotation', 'FontWeight', 'bold')
xlabel('Angle of Rotation (deg.)'), ylabel('Force (lbs.)')
legend('Normal Force', 'X-Dir. Force', 'Y-Dir. Force', 'Location', 'Best')
grid on, axis([.1 alpha_max_deg -25 25])                                 %Manually set axes for better plot viewing

%Generate plot of factor of safety of track roller based on jaw closing force:
FOS = follower_lim ./ F_out_array;       %Factor of safety (unitless)
figure(10), plot(alpha_deg_array, FOS, '-', 'LineWidth', line_width), hold on
title('Track Roller FOS vs. Angle of Rotation', 'FontWeight', 'bold')
xlabel('Angle of Rotation (deg.)'), ylabel('Factor of Safety')
grid on, axis([67.5 alpha_max_deg 0 7])                                  %Manually set axes for better plot viewing


if excel,                    %If excel is set to logical true...
   xlswrite('Position Data.xls', time', 'MATLAB', 'A1:A5411')            %Send time data to spreadsheet
   xlswrite('Position Data.xls', alpha_deg_array', 'MATLAB', 'B1:B5411')%Send angle data to spreadsheet
   xlswrite('Position Data.xls', y_rot_array', 'MATLAB', 'D1:D5411')     %Send position data to spreadsheet
   xlswrite('Position Data.xls', vel_array', 'MATLAB', 'E1:E5411')       %Send velocity data to spreadsheet
   xlswrite('Position Data.xls', accel_array', 'MATLAB', 'F1:F5411')     %Send acceleration data to spreadsheet
   xlswrite('Position Data.xls', F_out_array', 'MATLAB', 'G1:G5411')     %Send output force data to spreadsheet
end

if time_plots,                                                           %If time_plots is set to logical true...
   %Plot position, velocity, and acceleration curves vs. time
   figure(11), plot(time, y_rot_array, '-', 'LineWidth', line_width)     %Plot jaw displacement
   title('Follower Displacement vs. Time', 'FontWeight', 'bold')
   xlabel('Time (s)'), ylabel('Follower Displacement (in.)')
   grid on, axis([0 time_last .4 2])                                     %Manually set axes for better plot viewing

   figure(12), plot(time, vel_array, '-', 'LineWidth', line_width)
   title('Jaw Velocity vs. Time', 'FontWeight', 'bold')                  %Plot jaw velocity
   xlabel('Time (s)'), ylabel('Jaw Velocity (in./s)')
   grid on, axis([0 time_last -.25 0])                                   %Manually set axes for better plot viewing

   figure(13), plot(time, accel_array, '-', 'LineWidth', line_width)     %Plot jaw acceleration
   title('Jaw Acceleration vs. Time', 'FontWeight', 'bold')
   xlabel('Time (s)'), ylabel('Jaw Acceleration (in./s^2)')
```

```
    grid on, axis([.01 time_last -.2 0])                                %Manually set axes for better plot viewing

    figure(14), plot(time, F_out_array, '-', 'LineWidth', line_width)   %Plot jaw force capacity
    title('Output Force vs. Time', 'FontWeight', 'bold')
    xlabel('Time (s)'), ylabel('Output Force (lbs.)')
    grid on, axis([0 time_last 0 5*10^4])                               %Manually set axes for better plot viewing

    figure(15), plot(alpha_deg_array, F_out_array, '-', 'LineWidth', line_width), hold on
    plot(alpha_deg_array, follower_lim, 'r-', 'LineWidth', line_width)  %Plot jaw force capacity with follwer limits
    title('Force vs. Angle of Rotation', 'FontWeight', 'bold')
    xlabel('Angle of Rotation (deg.)'), ylabel('Force (lbs.)')
    legend('Output Force Capacity', 'Cam Follower Structural Limit', 'Location', 'Best')
    grid on, axis([.1 time_last 0 4000])                                %Manually set axes for better plot viewing
end
```

Appendix C

Additional Jaw Performance Metrics

As stated in Chapter 5, both MATLAB and CAD analysis features were used to generate plots of position, velocity, and acceleration. Data points from both programs were exported to Excel and were subsequently plotted together. These plots are shown below.

The CAD software was only capable of plotting these variables as functions of time. To make the necessary conversion, the known angular position was multiplied by a constant angular velocity of 13.9°/s to produce time. This rotation rate corresponds to that observed when powering the hand roll motor with the Ranger boards while applying no external load.

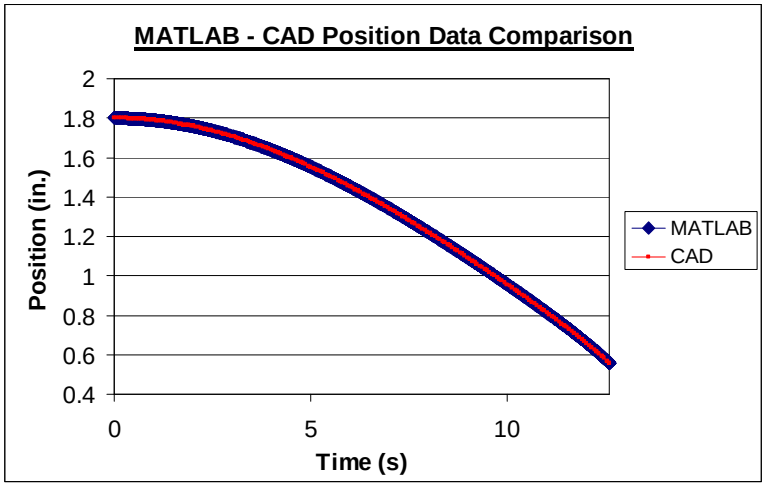


Figure C.1: Plot of follower displacement vs. time based on MATLAB and CAD data.

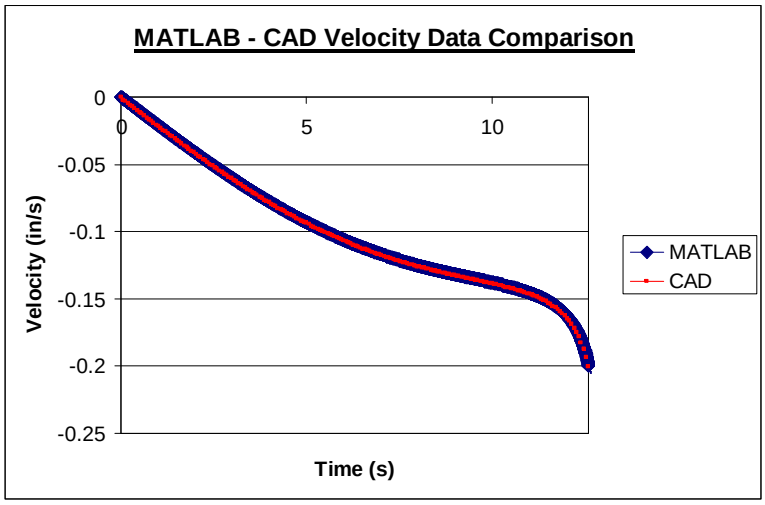


Figure C.2: Plot of follower velocity vs. time based on MATLAB and CAD data.

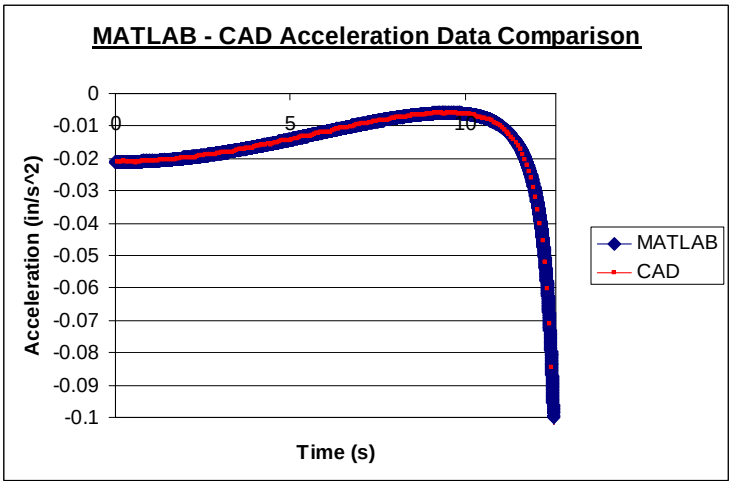


Figure C.3: Plot of follower acceleration vs. time based on MATLAB and CAD data.

Appendix D
End-Effector Structural Analysis and Test Data

Supplementary Structural Analysis

Analyses were performed for all relevant structural failure modes. Different types of stresses imposed on the various end-effector components are discussed, and the applicable equations are presented. All calculations performed in this appendix correspond to a maximum hand roll joint output torque of 144 lb-ft (195 N-m). The load, the component, and the FOS are then presented in Tables D.1 through D.4. A more detailed display of all the parameters used in the structural calculations is featured in Table D.5.

Track-roller loading and loads applied on the guied block bearings are not evaluated as those calculations were presented in Chapter 5.

D.1 Shear Stresses

The joint torque directly imposes a shear stress on the following end-effector components: the flange-adapter fastener ring, flange-cam disk adapter, adapter-cam disk fastener ring, and the cam disk itself.

Shear stress (τ) is calculated in (D-1), where T is the applied torque, r is the distance from the rotation axis to the outer surface of the part, and I_p is the polar moment of inertia. Because the majority of the end-effector components contained cross-sections more complicated than basic circles and rectangles, moments of inertia were found with CAD analyses.

$$\tau = \frac{T \cdot r}{I_p} \tag{D-1}$$

Table D.1: Shear stress summary.

Component	τ (psi)	FOS
Flange-Adapter Fastener Ring	2327.3	8
Flange-Cam Disk Adapter	119.6	184
Adapter-Cam Disk Fastener Ring	2741.7	7
Cam Disk Rear	116.0	190
Cam Disk Front (w/ grooves)	131.7	167

D.2 Bending Stresses

The horizontal force component in the cam follower force vector induces a bending moment on the side mounting brace. Each guide rail is impacted by one half of the total force value of 120.1 lb. This value is further reduced by an additional factor of 2 due to the connector plates which stabilize the guide rails. The total force acting on each of the rails is therefore 30.0 lb.

This force value is used to calculate the bending moment, *M*. Combining this with the maximum distance from the neutral axis to the plate edge, *c*, and the area moment of inertia, *I*, the bending stress (σ_b) is determined using Equation D-2.

$$\sigma_b = \frac{M \cdot c}{I} \tag{D-2}$$

Table D.2: Bending stress summary.

Component	σ (psi)	FOS
Side Mounting Brace	4540.0	9

D.3 Tensile Stresses

The ¼ in-thick aluminum connectors are used to stabilize the guide rails. These objects are in tension due to the opposing forces being applied to the respective rails. The equation for the tensile stress (σ_t) being exerted on the members is shown

in Equation D-3, where *F* and *A* represent the applied load and cross-sectional area, respectively.

$$\sigma_t = \frac{F}{A} \tag{D-3}$$

Table D.3: Tensile stress summary.

Component	σ (psi)	FOS
Guide Rail Connector	96.8	413

D.4 Compressive Stresses

The rail forces produce compression between the guide blocks and the guide rails. This stress is experienced by the guide blocks, guide rails, and Delrin thrust bearings. As there are four sets of thrust bearings on each side of the guide block, the load is divided by four for these components.

The equation for compression is identical to that shown in Equation D-3.

Table D.4: Compressive stress summary.

Component	σ (psi)	FOS
Thurst Bearings	58.6	17
Guide Rail	25.9	1544
Guide Block	72.3	554

D.5 Stresses Caused by Jaw Closure

When the jaws shut upon an object, the jaws pitch in opposite directions. This induces a bending moment on the seven screws connecting the jaws to the guide blocks. Additionally, a radial load is applied to four of the eight bearings on each of the guide blocks. The load value for these calculations is taken to be 87.7 lb, the jaw closing force at the closed position.

Equation D-2 is used to find the bending stress in the screws, and the applied load is assumed to be spread across each of the bearings equally.

Table D.5: Jaw closure-induced tress summary.

Component	Load	FOS
Guide Block-Jaw Attachment Fasteners	1049.5 psi	38
Ball Bearings	15.0 lb	22

D.6 Structural Analysis Data Table

Table D.6: Structural analysis data.

Load	Component	T (lb-ft)	T (lb-in)	r (in)	I_p (in^4)	τ (psi)
Shear Caused Directly by Joint Torque	Flange-Adapter Fastener Ring	144	1728	2.3	1.7077	2327.3
	Flange-Cam Disk Adapter	144	1728	2.3	33.225	119.6
	Adapter-Cam Disk Fastener Ring	144	1728	1.85	1.166	2741.7
	Cam Disk Rear	144	1728	2.3	34.27	116.0
	Cam Disk Front (w/ grooves)	144	1728	2.3	30.175	131.7
Load	Component	F (lb)	x (in)	M (lb-in)	c (in)	I (in^4)
Bending Caused by Track Roller Rail Force	Side Mounting Brace	30.025	7.079	212.54698	0.1875	0.008778
Load	Component				F (lb)	A (in^2)
Tension from Rail Force	Guide Rail Connector				30.025	0.3101
Load	Component		$F_{applied}$ (lb)	Quantity	$(F_{app})_i$ (lb)	A_i (in^2)
Compression Caused by Track Roller Rail Force	Thurst Bearings		60.05	4	15.0125	0.2577
					F (lb)	A (in^2)
	Guide Rail				60.05	2.318
	Guide Block				60.05	0.8309
Load	Component	$(F_{app})_i$ (lb)	x (in)	M (lb-in)	c (in)	I (in^4)
Bending Caused by Jaw Closing Force	Guide Block-Jaw Fasteners	8.578571429	4.75	40.748214	0.1875	0.00728
						$F_{applied}$ (lb)
	Ball Bearings					60.05

Load	Component	Material	σ_{yield} (psi)	τ_{yield} (psi)	FOS	
Shear Caused Directly by Joint Torque	Flange-Adapter Fastener Ring	18-8 Stainless	31200	18096	**8**	
	Flange-Cam Disk Adapter	Al 6061	40000	22000	**184**	
	Adapter-Cam Disk Fastener Ring	18-8 Stainless	31200	18096	**7**	
	Cam Disk Rear	Al 6061	40000	22000	**190**	
	Cam Disk Front (w/ grooves)	Al 6061	40000	22000	**167**	
Load	Component	σ (psi)	Material	σ_{yield} (psi)	FOS	
Bending Caused by Track Roller Rail Force	Side Mounting Brace	4540.049876	Al 6061	40000	**9**	
Load	Component	σ (psi)	Material	σ_{yield} (psi)	FOS	
Tension from Rail Force	Guide Rail Connector	96.82360529	Al 6061	40000	**413**	
Load	Component	σ (psi)	Material	σ_{rating} (psi)	FOS	
Compression Caused by Track Roller Rail Force	Thurst Bearings	58.25572371	Delrin	1000	**17**	
		σ (psi)	Material	σ_{yield} (psi)	FOS	
	Guide Rail	25.90595341	Al 6061	40000	**1544**	
	Guide Block	72.27103141	Al 6061	40000	**553**	
Load	Component	σ (psi)	Material	σ_{yield} (psi)	FOS	
Bending Caused by Jaw Closing Force	Guide Block-Jaw Fasteners	1049.490409	Al 6061	40000	**38**	
		Quantity	(Fapp)i (lb)	F_{rated} (psi)	FOS	
	Ball Bearings	4	15.0125	332	**22**	

End-Effector Closing Force Test Data

Table D.7: End-effector closing force test data.

Input Current (A)	Motor Current (A)	Measured Force (lbs)	Actual Force (lbs)	Theoretical Force (lbs)
0.75	0.01	3.81	19.05	24.39
0.75	0.01	5.04	25.2	24.39
0.75	0.01	4.9	24.5	24.39
0.75	0.01	3.99	19.95	24.39
0.75	0.01	4.41	22.05	24.39
0.76	0.02	8.7	43.5	48.69
0.76	0.02	8.78	43.9	48.69
0.76	0.02	9.05	45.25	48.69
0.76	0.02	8.78	43.9	48.69
0.76	0.02	8.62	43.1	48.69
0.77	0.03	11.75	58.75	72.99
0.77	0.03	11.52	57.6	72.99
0.77	0.03	12.24	61.2	72.99
0.77	0.03	12.22	61.1	72.99
0.77	0.03	12.43	62.15	72.99
0.78	0.04	13.47	67.35	97.29
0.78	0.04	13.61	68.05	97.29
0.78	0.04	13.69	68.45	97.29
0.78	0.04	14.21	71.05	97.29
0.78	0.04	14.57	72.85	97.29
0.79	0.05	16.17	80.85	121.59
0.79	0.05	16.86	84.3	121.59
0.79	0.05	16.98	84.9	121.59
0.79	0.05	17.32	86.6	121.59
0.79	0.05	17.15	85.75	121.59

Appendix E

Sample Container Buoyancy Analysis

The objective of this analysis is to determine the wet weight of a sample container and the quantity of syntactic foam that would be required to render that container neutrally buoyant.

The net force of an object is given by the buoyancy relation shown in (E-1).

$$F_{net} = m \cdot g - \rho \cdot V \cdot g \tag{E-1}$$

The weight of the submerged object is represented by $m \cdot g$, while $\rho \cdot V \cdot g$ is the weight of the displaced water. To determine the weight, the manufacturer specifications were converted from 11.956 lb/ft to lb/in^3.

$$\rho_{PVC} = \frac{m}{V} = \frac{m}{l \cdot A} = \frac{11.956 lb_m}{(12in) \cdot \left[\frac{\pi}{4} \left(10.75in^2 - 9.492in^2 \right) \right]} = .0498 \frac{lb_m}{in^3} \tag{E-2}$$

The mass of the PVC was found by multiplying this density by the total PVC volume, which is 244.4 in^3 (0.141 ft^3).

$$m_{PVC} = (\rho_{PVC}) \cdot (V_{PVC}) = \left(.0498 \frac{lb}{in^3} \right) \cdot \left(244.4 in^3 \right) = 12.17 lb_m \tag{E-3}$$

The buoyancy calculations are performed for salt water density at sea-level as that is where the sample containers will first be tested. This value corresponds to 1.98 slugs/ft^3 (1020 kg/m^3). At a depth of 6000 m, the water density increases marginally to 2.04 slugs/ft^3 (1050 kg/m^3).

$$F_{net} = 12.17 lb_f - \left(1.98 \frac{slugs}{ft^3} \right) \cdot \left(0.141 ft^3 \right) \cdot \left(32.2 \frac{ft}{s^2} \right) = 3.16 lb_f \tag{E-4}$$

This value signifies that the PVC is 3.16 lb negatively buoyant. Repeating the procedure for the stainless steel screws and washers produces an additional wet weight of 0.31 lb, making the total net weight of the assembly **3.47 lb**.

Calculations are completed to determine how much foam would be required to make the container neutrally buoyant. The density of AZ (Abyssopelagic Zone) deep water foam that would potentially be used is 34 lb/ft^3. With that value, it is possible to solve for the required foam volume.

$$V_{foam} = \frac{F_{net}}{g \cdot \left(\rho_{foam} - \rho_{H2O}\right)} = \frac{-3.47lb}{32.2\frac{ft}{s^2} \cdot \left(34\frac{lb}{ft^3} - 63.7\frac{lb}{ft^3}\right)} = 0.117\,ft^3 = 202.0in^3 \qquad \text{(E-5)}$$

Thus, **202 in^3** of syntactic foam would be required to compensate for the 3.47 lb net force of the sample container assembly.

Appendix F
SAMURAI Inverse Kinematics

The preexisting inverse kinematics software composed by Carignan employs equivalent angle-axis representation to locate Frame 4 from the tool tip [27]. In the updated program, the user inputs sample container position and orientation, which are then used to specify tool tip position and generate a vector pointing to Frame 4. In (F-1), ${}^{0}P_{4}$ is the position of Frame 4 relative to Frame 0, and ${}^{4}P_{T}$ is that of the tool frame relative to Frame 4. ${}^{0}P_{T}$ and ${}^{0}_{T}R$ are the user-input sample container position and orientation, respectively. These values double as the tool tip location and the tool frame rotation matrix relative to the base frame.

$$ {}^{0}P_{4} = {}^{0}P_{T} - {}^{0}_{T}R \cdot {}^{4}P_{T} \tag{F-1} $$

After determining the location of Frame 4, a geometric approach is employed to determine the joint angles of the first three joints (θ_1, θ_2, θ_3) [27]. Recalling that Joint 1 is the only joint that produces yawing motion, θ_1 is found by taking the arctangent of the y-position (${}^{0}y_{4}$) divided by the x-position (${}^{0}x_{4}$), as shown in (F-2).

$$ \theta_1 = a\tan\left(\frac{{}^{0}y_4}{{}^{0}x_4}\right) = a\tan\left(\frac{{}^{0}P_4(2,1)}{{}^{0}P_4(1,1)}\right) \tag{F-2} $$

The position of Frame 2 relative to the base frame (${}^{0}P_{2}$) is a function of θ_1 exclusively and is known once (F-2) has been solved. Equation F-3 can then be applied to find ${}^{2}P_{4}$, the location of Frame 2 relative to Frame 4.

$$ {}^{2}P_{4} = {}^{0}P_{4} - {}^{0}P_{2} \tag{F-3} $$

The distance between Frames 2 and 4 in the x-y plane is set to r_{xy}, and r_z is defined as the z-separation between the joints. The total scalar distance between the frames (r) can be found with Equations F-4 through F-6.

$$r_{xy} = \sqrt{\left({}^2x_4\right)^2 + \left({}^2y_4\right)^2} = \sqrt{{}^2P_4(1,1)^2 + {}^2P_4(2,1)^2} \tag{F-4}$$

$$r_z = {}^2z_4 = {}^2P_4(3,1) \tag{F-5}$$

$$r = \sqrt{\left(r_{xy}\right)^2 + \left(r_z\right)^2} \tag{F-6}$$

These variables and others to be discussed can be seen in Figure F.1.

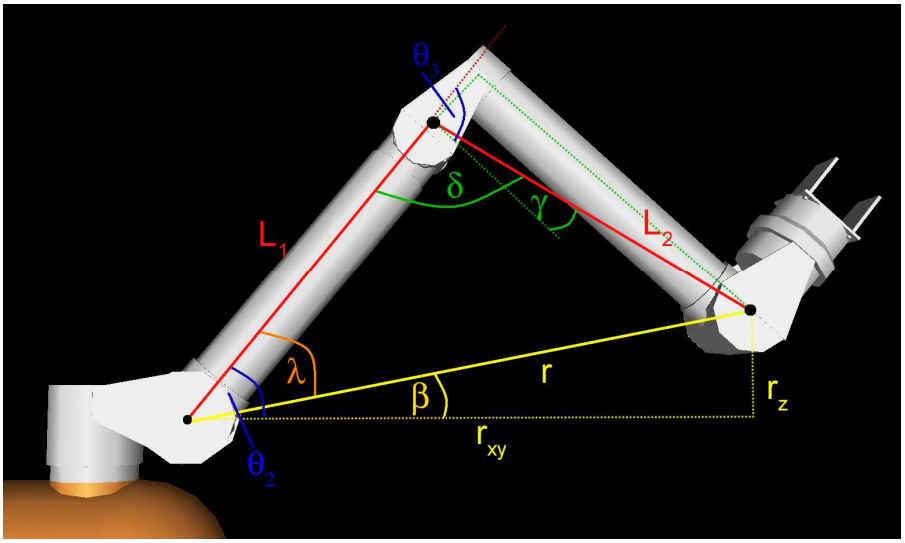


Figure F.1: Illustration of inverse kinematics solution variables.

The two manipulator links, L_1 and L_2, and r represent the three sides of a triangle. The angle opposite side r is defined as δ and is calculated using the law of cosines in (F-7).

$$\delta = a\cos\left[\frac{r^2 - {L_1}^2 - {L_2}^2}{-2 \cdot L_1 \cdot L_2}\right] \tag{F-7}$$

Examining the relationship at Joint 3 in Figure F.1, it is observed that θ_3 and δ minus γ sum to π radians. γ represents the geometric pitch from Joint 3 to Joint 4, which is constant regardless of manipulator configuration. Equation F-8 solves for θ_3

$$\theta_3 = \pi - \delta + \gamma \tag{F-8}$$

In Figure F.1, λ is the angle in the triangle opposite link L_2. Another application of the law of cosines to solve for this angle yields (F-9).

$$\lambda = a\cos\left[\frac{r^2 + {L_1}^2 - {L_2}^2}{2 \cdot L_1 \cdot r}\right] \tag{F-9}$$

Once β is determined by taking the arctangent of r_z/r_{xy}, θ_2 is found using (F-10).

$$\theta_2 = \beta + \lambda \tag{F-10}$$

With the determination of θ_1, θ_2, and θ_3, it is possible to calculate rotation matrices through Frame 3. This is represented in (F-11).

$${}^0_3R = {}^0_1R \cdot {}^1_2R \cdot {}^2_3R \tag{F-11}$$

As ${}^0_TR$ is the user-input sample container position and ${}^0_3R$ is now known, ${}^3_TR$ can be calculated with (F-12).

$${}^3_TR = \left({}^0_3R\right)^{-1} \cdot {}^0_TR \tag{F-12}$$

${}^3_TR$ is composed of the product of the ${}^3_4R$, ${}^4_5R$, and ${}^5_TR$ matrices, the latter component resulting from the fact that ${}^5_6R$ and ${}^5_TR$ are equivalent. These matrices are functions of θ_4, θ_5, and θ_6, and this is represented in (F-13).

$${}^3_TR(\theta_4, \theta_5, \theta_6) = {}^3_4R \cdot {}^4_5R \cdot {}^5_TR = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 \\ 0 & 0 & 1 \\ -s\theta_4 & -c\theta_4 & 0 \end{bmatrix} \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 \\ 0 & 0 & -1 \\ s\theta_5 & c\theta_5 & 0 \end{bmatrix} \begin{bmatrix} c\theta_6 & -s\theta_6 & 0 \\ 0 & 0 & 1 \\ -s\theta_6 & -c\theta_6 & 0 \end{bmatrix} \tag{F-13}$$

These respective matrices were found using symbolic variables in the forward kinematics portion of the program. Equation F-14 shows the product of these matrices.

$$ {}^{3}_{T}R(\theta_4,\theta_5,\theta_6)=\begin{bmatrix} c\theta_4c\theta_5c\theta_6-s\theta_4s\theta_6 & -c\theta_4c\theta_5s\theta_6-s\theta_4c\theta_6 & -c\theta_4s\theta_5 \\ s\theta_5c\theta_6 & -s\theta_5s\theta_6 & c\theta_5 \\ -s\theta_4c\theta_5c\theta_6-c\theta_4s\theta_6 & s\theta_4c\theta_5s\theta_6-c\theta_4c\theta_6 & s\theta_4s\theta_5 \end{bmatrix} \quad \text{(F-14)} $$

θ_5 is found by isolating the θ_5 sine terms in the first two columns of the second row and dividing the result by the cosine term from the third column. This process is illustrated in (F-15).

$$ \theta_5 = a\tan\left(\frac{\sqrt{(s\theta_5c\theta_6)^2+(-s\theta_5s\theta_6)^2}}{c\theta_5}\right)=a\tan\left(\frac{s\theta_5}{c\theta_5}\right) \quad \text{(F-15)} $$

With *sin(θ_5)* known, the θ_4 terms in the third column can be isolated, and θ_4 is calculated by taking the arctangent of the parameters as shown in (F-16).

$$ \theta_4 = a\tan\left(\frac{\frac{s\theta_4s\theta_5}{s\theta_5}}{\frac{-c\theta_4s\theta_5}{-s\theta_5}}\right)=a\tan\left(\frac{s\theta_4}{c\theta_4}\right) \quad \text{(F-16)} $$

θ_6 is determined in a similar manner to θ_4, though the terms of interest in this case are the first and second columns of the second row. θ_6 is computed in (F-17).

$$ \theta_6 = a\tan\left(\frac{\frac{-s\theta_5s\theta_6}{-s\theta_5}}{\frac{s\theta_5c\theta_6}{s\theta_5}}\right)=a\tan\left(\frac{s\theta_6}{c\theta_6}\right) \quad \text{(F-17)} $$

While θ_6 does not correspond to a rotating joint, it represents the relative orientation between the end effector and the sample container. For example, if θ_6 is

equal to 45°, it signifes that the sample container must be rotated 45° in order for the jaws to properly align with the sample container lid.

Having calculated θ_6, all of the joint angles have been determined and can be input to the forward kinematics software to generate the desired matrices and plots.

Appendix G

SAMURAI Range of Motion Determination

The manipulator joint ranges, previously detailed in Table 7.2, are reproduced in Table G.1.

Table G.1: SAMURAI joint ranges of motion.

Joint Number	Range of Motion (deg.)
1 (Shoulder Yaw)	220°
2 (Shoulder Pitch)	225°
3 (Elbow Pitch)	210°
4 (Elbow Roll)	540°
5 (Wrist Pitch)	215°
6 (Wrist Roll)	540°

Joint 1 (Shoulder Yaw):

The shoulder yaw joint contains a wedge-shaped hard stop in the outer housing measuring 95°. This works in conjunction with a 45° hard stop located in the support bearing. This 45° hard stop limits the joint motion by 22.5° on either side of the outer housing, or 45° total. Adding this to the 95° stop in the outer housing yields 140° total. Thus, the total range of motion of the shoulder yaw joint is 220°.

Joint 2 (Shoulder Pitch):

The shoulder pitch joint has no hard stops internal to the joint. Rather, it relies on the hard stops associated with the manipulator itself. Stated another way, the joint will operate until it forces the arm to collide with itself. This is true of the other two pitch joints as well. To determine the range of motion, the arm was driven until it triggered the OCP soft stop, and pictures were taken at these locations. Lines were drawn on the images representing various axes, and a protractor was used to

determine the approximate range of motion. The images for Joint 2 are shown below in Figure G.1.

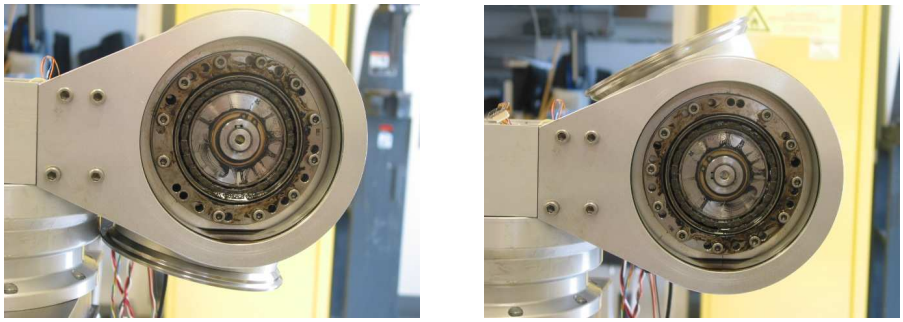


Figure G.1: Shoulder pitch joint at extreme pitch angles.

There are several limitations associated with this method. First, the resolution of the protractor imposes accuracy restrictions. Additionally, the positioning of the camera will affect the perspective and therefore the angle. The positioning of the Marman bands will produce different angles as the clasps extend further out than the bands themselves. Even the user-adjusted OCP limits will affect the angle as a higher limit will cause the joint to "push" harder into the manipulator. As a result of these factors, the joint angles were conservatively estimated to the nearest ±5°. For Joint 2, the range of motion was determined to be 225°.

Joint 3 (Elbow Pitch):

The range of motion for Joint 3 was found in a manner identical to that used for Joint 2. The images used to take angle measurements are shown below in Figure G.2. With these images, the Joint 3 range of motion was found to be 210°.

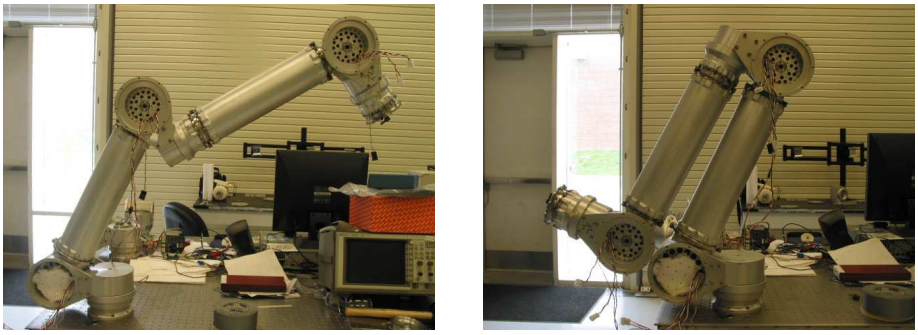


Figure G.2: Elbow pitch joint at extreme pitch angles.

Joint 4 (Elbow Roll):

Joint 4 does contain hard stops, but idlers allow it to rotate beyond 360° to 540°. In the MATLAB scripts that generate the work envelope, this will be limited to 360° as the extra 180° does not extend the workspace.

Joint 5 (Wrist Pitch):

The range of motion for Joint 5 was found in a manner identical to that used for Joints 2 and 3. The images used to take angle measurements are shown in Figure G.3. These images yielded a Joint 5 range of motion of 215°.

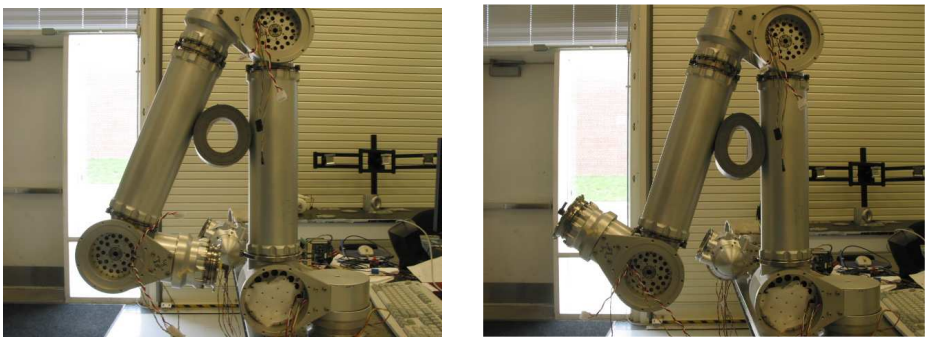


Figure G.3: Wrist pitch joint at extreme pitch angles.

Joint 6 (Hand Roll):

The hand roll joint functions in a manner identical to that of the elbow roll joint. Its range is also 540°.

Appendix H
MATLAB Kinematics Files

The MATLAB kinematics software is composed of 12 different functions. A brief description of the role of each of these functions is provided below.

Overview of the Different MATLAB Kinematics Functions:

astepgui.m

astepgui.m represents the script for the user interface. This is where the software reads the user input data, searching for active checkboxes and recording values for the joint angles, sample container positions, and workspace iterations. After reading-in all of these parameters, astepgui.m sends the settings to kinematics.m.

kinematics.m

kinematics.m is the controller of the kinematics software. It collects data from astepgui.m and distributes variables to their appropriate locations. Additionally, while DH parameters appear in several locations, this is where they are seen first.

There is an option in kinematics.m to use symbolic or numeric DH parameters. The GUI (astepgui.m) does not allow for this option. If there is desire to see the matrices displayed in symbolic form, it must be accessed through kinematics. To use this feature, set Line 11 to 1 (variable *sym* to logical true).

After receiving parameters from astepgui.m, kinematics.m communicates with inv_kin.m, TransformMat.m, TransformW.m, TransformPos.m, Outputs.m, armplot.m, and work.m.

inv_kin.m

inv_kin.m receives the user-input sample container positions and orientations. The function applies inverse kinematics in the manner described in Chapter 7 and returns the joint angles to kinematics.m.

TransformMat.m

TransformMat.m receives the DH parameters from kinematics.m and calculates the transformation matrices based on these values. After calculating a transformation matrix, it breaks it up into a rotation matrix and a position vector. It returns these matrices and vector to kinematics.m. It can perform the calculations in either symbolic or numeric form.

kinematics.m calls TransformMat.m in a for loop. Thus, matrices are computed joint to joint, all throughout the manipulator.

TransformW.m

TransformW.m receives the individual rotation matrices from kinematics.m and uses them to calculate the cumulative rotation parameters at each arm position. It then returns the cumulative rotation matrices (now called W matrices) to kinematics.m.

TransformPos.m

TransformPos.m receives the W rotation matrices determined by TransformW.m and position vectors calculated by TransformMat.m. Those original positions were local. To make them absolute, the cumulative W matrices are

multiplied by individual position vectors to get the vector change in position from one frame to the next. These vectors are added together to get the absolute position of each frame. The changes in position and absolute positions are returned to kinematics.m.

Outputs.m

Outputs.m takes all of the calculated parameters (theta array, T matrices, R matrices, p vectors, and other calculated arrays) along with output control settings from kinematics.m. Outputs.m will then output the desired matrices to the command window. Assuming symbolic variables are not being used, it will also generate a data file called MatrixData.dat, which gets saved in the active directory. The data file contains the joint angles, rotation matrices, position vectors, and the additional important matrices. The data file gets created automatically, regardless of user inputs.

armplot.m

armplot.m is one of the most complicated of all the kinematics functions. It reads in the joint angles, transformation matrices, and plot settings from kinematics.m. After redefining the DH parameters, it progresses down the arm using the transformation matrices. The matrices are used to determine the next significant point, which is not necessarily the next coordinate frame. Using the current point and the next one, the script connects the two with a line, and plots a cross at each of the points. Changes in the user-input joint angles will produce different transformation matrices, which will generate different arm positions.

armplot.m contains numerous if/else statements. These are included to account for any possible configuration. For example, in one configuration Frame 4 may have a greater x-coordinate than Frame 2, but in another situation, the case may be reversed. A single array in MATLAB will not account for both situations and will produce an error for one of the scenarios. The if/else statements allow for all possibilities.

At the end of the program, armplot.m plots an isometric image of the arm in addition to 2D models in all planes, producing four total plots. If *spread* is selected on the GUI, it will then space the four plots out around the screen.

work.m

work.m is the function designed to plot the SAMURAI workspace. The majority of the file functions similarly to armplot.m, but work.m contains embedded loops and calculates the tool tip position at the end of the work envelope. These positions are stored in an array, which is continually growing. When the loops have been completed, the points are graphed in the same plots generated by armplot.m. The resolution of the workspace depends on the number of iterations, but with embedded loops, a very high resolution may require significant computing time.

TransformWork.m

TransformWork.m serves an identical function to TransfromMat.m, but it performs strict numeric calculations (no symbolic expressions) and does not disassemble the transformation matrices into position vectors and rotation matrices. This function was created in the interest of computing efficiency.

R03testf.m

R03testf.m is used to find the rotation matrix from Frame 0 to Frame 3 for use in the inverse kinematics function. It receives the DH parameters and joint angles for Joints 1, 2, and 3 and returns ${}^{0}_{3}R$ to inv_kin.m.

sample_container.m

sample_container.m receives the user-input sample container position and orientation and combines these values with the known sample container geometry to generate numeric representations of the cylinders as well as the location of the upper surface. The numeric cylinder representations are ultimately plotted using armplot.m. The location of the upper surface is used for the manipulator tool tip position if the user selects the sample container pre-insertion option.

H.1 Function astepgui.m

