Final Report

on the

Design of a Robotic Manipulator for a Wheelchair

Gateway Coalition Program

1999 - 2000

ABSTRACT

The 1999-2000 multi-university design team designed and manufactured a wheelchair mounted robotic arm to assist disabled children. The Gateway Engineering Coalition sponsored this intercollegiate design project. The schools involved in the project included The Ohio State University, Wright State University, and Sinclair Community College. The purpose of this project was dual-faceted: educational and social responsiveness. The benefit for the students was an educational experience in design and prototyping as well as communication. It was socially responsive in the goals of assisting paraplegic children and servicing a market too small to be economically addressed by industry.

The following report begins with background information on the Gateway project and an overview of the preceding three years of robotic arm design. The project consisted of two phases. The incremental design process and the phases are explained after the background section. Then, the design of each portion of the actual arm (base, arm, gripper, and control system) is presented from brainstorming ideas to prototype testing. A cost analysis is included because keeping the cost of the arm small was a primary goal of this project. Finally, recommendations from the experience of this year's team are offered in hopes that with a few improvements, a manipulator of this sort can be produced safely and economically to assist those that are wheelchair-bound.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The 1999-2000 Gateway Coalition group would like to thank the National Science Foundation and Invacare for the primary funding to this project. Without their help and support, this project would have still been only an idea.

Special thanks to the faculty advisors, Dr. Gary Kinzel (Ohio State University), Dr. Ramana Grandhi and Dr. James Menart (Wright State University), and Dr. Beth Johnson (Sinclair Community College) for their indispensable advice at every step of the project.

Many thanks to Keith Rogers, the “in-house” machine shop wizard (OSU). He has provided advice and helped us through many a machining dilemma. If anyone deserves a fine cup of Colombian coffee, it is Keith. Twelve ounces of gratitude go to the electronics guru, Joe West (OSU). He made our printed circuit boards, designed our first layout, and was generally a great help to us. Chris Fearon (OSU) acted as our student advisor and was always available to answer questions. Good luck to him on his Master’s degree endeavors, optimization of a robotic manipulator.

Sudhakar Mahalingam (WSU graduate student) deserves a big share of our appreciation for his endeavors in helping us overcome computer related hurdles. Greg Wilt (WSU) helped us in more than one way during the project, and we would like to thank him for his time and effort. We thank Motoman (Dayton, OH) for their time invested in our project in the form of plant tours and advice on the design of various aspects of the robotic arm.

Last, but not the least, we would like to thank Scott (Sinclair Community College) for his machining expertise and help in manufacturing various parts of the arm.

Sincerely,

Gateway Coalition

2000 Design Team

1 PROJECT OVERVIEW

1.1 GATEWAY COALITION PROGRAM BACKGROUND

The Gateway Coalition, an organization supported by the National Science Foundation, is a collaboration of seven institutions dedicated to improving engineering education. One of the programs sponsored by the Coalition is a multi-university senior design project. This year the senior design project was comprised of students and faculty from three schools: The Ohio State University, Wright State University, and Sinclair Community College. The goal this year, as for the previous three years, was to develop a wheelchair-mounted robotic device that paraplegic children can use to assist them in their daily lives.

1.2 DESIGN GOALS

There have been several attempts made in industry to design a wheelchair-mounted robotic arm that is both easy to use and reasonably priced. Currently, the Helping Hand and MANUS are two of the most popular commercial models of the orthotic arms that are available. Both of these models are relatively expensive and are difficult to control by many disabled people. The main purpose of this project is thus to develop a low-cost and easy-to-use robotic manipulator that will assist children who are confined to wheelchairs and have limited use of their arms and hands.

Although great strides have been made, efforts by the Gateway Coalition in the past three years have fallen short of producing a practical design that is usable by children with disabilities. Following is a brief history of the previous three year's designs.

1.2.1 1996-1997 Design

Figure 1.1 is a picture of the 1996-97 final design. This device was a 3 link, 6 degree of freedom device with a transmission system consisting of cables and transmission pulleys. The purpose of the transmission pulleys was to keep the motors on the base of the robot, thus decreasing the torque on the arm. Unfortunately, a mistake was made in the initial torque analysis, so inappropriate motors were selected, which were not strong enough to easily move the arm. Its overall size complicated manufacturing requirements, and predicted high maintenance costs was another shortcoming of this design.

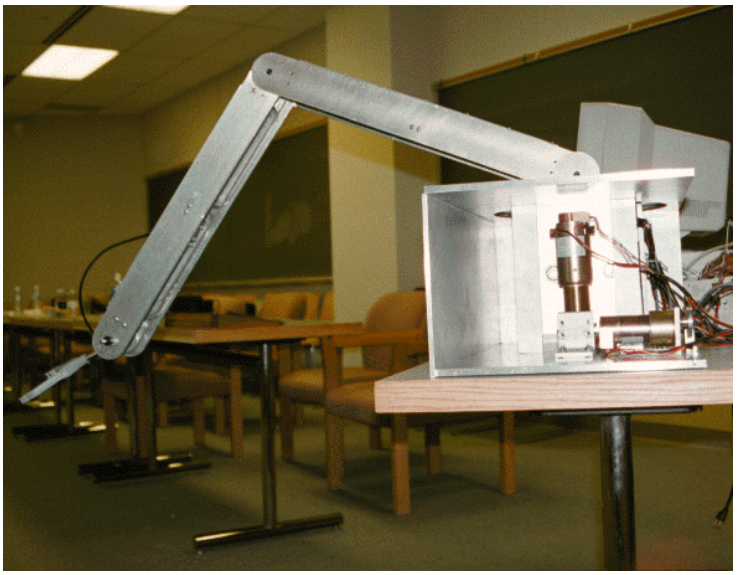


Figure 1.1 - 1996-1997 final design.

1.2.2 1997-1998 Design

An image of a design concept proposed by the 1997-98 Ohio State team is seen in Figure 1.2. This design featured 5 degrees of freedom, with vertical motion controlled by a lead screw and horizontal motion accomplished through the use of a motor at the elbow motors are mounted directly on each joint (motor-on-joint control). Another feature of this design is that it utilized more off the shelf parts such as a rotating base. Advantages of this design over the cable and pulley system were that it has lower assembly and maintenance costs. A drawback however, was that the required off the shelf components were quite expensive. It was also questionable whether the structure

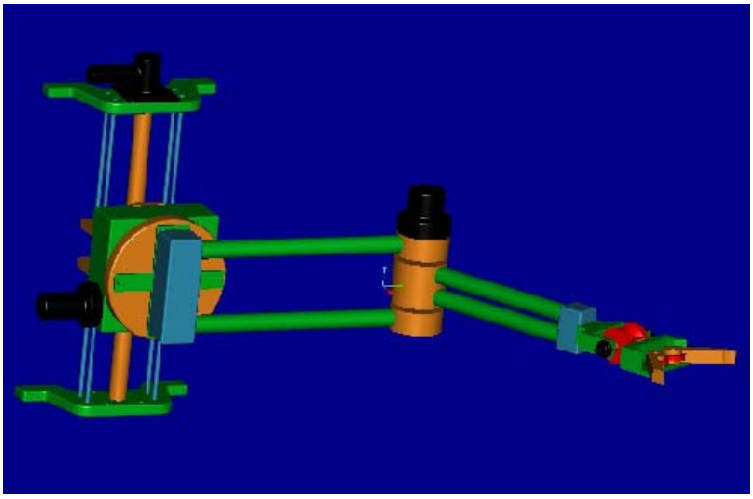


Figure 1.2 - Ohio State's 1997-1998 initial design.

of the arm was rigid enough to support the torque produced by both the object to be picked up and the motors required to manipulate the arm joint. This design differed significantly from the first design in that no cables or pulleys were used, since the

The final 1997-98 design (pictured in Figures 1.3 and 1.4) incorporated a transmission system similar to that of the previous year's design. Some of the improvements made over the 1996-97 design are the addition of a knuckle joint and a more compact base that actually rotated. Although this design was structurally and functionally better than the previous year's design, it was very expensive due to the large amount of machining required, and it was never mounted to a wheelchair. Another drawback of this design is that the gripper could only be actuated in one direction. It was spring-loaded in the open position and closed as a cable was wound with a small motor.

Figure 1.3 - 1997-1998 final design (folded position).

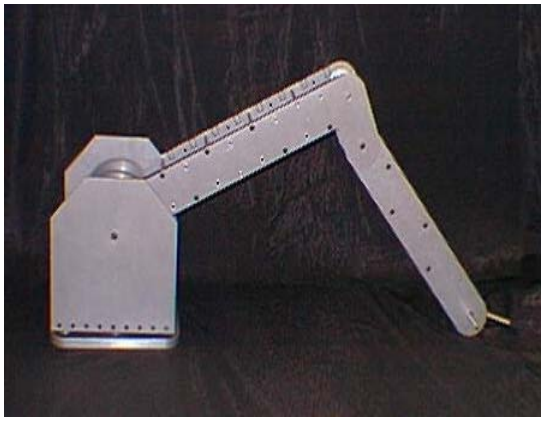


Figure 1.4 - 1997-1998 final design (extended position).

1.2.3 Design Parameters for 1999-2000 Design

At the kick-off meeting in September 1999 at The Ohio State University, representatives from each participating university met to discuss how the project would be approached for the 1999-2000 manipulator design. At this meeting, participants agreed upon several general design goals based on the two years of prior experience with the project. By reviewing the benefits as well as shortcomings of these previous attempts to construct a fully functional and marketable robotic arm, we were able to narrow our design goals quickly and focus on the specific design and manufacturing areas that required the most improvement.

Some of this year's design goals were adapted from last year's, such as the basic kinematic layouts and the user interface and control systems. Other design goals for this year were to reduce the manufacturing costs of the arm, increase the range, improve structural aspects and gripper design, and mount the entire assembly to a wheelchair for testing and evaluation. Specific design parameters for the 1998-99 academic year were as follows:

- 6 degrees of freedom (5 for the arm, 1 for the gripper)
- Knuckle joint for elbow (See Figures 1.3 and 1.4)
- Motor-on-joint arm
- Maximum weight of the entire assembly to be 30 lbs.
- 6 degree of freedom trackball / joystick user interface
- Control gripper movement in both directions
- Each linkage must move no more than 0.5 m/s
- Lift 1 kg of mass (2.2 lbs.) with 7.5 cm in largest dimension
- Maximum cost of $4,500 (excluding controls)

1.3 DESIGN PHASES

Through the use of the Internet and video/tele-conferencing, student groups compared their own design concepts and coordinated the manufacturing of the robotic arm in two major phases. In Phase I, each team independently developed a design for the arm assembly using the design criteria specified at the kick-off meeting in

September 1999. Then, in January, a meeting was held to compare the independent design proposals from each school. Based upon those design proposals, student groups decided upon a final design and delegated responsibilities for component developments to each school, which signified the beginning of Phase II.

As a result of the discussion among the schools, The Ohio State University was selected to develop the forearm and gripper portion of the arm. The Ohio State University also took responsibility for the arm controls as well. Wright State University was selected to be in charge of developing the upper arm and base structure. Sinclair Community College collaborated with Wright State University and completed the manufacturing of that portion of the arm.

1.4 COMMUNICATION TECHNIQUES

For an inter-collegiate project of this magnitude to function, communication techniques must be effective. Technical information must be shared continually despite significant geographical separation distances and conflicting schedules.

The participating teams met several times in person over the course of the year, and communications continued on a weekly basis. Electronic mail became the most suitable means for inter- and intra-school communication for project management, as well as technical information sharing. Each school also maintained its own Web page in which current design information and drawings could be found. These, of course, also invited the public to learn about the project and the participants.

In addition to sharing decisions made by individual teams, participants of the entire Gateway team were required to agree mutually on certain project management and design specifics. Weekly tele-conferences were scheduled originally to accomplish this. These meetings were supplemented with the Net-based videoconference software CU-SeeMe. Although Net-based videoconference material is inexpensive, it is limited by the speed of the data transfer. Thus, the video and sound that were transferred were not of sufficient quality for an efficient meeting. The tele-conference became the multi-media exchange of choice as the participants referenced drawings posted on respective Web-pages.

1.5 PARTICIPANTS

THE OHIO STATE UNIVERSITY

Fred Griesemer
Matthew Hunt
Bruce Isler
Douglas Rutter
Robert Siston

SINCLAIR COMMUNITY COLLEGE

Steve Childers
Pat Oscar
Chris Rhinehart
Brian Shively
Lynn Stonerock

WRIGHT STATE UNIVERSITY

Jenny Broering
Michael Hill
Rahul Shah
Michael Wasco

1.6 REPORT CONTENTS

The report breakdown is outlined in the table of contents, table of figures, and table of tables. The report moves through the sections of base, upper arm, forearm, gripper, and controls system design. Finite element analyses are then discussed, a commentary is given on manufacturing, future recommendations are given, and finally the report presents its appendices. The appendices include control code and other programs written for design analysis and optimization, parts and cost lists, and final design revisions. The information presented in the appendices is a supplement to this report, intended to aid future designers who work on a similar robotic manipulator project (for example, use the programs for design optimization).

2.1 BRAINSTORMING AND RESEARCH

The Phase I design of the base was rather simple and was not capable of a swivel motion. The initial design of the base is shown in Figure 2.1. The original assumption was that the wheelchair could be moved to replace the need for the arm base rotation. However, at the January meeting that ended Phase I and begun Phase II it was decided that a swivel motion would be required to make the arm more marketable.

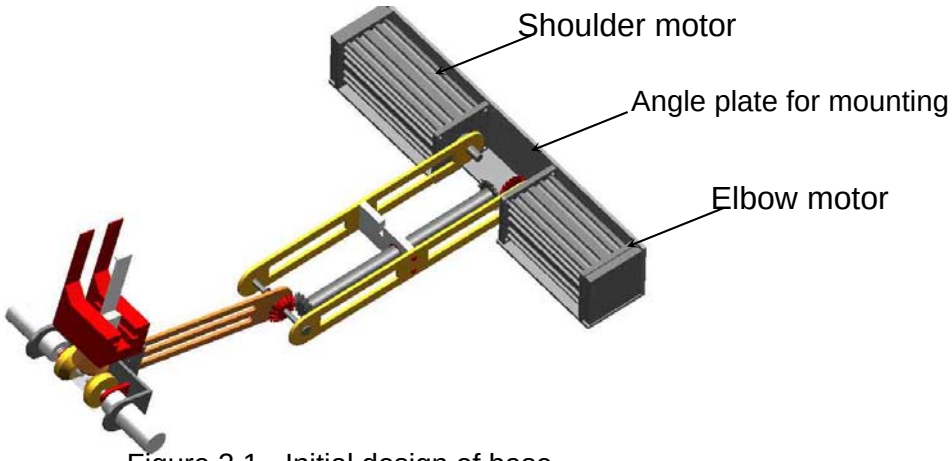


Figure 2.1 - Initial design of base

The addition of a swivel motion brought about many changes in the design shown above. These changes were:

- The shoulder motor was changed from being horizontal to vertical.
- An additional "swivel-plate" was added on which the entire arm assembly was placed.
- A motor to drive the swivel plate was added.
- A thin-section bearing was placed in between the two plates. This thin section bearing was chosen for its ability to take large moments as well as axial loads. The bearing was held in place with collars.

- Other modifications included the changes in profiles of the machined parts in order to accommodate other parts.

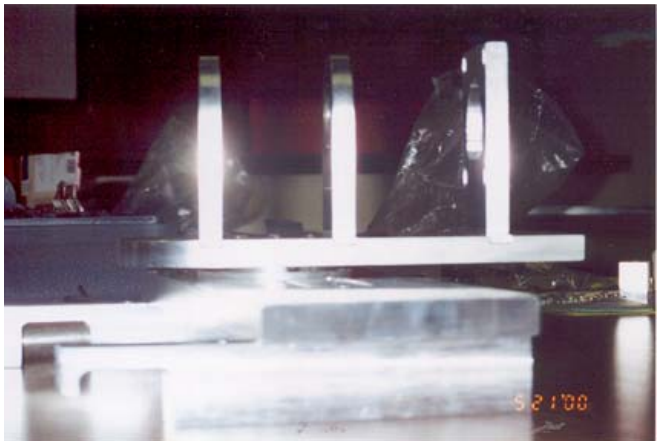


Figure 2.2 - Final base design.

Figure 2.2 shows the complete base plate assembly. The overall base design is explained in detail in the following sections.

2.2 MOUNTING CLAMPS AND MOUNTING METHOD

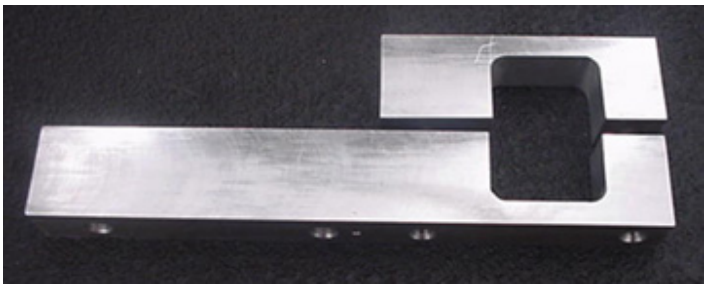


Figure 2.3 - Mounting Clamp

The most structurally sound part of the wheelchair is the rectangular cross-sectional beam that makes up its frame. For this reason the arm is attached to this part of the wheelchair. The wheelchair frame runs between the front and back wheels of the chair and thus makes mounting the robotic arm to the side of the wheelchair possible.

