Low-Cost, Intention-Detecting Robot to Assist the Movement of an Impaired Upper Limb

By

Eric Young

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Mechanical Engineering

December, 2015

Nashville, Tennessee

Approved:

Nilanjan Sarkar, Ph.D.

Zachary E. Warren, Ph.D.

Thomas J. Withrow, Ph.D.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Dr. Nilanjan Sarkar, for graciously introducing me to the world of research and consistently making me feel welcome and capable. From passing on his love for dynamics, to offering me opportunities to explore diverse topics such as gesture recognition, programming, and robotics, to giving me a glimpse of teaching via grading and holding tutorials, Dr. Sarkar helped make my undergraduate career as rewarding as could be. His natural mix of guidance and lenience allowed me to make my projects my own, and develop a greater appreciation for research than I ever expected. I would also like to thank the members of my thesis committee. Dr. Zachary Warren for teaching me, by example, that research can truly make the world a better place, and Dr. Thomas Withrow for all of the generous insight provided via early-morning emails, and giving me my first introduction to hands-on mechanical engineering and continuing to teach me ever since.

I would also like to thank all of the lab members from the Robotics and Autonomous Systems Laboratory (RASL) for their assistance and never-ceasing friendliness. Working with this wonderful group never felt like work, and I can only hope to find myself surrounded by people as hospitable in the future. I would particularly like to thank Jenny Zheng for kindly guiding me through my first months in the lab and for providing constant insight since, and Josh Wade for consistently being willing to volunteer his day to help a fellow lab member, whether it be with a masters project or a just-for-fun online card game. I am additionally grateful for Vanderbilt's mechanical engineering department as a whole. Countless students took time to assist in machining, laser-cutting, and 3D-printing with nothing but a "thank you" in return, and countless professors made clear their willingness to put the students first. I would like to especially thank Phil Davis for taking the time to teach an inexperienced machinist the ropes.

Finally, I would like to offer my greatest gratitude to my friends and family. To my friends, for providing me with a perfect mix of all things nerdy and much needed breaks from all things nerdy. To Mom, Dad, Aaron and Evan, for your constant love and support, and being the coolest family a guy could ask for.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Background

Despite advances in methods of prevention and rehabilitation associated with disability-causing conditions, a large portion of the world's population continues to live with some degree of upper limb physical impairment. Stroke is the leading cause of such long-term disability in developed and developing countries, and the prevalence of stroke is increasing due to the aging of worldwide populations [1,2]. Approximately 795,000 persons in the United States suffer a new or recurrent stroke each year, and the combined direct and indirect costs of stroke exceed $34 billion annually [3]. With stroke having such a clear negative impact on individuals and society as a whole, many studies over the past few decades have been dedicated to determining better methods of alleviating the social, economic and physical burdens of stroke.

Recently, increased attention has been given to stroke rehabilitation, particularly robot-assisted rehabilitation, in hopes of easing the difficulties associated with post-stroke life. Specific examples of robotic systems will be given in the following section, but in general robot-assisted rehabilitation shows enormous promise in the field of rehabilitation due to its potential cost-effectiveness, adaptability, and mobility. Yet despite improving technology and continued development of effective robotic rehabilitative devices, vast numbers of stroke survivors continue to live with physical impairments. Among those who recover from stroke, only 10% recover completely, and many of the remaining survivors need rehabilitation due to continued impairments [4]. Approximately 50% of stroke survivors experience chronic hemiparesis (or weakness of one side of the body) and 26% become dependent in activities of daily living

(ADLs) [5]. These numbers depict a global population still in need of alternative forms of stroke relief.

Life-long physical impairment is not limited to stroke patients. Spinal cord injuries are another common cause of long-term disability. Approximately two million people worldwide live with a spinal-cord injury (SPI), nearly 250,000 of whom live in the United States [6]. Among them, 36.7% sustain paraplegia (impairment of lower limbs) and 52.2% sustain tetraplegia (impairment of all limbs and torso) [7]. Due, in part, to the physical barriers to basic mobility experienced by those with SPIs, the global unemployment rate of adults with spinal cord injury is over 60% [8]. Research studies show a rehabilitative environment that is continuously evolving and improving, but the numbers show a large population still experiencing a significantly reduced standard of living due to some form of physical impairment.

While pursuing better rehabilitation methods is a crucial endeavor, it is also important to acknowledge that many people need an alternative form of assistance for physical impairments, both while undergoing rehabilitation and in the unfortunate but common scenario of rehabilitation providing insufficient improvements. The aim of this thesis is to present a low-cost robotic assistive device which may serve as a complement to rehabilitation procedures. The proposed system determines the intended movement of a user's upper arm and assists said movement. In this manner, the system may provide immediate relief for someone suffering from physical impairments in their upper limbs, either as a complement to ongoing rehabilitation therapy or as a partial solution in the case of insufficient improvements from rehabilitation.

1.2. Robotic Assistive Devices

As robotic technology has advanced, robotic systems have been sought out to assist in nearly all aspects of human life. Many studies have shown the potential of robotic systems to effectively complement conventional rehabilitation therapy [9,10,11,12,13], encouraging rapid growth in robotic rehabilitation in recent years. The projects discussed here all aim to assist the rehabilitation of those with impaired movements of upper limbs. While non-rehabilitative robotic assistance (the purpose of this thesis) and robotic rehabilitation have different objectives, both share a common necessity to assist the movement of an impaired limb to a specific location.

1.2.1. Robot-aided Rehabilitation

The adaption of robotic systems to rehabilitative environments has evolved rather quickly in recent years. In the early 1990s, Hogan et al. developed the MIT-MANUS, a two degrees-of-freedom (DOF) robot designed to move a patient's upper limb within a two-dimensional plane [14]. This system consisted of a two-link serial robot responsible for the unrestricted movement in the horizontal plane, and a splint attached to both the robot and the patient, which allowed the motion of the robot's end effector to be translated to the user's forearm. When used by patients with post-stroke hemiparesis, the MIT-MANUS was shown to produce benefits in the recovery stage which were sustained over three years after hospital discharge [15,16]. The MIT-MANUS helped lay a foundation for rehabilitative robotics, and has since become one of the most well-known and widely used robot for arm rehabilitation. The ARM (Assisted Rehabilitation and Measurement) Guide, developed by Reinkensmeyer et al, was soon developed to allow for motion in a three-dimensional space, although three of the four degrees-of-freedom were manually controlled [17]. The device primarily assisted "reaching" motions, and included a

motor-driven linear track and manually adjustable yaw, pitch and height. Lum et al. presented a rehabilitation robot with an even greater workspace in the Mirror-Image Motion Enabler (MIME) [18]. MIME, which incorporated a Puma robot, used the motion of a patient's unaffected arm to assist the motion of the affected arm. The system included 6 degree-of-freedom load cells and was capable of 6 degree-of-freedom motion, meaning both the orientation and position of the user's forearm could be controlled. Once rehabilitation robots were able to manipulate all 6 degrees-of-freedom of a user's forearm, many systems were developed in hopes of finding more powerful, cost-effective and space-effective designs. The WREX family of robots, including WREX, T-WREX and Pneu-WREX, all continued the serial link based robotic design, with each progression providing a slightly new take on design features [19]. T-WREX focused on the use of counterbalancing to reduce the force needed by the robot, and thus the cost and size [20]. Pneu-WREX added pneumatic actuators to the passive counterbalancing design, which allowed significant forces to be applied to the patient [21].

To further reduce the cost and size of rehabilitation robotics, wire-based designs began to emerge. Cable-driven systems used relatively light-weight cables instead of rigid links to transmit desired motion to a user's limb, and became well known for their low mass and large workspace [22,23,24]. Various wire-based robots were developed, including an upper limb motion robot by Takahashi et al [25], a 7 degree-of-freedom virtual interface named SPIDAR-G [26], and GENTLE/s, which included both cable-based actuation and rigid robotic links [27]. The NeReBot, later adapted to the MariBot, consisted of a rigid-link robotic system which moved in a horizontal plane above the user and acted as a foundation from which cables could be dropped and driven to move a splint located below [28]. Initial trials of the NeReBot yielded promising results, with patients demonstrating significant improvement after 25 sessions with

robot-led training compared to patients in the control group [29]. The group is continuing to develop the system with a progression named MariBot. Only two of MariBot's five degrees-of-freedom are in the rigid robotic arm, allowing the arm to be low-inertia and relatively low-cost [30]. The use of cables also produces a more visually clean workspace, which is important in promoting the use of rehabilitation robotics outside research laboratories. In the past decade, systems based entirely on cables have been developed, allowing for extremely low inertias and large workspaces. The Multi-Axis Cartesian-based Arm Rehabilitation Machine (MACARM) features an array of eight motors mounted at the corners of a stationary cubic workspace, each of which drives a cable attached to a centrally located end-effector [31]. The MACARM eliminated all rigid body motion, except that of the central end-effector, significantly reducing the inertia of the system.

Researchers have also begun to combine the designs of serial robotic devices and cable-driven robotic devices to form exoskeleton devices. These devices typically attach each segment of the user's limb to a similarly structured robotic segment, which allows the device to more accurately control each individual component of the user's arm movement. Early exoskeleton robotic devices include Armin [32], BONES [33], Rupert [34], and Dampace [35]. Later exoskeletons such as CADEN-7 [36] and MEDARM [37] attempted to reduce the moment of inertia by using wires to transmit torque from stationary motors to the joints of the exoskeleton. Yang et al. combined the ideas behind serial link based and cable based robotics with the introduction of a seven degrees-of-freedom cable-driven exoskeleton for the arm [38]. This design was later adapted to a four degrees-of-freedom cable-driven exoskeleton with a more robust control architecture at the University of Delaware [39], and eventually to CAREX [40]. Such a design maintained the desirable low inertia of cable-driven systems by allowing the

motors to be mounted away from movement, while avoiding the large amounts of free space needed for non-exoskeleton cable-driven devices such as the MACARM.

Clearly, progress in the field of robotic rehabilitation for upper limbs has been swift and is likely to continue its rapid evolution. However, none of the systems described attempt to bridge the gap between robotic rehabilitation and robotic assistance in the outside world. The solutions to rehabilitation and assistance have many overlaps, such as control methods, forms of human-robot interface, a preference for low inertia, etc. However, each of the designs presented so far has limitations in its ability to be adapted to real-world assistive robotic systems. To begin, for an assistive robot to reach its full potential, it must assist the user in the widest possible range of motion while occupying the smallest workspace. Many of the rehabilitative robots described are used in research settings, where obstacles can be displaced or removed. The ultimate goal of assistive robotics is to help the user in whatever setting the user needs assistance, which often may not be as controllable as a laboratory setting. Serial link based robots, such as MIT-MANUS, ARM Guide, and MIME are all fairly bulky, which would be undesirable in a work or home setting. Cable-driven robots such as MACARM require cables to extend from the end-effector in all directions, which would render a large fraction of any environment unusable. CAREX appears to be a reasonable solution, but still requires substantial setup and would be difficult to quickly attach and detach if so desired. In addition to workspace limitations, current rehabilitative robotic systems have no method of recording human intention. For rehabilitation, the robot is conventionally driven along a pre-planned repetitive motion. Many systems include user input, such as force sensors and control effort detection, but all use such input to alter desired motion, not to define it.

1.3. Human Intention Detection

An effective assistive robotic device requires the addition of human intention detection. As robots have become more commonplace in industry and academia, greater emphasis has been placed on finding seamless methods of communication between humans and robots. Many robots have been designed to monitor a human for a wide array of cues, from which human intention can then be interpreted. As described by Kulić and Croft, two categories currently exist for robot-human monitoring systems. One category is mechanical-based, where the system measures forces and displacements during physical contact between user and robot, and the other is communication-based. Communication-based systems can be further divided into monitoring visual cues, such as eye-gaze, head position, facial expression or gestures, and monitoring physiological cues, such as heart rate, skin conductance and brain activity [41].

One example of mechanical-based monitoring is the Extender technology developed by Kazerooni et al. This system senses and amplifies the force applied by a human in order to assist in the manipulation of a heavy object [43-46]. Other systems require a joint effort between human and robot by having a user guide an object that is carried by a robot, thus reducing its inertia [47,48]. In an attempt to reduce the need for force sensors, some researchers have explored techniques to detect input forces by observing changes in the actuation effort rather than using force sensors directly [49-51]. A system developed by Erden and Tomiyama expands upon this technique. The control scheme of this system estimates the intention of the human by observing the change in control effort [42]. Thus the user applies a force to the robot, the robot recognizes the force and calculates the user's intended motion, and then the robot assists the user in said motion. While these systems have given promising results, they both rely on the ability of the user to give an initial input. In the realm of assistance for mobility impairments, the user

may not always be capable of providing the initial force input. Thus, while mechanical monitoring may be beneficial for giving the human increased control, alternative monitoring mechanisms are also needed.

The field of communication-based monitoring is much larger. One proposed method is the monitoring of physiological signals. While physiological signals tend to be more difficult to interpret, studies have shown this can be overcome with careful classification and evaluation. Sarkar has proposed using multiple physiological signals to examine the user's emotional state, which can then be used to modify the robot to make the user more comfortable [52]. Picard et al. uses physiological signals as well, although to interact with a computer-interface rather than a robotic system [53]. Other projects draw inspiration from human-to-human communication. Communication-based monitoring is important during interperson interaction, where non-verbal cues such as eye-gaze direction, facial expression and gestures are all used as modes of communication [54]. Nehaniv et al. classified types of gestures into 5 main classes as a primer for inferring intent from gestures in human-robot interaction [55]. Researchers have proposed using gesture recognition to record and repeat the virtual drawing of letters [58], guide a wheeled robot to specific locations [59,60], and even control and program a robotic vacuum [61]. Ho et al. used two integrated sub-processes to isolate human intended actions from ordinary walking behavior, which could then be used as control inputs for a robotic system [56]. Song et al. took a slightly different approach, and proposed a controller for a wheelchair-based robotic arm based on the user's mouth gestures [57]. The use of speech recognition has been implemented in a wide array of systems, from everyday cellphone use to setting the affective state of a robot [62].

In terms of driving an assistive robot to a desired location, the goal is to allow the user to set the location without physical movement via communication monitoring. Although gesture

recognition and physiological data may prove beneficial to such systems, eye-gaze can be used to allow the user to set the location in a much more natural manner. Eye-gaze is particularly attractive, as it has potential to be observable without direct contact and is not physically demanding.

1.3.1. Eye-Gaze Implementation

Even within the field of eye-gaze tracking there are many different approaches to gathering the most accurate data in the most seamless manner. Kaufman et al. describes the electro-oculogram (EOG) method, where electrodes are placed around the eye and small differences in skin potential are measured [63]. These small differences correspond to the eye position, allowing eye-gaze to be calculated. Chen and Newman discuss the design and implementation of an electrooculography-based gaze-controlled robotic system. The group concluded that using electrooculography to determine user intention has advantages such as high accuracy, but requires physical contact (for the electrodes), and a relatively significant setup [64].

In their review of eye-tracking technology, Morimoto and Mimica discusses the appeal of remote eye gaze trackers (REGTs), noting REGTs offer comfort of use and an easier and faster setup [65]. However, only recently has remote eye-tracking technology been accurate enough and cheap enough to incorporate into low-cost robotic systems. In 2000, Schnipke and Todd found that existing commercial REGTs eye-tracking were not ready to be employed in usability laboratories. The pair found that even an eye-tracker operator with one year of experience was only able to make an REGT successfully track 6 out of 16 participants [66]. Since then, studies have steadily improved upon existing techniques, making eye-trackers more forgiving of

different head poses and thus more accurate. Nguyen et al. explored the infrared bright pupil response of human eyes to help give possible explanations for variation between subjects [67]. Variations of infrared eye-tracking include placing IR light emitting diodes and IR photo-transistors above and below the eye [68], generating bright and dark pupil images by using IR sources with different wavelengths [69] or using IR sources in different locations [70]. Morimoto et al. proposed a method of using a single camera and at least two near infra-red light sources to first detect the position of the human's eyes, then use the position to better estimate the eye-gaze [71]. Yoo et al. proposed using five infra-red lights and a single camera to compute the user's eye-gaze without a need for computing the geometrical relationship between the user, computer and camera [72]. Due to simplicity and reasonable accuracy, many REGTs today are based on the infra-red (or near infra-red) corneal reflection technique [73-78].

Remote eye-tracking has grown tremendously in recent years and now even commercially available products, such as the Tobii Eye X which will be discussed later, give high accuracy at a low cost. Equipping an assistive robotic device with a remote eye-tracking has potential to effectively give the system a means of monitoring a human's eye-gaze and thus detecting human intention.

1.4. Project Overview

The system proposed in this thesis combines desirable elements from rehabilitative robotics and human intention detection via eye-tracking. From the beginning, the device was hoped to be used in common everyday settings, such as a workplace or home. For this initial version, the designated target environment was a desktop, similar to what one would find in an office. Thus the mechanical design had to occupy a small workspace, attach to and detach from

the user effortlessly, and use low-cost methods wherever possible. Of the rehabilitative robots presented before, this most closely matches the NeReBot and MariBot, which house all rigid robotic components in a region of space otherwise unused by the human.

The proposed system can be subdivided into a subsystem responsible for linear motion and a subsystem responsible for rotational motion. The actuators for both subsystems are housed at one end of the device, creating a low moment of inertia and allowing the system to drive the end-effector to all parts of a large workspace while requiring little space to be left unoccupied for the device itself to move within. Currently the linear subsystem rests on the tabletop, which would ultimately render much of the desktop useless. Ideally, this prototype will be adapted to attach to a support above the user and assist the human's movements from above, allowing the desk fully cluttered without hindering the use of the robotic assistive device. The device collects input from the human via force sensors in the wrist-interface and a Tobii Eye X eye-tracker which follows the user's gaze. The control scheme gives priority to force inputs, as the human should be allowed to move to any desired location if capable, and then accepts eye-gaze data as a secondary input should the user not be able to provide large enough forces to drive the system.

Figure 1-1: Conceptual Model of Entire System

Along with providing the desired assistive motion, a secondary goal of this project was to explore the use of low-cost technology, so to make the system as impactful as possible. One way this was done was by using a passive spring to induce linear motion in one direction. This will be further explained in Chapter 2, but the use of a passive element to create motion reduces the inertia of the system as well as its cost. In addition, common, low-cost electronics were used wherever applicable. The wrist-interface acting as the end effector of the system houses four force-sensing resistors (FSRs), which are capable of collecting forces in all directions at a fraction of the cost of traditional load cells. Potentiometers were used to measure the position of the end effector with sufficient accuracy, eliminating the need for expensive encoders. Arduino was used for acquiring data from the FSRs and potentiometers, and for sending control signals to the motor controller. As mentioned before, eye-tracking technology has advanced significantly in the past decade, and current commercial remote eye-gaze trackers (REGTs) are both cost-effective and sufficiently accurate, as will be shown in Chapter 6 under eye-tracker validation. The system combines low-cost with higher-quality components (such as the motor controller and rotational motor/gearbox combination) in a manner that allows the user to seamlessly interact with all of them.

Of course, the robotic assistive device presented could be used for rehabilitation if so desired, similar to many of the robots presented before. However, it can also be used as an immediate helping hand for the physically impaired due to the addition of human intention detection via an eye-tracker. Preliminary testing shows the device is able to accurately detect a desired location input from the user, and drive the end effector to the designated position with or without a load.

1.5. Thesis Outline

This thesis aims to describe the design, control scheme, and preliminary performance evaluations for a low-cost, human intention detecting assistive robot for those with upper-limb physical impairments. The subsequent chapters are organized as follows. Chapter 2 describes the mechanical design and construction of the entire system, from the linear one degree-of-freedom subsystem, to the rotational subsystem, to the wrist interface. Chapter 3 discusses the electronic and hardware architecture of the system. Chapter 4 presents the software architecture and the methods of communication between Arduino, MATLAB, and a C# application. Chapter 5 gives a detailed explanation of the code present in all portions of the software, thus showing the overall control scheme for the system. Chapter 6 presents preliminary experimental results validating the potential of the system. Chapter 7 concludes this thesis.

CHAPTER 2

MECHANICAL DESIGN

This chapter discusses the mechanical design and construction of the system. The linear and rotational portions of the design will be discussed separately in this section, as the mechanical design of each is independent of the other. An overview of the entire system will be presented, followed by the initial design requirements, and then the linear and rotational portions will each be discussed.

2.1. Mechanical Design Overview

The overall design of the system consists of two mechanically independent subsystems: one controlling linear motion and one controlling rotational motion. The linear subsystem allows controllable linear motion of over 0.6 meters, which will be discussed in detail later. The entire linear subsystem rotates about a vertical axis located just outside, but collinear with, the region of allowed linear movement. In this way, the workspace is expanded to an annular wedge. Contrary to systems which consist of multiple linkages, this design allows both motors and nearly all electronics to be located in one central location, which keeps the workspace free of clutter and reduces the inertia of the system. The pictures below show the CAD model of the entire system.

Figure 2-1: Front View of Entire System

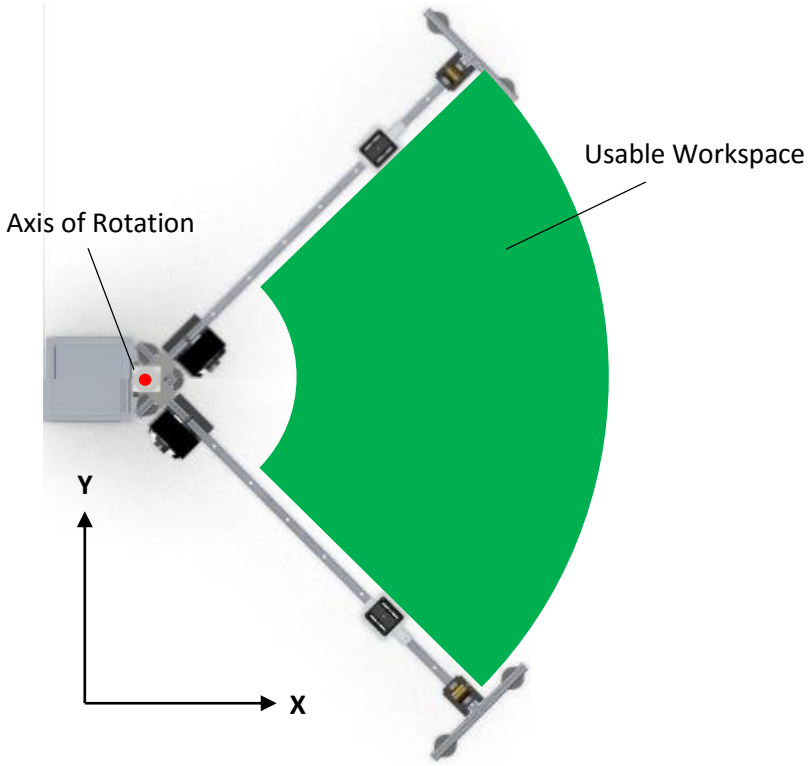


Figure 2-2: Top View of Entire System

The above picture shows a birds-eye view of the system. The linear portion of the system rotates about the fixed rotational portion of the system, which is housed in the framing seen at the left of the picture. The red dot indicates the axis about which the linear subsystem rotates, and the green annulus depicts the resulting workspace of the device.

2.2. Design Requirements

Before any design or development could begin, it was first necessary to determine the mechanical objectives of the system. Of primary concern was the desired force and speed capabilities. Since the goal of the system was to drive a participant's impaired arm to a desired location, the system needed to be able to approximate natural human movement. For a first design, the desired maximum velocity of the end-effector was set at 1 m/s. While faster natural

arm movement is possible, a speed of one meter per second allows the end-effector to transverse a desktop in a short time without the concern of moving so quickly the user becomes uncomfortable. It was also desired that the system be able to drive the end-effector to its maximum speed quickly, or within half of a second, giving the end-effector a desired maximum acceleration of 2 m/s^2. In his study of various segments of the human body, Clauser et al. show that the mass of a typical arm is under 5 kg [79]. In fact, all arms examined in the study were below this mass, showing it is a safe upper limit on arm mass. With a desired acceleration of 2 m/s^2 and an estimated mass of 5 kg, the system should be able to apply 10 N (roughly 2.5 lbs) of force in all directions. To ensure the system is able to meet and exceed the requirements presented, both the linear and rotational subsystem were designed to reach velocities of 1 m/s and accelerations of 2 m/s^2, and provide 20 N of force to the end-effector.

2.3. Linear Actuation

The linear portion of the design centers on the use of an active motor and a passive constant-force spring to control linear motion. The spring provides a constant force away from the motor, meaning a force from the motor smaller than the force of the spring will result in a net force away from the motor, while a force from the motor greater than the force of the spring will result in a new force toward the motor. In this way, a single motor is able to apply a force in either direction, while only being attached at one end. The use of a passive spring simplifies the system, reduces mass and cost, and eliminates the issues of having two motors pulling against each other.

The constant-force spring is a flat strip of metal wound into a cylinder, and changes length by rotating about its central axis. A holder for the spring was made using ¼'' acrylic and

a laser-cutter, and houses a shoulder bolt around which the spring rotates. Since the spring's inner diameter is fairly large, a low-friction UHMW polyethylene rod is used as the contact between the spring and the shoulder bolt. The outer diameter of the rod is 1'', which is slightly larger than the spring's natural inner diameter, and causes the spring to be held tightly in place with respect to the polyethylene rod. The rod was drilled to allow it to rotate about the shoulder bolt, and its slippery surface reduces the friction associated with the spring's movement.

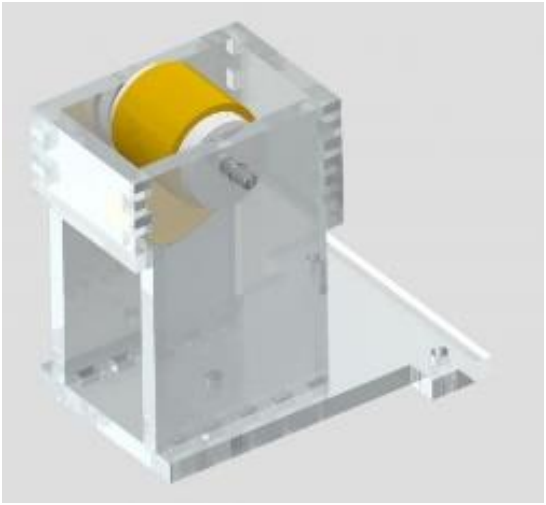


Figure 2-3: Spring-Holder Design

A low-friction carriage and rail combination is used to reduce frictional losses while maintaining sufficient rigidity. The rail runs from the spring to the motor and rests on structural framing for additional support. The carriage houses the wrist interface, which includes a 3D-printed mount for the wrist brace, a set of four force-sensing resistors (FSRs), and an XBee module for wireless data transmission. The design of the wrist interface will be discussed in section 3.5.

A potentiometer is used to determine the position of the carriage along the rail. The potentiometer is located on the rail near the motor, in order to keep wiring in a central location

and allow the greatest range of measureable movement. The picture below shows the setup of the entire linear portion of the system.

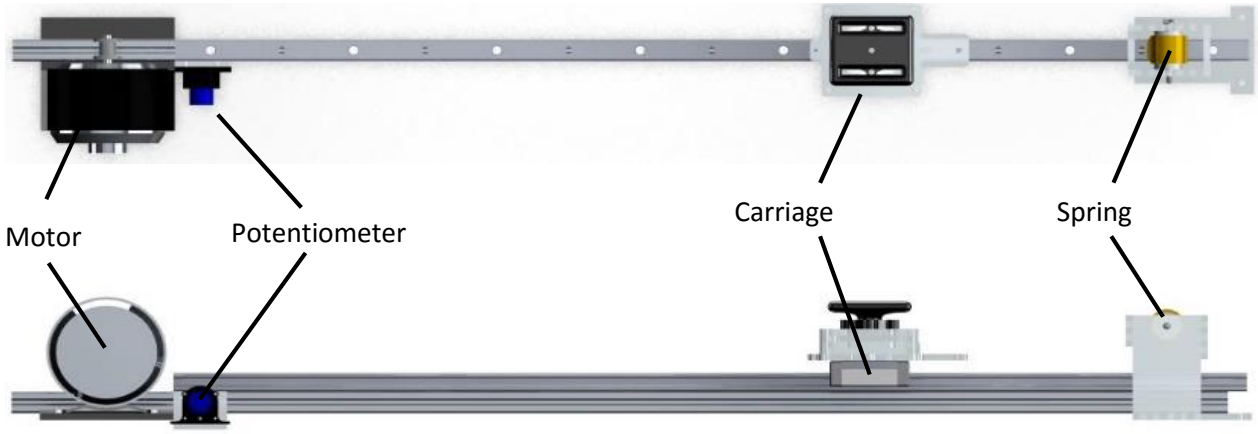


Figure 2-4: Above (Top) and Front (Below) Views of Linear Subsystem

2.3.1. Motor Selection

The primary concern for motor selection was meeting the design requirements laid out earlier in the chapter. The linear motor needed to be able to apply at least 20 N (~5 lbs) of force to the end-effector and drive the carriage at up to 1 m/s. Since the system was to be built at a low cost, the goal was to find a DC motor which met these requirements, as DC motors and their controllers are typically less expensive than their AC counterparts. The motor found to perform the actuation for the linear subsystem is a MY1016 brushed DC motor from Unite Motor Co. This 24V 250W motor has a rated torque of 0.9 N-m and a rated speed of 2750 rpm. A 3D-printed spool with a diameter of 14.2 mm was attached to the shaft of the motor, which allows the motor to provide over 50 N (~11 lbs) of linear force and drive the end-effector at up to 2 meters per second. Even if driven with a 12 V battery, as ended up being the case, the maximum velocity is just over 1 meter per second.

2.3.2. Spring Selection

The spring chosen for the system also had to meet the requirements laid out earlier in the chapter. Since the spring would be a passive element, the speed requirement does not apply, but the spring still needed to be able to apply 20 N at all times and all locations. In order to ensure this force existed at all locations along the rail, a constant-force spring was used, which applies the same force regardless of position, unlike a traditional spring. A constant-force spring rated to provide 5 lbs (~22.24 N) of force at all lengths was selected. Once the spring was ordered, its characteristics were tested to verify its ability to apply a constant force. Due to friction in the spring-holding device, the spring would remain stationary at a range of lengths for any applied force. For the sake of evaluation, a relationship was found between the applied force and the average spring length at which the spring would remain stationary under the applied force. The graph below shows the results.

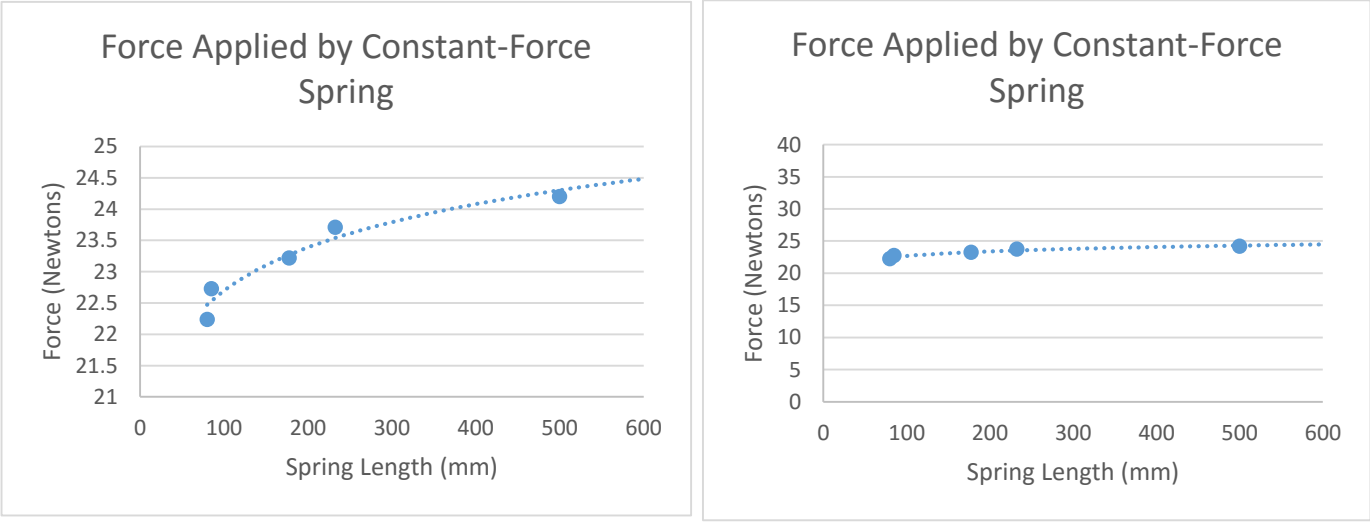


Figure 2-5: Force vs. Length of Constant Force Spring

As shown in the graph, while not perfectly constant, the constant-force spring provides a force of nearly 22 N at all lengths. The resting length of the spring was 80 mm and the spring stayed at this length until over 22 Newtons was applied. The maximum force applied by the spring was at its maximum used length of 900 mm, with a force of 25 Newtons.

2.3.3. Potentiometer Integration

The only sensor needed for the linear portion of the system was a potentiometer to determine the current position of the carriage along the rail. A kit was bought from AndyMark [84] that uses a cheap design to attach a string and a spring to potentiometer, which ensures the string will always be taught and accurately give the position of the carriage. A diagram of the potentiometer is shown below.

