5-1-2018

IMPROVEMENT OF THE FORM AND FUNCTION OF THE AL5D ROBOTIC ARM

Cristian Almendariz

IMPROVEMENT OF THE FORM AND FUNCTION OF THE AL5D ROBOTIC ARM

by

CRISTIAN ALMENDARIZ

Presented to the Faculty of the Honors College of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

HONORS BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2018

ACKNOWLEDGMENTS

Completion of this project, degree, and overall progress in my academic career could not have occurred without the combined efforts of the faculty and staff of the University of Texas at Arlington's Honors College and Department of Mechanical Engineering. I would like to express my most sincere thanks to my faculty mentor, Dr. Raul Fernandez, for leading the Senior Design lectures and providing expert advice and guidance. I would also like to share my most sincere thanks to the Honors College for providing opportunity to those students who seek to expand their academic experience.

Thank you, Dr. Aditya Das and Dr. Hakki Sevil, for serving as our Faculty Advisors in our robotics endeavor and our hosts at the University of Texas at Arlington's Research Institute. I would also like to thank the UTARI Facilities Staff, IT staff, and researchers: Jared Beaty, Ron LaPosa, Cody Lundberg, and the late Mr. Norman Spayd who supported our time at the institute.

The hard work of Tesleem Lawal, Iris Romero, and Eric McDaniels, the ARLM team members, allowed for great steps forward of our project and without them this could not have been accomplished. They were supportive and accommodating in my research.

I am absolutely thankful for the friends and family who have stood by my side in support while I worked to complete my honors degree. I owe the entirety of my academic achievements to my parents and grandparents. They taught me the fundamentals of working hard, staying dedication, and the meaning of sacrifice.

May 4, 2018

ABSTRACT

IMPROVEMENT OF THE FORM AND FUNCTION OF THE AL5D ROBOTIC ARM

Cristian Almendariz, B.S. Mechanical Engineering

The University of Texas at Arlington, 2018

Faculty Mentor: Raul Fernandez

The University of Texas at Arlington Research Institute's Automation and Intelligent Systems Lab assigned the task of developing a multi-robotic framework to autonomously identify, sort, and deliver various objects to desired locations in a workspace. One main component of the framework is the AL5D Robotic Arm, which serves as the sorting machine. In an attempt to mitigate the limitations of the hobby grade motors that the AL5D uses, modifications were made to implement a position feedback to its controller. The AL5D's servo motor showed negligible change in the system response with the application of a digital compensator to the electronic controller. Other alterations made to the AL5D include the implementation of position teaching program and a 3D printed cover for the joints and links. The suggested alterations and the additional functionality implemented onto the AL5D robotic arm serve as a proof of concept for future improvements.

TABLE OF CONTENTS

ACKNOWLEDGMENTS iii

ABSTRACT iv

LIST OF ILLUSTRATIONS vii

LIST OF TABLES viii

Chapter

1. INTRODUCTION 1

2. UTARI LIFELONG LEARNING MACHINE 3

    2.1 Project Scope 3

    2.2 Team members and Tasks 4

        2.2.1 Navigation & Primary Vision 4

        2.2.2 Secondary Vision System 5

    2.3 Senior Design Assigned Task 5

        2.3.1 Linear Rail System 5

        2.3.2 Inverse Kinematics 6

        2.3.3 Programming Motion 7

3. METHODS OF IMPROVEMENT 9

    3.1 Modifying Servo Motors 9

        3.1.1 How Servo Motors Work 9

        3.1.2 Control System Improvement 10

    3.2 3D Printing Custom Components 11

3.3 Method of Position Teaching 11

4. IMPLEMENTATION 12

4.1 Modified Servo Motors 12

4.2 3D Printed Components 18

4.3 Implementing the Position Teaching Mode 19

5. CONCLUSION 21

Appendix

A. SIMPLIFIED INVERSE KINEMATICS 23

B. ROS TOPIC AND NODE FLOW CHART 26

C. ROS NODES 28

D. 3D PRINTED COMPONENTS 37

E. POSITION TEACHING PROGRAM 44

F. POSITION TEACHING VISUALS 47

REFERENCES 50

BIOGRAPHICAL INFORMATION 52

LIST OF ILLUSTRATIONS

Figure		Page
2.1	Workspace of Robotic Framework for Automated Sorting	3
2.2	Workspace Improvement with Linear Rail System	6
2.3	Linear Rail with AL5D Rendering	6
2.4	Visual of AL5D Joint Locations and Plane of Motion	7
3.1	Components of a Servo Motor	10
3.2	Feedback Loop of Unmodified Servo Motor	10
3.3	Feedback Loop of Modified Servo Motor	11
4.1	Unmodified Servo Motor Block Diagram	12
4.2	Unmodified Root Locus and Step Response	13
4.3	LabVIEW Block Diagram for Measuring Angular Position	14
4.4	LabVIEW VI Displaying Servo Motor Response	14
4.5	Modified Servo Response – No Compensators	15
4.6	Comparison of the Transient Response Time Approximations for the Simulated Response (a) and the Measured Response Overlaid (b).	16
4.7	Modified Servo Motor Block Diagram	16
4.8	Oscillatory Response Brought on by Bad PID Tuning	17
4.9	Response with PID Compensator and Proper Tuning	17
4.10	First Layer of 3D Printing on PolyPrinter	18
4.11	The AL5D Robotic Arm with 3D printed Link Covers	19

LIST OF TABLES

Table Page

2.1 Custom Data Types Used in ROS 8

CHAPTER 1

INTRODUCTION

Automation is an important component of the modern world. The implementation of machines to complete routine and simple tasks in nearly every industry has created an immense rise in productivity. The disruptive nature of integrating automated processes can be seen on a global economic scale, leading to some fearing the loss of human labor forces in many industries [1]. Automation serves to empower human productivity and thus support higher skilled and specialized workers for more job markets. Combine machine learning with automation technology and the result is a safer workplace with higher productivity yields [2].

"Artificial intelligence isn't magic. In fact, it's not really about intelligence at all. It's better understood as simply the latest advance in automation"

Jerry Kaplan, author of Humans Need Not Apply [3].

It is projected that the robotics industry will reach $100 billion by 2020. The next decade for robotics will be defined by the means of implementation of new devices into other industries. With the addition of Artificial Intelligence, new robotic systems will be capable of enhanced machine learning and prove more useful in complex tasks [4]. As of today, there are many applications for robotic systems. Well known devices, such as the Da Vinci Surgical Robot, enhance the capabilities of the world's surgeons, leading to safer and more innovative surgeries. The implementation of force sensing control systems has allowed such devices as surgical robots to be developed [5]. With the combination of force

sensing control systems, artificial intelligence, and machine learning, the creation of Collaborative Robots has allowed smaller and safer industrial robots to be implemented into more of the world's manufacturing facilities. Despite the higher level of safety, there is still a concern for the lack of research in this field of robotics [2].

Collaborative robots are unique due to their ability to directly interact with human operators. The space between human operators and robotic technology is ever decreasing, leading to streamlined integration into existing manufacturing lines. The safety of these systems is still under investigation but is proving promising. The collaborative robots can utilize any combination of vision systems, force sensing, and virtual surfaces to constrain their motions and allow workers to increase interaction [6].

Current research is focused on improving the safety systems used in collaborative robots that are implemented in existing manufacturing lines, as well as how to best implement them for optimized performance and human interaction. Optimizing the transition of collaborative robots to receiving and performing new tasks is major area of focus for today's research [7].

CHAPTER 2

UTARI'S LIFELONG LEARNING MACHINES

2.1 Project Scope

The University of Texas at Arlington Research Institute's Automation and Intelligent Systems Lab assigned the Senior Design Team, ARLM, the task of developing a multi-robotic framework as a proof of concept for showcasing machine learning. ARLM was provided the means of how to best showcase machine learning in a simple and effective manner. Under the supervision of the faculty advisors, Dr. Das and Dr. Sevil, the team decided on a system that sorts colored blocks and delivers these blocks to areas within a given workspace. To accomplish this task, ARLM would use a robotic arm coupled to a secondary vision system to sort colored blocks. The blocks would be placed onto rovers, which would use a primary vision system to navigate to a desired delivery location.

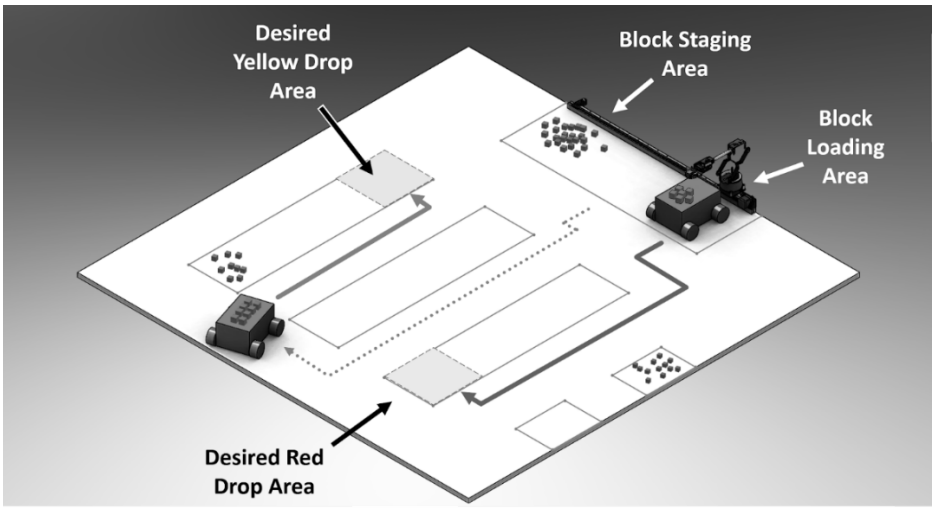


Figure 2.1: Workspace of Robotic Framework for Automated Sorting

2.2 Team Members and Tasks

The Senior Design Team, ARLM, was comprised of four students, including myself. The task set before the team was divided into four main subtasks to be worked on individually and in a manner that would not allow one item to greatly limit the progress of any other. The common communications platform between each robotic system is a software called Robotic Operating System (ROS). ROS is a cross-platform communication program that allows for data abstraction and communication between many machines and many languages. More generalized, ROS is a software that allows for programs to talk to each other whether or not they speak the same language. The advantage of using ROS is that we are not limited to having to find a single software package or hardware that can encompass the entirety of the project. We are able to find any system and implement it with little trouble.

2.2.1 Navigation & Primary Vision

Tesleem Lawal served as the team captain for ARLM and primarily worked on the development of the navigation system that would utilize the Hercules Rovers to transport the sorted blocks around the workspace. Eric McDaniels assisted Tesleem with this task. Tesleem used ROS and available software packages to get the rovers to be able to navigate within the workspace. The primary vision system used was VICON, a motion capture system comprised of a series of cameras. The VICON Camera System was made available to us by UTARI. VICON allows for accurate positioning of the Rovers within the defined workspace, that workspace being a cage-like structure that has been built in a large UTARI Lab. Eric worked with Tesleem on getting the VICON system to feed positions to the rover for navigation development. The staff at UTARI previously had the VICON system

calibrated to the Hercules Rovers for other projects, allowing for easy implementation of VICON into our project.

2.2.2 Secondary Vision System

Iris Romero used ROS software packages and OpenCV, an open source vision program, to develop a smaller, secondary camera system for detecting individual colored blocks. The specified colored block's locations are sent to the AL5D Robotic Arm and used for sorting. She used a 1080p 60fps USB camera with custom mounting and a custom 3D printed case. Iris placed the camera in a manner that overlooks the area within the reach of the AL5D robotic arm.

2.3 Senior Design Assigned Task

The fourth sub task for ARLM was the development of the robotic arm system. The robotic arm chosen was the AL5D Robotic Arm by Lynxmotion. Implementation of the arm into the robotic framework consisted of integrating ROS communication, implementing motion control software, and increasing the working area of the arm.