```
function varargout = astepgui(varargin)
% ASTEPGUI M-file for astepgui.fig
%      ASTEPGUI, by itself, creates a new ASTEPGUI or raises the existing
%      singleton*.
%
%      H = ASTEPGUI returns the handle to a new ASTEPGUI or the handle to
%      the existing singleton*.
%
%      ASTEPGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ASTEPGUI.M with the given input arguments.
%
%      ASTEPGUI('Property','Value',...) creates a new ASTEPGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs
%      are
%      applied to the GUI before astepgui_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to astepgui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Copyright 2002-2003 The MathWorks, Inc.
% Edit the above text to modify the response to help astepgui
% Last Modified by GUIDE v2.5 15-Aug-2008 19:39:25
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @astepgui_OpeningFcn, ...
                   'gui_OutputFcn',  @astepgui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

%set(figure(astepgui), 'units', 'normalized', 'outerposition', [.25 .25 .15 .375])
%set(figure(astepgui), 'position', [.25 .25 .15 .375])

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before astepgui is made visible.
function astepgui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to astepgui (see VARARGIN)

% Choose default command line output for astepgui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes astepgui wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = astepgui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Forward kinematics inputs (joint_angles)
theta_1_deg = handles.edit1;
theta_2_deg = handles.edit2;
theta_3_deg = handles.edit5;
theta_4_deg = handles.edit6;
theta_5_deg = handles.edit7;
theta_6_deg = handles.edit8;

%Workspace joint angle iteration size
workspace_iter = handles.edit9;

%Inverse kinematics inputs (sample container pos. and orientatation)
x_sc = handles.edit10;
y_sc = handles.edit11;
z_sc = handles.edit12;
pitch_sc_deg = handles.edit13;
roll_sc_deg = handles.edit14;
yaw_sc_deg = handles.edit15;


%Use if statements to account for intial case when GUI outputs strings as substitutes for logic values
%We want the following varialbes to be 0 or 1, and the if statements make this happen
if handles.checkbox1 ~= 1,              %Display rotation matrices in CW
   rot = 0;
else
   rot = handles.checkbox1;
end
if handles.checkbox2 ~= 1,                              %Display position matrices in CW
   pos = 0;    else    pos = handles.checkbox2;    end
if handles.checkbox3 ~= 1,                               %Display transformation matrices in CW
   trans = 0;  else    trans = handles.checkbox3;  end
if handles.checkbox4 ~= 1,                              %Display all relevant matrices in CW
   disp = 0;   else    disp = handles.checkbox4;   end
if handles.checkbox5 ~= 1,                              %Display "important" matrices in CW
                                                        %Matrices include P04, R03, and R36
   mat_impt = 0;   else    mat_impt = handles.checkbox5;   end
if handles.checkbox6 ~= 1,                      %Spreads plots across the desktop, making them all visible at once
   spread = 0;     else    spread = handles.checkbox6;     end
if handles.checkbox7 ~= 1,                              %Hides the upper Jaguar clyinder and the support beam
                                                        %Cylinder sometimes obscures the plot
jaguar = 0;         else    jaguar = handles.checkbox7;     end
if handles.checkbox8 ~= 1,                              %Uses plot settings for laptop (single window)
   laptop = 0;     else    laptop = handles.checkbox8;     end
if handles.checkbox9 ~= 1,                              %Uses plot settings for iMac (dual monitors)
   imac = 0;       else    imac = handles.checkbox9;       end
if handles.checkbox11 ~= 1,                             %Activate workspace plot
   workspace = 0;  else    workspace = handles.checkbox11; end
if handles.checkbox12 ~= 1,                             %Activate SAMURAI plots
   plots = 0;      else    plots = handles.checkbox12;     end
if handles.checkbox13 ~= 1,                             %Activate multi-colored links in plots
   colors = 0;     else    colors = handles.checkbox13;    end
if handles.checkbox16 ~= 1,                             %Display sample containers coords. in CW
   containers = 0; else    containers = handles.checkbox16;    end
if handles.checkbox17 ~= 1,                             %Display tool tip position in CW
```

```
    tool_tip = 0;   else    tool_tip = handles.checkbox17;      end
if handles.checkbox18 ~= 1,                             %Activate inverse kinematics
    inverse = 0;   else     inverse = handles.checkbox18;      end
if handles.checkbox19 ~= 1,                             %Display Joint Angles
    ang_disp = 0;   else    ang_disp = handles.checkbox19;     end
if handles.checkbox20 ~= 1,                             %Display SAMURAI After Sample Container Insertion
    insert = 0;   else    insert = handles.checkbox20;     end
if handles.checkbox21 ~= 1,                             %Display SAMURAI Pre-Sample Container Insertion
    pre_insert = 0;   else    pre_insert = handles.checkbox21;     end

%Send variables and logic values to kinematics code (primary code)
kinematics(theta_1_deg, theta_2_deg, theta_3_deg, theta_4_deg, theta_5_deg,...
    theta_6_deg, plots, rot, pos, trans, disp, mat_impt, spread, jaguar, laptop, imac,...
    workspace, workspace_iter, colors, containers, tool_tip,...
    x_sc, y_sc, z_sc, pitch_sc_deg, roll_sc_deg, yaw_sc_deg, inverse, ang_disp,...
    pre_insert, insert)

%%%Theta 1%%%
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit1 = NewVal;
guidata(hObject, handles);
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%%%Theta 2%%%
function edit2_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit2 = NewVal;
guidata(hObject, handles);
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%%%Theta 3%%%
function edit5_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit5 = NewVal;
guidata(hObject, handles);
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


%%%Theta 4%%%
function edit6_Callback(hObject, eventdata, handles)
```

```
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit6 = NewVal;
guidata(hObject, handles);
function edit6_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%%%Theta 5%%%
function edit7_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit7 = NewVal;
guidata(hObject, handles);
function edit7_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%%%Theta 6%%%
function edit8_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit8 = NewVal;
guidata(hObject, handles);
function edit8_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


%%%Activate Manipulator Plots%%%
% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of checkbox1
NewStrVal = get(hObject, 'Value');
handles.checkbox1 = NewStrVal;
guidata(hObject, handles);

%Display Position Matrices:
function checkbox2_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox2 = NewStrVal;  guidata(hObject, handles);

%Display Transformation Matrices:
function checkbox3_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox3 = NewStrVal;  guidata(hObject, handles);

%Display All Relevant Matrices:
function checkbox4_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox4 = NewStrVal;  guidata(hObject, handles);

%Display "Important" Matrices:
function checkbox5_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox5 = NewStrVal;  guidata(hObject, handles);

%Spread Plots Across Desktop:
```

```
function checkbox6_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox6 = NewStrVal;  guidata(hObject, handles);

%Hide the Jaguar Upper Cylinder:
function checkbox7_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox7 = NewStrVal;  guidata(hObject, handles);

%Use Plot Settings for Laptop (Windows)
function checkbox8_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox8 = NewStrVal;  guidata(hObject, handles);

%Use Plot Settings for Imac
function checkbox9_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox9 = NewStrVal;  guidata(hObject, handles);

%Activate Workspace Plots
function checkbox11_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox11 = NewStrVal; guidata(hObject, handles);

%Specify Workspace Joint Angle Iteration Size (User-Input Value)
function edit9_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit9 = NewVal;
guidata(hObject, handles);
function edit9_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%Activate SAMURAI Plots
function checkbox12_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox12 = NewStrVal; guidata(hObject, handles);

%Activate Plots with Multi-Colored Links
function checkbox13_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox13 = NewStrVal; guidata(hObject, handles);

%Display Sample Container Coordinates in CW
function checkbox16_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox16 = NewStrVal; guidata(hObject, handles);