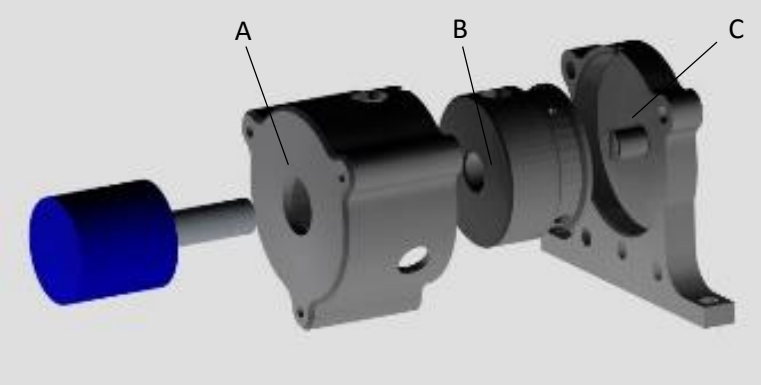


Figure 2-6: Linear Potentiometer Setup

Part B in the diagram is attached to potentiometer and free to rotate within the case created by parts A and C. The 1-meter string is wound around the labeled groove, with one end tied to Part B and the other end exiting the case and left free to connect to the carriage. One end of the spring coil is attached to Part C and the other end is attached to the rotating Part B, allowing the spring to apply a small force on the string at all times and keep the string taught.

All system dynamics were calibrated and tested with the potentiometer in place, eliminating the need to worry about the force applied by the spring on the carriage.

2.4. Rotational Actuation

The rotational portion of the system consists of a motor and gearbox combination, a potentiometer to measure angular position, and a means of transmitting torque from the motor to the linear subsystem. The motor and gearbox are aligned vertically and are attached to an aluminum plate on a fixed housing. The housing acts as a central location for electrical components, as well as a sturdy, fixed structure capable of providing the reaction forces necessary to keep the motor in place. The shaft of the gearbox is supported by a bronze bearing and transmits torque via a key to a keyed bushing which is rigidly attached to the linear subsystem. The end of the gearbox shaft is suspended above the table, allowing a potentiometer to sit beneath and collinear with the axis of rotation. A simple 3D-printed connector rests between the motor shaft and the potentiometer and keeps the two aligned. Four ball transfers were attached to the bottom of the linear subsystem to allow low-resistance rotation.

2.4.1. Motor/Gearbox Selection

Following the derivation laid out previously in the chapter, the rotational motor needed to be able to apply at least 20 N of force to the end-effector and drive the carriage at up to 1 m/s. However, the angular velocity needed to drive the end-effector at a linear velocity of 1 m/s depends on the distance between the end-effector and the axis of rotation. If the end-effector is only 10 mm from the motor shaft, the angular velocity would need to be 100 rad/s, which would result in the far end of the meter-long rail travelling at an unreasonable speed of 100 m/s. If the

end-effector's minimum distance from the motor shaft is set to be 300 mm, the angular velocity needed can be reduced to 3.33 rad/s. With regards to torque, the greatest requirements on the motor will be needed to apply 20 N to the end-effector when it is the maximum allowed distance from the axis of rotation, or 1 meter. The torque needed by the motor is 20 N-m.

The selected motor is a BaneBots FIRST CIM 12V DC motor, with a maximum torque of up to 2.4 N-m and maximum angular velocity of over 5000 rpm, or 520 rad/s. The motor is used with a 64:1 P80 planetary gearbox, which brings the maximum torque to 115 N-m (due to limits on the gearbox) and a maximum angular velocity of just over 8 rad/s. These specifications meet the presented requirements even with an efficiency as low as 50%.

2.4.2. Torque Transmission

The output shaft of the gearbox is connected to the linear sub-system via a key and keyed bushing. The picture below shows the connection.

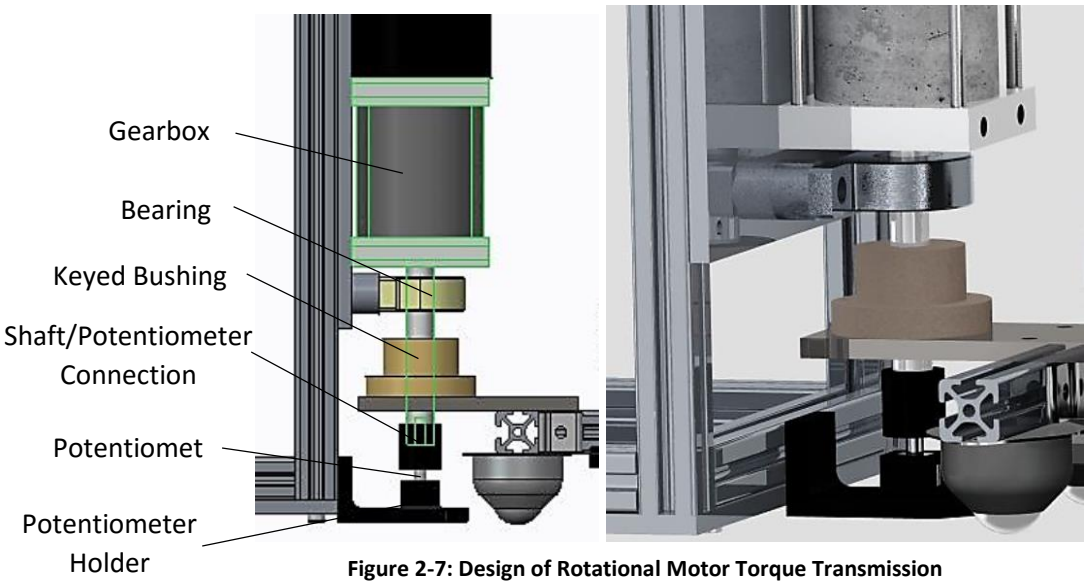


Figure 2-7: Design of Rotational Motor Torque Transmission

The gearbox shaft drives the keyed bushing via a 1/8'' steel key. The bushing is rigidly attached to a 1/4'' thick piece of aluminum, which is in turn rigidly attached to the linear subsystem. The rotational potentiometer housing is shown below the motor, as well as the 3D-printed connection between the gearbox shaft and the potentiometer.

The most likely point of failure during torque transmission is either the key or the connection between the bushing and the aluminum plate. The factor of safety calculation for the key, along with an accompanying diagram, is shown below, following the guidelines of Beardmore [80].

$$\sigma = \frac{T * K_s}{x * L * r} \quad [1]$$

where T is the applied torque, K_s is a factor explained more thoroughly by Beardmore, x is key depth, L is key length, and r is shaft radius. Using the equation above, the maximum stress is calculated to be 139 MPa. Since the strength of the key is given as 63800 psi, or approximately 440 MPa, the factor of safety for the key is over 3.

To check the safety of the connection between the bushing and the aluminum plate, both the screws and the aluminum plate must be checked. The three cap screws are located 20 mm from the bore center, with each having a diameter of 4.826 mm and a strength of over 300 MPa. The maximum expected stress is given by:

$$\sigma = \frac{22\text{N} * \text{m}}{0.02\text{m}} * \frac{1}{3 * \pi * 0.002413\text{m}^2} = 20\ \text{MPa}$$

The 6061 Aluminum plate has a strength of over 100 MPa and a thickness of 1/4''. The maximum expected stress in the aluminum is given by:

$$\sigma = \frac{22\text{N} * \text{m}}{0.02\text{m}} * \frac{1}{3 * 0.00635\text{m} * 0.004826\text{m}} = 11.9\ \text{MPa}$$

Both of these calculations yield safety factors of over 10, showing the connections suggested are viable means of transmitting the torque required by the system.

2.5. Wrist Interface Design

The interface between the user's wrist and the system is responsible both for transmitting the system's generated forces to the user and for recording the resulting forces between the user and the system. The wrist interface design was responsible for meeting the requirements associated with both of these responsibilities, including comfort, sufficient rigidity, and low force-loss during force transmission.

2.5.1. Transmitting Force

With regards to transmitting the system's generated forces to the user's wrist, the primary objective was achieving a desirable combination of rigidity and comfort. If the interface were too flexible, the force applied to the user would be reduced and additional dynamics would exist in the system in the form of the flexible interface's motion. However, if the interface were too rigid, it would likely be uncomfortable, or even painful, for the user. A happy medium was found by combining a rigid platform with a flexible wrist brace, giving a rigid location from which system dynamics can be calculated, while allowing the user to tighten or loosen the brace for a desirable level of comfort. A ball bearing with its rotation axis oriented vertically connects the rigid platform to the carriage, allowing the wrist interface to rotate freely and align with the user regardless of the orientation of his forearm. The picture on the following page shows the entire design of the wrist-interface.

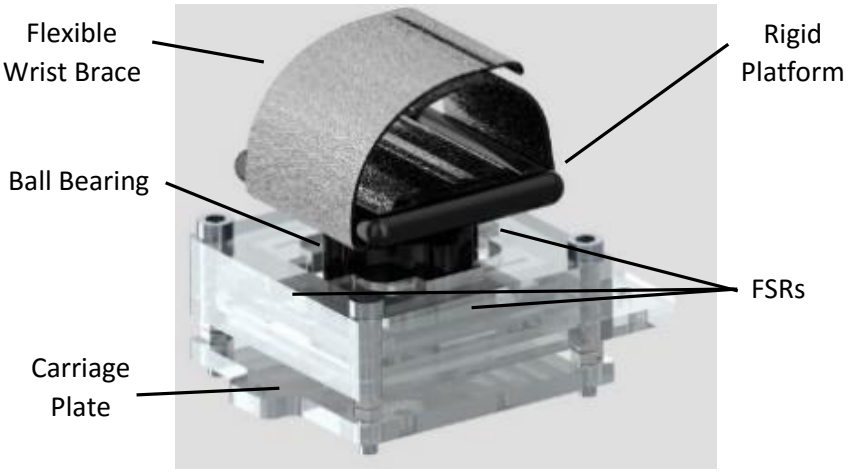


Figure 2-8: Complete Design of Wrist-Interface

2.5.2. Recording Forces

The second responsibility of the wrist interface is to record the net force between the system and the user. The recording of interaction forces helps ensure excessive forces are avoided and allows force inputs to be used as an additional control method for the system. While many varieties of load cells exist which are capable of recording torque and force in all directions, the high cost of such devices would go against the secondary goal of keeping the project as cost-effective as possible. Force-sensing resistors (FSRs) were used as a low-cost alternative to load cells. FSRs are resistors whose resistance depends on the amount of force being applied to the sensor. The simple circuit shown below allows the resistance of an FSR to be determined by measuring the voltage at V_A, which then allows the force applied to the sensor to be calculated, assuming the relationship between force and resistance is known.

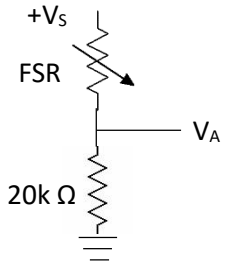


Figure 2-9: Schematic for FSR Circuitry

For this design, the torques and vertical force applied by the user to the system and vice versa were deemed unnecessary, since the system was only capable of providing actuation in the horizontal 2-D plane. Furthermore, the force input would be used to determine general intentions of the user, not calculate exact desired locations, meaning only the direction and approximation of magnitude of the forces were needed, not the exact magnitude. FSRs would be able to meet these conditions, as long as the design wrist interface allowed the FSRs to measure forces in all directions in the 2-D plane and transmit forces between the user, FSRs, and carriage with as little loss as possible.

2.5.3. Design and Validation

Various designs of the FSR-holding apparatus were tested via ANSYS, in order to determine which design best translated force readings to true force inputs. An early design and the final design are both shown on the following page, to emphasize the final design's ability to decouple moments and linear forces relative to other designs. The first column of pictures show a preliminary design and the second column show the final design. The upper ANSYS figures show the stresses resulting from a horizontal force acting on the center of the design towards the upper right side. As seen in both instances, higher stresses exist in the direction of the force, suggesting the FSRs would be able to accurately detect the applied force. The lower ANSYS figures show the stresses resulting from a horizontal force acting on a location slightly to the right of the center towards the lower corner. Since the force is slightly off-center, it creates a moment about the center as well as a linear force. With the preliminary design, this moment causes compression around the entire apparatus, which would result in the FSRs detecting forces in all directions.

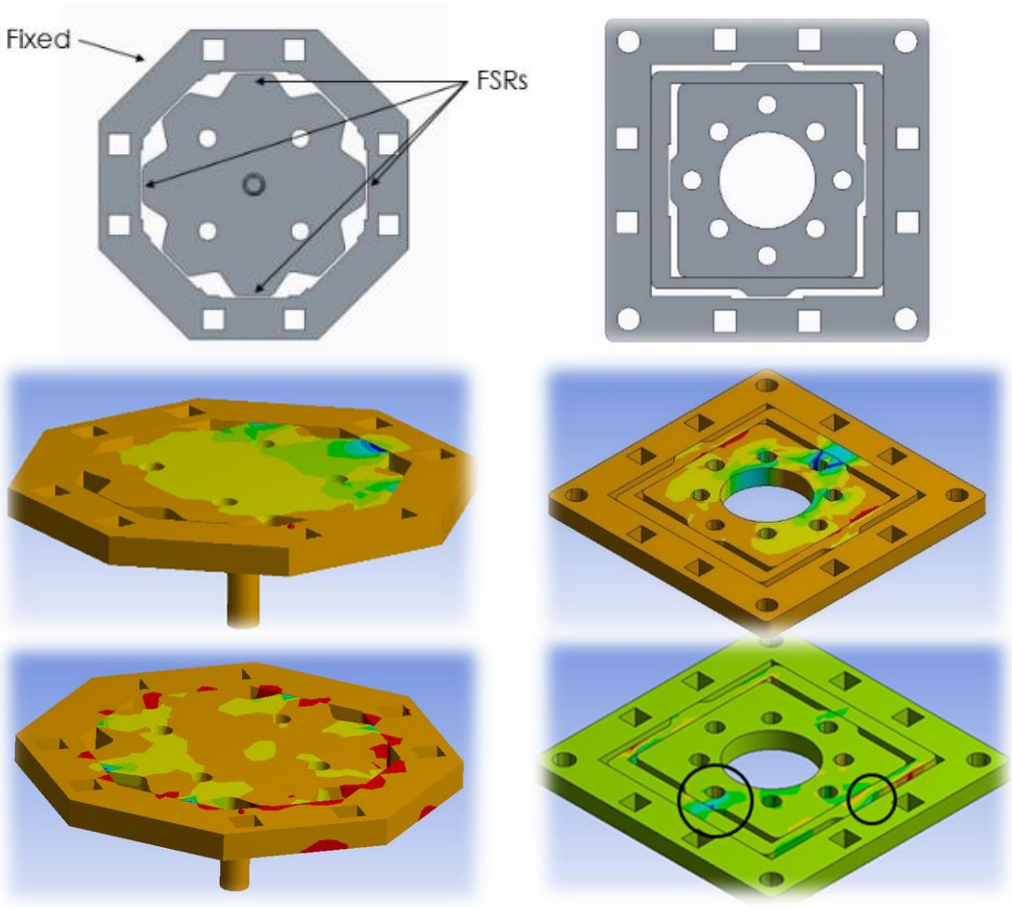


Figure 2-10: FSR Holder Design and ANSYS Stress Evaluation. The left column represents the original design and the second column shows the final design. Top figures show the mechanical design, middle figures show the stress from a centrally located horizontal force directed toward the upper right side, and bottom figures show the stress from an off-center horizontal force toward the bottom corner.

The final design solves this by decoupling moments from linear forces. As shown in the upper figure, the final design separates forces into components acting along two perpendicular axes. Of course induced moments still elicit reaction forces, but these forces are handled away from the FSR locations. The ANSYS simulation of the final design shows that for the example described above, it is able to decouple moments from linear forces and only cause stresses in the

direction of the applied force. Testing the final design with real loads verified the simulations, with the FSRs only registering forces in the direction of the applied force.

2.6. Eye-Tracker Mount

Another important consideration for the mechanical design of the system developing a mount for the eye-tracker that was both adjustable and able to be locked into place. For different users, it was essential that the eye-tracker mount be able to adjust to differences in height. However, during a session it was necessary to keep the eye-tracker locked in place once calibration had completed. The eye-tracker mount also needed to allow ample space for the user to reach the wrist interface without having to bend unnaturally to avoid parts of the mount. The design shown below allows the eye-tracker to be repositioned vertically for different user heights. The eye-tracker can also slide left to right or be set to a different angle, which allows slight variations to be used for individuals the eye-tracker may otherwise have difficulty observing.

Figure 2-11: Design of the Eye-Tracker Mount

The eye-tracker hangs on the underside of the cross piece, which gives as much room as possible between the table and the mount for the user's arm to move freely. The second picture shows that the wooden eye-tracker holder is mounted to the sliding component by hinges which allow the eye-tracker to pivot.

2.7. Mechanical Safety Considerations

The majority of the safety features of the design exist in the coding, but a few are included in the mechanical design as a failsafe, in case a glitch in code or hardware cause the system to behave erratically. To begin, the system has a few features to protect itself in case of failure. The carriage and the spring-holder are padded at the locations where the two meet when the spring is fully compressed. Thus if the motor were killed, the padding would absorb a portion of the energy released by the retraction of the spring, which would reduce the likelihood of fracture within the rigid components of the carriage and the spring-holder. The system also has mechanical stops in all directions, meaning in a worst-case scenario the system will still be confined to a set workspace.

With regards to protecting the user, the mechanical stops provide a first measure of safety. In addition, the wrist-interface currently rests in place with four bolts placed through holes which are rigidly attached to the system. The bolts are sufficient to hold the wrist-interface in place if only horizontal forces are present, but should the user apply a vertical load, the wrist-interface will freely slip up and out of its position, and away from all contact with the rest of the system, as shown in the diagram below.

Figure 2-12: Wrist-Interface shown Connected and Disconnected from Carriage Plate

Finally, a mechanical disconnect is attached to the positive terminal of the battery, allowing electrical power to be cut from the entire system.

2.8. Bill of Materials

Table 2-1: Cost of Materials for Mechanical Components of Entire System

Quantity	Part Number	Description	Supplier	Unit Cost	Total Cost
		Mechancal Construction			
1	8589K84	1/4" Thick, 24" x 48" Acrylic	McMaster-Carr	$55.76	$55.76
1	9293K56	5lb Constant-Force Spring	McMaster-Carr	$7.85	$7.85
1	8701K45	UHMW Polyethylene Rod - 1" Diameter x 1' Length	McMaster-Carr	$3.08	$3.08
1	91259A102	1/4" x 1-3/4" Shoulder Screw, 10-24 Thread	McMaster-Carr	$1.53	$1.53
1	93070A121	M5 x 10mm Socket Head Cap Screws (Qty: 50)	McMaster-Carr	$9.40	$9.40
1	90327A126	M5 x 12mm Socket Head Cap Screws (Qty: 50)	McMaster-Carr	$6.33	$6.33
1	9059A012	M5 Plain Steel Hex Nut (Qty: 100)	McMaster-Carr	$1.70	$1.70
4	5674K1	Flange-Mount Ball Transfer	McMaster-Carr	$3.07	$12.28
20	5537T454	Steel End-Feed Fastener, M5 (Pack of 4)	McMaster-Carr	$2.55	$51.00
2	5537T101	Aluminium T-Slotted Framing Extrusion (8ft)	McMaster-Carr	$21.58	$43.16
2	92510A357	Aluminum Unthreaded Spacer, 3/8" Screw Size	McMaster-Carr	$2.32	$4.64
1	5912K5	Self-Lubricating Bronze Bearing, 1/2"	McMaster-Carr	$12.28	$12.28
1	6086K111	Quick-Disconnect Bushing, 1/2" Bore	McMaster-Carr	$12.24	$12.24
1	5972K326	Steel Ball Bearing, Double Shielded, 10mm Diameter	McMaster-Carr	$4.80	$4.80
1	161659	2" x 4" x 10 ft Lumber	Home Depot	$4.09	$4.09
1	B00R575B46	80/20 Aluminum Corner Bracket w/Tabs (Qty: 25)	Amazon	$18.50	$18.50
2	B009YSHYTE	6061 Aluminum Sheet, 1/4" x 8" x 8"	Amazon	$13.23	$26.46
1	B001AXF31M	50-lb Fishing Line	Amazon	$2.10	$2.10
1	B006AR54TY	Futuro Sport Wrap Around Wrist Support	Amazon	$6.99	$6.99
1	B0035FZTSO	Steel Key Stock, 1/8" x 1/8"	Amazon	$3.65	$3.65
1	TBI-TR20N-0900-30-30	900 mm Linear Guide Rail	Anaheim Automation	$78.00	$78.00
1	TBI-TRS20VN-N-Z0	Carriage	Anaheim Automation	$35.00	$35.00
			Total:		$400.84

The total cost of the mechanical components for the system is roughly $400. The system also included the following 3D-printed materials: Motor spool, holder for the rotational potentiometer, rotational motor to potentiometer connector, wrist brace attachment, and a housing for the bearing on the wrist interface. The pieces laser-cut from the 24'' by 48'' piece of acrylic included a cover for the electronic component housing, a platform for the linear motor, a plate attached to the carriage, six pieces comprising the wrist-interface, and three pieces comprising the spring holder.

CHAPTER 3

ELECTRONICS AND HARDWARE ARCHITECTURE

This chapter will give an overall schematic to help depict the structure of the hardware and communication needed between components. The chapter will then elaborate on the specifications of the electronic components of the system.

3.1. Overall Schematic

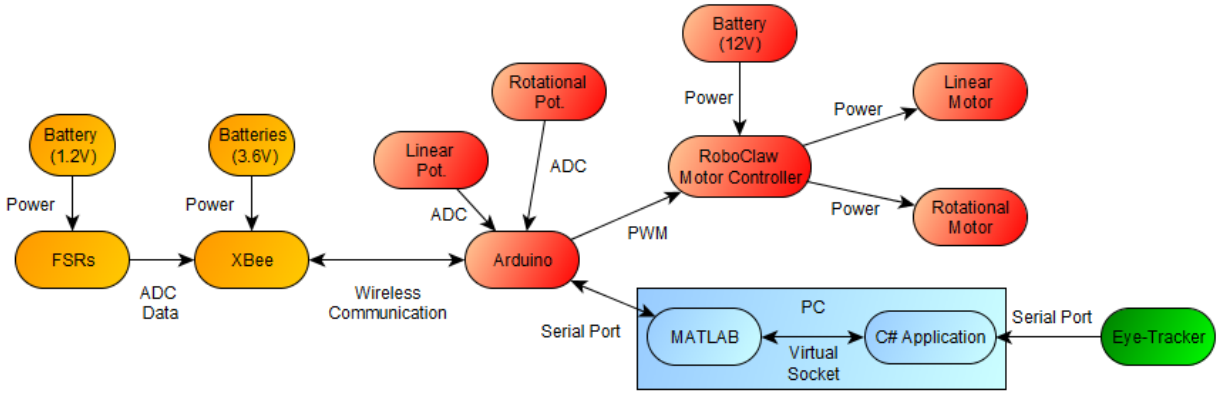


Figure 3-1: Schematic of Electronic Components for Entire System

3.2. Brushed DC Motor for Linear Subsystem

The motor used to perform the actuation for the linear subsystem is a MY1016 brushed DC motor from Unite Motor Co. This 24V 250W motor has a rated torque of 0.9 N · m and a rated speed of 2750 rpm. As described in Chapter 2, a 3D-printed spool with a diameter of 14.2 mm was attached to the shaft of the motor, which allows the motor to provide over 50 N (~11 lbs) of linear force and drive the end-effector at up to 2 meters per second. Even if driven with a 12 V battery, as ended up being the case, the maximum velocity is just over 1 meter per second,

meeting the requirements laid out in Chapter 2. Below is a picture of the motor used for linear motion.

Figure 3-2: MY1016 DC Motor Used for Linear Actuation

3.3. Brushed DC Motor/Gearbox for Rotational Subsystem

The motor used to perform the actuation for the rotational subsystem is a M4-R0062-12 FIRST CIM brushed DC motor from BaneBots LLC. This motor has a nominal voltage of 12V and provides 18.2 mN · m of torque per amp, up to a maximum torque of 2.42 N · m at 133 amps. Its no-load speed is 5310 rpm. The picture below and to the left is of the FIRST CIM motor.

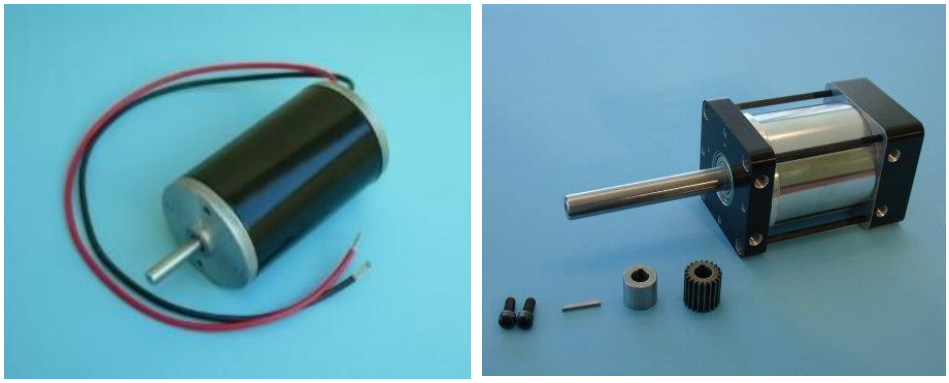


Figure 3-3: FIRST CIM Motor (Left) and P80 Gearbox (Right) Used for Rotational Actuation

The picture above and to the right is of the BaneBots P80 Planetary Gearbox with a 64:1 ratio, to which the FIRST CIM motor was paired. With the gearbox, the effective maximum torque is roughly 150 N · m (above the maximum suggested torque of 115 N · m), and a no-load speed of 83 rpm. Even with an efficiency of 50%, this is able to meet the requirements laid out in Chapter 2.

3.4. RoboClaw Brushed DC Motor Controller

The motor controller for our system needed to be able to control the two brushed DC motors previously described. Both motors needed to be drivable in both directions, and ideally the controller would have a simple interface to keep the cost and complexity of the system as minimal as possible. The maximum current for the linear subsystem's motor was 14 amps, while the rotational subsystem's motor could handle 100 amps before exceeding the maximum suggested torque of 115 N · m, assuming perfect efficiency. Although perfect efficiency is impossible in a real system, limiting the maximum current to 100 amps ensured the BaneBots gearbox would not be damaged with some factor of safety. The RoboClaw 2 x 60A met the requirements presented by the motors with two channels, each having a continuous current rating of 60A and a peak current rating of 120A.

Figure 3-4: RoboClaw Motor Controller

The RoboClaw has an operating voltage of between 6V and 34V, which easily covers the range of both motor voltages. The RoboClaw can also receive control signals as analog, serial or RC signals, making it a very versatile controller and easily controllable with an Arduino, which was the microcontroller used for the system.

3.5. Arduino Uno Microcontroller

From the beginning, a goal of this project was to keep the system as low-cost and accessible as possible. The Arduino Uno provided the perfect opportunity to do just this, as its specifications met the requirements of the system at a fraction of the cost of more powerful microcontrollers. The raw requirements for the microcontroller included six analog inputs (four FSRs and two potentiometers), at least two PWM pins for controlling the motors, and the ability to communicate with the computer, in this case via a USB serial port. The Arduino Uno featured all of these components, with the additional benefit of being widely used by the public, meaning control of the system would be easily understood by outsiders with no previous knowledge of the system. While offering no immediate benefit to the system presented in this paper besides keeping the cost to a minimum, the use of popular components, such as an Arduino Uno, will hopefully encourage use and continued development of the system in the future.

Figure 3-5: Arduino Uno Microcontroller

3.6. XBee Wireless Module

Two XBee S2 modules are used to enable wireless communication between the FSRs and the Arduino. Since the FSRs were attached to the carriage as it moved around the workspace of the system, reading from the FSRs via eight wires would likely be incredibly messy. While a method of organizing the wiring could be developed if necessary, a more clean solution is to transmit the information wirelessly. The XBee S2 modules made this possible, by connecting one XBee to the FSRs in the carriage and connecting another XBee to the Arduino located at the electronic hub of the system. The following table gives the specifications for the XBee S2 module.

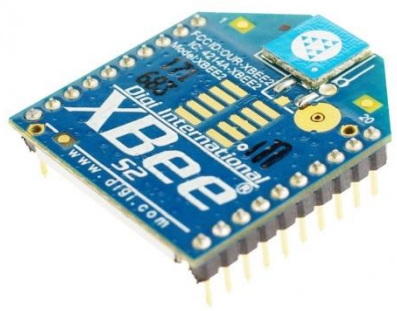


Figure 3-6: XBee S2 Module

Table 3-1: XBee Specifications [83]

Specification	XBee	XBee-PRO (S2)
Performance		
Indoor/urban range	up to 133 ft. (40 m)	Up to 300 ft. (90 m), up to 200 ft (60 m) international variant
Outdoor RF line-of-sight range	up to 400 ft. (120 m)	Up to 2 miles (3200 m), up to 5000 ft (1500 m) international variant
Transmit power output	2 mW (+3dBm), boost mode enabled 1.25 mW (+1dBm), boost mode disabled	50 mW (+17 dBm) 10 mW (+10 dBm) for International variant
RF data rate	250,000 b/s	250,000 b/s
Data throughput	up to 35000 b/s (see XBee ZB transmission, addressing, and routing on page 60)	up to 35000 b/s (see XBee ZB transmission, addressing, and routing on page 60)
Serial interface data rate (software selectable)	1200 b/s - 1 Mb/s (non-standard baud rates also supported)	1200 b/s - 1 Mb/s (non-standard baud rates also supported)
Receiver sensitivity	-96 dBm, boost mode enabled -95 dBm, boost mode disabled	-102 dBm
Power Requirements		
Supply voltage	2.1 - 3.6 V	3.0 - 3.4 V

The supply voltage and standard baud rate compatibility, in addition to an operating current of 45mA, make the XBee S2 compatible with the Arduino Uno, while the data transmission rate of 250,000 b/s and the range of 90m show the XBee S2 satisfies any requirements of the system.

3.7. Potentiometer for Linear Subsystem

The potentiometer for the linear subsystem was part of a setup described in Chapter 2 that allowed it to read the position of a string attached to the carriage while keeping the string taught [84]. The primary concern for this potentiometer was being able to read a wide range of motion. While the diameter of the spool to which it attaches can be changed to give any linear range for any potentiometer, ideally the potentiometer would have a large range to allow the spool to be as compact as possible. The chosen potentiometer is a 10-turn potentiometer, which gives a total measurable linear range of 0.94 meters when attached to a 30mm-diameter spool. Since the Arduino has a 10-bit ADC, the ten revolutions of the potentiometer map to 1024 counts on the Arduino, giving a total resolution of 3.5 degrees. However, the more important measure of resolution is linear distance, which will be calculated in Chapter 5.

Figure 3-7: Potentiometer Used for Measurement along Linear Subsystem

3.8. Potentiometer for Rotational Subsystem

The goal for the potentiometer for the rotational system was the opposite – to have a small mechanical range, which would mean the small rotational range of the system would have the greatest resolution possible. The chosen potentiometer, part PTV09A-4020F-B103 from Mouser Electronics, is a ¾ turn linear potentiometer, with a mechanical angular range of roughly 280 degrees. When read by an Arduino, this potentiometer gives the rotational subsystem a resolution of 0.00076 degrees, or 4.77 mm at one meter out.

Figure 3-8: Potentiometer Used for Measurement of Rotational Subsystem

3.9. Force-Sensing Resistors (FSRs)

The force interactions between the system and the user are measured by force-sensing resistors. As described in Chapter 2, FSRs are resistors whose resistance depends on the amount of force being applied to the sensor. The inclusion of an FSR in a simple circuit allows the resistance of the FSR to be measured, which in turn allows the applied force to be calculated once a relationship between force and resistance is determined. The primary benefit of force-sensing resistors, in addition to their relative simplicity, is their extremely low cost. Whereas load cells typically run from hundreds to thousands of dollars, the short-tailed 0.6’’ diameter FSR used here, Pololu part 2728, costs under $6. The FSRs used have a force sensitivity range

of ~0.2 to 20N, which corresponds to a resistance of over 100MΩ when no force is applied to a few hundred Ohms when the maximum load is applied. A short-tailed FSR is shown below.

Figure 3-9: Short-tailed Force-Sensing Resistor

3.10. Batteries

Batteries are used to power the entire system. A set of four smaller AAA batteries provides power to the wireless system, including four FSRs and an XBee S2 wireless module. A larger battery located with the electrical components in a stationary position powers the RoboClaw motor controller, and thus both motors. The main battery is a CSB HR1290W high capacity lead acid battery. This 12V battery has a capacity of 23 amp-hours and is rated for a maximum discharge current of 300 amps, which is enough to supply both motors with their maximum currents.

Figure 3-10: CSB HR1290W High Capacity Lead Acid battery

3.11. Eye-Tracker

The eye-tracker used for the system is a Tobii EyeX Controller. This eye-tracker is commercially available and uses near-infrared microprojectors to create reflection patterns on the user's eye, which are then registered by the controller. The reflected image is processed to find the location and gaze direction of the user's eyes. The eye-tracker has a length of 12.5'' and a weight of 91 grams. The EyeX eye-tracker is still under development and evaluation, meaning technical specifications have not yet been released. However, previous versions of Tobii eye-trackers have obtained precisions of less than 1 degree, even under varying illumination settings [78]. Evaluation of the Tobii EyeX eye-tracker alone is given in Chapter 5, and evaluation of the eye-tracker when implemented with the system is given in Chapter 6. A picture of the Tobii EyeX is shown below.