2.3.1 Linear Rail System

To accomplish the task outlined by the Automation and Intelligent Systems lab, it was decided to increase the working volume of the AL5D's reach with the implementation of a linear rail. This was decided in order to allow for a larger area for block placement and allow for the Rovers to navigate closer to the arm for loading and unloading. The increase in area is presented graphically in Figure 2.2.

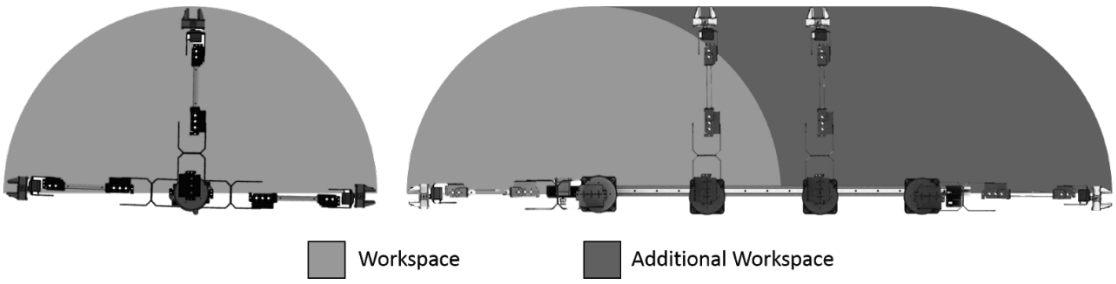


Figure 2.2: Workspace Improvement with Linear Rail System

The linear rail system used a hobby grade CNC rail and an associated sliding block. A stepper motor was implemented as the means of actuation, and custom components were produced to mount the AL5D to the sliding block and the motor to the rail. The completed rail system rendering is presented in Figure 2.3.

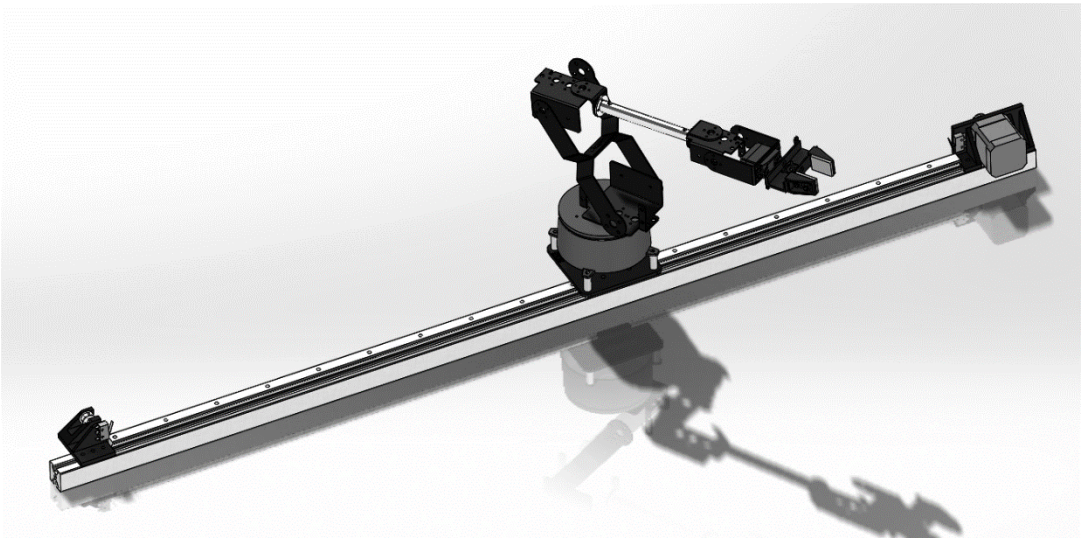


Figure 2.3: Linear Rail with AL5D Rendering

2.3.2 Inverse Kinematics

The program that would be used to control the motion of the AL5D robotic arm in 3D space would need to follow a strict algorithm to be able to direct the motion of each joint and thus direct the position and orientation of the AL5D gripper. To do this, inverse kinematics were used and implemented into a python script to allow for joint angles to be

calculated, leading to position and orientation control of the end effector of the AL5D. To find the Inverse Kinematics, the Modified Denavit–Hartenberg method presented by Craig, in *Intro to Robotics* was used and can be found in Appendix A [8]. The motion of the AL5D was reduced from a 5 DOF robotic arm to a 3 DOF-single plane robotic arm (Figure 2.4). This was done to simplify the motion and complexity of the programming involved. With the implementation of the inverse kinematics, instead of hard coding every movement of every joint of the AL5D links, we only needed to input the desired position and orientation of the AL5D gripper in cartesian coordinates.

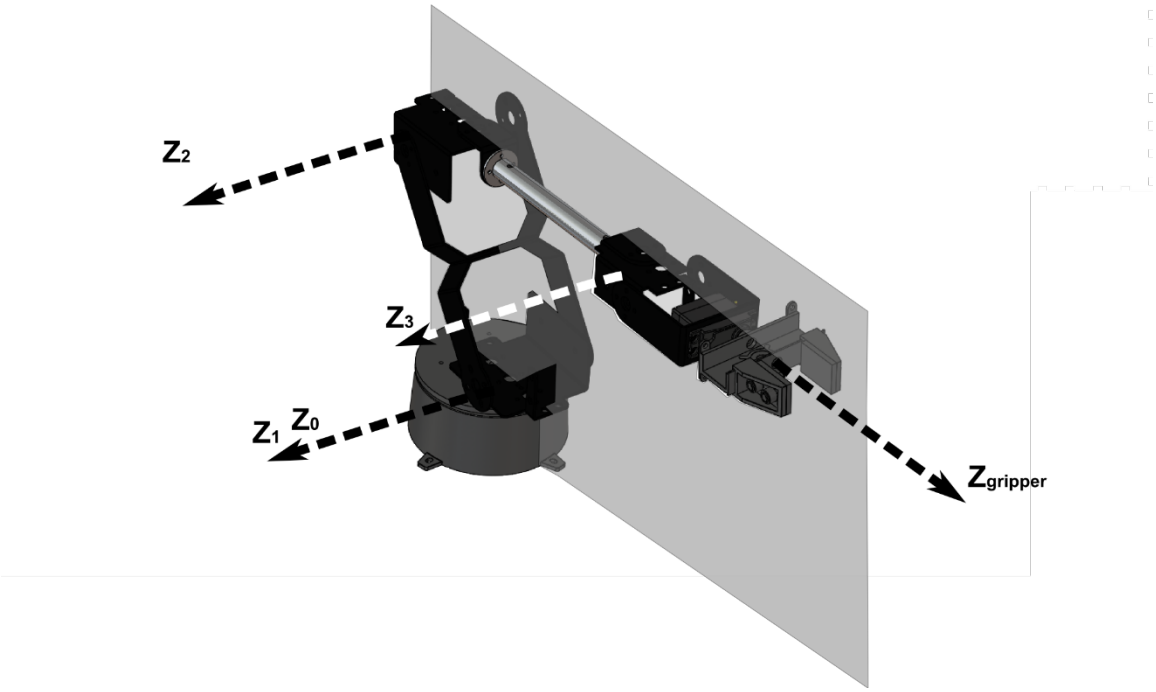


Figure 2.4: Visual of AL5D Joint Locations and Plane of Motion

2.3.3 Programming Motion

To program the AL5D and the other robotic systems, various Python and C++ scripts were written and utilized in ROS. Each script of code used lived within a ROS node. These nodes are instances of a terminal within the computer that houses the software. The AL5D robotic arm is controlled through four nodes. The “Navigator” node orchestrated the

process of retrieving and placing the desired blocks. Those steps being: aligning the AL5D with the desired block along the length of the linear rail, moving the arm to grab the block along the plane of motion previously defined, placing the block on the rover, and returning the arm to a home position. The “InverK” node housed the calculations for the inverse kinematics of the AL5D. The “Navigator” node would call on the “InverK” node to act as a function, returning each joint angle to place the gripper in the needed location. Every command that would move the servo motors of the AL5D needed to be passed from the “Navigator” node through a serial connection to the servo controller, the SSC-32U. This process was done by the “Servo_Run2” node. The “Stepper_Run2” node accomplished the equivalent task for getting the stepper motor to translate the AL5D along the rail system.

Each node in ROS communicates via a TCP/IP Protocol, taking advantage of a publish-subscribe system. This is done by sending data packets, either predefined data types or custom ones (called msg types), through topics. Each node can publish messages in topics or be subscribed to topics to receive the published messages. Every node written for the complete motion of the robotic arm sub-system can be found in Appendix B.

Table 2.1: Custom Data Types Used in ROS

EndEff	ItemPos	StepCtrl	FullCtrl	
Float64 x	Int64 x	Bool dir0	Uint16 pwm0	Uint16 pwm4
Float64 y	Int64 y	Float64 vel0	Uint16 pwm1	Uint16 pwm5
Float64 phi	Int8 cartID	Uint64 stepNum	Uint16 pwm2	Uint16 time0
			Uint16 pwm3	

CHAPTER 3

METHODS OF IMPROVEMENT

The AL5D serves as a very affordable and easy-to-use 5 DOF robotic arm. In order to maintain the affordability of the AL5D, Lynxmotion used commonly available hobby grade components for the construction of each arm. They also provided a variety of electronic controllers and controller software to purchase with the arm. The affordability of the components and controller results in reasonable limitations on performance and functions available to the user. Hitec servo motors stand as the primary servo motor used to control the rotation of each joint. The AL5D serves as a starting point in developing an understanding of robotic systems. If the AL5D is the only available system to a user, then there should be a means of improving the performance and functionality of the system. The work presented here serves as a proof of concept that improvements can be made on the AL5D while retaining as much of the original hardware as possible.

3.1 Modifying Servo Motors

3.1.1 How Servo Motors Work

The AL5D robotic arm uses analog servo motors to actuate each joint. A servo motor is comprised of a DC motor coupled to an output shaft and potentiometer via an internal gearing system (Figure 3.1). Once a position angle is desired and the command signal sent to the servo motor, the motor will spin, and the changes in angular position will be read by the potentiometer. The motor stops spinning once the difference between the angular position of the output matches the desired angular position that was sent as an input

command. The servo motor will constantly adjust its angular position to compensate for any changes or loading on the output shaft.

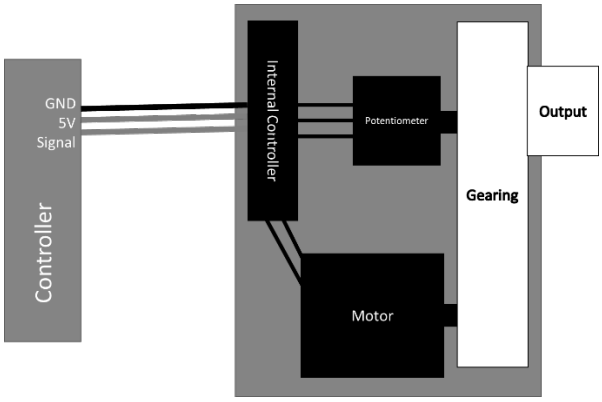


Figure 3.1: Components of a Servo Motor

This system is a self-contained feedback loop. Figure 3.2 shows a simplified model of the feedback system. In this case the motor, potentiometer, and the output are all physical components of the servo motor. The controller used to send the desired angular positions to the servo motor initially was the SSC-32U but was later switched to an Arduino UNO.

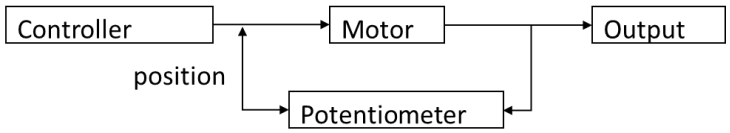


Figure 3.2: Feedback Loop of Unmodified Servo Motor

3.1.2 Control System Improvement

To improve the performance and the functionality of the analog servo motor, the angular position of the output is fed back to the controller. This is done by severing the connections between the potentiometer and the internal controller of the servo motor. With the angular position feeding back to the controller, a digital compensator can be used to fine tune the control. Another advantage to feeding back the position to the controller is

that additional software could implement the newly added feedback for new purposes and functions. Figure 3.3 depicts the angular position feedback to the controller.