%Diplay Tool Tip Position in CW
function checkbox17_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox17 = NewStrVal; guidata(hObject, handles);

%Sample Container X-Position:
function edit10_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit10 = NewVal;
guidata(hObject, handles);
function edit10_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
%Sample Container Y-Position:
function edit11_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit11 = NewVal;
guidata(hObject, handles);
function edit11_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%Sample Container Z-Position:
function edit12_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit12 = NewVal;
guidata(hObject, handles);
function edit12_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%Sample Container Pitch:
function edit13_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit13 = NewVal;
guidata(hObject, handles);
function edit13_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%Sample Container Roll:
function edit14_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit14 = NewVal;
guidata(hObject, handles);
function edit14_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%Sample Container Yaw:
function edit15_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'String');
NewVal = str2double(NewStrVal);
handles.edit15 = NewVal;
guidata(hObject, handles);
function edit15_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%Activate Inverse Kinematics:
% --- Executes on button press in checkbox18.
function checkbox18_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox18 = NewStrVal;  guidata(hObject, handles);
```

```
%Diplay Joint Angles in Command Window:
function checkbox19_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox19 = NewStrVal;  guidata(hObject, handles);

%Display SAMURAI at Sample Container Insertion
function checkbox20_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox20 = NewStrVal;  guidata(hObject, handles);

%Display SAMURAI Just Before Sample Container Insertion
function checkbox21_Callback(hObject, eventdata, handles)
NewStrVal = get(hObject, 'Value');
handles.checkbox21 = NewStrVal;  guidata(hObject, handles);
```

H.2 Function kinmatics.m

```
function kinematics(theta_1_deg, theta_2_deg, theta_3_deg, theta_4_deg, theta_5_deg,...
   theta_6_deg, plots, rot, pos, trans, disp, mat_impt, spread, jaguar,...
   laptop, imac, workspace, workspace_iter, colors, containers, tool_tip,...
   x_sc, y_sc, z_sc, pitch_sc_deg, roll_sc_deg, yaw_sc_deg, inverse, ang_disp,...
   pre_insert, insert)
%SAMURAI Kinematics - Central Function

%close all, clear all

%Control Settings:
sym = 0;                    %Set to 1 to use symbolic variables (0 computes actual numbers)
if sym,                     %If symbolic variables are being used, use these settings...
   disp = 0;                %Set to 1 to display ALL matrices in CW (0 allows for individual settings)
   rot = 0;                 %Set to 1 to display rotation matrices in CW (0 hides values)
   pos = 0;                 %Set to 1 to display position matrices in CW (0 hides values)
   trans = 1;               %Set to 1 to display transformation matrices (0 hides values)
   mat_impt = 0;            %Set to 1 to display "important matrices": p04, R03, R36 (0 hides values)
   containers = 0;          %Set to 1 to display sampler container coordinates (0 hides values)
   tool_tip = 0;            %Set to 1 to display tool tip coordinates (0 hides values)
   plots = 1;               %Set to 1 to display manipulator plots (0 hides plots)
   spread = 1;              %Set to 1 to spread plots across window (0 stacks plots)
   jaguar = 0;              %Set to 1 to hide Jaguar upper cylinder (0 plots it)
   axis5 = 0;               %Set to 1 to plot joint 5 axis (0 hides it)
   colors = 0;              %Set to 1 to plot manipulator in colors (0 plots in all black)
   workspace = 0;           %Set to 1 to plot manipulator workspace (0 ignores plots)
   laptop = 1;              %Set to 1 if using laptop and want to see GUI (0 if using Mac)
   imac = 0;                %Set to 1 if using Imac and want to see GUI (1 if using Windows)
   %Deactivate plot settings:
   plots = 0;  workspace = 0;  imac = 0;  laptop = 0;  theta_3_deg = 0; inverse = 0;
   pitch_sc_deg = 0;   roll_sc_deg = 0;    yaw_sc_deg = 0;
   x_sc = 0; y_sc = 0; z_sc =0;
end


%There is a 90 deg. theta rotation from Frame 2 to Frame 3
%This is being accounted for here:
theta_3_deg = theta_3_deg-90;

pitch_sc = pitch_sc_deg * pi/180;          %Convert sample container pitch to radians
roll_sc = roll_sc_deg * pi/180;            %Convert sample container roll to radians
yaw_sc = yaw_sc_deg * pi/180;              %Convert sample container yaw to radians

sc_plot = inverse;              %Create variable for sample container plotting equal to inverse
if inverse,        %If computations are to be for the inverse kinematics:
   %Return theta array obtained via inverse procedure
   [theta_1_deg, theta_2_deg, theta_3_deg, theta_4_deg, theta_5_deg, theta_6_deg]...
      = inv_kin(containers, x_sc, y_sc, z_sc, pitch_sc, roll_sc, yaw_sc,...
        pre_insert, insert, sc_plot);
end

if ~sym,
   %Assemble theta array (array of joint angles in degrees):
   theta_deg = [theta_1_deg theta_2_deg theta_3_deg theta_4_deg theta_5_deg theta_6_deg];
end


if disp,                        %If display is logical true, turn on CW outputs for all matrices
   rot = 1;    pos = 1;    trans = 1;      mat_impt = 1;
end

dof = 6;                        %Set number of degrees of freedom
nframes = dof;  %Set number of coord. frames equal to DOFs

if sym,        %If symbolic variables are being used:
   %Declaration of Symbolic Variables
```

```
   syms a0 a1 a2 a3 a4 a5
   syms d1 d2 d3 d4 d5 d6
   syms theta1 theta2 theta3 theta4 theta5 theta6

   %Table of D-H Parameters
   alpha0 = 0;        a0 = 0;    d1 = d1;          theta1 = theta1;
   alpha1 = -pi/2;    a1 = a1;   d2 = 0;           theta2 = theta2;
   alpha2 = 0;        a2 = a2;   d3 = 0;           theta3 = theta3;
   alpha3 = -pi/2;    a3 = a3;   d4 = d4;          theta4 = theta4;
   alpha4 = pi/2;     a4 = 0;    d5 = 0;           theta5 = theta5;
   alpha5 = -pi/2;    a5 = 0;    d6 = 0;           theta6 = theta6;

   theta = [theta1, theta2, theta3, theta4, theta5, theta6];      %Array of theta angles (sym)
else,          %If actual numbers are being used:
   %Table of D-H Parameters
   alpha0 = 0;        a0 = 0;          d1 = .10795;
   alpha1 = -pi/2;    a1 = .1524;      d2 = 0;
   alpha2 = 0;        a2 = .6096;      d3 = 0;
   alpha3 = -pi/2;    a3 = .1143;      d4 = .6096;
   alpha4 = pi/2;     a4 = 0;          d5 = 0;
   alpha5 = -pi/2;    a5 = 0;          d6 = 0;

   theta = theta_deg * pi/180;         %Convert theta array to radians (rad)
end

%Convert D-H Parmeters in Table into MATLAB Arrays
alpha = [alpha0, alpha1, alpha2, alpha3, alpha4, alpha5];      %Create array of alpha angles (rad)
a = [a0, a1, a2, a3, a4, a5];                  %Array of "a" offset vectors
d = [d1, d2, d3, d4, d5, d6];                  %Array of "d" offset vectors

for i = 1:nframes,
   %TransformMat computes transformation, rotation, and position matrices
   [T(:,:,i), R(:,:,i), P] = TransformMat(a(i),alpha(i),d(i),theta(i), sym);   %Transformations from frame i to i-1
   %Redefine variable names for matrices and output matrices to data file:
   if i == 1, T1 = T(:,:,i); R1 = R(:,:,i); P1 = P; end
   if i == 2, T2 = T(:,:,i); R2 = R(:,:,i); P2 = P; end
   if i == 3, T3 = T(:,:,i); R3 = R(:,:,i); P3 = P; end
   if i == 4, T4 = T(:,:,i); R4 = R(:,:,i); P4 = P; end
   if i == 5, T5 = T(:,:,i); R5 = R(:,:,i); P5 = P; end
   if i == 6, T6 = T(:,:,i); R6 = R(:,:,i); P6 = P; end
end

[W0, W1, W2, W3, W4, W5, W6] = TransformW(R1, R2, R3, R4, R5, R6);                    %Get W
Transformations (W(i) = W(i-1)*R(i)
[dx1, dx2, dx3, dx4, x0, x1, x2, x3, x4] = TransformPos(W0, W1, W2, W3, W4, W5, P1, P2, P3, P4, P5, P6);   %Get
positions of coordinate frames
if sym,     %Simplify function will only apply for symbolic variables
   p04 = simplify(x4);                         %Simplify trigonometric terms in x4 matrix
   R03 = simplify(W3);                         %Simplify trigonometric terms in W3 matrix
   R36 = simplify(W6);                         %Simplify trigonometric terms in W6 matrix
else       %If actual values are being used...
   p04 = x4;                                   %MATLAB will simplify automatically
   R03 = W3;                                   %MATLAB will simplify automatically
   R36 = W6;                                   %MATLAB will simplify automatically
end

%Function Outputs contains controls to output data to command window and data file
if sym,     %If symbolic variables are being used, set the numeric arrays to arbitrary values
         %This allows function "Outputs" to be activated and the desired arrays to be displayed.
   theta_deg = [0 0 0 0 0 0];          p04 = [0; 0; 0];
   R03 = [0 0 0; 0 0 0; 0 0 0];        R36 = [0 0 0; 0 0 0; 0 0 0];
   p0T = [0; 0; 0];                    R0T = [0 0 0; 0 0 0; 0 0 0];
   ang_disp = 0;
end

if ~sym,
   %Access manipultor plotting function
   close(figure(2), figure(3), figure(4), figure(5))      %Closes any figures that may be open from previous run
   [p0T, R0T] = armplot(theta_deg, spread, jaguar, T1, T2, T3, T4, T5, T6,...
   laptop, imac, colors, containers, plots, tool_tip,...
```

```
   x_sc, y_sc, z_sc, pitch_sc, roll_sc, yaw_sc, sc_plot);
end

%Send all data to function Outputs, which controls matrices output to the...
%command window and numeric matrices in the data file
Outputs(theta_deg, R1, R2, R3, R4, R5, R6, P1, P2, P3, P4, P5, P6, T1, T2, T3, T4, T5, T6,...
      p04, R03, R36, p0T, R0T, rot, pos, trans, mat_impt, sym, ang_disp)


%If workspace plots are desired:
if workspace,
   close(figure(6), figure(7), figure(8), figure(9))      %Closes any figures that may be open from previous run
   work(spread, jaguar, laptop, imac, workspace_iter, nframes),   end

%Fix the position of the GUI interface
if imac,        %Settings for a dual monitor Mac setup
   set(figure(astepgui), 'units', 'normalized', 'outerposition', [.4285 .75 .25 .25])
end
if laptop,      %Settings for a single monitor Windows setup
   set(figure(astepgui), 'units', 'normalized', 'outerposition', [0 .52 .666667 .485])
end
```

H.3 Function inv_kin.m

```
function [theta_1_deg, theta_2_deg, theta_3_deg, theta_4_deg, theta_5_deg, theta_6_deg]...
   = inv_kin(containers, x_sc, y_sc, z_sc, pitch_sc, roll_sc, yaw_sc, pre_insert, insert, sc_plot)

  %Table of D-H Parameters
  alpha0 = 0;      a0 = 0;       d1 = .10795;
  alpha1 = -pi/2;  a1 = .1524;   d2 = 0;
  alpha2 = 0;      a2 = .6096;   d3 = 0;
  alpha3 = -pi/2;  a3 = .1143;   d4 = .6096;
  alpha4 = pi/2;   a4 = 0;       d5 = 0;
  alpha5 = -pi/2;  a5 = 0;       d6 = 0;

%Convert D-H Parmeters in Table into MATLAB Arrays
alpha = [alpha0, alpha1, alpha2, alpha3, alpha4, alpha5];      %Create array of alpha angles (rad)
a = [a0, a1, a2, a3, a4, a5];                                  %Array of "a" offset vectors
d = [d1, d2, d3, d4, d5, d6];                                  %Array of "d" offset vectors

joint6_length = 0.2171;                                        %Joint 5 axis to face of hand roll joint (m)
ee_length = .2237;                                             %End-effector length (m)

nframes = 6;       sym = 1;

[X_sc, Y_sc, Z_sc, C_sc, X2_sc, Y2_sc, Z2_sc, X3_sc, Y3_sc, Z3_sc,...
   X4_sc, Y4_sc, Z4_sc, X_lower, X_upper, Y_lower, Y_upper, Z_lower, Z_upper] = ...
   sample_container(containers, x_sc, y_sc, z_sc, pitch_sc, roll_sc, yaw_sc, sc_plot);

   %Set tool offset from Frames 4,5,6
   pTool = [0; 0; joint6_length + ee_length];                  %Tool offset in tool frame (m)
   if pre_insert,  %SAMURAI just before entering sample containers
      p0Tval = [X_upper; Y_upper; Z_upper];                    %User-input location of tool tip (m)
   else          %SAMURAI inside of sample containers
      p0Tval = [X_lower; Y_lower; Z_lower];                    %User-input location of tool tip (m)
   end

   %From sample container calculations:
   R_yaw = [cos(yaw_sc) -sin(yaw_sc) 0;...                     %Rotation matrix for sample container yaw (Frame 0 z-axis)
         sin(yaw_sc) cos(yaw_sc) 0;...
         0 0 1];
   R_pitch = [cos(pitch_sc) 0 sin(pitch_sc);...                %Rotation matrix for sample container pitch (Frame 0 y-axis)
            0 1 0;...
            -sin(pitch_sc) 0 cos(pitch_sc)];
   R_roll = [1 0 0;...                                         %Rotation matrix for sample container roll (Frame 0 x-axis)
           0 cos(roll_sc) -sin(roll_sc);...
           0 sin(roll_sc) cos(roll_sc)];
   Rsc = R_yaw*R_pitch*R_roll;                                 %Combine rotataion matrices into one matrix - note the order...
   R0Tval = Rsc;                                               %Redefine rotation matrix as sample container rotation

   %%%Inverse Kinematics%%%
   p04val = p0Tval + R0Tval*pTool;                             %Position of Frame 4 relative to Frame 0

   %Find theta1:
   theta1 = atan2(p04val(2,1),p04val(1,1));                    %atan(y/x) of p04, (rad)

   %Find theta2 and theta3:
   L1 = a2;                                                    %Distance from frame 2 to frame 3, Link 1 (m)
   L2 = sqrt(a3^2 + d4^2);                                     %Distance from frame 3 to frame 4, Link 2 (m)
   gamma = atan(a3/d4);                                        %Fixed angle between frames 3 and 4 (rad)

   pos0 = [0; 0; 0];                                           %Position of frame 0, fixed (m)
   pos2 = [a1*cos(theta1); a1*sin(theta1); d1];                %Position of frame 2 (m)
                                          %Note: Pos. of frame 2 is only dependent upon theta1, joint 1 yaw
   p02 = pos2 - pos0;                     %Vector distance between frames 0 and 2 (m)
   p24val = p04val - p02;                 %Vector distance between frames 2 and 4 (m)
   r_xy = sqrt(p24val(1,1)^2 + p24val(2,1)^2);    %Scalar distance between frames 2 and 4 in the x-y plane (m)
   r_z = -p24val(3,1);                    %Scalar distance between frames 2 and 4 and in the z-dir (m)
   r = sqrt(r_xy^2+r_z^2);                %Total scalar distance between frames 2 and 4 (m)
```

```
    t3prime = acos((r^2-L1^2-L2^2)/(-2*L1*L2));     %Angle in triangle formed by L1, L2, and vector between...
                                                    %frames 2 and 4 obtained w/ law of cosines (rad)

    theta3 = pi + gamma - t3prime;                  %Theta3 after adding offsets (rad)


    psi = acos((r^2 + L1^2 - L2^2)/(2*L1*r));       %Angle between L1 and vector between frames 2 and 4 (rad)
                                                    %Again, obtained via law of cosines
    beta = atan2(r_z, r_xy);                %Pitch angle between frames 2 and 4, see determination of theta2 (rad)
    theta2 = beta - psi;                    %Subtract pitch due to pitching of joint 3 to find theta2 (rad)

    t1 = theta1;        %Redefine t1soln as t1
    t2 = theta2;        %Redefine t2soln as t2
    t3 = theta3;        %Redefine t3soln as t3

    %Solve for wrist joints 4-6
    t3 = t3 - pi/2;                         %Adjust -pi/2 rotation to Frame 3 in DH Parameters (rad)
    theta_inv = [t1, t2, t3, 0, 0, 0];      %Create new theta array for solution purposes
                                            %Only the 1st 3 values are of interest (at this point)
                                            %The last 3 are place holders to enable function use
    R03 = R03testf(theta_inv, a, alpha, d, sym, nframes);  %Send data to function R03testf
                                            %R03testf is the function that determines R03test

    R3T = inv(R03) * -R0Tval;   %Find rotation matrix (R) from Frame 3 to point to the tool tip (T)
                        %Note: R03 is an orthonormal matrix, making its transpose and inverse identical matrices
                        %R3T is also an orthonormal matrix
                        %R0T is multiplied by -1 because we previously pointed...
                        %from the tool frame to  Frame 4.  Now we're doing the opposite.
s5 = -sqrt((R3T(2,1))^2+(R3T(2,2))^2);
c5 = R3T(2,3);
theta5 = atan2(s5,c5);
theta4 = atan2(R3T(3,3)/s5, -R3T(1,3)/s5);
theta6 = atan2(R3T(2,1)/s5, -R3T(2,2)/s5);

t4 = theta4;    t5 = theta5;     t6 = theta6;

    theta = [t1; t2; t3; t4; t5; t6];       %Assemble array of modified theta's
    theta_deg = theta * 180/pi;          %Convert theta2 from radians to degrees

%Separate variables for return to function kinematics:
    theta_1_deg = theta_deg(1,1);   theta_2_deg = theta_deg(2,1);
    theta_3_deg = theta_deg(3,1);   theta_4_deg = theta_deg(4,1);
    theta_5_deg = theta_deg(5,1);   theta_6_deg = theta_deg(6,1);
    R0T = R0Tval;      p0T = pTool;
```

H.4 Function TransformMat.m

```
function [T R P] = TransformMat(a, alpha, d, theta, sym)
% Returns T transform matrix for manipulator kinematics per
% Craig eq. 3.6 (Intro. to Robotics, 3rd ed.)


if sym                          %If symbolic variables are being used, the "round" function applies...
   %Row 1 of tranformation matrix
   T(1,1) = cos(theta);
   T(1,2) = -sin(theta);
   T(1,3) = 0;
   T(1,4) = a;
   %Row 2 of transformation matrix
   T(2,1) = sin(theta)*round(cos(alpha));
   T(2,2) = cos(theta)*round(cos(alpha));
   T(2,3) = round(-sin(alpha));
   T(2,4) = round(-sin(alpha))*d;
   %Row 3 of transformation matrix
   T(3,1) = sin(theta)*round(sin(alpha));
   T(3,2) = cos(theta)*round(sin(alpha));
   T(3,3) = round(cos(alpha));
   T(3,4) = round(cos(alpha))*d;
   %Row 4 of transformation matrix
   T(4,1) = 0;
   T(4,2) = 0;
   T(4,3) = 0;
   T(4,4) = 1;

else                            %If actual numbers are to be computed...
   %Row 1 of tranformation matrix
   T(1,1) = cos(theta);
   T(1,2) = -sin(theta);
   T(1,3) = 0;
   T(1,4) = a;
   %Row 2 of transformation matrix
   T(2,1) = sin(theta)*cos(alpha);
   T(2,2) = cos(theta)*cos(alpha);
   T(2,3) = -sin(alpha);
   T(2,4) = -sin(alpha)*d;
   %Row 3 of transformation matrix
   T(3,1) = sin(theta)*sin(alpha);
   T(3,2) = cos(theta)*sin(alpha);
   T(3,3) = cos(alpha);
   T(3,4) = cos(alpha)*d;
   %Row 4 of transformation matrix
   T(4,1) = 0;
   T(4,2) = 0;
   T(4,3) = 0;
   T(4,4) = 1;
end

T = [T(1,1) T(1,2) T(1,3) T(1,4);...
    T(2,1) T(2,2) T(2,3) T(2,4);...
    T(3,1) T(3,2) T(3,3) T(3,4);...
    T(4,1) T(4,2) T(4,3) T(4,4)];

%Rotation Matrices
%Row 1 of rotation matrix
r11 = T(1,1);
r12 = T(1,2);
r13 = T(1,3);
%Row 2 of rotation matrix
r21 = T(2,1);
r22 = T(2,2);
r23 = T(2,3);
%Row 3 of rotation matrix
```

```
r31 = T(3,1);
r32 = T(3,2);
r33 = T(3,3);
R = [r11 r12 r13; r21 r22 r23; r31 r32 r33];

%Position Matrices/Arrays
px = T(1,4);
py = T(2,4);
pz = T(3,4);
P = [px; py; pz];
```

H.5 Function TransformW.m

```
function [W0, W1, W2, W3, W4, W5, W6] = TransformW(R1, R2, R3, R4, R5, R6)

W0 = eye(3);          %Set base frame W matrix equal to identity matrix

W1 = W0*R1;           %Rotation to Frame 1
W2 = W1*R2;           %Rotation to Frame 2
W3 = W2*R3;           %Rotation to Frame 3
W4 = R4;              %Frame 4
W5 = W4 * R5;         %Rotation to Frame 5
W6 = W5 * R6;         %Rotation to Frame 6
```

H.6 Function TransformPos.m

```
function [dx1, dx2, dx3, dx4, x0, x1, x2, x3, x4] = TransformPos(W0, W1, W2, W3, W4, W5, P1, P2, P3, P4, P5, P6)

dx1 = W0 * P1;      %Change in position of baseframe to frame 1
dx2 = W1 * P2;      %Change in position of frame 1 to frame 2
dx3 = W2 * P3;      %Change in position of frame 2 to frame 3
dx4 = W3 * P4;      %Change in position of frame 3 to frame 4

%Absolute Positions of the Coordinate Frames
x0 = [0; 0; 0];     %Define position of base frame
x1 = x0 + dx1;      %Frame 1
x2 = x1 + dx2;      %Frame 2
x3 = x2 + dx3;      %Frame 3
x4 = x3 + dx4;      %Frame 4
```

H.7 Function Outputs.m

```
function Outputs(theta_deg, R1, R2, R3, R4, R5, R6, P1, P2, P3, P4, P5, P6, T1, T2, T3, T4, T5, T6,...
            p04, R03, R36, p0T, R0T, rot, pos, trans, mat_impt, sym, ang_disp)


%Command Window Output Commands:
%Display Rotation, Position, and Transformation Matrices if Desired
if rot,
   fprintf('\nROTATION MATRICES:\n')
   R1, R2, R3, R4, R5, R6
end
if pos,
   fprintf('POSITION MATRICES:\n')
   P1, P2, P3, P4, P5, P6
end
if trans,
   fprintf('\nTRANSFORMATION MATRICES:\n')
   T1, T2, T3, T4, T5, T6
end
if mat_impt,
   fprintf('\nIMPORTANT MATRICES:\n')
   p04, R03, R36, p0T, R0T
end
%In kinematics and/or inv_kin, 90 deg. were subtracted from theta3 to account for the...
%frame rotation.  These must be put back before displaying the user-input joint angle:
theta_deg(3) = theta_deg(3) + 90;                    %Add the 90 degrees to theta_3_deg

if ang_disp,
   fprintf('\nSAMURAI JOINT ANGLES:\n')
   fprintf('Theta 1: %.4g deg.\n', theta_deg(1))
   fprintf('Theta 2: %.4g deg.\n', theta_deg(2))
   fprintf('Theta 3: %.4g deg.\n', theta_deg(3))
   fprintf('Theta 4: %.4g deg.\n', theta_deg(4))
   fprintf('Theta 5: %.4g deg.\n', theta_deg(5))
   fprintf('Theta 6: %.4g deg.\n', theta_deg(6))
end