Two custom made clamps are used to connect the stationary base plate of the arm to the frame of the wheelchair. The clamps consist of two pieces; one part called the "long piece" is bolted directly to the stationary plate while the other part called the "short piece" is bolted to the long piece of the clamp, with the frame of the wheelchair being between the short and long pieces. Figure 2.3 shows a photograph of the clamps

and Figure 2.4 shows a solid model of how the clamps are used to attach the stationary base plate to the frame of the wheelchair. The frame of the wheelchair is the red rectangular bar at the front, right-hand side of the drawing. The stationary plate is the pink, longer plate blow the gray swivel plate. The gray swivel plate is the plate that is located in the top of the drawing.

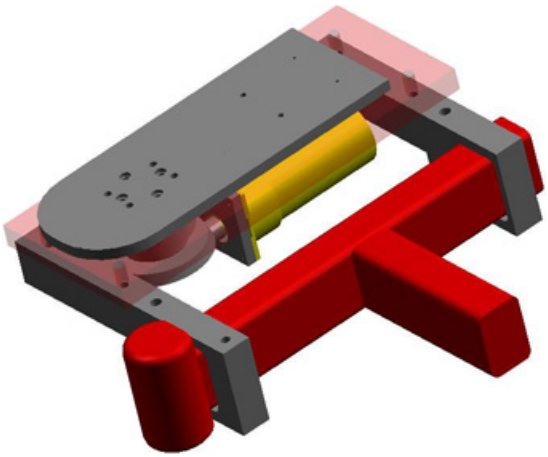


Figure 2.4 - Clamp Attachment to Wheelchair

This type of clamp design offers an improvement over last year's model in that it clamps the beam from the top as well as the bottom in two different places. The clamps were designed to the contours of the beam to provide a good fit. To keep the clamps from scratching the frame of the wheelchair, rubber is fixed to the insides of the clamps.

2.3 SWIVEL DESIGN & MOUNTING PLATES

The stationary plate shown in Figure 2.5 acts as the interface between the arm and the wheelchair. It is the main support for the robotic arm. All parts of the robotic arm, except the short pieces of the clamps are attached to the stationary plate. This

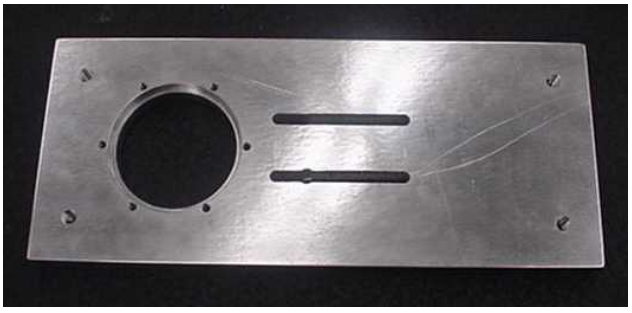


Figure 2.5 - Stationary Plate

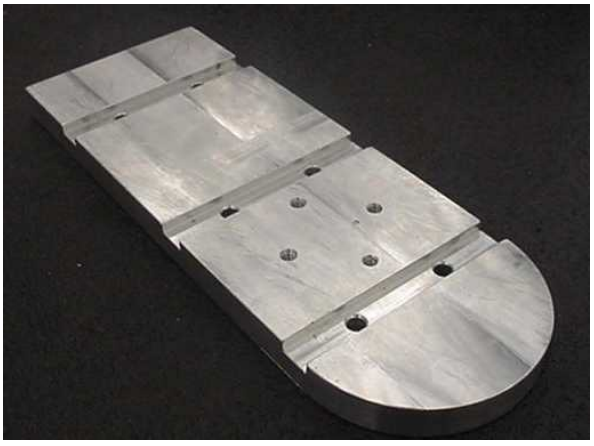


Figure 2.6 - Swivel Plate

makes it possible to remove and attach the arm to the chair as a unit.

As the base of the robotic arm was being designed, an additional criterion was realized: the base should not extend more than six inches from the edge of the

outermost part of the wheelchair. The reason for this constraint is that an average sized doorway is about 36 inches wide, while the wheelchair is approximately 30 inches in width. So that the wheelchair, with the arm mounted to it, can fit through a standard doorway the width of the base can not exceed six inches. The stationary plate is a 0.75 inch thick flat plate that is 5.5 inches wide and 13.25 inches long with the appropriate holes and slots machined into it. When the robotic arm is attached to the wheelchair the total width is less than 36 inches. The large hole shown in the base plate in Figure 2.5 holds the swivel bearing which is discussed below.

Directly on top of the base plate is another flat plate called the swivel plate (see Figure 2.6). The swivel plate is attached to the base plate via a bearing. This allows the arm to have a swivel motion about a vertical axis. For the purpose of this report, this will be referred to as the swivel- or S-axis.

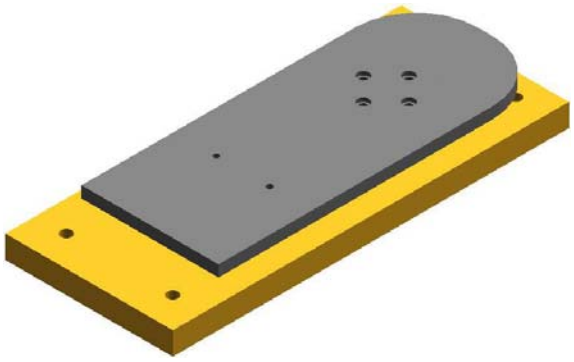


Figure 2.7 - Drawing of Multiple Plate Design

The purpose of the S-axis motion is to give another degree of motion to the arm. The addition of this extra degree of freedom was seen as being necessary for commercial viability.

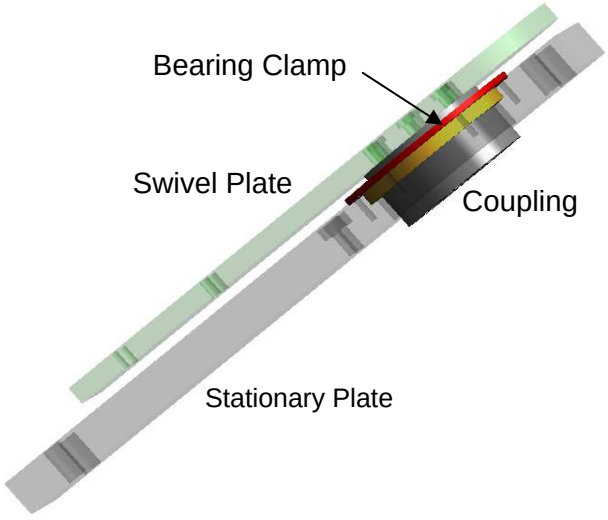


Figure 2.8 - Plate/Coupling Design

The other possibility for obtaining this S-axis motion is to rotate the wheelchair itself. After a great deal of deliberation, it was decided that the arm should possess an S-axis motion and not be dependent on the movement of the wheelchair. In adding this extra degree of freedom to the arm, the complexity and cost of the arm was increased. A swivel motion at the shoulder joint means the addition of a costly bearing and motor. The addition of another motor also increases the complexity of the control system required. In addition the

shoulder motor now has to swivel with the arm and must be placed on the swivel plate. To implement the swivel motion the two-plate design discussed above was incorporated into the base of the arm (see Figure 2.7). This is an alteration of the base design performed in Phase I as shown in Figure 2.1.

A shaft-type coupling is used to connect the swivel plate and stationary plate (see Figure 2.8). The coupling consists of two pieces: the upper-bearing collar and the lower-bearing collar (see Figure 2.9). The upper-bearing collar is attached to the swivel plate and extends about halfway through the inner race of the swivel bearing. The lower bearing collar acts as the interface between the shaft and the swivel gearing system,

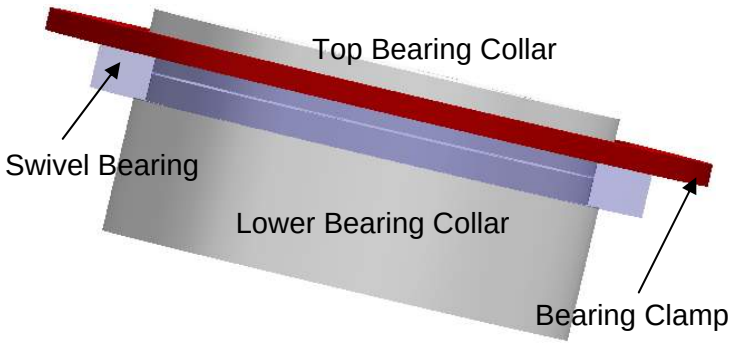


Figure 2.9 - Bearing/Collar Design

and extends about halfway through the inner race of the swivel bearing from the opposite direction as the upper bearing collar. Four bolts connect the swivel plate, upper bearing collar, and lower bearing collar together. This allows the collars to "squeeze" the bearing and smoothly transfer the motion. The reason for the two-collar design, as opposed to a single shaft design, is to keep the swivel plate from moving up and out of the stationary plate. The two collars squeeze the assembly together, while the clamp holds the bearing stationary in the vertical direction.

The swivel bearing chosen was a Kaydon JB025XP0 four-point contact ball bearing. This bearing has dynamic radial, axial, and moment load ratings of 517 lbs., 1293 lbs., and 727 in-lbs., respectively. Support of the swivel plate requires a bearing that can withstand a large thrust and moment. The swivel bearing is housed in the stationary plate (see Figure 2.5). The outer race of this bearing is press-fit into the stationary plate and is held in place by a bearing clamp (see Figure 2.8).

The swivel motor is mounted to the underside of the stationary plate. This is a small motor that only needs to provide enough torque to overcome friction in the swivel bearing. This motor is secured to a vertical mounting plate attached to the underside of the base plate. The mounting plate supports and aligns the motor so that the swivel gears mesh correctly. Attached to the motor is a pinion gear called the swivel pinion. The swivel pinion meshes with a swivel gear that is attached to the lower bearing collar by four bolts.

3.1 BRAINSTORMING AND RESEARCH

The initial idea for the upper-arm was to use square tubing (which was the same as last year) and place all the parts within the tubing. This idea was discarded and the idea of using two plates was implemented. The advantages of using two plates are as follows:

- The plates can easily be customized to incorporate different parts.
- Thicker plates can easily be obtained to increase strength. Thicker walled square tubing is difficult to find.
- The plate design makes it easier to work on the upper arm once the parts are assembled.

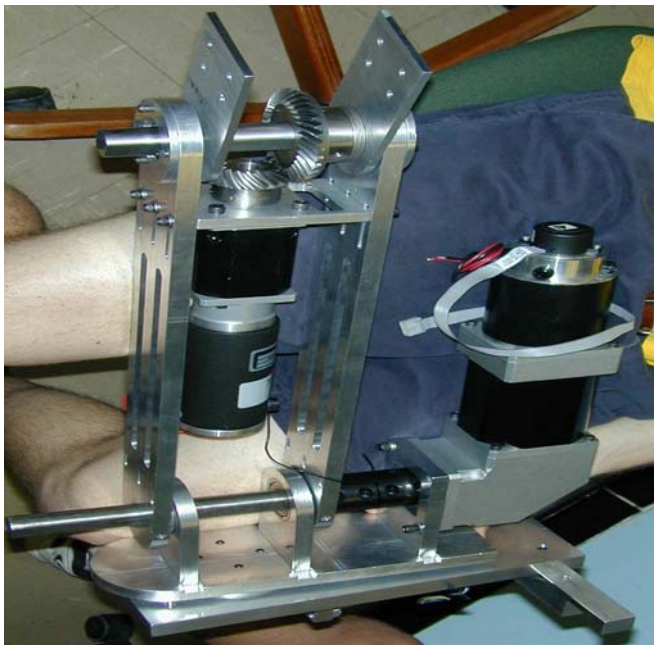


Figure 3.1 - Final design of upper-arm.

The upper arm consists of three major sections discussed in detail in the following sections: the shoulder joint, the arm links, and the elbow joint.

3.2 MATERIAL SELECTION

One of the very first decisions that had to be made in the design of the upper arm was what material it would be produced from. To keep things simple, it was decided that the entire WSU assembly would be made from the same material. This means the material chosen for the upper arm is also used for the base. Aluminum 2024 was chosen due the fact that it has an excellent strength-to-weight ratio.

Weight is a major issue for the moving portion of a robotic arm, therefore materials with high strength-to-weight ratios had to be used. The larger this ratio, the less material the arm requires for support; thus minimizing the weight. The chart below shows the strength-to-weight ratio of a number of materials that were considered.

	Aluminum			Steel (ASTM)		Other
	7075	6061-T6	2024	A242	A572	Titanium
Weight Density (lb/in^3)	0.101	0.098	0.101	0.28	0.28	0.16
Yield Strength (ksi)	73.00	35.00	47.00	50.00	42.00	120.00
Strength-to-Weight	0.723	0.357	0.465	0.176	0.148	0.745

Table 3.1 – Comparison of prospective materials.

One of the problems with many materials that have high strength-to-weight ratios is their cost. Aluminum 2024 is less expensive than materials such as titanium and aluminum 7075, which also have a high strength to weight ratios. Since aluminum 2024 helps keep the weight to a minimum and also keeps the cost low, it is the material used to construct the arm.

Using aluminum for the arm structure has another advantage. Since aluminum is relatively soft, machining is much easier and carbide tooling is not required to make the parts. Machining time and cost is therefore reduced over that required if a hard steel or more exotic material were used.

While all the machined parts in the upper arm assembly are made from aluminum, the off-the-shelf items that were purchased from different manufacturers are not. Many of the power train parts are made from steel because that is what was commercially available. Also note that the forearm uses a variety of different materials for its construction (see chapter 4 for details).

3.3 SHOULDER JOINT

The shoulder joint was very crucial to the design of the robotic arm. The action of moving the entire arm up and down is the main function of the shoulder joint. To start the explanation of the shoulder joint, the shoulder motor search will be examined and then the actual shoulder joint will be discussed.

First of all, the motor search was limited to DC servomotors. The past year's project used a DC stepper motor to provide the shoulder movement instead of using a DC servomotor. Generally speaking, stepper motors have a better holding torque than servo motors and can even output a larger amount of torque in a smaller, lighter package. However, from a controls standpoint, it is easier to design the controls if all the motors are servo type instead of a mixture of stepper and servo. For this reason the search was targeted to servomotors only.

Torque and speed calculations were performed in order to choose the proper motors for the project (see Appendix D). It was initially found that 393 in-lb of torque were needed for the shoulder motor. This torque requirement includes an application factor of 1.5. Also, it was found that this motor could not rotate faster than approximately 2 rpm. The wheelchair's battery pack supplies 24 volts to run all of the motors involved on the arm. Based on these three criteria, a motor and gearhead were located from Servo Systems, Inc.

The combination of the chosen shoulder motor and gearhead provides 477 in-lb. of torque at a speed that can be set in the control algorithm. This is slightly above the required 393 in-lbs. The shoulder motor is a K & D magmotor (see Figure 3.2). This is a DC servo motor with an optical encoder, model # C33-I-200E08. The right angle gearhead has a 100:1 ratio and is a model # Carson 34RE100 NEMA 34 right angle gearhead (see Figure 3.2). The right angle gearhead was chosen over a straight gearhead to eliminate the need for separate gears to be placed on the output shaft. Furthermore, the right angle gearhead was needed because the orientation of the shoulder motor was vertical. The shoulder motor was placed vertical to accommodate the swivel design

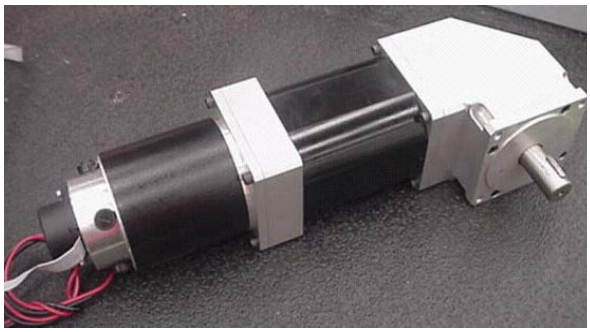


Figure 3.2 – Shoulder motor and right angle gearhead.

as explained in Section II.C.

The K & D magmotor chosen to move the shoulder joint was by no means the only possible choice for this application. Several other motors were found that could provide the required torque; however, some weighed 30 pounds or more and some cost in excess of $3,000. After the selection was made, it was suggested that cordless drill motors would be an excellent source of power for moving the shoulder joint.

The shoulder motor is attached to the swivel plate by means of a support bracket called the shoulder motor support bracket (see Figure 3.3). This bracket is mounted to the swivel plate by means of two bolts. The holes in the swivel plate through which these bolts are inserted were counter-bored to prevent screw heads from sticking out from the bottom of the swivel plate. In addition a groove is machined into the swivel plate (see Figure 2.5) so that the shoulder motor and gearhead are properly aligned with the shoulder shaft.

Figure 3.3 - Shoulder motor support bracket.

Figure 3.4 - Base mounting bracket (left shown).

The actual shoulder joint is composed of two brackets and a shaft. The two brackets that support the shaft are called the base mounting brackets. The left mounting bracket is shown in Figure 3.4; the right mounting bracket is similar. These brackets were constructed so that the pivot point of the shoulder joint is 3 inches from the top of the swivel plate. Raising the pivot point of the arm gives the upper arm a greater rotational range. The base mounting brackets contain a large hole to house the bearings that fit on the shoulder shaft. Both mounting brackets are fastened to the swivel plate with two bolts.

Like the shoulder motor support bracket, the holes made for the mounting brackets were counter-bored to prevent screw heads from sticking out beneath the swivel plate.

The shaft made for the shoulder joint is shown in Figure 3.5. This 1 inch shaft was made with flats on both ends so that the arm links can not rotate relative to the shaft. As mentioned before, the shaft was fitted with two double shielded ball bearings having an outside diameter of 1.5 inches. These ball bearings were also ordered from McMaster-Carr (model # 60355K79). To keep the ball bearings in the mounting brackets and from moving along the axis of the shaft, aluminum bearing clamps were made and placed on the outside of the base mounting brackets. These bearing clamps are flat aluminum disks.