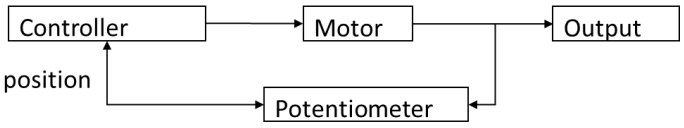


Figure 3.3: Feedback Loop of Modified Servo Motor

3.2 3D Printing Custom Components

The AL5D is comprised of powder coated brushed aluminum links and other components. Any modifications to the links or adding additional components requires deconstruction of the AL5D arm assembly and taking those components to a power tool. While an easy process for most individuals who have access to the proper tools and facilities, some may have a lack of such access. I designed and 3D printed custom link covers as a proof of concept that increased functionality could be added to the arm without the complete deconstruction of the AL5D robotic arm assembly. Having 3D printed components could accommodate additional hardware or electronics.

3.3 Method of Position Teaching

A position teaching mode was added to the functionality of the AL5D robotic arm to act as a proof of concept for new software that can easily be implemented. Software additions are not necessarily a part of the physical AL5D system but can be added to the external controller. Given the addition of angular feedback from servo motors to the external controller, the software can utilize this to provide a recording of angular positions, making it simple to implement a position teaching method. With the addition of more sensors and modifications, it should be possible to write more complex software to provide the whole AL5D system with increased functionality.

CHAPTER 4

IMPLEMENTATION

4.1 Modified Servo Motors

The AL5D robotic arm uses the Hitec brand of servo motors. An analysis of the physical control system and response of a Hitec servo motor was completed by Piateck of the University of Science and Technology, Krakow, Poland [9]. Piateck was able to determine the original unmodified control system transfer function, gain, k, and DC motor inertia, T, that comprises the analog servo motor (Figure 4.1). The value for gain, k, is (20.8), and that for DC motor inertia, T, is (0.0181).

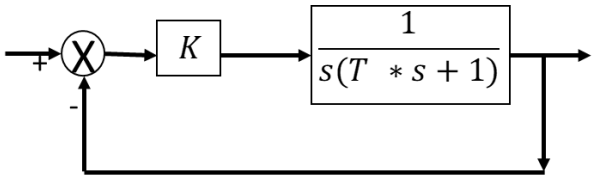


Figure 4.1: Unmodified Servo Motor Block Diagram [9]

This information was modeled using MATLAB to provide the root locus (Figure 4.2) and position response under a step input (Figure 4.3). The root locus of a control system is used to be able to predict the systems response given certain input parameters. The root locus of the unmodified, servo motor provides the damping ratio of 0.814 and the percent over shoot of 1.22%. These values are important in defining how the system responds to varying inputs. The damping ratio indicates how quickly and with how much oscillation a system will have when adjusting to an input; this is known as the transient response. A value less than one for the damping ratio, represents an underdamped system response. The

transient response will have a high number of oscillations and take a relatively long time to reach the desired change of state. A critically damped system will have a damping ratio of one and be quite quick in its response with no overshoot. A damping ratio greater than one will also have no overshoot but will take longer and longer to reach the desired state. This is an overdamped system. The percent overshoot shows how much of an initial overshoot the system will have during the transient response. The unmodified servo motor has a fairly good response with a damping ratio of 0.814, meaning that it has slight oscillations in its response, but it still reaches the desired state quickly.

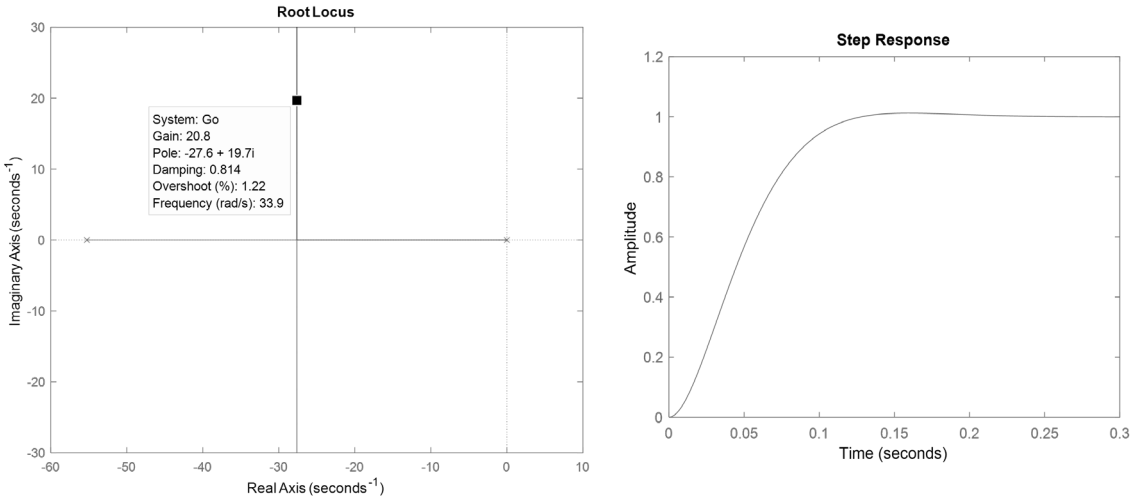


Figure 4.2: Unmodified Root Locus and Step Response

In implementing the modified Hitec servo motors, the first step was to determine whether the measured system response would match the response presented by Piateck. Piateck used the Hitec HS-475HB Standard Deluxe Servo while the one tested for this application was the Hitec HS-322HD Standard Deluxe Servo. Both servo motors require the same voltage and current and they both have similar response times at 0.23sec/60° for the HS-475HB and 0.19sec/60° for the HS-322HD. The modification that was made to the HS-322HD was the implementation of angular position feedback. The servo motor was

subjected to a 90° step input. A NI USB-6009 DAQ was used in conjunction with LabVIEW to measure and record this response. Figure 4.3 shows the wiring diagram for the LabVIEW program that was used. The LabVIEW DAQ Assistant was set to use a sample rate of 1000 Hz over two seconds. The results were recorded in excel while simultaneously displayed in the LabVIEW VI (Figure 4.4).

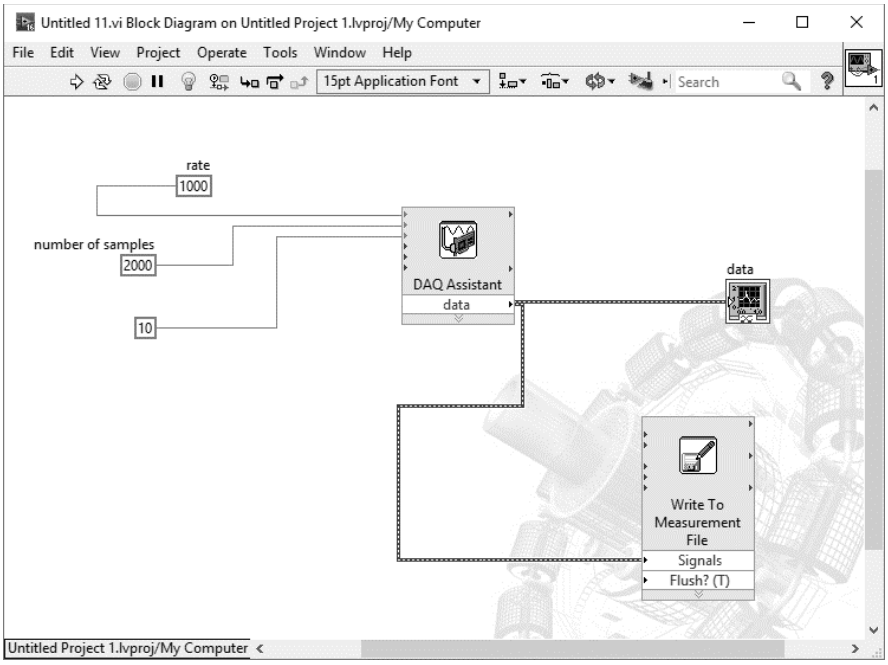


Figure 4.3: LabVIEW Block Diagram for Measuring Angular Position

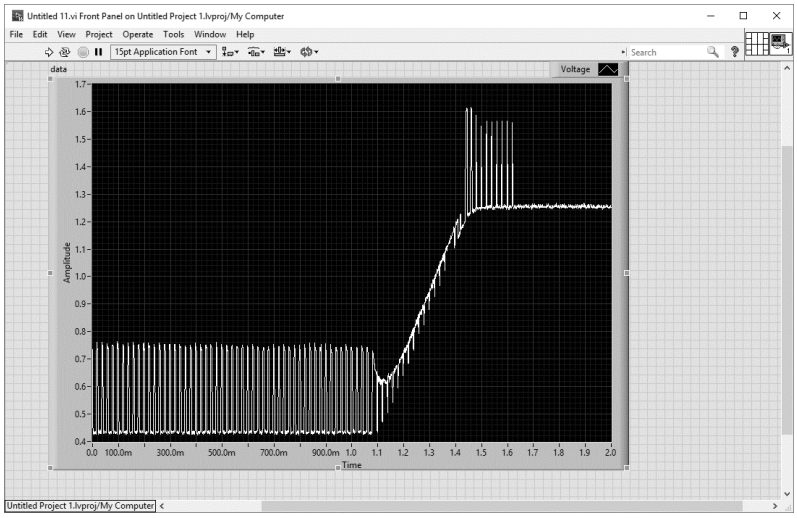


Figure 4.4: LabVIEW VI Displaying Servo Motor Response

The results from testing the modified servo motor are also displayed in Figure 4.5.

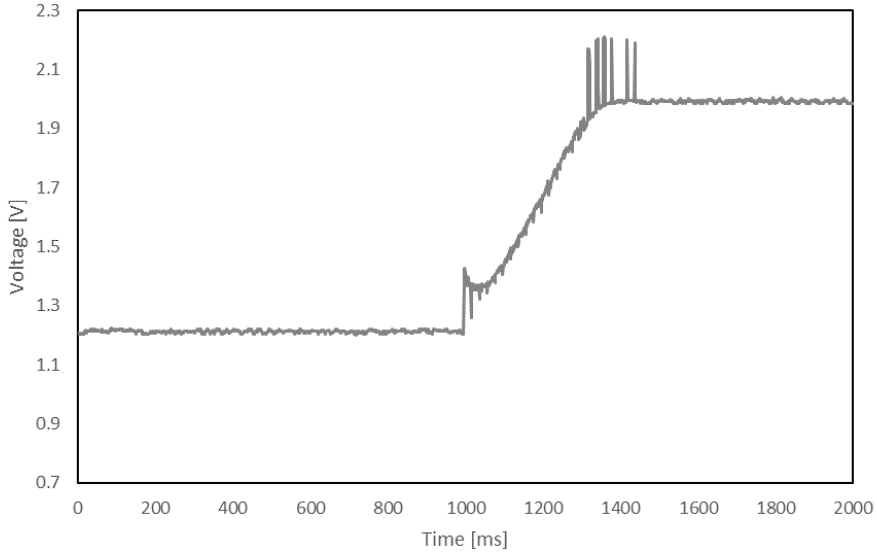
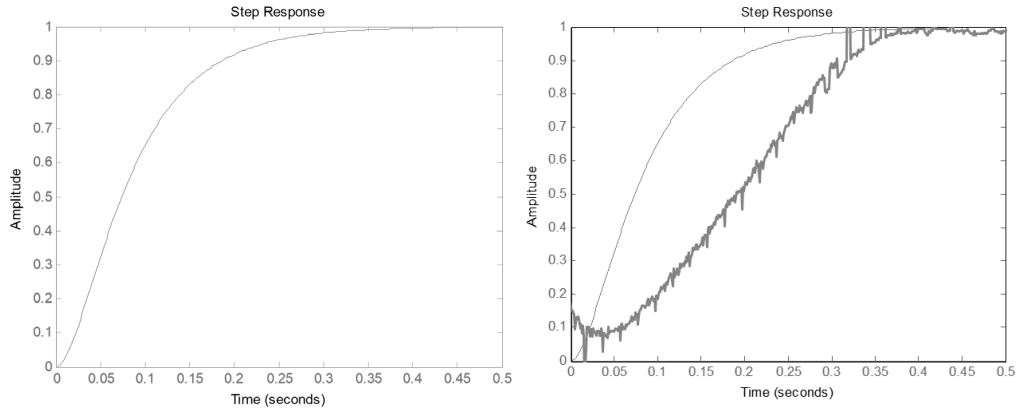


Figure 4.5: Modified Servo Response – No Compensators

The step input from a 90° shift in angular position correlated to a ~1V drop across the potentiometer and a transient response time of ~400ms. This result differs from the transient response time proposed by Piateck, that time being ~150ms for a 1V step input. The value that was achieved may have been from unideal conditions, either caused by the electronic controller or the measurement device, used to record the response. In order to accommodate for this, the value for gain, k, used in the simulated response of the servo motor, was changed from k = 20.8 to k = 10.8. The new value for gain, k, was determined by altering the simulated transfer function step response in MATLAB until the transient response time matched the recorded value for the unmodified control system with the assumption that the reported DC motor inertia, reported by Piateck, was still valid for an unloaded Hitec HS-475HB Standard Deluxe Servo. This assumption was made due to any change in value of the DC motor inertia, T, also altering the damping ratio of the system response, leading to either a simulated response with too much or not enough oscillation. The DC motor inertia, T, would not result in the desired change in transient response time

needed to match the measured response time while the gain, k, would. Figure 4.6 displays the resulting MATLAB simulation, now more closely matching the actual transient response time. The actual system response still varies from the simulated response.