Figure 3-11: Tobii EyeX Eye-Tracker

3.12. Bill of Materials

Table 3-2: Bill of Materials for Electonic Components

Quantity	Part Number	Description	Supplier	Unit Cost	Total Cost
Electronic Compontents					
1	B00ITELF12	Battery Disconnect Cut Off	Amazon	$8.62	$8.62
1	B00INVF468	Black 10-Gauge Wire	Amazon	$10.28	$10.28
1	B000K7GRCI	Solderless Wire Terminal and Connection Kit	Amazon	$12.56	$12.56
1	B00NNDAFW4	EBL AAA Charger w/ 8 AAA Batteries	Amazon	$15.99	$15.99
1	B00829IN36	3 x 1.5 V AAA Battery Holder	Amazon	$3.40	$3.40
1	B00H8T6J3S	1 x 1.5 V AAA Battery Holder	Amazon	$3.57	$3.57
1	HR1290W	CSB High Rate AGM Battery	AtBatt	$34.99	$34.99
1	LC1-12-3A	Leoch 12V/3A SLA Battery Charger	AtBatt	$26.99	$26.99
1	AM-2618	Sting Potentiometer Kit	AndyMark	$17.00	$17.00
1	FIRST CIM Motor	FIRST CIM Motor	BaneBots	$28.00	$28.00
1	P80 Gearbox	Planetary P80 CIM Gearbox, 64:1	BaneBots	$143.25	$143.25
1	MY1016	United 250W 24V DC Motor	Motion Dyanamics	$45.95	$45.95
1	1499	RoboClaw 2x60A Motor Controller	Pololu	$199.95	$199.95
4	2728	Force-Sensing Resistor: 0.6" Diameter, Short Tail	Pololu	$5.80	$23.20
1	N/A	Tobii EyeX Eye Tracker and Development Kit	Tobii	$139.00	$139.00
1	N/A	Arduino Uno Rev3	Arduino	$24.95	$24.95
2	WRL-10414	Xbee 2mW Wireless Antenna Series 2	Sparkfun	$22.95	$45.90
				Total:	$783.60

The total cost of the electronic components for the entire system is under $800.

CHAPTER 4

SOFTWARE ARCHITECTURE

The current chapter explains the communication between the hardware components described in the previous chapter. A full schematic will be given with an accompanying explanation, followed by specific details for each communicative link in the system. Details on the setup of each connection will be discussed, as well as the resulting speed of data transmission achieved by each.

4.1. Overall Schematic

The schematic below shows the overall structure of the system's software, following the color scheme of the schematic in Chapter 3.

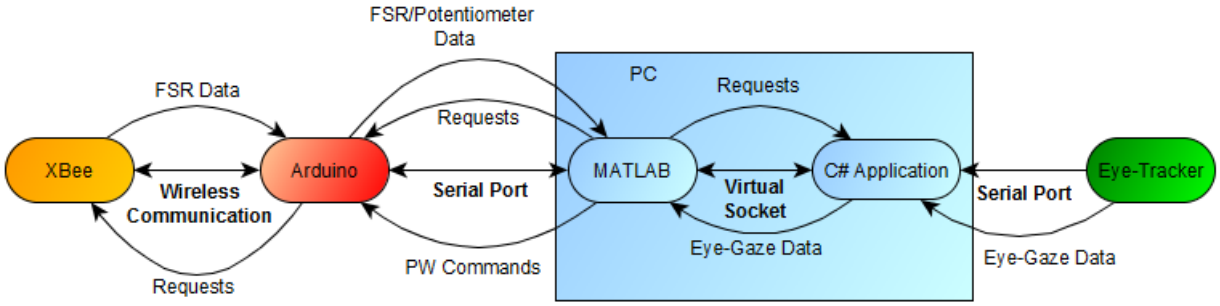


Figure 4-1: Schematic of Software Layout for Entire System

4.2. XBee to Arduino Wireless Communication

There are actually two XBee modules used in the system. One is attached to the wrist-interface and is used to transmit FSR data wirelessly, and the other is a stationary module connected to two Arduino digital pins and used to receive and relay wireless information from

the remote XBee module. The Arduino is first given a virtual serial port, using the AltSoftSerial library [82]. The XBee module's receive and transmit pins are directly connected to the pins designated for Arduino's virtual serial port, which allows Arduino's serial communication over the port to be automatically broadcast wirelessly. In this manner, there isn't true communication between the Arduino and the stationary XBee module, but rather the XBee is hardwired to be a wireless extension to the Arduino's virtual serial port.

During Arduino startup, the Arduino issues a command to set the baud rate of its virtual serial port to 57600 bits per second, the fastest allowed for XBee communication. Once initialized, the Arduino only requests wireless FSR data when it receives a MATLAB command to do so. If such a request is received, the Arduino sends a 19 byte request command through its virtual port, which is broadcast wirelessly via the stationary XBee. This request tells any receiving XBee modules to measure the analog readings on all four ADC pins and wirelessly return the results.

After sending the request, the Arduino continues normal operation. The virtual serial port is polled with every loop and any available data is stored. Once all four ADC values are received from the wireless XBee module, the data is compiled and returned to the MATLAB application. The entire process from Arduino requesting wireless data to Arduino receiving wireless data takes roughly 50 milliseconds, meaning the FSR data can be read at roughly 20 Hz.

4.3. Arduino to MATLAB Communication

The Arduino is connected to the central MATLAB application via a wired serial port. This connection is responsible for sending MATLAB requests for FSR data to Arduino, sending MATLAB pulse-width commands to Arduino, and returning FSR data from Arduino to

MATLAB. There are a few libraries that exist to facilitate communication between MATLAB and Arduino. Originally a MATLAB library was used for communication, but it was considerably slower than direct serial communication. The two pictures below show code which requests 100 sets of analog data to be sent from Arduino to MATLAB using the available library, and the resulting average time of each request.

```
>> iter=0;
>> sumTimes=0;
>> while(iter<100)
iter=iter+1;
tic;
readVoltage(a,0);
sumTimes=sumTimes+toc/100;
end
>> sumTimes

sumTimes =

    0.0231
```

```
>> iter=0;
avgTimes=0;
while(iter<100)
iter=iter+1;
tic;
readVoltage(a,0);
readVoltage(a,1);
readVoltage(a,2);
readVoltage(a,3);
readVoltage(a,4);
readVoltage(a,5);
avgTimes=avgTimes+toc/100;
end
>> avgTimes

avgTimes =
```

Figure 4-2: Times for Requesting One (Left) and Six (Right) Arduino ADC Data Values using an Arduino Library

The picture on the left shows that it took roughly 23.1 milliseconds to request a single analog voltage reading, which was quicker than the FSR data would be wirelessly received and would thus be sufficiently fast. However, the picture on the right shows that this time is linearly proportional to the number of analog readings requested. Thus to read four FSR voltages and two potentiometer voltages would take over 132 milliseconds, which would create significant lag and is unacceptable for this system.

To improve the speed of communication between MATLAB and Arduino, the existing library was abandoned and a simple communication infrastructure was implemented on both ends. At startup, MATLAB is set to open a serial port with the Arduino with a baud rate of

115200 bits per second, the fastest suggested for Arduino applications. MATLAB then communicates with Arduino via custom message arrays, where the first byte entry indicates the type of message being transmitted. The Arduino code receives these message arrays, determines the message type based on the first byte entry, and responds accordingly. The example below shows the average times for the custom communication handling. The picture on the left shows the results when the Arduino is programmed to respond to a message of [1,0,0] with a single analog value. The picture on the right shows the results when Arduino is programmed to respond by reading and returning six analog values.

```
>> iter=0;
avgTimes=0;
tic;
while(iter<100)
    iter=iter+1;
    if(a.TransferStatus == 'idle')
        readasync(a,2);
    else
        continue;
    end
    fwrite(a,[1,0,0]);
    while(a.BytesAvailable<2)
    end
    fread(a,2);
end
avgTimes = toc/100

avgTimes =

    0.0020
```

```
>> iter=0;
avgTimes=0;
tic;
while(iter<100)
    iter=iter+1;
    readasync(a,12);
    fwrite(a,[1,0,0]);
    while(a.BytesAvailable<12)
    end
    fread(a,12);
end
avgTimes = toc/100

avgTimes =

    0.0071
```

Figure 4-3: Times for Requesting One (Left) and Six (Right) Arduino ADC Data Values using Custom Methods

The pictures above show a significantly reduced time needed for MATLAB to Arduino communication, with all six analog values being requested and returned in under 10 milliseconds. Additionally, even though it takes longer for six analog readings than one, the

relationship is no longer linear, which means additional data transmission will have less of an impact on the communication speed than if the MATLAB library were used.

The final structure for communication between MATLAB and Arduino has three message types. One for initializing the Arduino, one for requesting analog data from the Arduino, and one for commanding the Arduino to send desired pulses to the RoboClaw motor controller. Once MATLAB sends a request, it will wait until the response is received or until 20 milliseconds has passed with no response, in which case it will send the request again. The final setup allows the MATLAB application to receive potentiometer data at over 50 Hz, receive FSR data at over 10 Hz, and send pulse commands at over 50 Hz.

4.4. MATLAB to C# Application Communication

The MATLAB and C# applications communicate via a virtual socket connection. Since the only information sent from MATLAB to C# is a request for eye-tracker data, the C# application doesn't need to sort the messages it receives. Instead, it is set to continuously poll the socket connection and return the most recent eye-tracker data collected if any communication has occurred over the socket connection. MATLAB requests eye-tracker data with every cycle, or at roughly 50 Hz, and does not wait for a response. Instead, MATLAB simply checks whether the response has been received with every loop. If so, the eye-gaze data is updated and the controller continues. The system is able to retrieve eye-tracker data at over 40 Hz.

4.5. Eye-Tracker to C# Application Communication

The Tobii Eye X eye-tracker is connected to the C# application through a USB serial port. An SDK from Tobii handles the communication with the eye-tracker and is included in the C# code. When eye-tracker data is available, an interrupt is called and the data is stored within variables in the C# application itself. These variables are then passed to MATLAB whenever a request is received, as explained in the previous section.

CHAPTER 5

CONTROLLER DESIGN

The following chapter gives the structure of the controller design for the entire system. The controller is based in the MATLAB application, which as described in the previous chapter, sends and receives data to and from the Arduino and the eye-tracker. Thus this chapter will only look at the portion of the overall logic which relates to interpreting the data once it has been sent to the central MATLAB application. First a description of the structure of the entire controller will be given, followed by descriptions of the controller under force-based control and then position-based control. Each section will include details regarding the calibration needed for that particular control mode.

5.1. Overall Controller Structure

The overall controller can be divided into two separate sub-controllers, one of which controls the system via force inputs and the other of which controls the system via eye-tracker inputs and position data. Each sub-controller outputs a desired velocity, which is then fed into the main controller and converted to the appropriate pulse signal to send the RoboClaw. Only one sub-controller is active at any given time, meaning the system is either in force mode or position mode. The exact criteria for switching between control modes will be explained within each section of this chapter, but priority is given to the force-based controller, as any force inputs from the user indicates a desire to deviate from the current system trajectory. The schematic on the following page gives an overview of the entire controller.

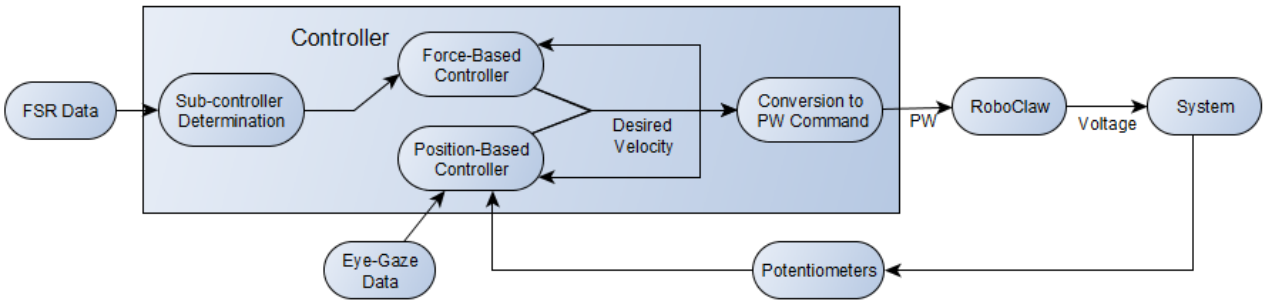


Figure 5-1: Schematic of the System Controller

It is important to note that at no point does the system measure velocity. Although the potentiometer data is accurate for positioning, the relatively low resolution of the Arduino ADC makes it unreasonable to calculate velocity by integrating. During each loop of the controller, the potentiometer values only change by a few counts. Thus the range of measurable velocities would be relatively small unless the potentiometer values were summed over a longer period of time, which would in turn cause significant lag. Instead of measured velocity data, the controller uses position data and estimated velocity for feedback. It will be shown that the system is still able to follow desired velocity trajectories.

5.2. Force-Based Control

The ultimate goal for the controller in force mode is to simulate a massless system should the user have the strength and desire to move the impaired arm without the aid of the system. Although it is not expected that this mode be used frequently, if the user finds the strength to perform a given movement, an effective assistive robotic device should never punish such movement by imposing extra resistance. Instead, this effort should be rewarded with low-inertia compliance.

The force-based control mode is initiated if the system detects a force that is due to intention rather than inertial reaction. Once initiated, the system will remain in force mode until a given time duration has passed without the system receiving any additional sufficiently large force inputs. Force mode ignores position data and instead converts force inputs (recorded by the FSRs) to acceleration inputs, which are then used to adjust the desired velocity accordingly.

5.2.1. Desired Characteristics

The requirements of the force-based controller can be simplified into two main components: decoding user intention from force inputs, and completing the desired movement. In terms of using force inputs to determine intention, the controller must first distinguish deliberate forces from involuntary reactionary forces. The controller must then combine any intentional forces with the current kinematic state of the system to determine a desired velocity to which the system should be driven. Finally, the controller must drive the system to this desired state with reasonable speed and accuracy. The force-based controller must cycle through all of these steps in a smooth manner, making the system kinetically invisible to the user.

5.2.2. Force-Feedback Structure

The system is set to force control whenever the user force input exceeds a force threshold. This threshold is proportional to the desired velocity of the system, in order to isolate meaningful force inputs from force inputs due to the inertia of the user's arm. For example, when the end-effector is being driven to a certain position, one would expect a relaxed arm to exert a force in the opposite direction. Although mathematically this force should be proportional to the acceleration, setting the threshold proportional to the velocity gives the user

greater control earlier in the process of slowing down and helps dissipate the appearance of lag in the resulting motion. A schematic of the sub-controller selection process is shown below.

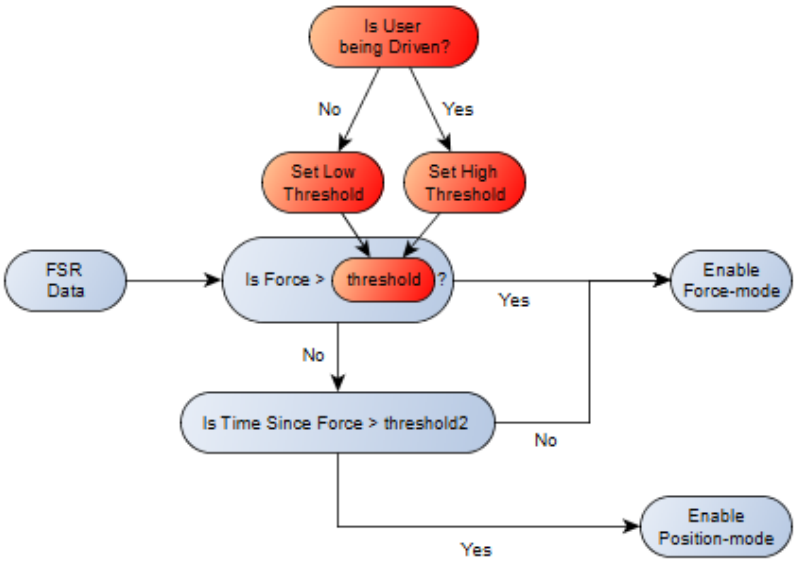


Figure 5-2: Schematic Depicting the Process of Choosing which Sub-controller to Implement

Once force mode is initiated, the controller is conceptually straightforward. Input forces are directly converted to acceleration cues, which are then multiplied by the elapsed time since the previous cycle to obtain a desired change in velocity. A schematic of the entire force-based sub-controller is shown below.

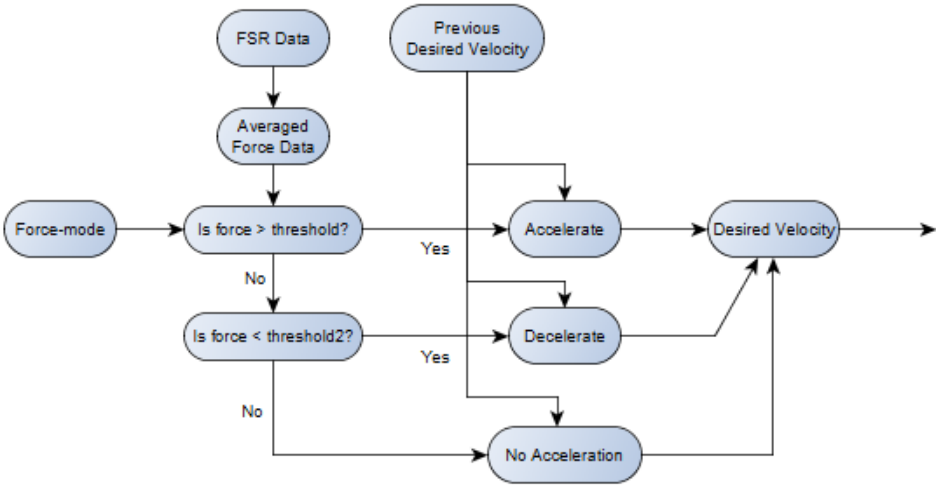


Figure 5-3: Schematic of the Force-Based Controller

Due to the imperfect nature of the low-cost electronic components used in the system, a few minor additions were made to the force mode controller. To begin, the FSRs work well for detecting forces, but are not meant to determine the exact magnitude of forces, especially small forces. Even when trying to exert no force on the end-effector, the FSRs will often detect small forces, which can then lead to a jump in the system's motion. To prevent jittery movement, force inputs below a certain force threshold are deemed negligible and do not lead to an acceleration command. In addition, all FSR signals are averaged over a few cycles (the number of which is determined by an adjustable constant) before being fed into the controller.

One consequence of averaging the force data over a set number of cycles is an increase in the lag between force input and system response. This lag is most noticeable to the user when the system is slowing down. When the system is at rest and the user applies a force, even though the force is averaged with the previous negligible forces, the average still has a non-zero magnitude and thus the system begins to move. However, when the system is in motion and the user applies an opposing force to stop movement, the average force is typically still in the direction of the motion, meaning the system will continue to move. To make the system feel more responsive, the end-effector is set to begin slowing down when the force input is below a second force threshold. The deceleration begins before the force input reaches zero, which is earlier than would previously occur, thus reducing the appearance of lag.

Once the force inputs and the described modifications are made, the output of the force-based sub-controller is a new desired velocity, which is the sum of the previous desired velocity and the change in velocity obtained from the force inputs.

5.3. Position-Based Control

A schematic of the entire position-based sub-controller is shown below.

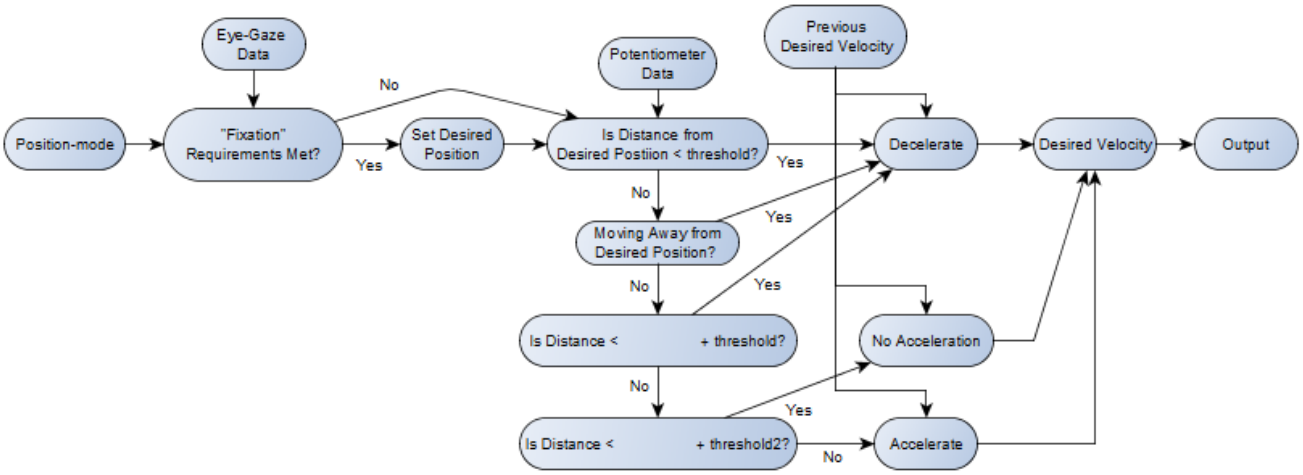


Figure 5-4: Schematic of the Position-Based Controller

The overall goal for the controller in position mode is to assist the movement of the user's arm to a desired position should the user be unable to perform the movement alone. The position sub-controller must first determine the intention of the human, in terms of desired end position, and then drive the end effector to the position with sufficient accuracy and comfort. Position mode is the default mode, and will be used as long as no deliberate force inputs are detected. In position mode, eye-tracker data is examined to determine a desired position. The desired position, along with the current desired velocity and the current position, is then used to calculate a desired velocity, which is the output of the sub-controller. As described before, this desired velocity is then converted to an analogReference value and sent to the RoboClaw to control the motors.

5.3.1. Desired Characteristics

Similar to the force-based controller, the requirements for the position sub-controller can be broken into two main components: calculating a desired position from eye-tracker data, and driving the end-effector to the desired position. With regards to deriving a desired position from eye-tracker data, the controller must be intuitive and easy to use. In terms of driving the end-effector to the desired position, the system must meet the dynamic requirements laid out in Chapter 2 by having an acceleration of 1000 $\frac{\text{mm}}{\text{s}^2}$. The combination of eye-gaze data and position feedback must allow the position sub-controller to reach a final position reasonably close to the user's true desired position. The accuracy of the controller will be evaluated in Chapter 6.

5.3.2. Position-Feedback Structure

The position sub-controller is enabled if the force sub-controller is not, namely if an intentional force input has not been detected for a specified time duration. Upon entering position mode, the desired position of the end-effector is updated to its current position, which prevents the system from ignoring adjustments made under force control. Once the position-based controller had been entered, the eye-tracker data is first analyzed to determine if the data meets the criteria for eye-gaze fixation laid out by Chen and Newman [64]:

1) The dispersion of eye-gaze points is less than some dispersion threshold

2) The persistence of these fixation points must be larger than a time threshold…"

If this criteria is met, the desired position is updated to match the position dictated by the user's gaze. If this criteria is not met, the desired position is left alone. Thus in order to drive

the system to a certain position, the user must look at said position for a time greater than a threshold, without looking elsewhere.

A third criterion was added to the requirements above to allow a more seamless integration of the force and position sub-controllers. The final requirement was that the eye-gaze must be directed toward a location more than 10 cm from the end-effector's current position. This criterion prevents jittery motion that may occur due to quick eye movements, even though the general direction of eye-gaze remains the same. The additional requirement also allows the force-based controller to be used to make any minor adjustments to desired position without those adjustments being overruled by a slight discrepancy in eye-gaze direction.

Once the desired position has been updated (or left alone), the controller determines whether the end-effector needs to accelerate or decelerate by placing the current state of the system into one of five mutually exclusive possible states. This entire process is done separately for x and y components. First, the distance between the current position and the desired position is calculated, and if this distance is less than a given threshold the system is in state one and the end-effector is set to decelerate. Thus the controller acts as a virtual damper when the end-effector is within a certain distance of the desired position, which helps the end-effector settle to a resting position near its objective. If the distance is greater than the threshold, then the system checks if the end-effector is moving away from the desired position. If so, the system is in state two and the end-effector is again set to decelerate. If the system doesn't fall into this category either, then the system checks whether it is approaching the desired position at too great of a speed. For constant acceleration,

$$V_f^2 = V_0^2 + 2a\Delta x$$

Since the system is designed to produce a constant acceleration when slowing down, the distance needed to stop ($V_f = 0$) is given by:

$$\Delta x = \frac{-{V_0}^2}{2a} \quad [2]$$

The controller checks if the distance to the desired position is less than the magnitude of $\frac{-{V_0}^2}{2a}$ minus a threshold distance. If so, the system is in state three and the end-effector is set to decelerate. The threshold distance is equal to a constant multiplied by the estimated velocity, and is used to compensate for the lag in the system. The fourth check is identical to the third, but with a slightly larger threshold distance. If the distance to the desired position is less than the magnitude of $\frac{-{V_0}^2}{2a}$ minus threshold2, the system is in state four and the end-effector is set to maintain a constant velocity. The purpose of state four is to give a small region of no acceleration between the regions of acceleration and deceleration, which eliminates the chatter that would otherwise constantly exist. Finally, if no previous check has been satisfied, the system is in state five. State five means the end-effector is far enough away from the desired position that it is free to move as quickly as possible, and thus is set to accelerate.

5.3.3. Eye-Tracker to Position Calibration

The eye-tracker was initially evaluated separately, to determine the general characteristics and accuracy of eye-gaze data it could provide. The general process for evaluation was to continuously collect groups of 50 frames of data from the eye-tracker and to print out key values, such as average eye-gaze coordinates, percentage of frames with successful tracking, etc. This allowed continuous updates on the performance of the eye-tracker while various adjustments were made, such as stand height and angle of elevation. Before significant

data was collected, multiple stand heights, elevation angles, and head positions were explored to get a general feel for the eye-tracker behavior. The eye-tracker was found to be fairly robust to head movement, as long as the eye-tracker was pointed directly at the user and the user's gaze was never directed below the eye-tracker. For a 6-foot-tall user, the ideal positioning is shown below. The user's head should be positioned such that the closest boundary of allowable motion appear just above the eye-tracker.

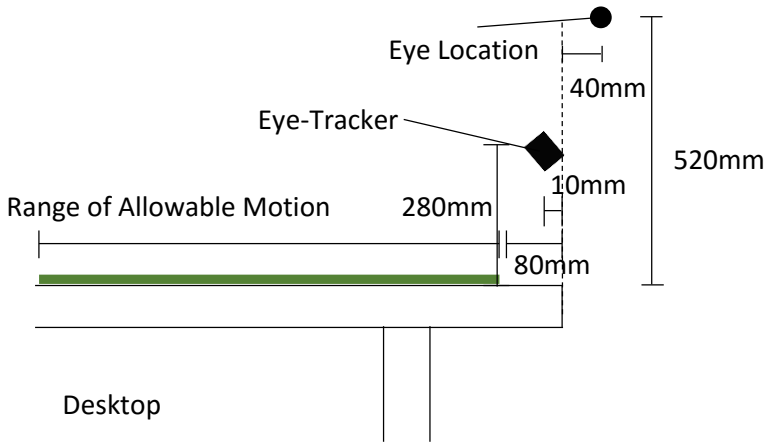


Figure 5-5: Eye-Tracker Setup Dimensions

The first true eye-tracker evaluation was to have a user look repeatedly between two points to find the average difference between true eye-gaze location and measured eye-gaze location, as well as observe the potential for drift. The graph on the following page shows the eye-gaze locations as measured by the eye-tracker for 50 consecutive fixations between two points located 100 mm apart. The measurements showed a range of 0.1 units (~50mm) in the y-direction for the points and a range of 0.06 units (~30mm) in the x-direction for the points. Although this was a preliminary test, it showed that the eye-gaze data collected by the eye-tracker should be assumed accurate to no more than 0.1 eye-tracker units.

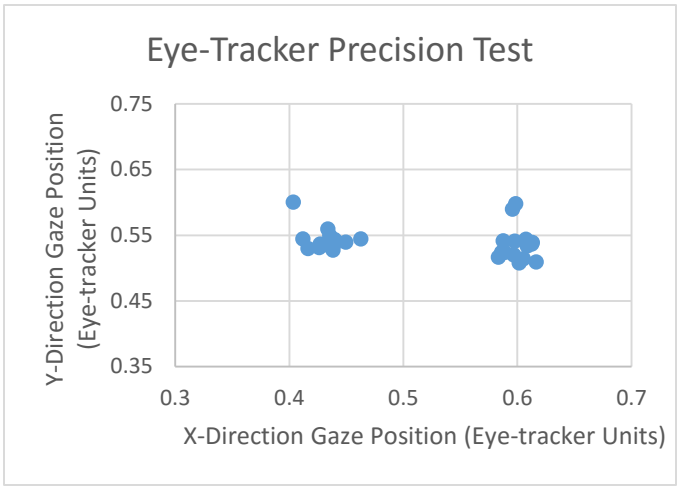


Figure 5-6: Eye-tracker Position Outputs for 50 Consecutive Fixations between Two Locations

Eye-gaze data was then collected for 30 points spread evenly across the desktop to examine the relationship between eye-tracker units and real-world position. The graph below on the left shows the true locations of the 30 points, which covered a 2-D desk space of 0.5 meters by 0.4 meters. The graph below on the right was created by collecting 50 frames of eye-gaze data and averaging the results at each of the 30 points.

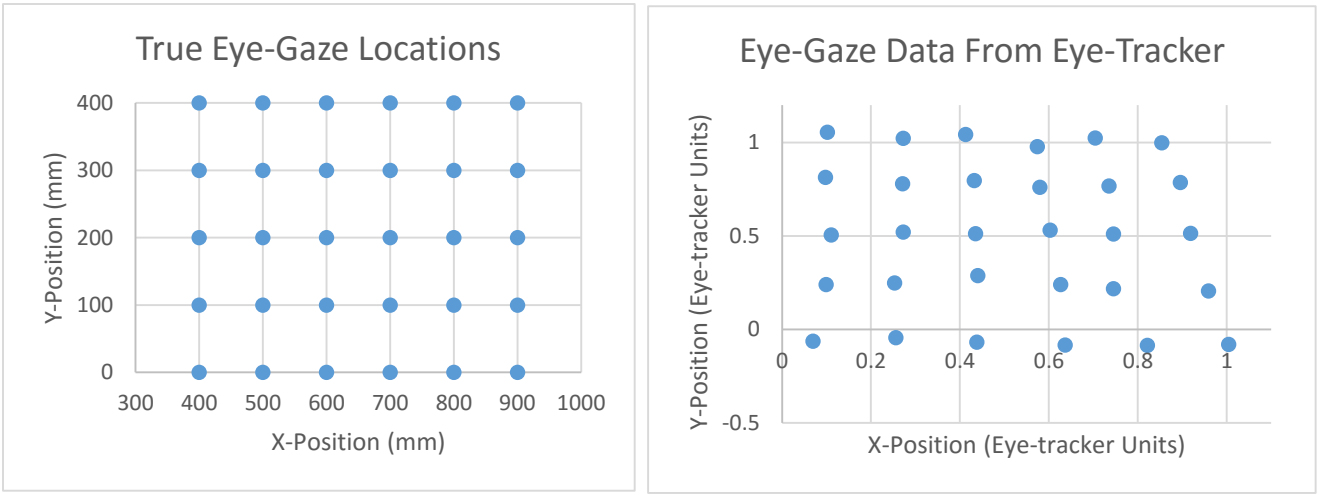


Figure 5-7: True Eye-Gaze Position (Left) and Eye-Gaze Position Detected by the Eye-Tracker (Right) for an Array of 30 Eye-Gaze Locations

The graph on the right shows a mapping from real position to eye-tracking units that resembles a trapezoid, with the eye-tracker outputting a slightly smaller range of x-values as the

y position of the eye-gaze increases. However, the maximum y-coordinate which was reachable by the user, and thus would be used by the system, was 200 mm. Between y values of 0mm and 200mm, the x-coordinate boundaries only varied by a combined 0.1 eye-tracker units, which is equal to the previously measured maximum accuracy of the eye-tracker.

Since the measured eye-gaze values appeared to be linearly proportional to the real-world endpoint of the user's gaze, a simple 4-corner calibration is performed at the beginning of a session to determine the conversions from eye-tracker units to real-world gaze position. This calibration has the user look at the four corners bounding the useable workspace for the trial, and then determine the maximum, minimum, and range for x-direction and y-direction gaze data. The accuracy of this quick calibration will be evaluated in Chapter 6.

5.3.4. Potentiometer to Position Calibrations

The potentiometers used for feedback in the position sub-controller were calibrated to allow the conversion from Arduino analog readings to real position data. For the linear potentiometer, the system was driven to various analog potentiometer values and the corresponding real positions along the x-axis were measured. The first graph on the following page shows the results. A similar calibration was done for the rotational potentiometer. The linear rail was manually rotated such that the x and y coordinates of a given point on the rail could be measured and converted to an angle. The potentiometer was then read and the value recorded. The keyed connection between gearbox shaft and the linear rail allowed the rail to rotate roughly 0.03 radians without any movement of the gearbox shaft. To account for this, multiple measurements were made after moving the rail in both angular directions and the values were averaged. The second graph on the following page shows the results.