To transfer the motion from the shoulder motor gearhead to the shoulder shaft, a coupling was purchased from McMaster-Carr (model #61005K56, see Figure 3.6). Since the shaft coming out of the gearhead was 3/4 inch and the shoulder shaft was 1 inch, the coupling also acts as an adapter. The coupling is very large and could probably be reduced in size for future designs.

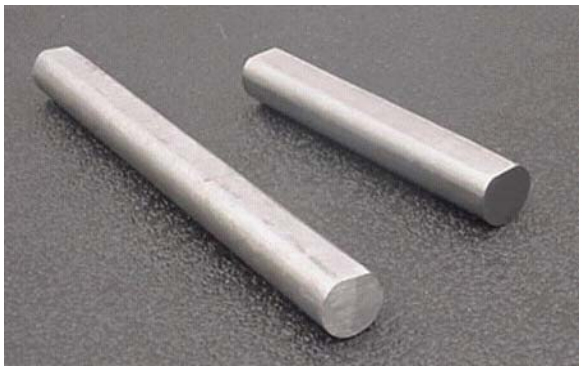


Figure 3.5 – Shoulder shaft (left) and elbow shaft (right).

Figure 3.6 – Shoulder joint coupling.

3.4 ARM LINKS

The arm-links (also known as the arm-plates) are the structural members of the upper arm. Most of the loads and forces experienced by the arm act on these plates. Thus the arm-links are one of the most important components of the whole arm. Extensive finite element analyses were carried out on these plates to make sure that the arm can support the torque generated by the shoulder motor, the weight of the arm, and the weight of the payload, as well as impact loads that might result from the user's running into obstructions.

Figure 3.7 - The upper arm link.

The arm-links have been designed for maximum strength while trying to keep the weight at a reasonable value. Figure 3.7 shows the left arm-link; again, the right arm-link is similar.

3.5 ELBOW JOINT

The elbow design contains 10 different parts. Starting with the two elbow brackets, these are the links that connect the forearm to the upper arm (see Figure 3.8). These brackets are of a fairly simple design. They are 3/16 inch thick, 3 inches wide, 6 inches long and weigh about 0.6 lb. The length of the elbow bracket was determined by finding the minimum amount of clearance required to allow the forearm tube to rotate around the spiral bevel gear (see Figure 3.9). Each elbow bracket has four holes in one end to

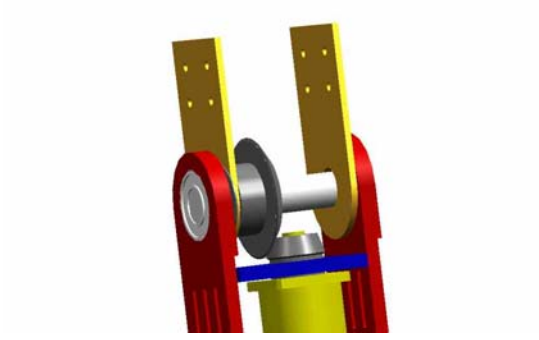


Figure 3.8 – Elbow brackets used to connect upper arm to forearm

Figure 3.9 – Spiral bevel gear and pinion used at the elbow

connect to the forearm and one hole with a flat in the other end. Because the upper arm is wider than the forearm spacers are used in the connection between the elbow brackets and the forearm. One 0.5 inch thick spacer is used for each the four holes in the elbow bracket used to connect to the forearm. These spacers center the forearm between the two upper arm brackets.

The steel shaft that runs through the upper arm links and the elbow brackets was purchased from McMaster-Carr and has a diameter of 1 inch (see Figure 3.5). This shaft is large enough to accommodate the large bore size of the spiral bevel gear used at the elbow joint. The gears chosen where Boston gear's SS102 spiral bevel gear set with a 2:1 ratio (see Figure 3.9). The spiral bevel gear has a 1 inch bore and a pitch diameter of 3.40 inches. The pinion gear has a bore of 5/8 inch and a pitch diameter of 1.7 inches. The reason these gears are so large is that at the time of purchase it was thought that the elbow joint was subject to 150 in-lb of torque. This is actually about twice that which is needed, and these gears can be made smaller in future designs. The purchased gears are rated for 176 in-lb at 100 rpm. Also, the elbow motor has a shaft of 5/8 inch, which fits the pinion perfectly.

In order to keep the shaft from bending it is connected to the arm links via two bearings. These bearings are ABEC-1 R16's, which can handle a load of 2,260 lb. This is more load than then will ever be applied to them.

Figure 3.10 – Elbow motor with gearhead.

The elbow motor is a 23SMDC-LCSS-brush type DC motor that provides 2.9 in-lb of torque. In order to boost the torque output of the motor, a 40:1 Carson 23EP040 size 23 NEMA gearhead is attached to it. This motor-gearhead combination provides 116 in-lb of torque. Including the 2:1 gear ratio from the Boston gears a maximum torque of 232 in-lb is available at the elbow. According to the torque calculations, the elbow joint requires only 88 in-lb, therefore the elbow

motor provides more than enough torque and can be downsized in future designs.

In order to support the weight of the elbow motor, a support plate was designed (see Figure 3.11). This plate is bolted to the shaft side of the motor and to both the upper arm links. The upper arm links have slots cut for these bolts so that the position of the motor can be adjusted. This is necessary so that the pinion gear attached to the motor shaft properly meshes with the spiral bevel gear attached to the elbow shaft.

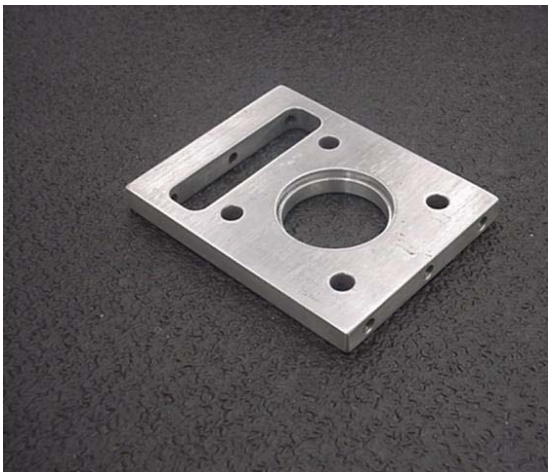


Figure 3.11 - Elbow motor support plate.

4.1 FOREARM TUBING

The forearm section of the arm extends from the elbow joint to the differential gear portion of the wrist. The forearm portion of the manipulator allows two degrees of freedom (bending and twisting), both in the wrist. The forearm design concept is fairly simple, and most of the design comes from last year's robot. However, much thought has gone into this year's design and many revisions have been made to last year's design. Initially, an I-beam was considered for the supporting structure of the forearm to increase strength and make assembly easier. However, further analysis showed that an I-beam was heavier than a simple piece of square aluminum tubing. This added weight causes higher torques at the elbow and shoulder motors. As a result, square tubing was chosen for the supporting structure of the forearm. The square tubing chosen for this project was 2.5 X 2.5 inch aluminum with 1/8 inch thick walls. This tubing was easy to machine, lightweight, and strong enough for this application. In addition, the components of the forearm could all fit inside the tubing. Another benefit associated with the square tubing is that it encloses all of the moving parts, protecting the user from possible injury.

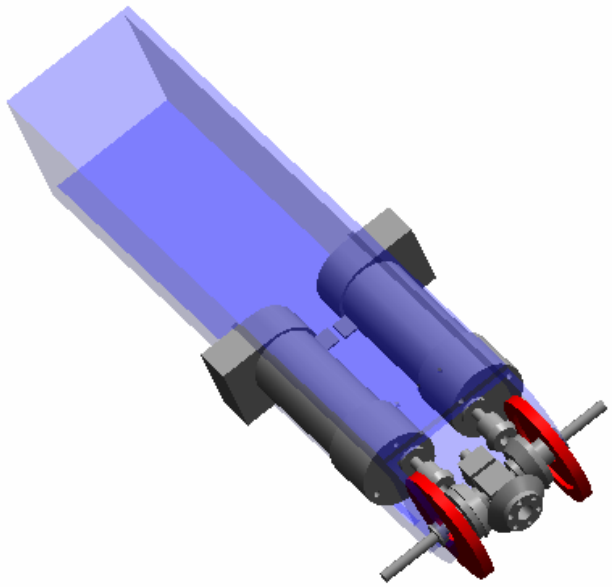


Figure 4.1 - Isometric View of Forearm.

The wrist end of the forearm tubing was designed to allow for a large range of motion at the wrist while providing enough strength to resist failure due to side impact

forces, which could occur if the arm were bumped into something while maneuvering. In initial designs, the end of the tubing was designed to be a semicircle, but this shape did not permit a large enough range of motion in the bending direction. A triangular end was subsequently designed, and this shape was chosen to allow the wrist to have a range of motion of about 120° in bending. Drawings of the two designs can be seen in Figures 4.2a and 4.2b.

Figure 4.2a - Semicircular design.

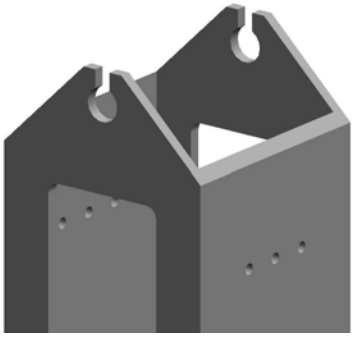


Figure 4.2b - Triangular design.

A picture of the actual forearm tubing, which shows the final triangular end, can be seen in Figure 4.3. The slots that were machined in the forearm tubing to allow for the wrist motor placement can also be seen in Figure 4.3.

Figure 4.3 - Triangular tubing at wrist joint.

The triangular shape significantly increased the range of motion of the wrist, but it also created some concern that the tubing would yield too easily if the arm were accidentally subjected to a side load. A finite element analysis was performed on the triangular end of the tubing with a 40-lb side load applied at the bearing hole for the differential gear, and the resulting stresses were fairly low (8000 PSI), as can be seen from Figure 4.4. The high stress concentration (10000 PSI)

in Figure 4.4 at the differential gear shaft bore is due to the placement of the side load and should be ignored.

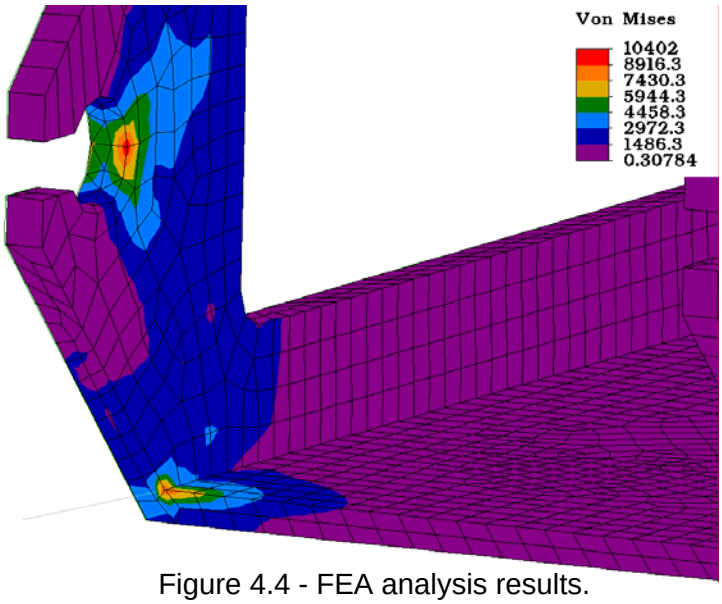


Figure 4.4 - FEA analysis results.

The configuration shown in Figures 4.1 and 4.3 has the wrist bevel gears bolted directly to the differential gear. This was the most compact arrangement possible, and this arrangement fits inside the 2.5 X 2.5 inch tubing. However, it was necessary to flip the wrist motors' encoders by 180° so that the motor shafts would be close enough together to accommodate this arrangement. This was a fairly simple process: the encoders were taken apart, a small notch was machined off of the casing between the encoder and the motor, and the encoders were flipped 180°. The motors chosen to drive the wrist did not fit completely inside the tubing, so slots were cut into the sides of the tubing to accommodate the motors as well as to ease the assembly process. Holes for the flange bearings that support the differential gear were also cut, and small slots were cut in the end of the forearm. This allows the differential gear to be slid into place, and then the flange bearing can be put into place to stabilize the differential gear.

The hole pattern that matches the elbow bracket was established by Wright State, who designed the elbow-connecting bracket. The hole pattern and slots made in the forearm tubing can be seen in Figure 4.5.

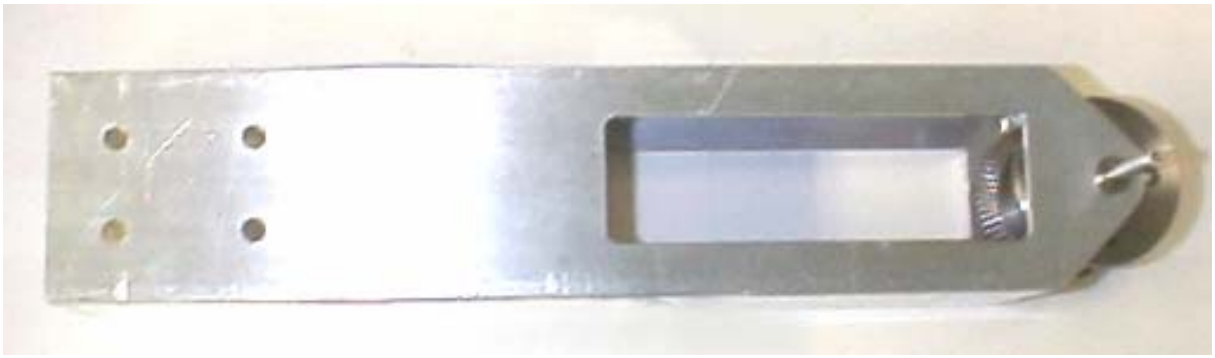


Figure 4.5 - Side View of Forearm Square Tubing

4.2 WRIST MOTOR MOUNTING BRACKET

To mount the two wrist motors within the forearm tubing, a mounting bracket was designed. Initially, the mounting brackets were designed as multi-piece brackets to be assembled in parts. The idea was to connect the two motors at the desired separating distance with one piece, then have separate pieces to actually connect them to the forearm tubing. However, the final mounting bracket design consisted of one piece that would be machined and formed into the desired shape. The bracket was made from a flat piece of 1/8 inch thick sheet metal. The mounting slots and holes that corresponded to the wrist motor were machined while the sheet metal was flat. Then, the two sides of the bracket were formed into the correct shape. The bracket was designed so it was small enough to simply slide into the forearm tubing and be manipulated into the correct position. The completed final mounting bracket can be seen in Figures 4.6 and 4.7.

Figure 4.6 - Motor Mounting Bracket

4.3 WRIST

The forearm is a fairly simple structure with the exception of the wrist. Since the wrist portion of the forearm was the most complicated component, it was designed first and everything else was designed around it. The concept for the wrist design was taken from last year's robot, and it consists of a differential gear driven by two small motors, as can be seen in Figure 4.7. This configuration allows both bending and twisting of the wrist using both motors. If each degree of freedom (bending and twisting) of the wrist were to be controlled by single motors (rather than the tandem motor configuration utilized), larger motors would be necessary. A required bending torque of approximately 18.8 in-lb was found from calculations based on initial estimates of weights in the gripper and object to be gripped. Through the bevel and pinion gear reduction of 4:1 (see Figure 4.7), the torque was further reduced by a factor of four, so the required torque for each motor was estimated to be 2.34 in-lb. The wrist motors chosen (see Parts List) are capable of 2.5 in-lb at 53 rpm, and the peak rated output torque is 10.9 in-lb.

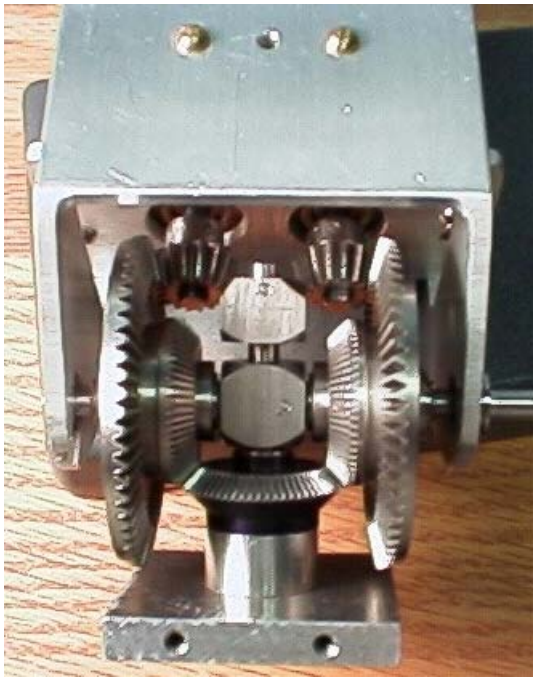


Figure 4.7 - Wrist Bevel and pinion gear configuration.

A pinion gear was placed on each wrist motor shaft, which drives a bevel gear connected to the wrist's differential gear, giving the 4:1 gear reduction/torque advantage in the wrist. The hubs on each of the bevel gears, which connect to the differential gear, were machined off to eliminate unnecessary material. This also allows for a more compact wrist configuration. The differential gear, along with the connecting bevel gears can be seen in Figure 4.7. The differential gear was chosen based on the stress developed on the teeth of the gears. Since the differential gear is small, the stresses on the teeth of the gears in the differential resulting from the bending torque became rather

high. A larger differential gear will have larger teeth, which will in turn have lower stresses. From these calculations it was determined that a 48 pitch differential gear should be utilized. However, the 48 pitch gear was too large to fit in the 2.5 x 2.5 inch tubing, so a 64 pitch differential gear was considered. The 64-pitch gear was determined to be adequate for the application, although it might be more susceptible to

Figure 4.8 - 64 pitch differential gear.

failure from fatigue than the 48-pitch gear.