%Data File
if ~sym,                                    %Data file cannot be written for symbolic variables
   MatDat = fopen('MatrixData.dat','wt');   %Create Data File for Storage of Matrix Data
   fprintf(MatDat, 'Theta = \n');           %Output character string "Theta =" to data file
   fprintf(MatDat, '%13.1f \n', theta_deg); %Output theta array 13 spaces from left margin & 1 sig. fig.
   textR = 'R1 = ';                         %Create character string and assign string to variable textR
   textP = 'P1 = ';                         %Create character string and assign string to variable textP
   fprintf(MatDat, '\n');                   %Insert new line in the data file
   fprintf(MatDat, '%21s', textR); %Print the character string inside the data file with 21 spaces ("s" denotes string)
   fprintf(MatDat,'%31s', textP);  %Print the character string inside the data file with 31 spaces ("s" denotes string)
   fprintf(MatDat, '\n');                   %Insert another new line in the data file
   for k = 1:size(R1,1),                    %For every k where k is an integer between 1 and size(R1,1)
                                            %size(R1,1) = 3, it's the number of rows in column 1
      fprintf(MatDat,'%28.4f %7.4f %7.4f',R1(k,:));
         %The ".4" values signify 4 sig. figs.
         %The 28 inserts 28 spaces between the left column and the right-most sig. fig.
         %The 7's insert 7 between between the previous column and the right-most sig. fig.
         %When k=1, this loop prints the first row and all columns to the data file
         %This is then repeated when k = 2 and 3
      fprintf(MatDat,'%15.4f \n',P1(k,:));  %Prints 1st row of P1 & inserts 15 spaces between left col. & right sig. fig.
   end

   %Same Process for R2
   fprintf(MatDat,'\n'); textR = 'R2 = '; textP = 'P2 = '; fprintf(MatDat, '%21s', textR); fprintf(MatDat,'%31s', textP);
   fprintf(MatDat, '\n'); for k = 1:size(R2,1), fprintf(MatDat,'%28.4f %7.4f %7.4f',R2(k,:)); fprintf(MatDat,'%15.4f
\n',P2(k,:)); end
   %Same Process for R3
   fprintf(MatDat,'\n'); textR = 'R3 = '; textP = 'P3 = '; fprintf(MatDat, '%21s', textR); fprintf(MatDat,'%31s', textP);
```

```
   fprintf(MatDat, '\n'); for k = 1:size(R3,1), fprintf(MatDat,'%28.4f %7.4f %7.4f',R3(k,:)); fprintf(MatDat,'%15.4f
\n',P3(k,:)); end
   %Same Process for R4
   fprintf(MatDat,'\n'); textR = 'R4 = '; textP = 'P4 = '; fprintf(MatDat, '%21s', textR); fprintf(MatDat,'%31s', textP);
   fprintf(MatDat, '\n'); for k = 1:size(R4,1), fprintf(MatDat,'%28.4f %7.4f %7.4f',R4(k,:)); fprintf(MatDat,'%15.4f
\n',P4(k,:)); end
   %Same Process for R5
   fprintf(MatDat,'\n'); textR = 'R5 = '; textP = 'P5 = '; fprintf(MatDat, '%21s', textR); fprintf(MatDat,'%31s', textP);
   fprintf(MatDat, '\n'); for k = 1:size(R5,1), fprintf(MatDat,'%28.4f %7.4f %7.4f',R5(k,:)); fprintf(MatDat,'%15.4f
\n',P5(k,:)); end
   %Same Process for R6
   fprintf(MatDat,'\n'); textR = 'R6 = '; textP = 'P6 = '; fprintf(MatDat, '%21s', textR); fprintf(MatDat,'%31s', textP);
   fprintf(MatDat, '\n'); for k = 1:size(R6,1), fprintf(MatDat,'%28.4f %7.4f %7.4f',R6(k,:)); fprintf(MatDat,'%15.4f
\n',P6(k,:)); end


   %Important Data Matrices:
   fprintf(MatDat, '\nImportant Data:\n\n');
   %Similar Process for P04 and P0T
   fprintf(MatDat,'\n'); textP = 'P04 = '; fprintf(MatDat,'%0s', textP);
   textP = 'P0T = '; fprintf(MatDat,'%20s', textP);
   fprintf(MatDat, '\n');
   for k = 1:size(p04,1),                          %Position matrices are the same size
      fprintf(MatDat,'%13.4f',p04(k,:));
      fprintf(MatDat,'%20.4f \n',p0T(k,:));
   end

   %Similar Process for R03, R36, and R0T
   fprintf(MatDat,'\n\n'); textR = 'R03 = '; fprintf(MatDat, '%0s', textR);
                  textR = 'R36 = '; fprintf(MatDat, '%35s', textR);
                  textR = 'R0T = '; fprintf(MatDat, '%35s', textR);
   fprintf(MatDat, '\n');
   for k = 1:size(R03,1),                          %Rotation matrices are the same size
      fprintf(MatDat,'%13.4f %7.4f %7.4f',R03(k,:));
      fprintf(MatDat,'%19.4f %7.4f %7.4f',R36(k,:));
      fprintf(MatDat,'%19.4f %7.4f %7.4f \n',R0T(k,:));
   end

   fclose(MatDat);                                 %Close Data File
end                                                %End of if ~sym statement
```

H.8 Function armplot.m

```
function [p0T, R0T] = armplot(theta_deg, spread, jaguar, T1, T2, T3, T4, T5, T6,...
   laptop, imac, colors, containers, plots, tool_tip,...
   x_sc, y_sc, z_sc, pitch_sc, roll_sc, yaw_sc, sc_plot)

   %Table of D-H Parameters
   alpha0 = 0;        a0 = 0;          d1 = .10795;
   alpha1 = -pi/2;    a1 = .1524;      d2 = 0;
   alpha2 = 0;        a2 = .6096;      d3 = 0;
   alpha3 = -pi/2;    a3 = .1143;      d4 = .6096;
   alpha4 = pi/2;     a4 = 0;          d5 = 0;
   alpha5 = -pi/2;    a5 = 0;          d6 = 0;

   theta = theta_deg * pi/180;         %Convert theta array to radians (rad)

   joint6_length = 0.2171;             %Joint 5 axis to face of hand roll joint (m)
   ee_length = .2237;                  %End-effector length (m)

res = .0001;                           %Resolution off increments composing lines
                                       %This must be sufficiently small to coincide
                                       %with values in kinematics codes
line_width = 2;                        %Set width of plotted lines

fig_color = [1 1 1];                   %Sets background in figures to white
set(0, 'DefaultFigureColor', fig_color)
title_size = 14;                       %Set the size of the font for plot text (26 for maximized windows)
xmin = -1.5; xmax = 1.75; ymin = -1.5; ymax = 1.5; zmin = -1.5; zmax = 1.5;   %Values for axes definitions

%Jaguar Details
radius = 0.1984;                       %Specify radius of Jaguar cylinder (m)
height = abs(xmin);                    %"height" or length of cylinder (m)
strut_loc = -.42;                      %x-displacement of Jaguar strut from base frame (0,0,0) in meters
strut_width = .12;                     %Strut width (m)
Jag_offset = 1.194;                    %Distance between Jaguar cylinder axes (m)


%Note: 3D Plot commands will be moved inside the "colors" if statement
%They will just be commented out with double %% though, so that their original locations are known
      %%figure(2)
%Create a point at the origin of Frame 0:
x0_pt = 0;     y0_pt = 0;     z0_pt = 0;
      %%plot3(x0_pt, y0_pt, z0_pt, 'r+', 'LineWidth', line_width), hold on      %Plot origin of Frame 0
%Create a point at the origin of Frame 1:
x1_pt = T1(1,4);    y1_pt = T1(2,4);    z1_pt = T1(3,4);
      %%plot3(x1_pt, y1_pt, z1_pt, 'r+', 'LineWidth', line_width), hold on      %Plot origin of Frame 1
%Create line between Frames 0 and 1
if abs(x1_pt-x0_pt) >= abs(y1_pt-y0_pt) && abs(x1_pt-x0_pt) >= abs(z1_pt-z0_pt),
   x_01 = x0_pt : res : x1_pt;     %Let x-array drive change from Frame 0 to 1
   y_01 = y0_pt : abs(y1_pt-y0_pt)/(size(x_01,2)-1) : y1_pt;  %Create y-array based on size of x-array
   z_01 = z0_pt : abs(z1_pt-z0_pt)/(size(x_01,2)-1) : z1_pt;  %Create z-array based on size of x-array
   if size(y_01, 2)==1,  y_01 = y0_pt + x_01.*0;  end    %If there is no change in y, create array of same mag.
   if size(z_01, 2)==1,  z_01 = z0_pt + x_01.*0;  end    %If there is no change in z, create array of same mag.
else if abs(y1_pt-y0_pt) >= abs(x1_pt-x0_pt) && abs(y1_pt-y0_pt) >= abs(z1_pt-z0_pt),
   y_01 = y0_pt : res : y1_pt;     %Let y-array drive change from Frame 0 to 1
   x_01 = x0_pt : abs(x1_pt-x0_pt)/(size(y_01,2)-1) : x1_pt;  %Create x-array based on size of y-array
   z_01 = z0_pt : abs(z1_pt-z0_pt)/(size(y_01,2)-1) : z1_pt;  %Create z-array based on size of y-array
   if size(x_01, 2)==1,  x_01 = x0_pt + y_01.*0;  end    %If there is no change in x, create array of same mag.
   if size(z_01, 2)==1,  z_01 = z0_pt + y_01.*0;  end    %If there is no change in z, create array of same mag.
   else       %the biggest change is in the z-direction
      z_01 = z0_pt : res : z1_pt;     %Let z-array drive change from Frame 0 to 1
      x_01 = x0_pt : abs(x1_pt-x0_pt)/(size(z_01,2)-1) : x1_pt;  %Create x-array based on size of z-array
      y_01 = y0_pt : abs(y1_pt-y0_pt)/(size(z_01,2)-1) : y1_pt;  %Create y-array based on size of z-array

      if size(x_01, 2)<=1,  x_01 = x0_pt + z_01.*0;  end    %If there is no change in x, create array of same mag.
      if size(y_01, 2)<=1,  y_01 = y0_pt + z_01.*0;  end    %If there is no change in y, create array of same mag.
   end
```

```
end
      %%plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on       %Plot line between Frames 0 & 1 in 3D

%Create a point at the origin of Frame 2:
T_current = T1*T2;          %Create new transformation matrix from origin to current frame
x2_pt = T_current(1,4);    y2_pt = T_current(2,4);    z2_pt = T_current(3,4);
      %%plot3(x2_pt, y2_pt, z2_pt, 'r+', 'LineWidth', line_width), hold on       %Plot origin of Frame 2
%Create line between Frames 1 and 2
%Case I: Greatest change between frames is in x-direction:
if abs(x2_pt-x1_pt) >= abs(y2_pt-y1_pt) && abs(x2_pt-x1_pt) >= abs(z2_pt-z1_pt),
   if x2_pt >= x1_pt,                %If x2 is greater than or equal to x1, create normal array
      x_12 = x1_pt : res : x2_pt;     %Let x-array drive change from Frame 1 to 2
   else x_12 = -x1_pt : res : -x2_pt;   x_12 = -x_12;   end    %Otherwise, flip array direction.
   size_x12 = size(x_12,2)-1;          %Size of x_12 array
   if y2_pt >= y1_pt,                %If y2 is greater than or equal to y1, create normal array
      y_12 = y1_pt : abs(y2_pt-y1_pt)/size_x12 : y2_pt;  %Create y-array based on size of x-array
   else
      y_12 = -y1_pt : abs(y2_pt-y1_pt)/size_x12 : -y2_pt;   y_12 = -y_12;   %Otherwise, flip array direction
   end
   if z2_pt >= z1_pt,
      z_12 = z1_pt : abs(z2_pt-z1_pt)/size_x12 : z2_pt;  %Create z-array based on size of x-array
   else
      z_12 = -z1_pt : abs(z2_pt-z1_pt)/size_x12 : -z2_pt;   z_12 = -z_12;   %Otherwise, flip array direction
   end
   if size(y_12, 2)<=1,  y_12 = y1_pt + x_12.*0;  end    %If there is no change in y, create array of same mag.
   if size(z_12, 2)<=1,  z_12 = z1_pt + x_12.*0;  end    %If there is no change in z, create array of same mag.
%Case II: Greatest change between frames is in y-direction:
else if abs(y2_pt-y1_pt) >= abs(x2_pt-x1_pt) && abs(y2_pt-y1_pt) >= abs(z2_pt-z1_pt),
   if y2_pt >= y1_pt,                %If y2 is greater than or equal to y1,
      y_12 = y1_pt : res : y2_pt;     %Let y-array drive change from Frame 1 to 2
   else y_12 = -y1_pt : res : -y2_pt;   y_12 = -y_12;  end    %Otherwise, flip array direction.
   size_y12 = size(y_12,2)-1;          %Size of y_12 array
   if x2_pt >= x1_pt,                %If x2 is greater than or equal to x1,
      x_12 = x1_pt : abs(x2_pt-x1_pt)/size_y12 : x2_pt;  %Create x-array based on size of y-array
   else
      x_12 = -x1_pt : abs(x2_pt-x1_pt)/size_y12 : -x2_pt; x_12 = -x_12;  %Create x-array based on size of y-array
   end
   if z2_pt >= z1_pt,
      z_12 = z1_pt : abs(z2_pt-z1_pt)/size_y12 : z2_pt;  %Create z-array based on size of y-array
   else
      z_12 = -z1_pt : abs(z2_pt-z1_pt)/size_y12 : -z2_pt;   z_12 = -z_12;  %Create z-array based on size of y-array
   end
   if size(x_12, 2)<=1,  x_12 = x1_pt + y_12.*0;  end    %If there is no change in x, create array of same mag.
   if size(z_12, 2)<=1,  z_12 = z1_pt + y_12.*0;  end    %If there is no change in z, create array of same mag.
%Case III: Greatest change between frames is in z-direction:
   else       %the biggest change is in the z-direction
      if z2_pt >= z1_pt,
         z_12 = z1_pt : res : z2_pt;     %Let z-array drive change from Frame 1 to 2
      else z_12 = -z1_pt : res : -z2_pt;   z_12 = -z_12;  end    %Otherwise, flip array direction.
      size_z12 = size(z_12,2)-1;          %Size of z_12 array
      if x2_pt >= x1_pt,                %If x2 is greater than or equal to x1,
         x_12 = x1_pt : abs(x2_pt-x1_pt)/size_z12 : x2_pt;  %Create x-array based on size of z-array
      else
         x_12 = -x1_pt : abs(x2_pt-x1_pt)/size_z12 : -x2_pt;  x_12 = -x_12;  end    %Otherwise, flip array direction.
      if y2_pt >= y1_pt,                %If x2 is greater than or equal to x1,
         y_12 = y1_pt : abs(y2_pt-y1_pt)/size_z12 : y2_pt;  %Create y-array based on size of z-array
      else
         y_12 = -y1_pt : abs(y2_pt-y1_pt)/size_z12 : -y2_pt;  y_12 = -y_12;  end    %Otherwise, flip array direction.
      if size(x_12, 2)<=1,  x_12 = x1_pt + z_12.*0;  end    %If there is no change in x, create array of same mag.
      if size(y_12, 2)<=1,  y_12 = y1_pt + z_12.*0;  end    %If there is no change in y, create array of same mag.
   end
end
      %%plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on       %Plot line between Frames 1 & 2 in 3D


%Create a point at the origin of Frame 3:
T_current = T_current*T3;          %Create new transformation matrix from origin to current frame
x3_pt = T_current(1,4);    y3_pt = T_current(2,4);    z3_pt = T_current(3,4);
      %%plot3(x3_pt, y3_pt, z3_pt, 'r+', 'LineWidth', line_width), hold on       %Plot origin of Frame 3
%Create line between Frames 2 and 3
```

```
%Case I: Greatest change between frames is in x-direction:
if abs(x3_pt-x2_pt) >= abs(y3_pt-y2_pt) && abs(x3_pt-x2_pt) >= abs(z3_pt-z2_pt),
   if x3_pt >= x2_pt,                %If x3 is greater than or equal to x2, create normal array
      x_23 = x2_pt : res : x3_pt;     %Let x-array drive change from Frame 2 to 3
   else x_23 = -x2_pt : res : -x3_pt;  x_23 = -x_23;  end    %Otherwise, flip array direction.
   size_x23 = size(x_23,2)-1;         %Size of x_12 array
   if y3_pt >= y2_pt,                %If y3 is greater than or equal to y2, create normal array
      y_23 = y2_pt : abs(y3_pt-y2_pt)/size_x23 : y3_pt;  %Create y-array based on size of x-array
   else
      y_23 = -y2_pt : abs(y3_pt-y2_pt)/size_x23 : -y3_pt;   y_23 = -y_23;   %Otherwise, flip array direction.
   end
   if z3_pt >= z2_pt,
      z_23 = z2_pt : abs(z3_pt-z2_pt)/size_x23 : z3_pt;  %Create z-array based on size of x-array
   else
      z_23 = -z2_pt : abs(z3_pt-z2_pt)/size_x23 : -z3_pt;   z_23 = -z_23;   %Otherwise, flip array direction.
   end
   if size(y_23, 2)<=1,  y_23 = y2_pt + x_23.*0;  end    %If there is no change in y, create array of same mag.
   if size(z_23, 2)<=1,  z_23 = z2_pt + x_23.*0;  end    %If there is no change in z, create array of same mag.
%Case II: Greatest change between frames is in y-direction:
else if abs(y3_pt-y2_pt) >= abs(x3_pt-x2_pt) && abs(y3_pt-y2_pt) >= abs(z3_pt-z2_pt),
   if y3_pt >= y2_pt,                %If y3 is greater than or equal to y2,
      y_23 = y2_pt : res : y3_pt;     %Let y-array drive change from Frame 2 to 3
   else y_23 = -y2_pt : res : -y3_pt;   y_23 = -y_23;  end    %Otherwise, flip array direction.
   size_y23 = size(y_23,2)-1;         %Size of y_23 array
   if x3_pt >= x2_pt,                %If x2 is greater than or equal to x1,
      x_23 = x2_pt : abs(x3_pt-x2_pt)/size_y23 : x3_pt;  %Create x-array based on size of y-array
   else
      x_23 = -x2_pt : abs(x3_pt-x2_pt)/size_y23 : -x3_pt; x_23 = -x_23;  %Create x-array based on size of y-array
   end
   if z3_pt >= z2_pt,
      z_23 = z2_pt : abs(z3_pt-z2_pt)/size_y23 : z3_pt;  %Create z-array based on size of y-array
   else
      z_23 = -z2_pt : abs(z3_pt-z2_pt)/size_y23 : -z3_pt;   z_23 = -z_23;  %Create z-array based on size of y-array
   end
   if size(x_23, 2)<=1,  x_23 = x2_pt + y_23.*0;  end    %If there is no change in x, create array of same mag.
   if size(z_23, 2)<=1,  z_23 = z2_pt + y_23.*0;  end    %If there is no change in z, create array of same mag.
%Case III: Greatest change between frames is in z-direction:
   else       %the biggest change is in the z-direction
      if z3_pt >= z2_pt,
         z_23 = z2_pt : res : z3_pt;     %Let z-array drive change from Frame 2 to 3
      else z_23 = -z2_pt : res : -z3_pt;   z_23 = -z_23;  end    %Otherwise, flip array direction.
      size_z23 = size(z_23,2)-1;         %Size of z_23 array
      if x3_pt >= x2_pt,                %If x3 is greater than or equal to x2,
         x_23 = x2_pt : abs(x3_pt-x2_pt)/size_z23 : x3_pt;  %Create x-array based on size of z-array
      else
         x_23 = -x2_pt : abs(x3_pt-x2_pt)/size_z23 : -x3_pt;  x_23 = -x_23;  end    %Otherwise, flip array direction.
      if y3_pt >= y2_pt,                %If y3 is greater than or equal to y2,
         y_23 = y2_pt : abs(y3_pt-y2_pt)/size_z23 : y3_pt;  %Create y-array based on size of z-array
      else
         y_23 = -y2_pt : abs(y3_pt-y2_pt)/size_z23 : -y3_pt;  y_23 = -y_23;  end    %Otherwise, flip array direction.
      if size(x_23, 2)<=1,  x_23 = x2_pt + z_23.*0;  end    %If there is no change in x, create array of same mag.
      if size(y_23, 2)<=1,  y_23 = y2_pt + z_23.*0;  end    %If there is no change in y, create array of same mag.
   end
end
      %%plot3(x_23, y_23, z_23, 'm-', 'LineWidth', line_width), hold on       %Plot line between Frames 2 & 3 in 3D


%%%Frame 3 to Frame 4a (Bend in Link 2)%%%
l4 = 0 : res : a3;                   %Create an array based on the length of top part of Link 2
phi_sum = theta(1);                  %Summation of all phi angles to this point is just theta(1), Joint 1 yaw
x4a = x3_pt + l4*sin(pi/2+theta(2)+theta(3))*cos(phi_sum);     %pi/2 is the fixed angle between d1 and the z0 axis
y4a = y3_pt + l4*sin(pi/2+theta(2)+theta(3))*sin(phi_sum);
z4a = z3_pt + l4*cos(pi/2+theta(2)+theta(3));

      %%plot3(x4a, y4a, z4a, '-', 'LineWidth', line_width), hold on     %Plot small portion of Link 2 (Frame 3 to 4a)
x4a_pt_loc = size(x4a,2);    y4a_pt_loc = size(y4a,2);        z4a_pt_loc = size(z4a,2);
x4a_pt = x4a(x4a_pt_loc);    y4a_pt = y4a(y4a_pt_loc);    z4a_pt = z4a(z4a_pt_loc);
      %%plot3(x4a_pt, y4a_pt, z4a_pt, 'r+', 'LineWidth', line_width), hold on       %Plot point at origin of Frame 4a
```