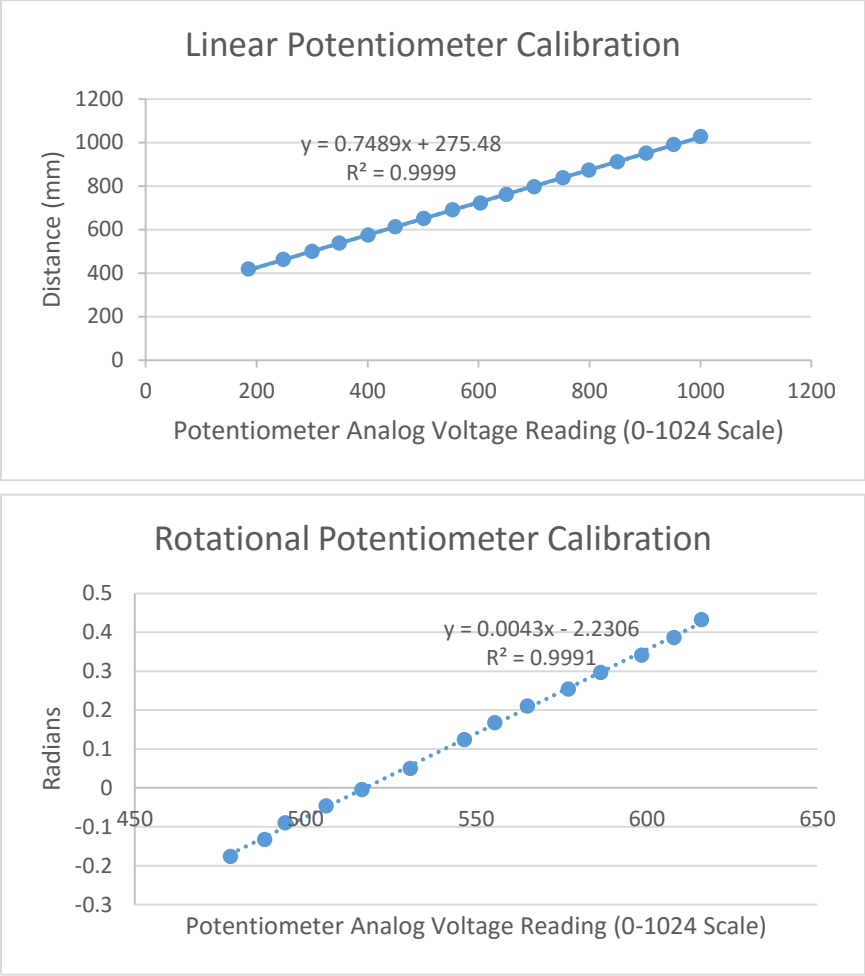


Figure 5-8: True Distance (mm) vs. Potentiometer Analog Voltage Reading for the Linear (Upper) and Rotational (Lower) Potentiometers

The final relationships between analog potentiometer readings and position are given below, with the respective potentiometer analog voltage reading denoted as PAVR:

$$\text{Linear Position (mm)} = 0.748951 * (\text{PAVR} + 367) \quad [3]$$

$$\text{Angular Position (radians)} = 0.0043 * (\text{PAVR} - 2.2306) \quad [4]$$

The high correlation values show that both relationships are reliable. However, despite the high correlation, the ability of the linear rail to rotate 0.03 radians without resistance means that angular position can only be known to within 0.03 radians.

5.4. Conversion of Sub-controller Outputs to Arduino Pulse Outputs

As has been shown, the controller as a whole is velocity based, with the desired velocity being determined by one of the two sub-controllers. The sub-controllers output x and y components of desired velocity, which are converted to polar coordinates and separately converted to pulse-widths for each respective motor.

5.4.1. Desired Velocity to Pulse-Width Calibration

The RoboClaw motor controller acts as a voltage source whose voltage is controlled via pulse-width modulation (PWM). Since the Arduino PWM signals are slow relative to the RoboClaw's ability to read voltage signals, using the RoboClaw in analog mode results in jittery output voltages. Instead, the RoboClaw is used in RC mode, in which the voltage is set based on the length of the pulse it receives. A 1000μs pulse sets the motor to full reverse and a 2000μs pulse sets the motor to full forward. Most PWM pins on the Arduino Uno operate at 490 Hz, which gives a maximum pulse-width of 2040μs. Arduino PWM functions by writing a value, which we will call analogReference, between 0 (full off) to 255 (full on). Thus the full range of pulses between 1000μs and 2000μs requires analogReference values between 125 and 250.

The RoboClaw had a few additional features that make it an effective motor controller for this system. The first feature is called exponential mode, which softens the middle control positions. This changes the relationship between analogReference and motor speed from linear to exponential, and gives much more control over the lower speeds. The graphs on the following page show the relationships between the velocity of the system and the analogReference value sent to the linear motor for both linear and exponential modes.

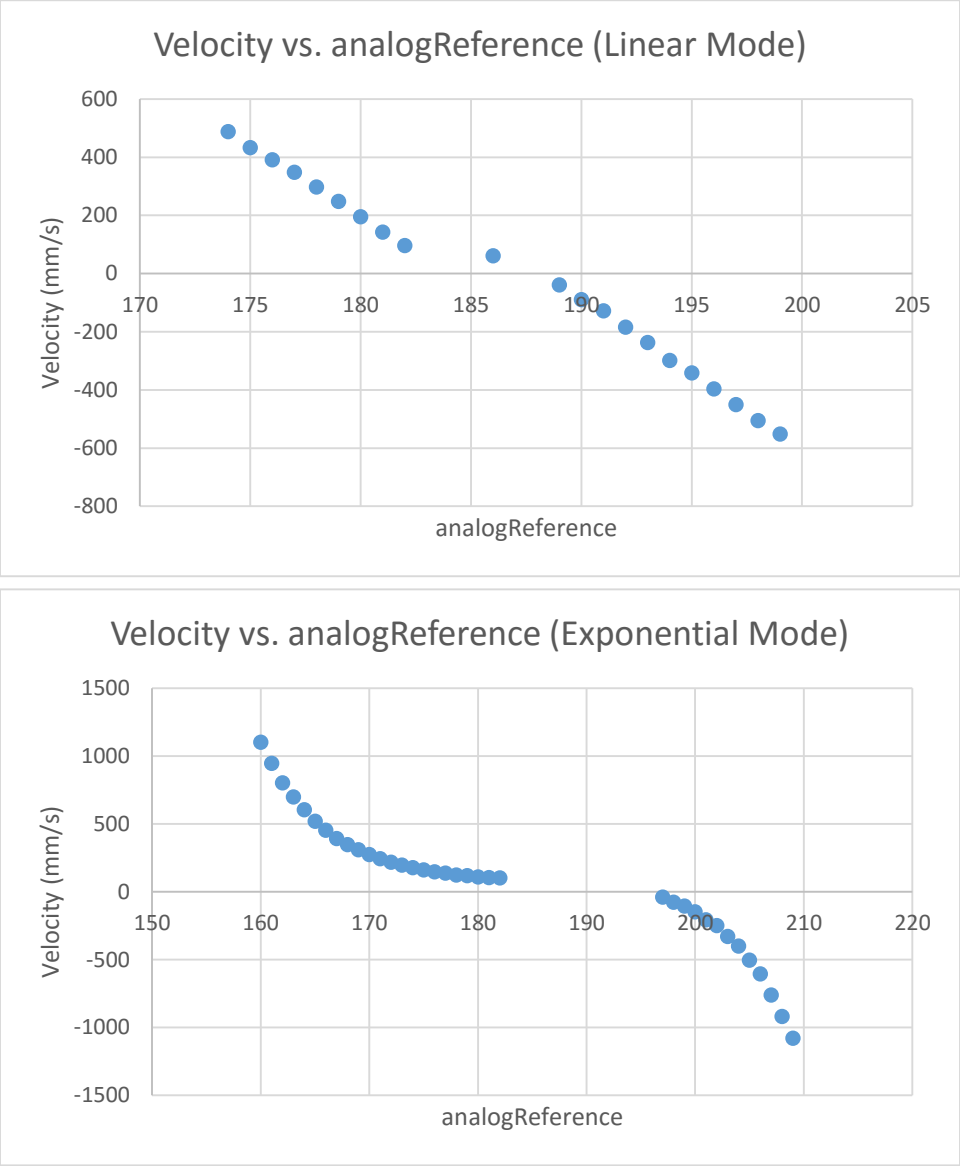


Figure 5-9: Velocity vs. analogReference for RoboClaw set to Linear (Upper) and Exponential (Lower) Modes

Since the system is most often used at low speeds, and since small changes in speed are more noticeable at low speeds, it was decided to use the controller in exponential mode. The second RoboClaw feature is the ability to auto-calibrate the range of pulses the controller receives, or more importantly, the ability to turn off auto-calibration. Auto-calibration ends up being more of a hassle during longer runs of the device, because a single pulse outside the range

of acceptable pulses recalibrates the motor controller and stores the calibration until the controller is powered off. Thus the occasional stray signal could lead to drastic changes in system behavior for the rest of a session. The RoboClaw was set to RC Mode 4, which enables exponential mode and disables auto-calibration.

The relationship between analogReference and motor velocity was found for each motor experimentally. The linear motor was driven with a constant analogReference value until the end-effector reached the end of the rail. The position data was collected and integrated to find the final velocity after the fact. The analogReference value was then increased (or decreased for testing in the opposite direction) until a speed of 1000 mm/s was reached. Below is an example of the data collected with an analogReference value of 209, which was the maximum analogReference value in the negative direction.

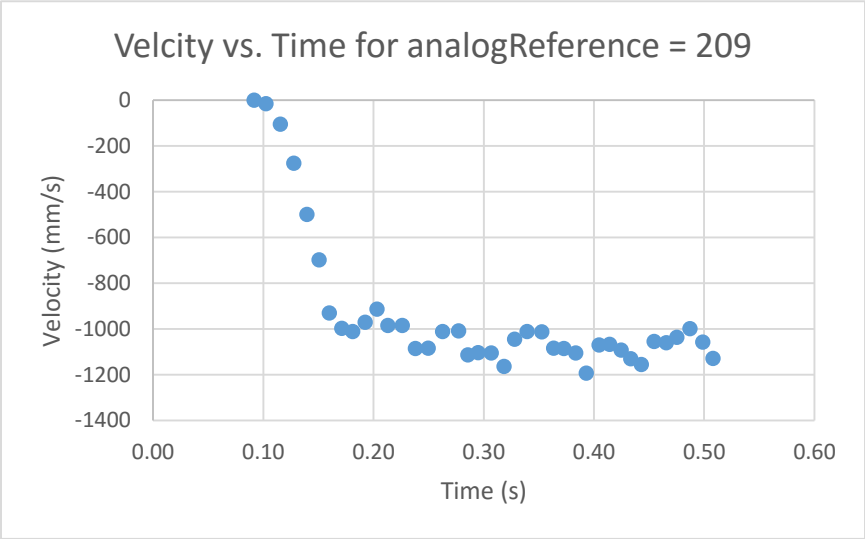


Figure 5-10: Velocity vs. Time Response of Linear System when Driven with analogReference Value of 209

As the graph shows, the system quickly drives the end-effector to a constant velocity (in this case, -1080 mm/s) and maintains that velocity until the end-effector travels the length of the linear rail. It is also important to note how quickly the linear system reaches its final velocity.

This was the longest settling time of all pulse-widths tested, yet still only took 0.16 seconds to reach a speed of over 1000 mm/s. The quickness with which the linear system reaches its target velocity allowed the desired velocity to be tracked accurately simply by sending the respective pulse duration; no other kinematic calculations were needed.

The graphs below show the final velocities of the linear system for the entire range of analogReference values. Since the ultimate goal was to convert a given desired velocity to an analogReference value, the independent variable was linear velocity.

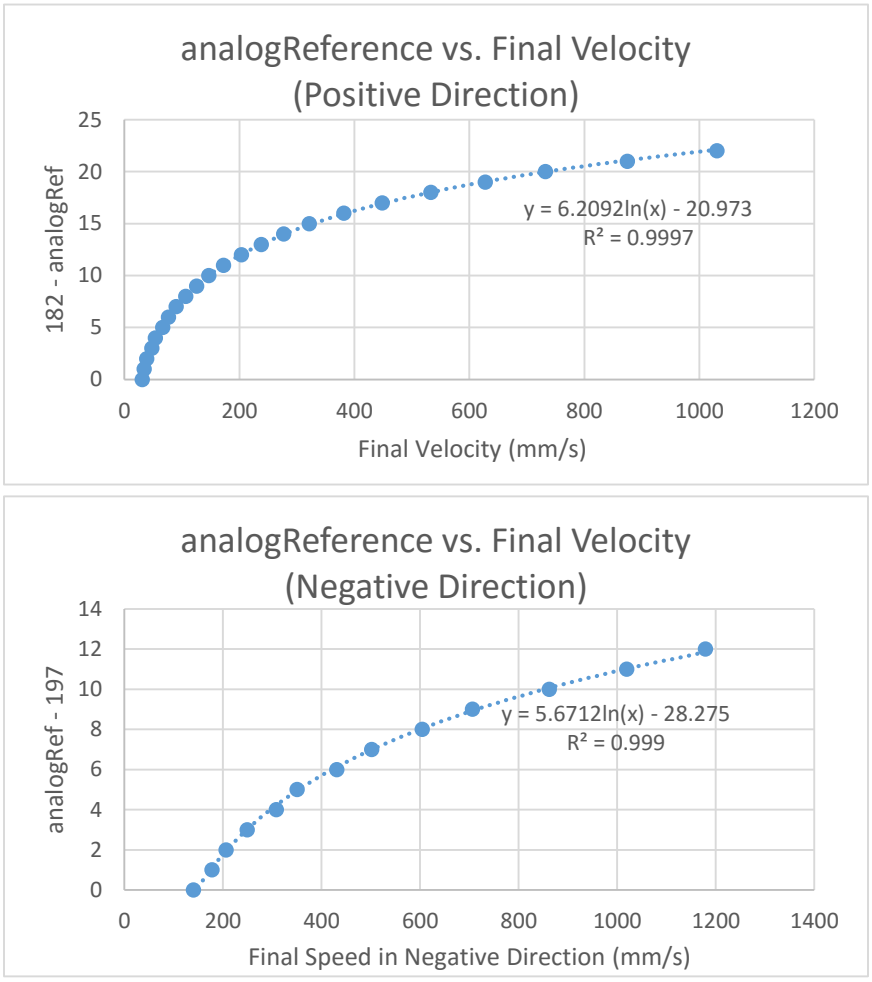


Figure 5-11: AnalogReference vs. Final Velocity of the Linear System in the Positive (Upper) and Negative (Lower) Directions

A logarithmic function gave the closest fit, and a velocity offset was adjusted to make the fit as close as possible. The final relationships between desired velocity v_{des} and analogReference (a_{ref}) are as follows:

For motion in the positive direction:

$$a_{ref} = 182 - (6.2092 * \ln(v_{des} - 70) - 20.973) \quad [5]$$

For motion in the negative direction:

$$a_{ref} = 197 + (5.6712 * \ln(100 - v_{des}) - 28.275) \quad [6]$$

The same procedure was followed to calibrate the rotational motor. The graph below shows the relationship between analogReference value and angular velocity of the rotational motor.

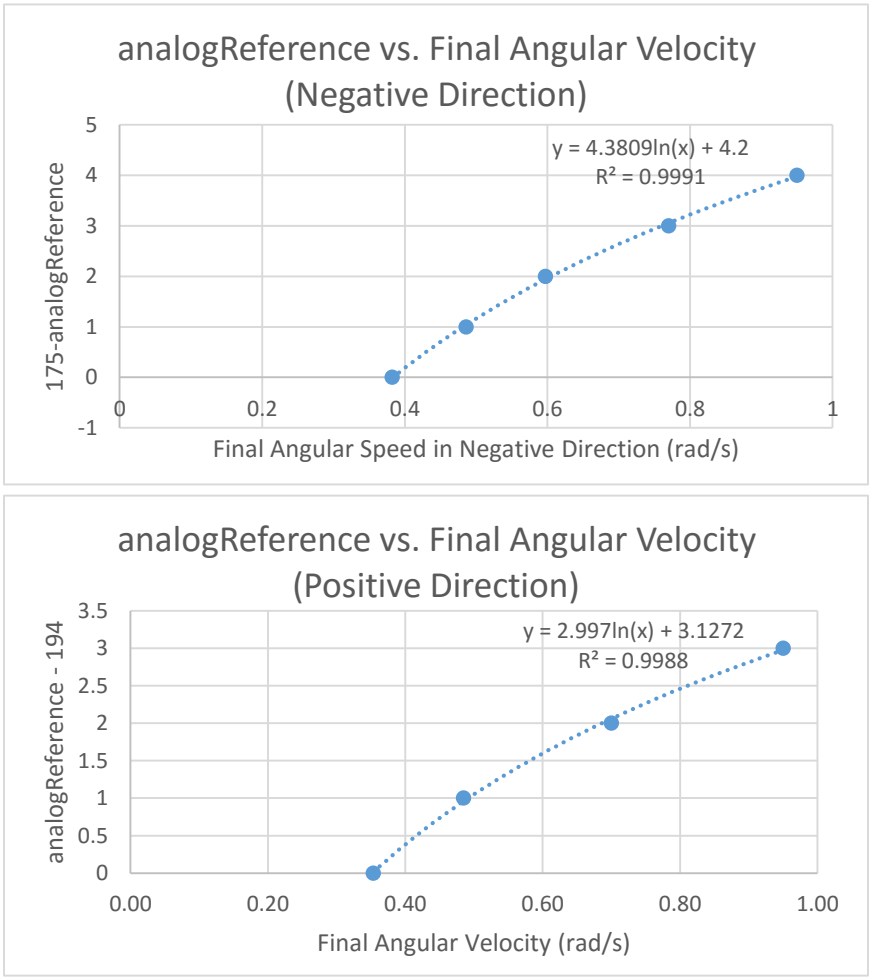


Figure 5-12: AnalogReference vs. Final Velocity of the Rotational System in the Negative (Upper) and Positive (Lower) Directions

The final relationships between desired angular velocity ω_{des} and analogReference (a_{ref}) are as follows:

For motion in the positive direction:

$$a_{ref} = 194 + (2.997 * \ln(\omega_{des} + 0.15) + 3.1272) \quad [7]$$

For motion in the negative direction:

$$a_{ref} = 175 - (4.3809 * \ln(0.25 - \omega_{des}) + 4.2 \quad [8]$$

However, during testing it was clear that the rotational subsystem responded much more slowly to the controller's commands than the linear subsystem. The graph below shows the angular velocity data when the rotational motor was driven with an analogReference value of 171, the lowest value used by the system.

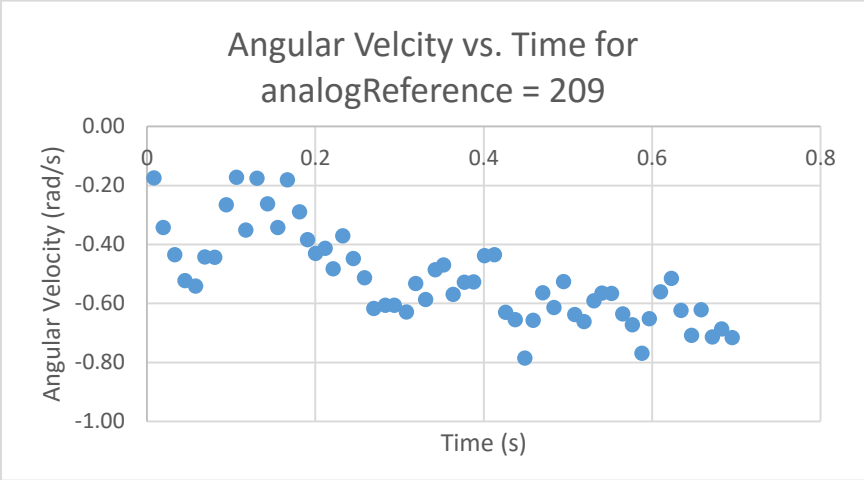


Figure 5-13: Response of Rotational System when Driven with analogReference value of 209

As is seen in the graph, the rotational motor took much longer to reach its final velocity than the linear motor, with settling times of nearly one second. Since the torque needed by the rotational motor was much larger, motor dynamics more visibly came into play as shown by the motor's exponential approach to its final velocity. Thus if the system were simply driven to a

desired angular velocity, the response would be much slower than desired. Instead, the dynamics of the motor were used to drive the rotational system to a desired angular acceleration.

As explained by Rojas [81], a DC motor can be modeled by the equation:

$$\dot{\omega} = \frac{T_s}{J}\left(1 - \frac{\omega}{\omega_f}\right)$$

where T_s is the stall torque, J is moment of inertia of the motor (or in our case of the entire system), ω is the current angular velocity, and ω_f is the final velocity for the applied voltage. The RoboClaw motor controller and the experimentally found relationship between analogReference and final velocity give the system direct control over final velocity. Thus, we want to solve this equation for ω_f. The stall torque, T_s, is also given as $T_s = \omega_f k_e$, allowing the equation to be rewritten:

$$\dot{\omega} = \frac{\omega_f k_e}{J}\left(1 - \frac{\omega}{\omega_f}\right) \quad \rightarrow \quad \dot{\omega} = \frac{k_e}{J}(\omega_f - \omega)$$

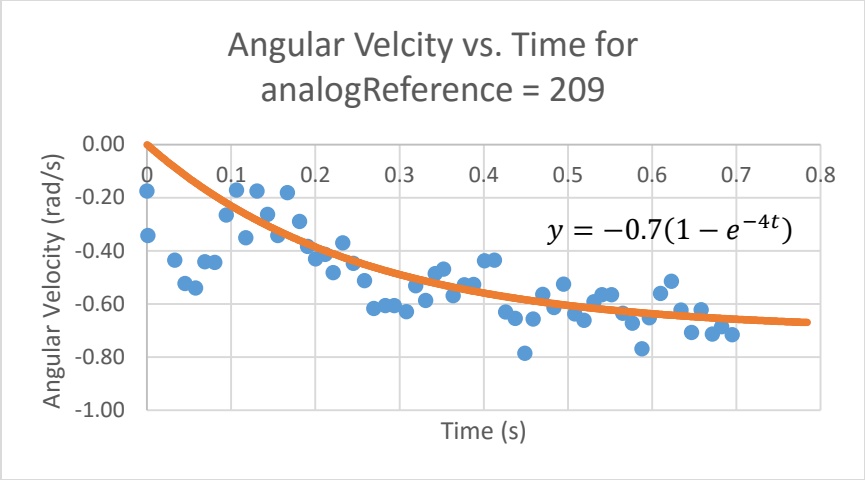
$$\omega_f = \left(\omega + \frac{J}{k_e}\dot{\omega}\right) \qquad [9]$$

Once the ratio $\frac{J}{k_e}$ is determined, if the current angular velocity and the desired angular acceleration are known, the equation above will give the final velocity to which the system should be driven. By creating an artificial desired velocity as described above, the system is able to more accurately track angular velocities, as will be shown in Chapter 6.

To determine $\frac{J}{k_e}$, Equation 9 was integrated and solved for ω to get:

$$\omega = \omega_f\left(1 - e^{\frac{-K_e}{J} * t}\right)$$

An exponential fit was then applied to the angular velocity vs time graphs, and the best estimate of $\frac{k_e}{J}$ was found.



Fgure 5-14: Response of Rotational System when Driven with analogReference value of 209. The Line of Best Fit gives the ratio K_e/J for the System

The ratio $\frac{k_e}{J}$ for the system without an arm attached was found to be roughly 4. However, this value would be expected to decrease when the device is used to move an impaired limb, as the limb will act as an inertial object to the system. In the future this value may be calibrated while the system is operating, as it would likely vary as the user experiences bouts of fatigue or energy. The current system uses the ratio $\frac{k_e}{J}$ as a comfort setting which is manually adjusted to the user's desire. For the results presented in Chapter 6, a value of 2 was used for $\frac{k_e}{J}$ when a user's arm was attached.

The steps described above are used with both force-based and position-based control modes. Once the respective sub-controller outputs a desired velocity, the equations described above are used to convert the desired velocity to pulse durations for each motor, which are then sent to the RoboClaw motor controller to drive each motor at the appropriate voltage.

5.5. Control-based Safety Considerations

In Chapter 2, various mechanical safety features were described. The following section describes safety features implemented in the system's code. To begin, maximum limits are set for velocities in the x, y, r and θ directions. Maximum positions are set for the r and θ directions, and if the end-effector meets or exceeds these positions it is only allowed to be driven inward, away from the boundary. If the potentiometer reads a sudden jump in position, indicating a brief stop in communication with the Arduino, an emergency stop is initiated and both motors will be held motionless until the system is reset. As a final safety, a maximum and minimum allowable pulse-lengths are given, preventing the Arduino from sending commands beyond these values.

CHAPTER 6

SYSTEM EVALUATION

This chapter presents performance data for the complete system. First the system without the eye-tracker will be evaluated and responses to step, ramp and sinusoidal inputs will be given. The system's response to user force inputs will then be shown with data from attempts to drive the end-effector to four points with the controller being held in force mode. Finally the entire system including the eye-tracker will be evaluated. Data will be presented on the effectiveness and efficiency of eye-tracker calibration before a session. Results from sessions in which the user was asked to use eye-gaze to direct the end-effector to a series of four points will then be presented, as means of evaluating the accuracy and speed of the integrated system. Since a priority of the project was to keep the cost at a minimum, a final means of evaluation will be the total cost of the project.

6.1. System Mechanical Responses

To evaluate the performance of the position-based controller, the system responses to various position inputs were recorded. These inputs included step inputs, ramp inputs, and sinusoidal inputs, each of which will be discussed separately as each examines different characteristics of the system. The responses were first measured in the x-direction alone, then the y-direction alone, then a combination of both. This allowed each sub-system to be evaluated individually and as part of the entire system. The responses were observed with loads of 0 lbs (0 N), 2.2 lbs (9.8 N), and 5 lbs (22.2 N) attached to the end-effector. Since the complete position-based controller was used to evaluate system performance, the responses are all subject to the

constraints imposed by the controller, such as maximum velocities and accelerations. These limitations are observable in many of the responses.

6.1.1. Step Response

The response of the system was first measured for step inputs. The following graphs show the system's responses when the (x,y) components of the desired position were switched from (900,0) to (400,0) and vice versa. Since no change in the y position was required, these step inputs isolated the linear subsystem from the rotational subsystem and was able to evaluate linear motion alone.

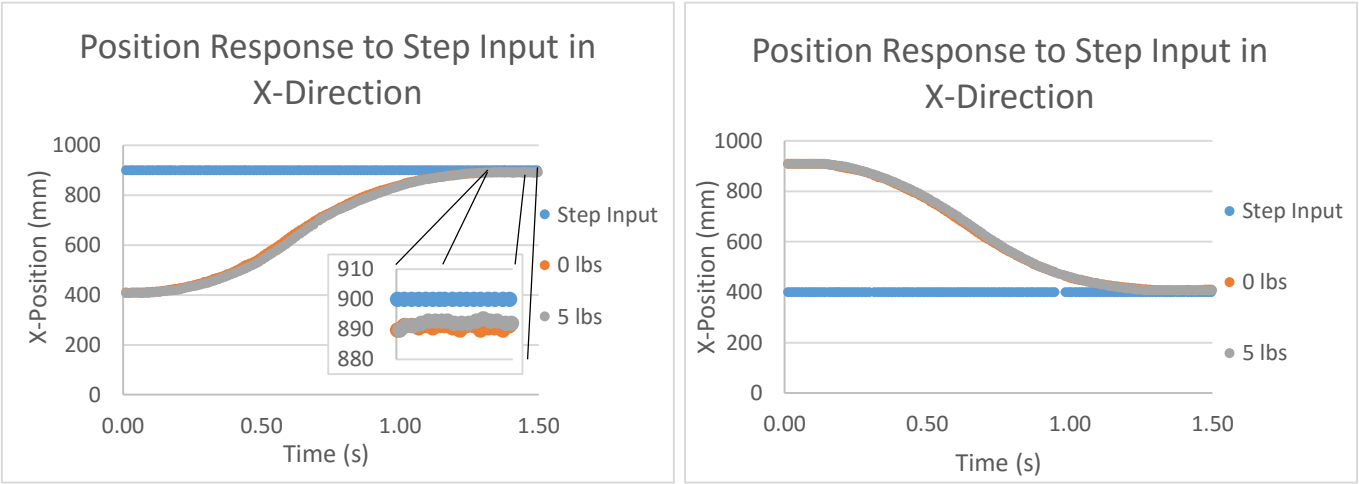


Figure 6-1: Position vs. Time Responses of the Linear System to Step Inputs in X-Direction

The graphs show that the linear subsystem was able to reach a steady-state error of less than 10 mm, and that the response was nearly identical with and without the 5-pound load. Even though the carriage was able to travel the entire range of linear motion in roughly one second, the position vs. time graphs make the response look somewhat slow, particularly immediately following the step input. The graphs below show velocity vs. time data for the same responses.

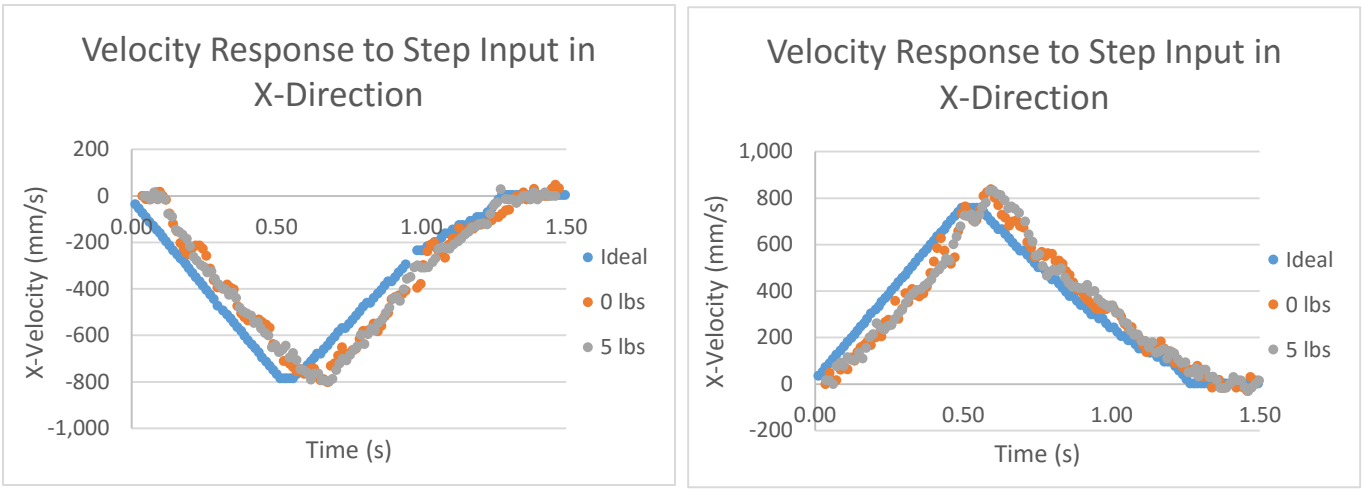


Figure 6-2: Velocity vs. Time Responses of Linear System to Step Inputs in X-Direction

These velocity vs. time graphs help show that the slow start to the step response is due to the acceleration limits placed on the motion by the controller. The true velocity follows the ideal velocity very closely, this ideal velocity is just limited by comfort constraints rather than mechanical constraints.

The below graphs show the system's responses when the (x,y) components of the desired position were switched from (700,300) to (700,0) and vice versa. Again, these points were selected as an attempt to isolate rotational motion from linear motion. While this motion does in fact require some movement along the linear subsystem, since the controller operates in the x-y coordinate system this step input was used as an approximation of rotational movement alone.

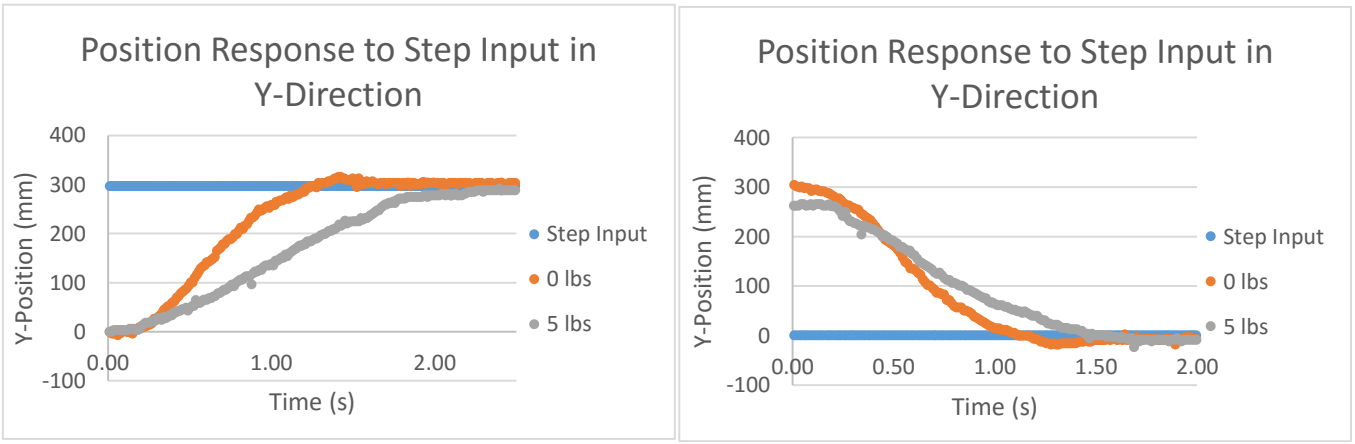


Figure 6-3: Position vs. Time Responses of System to Step Inputs in Y-Direction

It is clear from the response data that additional load has a larger impact on rotational motion than on linear motion. With a load of 5 lbs, the system takes roughly two seconds to travel the range of rotational motion. Although the response to a step input in the y-direction is somewhat slower, the steady-state error is still less than 15 mm, which is essential for the ultimate goal of assisting the user in reaching the desired location. The velocity data of the responses in the y-direction are shown below.