The six holes shown on the face of the differential gear in Figure 4.8 are for mounting the gripper to the differential gear.

5.1 BRAINSTORMING AND RESEARCH

The gripper received much attention during this project. As the gripping mechanism for last year's robot was not functional, the entire project could not complete its desired goal. The design from the previous year was simple, but not very well executed. This year, the team set out to design a simple gripping mechanism that could easily be assembled and functional. However, many steps were necessary to achieve this goal.

The first design for the gripper was an adaptation of a wrench. The unit would open and close via a worm gear, in much the same way as a wrench. This concept would have been easy to manufacture, as the wrench has been a reliable gripping device for many years. However, this design was rejected for two reasons. First, many members in the group questioned the precision that could be achieved from a worm gear. It was feared that the sliding motion needed to close the gripper might make grasping objects difficult, as objects might be in contact with only one of the gripping fingers as they are closing. Next, with two, straight "fingers", there would be considerable difficulty in manipulating rounded objects. As some of the main purposes of the robotic manipulator might be to assist with personal hygiene, eating, and drinking, the group could identify more round objects, such as cups and bottles, than purely flat objects, like books, that would need to be manipulated. The group concluded that a gripper that would conform to the contours of these rounded objects would be a more suitable design for this project.

The second design of the gripper had two main improvements over its predecessor. This gripper had three fingers, instead of two because a more stable and secure grasp can be achieved with three fingers. Next, the fingers themselves were angled to grasp curved objects.

Again, the team recognized the difficulty that this type of manipulator would have with flat objects, as only point contact would be achieved with curved fingers. A device that could securely grasp both flat and curved objects was desired, and a gripper whose fingers could bend much like a human finger, begin straight and also have the ability to curve around an object, was desired. However, the cost of such a finger and gripper were initially deemed too high, as every knuckle would have to be individually powered to achieve this goal.

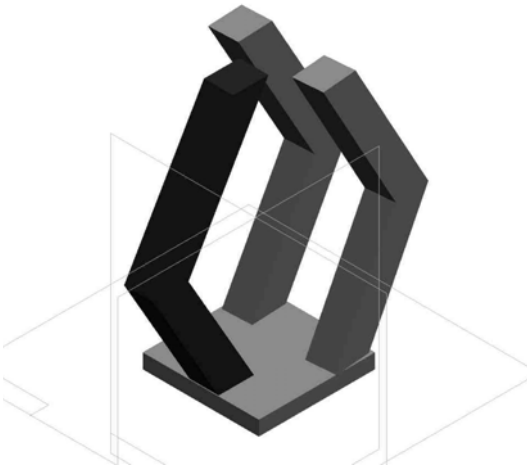


Figure 5.1 - Initial Gripper Design

The final concept for the gripper is a design that is able to grasp differently shaped objects yet is fairly inexpensive to construct. The premise behind the design is the concept of underactuation. A mechanism is underactuated if it has fewer actuators than degrees of freedom. An underactuated mechanism thus requires a limited number

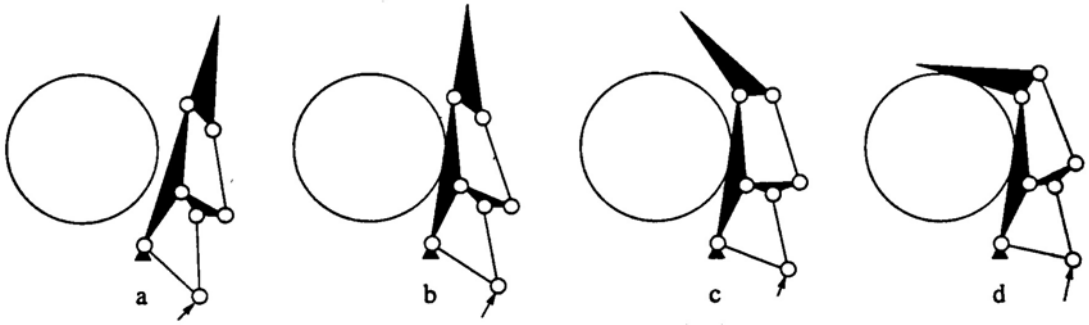


Figure 5.2 - Closing sequence of an underactuated system.

of actuators, but includes elastic elements (e.g. springs). The closing sequence of an underactuated system is shown in Figure 5.2.

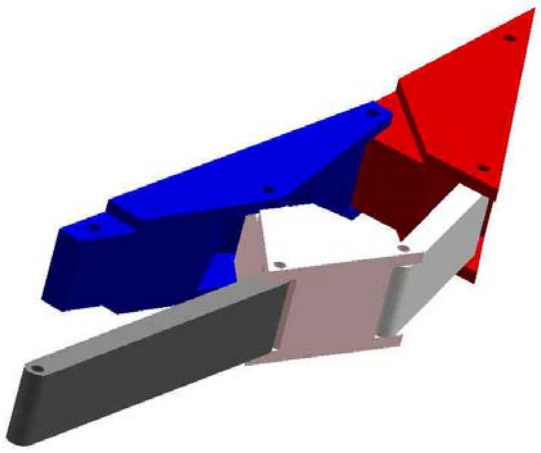


Figure 5.3 - Initial design of underactuated finger.

A design such as this allows the fingers to adapt to the shape of the object that they grasp. As two degrees of freedom are desired in the finger, the bending of the upper and lower knuckles can be controlled with only one torsion spring and one actuator. For this project, a nine bar, underactuated, two degree of freedom linkage was designed. The finger part of the design is displayed in Figure 5.3.

5.2 DESIGN OPTIMIZATION

After the initial design, a few modifications were made to the assembly. First, the parts were redesigned to eliminate some of the sharp edges. These edges were possible sources of stress concentration and failure, so they were removed in favor of "dumbbell-shaped" pieces. Next, a few of the parts were redesigned for ease of manufacturing.

The Matlab Script FIN4.m analyzed the kinematics and optimized the geometry of the underactuated finger. The scripts used are presented in Appendix E along with the program's results and a diagram of the finger. A static kinematic analysis was performed to determine the range of motion needed for the fingers and other parts of the gripper. The Matlab model simulated grasping cylindrical objects ranging in diameter form 1 to 4 inches, appropriate sizes for an object that fits the project constraints. Studying the forces for different sized objects showed how the forces changed at different finger positions. This analysis also studied the gripping force to determine if the gripper would be strong enough to pick up various objects. The force of the torsion spring was included. No dynamic analysis was performed because the velocities (and thus accelerations) are very low.

A subroutine, FUNK.m, optimized the geometry of the finger to maximize the gripping force while minimizing the forces on the joints. Matlab's minimax routine was given an initial guess for the geometry of the fingers and idealized the shapes with a sequential programming method (Brayton). This method was chosen because of its ability to minimize functions with multiple variables. The finger geometry was optimized to hold a 3-inch diameter cylinder.

As these components are relatively small, the longest piece measuring only two inches in length, the costs associated with machining become the paramount concern. Without a material that could be intricately machined at a very low cost, this type of design would not be practical. After consultation, the fingers were made out of two materials – aluminum and Lexan®. The majority of the components of the fingers were made from Lexan® (see Figure 5.4). This material features a high strength to weight

Figure 5.4 - Assembled finger and its components.

ratio, thus providing the necessary stability in the mechanism without adding unnecessary weight. The two most complex pieces of the finger mechanism were manufactured out of aluminum. Most likely, it would have been too difficult to cut such intricate details in a material such as Lexan®, as it would have splintered or broken in a milling machine. Aluminum was also the ideal choice for the more intricate pieces because of its durability and because it is ease of machining. Holes in this material can

be threaded, simplifying the assembly processes. Aluminum is also lightweight, inexpensive and readily available in many sizes.

The completed gripper assembly featured three of these linkages. This design maintained stability with three fingers while having the fingers able to conform to the shape of the grasped object (see Figure 5.5).

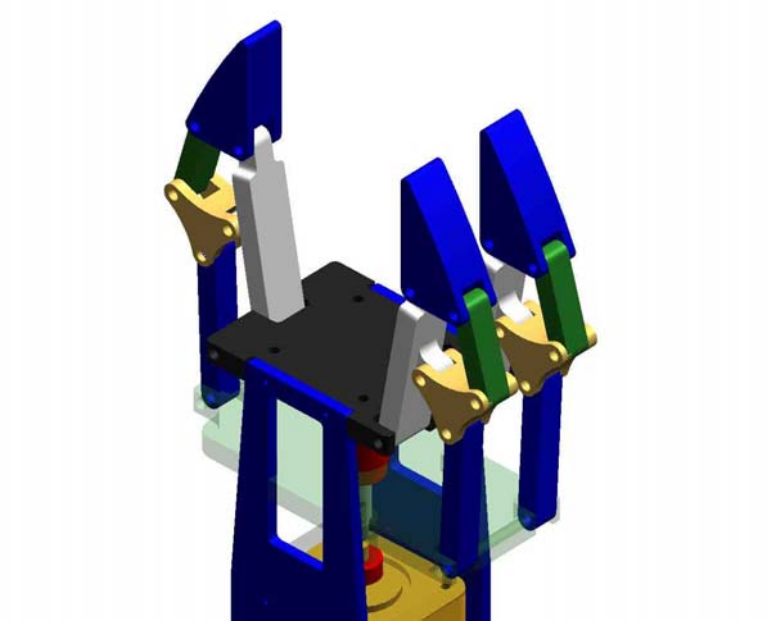


Figure 5.5 - Final Gripper Design

5.3 ASSEMBLY AND ACTUATION

Only one actuator was necessary for the gripper. A stepper motor, size Ht17-070 from Applied Motion Products, was used to move a ¼ inch acme lead screw purchased from PIC Design. As the greatest load occurs while the plate is moving away from the motor, a linear thrust bearing was placed around the lead screw to alleviate this load. Four Aluminum plates were needed to connect and operate the gripper. One plate was needed to connect the gripper to the differential and another plate mounted the motor to the gripper. The motor mounting plate also provided support for the thrust bearing. A spacer was sandwiched between the differential plate and the differential to provide clearance for the wrist to bend and rotate. The spacer length needed to give the wrist 130 degrees of bend was determined with a SolidEdge® model. Connecting the fingers

to the gripper and actuating them was accomplished with the two remaining plates. The third plate, the palm, secures the fingers to the hand and allows the fingers to rotate. It also protects the object being lifted from the lead screw. A plate was chosen because the design would be easy to manufacture and objects cannot become lodged in the gripper. The palm is connected to the fingers with 1/8 inch steel shafts. The force of the lead screw was transferred to the fingers with the moving plate, the fourth and final plate. It was necessary for this plate to move in only one direction to ensure a secure grip when picking up objects. Preventing the plate from rotating with the lead screw and balancing the forces on the fingers was necessary. All of the plates were made from ¼ inch aluminum.

Connecting the pieces of the gripper was a difficult task and the design went through several stages. The first design connected the palm, motor plate and differential plate with four rods (see Figure 5.6). The rods were bolted to the plates and held the plates parallel. The moving plate slid along the rods and was constrained by the rods. Because the rods contained little material, this design would have been light. It also would have provided the necessary support for the moving plate. However, difficulties arose in the design process. This design would not be able to resist torsion loading along the axis of the lead screw. The body of the gripper could twist easily. The rods would also be difficult to manufacture. It was also necessary to drill and tap holes into the ends of the small rods.

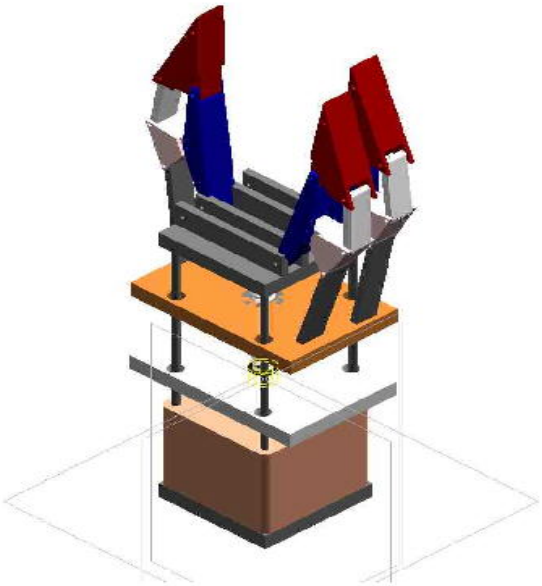


Figure 5.6 - Initial gripper design with rods.

A piece of square aluminum tubing was suggested to connect pieces of the gripper. The plates would have been made to fit inside the tube and the plates would

have been held in place with bolts. Slots could be machined into the side of the tube and matching tabs would have been made in the side of the moving plate to constrain the moving plate. This design was easy to manufacture and the parts were readily available. The tubing would have provided ample strength and torsional rigidity for the gripper. The tubing would also have enclosed the workings of the gripper, protecting the user. This design was not chosen, however, for three reasons. Compared to the chosen design, the tubing would have been heavy. Removing excess material from the tube would have added to the machining costs and decreased the operator's protection from moving parts. Assembling the small pieces inside the tube also would have been very difficult. The determining factor was the limited range of wrist bending imposed by the tube. The large square tubing would have required a long space to provide enough clearance for the wrist's range of motion

The final design used to connect the pieces of the gripper consisted of two 1/8 inch thick Lexan® plates. The gripper's plates were sandwiched between two Lexan® plates. A slot was machined into the Lexan® to fit a tab cut into the moving plate to constrain the moving plate. All of the other plates were bolted between the Lexan® plates.

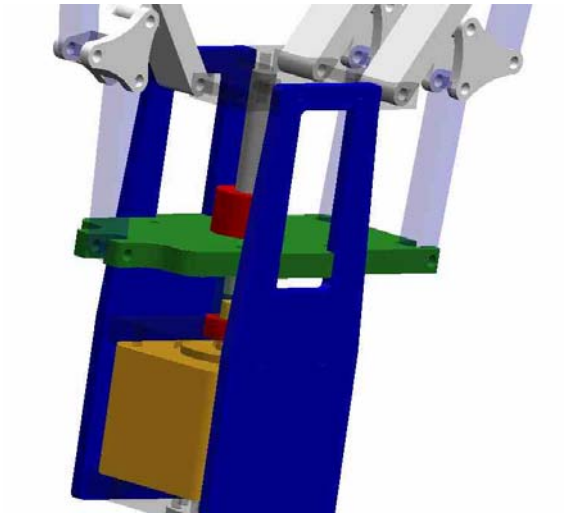


Figure 5.7 - Gripper design with Lexan® plates.

The rigidity of the gripper was increased by clamping the motor between the motor plate, the differential plate, and the side plates as shown in the picture. Aligning the slots and tabs of the moving plate was difficult because of the tight tolerances. A proper fit was achieved by machining the moving plate with an interference fit between the slots and the tabs. The excess material on the tabs was removed with hand tools to achieve the proper fit.

This design had several advantages. The two side plates were easy to make using a CNC mill. The sides were made identical to reduce the design time. The

Lexan® chosen for the plate was lightweight, inexpensive ($3.50 per ft^2, McMaster-Car Supply Company), and readily available from several sources. The plates also provided protection to the user from the inner workings of the gripper. The final gripper design is shown in Figure 5.8.

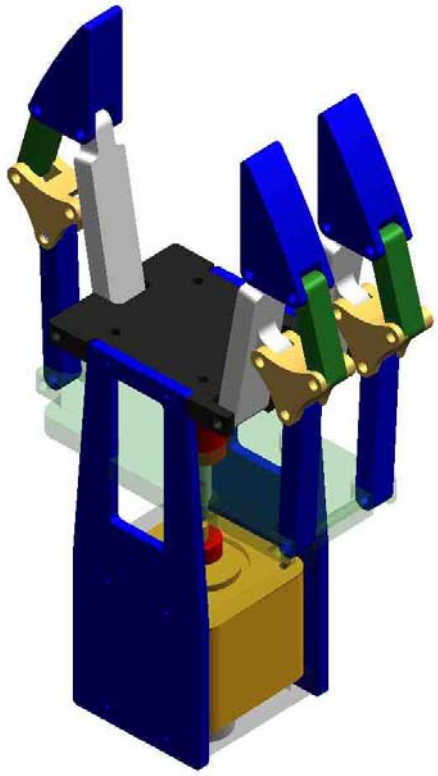


Figure 5.8 – Full view of final gripper design.

6.1 INTRODUCTION

Control system design for the manipulator has been an ever-developing component since the beginning of this project. The basic points guiding the design have been that the interface must be simple and intuitive enough to allow a quick understanding of the arm's operation. Additional considerations are outlined in the following list.

- The actual control mechanism must be simple enough that a person with extremely limited limb control is able to use it. For the purposes of this project, the user is assumed to have gross motor control of the antibrachium (forearm), but beyond the elbow has been treated as a single rigid member (this means no finger motion or movement of any part of the hand relative to the forearm).
- While the arm is in use, the wheelchair must not be able to move. This does not mean a locking of the wheels, but control of the wheelchair's movement *must not* occur simultaneously with that of the arm.
- As a general rule, the greater the "interchangeability", the better. Users of this device could have significantly different disabilities. A control system that could be easily altered to a specific individual's needs would be extremely beneficial to the user, as well as being a key selling point should the arm be put into production.