```
%%%Frame 4a to Frame 4 (Base of Link 2)%%%
%Create a point at the origin of Frame 4:
T_current = T_current*T4;          %Create new transformation matrix from origin to current frame
x4_pt = T_current(1,4);    y4_pt = T_current(2,4);    z4_pt = T_current(3,4);
      %%plot3(x4_pt, y4_pt, z4_pt, 'r+', 'LineWidth', line_width), hold on       %Plot origin of Frame 4
%Create line between Frames 4a and 4
%Case I: Greatest change between frames is in x-direction:
if abs(x4_pt-x4a_pt) >= abs(y4_pt-y4a_pt) && abs(x4_pt-x4a_pt) >= abs(z4_pt-z4a_pt),
   if x4_pt >= x4a_pt,                %If x4 is greater than or equal to x4a, create normal array
      x_4a4 = x4a_pt : res : x4_pt;     %Let x-array drive change from Frame 4a to 4
   else x_4a4 = -x4a_pt : res : -x4_pt;  x_4a4 = -x_4a4;  end    %Otherwise, flip array direction.
   size_x4a4 = size(x_4a4,2)-1;          %Size of x_4a4 array
   if y4_pt >= y4a_pt,                %If y4 is greater than or equal to y4a, create normal array
      y_4a4 = y4a_pt : abs(y4_pt-y4a_pt)/size_x4a4 : y4_pt;  %Create y-array based on size of x-array
   else
      y_4a4 = -y4a_pt : abs(y4_pt-y4a_pt)/size_x4a4 : -y4_pt;   y_4a4 = -y_4a4;   %Otherwise, flip array direction.
   end
   if z4_pt >= z4a_pt,
      z_4a4 = z4a_pt : abs(z4_pt-z4a_pt)/size_x4a4 : z4_pt;  %Create z-array based on size of x-array
   else
      z_4a4 = -z4a_pt : abs(z4_pt-z4a_pt)/size_x4a4 : -z4_pt;   z_4a4 = -z_4a4;   %Otherwise, flip array direction.
   end
   if size(y_4a4, 2)<=1,  y_4a4 = y4_pt + x_4a4.*0;  end    %If there is no change in y, create array of same mag.
   if size(z_4a4, 2)<=1,  z_4a4 = z4_pt + x_4a4.*0;  end    %If there is no change in z, create array of same mag.
%Case II: Greatest change between frames is in y-direction:
else if abs(y4_pt-y4a_pt) >= abs(x4_pt-x4a_pt) && abs(y4_pt-y4a_pt) >= abs(z4_pt-z4a_pt),
   if y4_pt >= y4a_pt,                %If y4 is greater than or equal to y4a,
      y_4a4 = y4a_pt : res : y4_pt;     %Let y-array drive change from Frame 4a to 4
   else y_4a4 = -y4a_pt : res : -y4_pt;   y_4a4 = -y_4a4;  end    %Otherwise, flip array direction.
   size_y4a4 = size(y_4a4,2)-1;          %Size of y_4a4 array
   if x4_pt >= x4a_pt,                %If x4 is greater than or equal to x4a,
      x_4a4 = x4a_pt : abs(x4_pt-x4a_pt)/size_y4a4 : x4_pt;  %Create x-array based on size of y-array
   else
      x_4a4 = -x4a_pt : abs(x4_pt-x4a_pt)/size_y4a4 : -x4_pt;   x_4a4 = -x_4a4;  %Create x-array based on size of y-
array
   end
   if z4_pt >= z4a_pt,
      z_4a4 = z4a_pt : abs(z4_pt-z4a_pt)/size_y4a4 : z4_pt;  %Create z-array based on size of y-array
   else
      z_4a4 = -z4a_pt : abs(z4_pt-z4a_pt)/size_y4a4 : -z4_pt;   z_4a4 = -z_4a4;  %Create z-array based on size of y-
array
   end
   if size(x_4a4, 2)<=1,  x_4a4 = x4a_pt + y_4a4.*0;  end    %If there is no change in x, create array of same mag.
   if size(z_4a4, 2)<=1,  z_4a4 = z4a_pt + y_4a4.*0;  end    %If there is no change in z, create array of same mag.
%Case III: Greatest change between frames is in z-direction:
   else       %the biggest change is in the z-direction
      if z4_pt >= z4a_pt,
         z_4a4 = z4a_pt : res : z4_pt;     %Let z-array drive change from Frame 4a to 4
      else z_4a4 = -z4a_pt : res : -z4_pt;   z_4a4 = -z_4a4;  end    %Otherwise, flip array direction.
      size_z4a4 = size(z_4a4,2)-1;          %Size of z_4a4 array
      if x4_pt >= x4a_pt,                %If x4 is greater than or equal to x4a,
         x_4a4 = x4a_pt : abs(x4_pt-x4a_pt)/size_z4a4 : x4_pt;  %Create x-array based on size of z-array
      else
         x_4a4 = -x4a_pt : abs(x4_pt-x4a_pt)/size_z4a4 : -x4_pt;  x_4a4 = -x_4a4;  end    %Otherwise, flip array
direction.
      if y4_pt >= y4a_pt,                %If y4 is greater than or equal to y4a,
         y_4a4 = y4a_pt : abs(y4_pt-y4a_pt)/size_z4a4 : y4_pt;  %Create y-array based on size of z-array
      else
         y_4a4 = -y4a_pt : abs(y4_pt-y4a_pt)/size_z4a4 : -y4_pt;  y_4a4 = -y_4a4;  end    %Otherwise, flip array
direction.
      if size(x_4a4, 2)<=1,  x_4a4 = x4a_pt + z_4a4.*0;  end    %If there is no change in x, create array of same mag.
      if size(y_4a4, 2)<=1,  y_4a4 = y4a_pt + z_4a4.*0;  end    %If there is no change in y, create array of same mag.
   end
end
      %%plot3(x_4a4, y_4a4, z_4a4, '-', 'LineWidth', line_width), hold on
      %%%Plot line between Frames 4a & 4 in 3D


%%%Frame 4 to Frame 5 (Still at Base of Link 2)%%%
T_current = T_current * T5;          %Create new transformation matrix from origin to current frame
```

```
R_5 = [T_current(1,1) T_current(1,2) T_current(1,3);...     %Isolate rotataion matrix
      T_current(2,1) T_current(2,2) T_current(2,3);...
      T_current(3,1) T_current(3,2) T_current(3,3)];

%%%Create a point at the end of Joint 6%%%
l6 = 0 : res : joint6_length;       %Create an array based on the distance between Joints 5 and 6
x_change_5 = l6.*0;      y_change_5 = l6;       z_change_5 = l6.*0;      %Create point at Joint 6 face
   %Note: Point is offset from the frame in the y_5 direction
pos_5 = [x_change_5; y_change_5; z_change_5];                    %Put coordinates in a vector array
pos_6 = R_5 * pos_5;                                             %Rotate vector based on rotation matrix

x_56 = x4_pt + pos_6(1,:);  y_56 = y4_pt + pos_6(2,:);  z_56 = z4_pt + pos_6(3,:);      %Add new location to previous
location (x4, y4, z4)

      %%plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on              %Plot connection between Joints 5
and 6
x_56_pt_loc = size(x_56,2);    y_56_pt_loc = size(y_56,2);    z_56_pt_loc = size(z_56,2);
x_56_pt = x_56(x_56_pt_loc);    y_56_pt = y_56(y_56_pt_loc);    z_56_pt = z_56(z_56_pt_loc);
      %%plot3(x_56_pt, y_56_pt, z_56_pt, 'r+', 'LineWidth', line_width), hold on       %Plot end of Joint 6

%%Create a point at the end of the End-Effector (the tool tip)%%%
joint_7_length = ee_length;         %This is the end-effector length from Joint 6 to tool tip (meters)
l_7 = 0 : res : joint_7_length;     %Create an array based on the distance between Joints 5 and 6

x_change_7 = l_7;      y_change_7 = l_7.*0;       z_change_7 = l_7.*0;   %Create end point corresponding to alpha =
beta = gamma = 0
pos_7 = R_5 * pos_5;              %pos_7 is just an extension of pos_6
                          %=> use same rotation matrix
x_67 = x_56_pt + pos_7(1,:);  y_67 = y_56_pt + pos_7(2,:);  z_67 = z_56_pt + pos_7(3,:);

      %%plot3(x_67, y_67, z_67, 'g-', 'LineWidth', line_width), hold on     %Plot connection between Joints 6 and EE
x_67_pt_loc = size(x_67,2);    y_67_pt_loc = size(y_67,2);    z_67_pt_loc = size(z_67,2);
x_67_pt = x_67(x_67_pt_loc);    y_67_pt = y_67(y_67_pt_loc);    z_67_pt = z_67(z_67_pt_loc);
      %%plot3(x_67_pt, y_67_pt, z_67_pt, 'r+', 'LineWidth', line_width), hold on       %Plot end of the EE

if tool_tip,               %If tool tip coordiantes are to be output to CW:
   fprintf('\nTool Tip Coordinates:\n')
   fprintf('   X-Coordinate: %.4g m\n', x_67_pt)
   fprintf('   Y-Coordinate: %.4g m\n', y_67_pt)
   fprintf('   Z-Coordinate: %.4g m\n', z_67_pt)
end

p0T = [x_67_pt; y_67_pt; z_67_pt];      %Vector from Frame 0 to tool tip (m)
T_current = T_current * T6;            %Final transformation matrix
R0T = [T_current(1,1) T_current(1,2) T_current(1,3);...     %Isolate rotataion matrix
      T_current(2,1) T_current(2,2) T_current(2,3);...
      T_current(3,1) T_current(3,2) T_current(3,3)];


%%%Create Jaguar Base%%%
R = [radius radius];                        %Create array with x = radius and y = radius
N = 25;                                     %Number of mesh segments comprising cylinder
[X,Y,Z] = cylinder(R,N);                    %Create x, y, and z components of cylinder
C = zeros(2,N);                             %Generate C to serve as basic colormap (will be lime green)
[r,s,t] = sphere(N);                        %Create a sphere with NxN segments
C2 = zeros(N,N);                            %Create variable C2 to serve as a colormap (lime green again)

%%%Create Upper Jaguar Cylinder%%%
if ~jaguar,                                 %If the upper Jaguar cylinder is NOT to be hidden:
   %%%Create Strut Between Jaguar Cylinders%%%
   %Create matrices of x and y to create plane for strut:
   [X_strut, Y_strut] = meshgrid(strut_loc : .01 : strut_loc + strut_width);
   Z_strut = X_strut + Y_strut;                     %Create z array (this will ultimately be the strut height
   Zcol_loc = (size(X_strut,2)+1)/2;                %Want the middle column of the Z matrix, this is an index
   Zcol = Z_strut(:,Zcol_loc);                      %Find the column corresonding to the index Zcol_loc
   Zmax = max(abs(Zcol));                           %Find the maximum value in the column vector
   Zcol = Zcol + Zmax;                              %Add to previous Zcol to translate the matrix to zero
   Y_strut = 0.*Y_strut;                            %Reset y values to zero (will assume strut has no thickness)
   count = 1;      Znew = [];                       %Initilize counter and Znew matrix
   while count <= size(Z_strut,2),                  %Create a new matrix consisting entirely of Zcol
```

```
      Znew = [Znew, Zcol];                    %This will produce a rectangular figure
      count = count + 1;                      %Iterate counter
   end
   Z_strut_max = max(abs(Znew));                    %Find the maximum value in the Znew matrix
   Z_strut_max = Z_strut_max(1,1);                  %Want only ONE maximum value
   Z_normalized = (Znew./Z_strut_max);              %Normalize the Z values
   Z_strut = Z_normalized * Jag_offset;             %Multiply the normalized value by the strut height
   C3 = zeros(size(Z_strut,2),size(Z_strut,2));    %Create variable C3 to serve as a colormap (lime green again)
end

[X_sc, Y_sc, Z_sc, C_sc, X2_sc, Y2_sc, Z2_sc, X3_sc, Y3_sc, Z3_sc,...
   X4_sc, Y4_sc, Z4_sc, X_lower, X_upper, Y_lower, Y_upper, Z_lower, Z_upper] = ...
   sample_container(containers, x_sc, y_sc, z_sc, pitch_sc, roll_sc, yaw_sc, sc_plot);

%%%PLOTS%%%
if plots,
if colors,
   figure(2)         %Plot of Manipulator in X-Z Plane
   plot3(x0_pt, y0_pt, z0_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 0 origin
   plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on         %Plot Joint 1 (Frame 0 to 1)
   plot3(x1_pt, y1_pt, z1_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 1 origin
   plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on         %Plot Joint 2 (Frame 1 to 2)
   plot3(x2_pt, y2_pt, z2_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 2 origin
   plot3(x_23, y_23, z_23, 'r-', 'LineWidth', line_width), hold on         %Plot Link 1 (Frame 2 to Frame 3)
   plot3(x3_pt, y3_pt, z3_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 3 origin
   plot3(x4a, y4a, z4a, '-', 'LineWidth', line_width), hold on             %Plot small portion of Link 2 (Frame 3 to 4a)
   plot3(x_4a4, y_4a4, z_4a4, '-', 'LineWidth', line_width), hold on       %Plot main portion of Link 2 (Frame 4a to
Frame 4)
   plot3(x4_pt, y4_pt, z4_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point on top of Link 2 axis
   plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on         %Plot connection between Joints 5 and 6
   plot3(x_56_pt, y_56_pt, z_56_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of Joint 6
   plot3(x_67, y_67, z_67, 'g-', 'LineWidth', line_width), hold on         %Plot connection between Joints 6 and EE
   plot3(x_67_pt, y_67_pt, z_67_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of EE (tool tip)
   surf(height*Z-height,Y,X-radius,C), hold on                             %Produce cylindrical surface
   surf(radius*r,radius*s,radius*t-radius,C2), hold on                     %Generate sphere
   if ~jaguar,                                                             %Create Jaguar Top
      surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on             %Produce cylindrical surface
                                                %Third variable represents the cylinder axis
      surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);             %Generate sphere
      surf(X_strut,Y_strut,Z_strut,C3)                                     %Plot the strut surface
   end
   surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                   %Produce cylindrical sample container
   fill3(X_sc(1,:), Y_sc(1,:), Z_sc(1,:),'r')                              %Plot sample container baseplate
   surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                %Repeat for S.C. #2
   fill3(X2_sc(1,:), Y2_sc(1,:), Z2_sc(1,:),'r')
   if ~sc_plot,                                                            %If plotting containers 3 & 4
      surf(X3_sc, Y3_sc, Z3_sc, C_sc), hold on                             %Repeat for S.C. #3
      fill3(X3_sc(1,:), Y3_sc(1,:), Z3_sc(1,:),'r')
      surf(X4_sc, Y4_sc, Z4_sc, C_sc), hold on                             %Repeat for S.C. #4
      fill3(X4_sc(1,:), Y4_sc(1,:), Z4_sc(1,:),'r')
   end
   title('Manipulator Plotted in 3 Dimensions', 'FontWeight', 'bold', 'FontSize', title_size)
   xlabel('X-axis (m)', 'FontSize', title_size), ylabel('Y-axis (m)', 'FontSize', title_size), zlabel('Z-axis (m)', 'FontSize',
title_size)
   axis([xmin xmax ymin ymax zmin zmax])
   grid on


   figure(3)         %Plot of Manipulator in X-Z Plane
   plot3(x0_pt, y0_pt, z0_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 0 origin
   plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on         %Plot Joint 1 (Frame 0 to 1)
   plot3(x1_pt, y1_pt, z1_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 1 origin
   plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on         %Plot Joint 2 (Frame 1 to 2)
   plot3(x2_pt, y2_pt, z2_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 2 origin
   plot3(x_23, y_23, z_23, 'r-', 'LineWidth', line_width), hold on         %Plot Link 1 (Frame 2 to Frame 3)
   plot3(x3_pt, y3_pt, z3_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 3 origin
   plot3(x4a, y4a, z4a, '-', 'LineWidth', line_width), hold on             %Plot small portion of Link 2 (Frame 3 to 4a)
   plot3(x_4a4, y_4a4, z_4a4, '-', 'LineWidth', line_width), hold on       %Plot main portion of Link 2 (Frame 4a to
Frame 4)
   plot3(x4_pt, y4_pt, z4_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point on top of Link 2 axis
```

```
    plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on          %Plot connection between Joints 5 and 6
    plot3(x_56_pt, y_56_pt, z_56_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of Joint 6
    plot3(x_67, y_67, z_67, 'g-', 'LineWidth', line_width), hold on          %Plot connection between Joints 6 and EE
    plot3(x_67_pt, y_67_pt, z_67_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of EE (tool tip)
    surf(height*Z-height,Y,X-radius,C), hold on                              %Produce cylindrical surface
    surf(radius*r,radius*s,radius*t-radius,C2), hold on                      %Generate sphere
    if ~jaguar,                                                              %Create Jaguar Top
       surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on              %Produce cylindrical surface
                                                                             %Third variable represents the cylinder axis
       surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);              %Generate sphere
       surf(X_strut,Y_strut,Z_strut,C3)                                      %Plot the strut surface
    end
    surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                    %Produce cylindrical sample container
    fill3(X_sc(1,:), Y_sc(1,:), Z_sc(1,:),'r')                               %Plot sample container baseplate
    surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                 %Repeat for S.C. #2
    fill3(X2_sc(1,:), Y2_sc(1,:), Z2_sc(1,:),'r')
    if ~sc_plot,                                                             %If plotting containers 3 & 4
       surf(X3_sc, Y3_sc, Z3_sc, C_sc), hold on                              %Repeat for S.C. #3
       fill3(X3_sc(1,:), Y3_sc(1,:), Z3_sc(1,:),'r')
       surf(X4_sc, Y4_sc, Z4_sc, C_sc), hold on                              %Repeat for S.C. #4
       fill3(X4_sc(1,:), Y4_sc(1,:), Z4_sc(1,:),'r')
    end
    title('Manipulator Plotted in X-Z Plane', 'FontWeight', 'bold', 'FontSize', title_size)
    xlabel('X-axis (m)', 'FontSize', title_size), ylabel('Y-axis (m)', 'FontSize', title_size), zlabel('Z-axis (m)', 'FontSize',
title_size)
    axis([xmin xmax ymin ymax zmin zmax])
    grid on
    view(0,0)                              %Viewpoint specification (AZ,EL), (0,0) is the x-z plane


    figure(4)          %Plot of Manipulator in X-Y Plane
    plot3(x0_pt, y0_pt, z0_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point at Frame 0 origin
    plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on          %Plot Joint 1 (Frame 0 to 1)
    plot3(x1_pt, y1_pt, z1_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point at Frame 1 origin
    plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on          %Plot Joint 2 (Frame 1 to 2)
    plot3(x2_pt, y2_pt, z2_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point at Frame 2 origin
    plot3(x_23, y_23, z_23, 'r-', 'LineWidth', line_width), hold on          %Plot Link 1 (Frame 2 to Frame 3)
    plot3(x3_pt, y3_pt, z3_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point at Frame 3 origin
    plot3(x4a, y4a, z4a, '-', 'LineWidth', line_width), hold on              %Plot small portion of Link 2 (Frame 3 to 4a)
    plot3(x_4a4, y_4a4, z_4a4, '-', 'LineWidth', line_width), hold on        %Plot main portion of Link 2 (Frame 4a to
Frame 4)
    plot3(x4_pt, y4_pt, z4_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point on top of Link 2 axis
    plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on          %Plot connection between Joints 5 and 6
    plot3(x_56_pt, y_56_pt, z_56_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of Joint 6
    plot3(x_67, y_67, z_67, 'g-', 'LineWidth', line_width), hold on          %Plot connection between Joints 6 and EE
    plot3(x_67_pt, y_67_pt, z_67_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of EE (tool tip)
    surf(height*Z-height,Y,X-radius,C), hold on                              %Produce cylindrical surface
    surf(radius*r,radius*s,radius*t-radius,C2), hold on                      %Generate sphere
    if ~jaguar,                                                              %Create Jaguar Top
       surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on              %Produce cylindrical surface
                                                %Third variable represents the cylinder axis
       surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);              %Generate sphere
       surf(X_strut,Y_strut,Z_strut,C3)                                      %Plot the strut surface
    end
    surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                    %Produce cylindrical sample container
    fill3(X_sc(1,:), Y_sc(1,:), Z_sc(1,:),'r')                               %Plot sample container baseplate
    surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                 %Repeat for S.C. #2
    fill3(X2_sc(1,:), Y2_sc(1,:), Z2_sc(1,:),'r')
    if ~sc_plot,                                                             %If plotting containers 3 & 4
       surf(X3_sc, Y3_sc, Z3_sc, C_sc), hold on                              %Repeat for S.C. #3
       fill3(X3_sc(1,:), Y3_sc(1,:), Z3_sc(1,:),'r')
       surf(X4_sc, Y4_sc, Z4_sc, C_sc), hold on                              %Repeat for S.C. #4
       fill3(X4_sc(1,:), Y4_sc(1,:), Z4_sc(1,:),'r')
    end
    title('Manipulator Plotted in X-Y Plane', 'FontWeight', 'bold', 'FontSize', title_size)
    xlabel('X-axis (m)', 'FontSize', title_size), ylabel('Y-axis (m)', 'FontSize', title_size), zlabel('Z-axis (m)', 'FontSize',
title_size)
    axis([xmin xmax ymin ymax zmin zmax])
    grid on
    view(0,90)                                 %Viewpoint specification (AZ,EL), (0,90) is the x-y plane
```

```
   figure(5)          %Plot of Manipulator in Y-Z Plane
   plot3(x0_pt, y0_pt, z0_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point at Frame 0 origin
   plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on           %Plot Joint 1 (Frame 0 to 1)
   plot3(x1_pt, y1_pt, z1_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point at Frame 1 origin
   plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on           %Plot Joint 2 (Frame 1 to 2)
   plot3(x2_pt, y2_pt, z2_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point at Frame 2 origin
   plot3(x_23, y_23, z_23, 'r-', 'LineWidth', line_width), hold on           %Plot Link 1 (Frame 2 to Frame 3)
   plot3(x3_pt, y3_pt, z3_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point at Frame 3 origin
   plot3(x4a, y4a, z4a, '-', 'LineWidth', line_width), hold on               %Plot small portion of Link 2 (Frame 3 to 4a)
   plot3(x_4a4, y_4a4, z_4a4, '-', 'LineWidth', line_width), hold on         %Plot main portion of Link 2 (Frame 4a to
Frame 4)
   plot3(x4_pt, y4_pt, z4_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point on top of Link 2 axis
   plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on           %Plot connection between Joints 5 and 6
   plot3(x_56_pt, y_56_pt, z_56_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of Joint 6
   plot3(x_67, y_67, z_67, 'g-', 'LineWidth', line_width), hold on           %Plot connection between Joints 6 and EE
   plot3(x_67_pt, y_67_pt, z_67_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of EE (tool tip)
   surf(height*Z-height,Y,X-radius,C), hold on                               %Produce cylindrical surface
   surf(radius*r,radius*s,radius*t-radius,C2), hold on                       %Generate sphere
   if ~jaguar,                                                               %Create Jaguar Top
      surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on               %Produce cylindrical surface
                                                  %Third variable represents the cylinder axis
      surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);               %Generate sphere
      surf(X_strut,Y_strut,Z_strut,C3)                                       %Plot the strut surface
   end
   surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                     %Produce cylindrical sample container
   fill3(X_sc(1,:), Y_sc(1,:), Z_sc(1,:),'r')                                %Plot sample container baseplate
   surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                  %Repeat for S.C. #2
   fill3(X2_sc(1,:), Y2_sc(1,:), Z2_sc(1,:),'r')
   if ~sc_plot,                                                              %If plotting containers 3 & 4
      surf(X3_sc, Y3_sc, Z3_sc, C_sc), hold on                               %Repeat for S.C. #3
      fill3(X3_sc(1,:), Y3_sc(1,:), Z3_sc(1,:),'r')
      surf(X4_sc, Y4_sc, Z4_sc, C_sc), hold on                               %Repeat for S.C. #4
      fill3(X4_sc(1,:), Y4_sc(1,:), Z4_sc(1,:),'r')
   end
   title('Manipulator Plotted in Y-Z Plane', 'FontWeight', 'bold', 'FontSize', title_size)
   xlabel('X-axis (m)', 'FontSize', title_size), ylabel('Y-axis (m)', 'FontSize', title_size), zlabel('Z-axis (m)', 'FontSize',
title_size)
   axis([xmin xmax ymin ymax zmin zmax])
   grid on
   view(90,0)        %Viewpoint specification (AZ,EL), (90,0) is the y-z plane
else
   figure(2)          %Plot of Manipulator in X-Z Plane
   plot3(x0_pt, y0_pt, z0_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point at Frame 0 origin
   plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on           %Plot Joint 1 (Frame 0 to 1)
   plot3(x1_pt, y1_pt, z1_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point at Frame 1 origin
   plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on           %Plot Joint 2 (Frame 1 to 2)
   plot3(x2_pt, y2_pt, z2_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point at Frame 2 origin
   plot3(x_23, y_23, z_23, 'k-', 'LineWidth', line_width), hold on           %Plot Link 1 (Frame 2 to Frame 3)
   plot3(x3_pt, y3_pt, z3_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point at Frame 3 origin
   plot3(x4a, y4a, z4a, 'k-', 'LineWidth', line_width), hold on              %Plot small portion of Link 2 (Frame 3 to 4a)
   plot3(x_4a4, y_4a4, z_4a4, 'k-', 'LineWidth', line_width), hold on        %Plot main portion of Link 2 (Frame 4a to
Frame 4)
   plot3(x4_pt, y4_pt, z4_pt, 'm+', 'LineWidth', line_width), hold on        %Plot point on top of Link 2 axis
   plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on           %Plot connection between Joints 5 and 6
   plot3(x_56_pt, y_56_pt, z_56_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of Joint 6
   plot3(x_67, y_67, z_67, 'k-', 'LineWidth', line_width), hold on           %Plot connection between Joints 6 and EE
   plot3(x_67_pt, y_67_pt, z_67_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of EE (tool tip)
   surf(height*Z-height,Y,X-radius,C), hold on                               %Produce cylindrical surface
   surf(radius*r,radius*s,radius*t-radius,C2), hold on                       %Generate sphere
   if ~jaguar,                                                               %Create Jaguar Top
      surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on               %Produce cylindrical surface
                                                                             %Third variable represents the cylinder axis
      surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);               %Generate sphere
      surf(X_strut,Y_strut,Z_strut,C3)                                       %Plot the strut surface
   end
   surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                     %Produce cylindrical sample container
   fill3(X_sc(1,:), Y_sc(1,:), Z_sc(1,:),'r')                                %Plot sample container baseplate
   surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                  %Repeat for S.C. #2
```