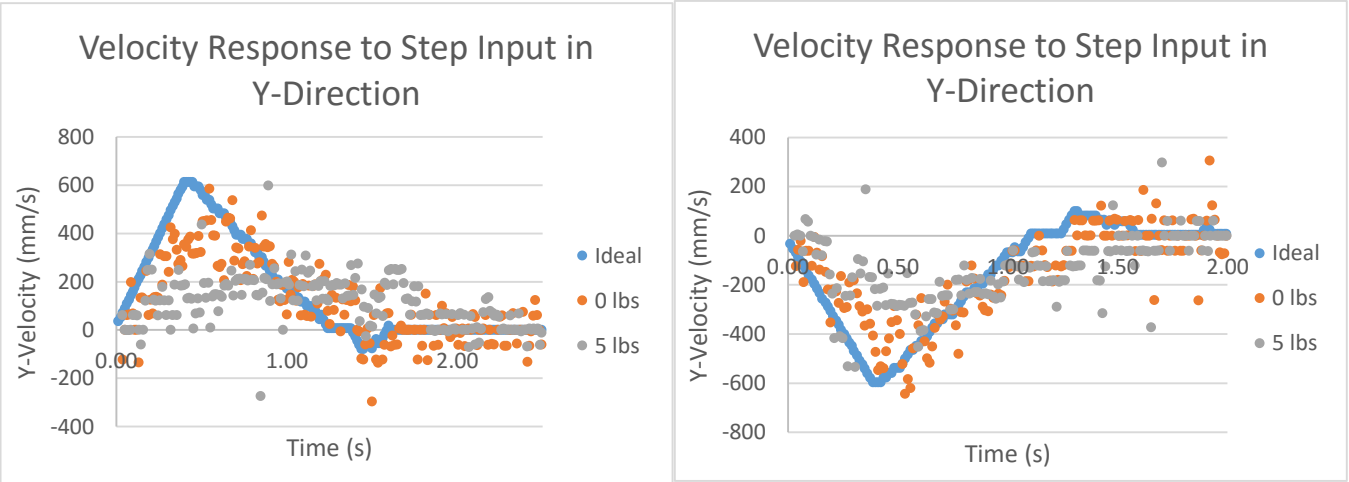


Figure 6-4: Velocity vs. Time Response of System to Step Inputs in Y-Direction

The disorganized data is a result of the relatively low resolution of the Arduino ADC combined with the small change in the rotational potentiometer's voltage. The velocity responses in the y-direction show that the true velocity in the y-direction slightly lags the desired velocity, and with an additional load isn't able to accelerate as quickly as desired. This may be addressed more directly in the future with better motor equations and in-the-loop moment of inertia calculations, but the y-direction step responses show the system is able to reach minimal steady-state error, which is the main concern for the device's purposes.

The following graphs show the system's responses when the (x,y) components of the desired position were switched from (900,0) to (600,250) and vice versa. This step input was used to evaluate the combination of linear and rotational motion.

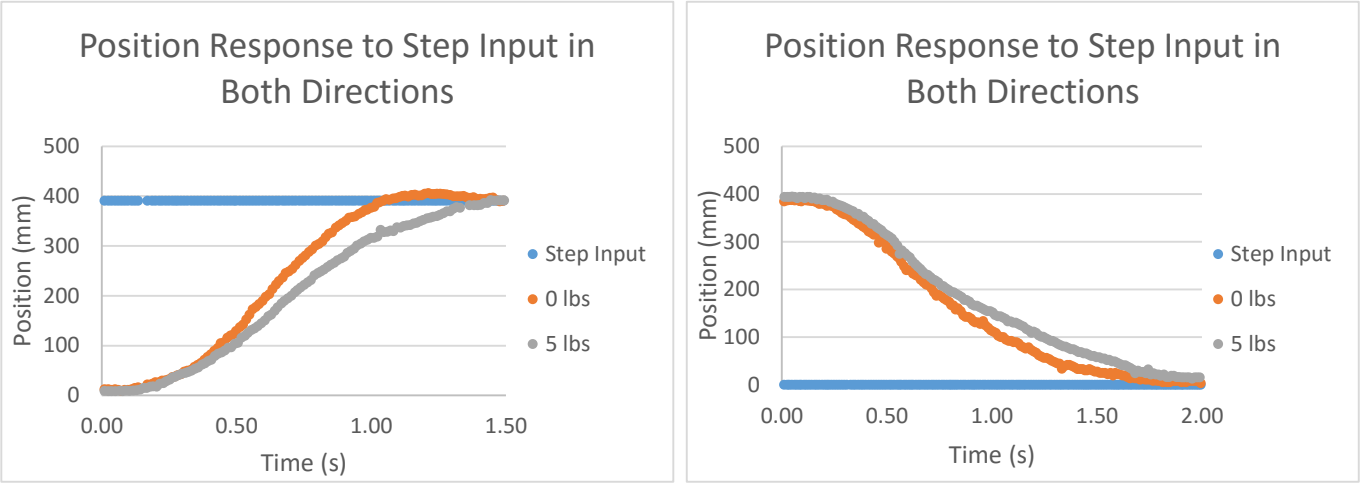


Figure 6-5: Position vs. Time Responses of System to Step Inputs in Both Directions

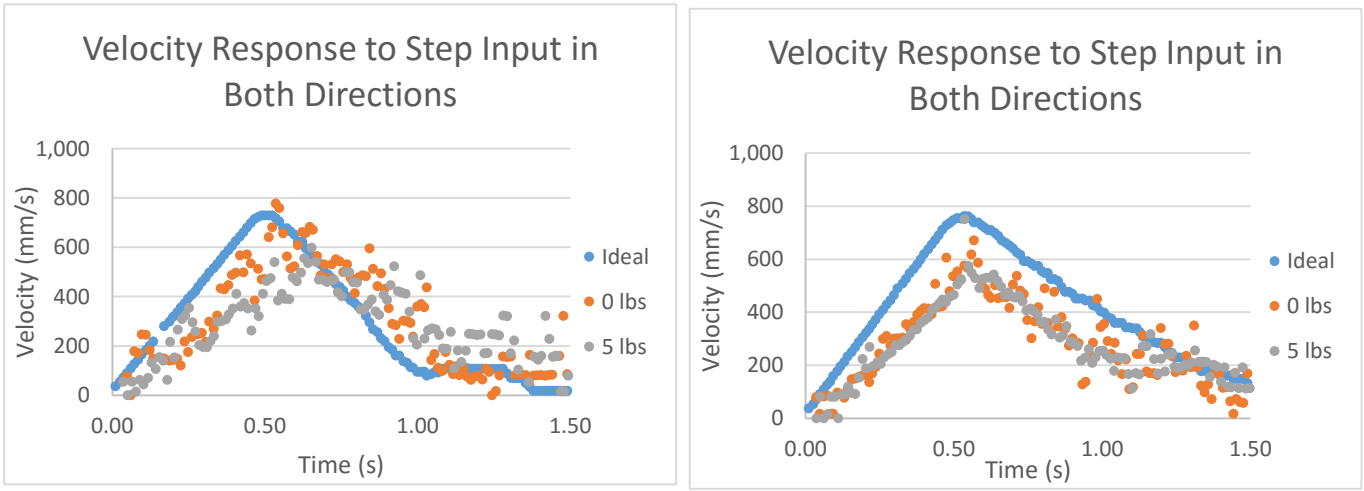


Figure 6-6: Velocity vs. Time Response of System to Step Inputs in Both Directions

The graphs show that the system as a whole reaches a steady-state error of less than 10mm, and that the velocity is fairly well driven to its desired value. The graph below shows the trajectory of the end-effector during the entire system step response.

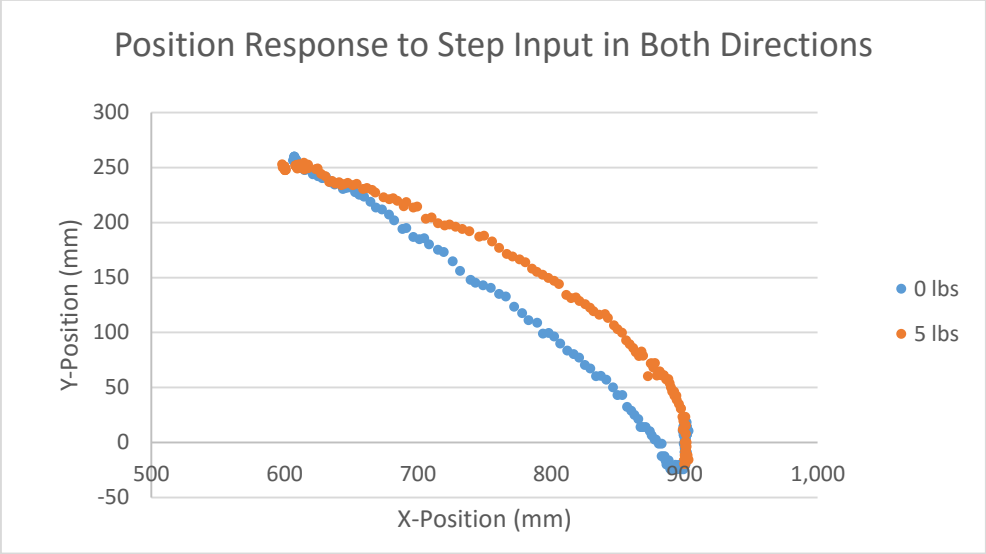


Figure 6-7: Trajectory of End-effector During Response to Step Input in Both Directions

The graphs show a slight delay in rotational motion to linear motion. Although the trajectory does seem to be more disrupted with an additional load, the trajectories on the whole appear to be fairly direct.

6.1.2. Ramp Response

The response of the system was also tested with ramp inputs. These followed the same desired trajectories as the step inputs, but instead of having the desired position jump the entire range of motion, the ramp inputs gradually adjusted the desired position by some designated velocity. While the ramp input simulates a desired velocity, it's important to note that the desired velocity that the controller outputs is a separate value, resulting from calculations explained in Chapter 5, such as proximity to destination. Graphs for both position vs. time and velocity vs. time will be shown, to better demonstrate the system's ability to track the controller's desired velocity vs. the input desired velocity. Since the step input evaluations showed the responses were similar regardless of direction, the ramp responses will only be

shown in one direction for each of the three ramp input types. Ramp inputs were given at 100 mm/s, 200 mm/s, 500 mm/s, 1000 mm/s and 1500 mm/s. These results will only discuss the responses to ramp inputs of 200 mm/s and 1000 mm/s, as these are the best indicators of the range of inputs the system could be expected to receive.

The following graphs show the system's responses when the (x,y) components of the desired position were switched from (400,0) to (900,0) at a rate of 200 mm/s.

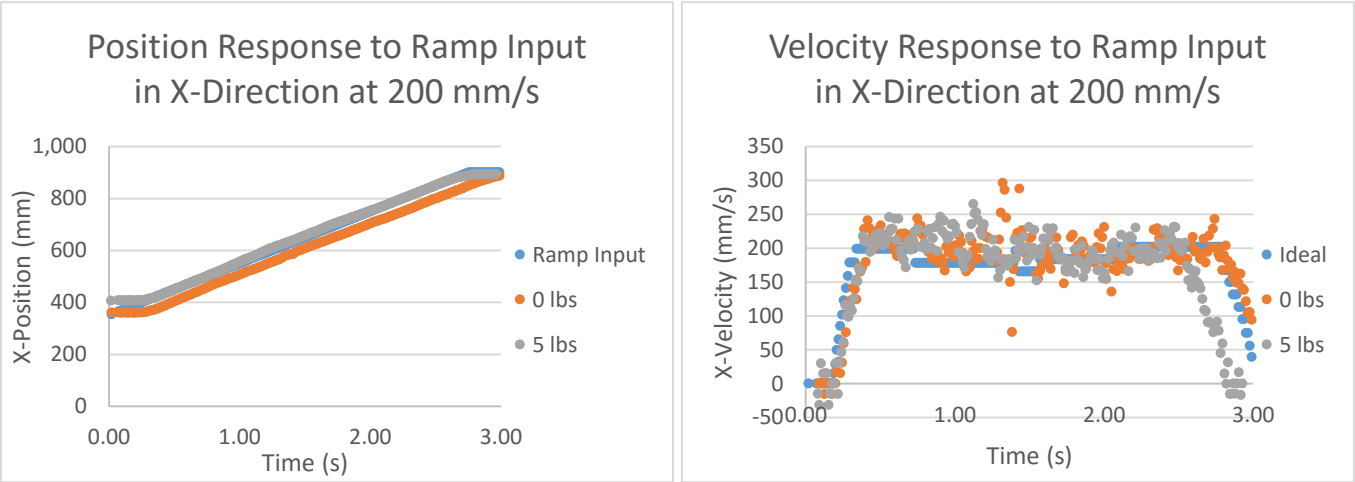


Figure 6-8: Position vs. Time Response (Left) and Velocity vs. Time Response (Right) of System to Ramp Input of 200mm/s in X-Direction

The graphs show the system is quickly able to reach a velocity of 200 mm/s and maintain a constant velocity until commanded to stop. Even though the position vs. time graphs shows no elimination of error as time continues due to the absence of an integrating term in the controller, the graph also shows no indefinite growth of error as time continues, which shows the linear sub-system is able to track ramp inputs without diverging from the desired velocity.

The graphs below show the same trajectories as the previous graphs, but at a greater velocity. These are the system's responses when the (x,y) components of the desired position were switched from (400,0) to (900,0) at a rate of 1000 mm/s.

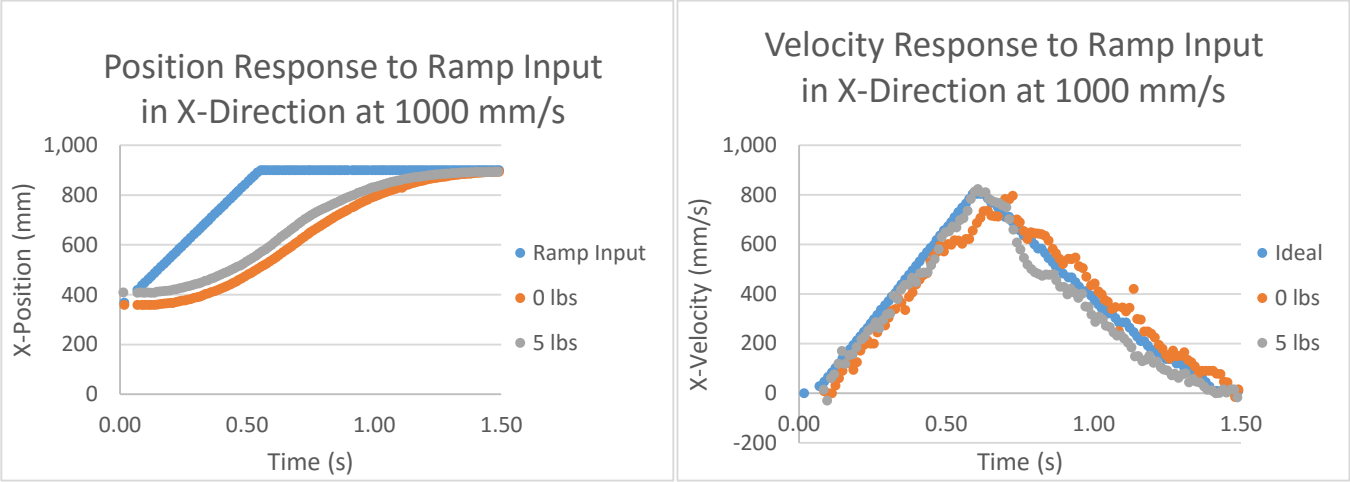


Figure 6-9: Position vs. Time Response (Left) and Velocity vs. Time Response (Right) of System to Ramp Input of 1000mm/s in X-Direction

While these graphs show much greater deviation from the desired position and the actual position, the velocity graph shows this is yet again due to acceleration constraints, not mechanical limitations of the system. The range of motion isn't large enough for the system to accelerate to the full 1000 mm/s, but the system was able to follow the desired velocity accurately up to 800 mm/s, with and without a 5 lb load.

The graphs on the following page show the system's responses when the (x,y) components of the desired position were switched from (700,0) to (700,300) at a rate of 200 mm/s. Similar to the step responses, the load on the end-effector has a larger effect on rotational motion than on linear motion. The steady-state error appears to be contained, although there is some lag between desired velocity and actual velocity.

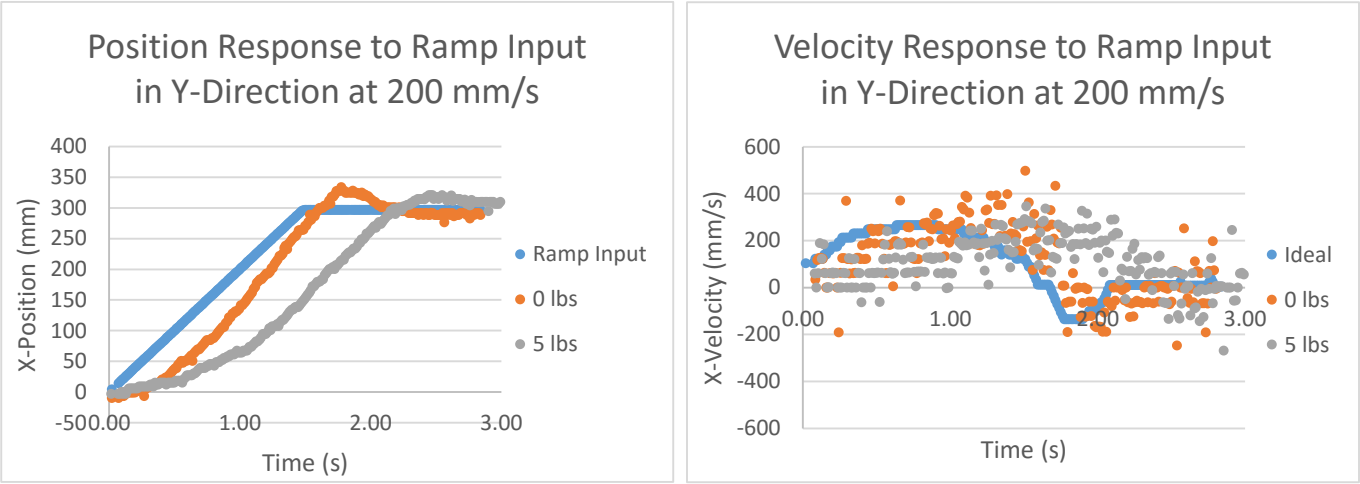


Figure 6-10: Position vs. Time Response (Left) and Velocity vs. Time Response (Right) of System to Ramp Input of 200mm/s in Y-Direction

The graphs below show the same trajectories, but at a greater velocity. These are the system's responses when the (x,y) components of the desired position were switched from (700,0) to (700,300) at a rate of 1000 mm/s.

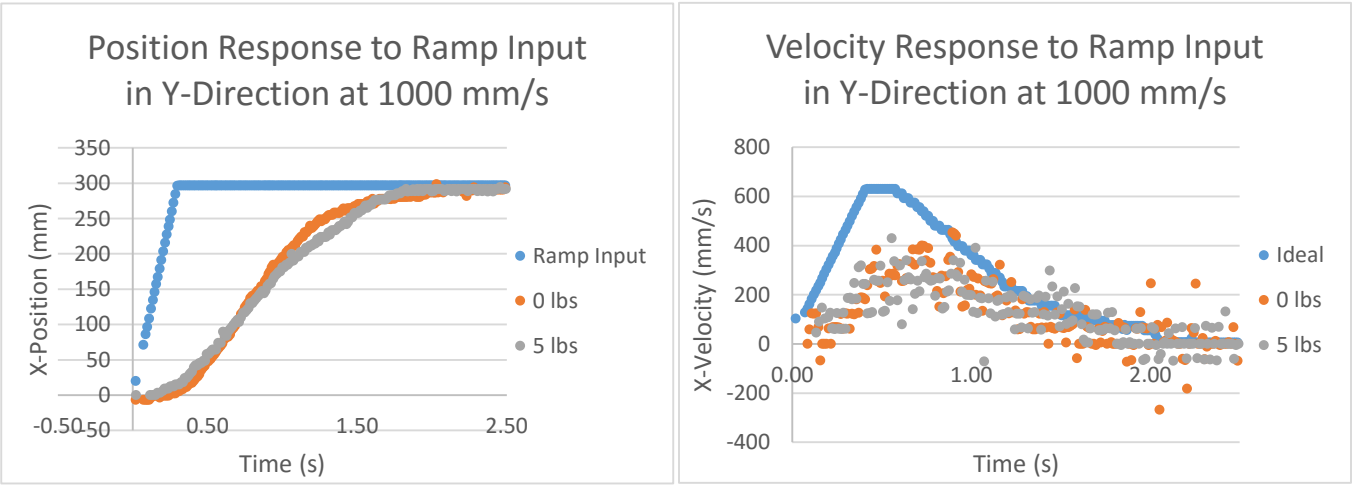


Figure 6-11: Position vs. Time Response (Left) and Velocity vs. Time Response (Right) of System to Ramp Input of 1000mm/s in Y-Direction

The range of rotational motion was not large enough for the system to reach its desired velocity, but the true velocity with and without a 5lb load continues to increase until the command to slow down is given. The steady-state error for high velocity in the y-direction was undetermined.

The following set of four graphs give the entire system response to a ramp input requiring both linear and rotational movement with a load of 0lbs and 5lbs. These are the system's responses when the (x,y) components of the desired position were switched from (900,0) to (650,200) at a rates of 200 mm/s and 1000 mm/s.

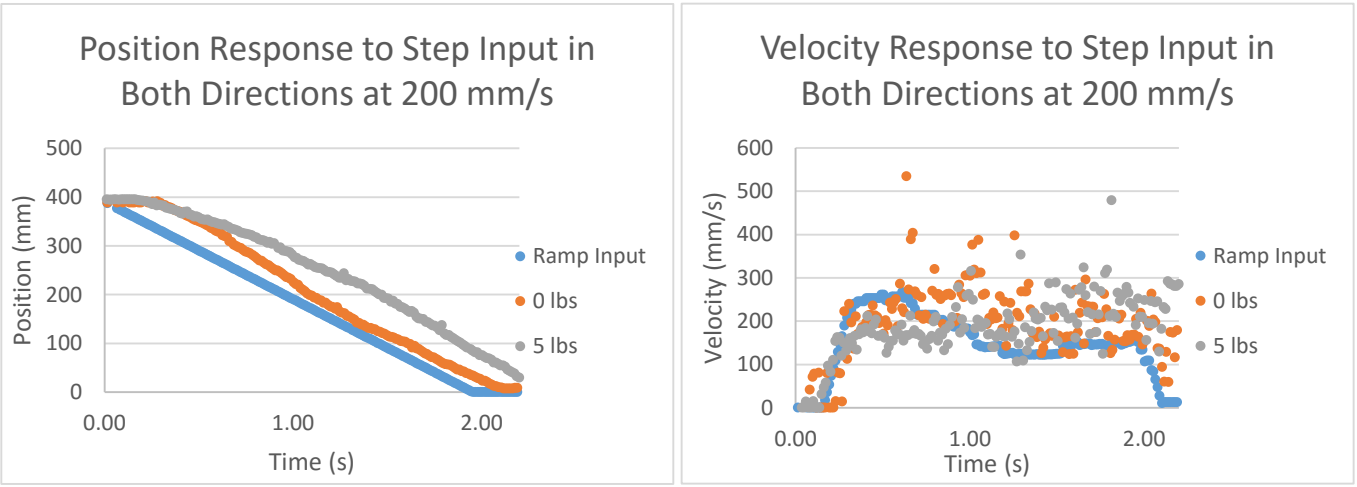


Figure 6-12: Position vs. Time Response (Left) and Velocity vs. Time Response (Right) of System to Ramp Input of 200mm/s in Both Directions

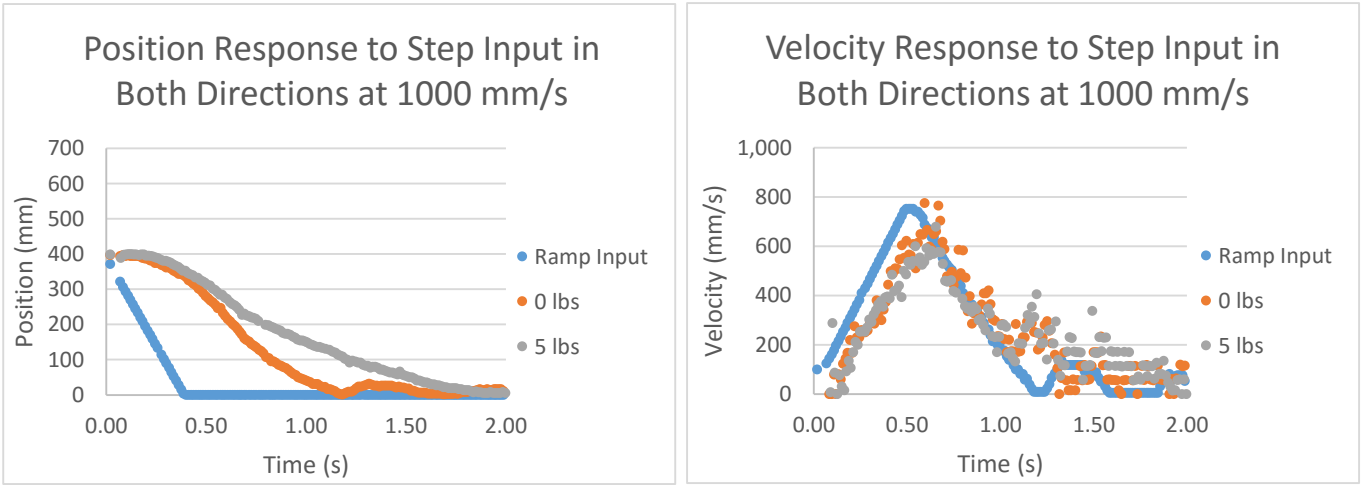


Figure 6-13: Position vs. Time Response (Left) and Velocity vs. Time Response (Right) of System to Ramp Input of 1000mm/s in Both Directions

These graphs reinforce the results from the unidirectional ramp inputs. The steady-state error to a ramp input appears to be contained, although is greater for higher velocities due to the

acceleration constraints of the controller. The graphs below show the trajectory of the end-effector during these responses.

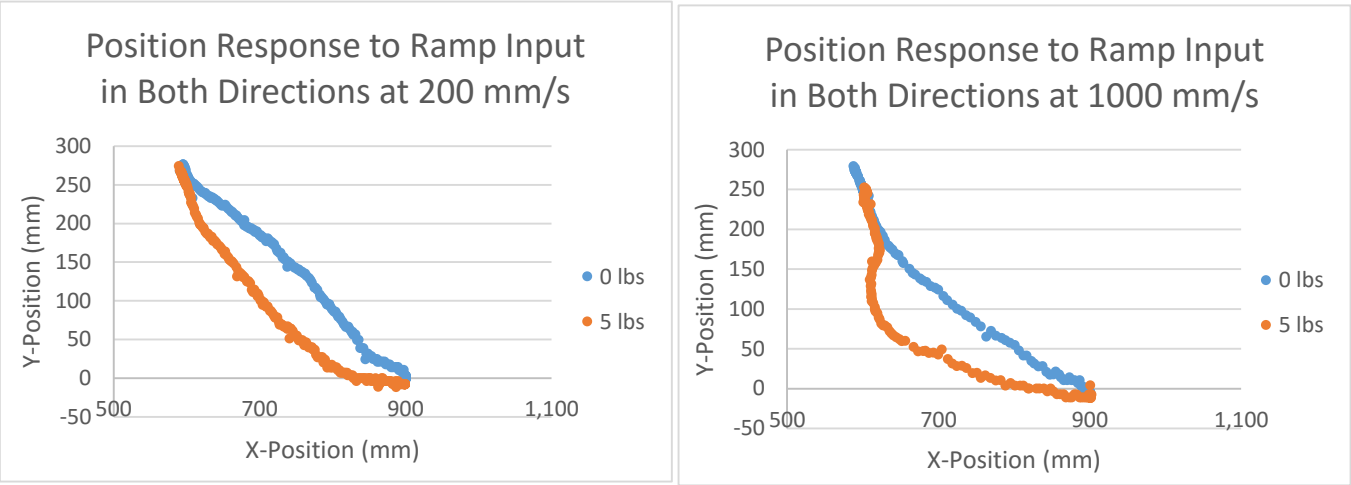


Figure 6-14: Trajectory of End-effector During Response to Ramp Input in Both Directions at 200 mm/s (Left) and 1000 mm/s (Right)

The end-effector takes a trajectory that is slightly less linear with increased load and increased speed, but even after the skewed initial trajectory shown in the second graph, the system is able to correct itself as motion continues.

6.1.3. Frequency Response

The system's frequency response was evaluated by setting the desired position to a sinusoidal trajectory with varying frequencies. First an example response will be given and then the overall Bode diagrams will be presented. The graphs on the following page show the system's response to an x-directional sinusoidal input which covered the entire range of linear motion at 0.2 Hz. The graphs show little difference in magnitude or phase between input and response, regardless of the mass of the additional load. Thus the system is very-well able to track x-direction sinusoidal inputs at frequencies around 0.2 Hz.

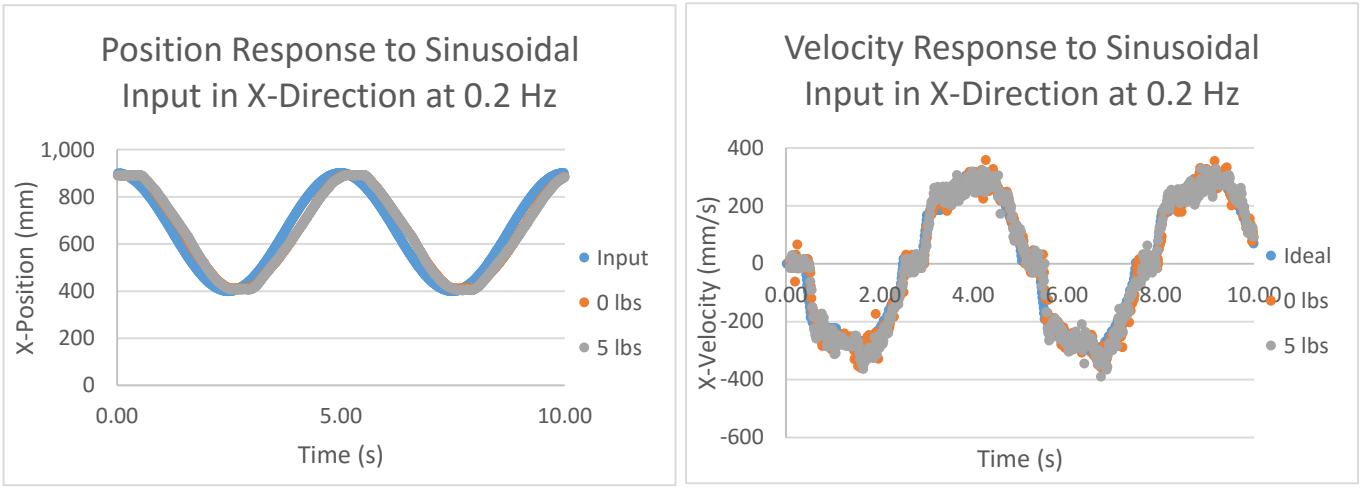


Figure 6-15: Position vs. Time Response (Left) and Velocity vs. Time Response (Right) of System to Sinusoidal Input of 0.2 Hz in X-Direction

The following graphs show the response at a higher frequency of 1 Hz. Again, the sinusoidal input was in the x-direction only.