(a) (b)

Figure 4.6: Comparison of the Transient Response Time Approximations for the Simulated Response (a) and the Measured Response Overlaid (b)

The original transfer function was modified to reflect the modified servo motors with the addition of a digital PID compensator integrated into the controller (Figure 4.7).

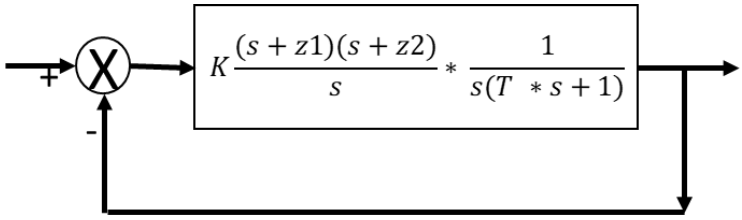


Figure 4.7: Modified Servo Motor Block Diagram

The servo controller was switched from the SSC-32U to an Arduino controller for ease of use in testing purposes. The Arduino controller used an existing PID library as the digital compensator in an attempt to make an improvement on the transient response time without sacrificing the damping levels already present in the servo motors. Initially, making changes to the parameters that control the tuning for the PID controller resulted in the response of the modified servo motor being thrown into an oscillatory response (Figure

4.8). The parameters of the PID controller are the proportional gain [Kp], integral gain [Ki], and a derivative gain [Kd], set at 20, 20, and 10, respectively.

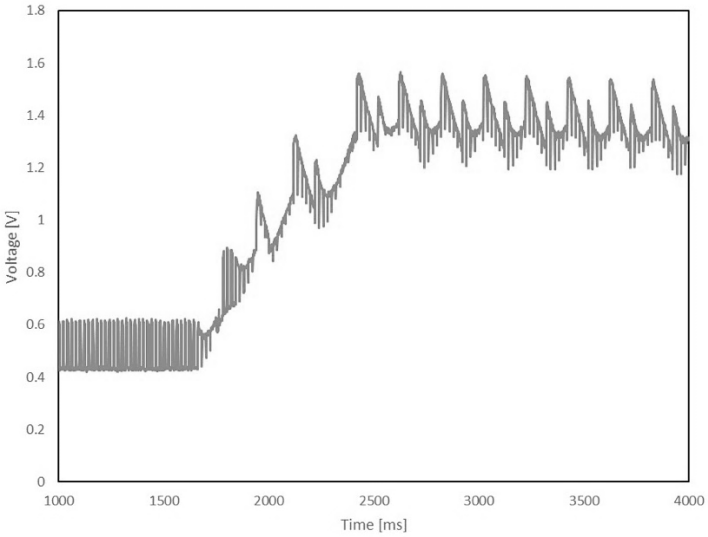


Figure 4.8: Oscillatory Response Brought on by Bad PID Tuning

The response of the system with more tuning of the PID parameters with the best results were Kp = 10, Ki=10, and Kd=1 (Figure 4.9).

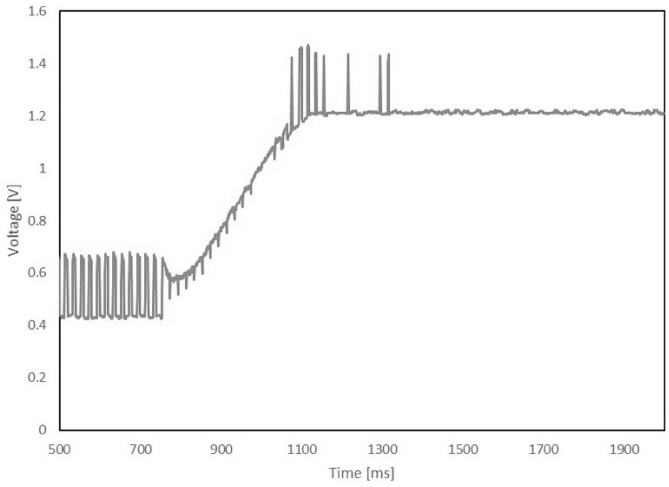


Figure 4.9: Response with PID Compensator and Proper Tuning

Despite the addition of the angular position feedback to the controller and the implementation of an Arduino based PID compensator, the best motor response achieved

matched the original response of the servo motor without the integrated Arduino PID compensator.

4.2 3D Printed Components

The 3D printed components that were added to the AL5D assembly are presented below, with the CAD models presented in Appendix C. There were 5 major components designed and printed for attachment to the AL5D. The components were designed in SolidWorks and printed on a PolyPrinter in The University of Texas at Arlington Central Library's FabLab. The components were designed to act as a wire management system. The aim was to retain the original dimensions and geometry of the AL5D; therefore, the additional components were needed to match the geometry of the existing links and be attached without making alterations. The printing material used was PLA and each layer had a thickness of 0.25mm (Figure 4.10).

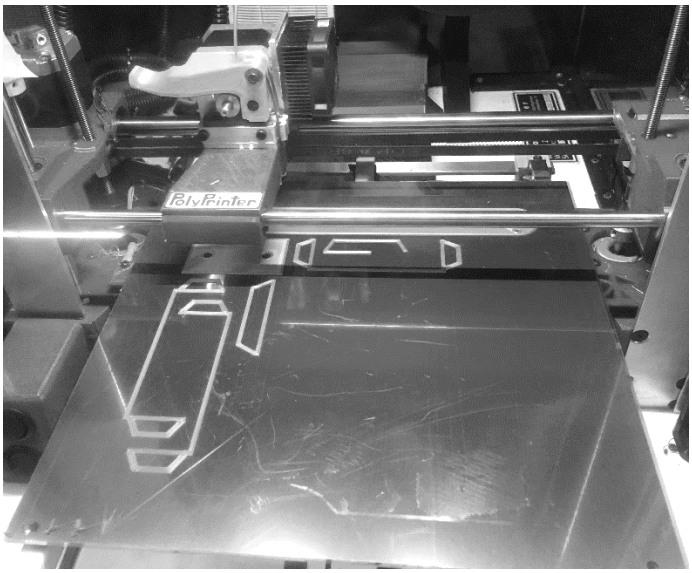


Figure 4.10: First Layer of 3D Printing on PolyPrinter

The 3D printed components were attached successfully and did not disrupt the range of motion of the arm given the explicit application used for the senior design project (Figure 4.11).

Figure 4.11: The AL5D Robotic Arm with 3D printed Link Covers

4.3 Implementing the Position Teaching Mode

The position teach mode was successfully implemented. The code was written in C and implemented using an Arduino as the electronic controller, available in Appendix D. The Basic PID Arduino library was used [10]. The function of teaching the robot arm a position was a proof-of-concept that new functions can be implemented that use the modified servos or additional sensors and feedback. To demonstrate the position teaching mode, first a button was also implemented into a separate circuit connected to the Arduino. The recording of the most recent angular position for each servo was initiated only when the button was pressed. While the button is pressed, the AL5D can be manipulated and placed into any orientation, the angles are then recorded as the button is released. The arm

then returns to a hardcoded default position. After five seconds the arm plays back the recorded position and will hold that position until the button is pressed again. Once the button is pressed for a second recording, the first angular position recorded for every joint is overwritten. This was done due to the limited memory of the Arduino. A single position being recorded stood as a proof-of-concept that something similar and improved could in fact be easily implemented. A video was produced to showcase the capability of the position teaching program. Several screenshots are available in Appendix E.

CHAPTER 5

CONCLUSION

The methods presented to improve the form and function of the AL5D robotic arm were meant to serve as a proof-of-concept that improvements could be made to the AL5D. The goal was to retain as much of the original hardware and electronics as possible. The methods of improvement include modifying the provided analog servo motors, adding 3D printed link covers, and to implement a position teaching program.

The Hitec servo motors were modified by implementing an angular position feedback to the electronic controller. This addition was utilized with an Arduino based PID compensator in an attempt to improve the transient response time of the servo motors when subjected to a step input of 90°. Given the negligible change in the system response, it is assumed that the quality of the servo motors will not allow for much improvement from what the manufacturer already provides as an internal controller for the given servo motor. Despite this, the addition of the angular position feedback allowed for the measurement of the system response, showing that it is possible to implement a tuning mechanism that can benefit some applications.

The addition of new 3D printed components (whether they be link covers, link replacements, or any other printed attachment) can be easily designed and fabricated with any conventional CAD software and 3D printer. This method of modification for improvement would allow for the easy addition or replacement of links. The printed components could be designed to house sensors, provide an altered geometry and thus

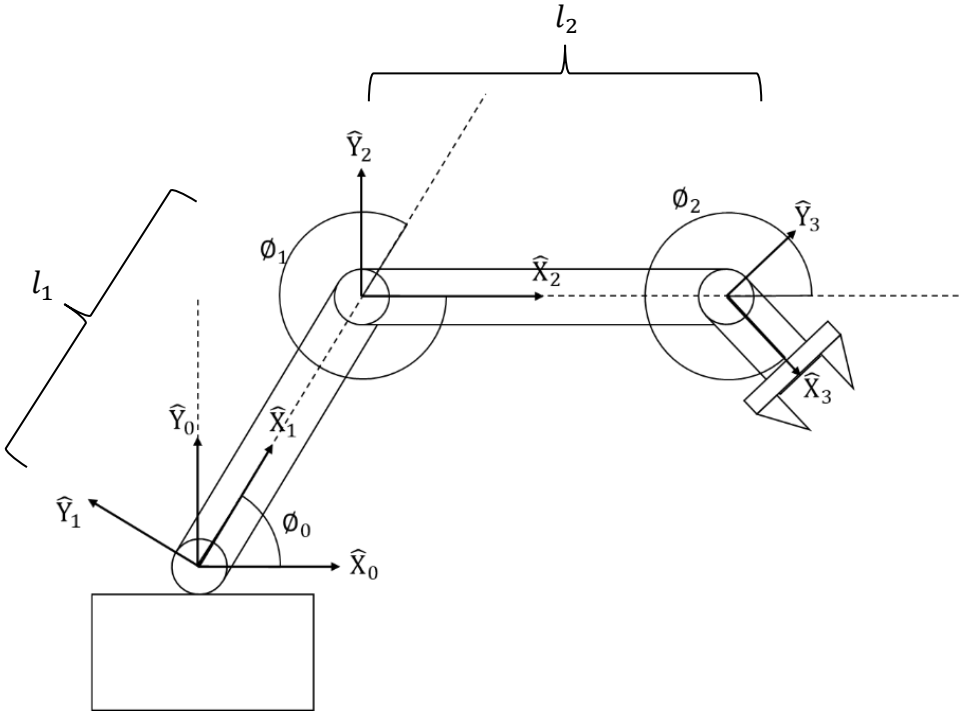
modified inverse kinematics or stand as a wire management component as demonstrated. If 3D printers are available to users, this method would be highly recommended as a means of providing modifications or additions to the AL5D robotic arm.