```
   fill3(X2_sc(1,:), Y2_sc(1,:), Z2_sc(1,:),'r')
   if ~sc_plot,                                          %If plotting containers 3 & 4
      surf(X3_sc, Y3_sc, Z3_sc, C_sc), hold on           %Repeat for S.C. #3
      fill3(X3_sc(1,:), Y3_sc(1,:), Z3_sc(1,:),'r')
      surf(X4_sc, Y4_sc, Z4_sc, C_sc), hold on           %Repeat for S.C. #4
      fill3(X4_sc(1,:), Y4_sc(1,:), Z4_sc(1,:),'r')
   end
   title('Manipulator Plotted in 3 Dimensions', 'FontWeight', 'bold', 'FontSize', title_size)
   xlabel('X-axis (m)', 'FontSize', title_size), ylabel('Y-axis (m)', 'FontSize', title_size), zlabel('Z-axis (m)', 'FontSize',
title_size)
   axis([xmin xmax ymin ymax zmin zmax])
   grid on


   figure(3)         %Plot of Manipulator in X-Z Plane
   plot3(x0_pt, y0_pt, z0_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 0 origin
   plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on         %Plot Joint 1 (Frame 0 to 1)
   plot3(x1_pt, y1_pt, z1_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 1 origin
   plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on         %Plot Joint 2 (Frame 1 to 2)
   plot3(x2_pt, y2_pt, z2_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 2 origin
   plot3(x_23, y_23, z_23, 'k-', 'LineWidth', line_width), hold on         %Plot Link 1 (Frame 2 to Frame 3)
   plot3(x3_pt, y3_pt, z3_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 3 origin
   plot3(x4a, y4a, z4a, 'k-', 'LineWidth', line_width), hold on            %Plot small portion of Link 2 (Frame 3 to 4a)
   plot3(x_4a4, y_4a4, z_4a4, 'k-', 'LineWidth', line_width), hold on      %Plot main portion of Link 2 (Frame 4a to
Frame 4)
   plot3(x4_pt, y4_pt, z4_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point on top of Link 2 axis
   plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on         %Plot connection between Joints 5 and 6
   plot3(x_56_pt, y_56_pt, z_56_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of Joint 6
   plot3(x_67, y_67, z_67, 'k-', 'LineWidth', line_width), hold on         %Plot connection between Joints 6 and EE
   plot3(x_67_pt, y_67_pt, z_67_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of EE (tool tip)
   surf(height*Z-height,Y,X-radius,C), hold on                             %Produce cylindrical surface
   surf(radius*r,radius*s,radius*t-radius,C2), hold on                     %Generate sphere
   if ~jaguar,                                                             %Create Jaguar Top
      surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on             %Produce cylindrical surface
                                                %Third variable represents the cylinder axis
      surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);             %Generate sphere
      surf(X_strut,Y_strut,Z_strut,C3)                                     %Plot the strut surface
   end
   surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                   %Produce cylindrical sample container
   fill3(X_sc(1,:), Y_sc(1,:), Z_sc(1,:),'r')                              %Plot sample container baseplate
   surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                %Repeat for S.C. #2
   fill3(X2_sc(1,:), Y2_sc(1,:), Z2_sc(1,:),'r')
   if ~sc_plot,                                                            %If plotting containers 3 & 4
      surf(X3_sc, Y3_sc, Z3_sc, C_sc), hold on                             %Repeat for S.C. #3
      fill3(X3_sc(1,:), Y3_sc(1,:), Z3_sc(1,:),'r')
      surf(X4_sc, Y4_sc, Z4_sc, C_sc), hold on                             %Repeat for S.C. #4
      fill3(X4_sc(1,:), Y4_sc(1,:), Z4_sc(1,:),'r')
   end
   title('Manipulator Plotted in X-Z Plane', 'FontWeight', 'bold', 'FontSize', title_size)
   xlabel('X-axis (m)', 'FontSize', title_size), ylabel('Y-axis (m)', 'FontSize', title_size), zlabel('Z-axis (m)', 'FontSize',
title_size)
   axis([xmin xmax ymin ymax zmin zmax])
   grid on
   view(0,0)                                  %Viewpoint specification (AZ,EL), (0,0) is the x-z plane


   figure(4)         %Plot of Manipulator in X-Y Plane
   plot3(x0_pt, y0_pt, z0_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 0 origin
   plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on         %Plot Joint 1 (Frame 0 to 1)
   plot3(x1_pt, y1_pt, z1_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 1 origin
   plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on         %Plot Joint 2 (Frame 1 to 2)
   plot3(x2_pt, y2_pt, z2_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 2 origin
   plot3(x_23, y_23, z_23, 'k-', 'LineWidth', line_width), hold on         %Plot Link 1 (Frame 2 to Frame 3)
   plot3(x3_pt, y3_pt, z3_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point at Frame 3 origin
   plot3(x4a, y4a, z4a, 'k-', 'LineWidth', line_width), hold on            %Plot small portion of Link 2 (Frame 3 to 4a)
   plot3(x_4a4, y_4a4, z_4a4, 'k-', 'LineWidth', line_width), hold on      %Plot main portion of Link 2 (Frame 4a to
Frame 4)
   plot3(x4_pt, y4_pt, z4_pt, 'm+', 'LineWidth', line_width), hold on      %Plot point on top of Link 2 axis
   plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on         %Plot connection between Joints 5 and 6
   plot3(x_56_pt, y_56_pt, z_56_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of Joint 6
```

```
   plot3(x_67, y_67, z_67, 'k-', 'LineWidth', line_width), hold on          %Plot connection between Joints 6 and EE
   plot3(x_67_pt, y_67_pt, z_67_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of EE (tool tip)
   surf(height*Z-height,Y,X-radius,C), hold on                              %Produce cylindrical surface
   surf(radius*r,radius*s,radius*t-radius,C2), hold on                      %Generate sphere
   if ~jaguar,                                                              %Create Jaguar Top
      surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on              %Produce cylindrical surface
                                                %Third variable represents the cylinder axis
      surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);              %Generate sphere
      surf(X_strut,Y_strut,Z_strut,C3)                                      %Plot the strut surface
   end
   surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                    %Produce cylindrical sample container
   fill3(X_sc(1,:), Y_sc(1,:), Z_sc(1,:),'r')                               %Plot sample container baseplate
   surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                 %Repeat for S.C. #2
   fill3(X2_sc(1,:), Y2_sc(1,:), Z2_sc(1,:),'r')
   if ~sc_plot,                                                             %If plotting containers 3 & 4
      surf(X3_sc, Y3_sc, Z3_sc, C_sc), hold on                              %Repeat for S.C. #3
      fill3(X3_sc(1,:), Y3_sc(1,:), Z3_sc(1,:),'r')
      surf(X4_sc, Y4_sc, Z4_sc, C_sc), hold on                              %Repeat for S.C. #4
      fill3(X4_sc(1,:), Y4_sc(1,:), Z4_sc(1,:),'r')
   end
   title('Manipulator Plotted in X-Y Plane', 'FontWeight', 'bold', 'FontSize', title_size)
   xlabel('X-axis (m)', 'FontSize', title_size), ylabel('Y-axis (m)', 'FontSize', title_size), zlabel('Z-axis (m)', 'FontSize',
title_size)
   axis([xmin xmax ymin ymax zmin zmax])
   grid on
   view(0,90)                                          %Viewpoint specification (AZ,EL), (0,90) is the x-y plane


   figure(5)          %Plot of Manipulator in Y-Z Plane
   plot3(x0_pt, y0_pt, z0_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point at Frame 0 origin
   plot3(x_01, y_01, z_01, 'k-', 'LineWidth', line_width), hold on          %Plot Joint 1 (Frame 0 to 1)
   plot3(x1_pt, y1_pt, z1_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point at Frame 1 origin
   plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on          %Plot Joint 2 (Frame 1 to 2)
   plot3(x2_pt, y2_pt, z2_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point at Frame 2 origin
   plot3(x_23, y_23, z_23, 'k-', 'LineWidth', line_width), hold on          %Plot Link 1 (Frame 2 to Frame 3)
   plot3(x3_pt, y3_pt, z3_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point at Frame 3 origin
   plot3(x4a, y4a, z4a, 'k-', 'LineWidth', line_width), hold on             %Plot small portion of Link 2 (Frame 3 to 4a)
   plot3(x_4a4, y_4a4, z_4a4, 'k-', 'LineWidth', line_width), hold on       %Plot main portion of Link 2 (Frame 4a to
Frame 4)
   plot3(x4_pt, y4_pt, z4_pt, 'm+', 'LineWidth', line_width), hold on       %Plot point on top of Link 2 axis
   plot3(x_56, y_56, z_56, 'k-', 'LineWidth', line_width), hold on          %Plot connection between Joints 5 and 6
   plot3(x_56_pt, y_56_pt, z_56_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of Joint 6
   plot3(x_67, y_67, z_67, 'k-', 'LineWidth', line_width), hold on          %Plot connection between Joints 6 and EE
   plot3(x_67_pt, y_67_pt, z_67_pt, 'm+', 'LineWidth', line_width), hold on %Plot end of EE (tool tip)
   surf(height*Z-height,Y,X-radius,C), hold on                              %Produce cylindrical surface
   surf(radius*r,radius*s,radius*t-radius,C2), hold on                      %Generate sphere
   if ~jaguar,                                          %Create Jaguar Top
      surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on              %Produce cylindrical surface
                                                %Third variable represents the cylinder axis
      surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);              %Generate sphere
      surf(X_strut,Y_strut,Z_strut,C3)                                      %Plot the strut surface
   end
   surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                    %Produce cylindrical sample container
   fill3(X_sc(1,:), Y_sc(1,:), Z_sc(1,:),'r')                               %Plot sample container baseplate
   surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                 %Repeat for S.C. #2
   fill3(X2_sc(1,:), Y2_sc(1,:), Z2_sc(1,:),'r')
   if ~sc_plot,                                         %If plotting containers 3 & 4
      surf(X3_sc, Y3_sc, Z3_sc, C_sc), hold on                              %Repeat for S.C. #3
      fill3(X3_sc(1,:), Y3_sc(1,:), Z3_sc(1,:),'r')
      surf(X4_sc, Y4_sc, Z4_sc, C_sc), hold on                              %Repeat for S.C. #4
      fill3(X4_sc(1,:), Y4_sc(1,:), Z4_sc(1,:),'r')
   end
   title('Manipulator Plotted in Y-Z Plane', 'FontWeight', 'bold', 'FontSize', title_size)
   xlabel('X-axis (m)', 'FontSize', title_size), ylabel('Y-axis (m)', 'FontSize', title_size), zlabel('Z-axis (m)', 'FontSize',
title_size)
   axis([xmin xmax ymin ymax zmin zmax])
   grid on
   view(90,0)                                               %Viewpoint specification (AZ,EL), (90,0) is the y-z
plane
end
```

```
if spread,       %If plots are to be spread across the window:
   if imac,     %If the figures are to be displayed on 2 Mac monitors:
      x_zoom = .15;
      y_zoom = .375;
      set(figure(5), 'units', 'normalized', 'outerposition', [.15 .25  x_zoom y_zoom])
      set(figure(4), 'units', 'normalized', 'outerposition', [0   .25  x_zoom y_zoom])
      set(figure(3), 'units', 'normalized', 'outerposition', [.15  1   x_zoom y_zoom])
      set(figure(2), 'units', 'normalized', 'outerposition', [0    1   x_zoom y_zoom])
   end
   if laptop,  %If the plots are to be shown on a standard Windows laptop
      y_zoom = .48075;
      x_zoom = 1/3;
      y_base = .039;
      set(figure(5), 'units', 'normalized', 'outerposition', [.6666   y_base  x_zoom y_zoom])
      set(figure(4), 'units', 'normalized', 'outerposition', [.3333   y_base  x_zoom y_zoom])
      set(figure(3), 'units', 'normalized', 'outerposition', [0       y_base  x_zoom y_zoom])
      set(figure(2), 'units', 'normalized', 'outerposition', [.6666   .52     x_zoom y_zoom])
   end
end
end
```

H.9 Function work.m

```
function work(spread, jaguar, laptop, imac, workspace_iter, nframes)

tic                                                                    %Initiate timer

%theta_1_deg_min = -110;   theta_1_deg_max = 110;                      %Max and Min Joint 1 Angles (degrees)
%Theta 1 min is set to zero as the workspace is symmetric
theta_1_deg_min = 0;      theta_1_deg_max = 110;                       %Max and Min Joint 1 Angles (degrees)
theta_2_deg_min = -110;   theta_2_deg_max = 110;                       %Max and Min Joint 2 Angles (degrees)
%theta_3_deg_min = -30;    theta_3_deg_max = 170;                      %Max and Min Joint 3 Angles (degrees)
%theta_4_deg_min = -180;   theta_4_deg_max = 180;                      %Max and Min Joint 4 Angles (degrees)
%theta_5_deg_min = -105;   theta_5_deg_max = 105;                      %Max and Min Joint 5 Angles (degrees)
theta_5_deg = 0;                                                       %Fix at zero to reach work envelope (deg)
theta_4_deg = 0;                                                       %Fix at zero to reach work envelope (deg)
theta_3_deg = -90;                                                     %Fix at -90 to reach work envelope (deg)
%Joint 6 will drive the EE and will not contribute to the workspace

%Table of D-H Parameters
   alpha0 = 0;      a0 = 0;       d1 = .10795;
   alpha1 = -pi/2;  a1 = .1524;    d2 = 0;
   alpha2 = 0;      a2 = .6096;    d3 = 0;
   alpha3 = -pi/2;  a3 = .1143;    d4 = .6096;
   alpha4 = pi/2;   a4 = 0;       d5 = 0;
   alpha5 = -pi/2;  a5 = 0;       d6 = 0;

   ee_length = .2237;        %End-effector length (m)

res = .01;                        %Resolution off increments composing lines
                                  %This must be sufficiently small to coincide
                                  %with values in kinematics codes
line_width = 2;                   %Set width of plotted lines
lw_mult = 2;                      %Line width multiplier for coord. sys origin point
title_size = 14;                  %Set the size of the font for plot text (26 for maximized windows)
xmin = -1.5; xmax = 1.5; ymin = -1.5; ymax = 1.5; zmin = -1.5; zmax = 1.5;   %Values for axes definitions
xmin = -2; xmax = 2; ymin = -2; ymax = 2; zmin = -2; zmax = 2;   %Values for axes definitions
fig_color = [1 1 1];              %Sets background in figures to white
set(0, 'DefaultFigureColor', fig_color)

%Jaguar Details
radius = 0.1984;                  %Specify radius of Jaguar cylinder (m)
height = abs(xmin);               %"height" or length of cylinder (m)
strut_loc = -.42;                 %x-displacement of Jaguar strut from base frame (0,0,0) in meters
strut_width = .12;                %Strut width (m)
Jag_offset = 1.194;               %Distance between Jaguar cylinder axes (m)

%Sample Container Details
radius_sc = .1365;                %Sample container outer radius (m) - (corresponds to Di = 10.75 in)
height_sc = .2667;                %Sample continer height (m) - (corresonds to h = 10.5 in)
x_loc_sc = .12;                   %Dist. from origin in x-direction (m)
y_loc_sc = .3350;                 %Dist. from origin in y-direction (m)
z_loc_sc = .3318;                 %Dist. from origin in z-direction (m)

theta_1_deg = theta_1_deg_min;          %Set initial Joint 1 angle to min value (deg)
theta_2_deg = theta_2_deg_min;          %Set initial Joint 2 angle to min value (deg)
%theta_3_deg = theta_3_deg_min;         %Set initial Joint 3 angle to min value (deg)
%theta_4_deg = theta_4_deg_min;         %Set initial Joint 4 angle to min value (deg)
%heta_5_deg = theta_5_deg_min;          %Set initial Joint 5 angle to min value (deg)
theta_6_deg = 0;                        %Joint 6 Angle is arbitrary and set to 0(deg)

x_67_array = [];        %Initialize array for x-coords. for tool tip
y_67_array = [];        %Initialize array for y-coords. for tool tip
z_67_array = [];        %Initialize array for z-coords. for tool tip

%Convert D-H Parmeters in Table into MATLAB Arrays
alpha = [alpha0, alpha1, alpha2, alpha3, alpha4, alpha5];       %Create array of alpha angles (rad)
a = [a0, a1, a2, a3, a4, a5];                   %Array of "a" offset vectors
```

```
d = [d1, d2, d3, d4, d5, d6];                    %Array of "d" offset vectors

counter = 0;                                     %Initialize iteration counter variable
while theta_1_deg <= theta_1_deg_max,
   theta_2_deg = theta_2_deg_min;                %Reset theta_2_deg to min value (deg)
    while theta_2_deg <= theta_2_deg_max,
       counter = counter + 1;                    %Iterate iteration counter
       %Create theta_deg array:
       theta_deg = [theta_1_deg, theta_2_deg, theta_3_deg, theta_4_deg, theta_5_deg, theta_6_deg];
       theta = theta_deg * pi/180;               %Convert theta array to radians (rad)

for i = 1:nframes,
   %TransformMat computes transformation, rotation, and position matrices
   [T(:,:,i)] = TransformWork(a(i),alpha(i),d(i),theta(i));   %Transformations from frame i to i-1
   %Redefine variable names for matrices and output matrices to data file:
   if i == 1, T1 = T(:,:,i); end
   if i == 2, T2 = T(:,:,i); end
   if i == 3, T3 = T(:,:,i); end
   if i == 4, T4 = T(:,:,i); end
   if i == 5, T5 = T(:,:,i); end
   if i == 6, T6 = T(:,:,i); end
end

%Create a point at the origin of Frame 0:
x0_pt = 0;      y0_pt = 0;      z0_pt = 0;
%Create a point at the origin of Frame 1:
x1_pt = T1(1,4);    y1_pt = T1(2,4);    z1_pt = T1(3,4);
%Create line between Frames 0 and 1
if abs(x1_pt-x0_pt) >= abs(y1_pt-y0_pt) && abs(x1_pt-x0_pt) >= abs(z1_pt-z0_pt),
   x_01 = x0_pt : res : x1_pt;     %Let x-array drive change from Frame 0 to 1
   y_01 = y0_pt : abs(y1_pt-y0_pt)/(size(x_01,2)-1) : y1_pt;  %Create y-array based on size of x-array
   z_01 = z0_pt : abs(z1_pt-z0_pt)/(size(x_01,2)-1) : z1_pt;  %Create z-array based on size of x-array
   if size(y_01, 2)==1,  y_01 = y0_pt + x_01.*0;  end    %If there is no change in y, create array of same mag.
   if size(z_01, 2)==1,  z_01 = z0_pt + x_01.*0;  end    %If there is no change in z, create array of same mag.
else if abs(y1_pt-y0_pt) >= abs(x1_pt-x0_pt) && abs(y1_pt-y0_pt) >= abs(z1_pt-z0_pt),
   y_01 = y0_pt : res : y1_pt;     %Let y-array drive change from Frame 0 to 1
   x_01 = x0_pt : abs(x1_pt-x0_pt)/(size(y_01,2)-1) : x1_pt;  %Create x-array based on size of y-array
   z_01 = z0_pt : abs(z1_pt-z0_pt)/(size(y_01,2)-1) : z1_pt;  %Create z-array based on size of y-array
   if size(x_01, 2)==1,  x_01 = x0_pt + y_01.*0;  end    %If there is no change in x, create array of same mag.
   if size(z_01, 2)==1,  z_01 = z0_pt + y_01.*0;  end    %If there is no change in z, create array of same mag.
   else       %the biggest change is in the z-direction
      z_01 = z0_pt : res : z1_pt;     %Let z-array drive change from Frame 0 to 1
      x_01 = x0_pt : abs(x1_pt-x0_pt)/(size(z_01,2)-1) : x1_pt;  %Create x-array based on size of z-array
      y_01 = y0_pt : abs(y1_pt-y0_pt)/(size(z_01,2)-1) : y1_pt;  %Create y-array based on size of z-array

      if size(x_01, 2)<=1,  x_01 = x0_pt + z_01.*0;  end    %If there is no change in x, create array of same mag.
      if size(y_01, 2)<=1,  y_01 = y0_pt + z_01.*0;  end    %If there is no change in y, create array of same mag.
   end
end