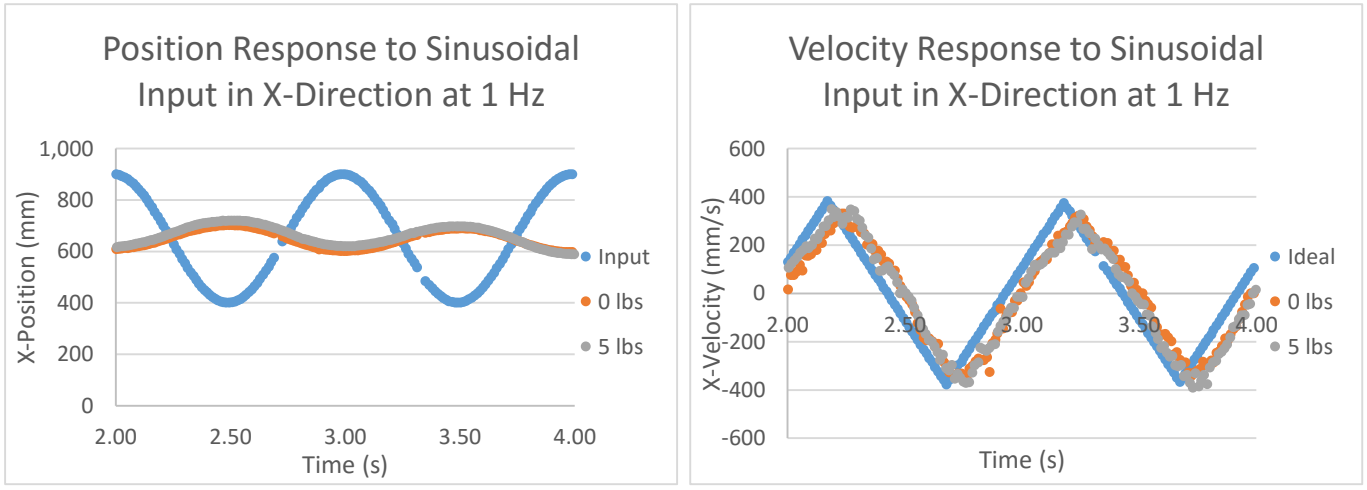


Figure 6-16: Position vs. Time Response (Left) and Velocity vs. Time Response (Right) of System to Sinusoidal Input of 1 Hz in X-Direction

These graphs show a response magnitude which is significantly reduced from the input magnitude, as well as a phase shift of roughly 180 degrees. The velocity graph shows that the phase shift and decrease in magnitude is largely due to the acceleration limits of the system, and that the system accurately tracked the controller's desired velocity output.

Similar sinusoidal inputs were tested at frequencies of 0.1 Hz, 0.2 Hz, 0.4 Hz, 0.6 Hz, 0.8 Hz, 1 Hz, and 1.5 Hz. Separate sets of data were collected for x-direction inputs, y-directions inputs, and inputs that required motion in both the x and y directions. All responses were tested with no load and with a 5lb load. The following Bode plots show the magnitude and phase characteristics of the responses.

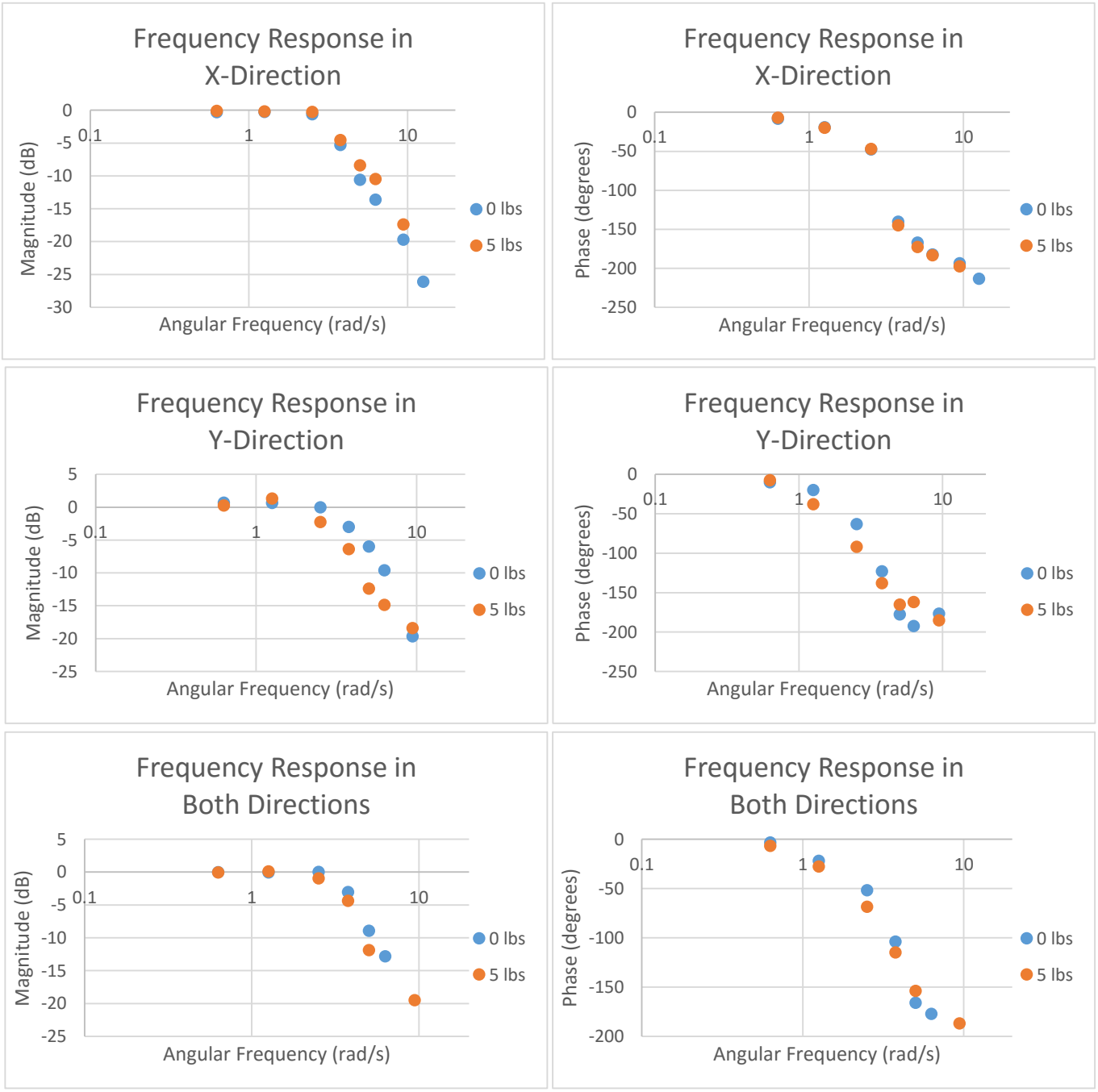


Figure 6-17: Bode Plots of System Responses to Sinusoidal Inputs in X-Direction (Top), Y-Direction (Middle), and Both Directions (Bottom)

The Bode plots show the system is able to track sinusoidal inputs with frequencies up to 1 rad/s with little change in phase or magnitude. All phase diagrams cross -180 degrees at between 5 rad/s and 10 rad/s, corresponding to response magnitudes of between -10 and -20 dB. The gain margin can be safely estimated at 10dB, supporting the explanation of the system's stability presented in Chapter 5. The Bode plots show no distinguishable difference between a load of 0lbs and a load of 5lbs.

6.2. System Response to Force Inputs

While the force-mode is not meant to be used frequently, it is important that the system moves with the user if the user has the strength and desire to move without assistance. To evaluate the force-based controller alone, the system was held in force mode and a user was asked to move the end-effector to four designated locations using the wrist interface. The system ignored eye-tracker data for these tests, and instead motion was based purely on force inputs registered by the FSRs. The user was asked to move the end-effector along a crossing pattern that was broken into four legs. The system began at the (x,y) coordinate (900,0) and the user was asked to move first to (600,200), then (900,200), then (600,0), and then return to (900,0). The graph below shows an example trajectory of this evaluation.

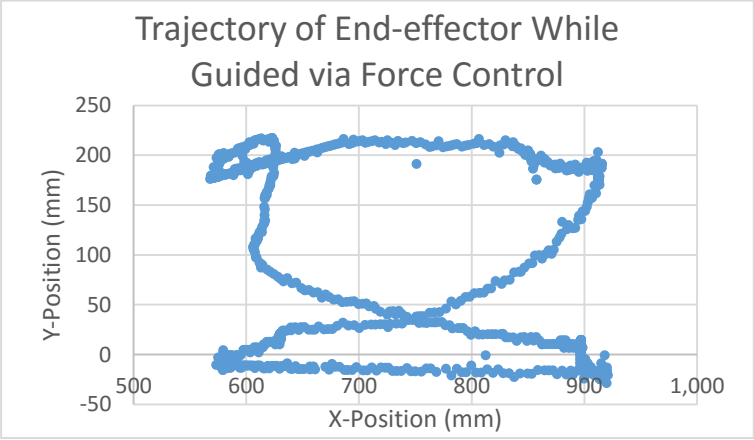


Figure 6-18: Trajectory of End-Effector with System in Force mode and Being Driven by Force to the Four Designated Points

The data presented here is based on two individuals performing the entire motion twice, giving a total of 16 force-controlled motions. To evaluate the motion, the data for the entire movement was broken into four legs based on when the user's motion came to a rest for each leg. In the above graph, it's difficult to tell where the user finished each leg of motion since time is not represented on either axis. The graph below shows the extracted begin and end points for the motion depicted in the above graph, which better shows the overall accuracy of the force-based controller.

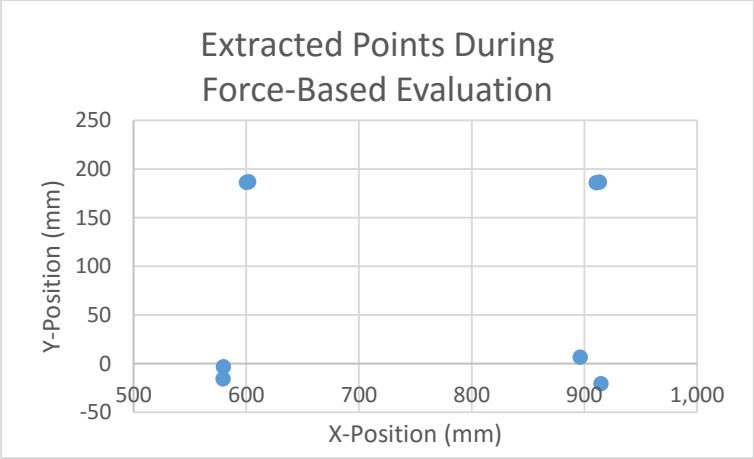


Figure 6-19: Extracted Begin and End Points for Example Motion During Force-Based Evaluation

Once beginning and ending points were extracted from all 16 force-driven motions, the distance between the final resting point and the desired destination and the total time required were calculated for each motion. One of the 16 motions triggered an emergency stop described in Chapter 5 (due to an outlier potentiometer reading). This motion was excluded from analysis. The calculations from the remaining 15 motions showed that the average position error was 31mm and that the average length of time from initial force input to reaching a steady final position was 3.08 seconds.

6.3. Eye-Tracker Evaluation

When the controller in in position mode, the desired position of the system is calculated from eye-tracker data. To evaluate the mechanical responses of the system, the desired position was taken directly from the input, and thus known to be accurate. For the complete system, the accuracy of the calculation of desired position from eye-gaze data must be verified.

For the following evaluation, the eye-tracker was calibrated to the user before each set of data was collected. While using saved calibration data was found to be an effective, time-saving alternative, the process of calibrating the eye-tracker and then collecting data was used as a means of evaluating the calibration as well. Calibration consisted of having the user look at four pre-determined points marking the boundary of allowable motion until a certain number of successful eye-gaze frames were collected. The carriage followed along with the calibration by traveling to the pre-determined locations, allowing the user to train the eye-tracker to associate the current gaze with the carriage's current position. If too few of the collected eye-gaze frames had valid data, the user was prompted to look at the same point and repeat the process. The most time-consuming part of the process was the carriage movement and the sequence of beeps which act as instructions for the user; the process of collecting frames for a given point took under two seconds. After running ten calibrations sets, the average time for calibration was 51.25s.

Following calibration, a trial consisted of having the user direct the end-effector to the same positions from the force-mode evaluation. The user was asked to use only eye-gaze to direct the system to (x,y) positions (600,200,), (900,200), (600,0), and then return to (900,0). These trials were performed with the user's right wrist attached to the wrist-interface. The force-sensors were operational, but the user was asked to relax the arm attached to the end-effector, in

order to simulate a physically impaired limb. Since this is a preliminary evaluation to show the potential of the system, a total of ten trials were performed and analyzed.

The graph below shows an example x-y position graph of the user's eye-gaze, the desired position output by the controller, and the end-effector's real position for one trial.

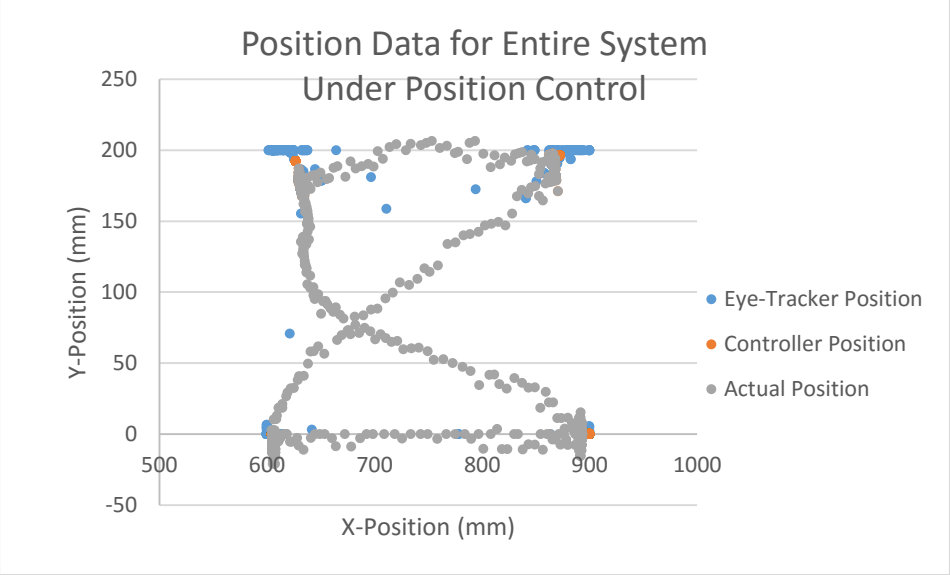


Figure 6-20: Three Data Series showing the Eye-Tracker Position, the Desired Position output by the Controller, and the Measured End-Effector Position during an Eye-Gaze Controlled Trial

For each trial, the key data points were separated from the others to perform mathematical analysis. For each coordinate there were three key data points. The first was collected when the eye-tracker first collected enough successful frames to update the eye-gaze to the new coordinate. The second data point was when the end-effector reached a steady position or when force input was received to override eye-tracker data. The third point was the data collected immediately before the eye-tracker successfully tracked the next coordinate. This final point allowed evaluation of the amount of drift associated with unintentional force input once the destination was reached. The graph on the following page shows the key data points from the above trial.

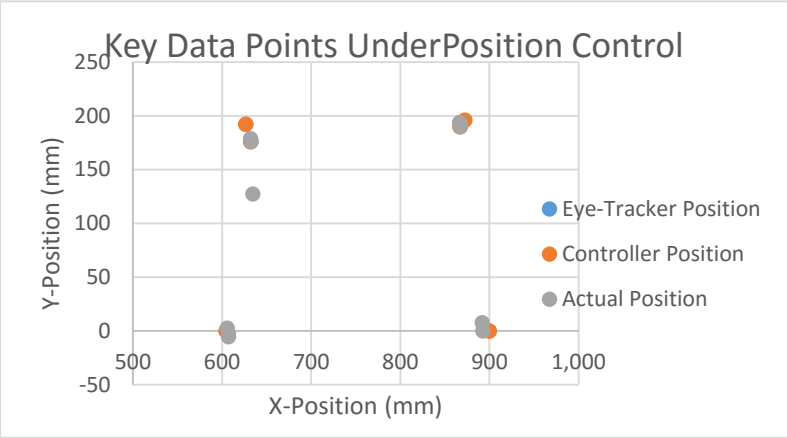


Figure 6-21: Key Data Points Extracted from the previously shown Eye-Gaze Controlled Trial

A total of ten trials were performed, resulting in 40 individual motions. The overall average distance between the eye-tracker output position and the true desired position was 15.98mm. The eye-tracker was slightly less accurate for the farther away gazes than the closer gazes. The individual motions to points (600,200) and (900,200) had an average eye-gaze error of 23.42mm, while motions to points (600,0) and (900,0) had an average eye-gaze error of only 8.53mm.

6.4. System Response to Combined Force and Eye-Tracker Inputs

The data and graphs shown in the previous section were also used to evaluate the entire system response to eye-tracker inputs. While the previous section explores the accuracy of the eye-tracker, this section discusses the accuracy of the end-effector when driven by the eye-tracker. The end-effector is expected to be somewhat less accurate than seen in the mechanical response section, since the load of a real human arm is often more resistive than a dead weight. As the end-effector moves the user's wrist away from the body, more and more weight from the arm resists the movement, which can even apply a force in the opposite direction due to gravity. Evaluating the end-effector's final position is somewhat difficult, since force inputs were

allowed to alter the desired position. Even though the user did not intentionally apply forces, the FSRs occasionally registered forces from inertial resistance described above.

In an attempt to best evaluate the system, the position error of the end-effector was calculated by three different methods. Method 1 measured the distance between the eye-tracker's desired location and the end-effector's location when a steady position was reached or an intentional force was registered. Method 2 measured the distance between the true desired location and the end-effector's location when a steady position was reached or an intentional force was registered. Method 3 measured the distance between the true desired location and the end-effector's position immediately before the eye-tracker output a new desired position. This final measurement was used to determine if there was significant drift due to unintentional forces. The table below shows the results from all ten trials. The "Back Coordinates" are (600,200) and (900,200), and the "Front Coordinates" are (600,0) and (900,0).

Table 6-1: Average Errors for System under Normal Operation as measured by Three Different Methods

	Distance (mm)		
	Method 1	Method 2	Method 3
Complete Average	20.81	31.63	31.07
Back Coordinates	32.12	49.46	40.67
Front Coordinates	9.51	13.80	11.88

Again, the system has a smaller position error for the front coordinates than the back coordinates, both in eye-gaze accuracy and in the accuracy of the end-effector's motion. The larger errors among the back coordinates are likely due to difficulty in reaching the back left position, as shown by the outlier in the graph of key data points. Since this location was the farthest from the user's free-hanging arm, the arm offered the most natural resistance to being driven to this location. No average distance was greater than 5cm, and if the back left location is

excluded this drops to less than 3cm. The similarity of the distances calculated by Method 2 and Method 3 show that unintentional force inputs did not play a large role in altering the desired positions, and thus were effectively ignored by the controller. Overall, the system was shown to be reasonably accurate, with a maximum position error of 50mm and an average position error of just over 30mm when driving a relaxed arm.

6.5. System Cost

Table 6-2: Cost of Components for Entire System

Quantity	Part Number	Description	Supplier	Unit Cost	Total Cost
Mechancal Construction					
1	8589K84	1/4" Thick, 24" x 48" Acrylic	McMaster-Carr	$55.76	$55.76
1	9293K56	5lb Constant-Force Spring	McMaster-Carr	$7.85	$7.85
1	8701K45	UHMW Polyethylene Rod - 1" Diameter x 1' Length	McMaster-Carr	$3.08	$3.08
1	91259A102	1/4" x 1-3/4" Shoulder Screw, 10-24 Thread	McMaster-Carr	$1.53	$1.53
1	93070A121	M5 x 10mm Socket Head Cap Screws (Qty: 50)	McMaster-Carr	$9.40	$9.40
1	90327A126	M5 x 12mm Socket Head Cap Screws (Qty: 50)	McMaster-Carr	$6.33	$6.33
1	9059A012	M5 Plain Steel Hex Nut (Qty: 100)	McMaster-Carr	$1.70	$1.70
4	5674K1	Flange-Mount Ball Transfer	McMaster-Carr	$3.07	$12.28
20	5537T454	Steel End-Feed Fastener, M5 (Pack of 4)	McMaster-Carr	$2.55	$51.00
2	5537T101	Aluminium T-Slotted Framing Extrusion (8ft)	McMaster-Carr	$21.58	$43.16
2	92510A357	Aluminum Unthreaded Spacer, 3/8" Screw Size	McMaster-Carr	$2.32	$4.64
1	5912K5	Self-Lubricating Bronze Bearing, 1/2"	McMaster-Carr	$12.28	$12.28
1	6086K111	Quick-Disconnect Bushing, 1/2" Bore	McMaster-Carr	$12.24	$12.24
1	5972K326	Steel Ball Bearing, Double Shielded, 10mm Diameter	McMaster-Carr	$4.80	$4.80
1	161659	2" x 4" x 10 ft Lumber	Home Depot	$4.09	$4.09
1	B00R575B46	80/20 Aluminum Corner Bracket w/Tabs (Qty: 25)	Amazon	$18.50	$18.50
2	B009YSHYTE	6061 Aluminum Sheet, 1/4" x 8" x 8"	Amazon	$13.23	$26.46
1	B001AXF31M	50-lb Fishing Line	Amazon	$2.10	$2.10
1	B006AR54TY	Futuro Sport Wrap Around Wrist Support	Amazon	$6.99	$6.99
1	B0035FZTSO	Steel Key Stock, 1/8" x 1/8"	Amazon	$3.65	$3.65
1	TBI-TR20N-0900-30-30	900 mm Linear Guide Rail	Anaheim Automation	$78.00	$78.00
1	TBI-TRS20VN-N-Z0	Carriage	Anaheim Automation	$35.00	$35.00
Electronic Compontents					
1	B00ITELF12	Battery Disconnect Cut Off	Amazon	$8.62	$8.62
1	B00INVF468	Black 10-Gauge Wire	Amazon	$10.28	$10.28
1	B000K7GRCI	Solderless Wire Terminal and Connection Kit	Amazon	$12.56	$12.56
1	B00NNDAFW4	EBL AAA Charger w/ 8 AAA Batteries	Amazon	$15.99	$15.99
1	B00829IN36	3 x 1.5 V AAA Battery Holder	Amazon	$3.40	$3.40
1	B00H8T6J3S	1 x 1.5 V AAA Battery Holder	Amazon	$3.57	$3.57
1	HR1290W	CSB High Rate AGM Battery	AtBatt	$34.99	$34.99
1	LC1-12-3A	Leoch 12V/3A SLA Battery Charger	AtBatt	$26.99	$26.99
1	AM-2618	Sting Potentiometer Kit	AndyMark	$17.00	$17.00
1	FIRST CIM Motor	FIRST CIM Motor	BaneBots	$28.00	$28.00
1	P80 Gearbox	Planetary P80 CIM Gearbox, 64:1	BaneBots	$143.25	$143.25
1	MY1016	United 250W 24V DC Motor	Motion Dyanamics	$45.95	$45.95
1	1499	RoboClaw 2x60A Motor Controller	Pololu	$199.95	$199.95
4	2728	Force-Sensing Resistor: 0.6" Diameter, Short Tail	Pololu	$5.80	$23.20
1	N/A	Tobii EyeX Eye Tracker and Development Kit	Tobii	$139.00	$139.00
1	N/A	Arduino Uno Rev3	Arduino	$24.95	$24.95
2	WRL-10414	Xbee 2mW Wireless Antenna Series 2	Sparkfun	$22.95	$45.90
			Total:	**$**	**1,184.44**

The total cost to rebuild the system is roughly $1,200. This includes the cost of components retrieved from past projects, such as the MY1016 motor, but does not include the cost of some smaller standard components, such as wiring and a few miscellaneous bolts and nuts.

CHAPTER 7

CONTRIBUTIONS AND FUTURE WORK

This thesis has introduced a low-cost robotic device capable of determining a user's intention from eye-gaze data and force input, and using such information to assist the movement of an impaired upper limb. While the past decade has seen robotics increasingly utilized to assist those with physical impairments, the applications thus far have been restricted to rehabilitative environments, where researchers and robotic systems determine trajectories of motion, not the individual with the physical impairment. These systems have proven to be wonderfully beneficial, yet the simple truth remains that millions of people continue to live with physically impaired limbs. The project presented in this thesis aims to allow robotics to reach a new level of assistance, in which the ultimate goal is to give the user himself control of an impaired limb, even in the face of ongoing or unsuccessful rehabilitation. Furthermore, the control presented is intuitive. The control scheme described in this thesis combines real-time force inputs and eye-gaze data in a way that requires no training, no detailed explanation, and no physical abilities. If the user has the physical strength to move to a location, the system adapts. If the user prefers instead to use eye-gaze, the system adapts. Such a scheme is absolutely essential if assistive robots are to reach their full potential in society. This thesis presents a control scheme which can be easily adapted to fit a wide array of human-robot interaction applications.

A secondary accomplishment of the system presented is its potential accessibility. All portions of the design, from mechanical to electronic, take accessibility into consideration. A design for the first prototype was developed based on polar coordinates, in order to keep the bulk of the system's mass in a single, stationary position. The portion of the system responsible for linear motion included a passive constant-force spring pulling against a DC motor, which kept

the cost and inertia to a minimum while still being able to create force and motion in two directions. A wrist-interface made of laser-cut materials and four low-cost force-sensing resistors was designed to decouple applied moments and linear forces. The electronics used in the system are consistent with the goal of keeping the system low-cost. From the Arduino communication hub to a commercially available Tobii EyeX eye-tracker, all electronic components were selected to be as cost-effective as possible. The low-cost nature of the system presented in this thesis, shown by the total cost of just over $1,000, suggests assistive robotics can be made accessible to the people who need them most, including the 26% of stroke survivors who are dependent in activities of daily living [5]. This thesis aims to take a step toward making that goal a reality.

7.1. Future Work

The work presented in this thesis provides a foundation for future development of an intention-detecting assistive robotic device to help individuals with physically impaired limbs. While this thesis has demonstrated the system's promise with a first prototype, there are various improvements which are necessary in order for the system to reach its full potential, including expansion to a 3D workspace, increased eye-tracker data, and improved controller robustness.

7.1.1. Mechanical Development

Since a goal of the described system is to be as low-cost as possible, improvements can always be made as technology and materials become more readily available. A more specific improvement in mechanical design would be the expansion to three dimensional movement. There are a few ways this system could be adapted to allow 3D movement, but it is important

that the method prevents the robot from interfering with desk space. The current design is meant to facilitate this conversion, as attaching it to a fixture above the user's desk would allow similar motion without interfering with the desktop in the way the current design does. Once the system is inverted, the design must allow motion in the vertical direction. The current design is most suited to adding a linear actuator of some sort to the carriage, which would allow the entire wrist-interface to move vertically below the carriage while the carriage moves in a 2D plane above the user. However, vertical motion could also be obtained via cables, although this would induce non-rigid dynamics and would require considerable modifications to the current design.

While the wrist-interface is effective in registering the presence of force inputs from the user, reducing the friction between the sliding components of the wrist-interface could allow the FSRs to be used to better estimate the magnitudes of applied forces. Simply finding a lower-friction alternative to the laser-cut acrylic would likely reduce the minimum detectable force and allow smoother force readings.

The way in which the user's wrist is attached to the wrist-interface could also be improved, as the current wrist brace is more time-consuming to attach than desired. Ideally, the user would be able to attach and detach the end-effector effortlessly, without needing to actually attach or detach anything.

7.1.2. Electronic/Software Development

The positioning of the eye-tracker is the area in most need of attention during future development. The current eye-tracker holder works well in terms of its adaptability to the user, due to its easily adjusted height, angle and position. However, the holder takes up space and obstructs some arm movements, which make it a hindrance to the system's ultimate goals. A

suspension system could be designed to "float" the eye-tracker above the table, but a more ideal solution is to simply place the eye-tracker on the table. Now only would this clear the desktop and free the workspace from any obstacles, but it would also allow the user to sit at the desk, something the current system does not allow. This setup would require a few improvements upon current commercially available eye-tracker technology. To begin, the eye-tracker must be able to recover eye-gaze data even if an arm obstructs the view from one sensor. In addition, the eye-tracker must be able to track eye-gaze data when the user looks beyond the width of the eye-tracker. Both of these could be solved with the inclusion of a second or third eye-tracker, as the total width of the eye-tracker collection would span the width of the desktop and if one eye-tracker is blocked another could relay the data instead. However, the Tobii EyeX is not currently compatible with multiple eye-trackers, since each could interfere with the others' infrared reflections. Theoretically the controller could switch between eye-trackers quickly to simulate all being active at once, but this may require considerable programming to implement.

Of course, improvements can always be made to the controller as well. The current controller displayed sufficient accuracy from eye-gaze input to end-effector position, but the conversions between desired velocity and pulse durations could be better formulated, particularly for the rotational motor. One such improvement would be to have the system automatically adjust, or calibrate, the estimated moment of inertia of the system. The current controller calculates the desired velocity based on the estimated current velocity and the ratio K_e/J, as dictated by motor dynamics. However, the system's moment of inertia J is currently adjusted to the user's comfort and then held constant, whereas the true value of J is constantly changing. Better estimation of the moment of inertia could elicit much improved responses from the rotational motor.

7.1.3. Experimentation with Intention-Detection

There are a vast number of potential ways to improve the intention detection capabilities of the system and the evaluation of which are best suited for the task will require countless well-designed experiments. User input could be given from blink data, a hand device such as a joystick, verbal communication, etc. The field of intention detection is expansive and ever-growing, and the specific ability to determine an exact location from subtle clues is far from being fully examined. The results shown in this thesis show the promise of using eye-gaze data to determine a user's desired movement, but future systems would benefit greatly from being able to combine a wide array of collected non-physical user inputs to augment eye-gaze data.