The implementation of a position feedback to the controller proved effective when coupled with additional software; a simple position teaching program written in C. This seemed the most significant advantage to adding the angular position feedback. As more information is fed back to the electronic controller of choice, more options for additional functionality are possible through the addition of adding new programs. Provided that the controller has the additional sources of feedback already implemented, then the addition of software to utilize that information is not limited by any physical capacity other than memory and processing capability of the servo controller.

APPENDIX A

SIMPLIFIED INVERSE KINEMATICS

Using the Modified Denavit–Hartenberg Method of Solutions for Inverse Kinematics [8]



C	N	#	Typ	Var	a	α	d	∅
0	1	0	R	∅	0	0	0	$\emptyset_0$
1	2	1	R	∅	l_1	0	0	$\emptyset_1$
2	3	2	R	∅	l_2	0	0	$\emptyset_2$

$$ {}^{0}_{1}\mathrm{T} = \begin{bmatrix} c0 & -s0 & 0 & 0 \\ s0 & c0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^{1}_{2}\mathrm{T} = \begin{bmatrix} c1 & -s1 & 0 & l_1 \\ s1 & c1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^{2}_{3}\mathrm{T} = \begin{bmatrix} c2 & -s2 & 0 & l_2 \\ s2 & c2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

$$ {}^{0}_{3}\mathrm{T} = \begin{bmatrix} c123 & -s123 & 0 & l_1c1 + l_2c12 \\ s123 & c123 & 0 & l_1s1 + l_2s12 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ \underbrace{{}^{0}_{3}\mathrm{T} = \begin{bmatrix} c123 & -s123 & 0 & l_1c1 + l_2c12 \\ s123 & c123 & 0 & l_1s1 + l_2s12 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{AL5D Generalized Kinematic Equations}} = \underbrace{\begin{bmatrix} c\varphi & -s\varphi & 0 & x \\ s\varphi & c\varphi & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Desired location and orientation of gripper}} $$

$$c\varphi = c123$$

$s\varphi$=s123

$$x = l_1 c1 + l_2 c12$$

$$y = l_1 s1 + l_2 s12$$

$$c_2 = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

$$s_2 = \pm\sqrt{1 - c_2^2}$$

$$\boxed{\emptyset_2 = Atan2(s_2, c_2)}$$

$$x = k_1 c1 - k_2 s1$$

$$y = k_1 s1 + k_2 c1$$

$$k_1 = l_1 + l_2 c2$$

$$k_2 = l_2 s2$$

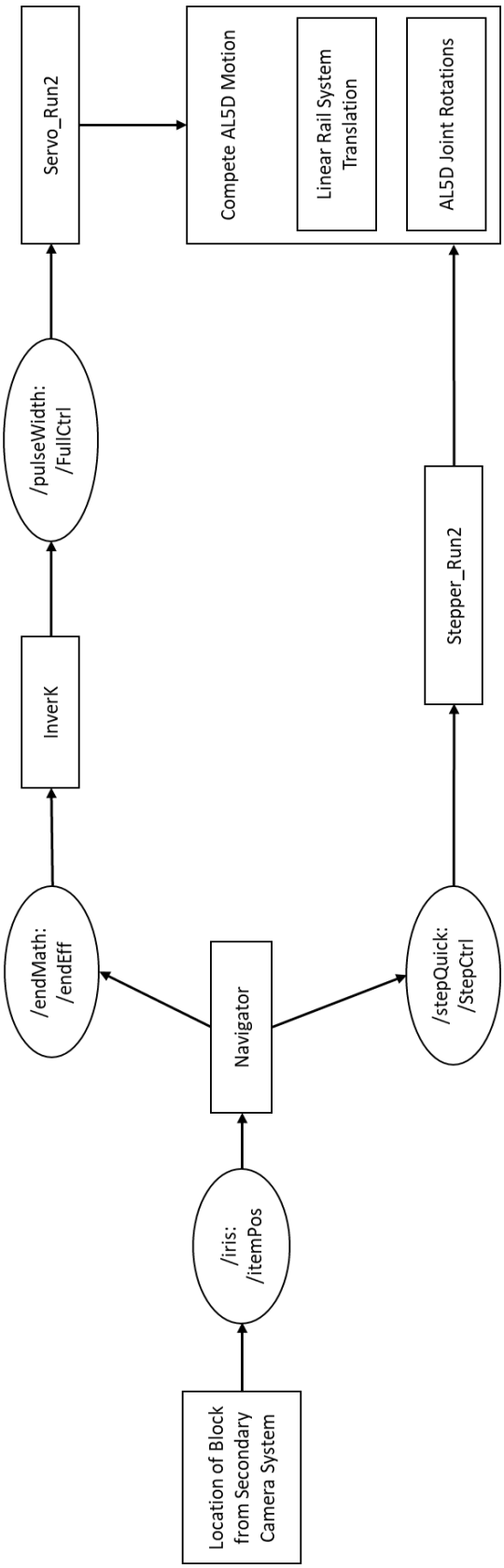
$$y = l_1 c1 + l_2 c12$$

$$\boxed{\emptyset_1 = Atan2(y, x) - Atan2(k_2, k_1)}$$

$$\boxed{\emptyset_3 = \varphi - \emptyset_1 - \emptyset_2}$$

APPENDIX B

ROS TOPIC AND NODE FLOW CHART


Location of Block from Secondary Camera System
/iris: /itemPos
Navigator
/endMath: /endEff
InverK
/pulseWidth: /FullCtrl
Servo_Run2
/stepQuick: /StepCtrl
Stepper_Run2
Compete AL5D Motion
Linear Rail System Translation
AL5D Joint Rotations

APPENDIX C

ROS NODES

Stepper_Run2.py

```
#!/usr/bin/env python
import serial
import sys
import time
import rospy
from std_msgs.msg import String
from al5d.msg import FullCtrl


#callback function: gets info from topic and constructs str to control all servos
#is the node for serial comms with ssc32u
def callback(data):
    rospy.loginfo(data)
    #pwm0:base pwm1:shoulder pwm2:elbow pwm3:wrist1 pwm4:wrist_rotate pwm5:claw
    servo = "#0P"+str(data.pwm0)+" "+"#1P"+str(data.pwm1)+" "+"#2P"+str(data.pwm2)+"
"+"#3P"+str(data.pwm3)+" "+"#4P"+str(data.pwm4)+" "+"#5P"+str(data.pwm5)+"T"+str(data.time0)+"\r"
    servo = servo.encode()
    print(servo)

    ssc32.write(servo)
    #time.sleep(2)
    #ssc32.write("#0P1500 #1P1500 #2P1500 #3P1500 #4P1500 #5P1500 T500\r".encode())
    print("end")


# Initiates ROS node to control servos, once topic has published data, callfunction ran
def servo_run2():
    # topic: pulseWidth
    # msg type: FullCtrl
    # msg is located in al5d package under msg dir
    rospy.init_node('servo_run2', anonymous=True)
    rospy.Subscriber("pulseWidth", FullCtrl, callback)
    rospy.spin()

#begins program, confirms start of node, opens serial port to ssc32u and also closes port
if __name__ == '__main__':
    ssc32 = serial.Serial("/dev/ttyUSB0", 9600)
    print("start")
    servo_run2()
    ssc32.close()
```

InverK.py

```
#!/usr/bin/env python

import rospy
import math
import time
from al5d.msg import StepCtrl
from al5d.msg import FullCtrl
from al5d.msg import ItemPos
from al5d.msg import EndEff

def callback(data):
    pub1 = rospy.Publisher('pulseWidth', FullCtrl, queue_size=10)
    rospy.loginfo(data)

    #conversion factor of pwm to
    k = 1000/90

    #link lengths [inches]
    l1 = 5.875
    l2 = 7.375

    x = data.x
    y = data.y
    phi = -90

    c2 = (math.pow(x,2) + math.pow(y,2) - math.pow(l1,2) - math.pow(l2,2)) / (2*l1*l2)

    #step1 = math.pow(c2,2)
    #step2 = 1-step1
    #step3 = math.fabs(step2)
    #step4 = math.sqrt(step3)

    si = -1*math.sqrt(math.fabs(1-math.pow(c2,2)))
    #si = -step4

    k1 = l1+l2*c2
    k2 = l2*si

    theta1 = (180/math.pi * (math.atan2(y,x) - math.atan2(k2,k1)))
    theta2 = (180/math.pi * (math.atan2(si,c2)))
    theta3 = phi - theta1 - theta2

    deg1 = theta1 - 90
    deg1o = theta1
    deg2 = theta2 + 90
    deg2o = theta2
    deg3 = phi - deg1o - deg2o + 90

    theta1 = k*deg1+1400
    theta2 = -k*deg2+1500
    theta3 = k*deg3+1450

    pwm0 = 1350
    pwm1 = int(theta1)
    pwm2 = int(theta2)
```

```
    pwm3 = int(theta3)
    pwm4 = 1600
    pwm5 = 1500
    time = 1000

    if pwm1 > 2500:
        pwm1 = 2500
    if pwm1 < 500:
        pwm1 = 500

    if pwm2 > 2500:
        pwm2 = 2500
    if pwm2 < 500:
        pwm2 = 500

    if pwm3 > 2500:
        pwm3 = 2500
    if pwm3 < 500:
        pwm3 = 500

    pub1.publish(pwm0, pwm1, pwm2, pwm3, pwm4, pwm5, time)
    pub1.publish(1500, 1500, 1500, 1500, 1500, 1500, 1000)
    print(pwm1)
    print(pwm2)
    print(pwm3)

def inversekin():
    rospy.init_node('inverseK', anonymous=True)
    rospy.Subscriber('endMath', EndEff, callback)
    rospy.spin()
    #main function

if __name__ == '__main__':
    print('start')
    inversekin()
    print('end')
```

Navigator.py

```
#!/usr/bin/env python

import rospy
import time
import math
from al5d.msg import StepCtrl
from al5d.msg import FullCtrl
from al5d.msg import ItemPos

def callback(data):
    global cart
    global blockLocation
    pub1 = rospy.Publisher('pulseWidth', FullCtrl, queue_size=10)
    pub2 = rospy.Publisher('stepQuick', StepCtrl, queue_size=10)

    print('entered callback')
    #callback loop to iris
    rospy.loginfo(data)
    pwm1 = -0.746*math.pow(data.y,2)-58.2983*(data.y)+1587.5353-200
    pwm2 = -8.602*math.pow(data.y,2)-39.3968*(data.y)+2081.2027
    pwm3 = -6.7599*math.pow(data.y,2)+23.8576*(data.y)+1096.3500+1000

    #k is the conversion from pixel distance to steps
    k=1
    blockLocation = k*data.x
    cart = data.cartID

    print('got data')

    def attack():
        #move down to surround block
        pub1.publish(1350, pwm1, pwm2, pwm3, 1600, 800, 3000)
        rospy.sleep(3)
        #close gripper
        pub1.publish(1350, pwm1, pwm2, pwm3, 1600, 2000, 50)
        rospy.sleep(1)
        print('attack')

    def return2ready():
        #prepares for motion by publishing pwm to joints
        pub1.publish(1350, 1400, 1500, 1500, 1600, 2000, 2000)
        rospy.sleep(2)
        print('return2ready')

    def standardRotation():
        #goes through middle position via pwm
        pub1.publish(1350, 1400, 1200, 1500, 1600, 800, 2000)
        rospy.sleep(3)
        print('std rot')

    def home():
        #returns to cart 1 location or cart 2 location
        #hardcoded block location to see if it corrects error
        global cart
        global blockLocation
```

```
        #blockLocation = 2500
        if cart > 2:
            cart = 2
        if cart == 1:
            blockLocation = blockLocation - 500
            pub2.publish(True, 0.005, blockLocation)
            temp = 0.01*blockLocation + 3
            rospy.sleep(temp)
        if cart == 2:
            pub2.publish(True, 0.005, blockLocation)
            temp = 0.01*blockLocation + 3
            rospy.sleep(temp)

        pub1.publish(1350, 1025, 1500, 1800, 1600, 2000, 1000)
        rospy.sleep(1.1)
        pub1.publish(1350, 1025, 1500, 1800, 1600, 800, 500)
        rospy.sleep(0.6)
        pub1.publish(1350, 1400, 1100, 1800, 1600, 800, 1000)
        rospy.sleep(1.1)

        if cart ==1:
            pub2.publish(True, 0.01, 500)
            rospy.sleep(5)
        print('home')

    def align():
        #aligns arm with block
        pub2.publish(False, 0.005, blockLocation)
        temp = 0.01*blockLocation
        rospy.sleep(temp)
        print('align')

    print('did functions')

    rospy.sleep(0.5)
    return2ready()
    align()
    standardRotation()
    attack()
    return2ready()
    home()

def navigator():
    rospy.init_node('navigator', anonymous=True)
    rospy.Subscriber('iris', ItemPos, callback)
    rospy.spin()
    #main function

if __name__ == '__main__':
    print('start')
    navigator()
    print('end')
```