%Create a point at the origin of Frame 2:
T_current = T1*T2;          %Create new transformation matrix from origin to current frame
x2_pt = T_current(1,4);    y2_pt = T_current(2,4);    z2_pt = T_current(3,4);
%%%plot3(x2_pt, y2_pt, z2_pt, 'r+', 'LineWidth', line_width), hold on       %Plot origin of Frame 2
%Create line between Frames 1 and 2
%Case I: Greatest change between frames is in x-direction:
if abs(x2_pt-x1_pt) >= abs(y2_pt-y1_pt) && abs(x2_pt-x1_pt) >= abs(z2_pt-z1_pt),
   if x2_pt >= x1_pt,                %If x2 is greater than or equal to x1, create normal array
      x_12 = x1_pt : res : x2_pt;     %Let x-array drive change from Frame 1 to 2
   else x_12 = -x1_pt : res : -x2_pt;   x_12 = -x_12;   end    %Otherwise, flip array direction.
   size_x12 = size(x_12,2)-1;         %Size of x_12 array
   if y2_pt >= y1_pt,                %If y2 is greater than or equal to y1, create normal array
      y_12 = y1_pt : abs(y2_pt-y1_pt)/size_x12 : y2_pt;  %Create y-array based on size of x-array
   else
      y_12 = -y1_pt : abs(y2_pt-y1_pt)/size_x12 : -y2_pt;   y_12 = -y_12;   %Otherwise, flip array direction
   end
   if z2_pt >= z1_pt,
      z_12 = z1_pt : abs(z2_pt-z1_pt)/size_x12 : z2_pt;  %Create z-array based on size of x-array
   else
      z_12 = -z1_pt : abs(z2_pt-z1_pt)/size_x12 : -z2_pt;   z_12 = -z_12;   %Otherwise, flip array direction
```

```
   end
   if size(y_12, 2)<=1,  y_12 = y1_pt + x_12.*0;  end    %If there is no change in y, create array of same mag.
   if size(z_12, 2)<=1,  z_12 = z1_pt + x_12.*0;  end    %If there is no change in z, create array of same mag.
%Case II: Greatest change between frames is in y-direction:
else if abs(y2_pt-y1_pt) >= abs(x2_pt-x1_pt) && abs(y2_pt-y1_pt) >= abs(z2_pt-z1_pt),
   if y2_pt >= y1_pt,                %If y2 is greater than or equal to y1,
      y_12 = y1_pt : res : y2_pt;     %Let y-array drive change from Frame 1 to 2
   else y_12 = -y1_pt : res : -y2_pt;   y_12 = -y_12;  end    %Otherwise, flip array direction.
   size_y12 = size(y_12,2)-1;          %Size of y_12 array
   if x2_pt >= x1_pt,                %If x2 is greater than or equal to x1,
      x_12 = x1_pt : abs(x2_pt-x1_pt)/size_y12 : x2_pt;  %Create x-array based on size of y-array
   else
      x_12 = -x1_pt : abs(x2_pt-x1_pt)/size_y12 : -x2_pt; x_12 = -x_12;  %Create x-array based on size of y-array
   end
   if z2_pt >= z1_pt,
      z_12 = z1_pt : abs(z2_pt-z1_pt)/size_y12 : z2_pt;  %Create z-array based on size of y-array
   else
      z_12 = -z1_pt : abs(z2_pt-z1_pt)/size_y12 : -z2_pt;   z_12 = -z_12;  %Create z-array based on size of y-array
   end
   if size(x_12, 2)<=1,  x_12 = x1_pt + y_12.*0;  end    %If there is no change in x, create array of same mag.
   if size(z_12, 2)<=1,  z_12 = z1_pt + y_12.*0;  end    %If there is no change in z, create array of same mag.
%Case III: Greatest change between frames is in z-direction:
   else       %the biggest change is in the z-direction
      if z2_pt >= z1_pt,
         z_12 = z1_pt : res : z2_pt;     %Let z-array drive change from Frame 1 to 2
      else z_12 = -z1_pt : res : -z2_pt;   z_12 = -z_12;  end    %Otherwise, flip array direction.
      size_z12 = size(z_12,2)-1;          %Size of z_12 array
      if x2_pt >= x1_pt,                %If x2 is greater than or equal to x1,
         x_12 = x1_pt : abs(x2_pt-x1_pt)/size_z12 : x2_pt;  %Create x-array based on size of z-array
      else
         x_12 = -x1_pt : abs(x2_pt-x1_pt)/size_z12 : -x2_pt;  x_12 = -x_12;  end    %Otherwise, flip array direction.
      if y2_pt >= y1_pt,                %If x2 is greater than or equal to x1,
         y_12 = y1_pt : abs(y2_pt-y1_pt)/size_z12 : y2_pt;  %Create y-array based on size of z-array
      else
         y_12 = -y1_pt : abs(y2_pt-y1_pt)/size_z12 : -y2_pt;  y_12 = -y_12;  end    %Otherwise, flip array direction.
      if size(x_12, 2)<=1,  x_12 = x1_pt + z_12.*0;  end    %If there is no change in x, create array of same mag.
      if size(y_12, 2)<=1,  y_12 = y1_pt + z_12.*0;  end    %If there is no change in y, create array of same mag.
   end
end
%%%plot3(x_12, y_12, z_12, 'k-', 'LineWidth', line_width), hold on       %Plot line between Frames 1 & 2 in 3D

%Create a point at the origin of Frame 3:
T_current = T_current*T3;          %Create new transformation matrix from origin to current frame
x3_pt = T_current(1,4);    y3_pt = T_current(2,4);    z3_pt = T_current(3,4);
%Create line between Frames 2 and 3
%Case I: Greatest change between frames is in x-direction:
if abs(x3_pt-x2_pt) >= abs(y3_pt-y2_pt) && abs(x3_pt-x2_pt) >= abs(z3_pt-z2_pt),

   if x3_pt >= x2_pt,                %If x3 is greater than or equal to x2, create normal array
      x_23 = x2_pt : res : x3_pt;     %Let x-array drive change from Frame 2 to 3
   else x_23 = -x2_pt : res : -x3_pt;  x_23 = -x_23;  end    %Otherwise, flip array direction.
   size_x23 = size(x_23,2)-1;          %Size of x_23 array
   if y3_pt >= y2_pt,                %If y3 is greater than or equal to y2, create normal array
      y_23 = y2_pt : abs(y3_pt-y2_pt)/size_x23 : y3_pt;  %Create y-array based on size of x-array
   else
      y_23 = -y2_pt : abs(y3_pt-y2_pt)/size_x23 : -y3_pt;   y_23 = -y_23;   %Otherwise, flip array direction.
   end
   if z3_pt >= z2_pt,
      z_23 = z2_pt : abs(z3_pt-z2_pt)/size_x23 : z3_pt;  %Create z-array based on size of x-array
   else
      z_23 = -z2_pt : abs(z3_pt-z2_pt)/size_x23 : -z3_pt;   z_23 = -z_23;   %Otherwise, flip array direction.
   end
   if size(y_23, 2)<=1,  y_23 = y2_pt + x_23.*0;  end    %If there is no change in y, create array of same mag.
   if size(z_23, 2)<=1,  z_23 = z2_pt + x_23.*0;  end    %If there is no change in z, create array of same mag.
%Case II: Greatest change between frames is in y-direction:
else if abs(y3_pt-y2_pt) >= abs(x3_pt-x2_pt) && abs(y3_pt-y2_pt) >= abs(z3_pt-z2_pt),

   if y3_pt >= y2_pt,                %If y3 is greater than or equal to y2,
      y_23 = y2_pt : res : y3_pt;     %Let y-array drive change from Frame 2 to 3
   else y_23 = -y2_pt : res : -y3_pt;   y_23 = -y_23;  end    %Otherwise, flip array direction.
```

```
   size_y23 = size(y_23,2)-1;          %Size of y_23 array
   if x3_pt >= x2_pt,                  %If x2 is greater than or equal to x1,
      x_23 = x2_pt : abs(x3_pt-x2_pt)/size_y23 : x3_pt;  %Create x-array based on size of y-array
   else
      x_23 = -x2_pt : abs(x3_pt-x2_pt)/size_y23 : -x3_pt; x_23 = -x_23;  %Create x-array based on size of y-array
   end
   if z3_pt >= z2_pt,
      z_23 = z2_pt : abs(z3_pt-z2_pt)/size_y23 : z3_pt;  %Create z-array based on size of y-array
   else
      z_23 = -z2_pt : abs(z3_pt-z2_pt)/size_y23 : -z3_pt;   z_23 = -z_23;  %Create z-array based on size of y-array
   end
   if size(x_23, 2)<=1,  x_23 = x2_pt + y_23.*0;  end    %If there is no change in x, create array of same mag.
   if size(z_23, 2)<=1,  z_23 = z2_pt + y_23.*0;  end    %If there is no change in z, create array of same mag.
%Case III: Greatest change between frames is in z-direction:
else        %the biggest change is in the z-direction
      if z3_pt >= z2_pt,
         z_23 = z2_pt : res : z3_pt;     %Let z-array drive change from Frame 2 to 3
      else z_23 = -z2_pt : res : -z3_pt;   z_23 = -z_23;  end    %Otherwise, flip array direction.
      size_z23 = size(z_23,2)-1;          %Size of z_23 array
      if x3_pt >= x2_pt,                  %If x3 is greater than or equal to x2,
         x_23 = x2_pt : abs(x3_pt-x2_pt)/size_z23 : x3_pt;  %Create x-array based on size of z-array
      else
         x_23 = -x2_pt : abs(x3_pt-x2_pt)/size_z23 : -x3_pt;  x_23 = -x_23;  end    %Otherwise, flip array direction.
      if y3_pt >= y2_pt,                  %If y3 is greater than or equal to y2,
         y_23 = y2_pt : abs(y3_pt-y2_pt)/size_z23 : y3_pt;  %Create y-array based on size of z-array
      else
         y_23 = -y2_pt : abs(y3_pt-y2_pt)/size_z23 : -y3_pt;  y_23 = -y_23;  end    %Otherwise, flip array direction.
      if size(x_23, 2)<=1,  x_23 = x2_pt + z_23.*0;  end    %If there is no change in x, create array of same mag.
      if size(y_23, 2)<=1,  y_23 = y2_pt + z_23.*0;  end    %If there is no change in y, create array of same mag.
   end
end

%%%Frame 3 to Frame 4a (Bend in Link 2)%%%
l4 = 0 : res : a3;                     %Create an array based on the length of top part of Link 2
phi_sum = theta(1);                          %Summation of all phi angles to this point is zero
x4a = x3_pt + l4*sin(pi/2+theta(2)+theta(3))*cos(phi_sum);     %pi/2 is the fixed angle between d1 and the z0 axis
y4a = y3_pt + l4*sin(pi/2+theta(2)+theta(3))*sin(phi_sum);
z4a = z3_pt + l4*cos(pi/2+theta(2)+theta(3));
x4a_pt_loc = size(x4a,2);    y4a_pt_loc = size(y4a,2);        z4a_pt_loc = size(z4a,2);
x4a_pt = x4a(x4a_pt_loc);    y4a_pt = y4a(y4a_pt_loc);    z4a_pt = z4a(z4a_pt_loc);

%%%Frame 4a to Frame 4 (Base of Link 2)%%%
%Create a point at the origin of Frame 4:
T_current = T_current*T4;          %Create new transformation matrix from origin to current frame
x4_pt = T_current(1,4);    y4_pt = T_current(2,4);    z4_pt = T_current(3,4);
%Create line between Frames 4a and 4
%Case I: Greatest change between frames is in x-direction:
if abs(x4_pt-x4a_pt) >= abs(y4_pt-y4a_pt) && abs(x4_pt-x4a_pt) >= abs(z4_pt-z4a_pt),
   if x4_pt >= x4a_pt,                 %If x4 is greater than or equal to x4a, create normal array
      x_4a4 = x4a_pt : res : x4_pt;     %Let x-array drive change from Frame 4a to 4
   else x_4a4 = -x4a_pt : res : -x4_pt;  x_4a4 = -x_4a4;  end    %Otherwise, flip array direction.
   size_x4a4 = size(x_4a4,2)-1;          %Size of x_4a4 array
   if y4_pt >= y4a_pt,                 %If y4 is greater than or equal to y4a, create normal array
      y_4a4 = y4a_pt : abs(y4_pt-y4a_pt)/size_x4a4 : y4_pt;  %Create y-array based on size of x-array
   else
      y_4a4 = -y4a_pt : abs(y4_pt-y4a_pt)/size_x4a4 : -y4_pt;   y_4a4 = -y_4a4;   %Otherwise, flip array direction.
   end
   if z4_pt >= z4a_pt,
      z_4a4 = z4a_pt : abs(z4_pt-z4a_pt)/size_x4a4 : z4_pt;  %Create z-array based on size of x-array
   else
      z_4a4 = -z4a_pt : abs(z4_pt-z4a_pt)/size_x4a4 : -z4_pt;   z_4a4 = -z_4a4;   %Otherwise, flip array direction.
   end
   if size(y_4a4, 2)<=1,  y_4a4 = y4_pt + x_4a4.*0;  end    %If there is no change in y, create array of same mag.
   if size(z_4a4, 2)<=1,  z_4a4 = z4_pt + x_4a4.*0;  end    %If there is no change in z, create array of same mag.
%Case II: Greatest change between frames is in y-direction:
else if abs(y4_pt-y4a_pt) >= abs(x4_pt-x4a_pt) && abs(y4_pt-y4a_pt) >= abs(z4_pt-z4a_pt),
   if y4_pt >= y4a_pt,                 %If y4 is greater than or equal to y4a,
      y_4a4 = y4a_pt : res : y4_pt;     %Let y-array drive change from Frame 4a to 4
   else y_4a4 = -y4a_pt : res : -y4_pt;   y_4a4 = -y_4a4;  end    %Otherwise, flip array direction.
   size_y4a4 = size(y_4a4,2)-1;          %Size of y_4a4 array
```

```
   if x4_pt >= x4a_pt,                  %If x4 is greater than or equal to x4a,
      x_4a4 = x4a_pt : abs(x4_pt-x4a_pt)/size_y4a4 : x4_pt;  %Create x-array based on size of y-array
   else
      x_4a4 = -x4a_pt : abs(x4_pt-x4a_pt)/size_y4a4 : -x4_pt;   x_4a4 = -x_4a4;  %Create x-array based on size of y-
array
   end
   if z4_pt >= z4a_pt,
      z_4a4 = z4a_pt : abs(z4_pt-z4a_pt)/size_y4a4 : z4_pt;  %Create z-array based on size of y-array
   else
      z_4a4 = -z4a_pt : abs(z4_pt-z4a_pt)/size_y4a4 : -z4_pt;   z_4a4 = -z_4a4;  %Create z-array based on size of y-
array
   end
   if size(x_4a4, 2)<=1,  x_4a4 = x4a_pt + y_4a4.*0;  end    %If there is no change in x, create array of same mag.
   if size(z_4a4, 2)<=1,  z_4a4 = z4a_pt + y_4a4.*0;  end    %If there is no change in z, create array of same mag.
%Case III: Greatest change between frames is in z-direction:
   else       %the biggest change is in the z-direction
      if z4_pt >= z4a_pt,
         z_4a4 = z4a_pt : res : z4_pt;     %Let z-array drive change from Frame 4a to 4
      else z_4a4 = -z4a_pt : res : -z4_pt;   z_4a4 = -z_4a4;  end    %Otherwise, flip array direction.
      size_z4a4 = size(z_4a4,2)-1;          %Size of z_4a4 array
      if x4_pt >= x4a_pt,                   %If x4 is greater than or equal to x4a,
         x_4a4 = x4a_pt : abs(x4_pt-x4a_pt)/size_z4a4 : x4_pt;  %Create x-array based on size of z-array
      else
         x_4a4 = -x4a_pt : abs(x4_pt-x4a_pt)/size_z4a4 : -x4_pt;  x_4a4 = -x_4a4;  end    %Otherwise, flip array
direction.
      if y4_pt >= y4a_pt,                   %If y4 is greater than or equal to y4a,
         y_4a4 = y4a_pt : abs(y4_pt-y4a_pt)/size_z4a4 : y4_pt;  %Create y-array based on size of z-array
      else
         y_4a4 = -y4a_pt : abs(y4_pt-y4a_pt)/size_z4a4 : -y4_pt;  y_4a4 = -y_4a4;  end    %Otherwise, flip array
direction.
      if size(x_4a4, 2)<=1,  x_4a4 = x4a_pt + z_4a4.*0;  end    %If there is no change in x, create array of same mag.
      if size(y_4a4, 2)<=1,  y_4a4 = y4a_pt + z_4a4.*0;  end    %If there is no change in y, create array of same mag.
   end
end

%%%Create a point at the end of Joint 6%%%
joint6_length = 0.2171;             %Joint 5 axis to face of hand roll joint (m)
l6 = 0 : res : joint6_length;       %Create an array based on the distance between Joints 5 and 6

%beta = pitch,                          alpha_2 = yaw,          %gamma = roll
   %Note: "alpha" is taken by the DH parameters
beta = theta(2)+theta(3)+pi/2+theta(5);   alpha_2 = theta(1);      gamma = theta(4);
   %Note: pi/2 is added to theta(3) because it is subtracted in astepgui.m due to the frame transformations.
   %Those transformations do not apply here

R_yaw = [cos(alpha_2) -sin(alpha_2) 0;...       %Rotation matrix for yaw
      sin(alpha_2) cos(alpha_2) 0;...
      0 0 1];

R_pitch = [cos(beta) 0 sin(beta);...            %Rotation matrix for pitch
         0 1 0;...
         -sin(beta) 0 cos(beta)];

R_roll = [1 0 0;...                         %Rotation matrix for roll
        0 cos(gamma) -sin(gamma);...
        0 sin(gamma) cos(gamma)];

R = R_yaw*R_roll*R_pitch;                   %Pitch first, then roll, then yaw (order is critical)
x_change_5 = l6;      y_change_5 = l6.*0;       z_change_5 = l6.*0;      %Create end point corresponding to alpha =
beta = gamma = 0
pos_5 = [x_change_5; y_change_5; z_change_5];                       %Put coordinates in a vectory array
pos_6 = R * pos_5;                                          %Rotate vector based on angles

x_56 = x4_pt + pos_6(1,:);  y_56 = y4_pt + pos_6(2,:);  z_56 = z4_pt + pos_6(3,:);      %Add new location to previous
location (x4, y4, z4)
x_56_pt_loc = size(x_56,2);    y_56_pt_loc = size(y_56,2);    z_56_pt_loc = size(z_56,2);
x_56_pt = x_56(x_56_pt_loc);    y_56_pt = y_56(y_56_pt_loc);    z_56_pt = z_56(z_56_pt_loc);


%%Create a point at the end of the End-Effector (the tool tip)%%%
```

```
joint_7_length = ee_length;         %This is the end-effector length from Joint 6 to tool tip (meters)
l_7 = 0 : res : joint_7_length;     %Create an array based on the distance between Joints 5 and 6

x_change_7 = l_7;      y_change_7 = l_7.*0;       z_change_7 = l_7.*0;   %Create end point corresponding to alpha =
beta = gamma = 0
pos_7 = R * pos_5;                  %Rotate vector based on angles

x_67 = x_56_pt + pos_7(1,:);  y_67 = y_56_pt + pos_7(2,:);  z_67 = z_56_pt + pos_7(3,:);
x_67_pt_loc = size(x_67,2);     y_67_pt_loc = size(y_67,2);     z_67_pt_loc = size(z_67,2);
x_67_pt = x_67(x_67_pt_loc);    y_67_pt = y_67(y_67_pt_loc);    z_67_pt = z_67(z_67_pt_loc);

x_67_array = [x_67_array, x_67_pt];    %Add latest tool tip x-coord. to array
y_67_array = [y_67_array, y_67_pt];    %Add latest tool tip y-coord. to array
z_67_array = [z_67_array, z_67_pt];    %Add latest tool tip z-coord. to array

      theta_2_deg = theta_2_deg + workspace_iter;           %Iterate theta_2_deg (degrees)
   end
   theta_1_deg = theta_1_deg + workspace_iter;            %Iterate theta_1_deg (degrees)
end

figure(6)       %Create figure for 3-D workspace plot
plot3(x0_pt, y0_pt, z0_pt, '+', 'LineWidth', line_width*lw_mult), hold on       %Plot origin of Frame 0
plot3(x_67_array, y_67_array, z_67_array, 'r+', 'LineWidth', line_width), hold on       %Plot end of the EE

%%%Create Jaguar Base%%%
R = [radius radius];                                    %Create array with x = radius and y = radius
N = 25;                                                 %Number of mesh segments comprising cylinder
[X,Y,Z] = cylinder(R,N);                                %Create x, y, and z components of cylinder
C = zeros(2,N);                                         %Generate C to serve as basic colormap (will be lime green)
surf(height*Z-height,Y,X-radius,C); hold on %Produce cylindrical surface
                                                        %Third variable represents the cylinder axis
[r,s,t] = sphere(N);                                    %Create a sphere with NxN segments
C2 = zeros(N,N);                                        %Create variable C2 to serve as a colormap (lime green again)
surf(radius*r,radius*s,radius*t-radius,C2);             %Generate sphere

if ~jaguar,
   %%%Create Upper Jaguar Cylinder%%%
   surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on %Produce cylindrical surface
                                                        %Third variable represents the cylinder axis
   [r,s,t] = sphere(N);                                 %Create a sphere with NxN segments
   C2 = zeros(N,N);                                     %Create variable C2 to serve as a colormap (lime green again)
   surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2); %Generate sphere
   %%%Create Strut Between Jaguar Cylinders%%%
    %Create matrices of x and y to create plane for strut:
   [X_strut, Y_strut] = meshgrid(strut_loc : .01 : strut_loc + strut_width);
   Z_strut = X_strut + Y_strut;                         %Create z array (this will ultimately be the strut height
   Zcol_loc = (size(X_strut,2)+1)/2;                    %Want the middle column of the Z matrix, this is an index
   Zcol = Z_strut(:,Zcol_loc);                          %Find the column corresonding to the index Zcol_loc
   Zmax = max(abs(Zcol));                               %Find the maximum value in the column vector
   Zcol = Zcol + Zmax;                                  %Add to previous Zcol to translate the matrix to zero
   Y_strut = 0.*Y_strut;                                %Reset y values to zero (will assume strut has no thickness)
   count = 1;      Znew = [];                           %Initilize counter and Znew matrix
   while count <= size(Z_strut,2),                      %Create a new matrix consisting entirely of Zcol
      Znew = [Znew, Zcol];                              %This will produce a rectangular figure
      count = count + 1;                                %Iterate counter
   end
   Z_strut_max = max(abs(Znew));                        %Find the maximum value in the Znew matrix
   Z_strut_max = Z_strut_max(1,1);                       %Want only ONE maximum value
   Z_normalized = (Znew./Z_strut_max);                  %Normalize the Z values
   Z_strut = Z_normalized * Jag_offset;                 %Multiply the normalized value by the strut height
   C3 = zeros(size(Z_strut,2),size(Z_strut,2));         %Create variable C3 to serve as a colormap (lime green again)
   surf(X_strut,Y_strut,Z_strut,C3)                     %Plot the strut surface
end

%%%Create Sample Cotiainers%%%
%Create Sample Container 1:
R_sc = [radius_sc radius_sc];                           %Create array with x = radius and y = radius
N_sc = 40;                                              %Number of mesh segments comprising cylinder
[X_sc,Y_sc,Z_sc] = cylinder(R_sc,N_sc);                 %Create x, y, and z components of cylinder
X_sc = X_sc - x_loc_sc;                                 %X translation to sample container position (m)
```