APPENDIX A

ARDUINO CODE

```
//Message received from MATLAB
char messageIn[]={0,0,0};
//Message to be sent to MATLAB
char messageOut[]={0,0};
//Pin Assignments
int motor_r = 3;
int potpin_r = 0;
int motor_theta = 11;
int potpin_theta = 5;
//PW controls the KEPCO current output.  Must be between 0-255.
int pw_r=0;
int pw_theta=0;
//Stores analog read values for FSR and potentiometers
int voltage[]={0,0,0,0,0,0};

#include <AltSoftSerial.h>
int xbeeByte;
unsigned long time;
int returnInd = 50;
int adcVal[] = {0,0,0,0,0,0,0,0};
byte requestADC[] = {126,0,15,23,1,0,19,162,0,64,140,88,78,255,254,2,73,83,37};
byte changeBaudLocal[] = {126,0,5,8,1,66,68,6,106};
AltSoftSerial xbee;

void setup() {
  Serial.begin(115200);
  //Set XBee communication to 57600 bits/s
  xbee.begin(9600);
  xbee.write(changeBaudLocal,9);
  xbee.end();
  xbee.begin(57600);
}

void loop(){
  //If no recent actions have occured...
  //...trigger reset by setting returnInd equal to 50
  if ((millis()-time)>100)
  {
    returnInd = 50;
  }

  //Receive message from MATLAB
  if(Serial.available()>0)
  {
    //The message will have 3 bytes.
```

```
  //[0] => Type of message. 1 = Begin.  2 = Send voltage values. 3 = Send PW values to current
controller.
  //[1] => Only used for sending PW  (see below)
  //[2] => Only used for sending PW  (see below)
  Serial.readBytes(messageIn,3);
  //Initialize
  if (messageIn[0]==1){
     Serial.write(1);
     Serial.write(1);
     returnInd = 50;
     int voltage[]={0,0,0,0,0,0};

     //Clear current XBee readings
     while (xbee.available()>0)
     {
      xbee.read();
     }
  }

  //MATLAB is asking for FSR and pot values. Send these to MATLAB.
  //Update these in the void loop.
  if (messageIn[0]==2){
    if (returnInd > 30)
    {
      time = millis();
      xbee.write(requestADC,19);
      returnInd = 0;
    }
    voltage[0] = analogRead(potpin_r);
    voltage[1] = analogRead(potpin_theta);

    //Sends a 2-byte message for each voltage value.
    //[0] = Voltage/256
    //[1] = Remainder
    for(int i=0;i<6;i++){
      messageOut[0]=voltage[i]/256;
      messageOut[1]=voltage[i]-messageOut[0]*256;
      for(int j=0;j<2;j++) {
        Serial.write(messageOut[j]);
      }
    }
  }

  //Receive PW value from MATLAB and send to current controllers
  if (messageIn[0]==3){
    pw_r=int(messageIn[1]);
    pw_theta = int(messageIn[2]);
    analogWrite(motor_r,pw_r);
    analogWrite(motor_theta,pw_theta);
  }
}
```

```
  //Read information from XBee if available
  if (xbee.available())
  {
    xbeeByte = int(xbee.read());
    if (returnInd == 22)
    {
      adcVal[0]=xbeeByte;
    }
    else if (returnInd == 23)
    {
      adcVal[1]=xbeeByte;
      voltage[2] = adcVal[0]*256+adcVal[1];
    }
    else if (returnInd == 24)
    {
      adcVal[2]=xbeeByte;
    }
    else if (returnInd == 25)
    {
      adcVal[3]=xbeeByte;
      voltage[3] = adcVal[2]*256+adcVal[3];
    }
    else if (returnInd == 26)
    {
      adcVal[4]=xbeeByte;
    }
    else if (returnInd == 27)
    {
      adcVal[5]=xbeeByte;
      voltage[4] = adcVal[4]*256+adcVal[5];
    }
    else if (returnInd == 28)
    {
      adcVal[6]=xbeeByte;
    }
    else if (returnInd == 29)
    {
      adcVal[7]=xbeeByte;
      voltage[5] = adcVal[6]*256+adcVal[7];
    }
    returnInd++;
  }
}
```

APPENDIX B

MATLAB CODE

```
function time=positionAndForce2_EyeGaze(a,x,y)
%Drives the 2D system to the location specified by "posDesired" in the
%constants section.  Accepts input from FSRs (priority)

%Assume Arduino returns voltages as [Pos, Left FSR, Right FSR, Front FSR, Back FSR]

%Call on command window beforehand...
%a=serial('COM3')
%a.BaudRate=115200
%a.ReadAsyncMode='manual'
%fopen(a)
%justPosition(a)

%% Constants
%data = ['Time','LeftForce','RightForce','FrontForce','BackForce','XposDes','XvelDes','XposCur','
useEyeTracker = 1;
doCalibrate = 0;
messageIn=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
messageOut=[0,0,0];
%XY Constants (Units = mm)
posDesired_x = x;
posDesired_y = y;
lastPwUpdate=0;
timeSincePwUpdate = 0;
%X,Y Constants (Units = mm)
acceptableDistance_r_Min = 15;
acceptableDistance_r_Max = 25;
acceptableDistance_r = 15;
acceptableDistance_theta = 0.01;
acceptableDistance_theta_Min = 0.01;
acceptableDistance_theta_Max = 0.02;
accel_x = 0;
accel_y = 0;
accel = 1500;
accelMax = 1500;
velMax_x = 1000;
velMax_y = 1000;
velDesired_x=0;
velDesired_y=0;
%R Constants (mm)
posMax_r=930;
posMin_r=350;
pwZero_r=193;
pwMin_r=160;
pwMax_r=209;
velMin_r = 20;
velMax_r = 1000;
velDesired_r = 0;
```

```
pw_r=pwZero_r;
pwPosition_r=0;
%Theta Constants (Units = radians)
posMax_theta=0.4;
posMin_theta=0;
pwZero_theta=184;
pwMin_theta=170;
pwMax_theta=198;
velMin_theta = 0.07;
velMax_theta = 0.9;
velDesired_theta = 0;
pw_theta=pwZero_theta;
pw_theta_Des = pwZero_theta;
pwPosition_theta=0;
velCalibrationOffset = 0;
arrived = 0;
posDif = 0;
lastPosDif = 0;
timeProgressMade = 0;
%Force Feedback Constants
force_x = 0;
force_y = 0;
force_r = 0;
force_theta = 0;
lastForce_r=0;
lastForce_theta=0;
lastForceUpdate=0;
timeLastForceDrive = 0;
smoothNum = 5;
positionDrive=1;
forceDrive=0;
forceVel_x = 0;
forceVel_y = 0;
forceAccel_x=0;
%Universal Constants
emerStop=0;
firstRun=1;
shutOffMin = 183;
shutOffMid = 184;
shutOffMax = 185;

%Eye-Tracker Calibration
if(useEyeTracker==1)
%Setup socket communication
s=tcpip('127.0.0.1', 20005, 'NetworkRole', 'client');
fopen(s);
%Send request for eye-tracker data
fwrite(s,'C');
currentFrame = 0;
calibrationPoints = 4;
calXCoords = [600,900,900,600];
calYCoords = [200,200,0,0];
calibrationXMin = 600;
calibrationXMax = 900;
calibrationYMin = 0;
calibrationYMax = 200;
```

```
%Arrays to store calibration data
framesPerPoint = 50;
calLeftX = zeros(calibrationPoints,framesPerPoint);
calLeftY = zeros(calibrationPoints,framesPerPoint);
calRightX = zeros(calibrationPoints,framesPerPoint);
calRightY = zeros(calibrationPoints,framesPerPoint);
avgLeftX = zeros(calibrationPoints,1);
avgLeftY = zeros(calibrationPoints,1);
avgRightX = zeros(calibrationPoints,1);
avgRightY = zeros(calibrationPoints,1);
calXAvg = zeros(calibrationPoints,1);
calYAvg = zeros(calibrationPoints,1);
failsAllowedCalibrate = 20;
failsAllowedPercentage = 0.2;
EyeGazeX=[-1];
EyeGazeY=[-1];
xGaze = 0;
yGaze = 0;
timeEyeGaze = zeros(1,1);
fprintf('Get ready to calibrate \n');
pause(2);
begin=0;

%Setup beep used to tell user calibration status
duration=1;
freq=3000;
fs = 3*freq;
values=0:1/fs:duration;
longBeep=sin(2*pi* freq*values);
duration=0.2;
freq=3000;
fs = 3*freq;
values=0:1/fs:duration;
shortBeep=sin(2*pi* freq*values);
end

%Initialize
%% Start
start=0;
%Initialize Arduino
tic;
while(start==0)
time=toc;
%readasync BEFORE sending message...
if(a.TransferStatus == 'idle')
readasync(a,2);
end
%Send command to initialize Arduino
messageOut=[1,1,0];
fwrite(a,messageOut);
while(a.BytesAvailable<2)
%Wait for response from Arduino
if (toc-time)>1
break;
end
```

```
end
%Error handling if a complete response isn't received
if(a.BytesAvailable<2&&a.BytesAvailable>0)
fread(a,a.BytesAvailable);
continue;
end
if(a.BytesAvailable==0)
continue;
end
%Read response from Arduino
messageIn=fread(a,a.BytesAvailable);
if (messageIn(1)==1)
if (messageIn(2)==1)
start=1;
end
end
end

%Setup Eye Tracker
if(useEyeTracker==1)
%Pre-calibrated Eye-Tracker Data is default
leftXAvg = 0.2661;
rightXAvg = 0.8503;
topYAvg = 0.5211;
bottomYAvg = -0.0244;
XRange = 0.5841;
YRange = 0.5455;
%Initialize Eye-tracker
while(~begin)
if s.BytesAvailable>0
event = char(fread(s, s.BytesAvailable));
index = 0;
if (event == ['B';'e';'g';'i';'n'])
begin = 1;
end
end
end
%Calibration Procedure
if(doCalibrate==1)
goToPoint(a,900,0);
for(currentPoint = 1:calibrationPoints)
%Read location for current calibration point
currentPointX = calXCoords(currentPoint);
currentPointY = calYCoords(currentPoint);
tic;
%Send end-effector to current calibration point
goToPoint(a,currentPointX,currentPointY);
fprintf('Look to the %s\n',calibrationPointNames(currentPoint));
pause(1.5);
sound(shortBeep,fs);
pause(1.5);
sound(shortBeep,fs);
pause(1.5);
sound(longBeep,fs);
%Request data from eye-tracker
fwrite(s,'C');
```

```
calibrate = 1;
while(calibrate)
%Read response from eye-tracker, if available
if s.BytesAvailable>0
event = char(fread(s, s.BytesAvailable));
index = 0;
currentFrame = currentFrame + 1;
%Seperate string into four doubles
for(i=1:length(event))
if event(i)==',';
if(index==0)
xLeft = event(1:i-1);
xLeft = str2double(xLeft);
index = index + 1;
start = i+1;
elseif (index == 1)
yLeft = event(start:i-1);
yLeft = str2double(yLeft);
index = index + 1;
start = i+1;
elseif (index == 2)
xRight = event(start:i-1);
xRight = str2double(xRight);
index = index + 1;
start = i+1;
elseif (index == 3)
yRight = event(start:i-1);
yRight = str2double(yRight);
index = index + 1;
end
end
end
%Store eye gaze
calLeftX(currentPoint,currentFrame) = xLeft;
calLeftY(currentPoint,currentFrame) = yLeft;
calRightX(currentPoint,currentFrame) = xRight;
calRightY(currentPoint,currentFrame) = yRight;
if currentFrame<framesPerPoint
%Keep collecting data...
fwrite(s,'C');
else
%If failed, start over
if
(sum(calLeftX(currentPoint,:)==0)>failsAllowedCalibrate)||(sum(calLeftY(currentPoint,:)==0)>failsAllowedCalibra
te)||(sum(calRightX(currentPoint,:)==0)>failsAllowedCalibrate)||(sum(calRightY(currentPoint,:)==0)>failsAllowed
Calibrate)
fprintf('Calibration Failed\n');
sound(longBeep,fs);
pause(1.5);
sound(longBeep,fs);
pause(1.5);
fprintf('Try calibration again\n');
sound(shortBeep,fs);
pause(1.5);
sound(shortBeep,fs);
pause(1.5);
```

```
sound(longBeep,fs);
pause(1.5);
currentFrame = 0;
fwrite(s,'C');
continue;
end
%When successful calibration occurs...
%Set variables to move to next calibration point
currentFrame = 0;
calibrate = 0;
%Calculate and store important values
avgLeftX(currentPoint,1) = sum(calLeftX(currentPoint,:))/sum(calLeftX(currentPoint,:)~=0)
avgLeftY(currentPoint,1) = sum(calLeftY(currentPoint,:))/sum(calLeftY(currentPoint,:)~=0)
avgRightX(currentPoint,1) = sum(calRightX(currentPoint,:))/sum(calRightX(currentPoint,:)~=0)
avgRightY(currentPoint,1) = sum(calRightY(currentPoint,:))/sum(calRightY(currentPoint,:)~=0)
calXAvg(currentPoint,1) = (avgLeftX(currentPoint,1)+avgRightX(currentPoint,1))/2
calYAvg(currentPoint,1) = (avgLeftY(currentPoint,1)+avgRightY(currentPoint,1))/2
continue;
end
end
end
end

%Final Calibration Calculations
leftXAvg = (calXAvg(1,1)+calXAvg(4,1))/2
rightXAvg = (calXAvg(2,1)+calXAvg(3,1))/2
topYAvg = 1 - (calYAvg(1,1)+calYAvg(2,1))/2
bottomYAvg = 1 - (calYAvg(3,1)+calYAvg(4,1))/2
XRange = rightXAvg - leftXAvg
YRange = topYAvg-bottomYAvg
calData = [leftXAvg,rightXAvg,topYAvg,bottomYAvg,XRange,YRange];
%Store calibration results
filename = 'Calibrate Four Points.xlsx';
xlswrite(filename,calData);
end
fwrite(s,'C');
end

%% Main Loop
tic;
lastPwUpdate = toc;
while(1)
%% Initialize
time=toc;
%   Read Data
if(a.TransferStatus == 'idle')
readasync(a,12);
end
messageOut=[2,0,0];
fwrite(a,messageOut)
while(a.BytesAvailable<12)
if (toc-time)>0.02
break;
end
end
```

```
%If incomplete, delete data and try again...
if(a.BytesAvailable<12&&a.BytesAvailable>0)
fread(a,a.BytesAvailable);
continue;
end
if(a.BytesAvailable==0)
continue;
end

%% Get values
messageIn=fread(a,a.BytesAvailable);
potValue_r=(messageIn(1)*256+messageIn(2));
potValue_theta = (messageIn(3)*256+messageIn(4));
rightForce=(messageIn(5)*256+messageIn(6));
backForce=(messageIn(7)*256+messageIn(8));
frontForce=(messageIn(9)*256+messageIn(10));
leftForce = (messageIn(11)*256+messageIn(12));
rightForce = round(1.25*rightForce);
leftForce = round(0.9*leftForce);

%Set maximum limits to force input.  Currently set to max possible ADC
%output.
if(leftForce>1024)
leftForce=0;
end
if(rightForce>1024)
rightForce=0;
end
if(backForce>1024)
backForce=0;
end
if(frontForce>1024)
frontForce=0;
end

%Store current position
if(firstRun==1)
posCurrent_r = 0.748951*(potValue_r+367);
posCurrent_theta = 0.004313*potValue_theta-2.3;
firstRun=0;
continue;
end

%Calculate current position
posLast_r = posCurrent_r;
posCurrent_r = 0.748951*(potValue_r+367);
posLast_theta = posCurrent_theta;
posCurrent_theta = 0.004313*potValue_theta-2.23413;
posCurrent_x = (posCurrent_r)*cos(posCurrent_theta);
posCurrent_y = (posCurrent_r)*sin(posCurrent_theta);

%% Calculate Desired Positions. Check boundaries...
%NOTE: Only update eyetracker posDes if last n gaze points were
%consistent AND gazepoint is L distance away from current point...
if(useEyeTracker==1)
```

```
%POSITION FEEDBACK (WITH EYE GAZE)
if s.BytesAvailable>0
event = char(fread(s, s.BytesAvailable));
index = 0;
for(i=1:length(event))
if event(i)==',';
if(index==0)
xLeft = event(1:i-1);
xLeft = str2double(xLeft);
index = index + 1;
start = i+1;
elseif (index == 1)
yLeft = event(start:i-1);
yLeft = str2double(yLeft);
index = index + 1;
start = i+1;
elseif (index == 2)
xRight = event(start:i-1);
xRight = str2double(xRight);
index = index + 1;
start = i+1;
elseif (index == 3)
yRight = event(start:i-1);
yRight = str2double(yRight);
index = index + 1;
end
end
end
end
%If successful read
if (xLeft>0)&&(xRight>0)&&(yLeft>0)&&(yRight>0)
%Calculate desired position from gaze data
yLeft = 1-yLeft;
yRight = 1-yRight;
xGaze = ((((xLeft+xRight)/2-leftXAvg)/XRange)*(calibrationXMax-calibrationXMin)+calibrationXMin);
yGaze = ((((yLeft+yRight)/2-bottomYAvg)/YRange)*(calibrationYMax-calibrationYMin)+calibrationYMin);
xGaze = min(xGaze,calibrationXMax);
xGaze = max(xGaze,calibrationXMin);
yGaze = min(yGaze,calibrationYMax);
yGaze = max(yGaze,calibrationYMin);
else %If failed read
xGaze = -1;
yGaze = -1;
end
%Delate first (oldest) entry
EyeGazeX(length(EyeGazeX)+1) = xGaze;
EyeGazeY(length(EyeGazeY)+1) = yGaze;
timeEyeGaze(length(timeEyeGaze)+1) = time;
while((timeEyeGaze(length(timeEyeGaze))-timeEyeGaze(1))>1.5)
EyeGazeX(1)=[];
EyeGazeY(1) = [];
timeEyeGaze(1) = [];
end
%Add newest eye gaze data
if (((sum(EyeGazeX~=(-1))/length(EyeGazeX))>failsAllowedPercentage)&&((sum(EyeGazeY~=(-
1))/length(EyeGazeY))>failsAllowedPercentage)) %If enough points were successful
```

```
%If close enough together...
if ((range(EyeGazeX(EyeGazeX~=(-1)))<30)&&(range(EyeGazeY(EyeGazeY~=(-1)))<30))
newPosDesired_x = sum(EyeGazeX)/sum(EyeGazeX~=(-1));
newPosDesired_y = sum(EyeGazeY)/sum(EyeGazeY~=(-1));
%If certain distance from current desired position
if((abs(newPosDesired_x-posDesired_x)>100)||(abs(newPosDesired_y-posDesired_y)>100))
posDesired_x = newPosDesired_x;
posDesired_y = newPosDesired_y;
end
else
%fprintf('Too Spread Out');
end
end

fwrite(s,'C');
end

posDesired_r = sqrt(posDesired_x^2+posDesired_y^2);
posDesired_theta = atan2(posDesired_y,posDesired_x);
posDif = sqrt((posDesired_x-posCurrent_x)^2+(posDesired_y-posCurrent_y)^2);

%FORCE FEEDBACK
force_r = rightForce-leftForce;
force_theta = backForce-frontForce;
lastForce_r(1)=[];
lastForce_theta(1)=[];
lastForce_r(smoothNum)=force_r;
lastForce_theta(smoothNum)=force_theta;
force_r = mean(lastForce_r);
force_theta = mean(lastForce_theta);
%Check if new force input was received
if((lastForce_r(smoothNum)~=lastForce_r(smoothNum-
1))||(lastForce_theta(smoothNum)~=lastForce_theta(smoothNum-1)))
lastForceUpdate=toc;
end
%If force hasn't been updated in 0.3s, assume data is invalid
%Prevents frozen XBee from sending large force indefinitely
if((time-lastForceUpdate)>0.3)
force_r=0;
force_theta=0;
end

force_x = force_r*cos(posCurrent_theta)-force_theta*sin(posCurrent_theta);
force_y = force_r*sin(posCurrent_theta)+force_theta*cos(posCurrent_theta);

%ignorableForce is the force below which force inputs will be
%ignored.  This allows inertial forces to be discarded.
magVel = sqrt(velDesired_x^2+velDesired_y^2);
ignorableForce = 10*magVel;
ignorableForce = max(ignorableForce,300);
ignorableForce = min(ignorableForce,1000);
magForce = sqrt(force_x^2+force_y^2);
%Determine Feedback Method
%If force is over 900, OR greater than 10*velocity...
if(magForce>ignorableForce)
```

```
%Set controller to force mode
forceDrive=1;
velCalibrationOffset = 0;
timeLastForceDrive = time;
positionDrive=0;
end

%If 2 seconds has passed without significant force input
if((time-timeLastForceDrive)>2)
%Set controller to position mode
forceDrive = 0;
forceVel_x = 0;
forceVel_y = 0;
positionDrive = 1;
end
timeSincePwUpdate = toc-lastPwUpdate;
if(forceDrive)
%Update desired velocity
forceVel_x = velDesired_x;
forceVel_y = velDesired_y;
posDesired_x=posCurrent_x;
posDesired_y = posCurrent_y;
if(forceVel_x==0)
forceAccel_x = force_x*0.8;
forceAccel_x = min(accel,forceAccel_x);
forceAccel_x = max(-accel,forceAccel_x);
forceVel_x = forceVel_x + forceAccel_x*timeSincePwUpdate;
elseif(forceVel_x>0)
if(force_x>300)
forceAccel_x = force_x*0.8;
forceAccel_x = min(accel,forceAccel_x);
forceAccel_x = max(-accel,forceAccel_x);
forceVel_x = forceVel_x + forceAccel_x*timeSincePwUpdate;
elseif(force_x<150)
forceAccel_x = -1500;
forceVel_x = forceVel_x + forceAccel_x*timeSincePwUpdate;
if(forceVel_x<0)
forceVel_x=0;
end
end
%else... 150 < force_x < 300 => keep velDes constant.
else %forceVel_x<0
if(force_x<-300)
forceAccel_x = force_x*0.8;
forceAccel_x = min(accel,forceAccel_x);
forceAccel_x = max(-accel,forceAccel_x);
forceVel_x = forceVel_x + forceAccel_x*timeSincePwUpdate;
elseif(force_x>(-150))
forceAccel_x = 1500;
forceVel_x = forceVel_x + forceAccel_x*timeSincePwUpdate;
if(forceVel_x>0)
forceVel_x=0;
end
end
end
velDesired_x = forceVel_x;
```

```
if(forceVel_y==0)
forceAccel_y = force_y*0.5;
forceAccel_y = min(accel,forceAccel_y);
forceAccel_y = max(-accel,forceAccel_y);
forceVel_y = forceVel_y + forceAccel_y*timeSincePwUpdate;
elseif(forceVel_y>0)
if(force_y>300)
forceAccel_y = force_y*0.5;
forceAccel_y = min(accel,forceAccel_y);
forceAccel_y = max(-accel,forceAccel_y);
forceVel_y = forceVel_y + forceAccel_y*timeSincePwUpdate;
elseif(force_y<150)
forceAccel_y = -2000;
forceVel_y = forceVel_y + forceAccel_y*timeSincePwUpdate;
if(forceVel_y<0)
forceVel_y=0;
end
end
else %forceVel_x<0
if(force_y<-300)
forceAccel_y = force_y*0.5;
forceAccel_y = min(accel,forceAccel_y);
forceAccel_y = max(-accel,forceAccel_y);
forceVel_y = forceVel_y + forceAccel_y*timeSincePwUpdate;
elseif(force_y>(-150))
forceAccel_y = 2000;
forceVel_y = forceVel_y + forceAccel_y*timeSincePwUpdate;
if(forceVel_y>0)
forceVel_y=0;
end
end
end
velDesired_y = forceVel_y;
end

%POSITION FEEDBACK
if(positionDrive)
%Set limits in x-y coordinate frame
if (posDesired_x<350)
posDesired_x = 350;
end
if(posDesired_x>900)
posDesired_x = 900;
end
if(posDesired_y>400)
posDesired_y = 400;
end
if(posDesired_y<0)
posDesired_y = 0;
end

posDesired_r = sqrt(posDesired_x^2+posDesired_y^2);
posDesired_theta = atan2(posDesired_y,posDesired_x);
```

```
% POSITION FEEDBACK
%Set limits in r-theta coordinate frame
if (posDesired_r>posMax_r)
posDesired_r=posMax_r;
end
if ((posDesired_r<posMin_r))
posDesired_r=posMin_r;
end
if (posDesired_theta>posMax_theta)
posDesired_theta=posMax_theta;
end
if ((posDesired_theta<posMin_theta))
posDesired_theta=posMin_theta;
end

posDesired_x = (posDesired_r)*cos(posDesired_theta);
posDesired_y = (posDesired_r)*sin(posDesired_theta);

%Calculate direction of desired acceleration
%If x OR y component of acceleration is greater than limit, adjust
%entire acceleration accordingly.
accelAngle = atan2((posDesired_y-posCurrent_y),(posDesired_x-posCurrent_x));
accel_x = accel*cos(accelAngle);
accel_y = accel*sin(accelAngle);
accel_r = accel_x*cos(posCurrent_theta)+accel_y*sin(posCurrent_theta);
accel_theta = (accel_y*cos(posCurrent_theta)-accel_x*sin(posCurrent_theta))/posCurrent_r;
multiplier = 1;
if((velDesired_r+accel_r*timeSincePwUpdate)>velMax_r)
multiplier = abs((velMax_r-velDesired_r)/(accel_r*timeSincePwUpdate));
end
if((velDesired_r+accel_r*timeSincePwUpdate)<-velMax_r)
multiplier = abs((-velMax_r-velDesired_r)/(accel_r*timeSincePwUpdate));
end
if((velDesired_theta+accel_theta*timeSincePwUpdate)>velMax_theta)
multiplier = abs((velMax_theta-velDesired_theta)/(accel_theta*timeSincePwUpdate));
end
if((velDesired_theta+accel_theta*timeSincePwUpdate)<-velMax_theta)
multiplier = abs((-velMax_theta-velDesired_theta)/(accel_theta*timeSincePwUpdate));
end

accel_x = multiplier*accel_x;
accel_y = multiplier*accel_y;

%Once the end-effector reaches the target, the acceptable distance
%is made slightly larger to prevent jittery motion
if((abs(posDesired_r-posCurrent_r)<acceptableDistance_r_Min)&&(abs(posDesired_theta-
posCurrent_theta)<acceptableDistance_theta_Min))
acceptableDistance_r = acceptableDistance_r_Max;
acceptableDistance_theta = acceptableDistance_theta_Max;
end
if((abs(posDesired_r-posCurrent_r)>acceptableDistance_r_Max)||(abs(posDesired_theta-
posCurrent_theta)>acceptableDistance_theta_Max))
acceptableDistance_r = acceptableDistance_r_Min;
acceptableDistance_theta = acceptableDistance_theta_Min;
end
```

```
%X-PositionFeedback
arrived=0;
%Check if we're close enough... If so, decelerate
if ((abs(posDesired_r-posCurrent_r)<acceptableDistance_r)&&(abs(posDesired_theta-
posCurrent_theta)<acceptableDistance_theta))
%Decelerate..
arrived = 1;
if (velDesired_x>0)
velDesired_x = velDesired_x - (accel*timeSincePwUpdate); %Update desired velocity
end
if (velDesired_x<0)
velDesired_x = velDesired_x + (accel*timeSincePwUpdate);
end
%Check if we are past the desiredPosition and moving away... If so,
%decelerate
elseif(((posDesired_x-posCurrent_x)*(velDesired_x))<0)
%Decelerate
if (velDesired_x>0)
velDesired_x = velDesired_x - (accel*timeSincePwUpdate); %Update desired velocity
elseif (velDesired_x<0)
velDesired_x = velDesired_x + (accel*timeSincePwUpdate);
end
%See if we're approaching from the correct direction... If so,
%decelerate
elseif ((abs(posDesired_x-posCurrent_x)-abs(velDesired_x*0.15))<=(velDesired_x^2/(2*accel)))
%Decelerate
if (velDesired_x>0)
velDesired_x = velDesired_x - (accel*timeSincePwUpdate); %Update desired velocity
end
if (velDesired_x<0)
velDesired_x = velDesired_x + (accel*timeSincePwUpdate);
end
%This is a small buffer to prevent too much switching by the controller
%If we're CLOSE to approaching, don't do anything
elseif ((abs(posDesired_x-posCurrent_x)-abs(velDesired_x*0.2))<=(velDesired_x^2/(2*accel)))
else %We are far enough away... Pedal to the metal
%Accelerate
%Check if max vel is reached... have to change other DOF too...
if (posDesired_x>posCurrent_x)
velDesired_x = velDesired_x + (accel_x*timeSincePwUpdate); %Update desired velocity
end
if (posDesired_x<posCurrent_x)
velDesired_x = velDesired_x + (accel_x*timeSincePwUpdate);
end
end

%Y-PositionFeedback
if ((abs(posDesired_r-posCurrent_r)<acceptableDistance_r)&&(abs(posDesired_theta-
posCurrent_theta)<acceptableDistance_theta))
%Decelerate..
if (velDesired_y>0)
velDesired_y = velDesired_y - (accel*timeSincePwUpdate); %Update desired velocity
end
if (velDesired_y<0)
```

```
velDesired_y = velDesired_y + (accel*timeSincePwUpdate);
end
%Check if we are past the desiredPosition and moving away... If so,
%decelerate
elseif(((posDesired_y-posCurrent_y)*(velDesired_y))<0)
%Decelerate
if (velDesired_y>0)
velDesired_y = velDesired_y - (accel*timeSincePwUpdate); %Update desired velocity
elseif (velDesired_y<0)
velDesired_y = velDesired_y + (accel*timeSincePwUpdate);
end
%See if we're approaching from the correct direction... If so,
%decelerate
elseif ((abs(posDesired_y-posCurrent_y)-abs(velDesired_y*0.15))<=(velDesired_y^2/(2*accel)))
%Decelerate
if (velDesired_y>0)
velDesired_y = velDesired_y - (accel*timeSincePwUpdate); %Update desired velocity
end
if (velDesired_y<0)
velDesired_y = velDesired_y + (accel*timeSincePwUpdate);
end
elseif ((abs(posDesired_y-posCurrent_y)-abs(velDesired_y*0.2))<=(velDesired_y^2/(2*accel)))
else %We are far enough away... Pedal to the metal
%Accelerate
if (posDesired_y>posCurrent_y)
velDesired_y = velDesired_y + (accel_y*timeSincePwUpdate); %Update desired velocity
end
if (posDesired_y<posCurrent_y)
velDesired_y = velDesired_y + (accel_y*timeSincePwUpdate);
end
end

%If the end-effector hasn't arrived yet
if(arrived==0)
%If progress hasn't been made in a certain amount of time, then
%increase velCalibrationOffset.  This variable acts as an
%integral component of the controller, and will make the
%controller stronger if the system's inertia prevents progress
%from being made.
if(posDif<(lastPosDif-20))
lastPosDif = posDif;
timeProgressMade = time;
elseif(posDif>lastPosDif)
lastPosDif = posDif;
end
if((time-timeProgressMade)>0.5)
velCalibrationOffset = velCalibrationOffset + timeSincePwUpdate;
velCalibrationOffset = min(velCalibrationOffset,10);
end
else
velCalibrationOffset = 0;
end
end

%Check Max Velocities
```

```
if (velDesired_x>velMax_x)
velDesired_x = velMax_x;
end
if (velDesired_x<-velMax_x)
velDesired_x = -velMax_x;
end
if (velDesired_y>velMax_y)
velDesired_y = velMax_y;
end
if (velDesired_y<-velMax_y)
velDesired_y = -velMax_y;
end

%CALCULATE AND SEND LINEAR MOTOR COMMANDS
velDesired_r = velDesired_x*cos(posCurrent_theta)+velDesired_y*sin(posCurrent_theta);

%Safety checks
if (velDesired_r>velMax_r)
velDesired_r = velMax_r;
end
if (velDesired_r<(-velMax_r))
velDesired_r = (-velMax_r);
end

%velCurrent is used for the conversion to pulse command.  We want to
%adjust this value without changing the true desired velocity.
velCurrent_r = velDesired_r;
if ((velCurrent_r<=-velMin_r)&&(velCurrent_r>-40))
velCurrent_r=-40;
end
if((velCurrent_r>=velMin_r)&&(velCurrent_r<100))
velCurrent_r=100;
end

%Calculate pwPosition from velDesired...
if (velCurrent_r<=-40)
pwPosition_r = 197+(5.6712*log(-velCurrent_r+100)-28.275);
elseif (velCurrent_r>=100)
pwPosition_r = 182 - (6.2092*log(velCurrent_r-70)-20.973);
else %-100<velCurrent<40
pwPosition_r = pwZero_r;
end
pwPosition_r = round(pwPosition_r);

%%CALCULATE AND SEND ROTATIONAL MOTOR COMMANDS
lastVelDesired_theta = velDesired_theta;
velDesired_theta = (velDesired_y*cos(posCurrent_theta)-velDesired_x*sin(posCurrent_theta))/posCurrent_r;

%Safety checks
if (velDesired_theta>velMax_theta)
velDesired_theta = velMax_theta;
end
if (velDesired_theta<(-velMax_theta))
velDesired_theta = (-velMax_theta);
end
```

```
%Implement motor dynamics
if(abs(velDesired_theta)>abs(lastVelDesired_theta))
accelDesired_theta = (velDesired_theta-lastVelDesired_theta)/timeSincePwUpdate;
velCurrent_theta = (accelDesired_theta/2)+lastVelDesired_theta;
else
velCurrent_theta = velDesired_theta;
end

%Set min values, since smaller values will not trigger any motion
if((velCurrent_theta*velDesired_theta)>0)
if ((velCurrent_theta<=-velMin_theta)&&(velCurrent_theta>-0.13))
velCurrent_theta=-0.13;
end
if((velCurrent_theta>=velMin_theta)&&(velCurrent_theta<0.35))
velCurrent_theta=0.35;
end
end

%Calculate pwPosition from velDesired...
if (velCurrent_theta<=-0.13)
pwPosition_theta = 175-(4.3809*log(-velCurrent_theta+0.25)+4.2+velCalibrationOffset);
elseif (velCurrent_theta>=0.2)
pwPosition_theta = 194 + (2.997*log(velCurrent_theta+0.15)+3.1272+velCalibrationOffset);
else %-100<velCurrent<40
pwPosition_theta = pwZero_theta;
end
pwPosition_theta = round(pwPosition_theta);

%% Safety Checks
pw_r=pwPosition_r;
pw_theta = pwPosition_theta;
lastPwUpdate=toc;

if((posCurrent_r<posMin_r)&&(pw_r>pwZero_r))
pw_r=pwZero_r;
end
if((posCurrent_r>posMax_r)&&(pw_r<pwZero_r))
pw_r=pwZero_r;
end
if((posCurrent_theta<posMin_theta)&&(pw_theta<pwZero_theta))
pw_theta=pwZero_theta;
end
if((posCurrent_theta>posMax_theta)&&(pw_theta>pwZero_theta))
pw_theta=pwZero_theta;
end

%Adjust pw commands to avoid 183-185, since these commands will turn
%off the RoboClaw
if (pw_r<=shutOffMid)&&(pw_r>=shutOffMin)
pw_r=shutOffMin-1;
end
if (pw_r>shutOffMid)&&(pw_r<=shutOffMax)
pw_r=shutOffMax+1;
end
```

```
if (pw_theta<=shutOffMid)&&(pw_theta>=shutOffMin)
pw_theta=shutOffMin-1;
end
if (pw_theta>shutOffMid)&&(pw_theta<=shutOffMax)
pw_theta=shutOffMax+1;
end

%If the system jumped, this means there was a lapse in communication,
%and all motion should be stopped
if (abs(posCurrent_r-posLast_r)>100)
emerStop = 1;
end
if (abs(posCurrent_theta-posLast_theta)>0.1)
emerStop = 1;
end

if(emerStop==1)
pw_r=pwZero_r;
pw_theta=pwZero_theta;
end

if (pw_r<pwMin_r)
pw_r=pwMin_r;
end
if (pw_r>pwMax_r)
pw_r=pwMax_r;
end
if (pw_theta<pwMin_theta)
pw_theta=pwMin_theta;
end
if (pw_theta>pwMax_theta)
pw_theta=pwMax_theta;
end

%For Easier Data Analysis
prevTime(6)=time;
prevTime(1)=[];
prevXPos(6) = posCurrent_x;
prevXPos(1)=[];
velXAvg = (prevXPos(5)-prevXPos(1))/(prevTime(5)-prevTime(1));
prevYPos(6) = posCurrent_y;
prevYPos(1)=[];
velYAvg = (prevYPos(5)-prevYPos(1))/(prevTime(5)-prevTime(1));
prevRPos(6) = posCurrent_r;
prevRPos(1)=[];
velRAvg = (prevRPos(5)-prevRPos(1))/(prevTime(5)-prevTime(1));
prevTPos(6) = posCurrent_theta;
prevTPos(1)=[];
velTAvg = (prevTPos(5)-prevTPos(1))/(prevTime(5)-prevTime(1));

%% Print
%fprintf('%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d
%d
%d\n',time,leftForce,rightForce,backForce,frontForce,forceVel_x,forceVel_y,velCalibrationOffset,posDesired_x,vel
Desired_x,posCurrent_x,velXAvg,posDesired_y,velDesired_y,posCurrent_y,velYAvg,posDesired_r,velDesired_r,p
```

```
osCurrent_r,velRAvg,velCurrent_r,pw_r,posDesired_theta,velDesired_theta,posCurrent_theta,velTAvg,velCurrent_t
heta,pw_theta,emerStop);
fprintf('%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d
%d\n',time,leftForce,rightForce,backForce,frontForce,forceVel_x,forceVel_y,velCalibrationOffset,xGaze,yGaze,pos
Desired_x,velDesired_x,posCurrent_x,velXAvg,posDesired_y,velDesired_y,posCurrent_y,velYAvg,posDesired_r,v
elDesired_r,posCurrent_r,velRAvg,velCurrent_r,pw_r,posDesired_theta,velDesired_theta,posCurrent_theta,velTAv
g,velCurrent_theta,pw_theta);

%Send signal to Arduino
messageOut=[3,pw_r,pw_theta];
fwrite(a,messageOut);
end
end

%USED FOR CALIBRATION... SAME METHOD AS IN MAIN FUNCTION
function goToPoint(a,calX,calY)
%% Constants
lastPwUpdate=0;
timeSincePwUpdate = 0;
acceptableDistance_r_Min = 15;
acceptableDistance_r_Max = 25;
acceptableDistance_r = 15;
acceptableDistance_theta = 0.01;
acceptableDistance_theta_Min = 0.01;
acceptableDistance_theta_Max = 0.02;
accel_x = 0;
accel_y = 0;
accel = 1500;
accelMax = 1500;
velMax_x = 1000;
velMax_y = 1000;
velDesired_x=0;
velDesired_y=0;
%R Constants (mm)
posMax_r=900;
posMin_r=350;
pwZero_r=193;
pwMin_r=160;
pwMax_r=209;
velMin_r = 20;
velMax_r = 1000;
velDesired_r = 0;
pw_r=pwZero_r;
pwPosition_r=0;
%Theta Constants (Units = radians)
posMax_theta=0.4;
posMin_theta=0;
pwZero_theta=184;
pwMin_theta=170;
pwMax_theta=198;
velMin_theta = 0.07;
velMax_theta = 0.9;
velDesired_theta = 0;
pw_theta=pwZero_theta;
```

```
pw_theta_Des = pwZero_theta;
pwPosition_theta=0;
velCalibrationOffset = 0;
arrived = 0;
posDif = 0;
lastPosDif = 0;
timeProgressMade = 0;
emerStop=0;
firstRun=1;
shutOffMin = 183;
shutOffMid = 184;
shutOffMax = 185;
%% Real Code
firstRun=1;
lastTimeAway = 0;
posDesired_x = calX;
posDesired_y = calY;
pointReached = 0;
while (pointReached==0)
%% Initialize
time=toc;
%   Read Data
if(a.TransferStatus == 'idle')
readasync(a,12);
end
messageOut=[2,0,0];
fwrite(a,messageOut)
while(a.BytesAvailable<12)
if (toc-time)>0.02
break;
end
end
%If incomplete, delete data and try again...
if(a.BytesAvailable<12&&a.BytesAvailable>0)
fread(a,a.BytesAvailable);
continue;
end
if(a.BytesAvailable==0)
continue;
end

%% Get values
messageIn=fread(a,a.BytesAvailable);
potValue_r=(messageIn(1)*256+messageIn(2));
potValue_theta = (messageIn(3)*256+messageIn(4));

if(firstRun==1)
posCurrent_r = 0.748951*(potValue_r+367);
posCurrent_theta = 0.004313*potValue_theta-2.2806;
firstRun=0;
continue;
end

posLast_r = posCurrent_r;
posCurrent_r = 0.748951*(potValue_r+367);
```

```
posLast_theta = posCurrent_theta;
posCurrent_theta = 0.004313*potValue_theta-2.23413;
posCurrent_x = (posCurrent_r)*cos(posCurrent_theta);
posCurrent_y = (posCurrent_r)*sin(posCurrent_theta);
posDesired_r = sqrt(posDesired_x^2+posDesired_y^2);
posDesired_theta = atan2(posDesired_y,posDesired_x);

% POSITION FEEDBACK
%Put limits on posdesired... (not really necessary for this function)
if (posDesired_r>posMax_r)
posDesired_r=posMax_r;
end
if ((posDesired_r<posMin_r))
posDesired_r=posMin_r;
end
if (posDesired_theta>posMax_theta)
posDesired_theta=posMax_theta;
end
if ((posDesired_theta<posMin_theta))
posDesired_theta=posMin_theta;
end

posDesired_x = (posDesired_r)*cos(posDesired_theta);
posDesired_y = (posDesired_r)*sin(posDesired_theta);
accelAngle = atan2((posDesired_y-posCurrent_y),(posDesired_x-posCurrent_x));
accel_x = accel*cos(accelAngle);
accel_y = accel*sin(accelAngle);
accel_r = accel_x*cos(posCurrent_theta)+accel_y*sin(posCurrent_theta);
accel_theta = (accel_y*cos(posCurrent_theta)-accel_x*sin(posCurrent_theta))/posCurrent_r;
multiplier = 1;
if((velDesired_r+accel_r*timeSincePwUpdate)>velMax_r)
multiplier = abs((velMax_r-velDesired_r)/(accel_r*timeSincePwUpdate));
end
if((velDesired_r+accel_r*timeSincePwUpdate)<-velMax_r)
multiplier = abs((-velMax_r-velDesired_r)/(accel_r*timeSincePwUpdate));
end
if((velDesired_theta+accel_theta*timeSincePwUpdate)>velMax_theta)
multiplier = abs((velMax_theta-velDesired_theta)/(accel_theta*timeSincePwUpdate));
end
if((velDesired_theta+accel_theta*timeSincePwUpdate)<-velMax_theta)
multiplier = abs((-velMax_theta-velDesired_theta)/(accel_theta*timeSincePwUpdate));
end

accel_x = multiplier*accel_x;
accel_y = multiplier*accel_y;

if((abs(posDesired_r-posCurrent_r)<acceptableDistance_r_Min)&&(abs(posDesired_theta-
posCurrent_theta)<acceptableDistance_theta_Min))
acceptableDistance_r = acceptableDistance_r_Max;
acceptableDistance_theta = acceptableDistance_theta_Max;
end
if((abs(posDesired_r-posCurrent_r)>acceptableDistance_r_Max)||(abs(posDesired_theta-
posCurrent_theta)>acceptableDistance_theta_Max))
acceptableDistance_r = acceptableDistance_r_Min;
acceptableDistance_theta = acceptableDistance_theta_Min;
```

```
end

timeSincePwUpdate = toc-lastPwUpdate;
%X-PositionFeedback
arrived=0;
%Check if we're close enough... If so, decelerate
if ((abs(posDesired_r-posCurrent_r)<acceptableDistance_r)&&(abs(posDesired_theta-
posCurrent_theta)<acceptableDistance_theta))
%Decelerate..
arrived = 1;
if (velDesired_x>0)
velDesired_x = velDesired_x - (accel*timeSincePwUpdate); %Update desired velocity
end
if (velDesired_x<0)
velDesired_x = velDesired_x + (accel*timeSincePwUpdate);
end
%Check if we are past the desiredPosition and moving away... If so,
%decelerate
elseif(((posDesired_x-posCurrent_x)*(velDesired_x))<0)
%Decelerate
if (velDesired_x>0)
velDesired_x = velDesired_x - (accel*timeSincePwUpdate); %Update desired velocity
elseif (velDesired_x<0)
velDesired_x = velDesired_x + (accel*timeSincePwUpdate);
end
%See if we're approaching from the correct direction... If so,
%decelerate
elseif ((abs(posDesired_x-posCurrent_x)-abs(velDesired_x*0.15))<=(velDesired_x^2/(2*accel)))
%Decelerate
if (velDesired_x>0)
velDesired_x = velDesired_x - (accel*timeSincePwUpdate); %Update desired velocity
end
if (velDesired_x<0)
velDesired_x = velDesired_x + (accel*timeSincePwUpdate);
end
%This is a small buffer to prevent too much switching by the controller
%If we're CLOSE to approaching, don't do anything
elseif ((abs(posDesired_x-posCurrent_x)-abs(velDesired_x*0.2))<=(velDesired_x^2/(2*accel)))
else %We are far enough away... Pedal to the metal
%Accelerate
%Check if max vel is reached... have to change other DOF too...
if (posDesired_x>posCurrent_x)
velDesired_x = velDesired_x + (accel_x*timeSincePwUpdate); %Update desired velocity
end
if (posDesired_x<posCurrent_x)
velDesired_x = velDesired_x + (accel_x*timeSincePwUpdate);
end
end

%Y-PositionFeedback
if ((abs(posDesired_r-posCurrent_r)<acceptableDistance_r)&&(abs(posDesired_theta-
posCurrent_theta)<acceptableDistance_theta))
%Decelerate..
if (velDesired_y>0)
velDesired_y = velDesired_y - (accel*timeSincePwUpdate); %Update desired velocity
```

```
end
if (velDesired_y<0)
velDesired_y = velDesired_y + (accel*timeSincePwUpdate);
end
%Check if we are past the desiredPosition and moving away... If so,
%decelerate
elseif(((posDesired_y-posCurrent_y)*(velDesired_y))<0)
%Decelerate
if (velDesired_y>0)
velDesired_y = velDesired_y - (accel*timeSincePwUpdate); %Update desired velocity
elseif (velDesired_y<0)
velDesired_y = velDesired_y + (accel*timeSincePwUpdate);
end
%See if we're approaching from the correct direction... If so,
%decelerate
elseif ((abs(posDesired_y-posCurrent_y)-abs(velDesired_y*0.15))<=(velDesired_y^2/(2*accel)))
%Decelerate
if (velDesired_y>0)
velDesired_y = velDesired_y - (accel*timeSincePwUpdate); %Update desired velocity
end
if (velDesired_y<0)
velDesired_y = velDesired_y + (accel*timeSincePwUpdate);
end
elseif ((abs(posDesired_y-posCurrent_y)-abs(velDesired_y*0.2))<=(velDesired_y^2/(2*accel)))
else %We are far enough away... Pedal to the metal
%Accelerate
if (posDesired_y>posCurrent_y)
velDesired_y = velDesired_y + (accel_y*timeSincePwUpdate); %Update desired velocity
end
if (posDesired_y<posCurrent_y)
velDesired_y = velDesired_y + (accel_y*timeSincePwUpdate);
end
end



if(arrived==0)
lastTimeAway = time;
if(posDif<(lastPosDif-20))
lastPosDif = posDif;
timeProgressMade = time;
%               velCalibrationOffset = 0;
elseif(posDif>lastPosDif)
lastPosDif = posDif;
end
if((time-timeProgressMade)>0.5)
velCalibrationOffset = velCalibrationOffset + timeSincePwUpdate;
velCalibrationOffset = min(velCalibrationOffset,10);
end
else
velCalibrationOffset = 0;
end

if(arrived ==1)
if((time-lastTimeAway)>1)
velDesired_x=0;
```

```
velDesired_y=0;
velDesired_theta=0;
pointReached=1;
end
end


%Check Max Velocities
if (velDesired_x>velMax_x)
velDesired_x = velMax_x;
end
if (velDesired_x<-velMax_x)
velDesired_x = -velMax_x;
end
if (velDesired_y>velMax_y)
velDesired_y = velMax_y;
end
if (velDesired_y<-velMax_y)
velDesired_y = -velMax_y;
end

velDesired_r = velDesired_x*cos(posCurrent_theta)+velDesired_y*sin(posCurrent_theta);

%Safety checks
if (velDesired_r>velMax_r)
velDesired_r = velMax_r;
end
if (velDesired_r<(-velMax_r))
velDesired_r = (-velMax_r);
end



velCurrent_r = velDesired_r;
if ((velCurrent_r<=-velMin_r)&&(velCurrent_r>-40))
velCurrent_r=-40;
end
if((velCurrent_r>=velMin_r)&&(velCurrent_r<100))
velCurrent_r=100;
end

%Calculate pwPosition from velDesired...
if (velCurrent_r<=-40)
pwPosition_r = 197+(5.6712*log(-velCurrent_r+100)-28.275);
elseif (velCurrent_r>=100)
pwPosition_r = 182 - (6.2092*log(velCurrent_r-70)-20.973);
else %-100<velCurrent<40
pwPosition_r = pwZero_r;
end
pwPosition_r = round(pwPosition_r);

%% Calculate pwPosition_theta
lastVelDesired_theta = velDesired_theta;
velDesired_theta = (velDesired_y*cos(posCurrent_theta)-velDesired_x*sin(posCurrent_theta))/posCurrent_r;
```

```
%Safety checks
if (velDesired_theta>velMax_theta)
velDesired_theta = velMax_theta;
end
if (velDesired_theta<(-velMax_theta))
velDesired_theta = (-velMax_theta);
end

%     if(abs(velDesired_theta)>abs(lastVelDesired_theta))
accelDesired_theta = (velDesired_theta-lastVelDesired_theta)/timeSincePwUpdate;
velCurrent_theta = (accelDesired_theta/4)+lastVelDesired_theta;
%     else
%         velCurrent_theta = velDesired_theta;
%     end

if((velCurrent_theta*velDesired_theta)>0)
if ((velCurrent_theta<=-velMin_theta)&&(velCurrent_theta>-0.13))
velCurrent_theta=-0.13;
end
if((velCurrent_theta>=velMin_theta)&&(velCurrent_theta<0.35))
velCurrent_theta=0.35;
end
end



%Calculate pwPosition from velDesired...
if (velCurrent_theta<=-0.13)
pwPosition_theta = 175-(4.3809*log(-velCurrent_theta+0.25)+4.2+velCalibrationOffset);
elseif (velCurrent_theta>=0.2)
pwPosition_theta = 194 + (2.997*log(velCurrent_theta+0.15)+3.1272+velCalibrationOffset);
%pwPosition_theta = 194 + (5.997*log(velCurrent_theta+0.15)+6.1272);
else %-100<velCurrent<40
pwPosition_theta = pwZero_theta;
end
pwPosition_theta = round(pwPosition_theta);

%% Safety... And send PW's
%Combine feedback into one pulse-width to send
%pw=pwForce+pwPosition+pwComp;
pw_r=pwPosition_r;
pw_theta = pwPosition_theta;
lastPwUpdate=toc;

if((posCurrent_r<posMin_r)&&(pw_r>pwZero_r))
pw_r=pwZero_r;
end
if((posCurrent_r>posMax_r)&&(pw_r<pwZero_r))
pw_r=pwZero_r;
end
if((posCurrent_theta<posMin_theta)&&(pw_theta<pwZero_theta))
pw_theta=pwZero_theta;
end
if((posCurrent_theta>posMax_theta)&&(pw_theta>pwZero_theta))
pw_theta=pwZero_theta;
end
```

```
if (pw_r<=shutOffMid)&&(pw_r>=shutOffMin)
pw_r=shutOffMin-1;
end
if (pw_r>shutOffMid)&&(pw_r<=shutOffMax)
pw_r=shutOffMax+1;
end
if (pw_theta<=shutOffMid)&&(pw_theta>=shutOffMin)
pw_theta=shutOffMin-1;
end
if (pw_theta>shutOffMid)&&(pw_theta<=shutOffMax)
pw_theta=shutOffMax+1;
end

if (abs(posCurrent_r-posLast_r)>100)
emerStop = 1;
end
if (abs(posCurrent_theta-posLast_theta)>0.1)
emerStop = 1;
end

if(emerStop==1)
pw_r=pwZero_r;
pw_theta=pwZero_theta;
end

if (pw_r<pwMin_r)
pw_r=pwMin_r;
end
if (pw_r>pwMax_r)
pw_r=pwMax_r;
end
if (pw_theta<pwMin_theta)
pw_theta=pwMin_theta;
end
if (pw_theta>pwMax_theta)
pw_theta=pwMax_theta;
end
fprintf('pw_r = %d. pw_t = %d. posDesX = %d. posDesY = %d. posR = %d. posDesR = %d. posT = %d. posDesT =
%d. velDesX = %d. velDesY = %d. stop =
%d.\n',pw_r,pw_theta,posDesired_x,posDesired_y,posCurrent_r,posDesired_r,posCurrent_theta,posDesired_theta,v
elDesired_x,velDesired_y,emerStop)
messageOut=[3,pw_r,pw_theta];
fwrite(a,messageOut);
end
end
```