Stepper_Run2.py

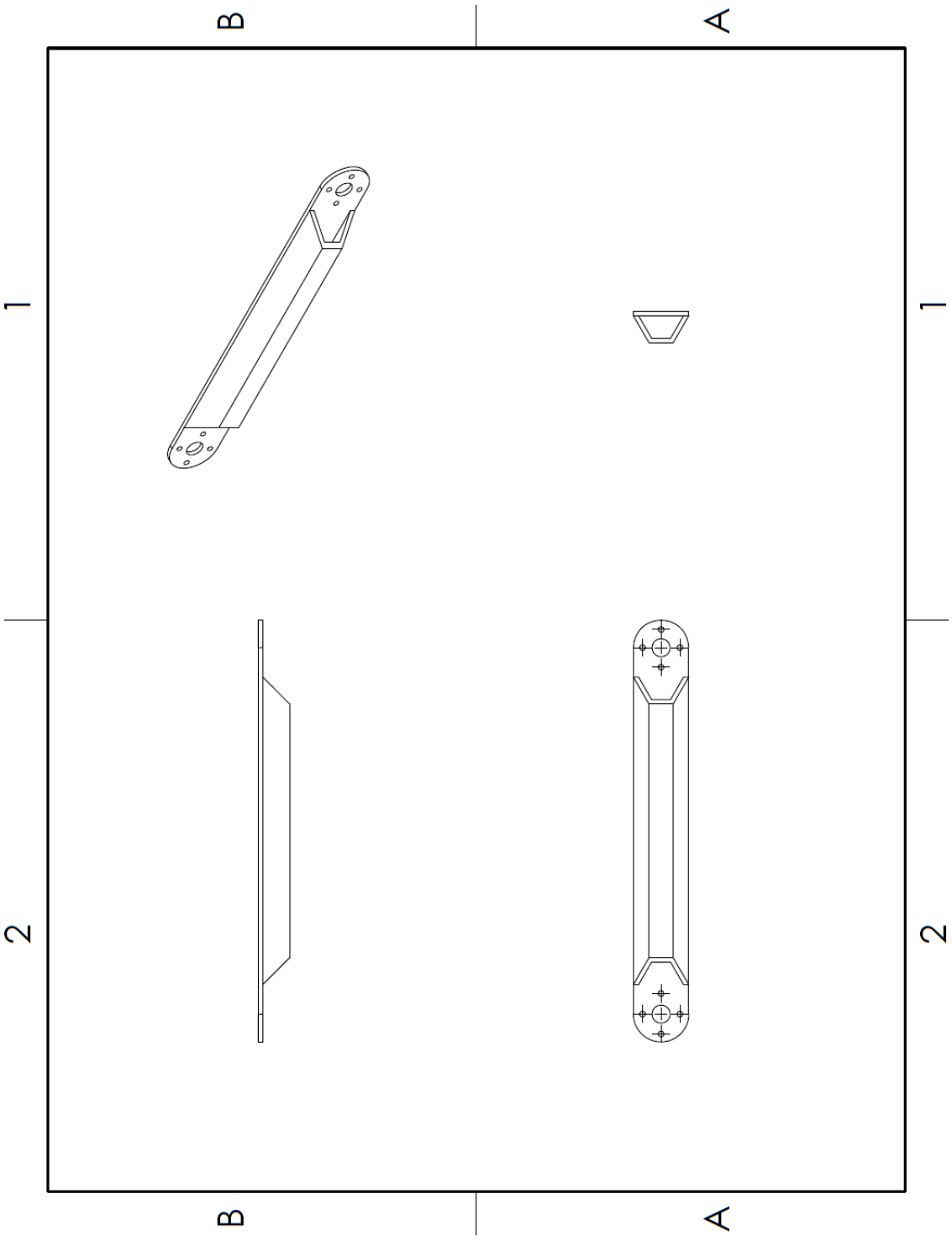
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import time
import rospy
#add msg file for stepper motor: #ofSteps, direction, and speed
from al5d.msg import StepCtrl
from std_msgs.msg import String
import RPi.GPIO as gpio

Dir0 = True
StepNum = 0
Vel0 = 0

#gpio.setmode(gpio.BCM)
#gpio.setwarnings(False)

class easydriver(object):
   gpio.setmode(gpio.BCM)
   gpio.setwarnings(False)

   def
__init__(self,pin_step=0,delay=0.1,pin_direction=0,pin_ms1=0,pin_ms2=0,pin_ms3=0,pin_sleep=0,pin_en
able=0,pin_reset=0,name="Stepper"):
      self.pin_step = pin_step
      self.delay = delay / 2
      self.pin_direction = pin_direction
      self.pin_microstep_1 = pin_ms1
      self.pin_microstep_2 = pin_ms2
      self.pin_microstep_3 = pin_ms3
      self.pin_sleep = pin_sleep
      self.pin_enable = pin_enable
      self.pin_reset = pin_reset
      self.name = name

      if self.pin_step > 0:
         gpio.setup(self.pin_step, gpio.OUT)
      if self.pin_direction > 0:
         gpio.setup(self.pin_direction, gpio.OUT)
         gpio.output(self.pin_direction, True)
      if self.pin_microstep_1 > 0:
         gpio.setup(self.pin_microstep_1, gpio.OUT)
         gpio.output(self.pin_microstep_1, False)
      if self.pin_microstep_2 > 0:
         gpio.setup(self.pin_microstep_2, gpio.OUT)
         gpio.output(self.pin_microstep_2, False)
      if self.pin_microstep_3 > 0:
         gpio.setup(self.pin_microstep_3, gpio.OUT)
         gpio.output(self.pin_microstep_3, False)
      if self.pin_sleep > 0:
         gpio.setup(self.pin_sleep, gpio.OUT)
         gpio.output(self.pin_sleep,True)
      if self.pin_enable > 0:
         gpio.setup(self.pin_enable, gpio.OUT)
```

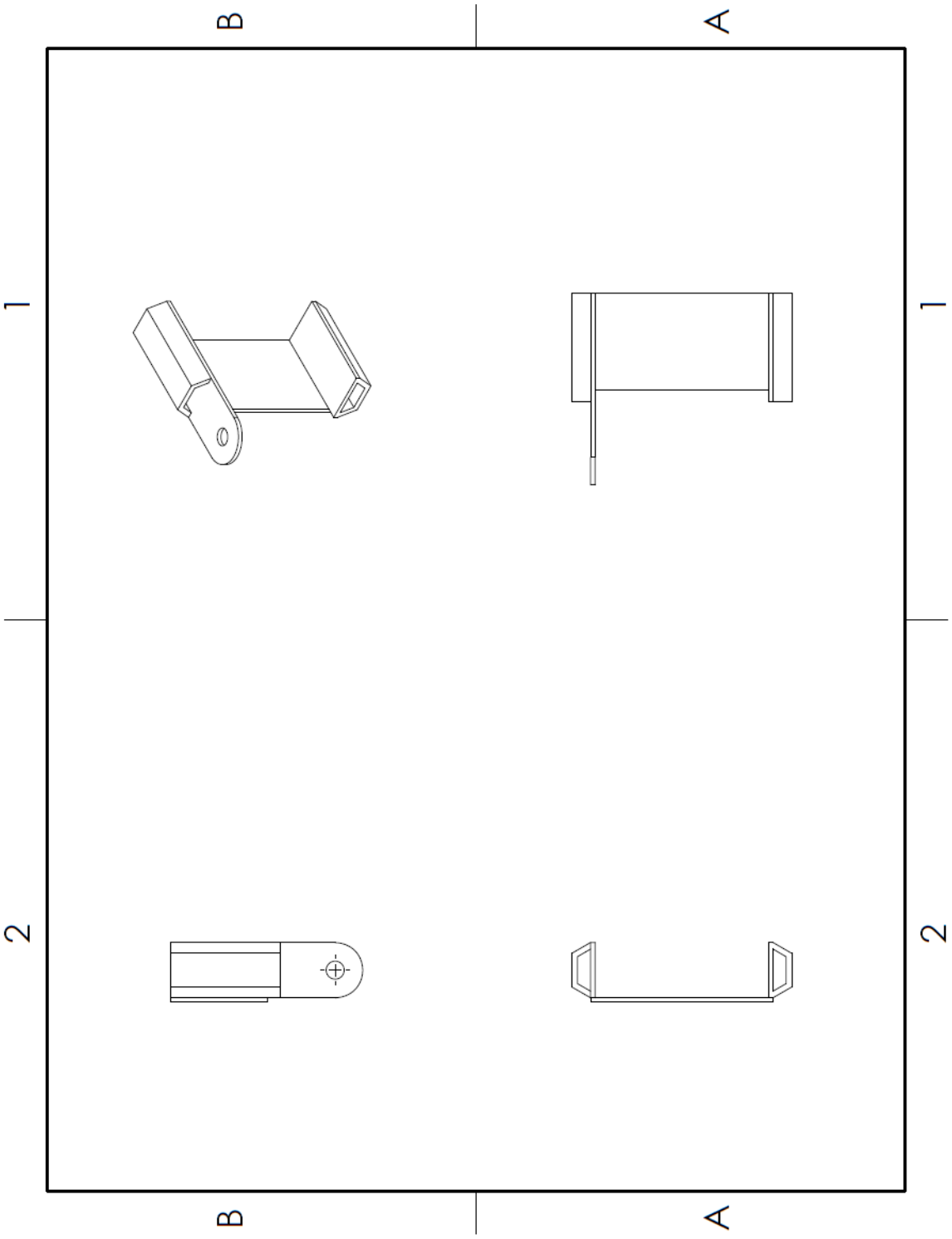
```
            gpio.output(self.pin_enable,False)
        if self.pin_reset > 0:
            gpio.setup(self.pin_reset, gpio.OUT)
            gpio.output(self.pin_reset,True)

    def step(self):
        gpio.output(self.pin_step,True)
        time.sleep(self.delay)
        gpio.output(self.pin_step,False)
        time.sleep(self.delay)

    def set_direction(self,direction):
        gpio.output(self.pin_direction,direction)

    def set_full_step(self):
        gpio.output(self.pin_microstep_1,False)
        gpio.output(self.pin_microstep_2,False)
        gpio.output(self.pin_microstep_3,False)

    def sleep(self):
        gpio.output(self.pin_sleep,False)

    def wake(self):
        gpio.output(self.pin_sleep,True)

    def disable(self):
        gpio.output(self.pin_enable,True)

    def enable(self):
        gpio.output(self.pin_enable,False)

    def reset(self):
        gpio.output(self.pin_reset,False)
        time.sleep(1)
        gpio.output(self.pin_reset,True)