```
Y_sc = Y_sc - y_loc_sc;                          %Y translation to sample container position (m)
Z_sc = height_sc*Z_sc - z_loc_sc;                %Adjust size and translate to container postion (m)
Xb_sc = X_sc(1,:); Yb_sc = Y_sc(1,:);            %Isolate single rows from the cylinder matrices
Zb_sc = Z_sc(2,:) - height_sc;                   %Isolate z-coord. and translate to cylinder base
C_sc = ones(2, N_sc);                            %Fix color to blue based on cylinder size
%Create Sample Container 2:
X2_sc = X_sc;   Y2_sc = -Y_sc;  Z2_sc = Z_sc;          %Base position of S.C. 2 on position of S.C. 1
X2b_sc = X2_sc(1,:);  Y2b_sc = Y2_sc(1,:);             %Isolate single rows from the cylinder matrices
Z2b_sc = Z_sc(2,:) - height_sc;                        %Isolate z-coord. and translate to cylinder base

surf(X_sc, Y_sc, Z_sc, C_sc), hold on                  %Produce cylindrical sample container
fill3(Xb_sc, Yb_sc, Zb_sc,'r')                         %Plot sample container baseplate
surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on               %Repeat for S.C. #2
fill3(X2b_sc, Y2b_sc, Z2b_sc,'r')


title('Work Envelope Plotted in 3-D Frame', 'FontWeight', 'bold', 'FontSize', title_size)
xlabel('X-axis', 'FontSize', title_size), ylabel('Y-axis', 'FontSize', title_size), zlabel('Z-axis', 'FontSize', title_size)
axis([xmin xmax ymin ymax zmin zmax])
%view(0, 0)        %Viewpoint specification (AZ,EL)  (-37.5, 30) is the MATLAB default
                   %(0,0) is the x-z plane,  (0,90) is the x-y plane
grid on


figure(7)          %Plot of Manipulator in X-Z Plane
plot3(x0_pt, y0_pt, z0_pt, '+', 'LineWidth', line_width*lw_mult), hold on      %Plot point at Frame 0 origin
%Plot end of EE (tool tip):
   plot3(x_67_array, y_67_array, z_67_array, 'r+', 'LineWidth', line_width), hold on
surf(height*Z-height,Y,X-radius,C), hold on                                    %Produce cylindrical surface
surf(radius*r,radius*s,radius*t-radius,C2), hold on                            %Generate sphere
if ~jaguar,
   surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on                    %Produce cylindrical surface
                                                                               %Third variable represents the cylinder axis
   surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);                    %Generate sphere
   surf(X_strut,Y_strut,Z_strut,C3)                                            %Plot the strut surface
end
surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                          %Produce cylindrical sample container
fill3(Xb_sc, Yb_sc, Zb_sc,'r')                                                 %Plot sample container baseplate
surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                       %Repeat for S.C. #2
fill3(X2b_sc, Y2b_sc, Z2b_sc,'r')
title('Work Envelope Plotted in X-Z Plane', 'FontWeight', 'bold', 'FontSize', title_size)
xlabel('X-axis', 'FontSize', title_size), ylabel('Y-axis', 'FontSize', title_size), zlabel('Z-axis', 'FontSize', title_size)
axis([xmin xmax ymin ymax zmin zmax])
grid on
view(0,0)          %Viewpoint specification (AZ,EL), (0,0) is the x-z plane


figure(8)          %Plot of Manipulator in X-Y Plane
plot3(x0_pt, y0_pt, z0_pt, '+', 'LineWidth', line_width*lw_mult), hold on      %Plot point at Frame 0 origin
%Plot end of EE (tool tip):
   plot3(x_67_array, y_67_array, z_67_array, 'r+', 'LineWidth', line_width), hold on
surf(height*Z-height,Y,X-radius,C), hold on                                    %Produce cylindrical surface
surf(radius*r,radius*s,radius*t-radius,C2), hold on                            %Generate sphere
if ~jaguar,
   surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on                    %Produce cylindrical surface
                                                                               %Third variable represents the cylinder axis
   surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);                    %Generate sphere
   surf(X_strut,Y_strut,Z_strut,C3)                                            %Plot the strut surface
end
surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                          %Produce cylindrical sample container
fill3(Xb_sc, Yb_sc, Zb_sc,'r')                                                 %Plot sample container baseplate
surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                       %Repeat for S.C. #2
fill3(X2b_sc, Y2b_sc, Z2b_sc,'r')
title('Work Envelope Plotted in X-Y Plane', 'FontWeight', 'bold', 'FontSize', title_size)
xlabel('X-axis', 'FontSize', title_size), ylabel('Y-axis', 'FontSize', title_size), zlabel('Z-axis', 'FontSize', title_size)
axis([xmin xmax ymin ymax zmin zmax])
grid on
view(0,90)         %Viewpoint specification (AZ,EL), (0,90) is the x-y plane
```

```
figure(9)          %Plot of Manipulator in Y-Z Plane
plot3(x0_pt, y0_pt, z0_pt, '+', 'LineWidth', line_width*lw_mult), hold on          %Plot point at Frame 0 origin
%Plot end of EE (tool tip):
   plot3(x_67_array, y_67_array, z_67_array, 'r+', 'LineWidth', line_width), hold on
surf(height*Z-height,Y,X-radius,C), hold on                                        %Produce cylindrical surface
surf(radius*r,radius*s,radius*t-radius,C2), hold on                                %Generate sphere
if ~jaguar,
   surf(height*Z-height,Y,X-radius + Jag_offset,C); hold on                        %Produce cylindrical surface
                                                                                   %Third variable represents the cylinder axis
   surf(radius*r,radius*s,radius*t-radius + Jag_offset,C2);                        %Generate sphere
   surf(X_strut,Y_strut,Z_strut,C3)                                                %Plot the strut surface
end
surf(X_sc, Y_sc, Z_sc, C_sc), hold on                                              %Produce cylindrical sample container
fill3(Xb_sc, Yb_sc, Zb_sc,'r')                                                     %Plot sample container baseplate
surf(X2_sc, Y2_sc, Z2_sc, C_sc), hold on                                           %Repeat for S.C. #2
fill3(X2b_sc, Y2b_sc, Z2b_sc,'r')
title('Work Envelope Plotted in Y-Z Plane', 'FontWeight', 'bold', 'FontSize', title_size)
xlabel('X-axis', 'FontSize', title_size), ylabel('Y-axis', 'FontSize', title_size), zlabel('Z-axis', 'FontSize', title_size)
axis([xmin xmax ymin ymax zmin zmax])
grid on
view(90,0)         %Viewpoint specification (AZ,EL), (90,0) is the y-z plane

if spread,      %If plots are to be spread across the window:
   if imac,     %If the figures are to be displayed on 2 Mac monitors:
      x_zoom = .15;
      y_zoom = .375;
      set(figure(9), 'units', 'normalized', 'outerposition', [.15 .25  x_zoom y_zoom])
      set(figure(8), 'units', 'normalized', 'outerposition', [0   .25  x_zoom y_zoom])
      set(figure(7), 'units', 'normalized', 'outerposition', [.15  1   x_zoom y_zoom])
      set(figure(6), 'units', 'normalized', 'outerposition', [0    1   x_zoom y_zoom])
   end
   if laptop,  %If the plots are to be shown on a standard Windows laptop
      y_zoom = .48075;
      x_zoom = 1/3;
      y_base = .039;
      set(figure(9), 'units', 'normalized', 'outerposition', [.6666   y_base  x_zoom y_zoom])
      set(figure(8), 'units', 'normalized', 'outerposition', [.3333   y_base  x_zoom y_zoom])
      set(figure(7), 'units', 'normalized', 'outerposition', [0       y_base  x_zoom y_zoom])
      set(figure(6), 'units', 'normalized', 'outerposition', [.6666   .52     x_zoom y_zoom])
   end
end
toc                                                                                %End timer
fprintf('Total Number of Iterations Performed: %.4g\n', counter)                   %Output number of iterations
speed = counter/toc;                                                               %Iterations per second
fprintf('Iterations Performed Per Second: %.7g\n\n', speed)                        %Output calculation speed
```

H.10 Function TransformWork.m

```
function [T] = TransformWork(a, alpha, d, theta)
% Returns T transform matrix for manipulator kinematics per
% Craig eq. 3.6 (Intro. to Robotics, 3rd ed.)

%Row 1 of tranformation matrix
   T(1,1) = cos(theta);
   T(1,2) = -sin(theta);
   T(1,3) = 0;
   T(1,4) = a;
   %Row 2 of transformation matrix
   T(2,1) = sin(theta)*cos(alpha);
   T(2,2) = cos(theta)*cos(alpha);
   T(2,3) = -sin(alpha);
   T(2,4) = -sin(alpha)*d;
   %Row 3 of transformation matrix
   T(3,1) = sin(theta)*sin(alpha);
   T(3,2) = cos(theta)*sin(alpha);
   T(3,3) = cos(alpha);
   T(3,4) = cos(alpha)*d;
   %Row 4 of transformation matrix
   T(4,1) = 0;
   T(4,2) = 0;
   T(4,3) = 0;
   T(4,4) = 1;

T = [T(1,1) T(1,2) T(1,3) T(1,4);...
    T(2,1) T(2,2) T(2,3) T(2,4);...
    T(3,1) T(3,2) T(3,3) T(3,4);...
    T(4,1) T(4,2) T(4,3) T(4,4)];
```

H.11 Function R03testf.m

```
function [R03test] = R03testf(theta2, a, alpha, d, sym, nframes)

for i = 1:nframes,
   %TransformMat computes transformation, rotation, and position matrices
   [T(:,:,i), R(:,:,i), P] = TransformMat(a(i),alpha(i),d(i),theta2(i), sym);   %Transformations from frame i to i-1
   if i == 1, T1 = T(:,:,i); R1 = R(:,:,i); P1 = P; end
   if i == 2, T2 = T(:,:,i); R2 = R(:,:,i); P2 = P; end
   if i == 3, T3 = T(:,:,i); R3 = R(:,:,i); P3 = P; end
   if i == 4, T4 = T(:,:,i); R4 = R(:,:,i); P4 = P; end
   if i == 5, T5 = T(:,:,i); R5 = R(:,:,i); P5 = P; end
   if i == 6, T6 = T(:,:,i); R6 = R(:,:,i); P6 = P; end
end
%Get W Transformations (W(i) = W(i-1)*R(i):
[W0, W1, W2, W3, W4, W5, W6] = TransformW(R1, R2, R3, R4, R5, R6);

%Get positions of coordinate frames:
[dx1, dx2, dx3, dx4, x0, x1, x2, x3, x4] = TransformPos(W0, W1, W2, W3, W4, W5, P1, P2, P3, P4, P5, P6);

R03test = W3;
```

H.12 Function sample_container.m

```
function [X_sc, Y_sc, Z_sc, C_sc, X2_sc, Y2_sc, Z2_sc, X3_sc, Y3_sc, Z3_sc,...
   X4_sc, Y4_sc, Z4_sc, X_lower, X_upper, Y_lower, Y_upper, Z_lower, Z_upper]...
   = sample_container(containers, x_sc, y_sc, z_sc, pitch_sc, roll_sc, yaw_sc, sc_plot)

%Sample Container Details
%radius_sc = .1206;                  %Sample container inner radius (m) - (corresponds to Di = 9.493 in)
radius_sc = .1365;                   %Sample container outer radius (m) - (corresponds to Do = 10.75 in)
height_sc = .2667;                   %Sample continer height (m) - (corresonds to h = 10.5 in)
   %Default Container Position:
   %x_loc_sc = -.25;   y_loc_sc = .3350;   z_loc_sc = -.3318;
if sc_plot,                          %If using the user-values...
   x_loc_sc = x_sc;                  %Dist. from origin in x-direction (m)
   y_loc_sc = y_sc;                  %Dist. from origin in y-direction (m)
   z_loc_sc = z_sc;                  %Dist. from origin in z-direction (m)
else                                 %Use default locations
   x_loc_sc = -.12;                  %Dist. from origin in x-direction (m)
   y_loc_sc = .335;                  %Dist. from origin in y-direction (m)
   z_loc_sc = -.3318;                %Dist. from origin in z-direction (m)
   pitch_sc = 0;                     %Default pitch angle (rad)
   roll_sc = 0;                      %Default roll angle (rad)
   yaw_sc = 0;                       %Default yaw angle (rad)
end

%%%Create Sample Containers%%%
%Create Sample Container 1:
R_sc = [radius_sc radius_sc];                %Create array with x = radius and y = radius
N_sc = 25;                                   %Number of mesh segments comprising cylinder
[X_sc,Y_sc,Z_sc] = cylinder(R_sc,N_sc);      %Create x, y, and z components of cylinder
Z_sc = height_sc*Z_sc;                       %"cylinder" sets Z_sc as array from 0 to 1, height_sc scales it
Xb_sc = X_sc(1,:); Yb_sc = Y_sc(1,:);        %Isolate single rows from the cylinder matrices
Zb_sc = Z_sc(2,:) - height_sc;               %Isolate z-coord. and translate to cylinder base
C_sc = ones(2, N_sc);                        %Fix color to blue based on cylinder size

%Rotation matrices for sample container position adjustments
R_yaw = [cos(yaw_sc) -sin(yaw_sc) 0;...      %Rotation matrix for sample container yaw (Frame 0 z-axis)
      sin(yaw_sc) cos(yaw_sc) 0;...
      0 0 1];
R_pitch = [cos(pitch_sc) 0 sin(pitch_sc);... %Rotation matrix for sample container pitch (Frame 0 y-axis)
         0 1 0;...                           %Pitch over y0-axis...
        -sin(pitch_sc) 0 cos(pitch_sc)];
R_roll = [1 0 0;...                          %Rotation matrix for sample container roll (Frame 0 x-axis)
        0 cos(roll_sc) -sin(roll_sc);...     %Roll over x0-axis
        0 sin(roll_sc) cos(roll_sc)];
R = R_yaw*R_pitch*R_roll;                    %Combine rotataion matrices into one matrix - note the order...
T = [R(1,1) R(1,2) R(1,3) x_loc_sc;...       %Create transformation matrix
    R(2,1) R(2,2) R(2,3) y_loc_sc;...        %Cylinder translation occurs here
    R(3,1) R(3,2) R(3,3) z_loc_sc;...
    0     0     0     1];
[N M] = size(X_sc);                          %Determine size of matrices
Ones_sc = ones(N, M);                        %Create matrix of that size composed of ones

pos = [X_sc(1,:); Y_sc(1,:); Z_sc(1,:); Ones_sc(1,:)];   %Form a position vector for one side of cylinder
pos2 = T*pos;                                            %Multiply position vector by transformation matrix
posb = [X_sc(2,:); Y_sc(2,:); Z_sc(2,:); Ones_sc(1,:)];  %Form a position vector for other side of cylinder
pos2b = T*posb;                                          %Multiply other position vector by transformation matrix

X_sc = pos2(1,:);                    %Isolate new x-coords. for one side of cylinder
Y_sc = pos2(2,:);                    %Isolate new y-coords. for one side of cylinder
Z_sc = pos2(3,:);                    %Isolate new z-coords. for one side of cylinder
Xi_sc = pos2b(1,:);                  %Isolate new z-coords. for other side of cylinder
Yi_sc = pos2b(2,:);                  %Isolate new z-coords. for other side of cylinder
Zi_sc = pos2b(3,:);                  %Isolate new z-coords. for other side of cylinder

Y_adjust = radius_sc - radius_sc*cos(roll_sc);   %Dist. container shifted from Jaguar due to rotation (m)
%Y_adjust = 0;
```

```
X_sc = [X_sc; Xi_sc];                       %Combine new x-coords., needed for cylinder plotting
Y_sc = [Y_sc; Yi_sc];                %Combine new y-coords., needed for cylinder plotting
Y_sc = Y_sc - Y_adjust;              %Shift y_coord. back to Jaguar
Z_sc = [Z_sc; Zi_sc];                %Combine new z-coords., needed for cylinder plotting

%Create Sample Container 2:
X2_sc = X_sc;   Y2_sc = -Y_sc;  Z2_sc = Z_sc;          %Base position of S.C. 2 on position of S.C. 1
                                                       %Negative y flips it to other side of Jaguar
%Create Sample Containers 3 & 4:
if sc_plot,                  %If using the user-inputs...
   X3_sc = 0;  Y3_sc = 0;  Z3_sc = 0;                  %Set variables to arbitrary values
   X4_sc = 0;  Y4_sc = 0;  Z4_sc = 0;                  %Set variables to arbitrary values
else                         %Use default values...
   X3_sc = X_sc + 2*radius_sc;       %Offset another row of sample containers by the container diameter
   Y3_sc = Y_sc;  Z3_sc = Z_sc;      %Y & Z positions don't change from SC 1

   X4_sc = X3_sc;  Y4_sc = -Y3_sc;  Z4_sc = Z3_sc;     %Adjust just as from SC 1 to 2
end

X_lower = (max(X_sc(1,:))+min(X_sc(1,:)))/2;     %Calculate average x postion on lower surface (m)
X_upper = (max(X_sc(2,:))+min(X_sc(2,:)))/2;     %Calculate average x postion on upper surface (m)
Y_lower = (max(Y_sc(1,:))+min(Y_sc(1,:)))/2;     %Calculate average y postion on lower surface (m)
Y_upper = (max(Y_sc(2,:))+min(Y_sc(2,:)))/2;     %Calculate average y postion on upper surface (m)
Z_lower = (max(Z_sc(1,:))+min(Z_sc(1,:)))/2;     %Calculate average z postion on lower surface (m)
Z_upper = (max(Z_sc(2,:))+min(Z_sc(2,:)))/2;     %Calculate average z postion on upper surface (m)

if containers,                       %Is sample containers are to be output to the command window...
%Output container angles to command window:
   fprintf('\nProgrammed Sample Container Angles:\n')
   fprintf('   Pitch: %.4g deg.\n', pitch_sc*180/pi)
   fprintf('   Roll:  %.4g deg.\n', roll_sc*180/pi)
   fprintf('   Yaw:   %.4g deg.\n', yaw_sc*180/pi)
%Output coordinates to command window:
   fprintf('\nCoordinates of Sample Container Cylinder Center:\n')
   fprintf('Upper Surface:\n')
   fprintf('   X-Coordinate: %.4g m\n', X_upper)
   fprintf('   Y-Coordinate: %.4g m\n', Y_upper)
   fprintf('   Z-Coordinate: %.4g m\n', Z_upper)
   fprintf('Lower Surface:\n')
   fprintf('   X-Coordinate: %.4g m\n', X_lower)
   fprintf('   Y-Coordinate: %.4g m\n', Y_lower)
   fprintf('   Z-Coordinate: %.4g m\n', Z_lower)
end
```

References

[1] Ishitsuka, M., Sagara, S., and K. Ishii. "Dynamics Analysis and Resolved Acceleration Control of an Autonomous Underwater Vehicle Equipped with a Manipulator," *Proc. of IEEE International Symposium of Underwater Technology*, Taipei, Taiwan, Apr. 20-23, pp. 277-281, 2004.

[2] Des Marais, D.J., et al. "The NASA Astrobiology Roadmap," *International Journal of Astrobiology*, Volume 3, Number 2, pp. 219-235, 2003.

[3] "Earth's Complex Complexion." WHOI.edu. 2004. Woods Hole Oceanographic Institute. 8 Aug. 2008 <http://www.whoi.edu/oceanus/viewArticle.do?id=2496>.

[4] Shank, T.M. "The Evolutionary Puzzle of Seafloor Life," *Oceanus Magazine*, Volume 42, Number 2, pp. 1-8, 2004.

[5] Monkman, G.J., Hesse, S., Steinmann, R., and H. Schunk. *Robot Grippers*. Wiley-VCH, Germany. 2007.

[6] "Application Examples - MAG Welding." ReisRobotics.de. 2007. Reis Robotics. 16 Aug. 2008 <http://www.reisrobotics.de/us/APPLICATIONS/Welding+technique.html>.

[7] "STS-123 Mission Overview Briefing Graphics." NASA.gov. 2008. NASA. 16 Aug. 2008 <www.nasa.gov>.

[8] Yuh, J. "Underwater Robotics," *Proc. of IEEE International Conference on Robotics and Automation*, San Francisco, CA, Apr. 24-28, pp. 932-937, 2000.

[9] "Hydra Magnum 100 hp ROV." Oceaneering.com. 2006. Oceaneering International, Inc. 16 Aug. 2008 <http://www.oceaneering.com/ROV.asp?id=596>.

[10} "Orion 7P/7R." Schilling.com. 2008. Schilling Robotics, Inc. 16 Aug. 2008 <http://www.schilling.com/products_ManipulatorSystems_ORION.php>.

[11] Whitcomb, L.L. "Underwater Robotics: Out of the Research Laboratory and Into the Field," *Proc. of IEEE International Conference on Robotics and Automation*, San Francisco, CA, Apr. 24-28, 2000.

[12] "Featured Photo." WHOI.edu. 2008. Woods Hole Oceanographic Institute. 16 Aug. 2008 < http://www.whoi.edu/oceanus/index.do>.

[13] Shank, T. “Survey of Future Needs and Upgrades for Deep-Submergence Biological Research,” University-National Oceanographic Laboratory System Meetings, 2004.

[14] Frantz, C.M. “Potential End-Effectors for the Autonomous Sample Collection of Hydrothermal Vent Sites,” Unpublished SSL Internal Report DT20-0036, Aug. 10, 2005.

[15] Gruntz, D.W. “Development and Evaluation of a Tool for EVA or Robotic Planetary Sampling,” Master of Science Thesis, University of Maryland, College Park, 2007.

[16] “Neutral Buoyancy Photo Archives.” SSL.UMD.edu. 2005. University of Maryland Space Systems Laboratory. 18 Aug. 2008 <http://spacecraft.ssl.umd.edu/SSL.photos/photos.html>.

[17] Lauletta, A. “Harmonic Drive Gearing,” *Gear Product News*, Apr., pp. 33-36, 2006.

[18] Smith, W. “Lift Capacity of ASTEP Manipulator,” Unpublished SSL Internal Report DT20-0009, June 7, 2005.

[19] Smith, W. “Re: next Tuesday evening.” Email to Craig Lewandowski. 14 May 2008.

[20] “Harmonic Drive Gearing.” HDInfoNet.com. 2008. HDinfoNet. 17 July 2008 <http://www.hdinfonet.com/advantages.html>.

[21] Bynum, W.L., Wise, M.A., Sliwa, N.E., and F.H. Willard. “The Parallel Jaw Gripper: A Robotic End-Effector System,” *NASA Technical Memorandum*, Document No. NASA-TM-101271, 1988.

[22] Rothbart, H.A. *Cam Design Handbook*. McGraw-Hill, New York. 2004.

[23] “Track Rollers.” McMaster.com. 2008 McMaster-Carr Catalog. 16 July 2008 <http://www.mcmaster.com>.

[24] George, S.B., and S. Boone. “The Ectosymbiont Crab Dissodactylus Mellitae–Sand Dollar Mellita Isometra Relationship,” *Journal of Experimental Marine Biology and Ecology*, No. 294, pp. 235-255, 2003.

[25] Lim, O., Cho, Y., Lee, J., and W. Lee. “Optimum Design for Raceway Groove Curvature of a Ball Bearing,” *Proc. of AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Long Beach, CA, Sept. 6-8, pp. 1-6, 2000.

[26] Bao, S. “Grip Strength and Hand Force Estimation,” Washington State Department of Labor and Industries Technical Report, Report No. 65-1-2000, May, 2000.

[27] Carignan, C. “Samuraiikin(rprwrist).nb,” Mathematica Software Program, 28 July 2006.

[28] Craig, J.J. *Introduction to Robotics Mechanics and Control*. Pearson-Prentice Hall, Upper Saddle River, New Jersey. 2005.

[29] Kim, J., Chung, W.K., and J. Yuh. “Dynamic Analysis and Two-Time Scale Control for Underwater Vehicle-Manipulator Systems,” *Proc. of IEEE International Conference on Intelligent Robots and Systems*, Las Vegas, NV, Oct., pp. 577-582, 2003.