6.2 GENERAL NOTES

Control system design for a robotic arm is an experience unlikely encountered by most mechanical engineering students. The control system this year alone has developed into an elaborate project, requiring the learning of numerous new electrical-engineering techniques not covered in simple introductory courses. At the onset of this project, potential microcontrollers were significantly researched. What was not realized was that beyond the microcontroller itself lies a world of options in terms of actual motor control. While it is possible to do servomotor control from a microcontroller like the Oopic (discussed below), the processing power required is fairly intense and would

significantly complicate the programming. Unfortunately, this was not realized until the last quarter of this project (spring quarter), and made design significantly more difficult and time intensive.

At the onset of this project, a micro-controller named the "Oopic" (an acronym for object oriented programmable integrated circuit) was discovered via an internet search (see Figure 6.1). The controller itself has 31 digital input/output (IO) lines and contains several useful hardware features, such as 4 analog to digital converters (ADC) (implemented as one ADC with a sample-and-hold chip), 2 pulse-width-modulators (PWMs) - useful for digital motor control, and several other features. More information, including the full user guide, can be found on-line at www.oopic.com.

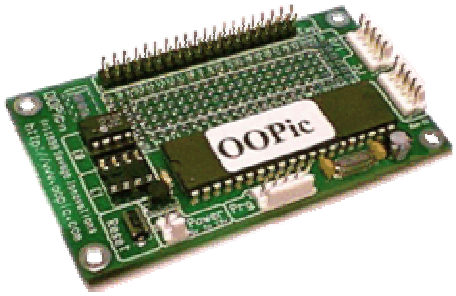


Figure 6.1 - OOPIC card.

One feature of the Oopic is networking capability, which made servomotor control (via the Oopic alone) an option. If this had been done, one Oopic would have been required for each motor in addition to one for the inputs. Up to 72 Oopics and/or other i^2c (a Phillips Company invention and industry standard) devices may be networked together. The math processing would be slow (relatively speaking) and could affect overall human interface performance. As a result it was decided to construct servomotor control boards from chip components. National Semiconductor (www.national.com) produces the necessary components to do this. The hardware interfacing requirements are significantly reduced (as compared to solely Oopic control, discussed in more detail below), using a chip select feature of the motor control chips. These integrated circuits produce a trapezoidal velocity profile, making them even more useful since they automatically prevent inertial loading.

The Oopic additionally features Visual Basic™ syntax programming, making customization of control algorithms a relatively simple task user-to-user. Any person so qualified could modify the standard control signature to a particular person's disability. Visual Basic code alone is much easier to read than assembly - the most frequently encountered code in programmable integrated circuits (PIC). Finally, the Oopic's price is right at $50.

6.3 ALGORITHM CONSIDERATIONS

Normal human day-to-day functioning is an automatic process – almost involuntary. The thought pattern required for what most consider simple movements are actually incredibly complex and intricate procedures. Consider drinking a glass of water; the thought process can be outlined in a few steps:

- Note glass location.
- Reach for and retrieve glass.
- Drink.

The background processing for these simple four steps is quite immense. Performing the same task in the control of a robotic system could take pages and pages of computer code – neglecting syntax formats and interface requirements and considering only the control calculations. There are several means of emulating natural thought patterns as they pertain to object manipulation. The mimicking options lie in the combination of both the control algorithm and the user interface design. That is, a "more powerful" input device such as an infrared motion detector (which could be used to detect motion in three dimensions) can receive better – or more natural – input from the user. An input device like this is similar to Windows™ based computing – point and "click." Whoever is using it needs only to look at an object and somehow indicate that it is what they would like picked up or manipulated. Another option is a multidimensional controller, such as that shown in Figure 6.2. This enables simultaneous control of several axes, but requires fine motor control skills and good

Figure 6.2 – Multidimensional controller.

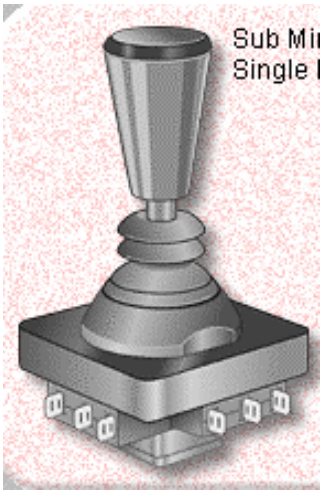


Figure 6.3 – 2-D on/off Joystick

dexterity. Something on the simpler end, such as the two-dimension on-off style joystick used this year (manufactured by Happ Controls, shown in Figure 6.3), is economically a better choice, but limited in its input style. In the case of the control algorithm designed this year, where two axes are simultaneously actuated, the control code must compensate for what the input device lacks.

6.4 CONTROLS SIMULATION

It was realized early in the project that some method of fine-tuning the control algorithm and user interface was needed, since doing so post-controls construction could prove expensive in terms of both money and time. To avoid this, a simulation of the arm was developed using Matlab™ software. Newer versions of this software (less

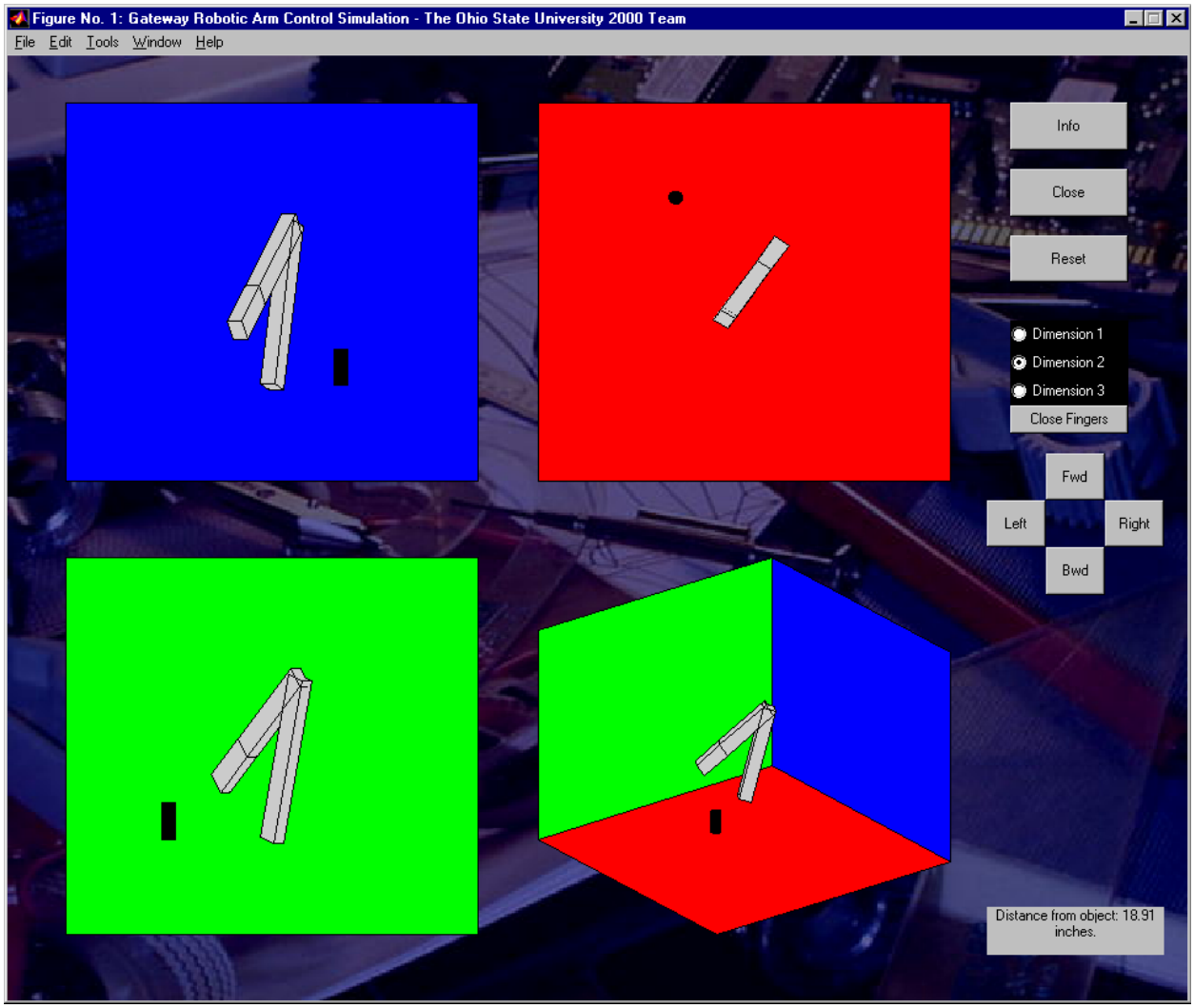


Figure 6.4 – Sample simulation screen.

than a few years old) have powerful graphical user interface capabilities. A sample screen of this is shown above, in Figure 6.4. The screen is composed of four plotting windows; one 3D, one top, right, and left each. The walls of the 3D view are color coded to make interpretation of the figures more understandable. In addition to the windows, there are four buttons setup in a 2-axis style arrangement. These comprise the virtual joystick of the simulation. The radio buttons labeled as “Dimension 1” to

“Dimension 3” emulate the buttons that allow selection of the control mode. Table 6.1 shows what each of these corresponds to in the most revised and planned control program. The gray box at the lower left of the simulation window indicates the distance between the object (black cylinder in the graphic windows) and the gripper’s center point. This allowed a sort of “usability timing” (from the home position to minimum object-gripper distance) to at least understand how well the user would be able to pick up something. While it is impossible to perform an actual timing test in a simulation like this, the overall feel is positive. Someone using this should be able to work quite efficiently. Other buttons on the simulation window are less important and are concerned with standard graphic window tasks, such as resetting the figure, closing the window, and getting help. The code is presented in Appendix F and may be downloaded from the Ohio State Gateway webpage at http://rclsgi.eng.ohio-state.edu/~gateway.

6.5 POWER CONSUMPTION AND MOTOR CONSIDERATIONS

A design concern for the controls system was the power required to move the motors. By conservation of power, using $v \cdot i = T \cdot \omega$ (where *v* is voltage in volts, *i* is current in amps, *T* is torque in N-m, and ω is joint speed in rad/s) it can be seen that even at low speeds the current drawn to the motors is fairly large (nearly 15 continuous amps at the shoulder alone). The batteries used on the wheelchair (Invacare Action Storm Series, electric powered) are 24V, 55 amp-hour. Since the first two control modes operate both the shoulder and elbow motors simultaneously, the *total* current draw was of the most concern. This was calculated using the Matlab script "tnp.m" (abbreviation of torque and power) provided in Appendix C, and was found to be around 11 A. It can also be seen in Figures 6.5 and 6.6. In short, the script varies the arm position (Theta 1 being the angle of the upper arm relative to an axis parallel to ground and Theta2p being the angle of the forearm relative to the upper arm) and calculates power requirements based on gripper speed constraints.

Table 6.1 - Control modes of operation.

Control Mode	Joystick Movements			
	Forward	Backward	Left	Right
1	Movement is restricted to in-plane parallel to ground.		Swivel arm around shoulder vertical axis.	
	Extend	Retract	Rotate toward wheelchair CL	Rotate away from wheelchair CL
2	Movement is restricted to in-plane perpendicular to ground		Swivel arm around shoulder vertical axis.	
	Raise	Lower	Rotate toward wheelchair CL	Rotate away from wheelchair CL
3	The entire arm – except the gripper – is locked in place. Only manipulation of the gripper and fingers can occur now.			
	Gripper rotates toward ground	Gripper rotates away from ground	Gripper rotates about its axis (pitch motion).	

Note: control mode numbers correspond to simulation radio button (dimension) numbers.

An interesting feature of control design is operation of the arm in any mode that rotates the shoulder away from the wheelchair or, in other words, toward the ground. The gravity-induced torque aids the arm's rotation in this direction and requires that any shoulder actuation in this particular mode should fight gravity. If the moment were applied in the same direction as gravity, the arm would rapidly accelerate into the ground. Regardless of the operator's intent, the manipulator would surely lose a

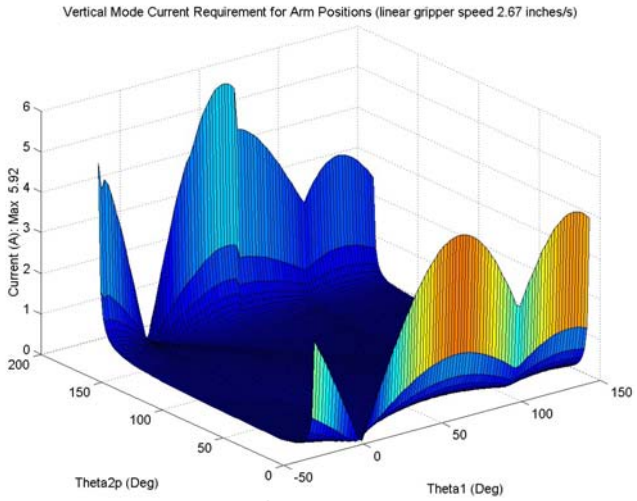


Figure 6.5

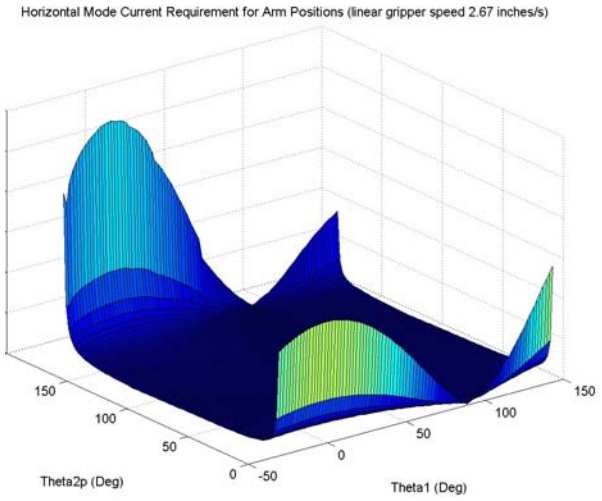


Figure 6.6

significant portion of its designed functionality. If the motor were run against the natural or gravity-assisted direction, the coil would be in a stall condition. Since stalling (or near stalling) of the motor is not a desirable situation with regards to motor longevity, this is an unacceptable means of control. Compensating for this situation means transforming the actuation of the arm in these modes to a controlled drop; a brake should be applied and the motor driven against it. This may not be possible this year, since the shoulder brake provides 34 ft-lb (408 in-lb) and the maximum shoulder torque is around 325 in-lb. The net difference (83 in-lb) is well within the motor's torque capability, but only occurs at one location. The shoulder brake will be angle-dependent actuated, to prevent full torque loading when the arm is not contributing as large a moment. The reader is directed to the "final design revisions" in Appendix G for more information on implementation.

It was recognized early in the project that in the event of power loss (for safety reasons) or operation in control mode 3, where the gripper is the sole arm link being actuated, the remainder of the arm needs to be locked in place. The only way to achieve this was through braking since the backdriving torque of the motors is relatively small. Theoretical backdriving torques were found using the manufacturer specified friction torques and the gearbox ratio. They were also experimentally measured and found to be in close agreement. That at the shoulder was borderline - near 300 in.-lb. - while the elbow did not exhibit sufficient torque to acquire a more accurate measurement. It was less than 6.25 in-lb by an experienced machinist's estimation. Several solutions to this dilemma were considered and included standard motor brakes (attached between the motor and gearbox, electromechanically actuated), custom-built band brakes, implementation of worm gears, ratcheting mechanisms, and other physical interference stops. The standard motor brakes were costly and had high lead times. A custom-built band brake was economically and time reasonable, but it is really only useful when the drum (or the braked shaft) is rotating in one direction. To allow the motor to move the arm when angular movement is opposite the desired braking direction, the band brake requires actuation to relieve drum pressure and thus shaft torque. When in the controlled drop the motor can drive through the brake. When raising the arm, the full arm torque plus that of the still "on" band brake would be too

great for the motor to overcome. Worm gears are lossy and tend to be built for heavy-duty industrial purposes. They are often enclosed in large steel units, making them weighty and physically large. While they are still a viable and cost-effective option, it was decided to pursue other routes due to time considerations. Electromagnetically released spring brakes (power-off brakes) were the first choice but were not discovered in a small or economical enough package for some time. Warner Electric (www.warnernet.com) does produce a brake that met all design requirements. Three brakes were purchased to lock the shoulder, elbow, and swivel joints. The elbow brake (model ERS-42) produces 84 in-lb while the shoulder model (ERS-57) creates 408 in-lb. Maximum torque at the elbow was calculated to be just less than 70 in-lb. with a 2-lb. object in the gripper. Shoulder maximum torque, with the addition of the 2.5 lb brake at the elbow joint is 325 in-lb. Since both brakes produce roughly 1.2 times the maximum force, they act as a safety in case of overload. If a child were to grab onto the arm and suspend itself, the arm would fall and prevent damage to one of its components. On the other hand, the braking torque is high enough to prevent arm rotation if it is somehow loaded slightly above expected operating conditions (an object heavier than 0.5 kg picked up by the gripper). From a power standpoint, both brakes are 24 VDC models and consume less than 1 amp each, making their power consumption almost negligible. Power on time (movement of the arm) will be a small fraction of the power off time.

6.6 CONTROL CODE

At the time this document was being written, control code and hardware problems are in the process of being debugged. More in-depth coverage of the programming will be covered in the final design revisions in Appendix G. However, all control flow can be simplified to the following:

1. Power on - reset and initialize all chips/processors.
2. Sense button and switch positions, obtain joint angles.
3. Wait for control input (joystick movement).
4. Calculate motor speeds and instruct motor control chips.
5. Repeat steps 3-5 until power-off.