    def set_delay(self, delay):
        self.delay = delay / 2

    def finish(self):
        gpio.cleanup()



def stepper_run():
    rospy.init_node('stepper_run', anonymous=True)
    rospy.Subscriber('stepQuick', StepCtrl, callback)
    rospy.spin()


def callback(data):
    rospy.loginfo(data)
    global StepNum
    global Dir0
    global Vel0
    global sub
```

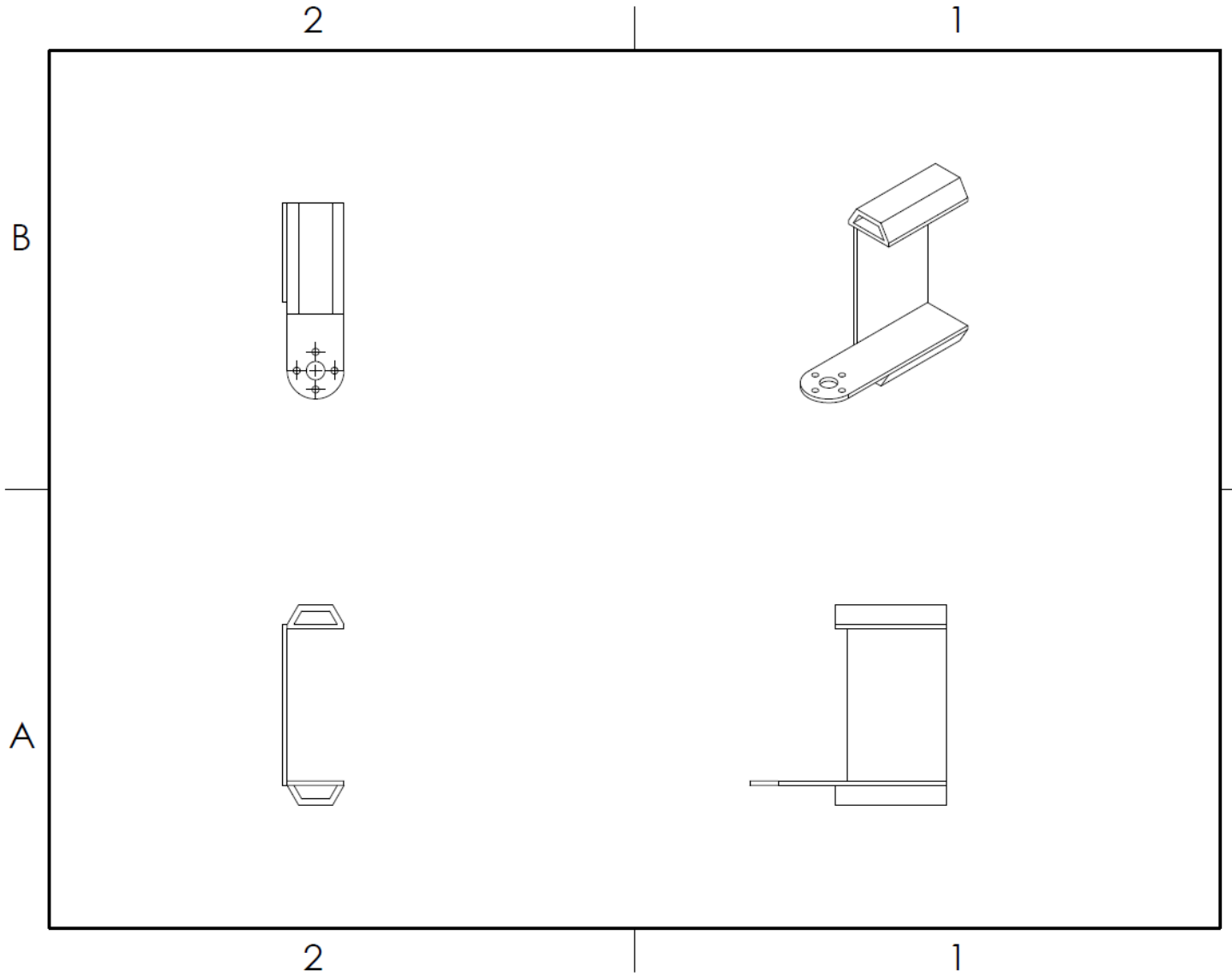
```
StepNum = data.stepNum
    Dir0 = data.dir0
    Vel0 = data.vel0

    print(Dir0)
    print(Vel0)
    print(StepNum)

    stepper = easydriver(14, Vel0, 15, 23, 24, 24)
    stepper.set_direction(Dir0)
    stepper.set_full_step()

    for k in range(0,StepNum):
        stepper.step()
        print(k)


if __name__ == '__main__':
    #gpio.setmode(gpio.BCM)
    #gpio.setwarnings(False)
    print('start')
    stepper_run()
    stepper = easydriver(14, Vel0, 15, 23, 24, 24)
    stepper.finish()
```

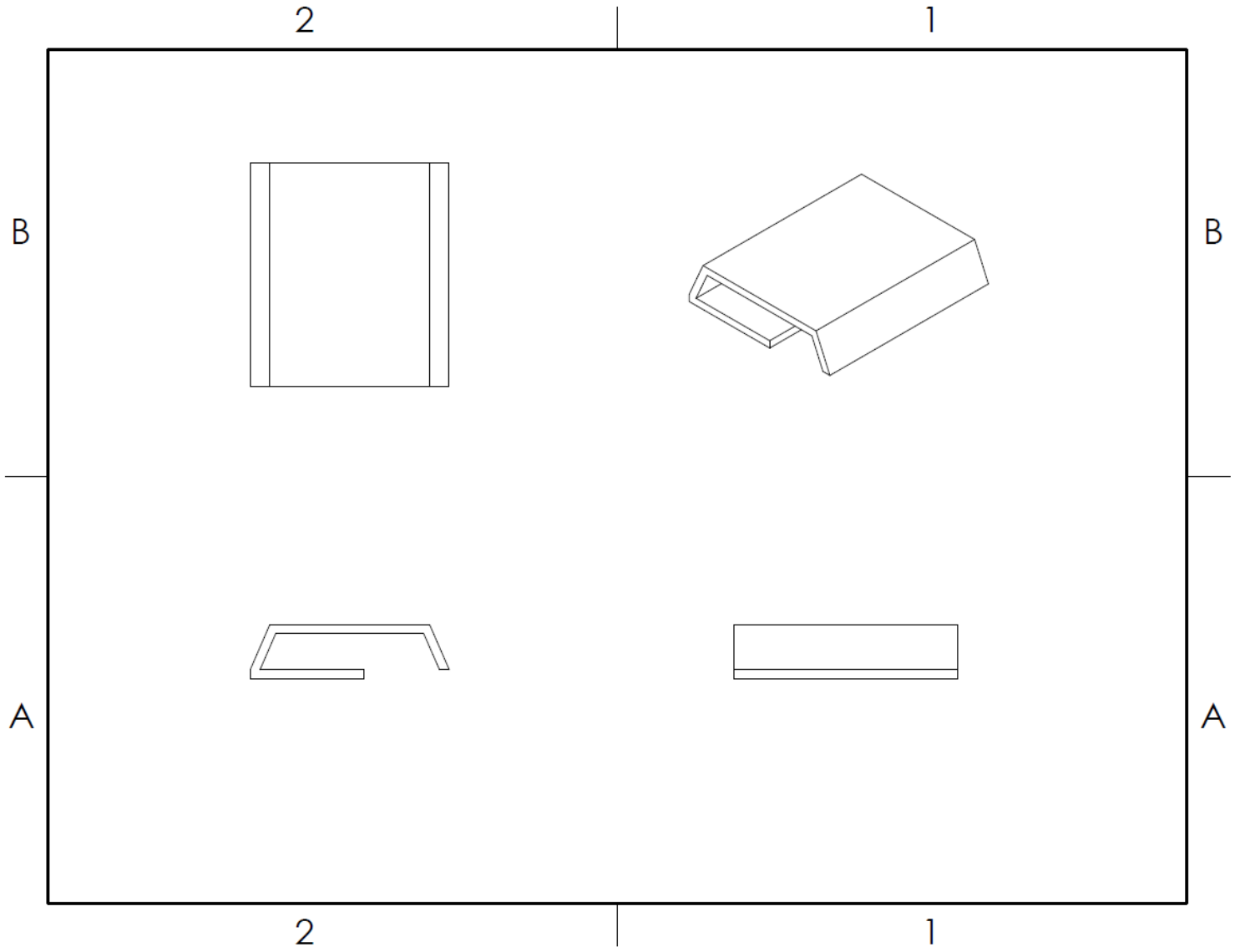
APPENDIX D

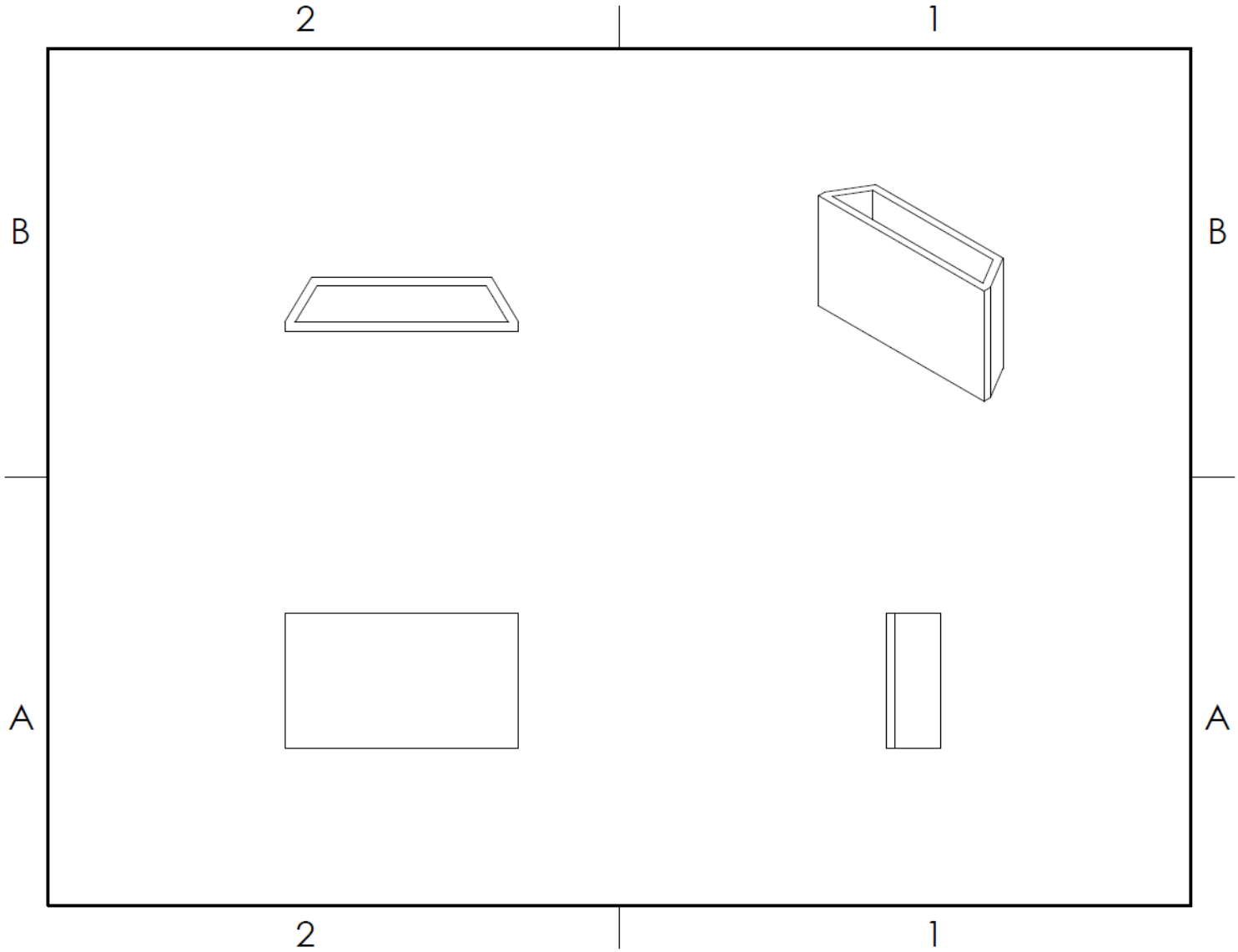
3D PRINTED COMPONENTS

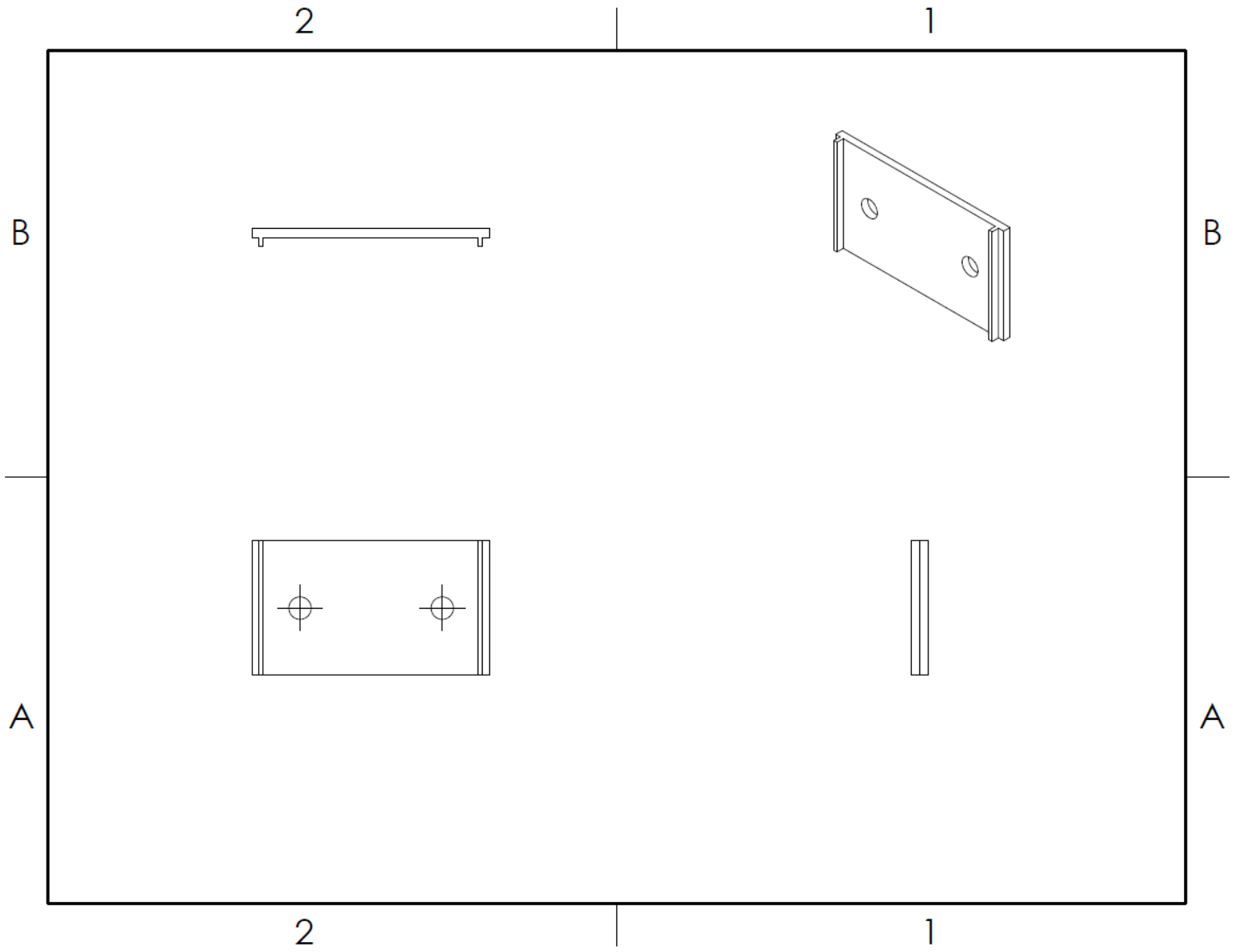

B
A
1
1
2
2
B
A


B
A
1
1
2
2
B
A


2
1
B
B
A
A
2
1


2
1
B
B
A
A
2
1


2
1
B
B
A
A
2
1


2
1
B
B
A
A
2
1

APPENDIX E

POSITION TEACHING PROGRAM

Teach Position C Script