REFERENCES

[1] G.R. Williams, “Incidence and characteristics of total stroke in the United States,” BMC Neural, vol. 1, pp. 2, 2001.

[2] P. Muntne, E. Garrett E, MJ Klag, J. Coresh. Trends in stroke prevalence between 1973 and 1991 in the US population 25 to 74 years of age. Stroke. vol. 33, pp. 1209-1213, 2002.

[3] Centers for Disease Control and Prevention, Stoke Facts. March 24, 2015. <http://-www.cdc.gov/stroke/facts.htm>.

[4] Rosenberg CH, Popelka GM. Poststroke rehabilitation. A review of the guidelines for patient management. Geriatrics 2000;55:7581.

[5] A. S. Association, http://www.strokeassociation.org/, 2005.

[6] "General Information." Campaign For Cure. International Campaign for Cures of Spinal Cord Injury Paralysis. 2000.

[7] National Spinal Cord Injury Statistical Center, Facts and Figures At a Glance. Birmingham, AL: University of Alabama at Birmingham, March 2013.

[8] "Spinal Cord Injury." WHO. World Health Organization, 1 Nov. 2013.

[9] Masiero S, Celia A, Rosati G, Armani M (2007) Robotic-assisted rehabilitation of the upper limb after acute stroke. Arch Phys Med Rehabil 88: 142–149.

[10] Volpe BT, Krebs HI, Hogan N. Is robot-aided sensorimotor training in stroke rehabilitation a realistic option? Curr Opin Neurol 2001;14:745-52.

[11] Prange GB, Jannink MJ, Groothuis-Oudshoorn CG, Hermens HJ, IJzerman MJ. Systematic review of the effect of robot-aided therapy on recovery of the hemiparetic arm after stroke. J Rehabil Res Dev 2006;43:171-84.

[12] Barreca S, Wolf SL, Fasoli S, Bohannon R. Treatment interventions for the paretic upper limb of stroke survivors: a critical review. Neurorehabil Neural Repair 2003;17:220-6.

[13] Kwakkel G, Wagenaar RC, Twisk JW, Lankhorst GJ, Koetsier JC. Intensity of leg and arm training after primary middle-cerebralartery stroke: a randomized trial. Lancet 999;354:191-6.

[14] Hogan, N.; Krebs, HI.; Sharon, A.; Charnnarong, J., inventors. Interactive robotic therapist. U.S. Patent. #5 466 213. 1995.

[15] Volpe BT, Krebs HI, Hogan N, Edelstein L, Diels C, Aisen M. A novel approach to stroke rehabilitation: robot-aided sensorimotor stimulation. Neurology 2000;54:1938-44.

[16] Volpe BT, Krebs HI, Hogan N, Edelstein L, Diels CM, Aisen ML. Robot training enhanced motor outcome in patients with stroke maintained over 3 years. Neurology 1999;53:1874-6.

[17] D. J. Reinkensmeyer, L. E. Kahn, M. Averbuch, A. McKenna-Cole, B. D. Schmit, and W. Z. Rymer, “Understanding and treating arm movement impairment after chronic brain injury: progress with the arm guide,” Journal of Rehabilitation Research and Development, vol. 37, no. 6, pp. 653–662, 2000.

[18] P. S. Lum, C. G. Burgar, and P. C. Shor, “Evidence for improved muscle activation patterns after retraining of reaching movements with the mime robotic system in subjects with post-stroke hemiparesis,” IEEE Trans. on Rehab. Engineering, vol. 12, no. 2, pp. 186–194, 2004.

[19] T. Rahman, W. Sample, and R. Seliktar, "Design and Testing of WREX," presented at The Eighth International Confrence on Rehabilitation Robotics, Kaist, Daejeon, Korea, 2003.

[20] R. Sanchez, P. Shah, J. Liu, S. Rao, R. Smith, S. Cramer, T. Rahman, J. E. Bobrow, and D. Reinkensmeyer, "Monitoring Functional Arm Movement for Home-Based Therapy after Stroke," presented at Proceedings of the 2004 IEEE Engineering in Medicine and Biology Society Meeting, San Francisco, California, September 1-5, 2004.

[21] Sanchez Jr. RJ, Wolbrecht E, Smith R, Liu J, Cramer S, Rahman T, Bobrow JE, Reinkensmeyer DJ. A pneumatic robot for retraining arm movement after stroke: Rationale and mechanical design. Proceedings of the 9th IEEE International Conference on Rehabilitation Robotics; 2005 June 28 - July 1; Chicago, USA: IEEE press; 2005. p. 500-504.

[22] S. Oh and S. K. Agrawal. Cable-suspended planar robots with redundant cables: controllers with positive tensions. IEEE Trans. Robot., 21:457-464, 2005.

[23] S. Oh and S. K. Agrawal. The feasible workspace analysis of a set point control for a cable-suspended robot with input constraints and disturbances. IEEE Trans. Control Syst. Technol., 14(4):735-742, 2006.

[24] S. Oh and S. K. Agrawal. Generation of feasible set points and control of a cable robot. IEEE Trans. Robot., 22(3):551-558, 2006.

[25] Y. Takahashi and T. Kobayashi, “Upper limb motion assist robot,” in Proceedings of the IEEE 6th International Conference on Rehabilitation Robotics ICORR1999, Stanford, CA, 1999.

[26] S. Kim, M. Ishii, Y. Koike, and M. Sato, “Development of a spidar-g and possibility of its application to virtual reality,” in Proceedings of the 10th VRST2000, Seoul, Korea, 2000, pp. 22–25.

[27] S. Coote, E. K. Stokes, M. B.T., and W. Harwin, "The Effect of GENTLE/s Robot Mediated Therapy on Upper Extremity Function Post Stroke," presented at International Confrence on Rehabilitation Robotics, Korea, 2003.

[28] C. Fanin, P. Gallina, A. Rossi, U. Zanatta, and S. Masiero, “Nerebot: a wire-based robot for neurorehabilitation,” in Proceedings of the IEEE 8th International Conference on Rehabilitation Robotics ICORR2003, Daejeon, Republic of Korea, April 2003.

[29] S. Masiero, A. Celia, V. Perticaro, M. Armani, G. Rosati, P. Gallina, A. Rossi, M. Ortolani, and C. Ferraro, “Robot-aided intensive training in post-stroke recovery,” 2005, editorial review, submitted for publication on Aging Clinical and Experimental Research.

[30] Rosati G, Gallina P, Masiero S, Rossi A: Design of a new 5 d.o.f. wire-based robot for rehabilitation. IEEE ICORR, Chicago; 2005:430-433.

[31] D. Mayhew, B. Bachrach, W.Z. Rymer, and R.F. Beer, “Development of the MACARM – a novel cable robot for upper limb neurorehabilitation,” 9th International conference on Rehabilitation Robotics, pp. 209-302, 2005

[32] T. Nef, M. Guidali, and R. Riener. Armin iii - arm therapy exoskeleton with an ergonomic shoulder actuation. Applied Bionics and Biomechanics, 6(2):127-142, 2009.

[33] J. Klein, S. J. Spencer, and J. Allington. Biomimetic orthosis for the neurorehabilitation of the elbow and shoulder (BONES). In Proc. IEEE International Conference on Biomedical Robotics and Biomechatronics, pages 535-541, 2008.

[34] S. Balasubramanian, R. Wei, M. Perez, B. Shepard, E. Koeneman, J. Koeneman, and J. He. Rupert: An exoskeleton robot for assisting rehabilitation of arm functions. In Virtual Rehabilitation, pages 163-167, August 2008.

[35] A. H. A. Stienen, E. E. G. Hekman, F. C. T. Van der Helm, G. B. Prange, M. J. A. Jannink, A. M. M. Aalsma, and H. Van der Kooij. Dampace: dynamic force coordination trainer for the extremities. In Proc. IEEE International Conference on Rehabilitation Robotics, pages 820-826, 2007.

[36] J. C. Perry, J. Rosen, and S. Burns. Upper-limb powered exoskeleton design. IEEE/ASME Trans. Mechatronics, 12(4):408{417, 2007.

[37] S. J. Ball, I. E. Brown, and S. H. Scott. Medarm: a rehabilitation robot with 5dof at the shoulder complex. In Proc. IEEE/ASME international conference on Advanced intelligent mechatronics, pages 1-6, September 2007.

[38] G. Yang, W. Lin, S. K. Mustafa, C. B. Pham, and S. H. Yeo. Kinematic design of a 7-dof cable-driven humanoid arm: a solution-in-nature approach. In Proc. IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pages 444-449, 2005.

[39] E. A. Brackbill, Y. Mao, S. K. Agrawal, M. Annapragada, and V. N. Dubey. Dynamics and control of a 4-dof wearable cable-driven upper arm exoskeleton. In Proc. IEEE International Conference on Robotics and Automation, pages 2300-2305, May 2009.

[40] Y. Mao and S. K. Agrawal. A cable driven upper arm exoskeleton for upper extremity rehabilitation. In Proc. IEEE International Conference on Robotics and Automation, pages 4163-4168, May 2011.

[41] D. Kulic and E. Croft, "Estimating Intent for Human-Robot Interaction," presented at IEEE International Conference on Advanced Robotics, Coimbra, Portugal, pp. 810-815, 2003.

[42] M. S. Erden and T. Tomiyama, “Human-intent detection and physically interactive control of a robot without force sensors,” IEEE Transactions on Robotics, vol. 26, no. 2, pp. 370–382, 2010.

[43] H. Kazerooni and M. G. Her, “The dynamics and control of a haptic interface device,” IEEE Trans. Robot. Autom., vol. 10, no. 4, pp. 453–464, Aug. 1994.

[44] H. Kazerooni, “The human power amplifier technology at the University of California, Berkeley,” Robot. Auton. Syst., vol. 19, pp. 179–187, 1996.

[45] H. Kazerooni, “Human–robot interaction via the transfer of power and information signals,” IEEE Trans. Syst., Man, Cybern., vol. 20, no. 2, pp. 450–463, Mar./Apr. 1990.

[46] H. Kazerooni, “Human power extender: An example of human–machine interaction via the transfer of power and information signals,” in Proc. 5th Int. Workshop Adv. Motion Control, Coimbra, Portugal, 1998, pp. 565–572.

[47] O. Khatib, K. Yokoi, O. Brock, K. Chang, and A. Casal, “Robots in human environments: Basic autonomous capabilities,” Int. J. Robot. Res., vol. 18, no. 7, pp. 684–696, 1999.

[48] O. Khatib, K. Yokoi, K. Chang, D. Ruspini, R. Holmberg, and A. Casal, “Vehicle/arm coordination and multiple mobile manipulator decentralized cooperation,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Osaka, Japan, 1996, pp. 546–553.

[49] K. S. Eom, I. H. Suh, W. K. Chung, and S. R. Oh, “Disturbance observer based force control of robot manipulator without force sensor,” in Proc. IEEE Int. Conf. Robot. Autom., Leuven, Belgium, 1998, pp. 3012–3017.

[50] K. Ohishi, M. Miyazaki, and M. Fujita, “Hybrid control of force and position without force sensor,” in Proc. IEEE Int. Conf. Power Electron. Motion Control, San Diago, CA, 1992, vol. 2, pp. 670–675.

[51] S. Tungpataratanawong, K. Ohishi, and T. Miyazaki, “Force sensor-less workspace impedance control considering resonant vibration of industrial robot,” in Proc. 31st Annu. Conf. IEEE Ind. Electron. Soc., Raleigh, NC, 2005, pp. 1878–1883.

[52] N. Sarkar. Psychophysiological Control Architecture for Human-Robot Coordination – Concepts and Initial Experiments. ICRA 2002.

[53] R. W. Picard et al. Toward Machine Emotional Intelligence: Analysis of Affective Physiological State. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 23(10): 1175 – 1191, 2001.

[54] R. Picard. Affective Computing. MIT Press. Cambridge, Masssachusetts, 1997.

[55] Chrystopher L. Nehaniv, Kerstin Dautenhahn, Jens Kubacki, Martin Haegele, Christopher Parlitz, and Rachid Alami, ‘A methodological approach relating the classification of gesture to identification of human intent in the context of human-robot interaction’, in 14th IEEE International Workshop on Robot and Human Interactive Communication (Ro-Man 2005), pp. 371–377, (2005).

[56] M.A.T. Ho, Y. Yamada, Y. Umetani, “An HMM-based temporal difference learning with modelupdating capability for visual tracking of human communicational behaviors”, In Proceeding of 5th IEEE International Conference on Automatic Face and Gesture Recognition, pp.163-168, 2002.

[57] W. K. Song et al. Visual Servoing for a User’s Mouth with Effective Intention Reading in a Wheelchair-based Robotic Arm. ICRA 2001, pp. 3662 – 3667.

[58] Billard, A., Epars, Y., Calinon, S., Cheng, G., and Schaal, S. Discovering optimal imitation strategies. Robotics & Autonomous Systems, Special Issue: Robot Learning from Demonstration 47, 2-3 (2004), 69.77.

[59] Kortenkamp, E. Huber, and P. Bonasso. Recognizing and interpreting gestures on a mobile robot. In Proceedings of AAAI-96, pages 915-921. AAAI Press/The MIT Press, 1996.

[60] S. Waldherr, S. Thrun, D. Margaritis, and R. Romero. Template-based recognition of pose and motion gestures on a mobile robot. In Proceedings of the Fifteenth National Conference on Arti.

[61] BREAZEAL, C., AND ARYANANDA, L. Recognition of affective communicative intent in robot-directed speech. Autonomous Robots 12, 1 (2002), 83.104.

[62] IBA, S., PAREDIS, C. J. J., AND KHOSLA, P. K. Interactive multimodal robot programming. In Proceedings of the 2002 IEEE International Conference on Robotics and Automation, Washington D.C., May 11-15, 2002 (2002).

[63] A. Kaufman, A. Bandopadhay, B. Shaviv, An eye tracking computer user interface, in: Proc. of the Research Frontier in Virtual Reality Workshop, IEEE Computer Society Press, 1993, pp. 78–84.

[64] Y. Chen and W. S. Newman. A human-robot interface based on electrooculography. In Proc. of the International Conference on Robotics and Automation (ICRA 2004), volume 1, pages 243–248, April 26–May 1, 2004.

[65] Morimoto CH, Mimica MR. Eye gaze tracking techniques for interactive applications. Comput Vis Image Underst. 2005;98(1):4–24.

[66] S.K. Schnipke, M.W. Todd, Trials and tribulations of using an eye-tracking system, in: Proc. ACM SIGCHI—Human Factors in Computing Systems Conference, 2000, pp. 273–274.

[67] K. Nguyen, C. Wagner, D. Koons, M. Flickner, Differences in the infrared bright pupil response of human eyes, in: Proc. of the Eye Tracking Research & Applications Symposium, New Orleans, LA, 2002.

[68] J. Reulen, j.T. Marcus, D. Koops, F. de Vries, G. Tiesinga, K. Boshuizen, J. Bos, Precise recording of eye movement: the iris technique, part 1, Med. Biol. Eng. Comput. 26 (1) (1988) 20–26.

[69] A. Tomono, M. Iida, Y. Kobayashi, A tv camera system which extracts feature points for non-contact eye movement detection, in: Proc. of the SPIE Optics, Illumination, and Image Sensing for Machine Vision IV, vol. 1194, 1989, pp. 2–12.

[70] Y. Ebisawa, S. Satoh, Effectiveness of pupil area detection technique using two light sources and image difference method, in: A. Szeto, R. Rangayan (Eds.), Proc. of the 15th Annual Internat. Conf. of the IEEE Eng. in Medicine and Biology Society, San Diego, CA, 1993, pp. 1268–1269.

[71] C. Morimoto, A. Amir, M. Flickner, Detecting eye position and gaze from a single camera and 2 light sources, in: Proc. of the Internat. Conf. on Pattern Recognition, Quebec, Canada, 2002.

[72] D. Yoo, J. Kim, B. Lee, M. Chung, Non contact eye gaze tracking system by mapping of corneal reflections, in: Proc. of the Internat. Conf. on Automatic Face and Gesture Recognition, Washington, DC, 2002, pp. 94–99.

[73] ASL. Available from: <http://www.a-s-l.com>.

[74] S.M.I. Inc, Eyelink gaze tracking. Available from: < http://www.smi.de>.

[75] LCTech, The eyegaze development system. Available from: < http://www.eyegaze.com>.

[76] The Eye Tribe. Available from: <http://theeyetribe.com>

[77] EyeWorks. The FOVIO eyetracker. Available from <http://www.eyetracking.com>

[78] Tobii. Tobii Eye X. Available from <http://www.tobii.com/en/eye-experience/>

[79] Clauser, Charles, John McConville, and J.W. Young. "Weight, Volume, and Center of Mass of Segments of the Human Body." National Technical Information Service, 1969.

[80] Beardmore, Roy. "Key Strength." *Key/Spline Strength Calculations*. 19 Jan. 2013. Web. <http://www.roymech.co.uk/Useful_Tables/Keyways/key_strength.html>.

[81] Rojas, Raul. "Models for DC Motors." Unpublished document.

[82] "AltSoftSerial Library." PRJC. Web. 12 Aug. 2015. <https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html>.

[83] Digi International. XBee/XBee-PRO ZigBee RF Modules User Guide, 2015. pp. 12.

[84] AndyMark. “String Potentiometer Kit (no housing).” <http://www.andymark.com/product-p/am-2618.htm>.