Once the data has been written to the motor control chips, they perform the rest of the servo-controlling action. From a hardware perspective this means that Oopic IO lines have been freed (where they were once needed as encoder inputs) since the motor encoder outputs are now directly connected to the servomotor chips. With fewer Oopic IO lines now required, only two or three Oopics are needed. One (or two) Oopic(s) function as the dedicated motor control host(s), while the others are the data acquisition boards - reading the joint angles and performing speed calculations. This frees resources on both boards and should enhance overall system performance.

The motor control boards were custom built to save money. As irony laden as that statement may be, single chip solutions or pre-manufactured boards that were able to handle the power requirements (mainly current) of the motors were more expensive than custom construction. From an in-production standpoint, once the first circuit board has been successfully laid out, the remainder can be mass-produced at a significantly reduced cost. Chip prices alone decrease rapidly as quantity increases. Total control system cost is outlined in Table 6.2, and again in Table A.1.

Controls Equipment: Parts List and Cost Reference.

OHIO STATE

Gateway Design Team
1999 - 2000

Part	Vendor	Quantity		Description	Cost			Use in control Scheme
		Ordered	Implemented		Each	Total Used	Total Spent	
Oopic	Savage Innovations	2	1	Main Controlling Board(s)	$ 50.00	$ 50.00	$ 100.00	Board 1: Master - acquires data and processes. Board 2: Slave - controls motors.
	www.oopic.com					$ -	$ -	
32k EEProm	MicroChip Technolo	4	1	Memory Chips to increase Oopic capacity	$ 1.88	$ 1.88	$ 7.52	Permit storage of more complex angular limit matrices; increases usefulness of arm AND increases safety.
	www.microchip.com					$ -	$ -	
LM 629	Digikey Electronics	6	5	Dedicated PID Motor Controller (single chip implementation), National Semiconductor Product	$ 37.00	$ 185.00	$ 222.00	Servo Motor, closed loop control chip.
	www.digikey.com					$ -	$ -	
LM 18200	same as above	16	10	3A, 55V H-Bridge Circuits for each of 4 motors	$ 23.55	$ 235.50	$ 376.80	Allow motors to be driven forward and reverse with a constant lead-wire connection to the control board.
						$ -	$ -	
Digital Joystick	Happ Controls	2	1	Simple 8-way ON-OFF joystick for user interface (SubMiniature Sealed).	$ 40.00	$ 40.00	$ 80.00	User interface/input device.
	www.happcontrols.com					$ -	$ -	
LCD Display	Digikey Electronics	1	0	Optrex 16x2 Backlit Character LCD Display, www.optrex.co.jp	$ 29.64	$ -	$ 29.64	Shortens learning curve and provides user feedback information about control system.
						$ -	$ -	
6.0 MHz Clock	Digikey Electronics	5	5	External Clock for LM629's	$ 2.63	$ 13.15	$ 13.15	Used as the input clock on LM629
						$ -	$ -	
LMD 18245	Pioneer Standard	3	2	Stepper Motor Driver Chip, National Semiconductor Product	$ 20.40	$ 40.80	$ 61.20	
	www.mypioneer.com					$ -	$ -	
Miscellaneous connectors and parts	Digikey Electronics	1	1	Socket to socket connectors, cable assemblies, and other fun stuff.	$ 75.00	$ 75.00	$ 75.00	Required for connections and construction!
	(31.90 + extras)					$ -	$ -	
Motor Encoders	Encoder Products	2	2	For servo control of non-encoded motors, such as drill motors and elbow motor.	$ 60.00	$ 120.00	$ 120.00	
						$ -	$ -	
Manufacturing time	Joe West	10	10	Creation of circuit board layouts and making them.	$ 20.00	$ 200.00	$ 200.00	
Manufacturing Parts	www.joewestrocks.co	5	5		$ 5.00	$ 25.00	$ 25.00	

Total Controls Cost	$ 986.33	$ 1,310.31
Total Remaining Budget (from $1500)	$ 513.67	$ 189.69

Table 6.2: Controls Cost

6.7 WIRING THEORY

The first problem encountered when simple circuits were being developed was that of switch wiring. In order to perform a simple test, a switch was connected as input

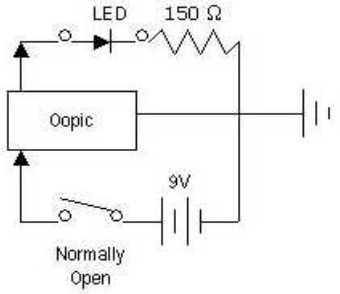


Figure 6.7: Switch Wiring Schematic 1

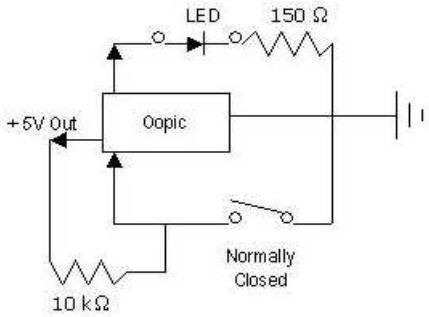


Figure 6.8: Switch Wiring Schematic 2

and an LED (light emitting diode) was connected as an output. This had some merit toward final controls setup since the three control mode indicators would be wired in a similar fashion. The desired effect is to press a momentary contact button and have a corresponding LED light (indicating which control mode is operating) for as long as no other switches are pressed. This is why the LED and switch could not be wired into series with the switch. The host processor needs to read the momentary switch press and light the appropriate LED - on a separate IO line - until another switch is pressed.

As with any digital signal, the Oopic can only detect an on/off switch by the voltage present on the specified input line. The first wiring scheme connected a battery directly to the input line via the switch (see Figure 6.7). This is bad practice for two reasons: the battery is essentially shorted through the controller and can exceed the maximum IO ratings, "frying" the circuit. What must happen instead is that the input line is presented with a return-to-ground line, shown in Figure 6.8. This means that when the switch is depressed (opening the return-to-ground line since the switch has been wired as normally closed), the input line goes logic high (+5 V) for as long as the switch is pressed. A “pull-up” resistor placed between the supply voltage high port and the input line limits the current. Imagine a capacitor as the sensor, with one side tied to ground and the opposite side connected to both voltage sensor and switch input; this is illustrated in Figures 6.9 and 6.10. When a logic high signal is applied, the capacitor takes on a charge. If the switch is then released, the voltage acquired on the capacitor

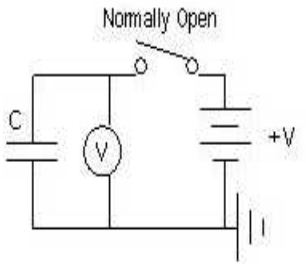


Figure 6.9:
Imaginary Sensor 1

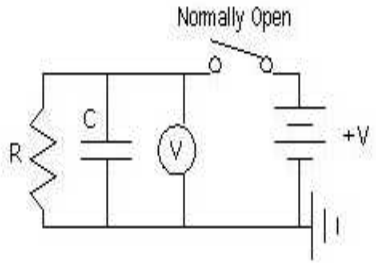


Figure 6.10:
Imaginary Sensor 2

has no path to decay (assuming infinite resistance in the voltmeter) and thus the voltmeter would continuously (in a realistic situation, for some long finite time) read logic high even though the switch had been turned off. Now imagine that the capacitor "input" (high side) is connected to both the switch and ground, via a resistor (10 kΩ used in this case). When the switch goes high, so does the capacitor input and thus the voltage sensor sees "on." Once the switch is released, the capacitor high side has an immediate track to ground and dissipates accordingly. The voltage sensor now reads low – or off – and control will function as expected.

7.1 INTRODUCTION

Not many designs are prototyped without carrying out a finite element analysis (FEA) on them. The Working Model ® add-on to Solid Edge ® was used to complete this task. The following parts were analyzed:

1. Mounting clamps
2. Stationary base plate
3. Swivel base plate
4. Lower collar for Swivel bearing
5. Upper arm links
6. Elbow brackets

The finite element analyses of the parts designed by Ohio State University, the forearm and the gripper, have been included in their respective sections.

All the upper arm and base parts were analyzed using twice the load that they were expected to take. This gives a factor of safety of at least 2. A Von Misses stress less than the yield strength of Aluminum 2024 (49 ksi) indicates that the part is strong enough to handle the applied load.

Some of the structural parts as well as parts used in power transfer were off-the-shelf items. This eliminated the need to perform a FEA on them and they were selected based on ratings from the respective manufacturers.

7.2 MOUNTING CLAMPS

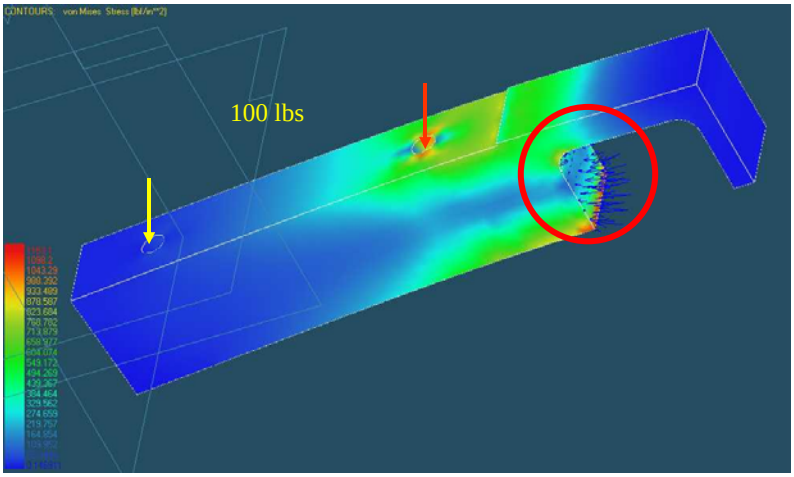


Fig. 7.1 – FEA of mounting clamp.

Figure 7.1 shows the output of the finite element analysis for the long piece of the mounting clamps. As shown in the picture, a total of 100 lb of load was applied on the clamp at the two holes. This resulted in a maximum stress of 1160 psi. The colors in the red circle show the reaction. This is one of the safest structural parts on the entire arm.

7.3 STATIONARY BASE PLATE

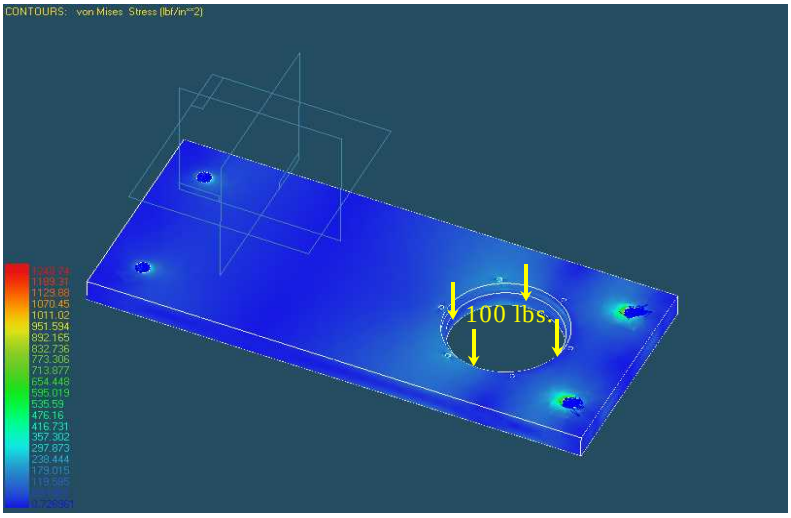


Fig. 7.2 - FEA of stationary plate.

The stationary plate was analyzed with a load of 100 pounds also. This is twice the weight of the parts from the shoulder brackets to the gripper. The plate was

constrained in all directions at the 4 holes where it was bolted on to the clamps. The analysis showed that this part of the arm plenty strong with a maximum Von Misses stress of 1280 psi. The holes on the right show higher stress values because the arm is not placed symmetrically on the plate.

7.4 SWIVEL PLATE

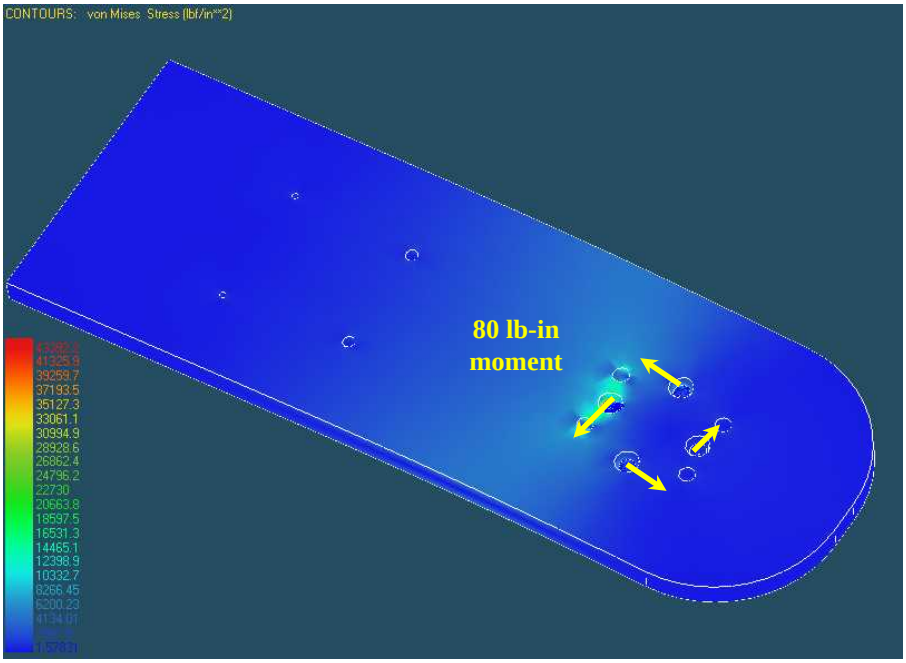


Fig. 7.3 – FEA of swivel plate.

The swivel plate shown in figure 7.3 was analyzed with an 80 in-lb moment applied, indicated by the arrows. This is twice the moment supplied by the swivel motor through the coupling, to the swivel plate. The plate was constrained at the four holes, which held the shoulder collar in place. An additional load of 100 lbs. was added at these holes to simulate the weight of the upper-arm and the forearm. The maximum stress experienced by the part was 43 ksi.

7.5 UPPER COLLAR FOR SWIVEL BEARING

The analysis of the collar shown in figure 7.4 is similar to that of the swivel plate. The 100-lb load was eliminated in this case, as the collar does not experience it (it is

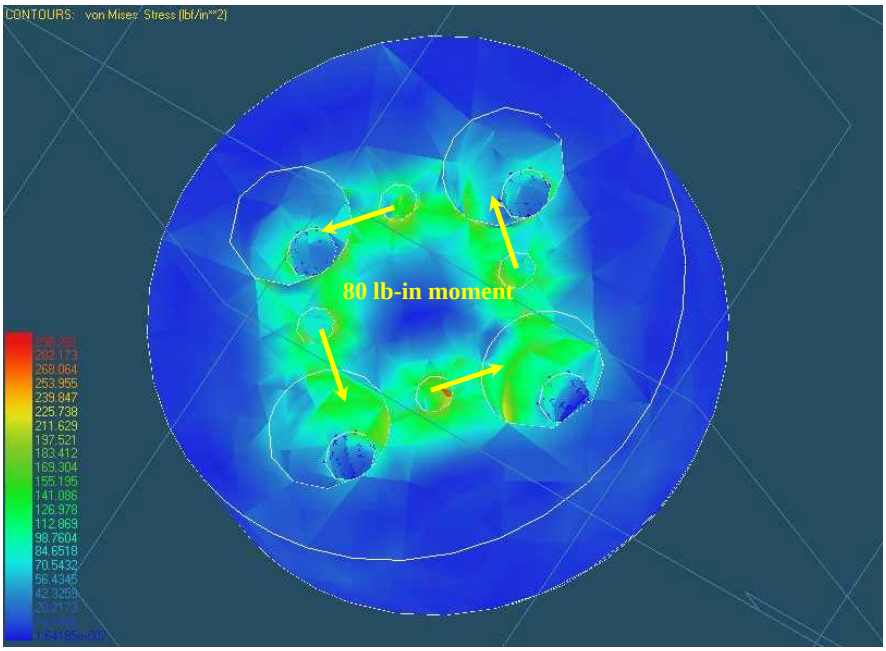


Fig. 7.4 – Upper swivel bearing collar.

transferred to the 4-point contact bearing). The analysis resulted in a maximum stress of 300 psi, which is well within the limit of 49,000 psi, that Aluminum 2024 is rated for.

7.6 UPPER ARM PLATE

The upper arm plate is one of the most important parts on the upper arm. It provides structural support to the forearm as well as the components that go in between the arm plates.

The upper arm plate was one of the most difficult parts to design and simulate. The reasons

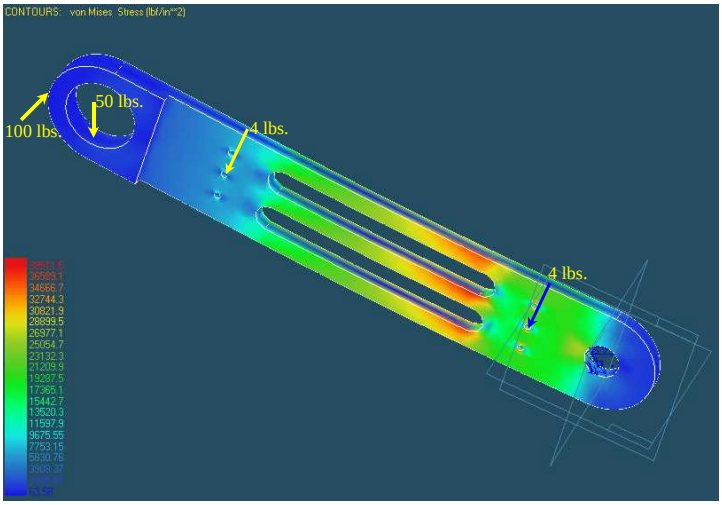


Fig. 7.5 – FEA of the Upper Arm Plate

for this are given in the Chapter 3, Section D. Figure 7.5 shows a picture of the output of the FEA analysis carried out on the upper arm plate.