```
#include <Servo.h>

Servo joint0;
Servo joint1;
Servo joint2;
Servo joint3;
Servo joint4;
Servo joint5;

int pos0 = 90;
int pos1 = 90;
int pos2 = 90;
int pos3 = 90;
int pos4 = 90;
int pos5 = 90;

int sensorPin0 = A0;
int sensorPin1 = A1;
int sensorPin2 = A2;
int sensorPin3 = A3;
int sensorPin4 = A4;
int sensorPin5 = A5;

int sensorVal0 = 0;
int sensorVal1 = 0;
int sensorVal2 = 0;
int sensorVal3 = 0;
int sensorVal4 = 0;
int sensorVal5 = 0;

int angle0 = 90;
int angle1 = 90;
int angle2 = 90;
int angle3 = 90;
int angle4 = 90;
int angle5 = 90;

const int buttonPin = 2;
int buttonState = 0;
int i = 0;

void setup() {
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);

}

void loop() {
  buttonState = digitalRead(buttonPin);
  Serial.println(buttonState);

  if(buttonState == HIGH){
    joint0.attach(0);
    joint1.attach(0);
    joint2.attach(0);
```

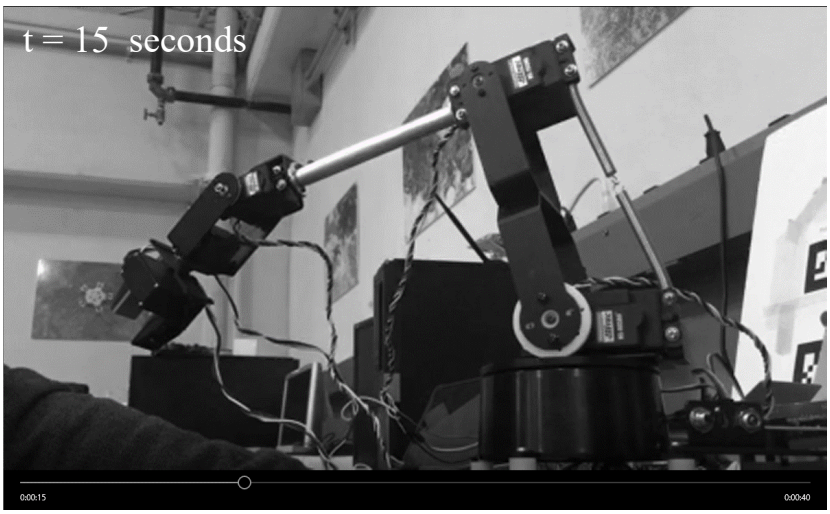
```
  joint3.attach(0);
  joint4.attach(0);
  joint5.attach(0);

  sensorVal0 = map(analogRead(sensorPin0),86,418,0,180);
  sensorVal1 = map(analogRead(sensorPin0),86,418,0,180);
  sensorVal2 = map(analogRead(sensorPin0),86,418,0,180);
  sensorVal3 = map(analogRead(sensorPin0),86,418,0,180);
  sensorVal4 = map(analogRead(sensorPin0),86,418,0,180);
  sensorVal5 = map(analogRead(sensorPin0),86,418,0,180);
  Serial.println(sensorVal0);

  angle0 = sensorVal0;
  angle1 = sensorVal1;
  angle2 = sensorVal2;
  angle3 = sensorVal3;
  angle4 = sensorVal4;
  angle5 = sensorVal5;
  Serial.println(angle0);

  if(buttonState == HIGH){
    delay(100);
  }
  i = 1;
 }else if(i==1){
  Serial.println(angle0);
  joint0.attach(3);
  joint1.attach(5);
  joint2.attach(6);
  joint3.attach(9);
  joint4.attach(10);
  joint5.attach(11);

  joint0.write(90);
  joint1.write(90);
  joint2.write(90);
  joint3.write(90);
  joint4.write(90);
  joint5.write(90);

  delay(5000);

  joint0.write(angle0);
  joint1.write(angle1);
  joint2.write(angle2);
  joint3.write(angle3);
  joint4.write(angle4);
  joint5.write(angle5);

  i = 0;

 } else{
  delay(100);
 }
}
```

APPENDIX F

POSITION TEACHING VISUALS


t = 8 seconds
0:00:08
0:00:47


t = 11 seconds
0:00:11
0:00:44


t = 15 seconds
0:00:15
0:00:40


t = 46 seconds
0:00:46
0:00:09


t = 50 seconds
0:00:50
0:00:05


t = 53 seconds
0:00:53
0:00:02

REFERENCES

[1] D. H. Autor, "Why Are There Still So Many Jobs? The History and Future of Workplace Automation," *Journal of Economic Perspectives,* vol. 29, no. 3, pp. 3-30, 2015.

[2] V. Kuts, M. Sarkans, T. Otto and T. Tahemaa, "Collaborative Work Between Humans and Industrial Robots in Manufacturing by Advanced Safety Monitoring Systems," in *Proceedings of the 28th DAAAM International Symposium*, Vienna, Austria, 2017.

[3] J. Kaplan, "on AI's place in the history of automation; the stanford professor and author talks about how AI is like previous waves of automation.," *Wall Street Journal (Online),* 20 November 2017.

[4] L. Pagliarini and H. H. Lund, "The Future of Robotics Technology," Centre for Playware, Technical University of Denmark, Lyngby, Denmark, 2017.

[5] P. Kazanzides, J. Zuhars, B. Mittelstadt and R. Taylor, "Force Sensing and Control For a Surgical Robot," in *IEEE International Conference on Robotics and Automation*, 1992.

[6] C. Schou and O. Madsen, "A Plug and Produce Framework for Industrial Collaborative Robots," *International Journal for Advanced Robotics Systems,* 2017.

[7] A. A. Malik and A. Bilberg, "Framework to Implement Collaborative Robots in Manual Assembly: A Lean Automation Approach," in *Proceedings of the 28th DAAAM International Symposium*, Vienna, Austria, 2017.

[8] J. J. Craig, Introduction to Robotics: Mechanics and Control, Pearson Education International, 2005.

[9] M. Piatek, "Identification of the Servo Motors Used in the Walking Robot," *Automatyka,* pp. 101-111, 2010.

[10] B. Beauregard, *Arduino Code [Computer Software],* https://github.com/br3ttb/Arduino-PID-Library/, 2017.

BIOGRAPHICAL INFORMATION

Cristian Almendariz began his academic career at the University of Texas at Arlington in Fall 2013 with the pursuit of his Bachelors of Science in Mechanical Engineering. Shortly after admittance into the program, Cristian joined the Honors College where he took part on the Honors College Council, holding the position of Treasurer and Vice President.

Cristian Almendariz began his undergraduate research career at University of Texas at Arlington's Research Institute (UTARI) in the Fall of 2016. As a paid researcher with UTARI, he performed work in the Biological Microsystems Lab under the supervision of Dr. Vinay Abhyankar. His work with the Biological Microsystems team was focused on developing a 3D-printed, high flowrate, sub-microliter dead volume, microfluidic degassing unit. The work on the degassing unit was presented at the 2017 BMES Conference in Phoenix, Arizona. Concurrent to working at UTARI, Cristian also performed research under Dr. Panos Shiakolas of the Manufacturing, Medical, and Robotics Systems (MARS) Lab in the Department of Mechanical and Aerospace Engineering. While under Dr. Shiakolas, his work included contributing to the development of an EEG controlled robotic hand.

Cristian Almendariz also served as an Undergraduate Teaching Assistant (TA) for ENGR 1300, under Dr. David Ewing. He was a TA for the inaugural ENGR 1300 class of Fall 2015 and continued working with Dr. Ewing through the Fall of 2017.