Three loads were used for this analysis: a 50-lb. load was used to simulate the weight of the forearm, 4-lb. forces were used to simulate the weight of the elbow motor and the components used to mount it, and a 100 lb force was used to simulate a case where the user would run into an obstruction. The analysis resulted in a maximum stress of 38.81 ksi.

7.7 ELBOW BRACKET

The elbow bracket connects the forearm to the upper-arm. Thus a total force of 50 lbs. was applied at the holes where the forearm was to be connected. The bracket was constrained in all directions at the 1 inch diameter hole. The maximum stress experienced by this part was 1550 psi (see Figure 7.6). This again is far within the 49,000 psi yield strength of the material.

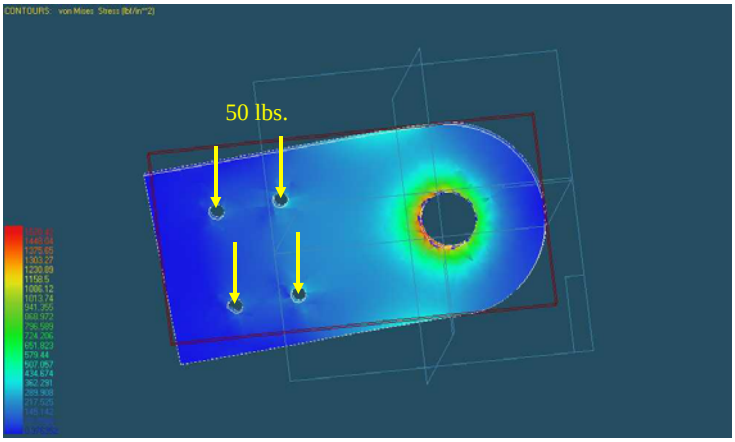
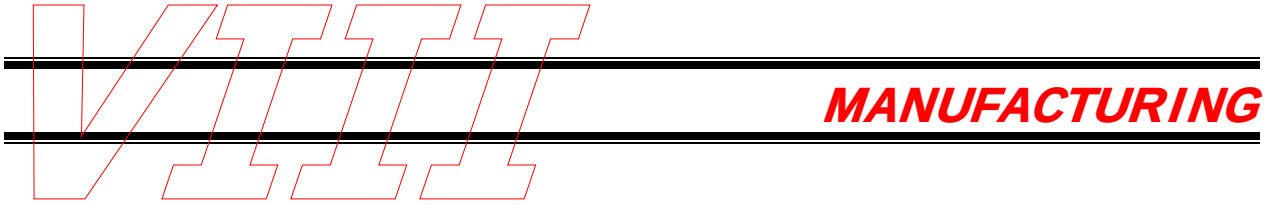


Fig. 7.6 – FEA of the elbow bracket.



8.1 INTRODUCTION

The third phase of the project comprised mainly of the manufacturing of various parts, and then assembling them. Students interacted with the machinists and gained an understanding of the possible complications in making a part. This was a real-world style experience in design for manufacturing, and exposed students to a number of different manufacturing and/or fabrication processes. Sinclair Community College manufactured the parts for the upper arm and base, Wright State's components. Most of Ohio State University's parts were manufactured in-house. The following is a parts list which details the various manufacturing times and costs.

Table 8.1 – Manufacturing Time and Material Cost				
Item No.	**Quantity**	**Part Description**	**Total Machine Hours**	**Total Material Costs**
Upper Arm				
1	2	Elbow bracket	1	
3	2	b) Bearing cap	0.4	
5	1	Gear elbow	0.2	
6	1	Pinion elbow	0.2	
8	1	Elbow shaft	0.6	
11	2	Upper arm plates	6.2	
12	1	Elbow motor support bearing plate mod	0.9	
13	2	Elbow motor support bearing plate	0.8	
		Section Totals	**10.3**	**$167.62**
Base				
15	2	Side Base Mounting Brackets	1	
19	1	Shoulder shaft	0.6	
21	1	Gearhead support plate	0.7	
22	1	Mounting base plate stationary	2.1	
23	1	Mounting plate swivel	1.5	
25	1	Top Swivel collar	0.9	
26	1	Bottom swivel collar	0.9	
27	1	Swivel bearing clamp	0.7	
28	1	Swivel bevel gear	0.2	
30	1	Swivel bevel pinion	0.2	
31	1	Swivel support plate	0.7	
32	1	Short clamp	0.9	
33	1	Long clamp	0.9	
34	1	Short clamp thick	0.9	
35	1	Long clamp thick	0.9	
		Section Totals	**13.1**	**$335.23**
Forearm				
37	1	2.5" Square Aluminum Tubing	1.5	$10.00
43	1	Wrist Motor Mounting Bracket, 1 / 8" Thick Sheet Metal	2.0	$3.50
		Section Totals	**3.5**	**$13.50**

Table 8.1 – Manufacturing Time and Material Cost (Continued)				
Item No.	**Quantity**	**Part Description**	**Total Machine Hours**	**Total Material Cost**
Gripper				
45	1	Lead Screw	0.5	$2.00
46	1	Moving Plate	2	$5.30
47	1	Gripper Motor Plate	1	$5.30
48	1	Differential Plate	1	$5.30
49	1	Palm	2	$5.30
50	2	Side Plates for Gripper Motor/Fingers Housing	0.5	$1.75
51	1	Spacer between Differential Gear and Wrist Motor Mount	0.5	$0.35
52	3	1.75" Finger Link	0.2	$0.32
53	3	1.00" Finger Link	0.2	$0.16
54	3	Iso-triangular Finger Link	0.75	$1.60
55	3	Straight Middle Finger Link	0.75	$1.60
56	3	Finger Tips	0.75	$1.60
60	1.5	Pins, 1.5 ft used	2	
63	1	Misc. machining	5	
		Section Totals	**17.15**	**$30.63**
Controls				
64	1	Control Interface Mounting Box (Machinist)	1	$0.50
65	1	Control Board Organizing Box (Machinist)	2	$1.00
66	6	Control Board Fabrication (Electronics Tech)	1.5	$0.10
67	1	Wiring of the Arm (Electronics Tech)	2	$2.00
		Section Totals	**14**	**$4.10**
Overall Totals			**58.05**	**$551.08**

As mentioned earlier, the material used was Al 2024. The above costs were determined using a machining cost of $25 per hour.

A more detailed price list is included in Appendix A. The totals calculated there are material costs of $546.98, machining costs of $1112.76, and motor costs of $1328.94.

IX FUTURE RECOMMENDATIONS

While this year's design of a robotic arm for quadriplegic children has made a number of advances over previous year's designs, there remains room for improvement. The following changes in the design are recommended:

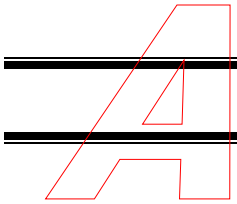
- ✓ Increase the width of the plate at the high stress area (see chapter VII, Figure 7.5) and reduce the width of the plate at the low stress areas to optimize the arm links, making them thinner and lighter. Reducing the length of the plate by the extra 2.9 inches that was added to include a brake will reduce the weight of the plate as well as the torque requirements of the motors.
- ✓ Reducing the thickness of the stationary plate and the swivel plate will also help reduce the weight of the entire assembly.
- ✓ Using hollow shafts instead of solid shafts is also recommended. This will give a high reduction in the weight of the shaft without much reduction in the torsional strength.
- ✓ Investigate off-the-shelf products whenever possible, including parts used in other products (e.g. general consumer goods like cordless drills). These parts will be significantly less expensive than custom pieces.

The following are some recommendations and advice for future groups that undertake this project:

- ♦ Promote communication and always keep the other teams updated on any design changes. It may be handy to keep a written record (somehow signed/approved by each team) of all joint decisions. This will reduce the confusion, the amount of work to be done, and the frustration level of all people involved.
- ♦ Keep solid model part and assembly files as up-to-date as possible.
- ♦ Interaction with the manufacturing team (e.g. preliminary manufacturing) should begin in the second quarter of the design process. This will give the

manufacturers and designers time to make changes in response to any unforeseen manufacturing problems.

- Layout a Gantt chart at the beginning of the project, early in Fall quarter. Feasible deadlines should be set (and met) so as to complete the project on time.



APPENDIX A – Parts & Costs Lists,

This appendix includes the parts and cost lists for all physical components. Drafts of custom-machined parts are included immediately following the tables. The tables have a nearly identical layout, but have been divided (into parts and part costs, and parts and part numbers) for clarity. These tables will be valuable references if any of the parts used in this year's project are going to be used on another similar project. Additionally, parts were priced at a purchased quantity of 50 whenever possible. This better reflects the true cost of the arm under production circumstances – a significant reduction as compared to the single-instance price and the totals used for this year's project.

Table A.1 – Part List, Costs						
Base						
Item No.	**Quantity**	**Part Description**	**Material Cost**	**Part Cost / Unit**	**Part Cost / Unit @ 50**	**Project Totals**
1	2	Side Base Mounting Brackets		**$12.50**	**$7.50**	**$25.00**
2	2	a) Bearing		**$8.65**	**$8.65**	**$17.30**
3	1	Base shaft coupling		**$37.37**	**$31.37**	**$37.37**
4	6	Lower bracket spacer		**$0.16**	**$0.16**	**$2.00**
5	1	Shoulder shaft		**$25.67**	**$19.35**	**$25.67**
6	1	Shoulder motor $ gearhead		**$1,500.00**	**$250.00**	**$250.00**
7	1	Gearhead support plate		$17.50	$12.50	$17.50
8	1	Mounting base plate stationary		$52.50	$47.50	$52.50
9	1	Mounting plate swivel		$37.50	$32.50	$37.50
10	1	Swivel bearing		$167.17	$167.17	$167.17
11	1	Top Swivel collar		$22.50	$17.50	$22.50
12	1	Bottom swivel collar		$22.50	$17.50	$22.50
13	1	Swivel bearing clamp		$17.50	$12.50	$17.50
14	1	Swivel bevel gear		$64.62	$59.62	$64.62
15	1	Swivel motor		$39.50	$39.50	$39.50
16	1	Swivel bevel pinion		$23.35	$18.35	$23.35
17	1	Swivel support plate		$17.50	$7.50	$17.50
18	1	Short clamp		$22.50	$17.50	$22.50
19	1	Long clamp		$22.50	$17.50	$22.50
20	1	Short clamp thick		$22.50	$17.50	$22.50
21	1	Long clamp thick		$22.50	$17.50	$22.50
22	26	Screws, Nuts, Washers, Bolts		$0.10	$0.10	$2.60
		Section Totals/Column	**$335.23**	**$557.24**		
		Section Total/unit @ 50			**$827.47**	
		Section Parts & Materials Project Total				**$2,516.27**
		Section Machining Totals (13.1 hr)	**$327.50**			
		Section Project Total (per unit, 1 unit)	**$2,843.77**			
		Section Total (per unit, 50 units)	**$1,154.97**			

Table A.1 – Part List, Costs (Continued)						
Upper Arm						
Item No.	**Quantity**	**Part Description**	**Material Cost**	**Part Cost / Unit**	**Part Cost / Unit @ 50**	**Project Totals**
23	2	Elbow bracket		$12.50	$12.50	$25.00
24	2	a) Bearings		$8.65	$8.65	$17.30
25	2	b) Bearing cap		$10.00	$5.00	$20.00
26	8	c) Spacers		$1.11	$1.11	$8.88
27	1	Gear elbow		$57.40	$52.70	$57.40
28	1	Pinion elbow		$36.52	$31.52	$36.52
29	4	Elbow Bracket Spacer		$0.16	$0.16	$2.00
30	1	Elbow shaft		$25.67	$19.35	$25.67
31	1	Elbow motor		$189.00	$75.00	$189.00
32	1	Elbow gearhead		$550.00	$75.00	$550.00
33	2	Upper arm plates		$155.00	$150.00	$310.00
34	1	Elbow motor support bearing plate mod		$22.50	$17.50	$22.50
35	1	a)Bearing		$5.22	$5.22	$5.22
36	2	Elbow motor support bearing plate		$20.00	$17.50	$40.00
37	34	Screws, Nuts, Washers		$0.10	$0.10	$3.40
		Section Totals/Column	**$167.62**	**$1,311.53**		
		Section Total/unit @ 50			**$844.13**	
		Section Parts & Materials Project Total				**$1,479.15**
		Section Machining Totals (10.3 hr)	**$257.50**			
		Section Project Total (per unit, 1 unit)	**$1,736.65**			
		Section Total (per unit, 50 units)	**$1,101.63**			
Forearm						
Item No.	**Quantity**	**Part Description**	**Material Cost**	**Part Cost / Unit**	**Part Cost / Unit @ 50**	**Project Totals**
38	1	2.5" Square Aluminum Tubing	$10.00	$47.50	$47.50	$47.50
39	1	64 Pitch, Standard Differential Gear		$302.22	$201.63	$302.22
40	2	Brush Commutated DC Gear Motor		$150.22	$101.33	$300.44
41	2	Boston Bevel Gear for Wrist(gear)		$23.30	$23.30	$46.60

Table A.1 – Part List, Costs (Continued)						
42	2	Boston Bevel Gear for Wrist(pinion)		$19.27	$19.27	$38.54
43	2	Double-Shielded, Flanged, Preci. Ball Brg.		$9.72	$7.63	$19.44
44	1	Wrist Motor Mounting Bracket (0.125” th.)	$3.50	$53.50	$53.50	$53.50
45		Screws, Nuts, Washers				
		Section Totals/Column	**$13.50**	**$1,365.48**		
		Section Total/unit @ 50			**$1,446.66**	
		Section Parts & Materials Project Total				**$821.74**
		Section Machining Totals (3.5 hr)	**$87.50**			
		Section Project Total (per unit, 1 unit)	**$909.24**			
		Section Total (per unit, 50 units)	**$1,534.16**			
Gripper						
Item No.	**Quantity**	**Part Description**	**Material Cost**	**Part Cost / Unit**	**Part Cost / Unit @ 50**	**Project Totals**
46	1	Lead Screw	$2.00	$14.50	$14.50	$14.50
47	1	Moving Plate	$5.30	$55.30	$55.30	$55.30
48	1	Gripper Motor Plate	$5.30	$30.30	$30.30	$30.30
49	1	Differential Plate	$5.30	$30.30	$30.30	$30.30
50	1	Palm	$5.30	$55.30	$55.30	$55.30
51	2	Side Plates for Gripper Motor/Fingers Housing	$1.75	$14.25	$14.25	$28.50
52	1	Spacer betwn. Diff. Gear and Wrist Motor	$0.35	$12.85	$12.85	$12.85
53	3	1.75" Finger Link	$0.32	$5.32	$5.32	$15.96
54	3	1.00" Finger Link	$0.16	$5.16	$5.16	$15.48
55	3	Iso-triangular Finger Link	$1.60	$20.35	$20.35	$61.05
56	3	Straight Middle Finger Link	$1.60	$20.35	$20.35	$61.05
57	3	Finger Tips	$1.60	$20.35	$20.35	$61.05
58	3	Torsional Springs for Gripper Fingers	$0.05	$0.05	$0.05	$0.15
59	1	Hybrid Step Motor		$48.00	$32.00	$48.00
60	1	Thrust Bearing for Gripper		$5.00	$5.00	$5.00
61	1.5	Pins, 1.5 ft used		$2.75	$2.75	$4.13
62	50	Retaining Clips		$0.15	$0.15	$7.50
63	1	Screws, Nuts, Bolts, Washers		$10.00	$10.00	$10.00

Table A.1 – Part List, Costs (Continued)						
	Section Totals/Column	**$30.63**	**$416.32**			
	Section Total/unit @ 50			**$430.95**		
	Section Parts & Materials Project Total				**$547.05**	
	Section Machining Totals (17.15 hr)	**$428.75**				
	Section Project Total (per unit, 1 unit)	**$975.80**				
	Section Total (per unit, 50 units)	**$859.70**				
Controls						
Item No.	**Quantity**	**Part Description**	**Material Cost**	**Part Cost / Unit**	**Part Cost / Unit @ 50**	**Project Totals**
64	2	Oopic Microcontroller	$0.00	$50.00	$39.00	$100.00
65	5	LM 629 PID servomotor control chip	$0.00	$37.00	$30.71	$185.00
66	10	LMD18200 H-bridge amplifier	$0.00	$23.55	$18.84	$235.50
67	2	LMD 18245 Full Bridge Stepper Driver	$0.00	$20.40	$16.82	$40.80
68	1	4 position digital joystick	$0.00	$40.00	$40.00	$40.00
69	2	1000 ppr encoders for drill motors	$0.00	$60.00	$60.00	$120.00
70	1	Miscellaneous Electronic Parts	$4.10	$75.00	$75.00	$75.00
	Section Totals/Column	**$4.10**	**$1,212.62**			
	Section Total/unit @ 50			**$1,123.64**		
	Section Parts & Materials Project Total				**$800.40**	
	Section Machining Totals (14 hr)	**$428.75**				
	Section Project Total (per unit, 1 unit)	**$1,229.15**				
	Section Total (per unit, 50 units)	**$1,552.39**				

Production Units	Total Cost per Unit
1 (Project Total)	$7,694.61
50 (Estimated Production Total)	$6,202.84

TABLE NOTES:

- The material cost column is given as a total for the indicated quantity; all other columns are recorded on a per part or per unit basis, as indicated.
- Machining costs were presented in the manufacturing section, with totals reflected in the summary table above.
- The totals given for 50 production units assume that drill motors would replace those purchased this year. (Price differences are reflected in the appropriate place)

TABLE A.2 - Part List - SUPPLIER				
Base				
Item No.	**Quantity**	**Part Description**	**Part Number**	**Vendor**
1a	2	Side Base Mounting Brackets		Sinclair
1b	2	Bearing	60355K79	Mc Master-Carr
2	1	Base shaft coupling	61005K56	Mc Master-Carr
3	6	Lower bracket spacer	97063a414	Mc Master-Carr
4	1	Shoulder shaft	5947K42	Mc Master-Carr
5	1	Shoulder motor		Lowe's home improvement
6	1	Gearhead support plate		Sinclair
7	1	Mounting base plate stationary		Sinclair
8	1	Mounting plate swivel		Sinclair
9	1	Swivel bearing	JB025XPO	Dayton Supplied Ind. Tech.
10	1	Top Swivel collar		Sinclair
11	1	Bottom swivel collar		Sinclair
12	1	Swivel bearing clamp		Sinclair
13	1	Swivel bevel gear	SB1.5-6015	Quality Trans. Comps.
14	1	Swivel motor	GMX-6MP013A	Servo Systems Co.
15	1	Swivel bevel pinion	SB1.5-1560	Quality Trans. Comps.
16	1	Swivel support plate		Sinclair
17	1	Short clamp		Sinclair
18	1	Long clamp		Sinclair
19	1	Short clamp thick		Sinclair
20	1	Long clamp thick		Sinclair
21	26	Screws, Nuts, Washers, Bolts		Sinclair
Upper Arm				
Item No.	**Quantity**	**Part Description**	**Part Number**	**Vendor**

TABLE A.2 - Part List – SUPPLIER (Continued)				
22	2	Elbow bracket		Sinclair
23a	2	Bearings	60355K79	Mc Master-Carr
23b	2	Bearing cap		Sinclair
23c	8	Spacers	92510A765	Mc Master-Carr
24	1	Gear elbow	11922 SS102-G	Dayton Supplied Ind. Tech.
25	1	Pinion elbow	11924 SS102-P	Dayton Supplied Ind. Tech.
26	4	Elbow Bracket Spacer	97063a414	Mc Master-Carr
27	1	Elbow shaft	5947K42	Mc Master-Carr
28	1	Elbow motor	23SMDC-LCSS	Servo Systems Co.
29	1	Elbow gearhead	23EP040	Servo Systems Co.
30	2	Upper arm plates		Sinclair
31a	1	Elbow motor support bearing plate mod		Sinclair
31b	1	Bearing	60355K76	Mc Master-Carr
32	2	Elbow motor support bearing plate		Sinclair
33	34	Screws, Nuts, Washers		Sinclair
Forearm				
Item No.	**Quantity**	**Part Description**	**Part Number**	**Vendor**
34	1	2.5" Square Aluminum Tubing		
35	1	64 Pitch, Standard Differential Gear	S9550A-TS2	Stock Drive Products
36	2	Brush Commutated DC Gear Motor	GM8724S027	Pittman Co.
37	2	Boston Bevel Gear for Wrist(gear)	GSS486Y-G	Bearing Distributors
38	2	Boston Bevel Gear for Wrist(pinion)	GSS486Y-P	Bearing Distributors
39	2	Double-Shielded, Flanged, Precision Ball Bearing	S9912Y-G1837FS2	Stock Drive Products
40	1	Wrist Motor Mounting Bracket, 1 / 8" Thick Sheet Metal		
41		Screws, Nuts, Washers		

TABLE A.2 - Part List – SUPPLIER (Continued)				
Gripper				
Item No.	**Quantity**	**Part Description**	**Part Number**	**Vendor**
42	1	Lead Screw	17-070-.25-20	PIC Inc.
43	1	Moving Plate		
44	1	Gripper Motor Plate		
45	1	Differential Plate		
46	1	Palm		
47	2	Side Plates for Gripper Motor/Fingers Housing		
48	1	Spacer between Differential Gear and Wrist Motor Mount		
49	3	1.75" Finger Link		
50	3	1.00" Finger Link		
51	3	Iso-triangular Finger Link		
52	3	Straight Middle Finger Link		
53	3	Finger Tips		
54	3	Torsional Springs for Gripper Fingers		
55	1	Hybrid Step Motor	HT17-070	Motion Technologies Co.
56	1	Thrust Bearing for Gripper	A727M0512	Stock Drive Products
57	1.5	Pins, 1.5 ft used	S121M2150	Stock Drive Products
58	50	Retaining Clips	A7X 1-0424A	Stock Drive Products
59	1	Screws, Nuts, Bolts, Washers		
60	1	Misc. machining		

APPENDIX B - Oopic Control Code

This section includes the code written for the main microcontroller, the Oopic. The majority of this code is communication with servomotor control device (National Semiconductor Chip LM629). The separate servomotor chip was a cost-effective way to increase the overall speed of the control system. In this way, the motor's desired speed is calculated by the Oopic and written to the LM629. That chip then takes full responsibility for monitoring the actual motor speed. The servomotor controller can sample much faster than the Oopic is capable, ensuring more accurate control of the speed. This freed the Oopic resources to poll for user input and make kinematic calculations. (Note, however, that this was never physically implemented; the code below WILL properly instruct the LM629 – it was the kinematic calculations that were never actually performed.) This code may be used again on another Oopic, providing that it is interfacing with the National Semiconductor LM629.

Other programs that were written have been included, but not tested or implemented. If the code is correct, when assembled together the entire set of programs will be a nearly complete control instruction set for full functionality of the arm. A major point should be made that none of these programs have been modified to include angular safety limits at the joints. This should be eventually done.

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  This program is the code that interfaces with the    ''
''  LM 629 Servo Motor Control Chip.                      ''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'  NOTES:
'  Chip Select is "on" or chip is selected for CS = 0.
'  Port Select, likewise.
'  Reset pin is active for pin low (0 V) or RSTpin = 0.

Dim RSTpin as New oDio1
Dim PSpin as New oDio1
Dim CSpin as New oDio1
Dim ComByte as New oDio8
Dim RDpin as New oDio1
Dim WRpin as New oDio1

Dim Tymer as New oTimer

Dim cc as New oByte
Dim bbit as New oByte
Dim delaytime as New oWord
Dim cval as New oByte

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''         Start Main         '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub Main()

   call InOutSetup
   CSpin.Value = 0:
RptRST:
   call LM629Rst
   call RDStat
   Select Case cval
     Case 132: Goto CONTINU:
     Case 196: Goto CONTINU:
   End Select
   Goto RptRST
CONTINU:
   call IntruptRst
   call RDStat
   Select Case cval
     Case 128: Goto Done_Rsting:
     Case 192: Goto Done_Rsting:
   End Select
   Goto RptRST
```

```
Done_Rsting:

   call LFParams
   call MskIntrupt
   call FwdLTrjData
   call StMoCntrl
  for cc = 1 to 250
  next cc
   call NormStop
   call StMoCntrl
   call RevLTrjData
   call StMoCntrl
   call NormStop
  for cc = 1 to 50
  next cc
   call StMoCntrl

   Oopic.Operate = cvOff:

End Sub
''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''      Start InOutSetup        '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub InOutSetup()

   RSTpin.Ioline = 13:
   RSTpin.Direction = cvOutput:
   RSTpin.Value = 1:

   CSpin.Ioline = 7:
   CSpin.Direction = cvOutput:
   CSpin.Value = 1:

   PSpin.Ioline = 11:
   PSpin.Direction = cvOutput:
   PSpin.Value = 1:

   RDpin.Ioline = 1:
   RDpin.Direction = cvOutput:
   RDpin.Value = 1:

   WRpin.Ioline = 12:
   WRpin.Direction = cvOutput:
```

```
   WRpin.Value = 1:

   ComByte.Iogroup = 2:
   ComByte.Direction = 0:
   ComByte.Value = 0:

   Tymer.Operate = 0:
   Tymer.ExtClock = 0:

End Sub
''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''Start Motor Control Chip Reset'''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub LM629Rst()

   RSTpin.Value = 0:

   delaytime = 7:
   call Weight:

   RSTpin.Value = 1:

   delaytime = 7500:
   call Weight:

End Sub
''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''   Start Reading Status Byte  '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub RDStat()

   ComByte.Direction = cvInput:
   PSpin.Value = 0:
   RDpin.Value = 0:
   delaytime = 1:
   call Weight:
   cval = ComByte.Value:
   RDpin.Value = 1:

End Sub
```

```
''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''   Start Interrupt(s) Reset   '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub IntruptRst()

   ComByte.Direction = cvOutput:
   ComByte.Value = 29:
   call CmndWrt

   ComByte.Direction = cvOutput:
   ComByte.Value = 0:
   call DHBWrt

   call DLBWrt

End Sub
''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''Start Loading Filter Parameters'''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub LFParams()

   ComByte.Direction = cvOutput:
   ComByte.Value = 30:
   call CmndWrt

   ComByte.Direction = cvOutput:
   ComByte.Value = 3:
   call DHBWrt

   ComByte.Value = 8:
   call DLBWrt

   ComByte.Direction = cvOutput:
   ComByte.Value = 0:
   call DHBWrt

   ComByte.Value = 10:
   call DLBWrt

'  Update filters/activate LFIL data.
```

```
   ComByte.Direction = cvOutput:
   ComByte.Value = 4:
   call CmndWrt

End Sub
''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''   Start Masking Interrupt(s) '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub MskIntrupt()

   ComByte.Direction = cvOutput:
   ComByte.Value = 28:
   call CmndWrt

   ComByte.Direction = cvOutput:
   ComByte.Value = 0:
   call DHBWrt

   ComByte.Value = 2:
   call DLBWrt

End Sub
''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''Start Loading Trajectory Params'''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub FwdLTrjData()

   ComByte.Direction = cvOutput:
   ComByte.Value = 31:
   call CmndWrt

   ComByte.Direction = cvOutput:
   ComByte.Value = 24:
   call DHBWrt

   ComByte.Value = 40:
   call DLBWrt

' Load Acceleration Data (HB of HW)
   ComByte.Direction = cvOutput:
```

```
  ComByte.Value = 0:
  call DHBWrt

' Load 2nd Acceleration Data (LB of HW)
  ComByte.Value = 50:
  call DLBWrt

' Load Acceleration Data (HB of LW)
  ComByte.Direction = cvOutput:
  ComByte.Value = 0:
  call DHBWrt

' Load 2nd Acceleration Data (LB of LW)
  ComByte.Value = 0:
  call DLBWrt

' Load Velocity Data (HB of HW)
  ComByte.Direction = cvOutput:
  ComByte.Value = 10:
  call DHBWrt

' Load 2nd Velocity Data (LB of HW)
  ComByte.Value = 200:
  call DLBWrt

' Load Velocity Data (HB of LW)
  ComByte.Direction = cvOutput:
  ComByte.Value = 200:
  call DHBWrt

' Load 2nd Velocity Data (LB of LW)
  ComByte.Value = 200:
  call DLBWrt

End Sub
''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''Start Loading Revrs Traj Params'''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub RevLTrjData()

  ComByte.Direction = cvOutput:
  ComByte.Value = 31:
  call CmndWrt
```

```
   ComByte.Direction = cvOutput:
   ComByte.Value = 8:
   call DHBWrt

   ComByte.Value = 0:
   call DLBWrt

End Sub
''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
"   Start Normal Stop Routine   '"
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub NormStop()

   ComByte.Direction = cvOutput:
   ComByte.Value = 31:
   call CmndWrt

   ComByte.Direction = cvOutput:
   ComByte.Value = 1:
   call DHBWrt

   ComByte.Value = 0:
   call DLBWrt

End Sub
''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
"Start STT (Start Motion Control)"
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub StMoCntrl()

' Start Motion Control with STT command
   ComByte.Direction = cvOutput:
   ComByte.Value = 1:
   call CmndWrt

End Sub
''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''     Start Command Write      '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub CmndWrt()

  PSpin.Value = 0:
  delaytime = 1:
  call Weight:
  WRpin.Value = 0:
  delaytime = 2:
  call Weight:
  WRpin.Value = 1:
  call ChipBusyBit:

End Sub
''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''  Start Data High Byte Write  '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub DHBWrt()

  PSpin.Value = 1:
  delaytime = 1:
  call Weight:
  WRpin.Value = 0:
  delaytime = 2:
  call Weight:
  WRpin.Value = 1:

End Sub
''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''  Start Data Low Byte Write   '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub DLBWrt()

  delaytime = 1:
  call Weight:
  WRpin.Value = 0:
  delaytime = 2:
  call Weight:
  WRpin.Value = 1:
```

```
  call ChipBusyBit:

End Sub
''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''     Start Busy Bit Check     '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub ChipBusyBit()

  ComByte.Direction = cvInput:
  cc = 1:
  Do
   bbit = (ComByte.Value mod 2):
   cc = cc + 1:
   if (cc > 500) then bbit = 0:
  Loop Until (bbit = 0)

End Sub
''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''     Start Delay Routine     '''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub Weight()

  Tymer.Operate = 1:
  Do
  Loop Until (Tymer.Value > delaytime)
  Tymer.Operate = 0:
  Tymer.Value = 0:

End Sub
''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
"  This code is INTENDED to control the      "
"  Stepper motor used in the gripper, via     "
"  the LMD18245 Full Bridge motor driver    "
"  Chips.  This is PROPOSED code, and has  "
"  not been tested yet (as of the inclusion   "
"  in this report.                                       "
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim m_a as New oDio4
Dim m_b as New oDio4
Dim dira as New oDio1
Dim dirb as New oDio1

Dim Tymer as New oTimer

Sub Main()

   call StepperSetup
   call GripperOpen
   call GripperClose

End Sub


Sub GripperClose()
' Reverse Direction
 Do
   call TakeaStep
   m_a.value = 8:
   m_b.value = 0:
   dira.value = 1:
   dirb.value = 0:

   call TakeaStep
   m_a.value = 0:
   m_b.value = 8:
   dira.value = 0:
   dirb.value = 0:

   call TakeaStep
   m_a.value = 8:
   m_b.value = 0:
   dira.value = 0:
   dirb.value = 1:

   call TakeaStep
```

```
   m_a.value = 0:
   m_b.value = 8:
   dira.value = 1:
   dirb.value = 1:
 Loop

End Sub


Sub GripperOpen()
' Forward Direction
 Do
   call TakeaStep
   m_a.value = 0:
   m_b.value = 8:
   dira.value = 0:
   dirb.value = 1:

   call TakeaStep
   m_a.value = 8:
   m_b.value = 0:
   dira.value = 0:
   dirb.value = 0:

   call TakeaStep
   m_a.value = 0:
   m_b.value = 8:
   dira.value = 1:
   dirb.value = 0:

   call TakeaStep
   m_a.value = 8:
   m_b.value = 0:
   dira.value = 1:
   dirb.value = 1:
 Loop

End Sub


Sub StepperSetup()

   m_a.iogroup = 1:
   m_a.nibble = 0:
   m_a.direction = cvOutput:
   m_a.value = 0:
```

```
   dira.ioline = 6:
   dira.direction = cvOutput:
   dira.value = 1:

   m_b.iogroup = 1:
   m_b.nibble = 1:
   m_b.direction = cvOutput:
   m_b.value = 0:

   dirb.ioline = 11:
   dirb.direction = cvOutput:
   dirb.value = 0:

   Tymer.ExtClock = cvFalse:
   Tymer.Operate = cvFalse:
   Tymer.Prescale = 3:

End Sub


Sub TakeaStep()

   Tymer.Operate = cvTrue:
   Do While (Tymer.Value < 1000)
   Loop
   Tymer.Operate = cvFalse:

End Sub
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'' This program is for A2D conversion of the potentiometers              ''
'' for the angles at each of the joints.  Preliminarily,                  ''
'' anyway.  It will likely be included in the final control program       ''
'' as a subroutine.  IT HAS NOT BEEN TESTED as of inclusion here.         ''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Theta1 as New oA2D:
Dim Theta2 as New oA2D:
Dim Theta3 as New oA2D:
Dim Theta1d as New oByte:
Dim Theta2d as New oByte:
Dim Theta3d as New oByte:
Dim Converter(3) as New oMath:
Dim Theta1dot as New oWord:
Dim Theta2dot as New oWord:
Dim Theta3dot as New oWord:
Dim ThetaSign as New oDio1:
Dim v2deg as New oWord:
'  Volts --> degrees conversion, done by:
'  Total Pot Resistance * Usable Degrees of Pot / Oopic a2d resolution
'  For example, with Rt = 1kohm, 270 pot degrees, and 2^8 resolution
'  v2deg = 1000*270/256 = 1055

Const UpArm = 18:   'inches long
Const ForArm = 12: 'inches long
Const Gripper = 7: 'inches long

Sub Main()
        call Startup:
End Sub

Sub Startup()
        Theta1.Ioline = 1:
        Theta2.Ioline = 2:
        Theta3.Ioline = 3:
        Theta1.Operate = 1:
        Theta2.Operate = 1:
        Theta3.Operate = 1:
        Oopic.ExtVRef = 0:  'Set voltage reference to +5 V
        Converter(1).Mode = 1:
        Converter(2).Mode = 1:
        Converter(3).Mode = 1:
        Converter(1).Input1.Link(Theta1.Value):
        Converter(1).Input2.Link(v2deg):
        Converter(1).Output.Link(Theta1d):
```

```
        Converter(2).Input1.Link(Theta2.Value):
        Converter(2).Input2.Link(v2deg):
        Converter(2).Output.Link(Theta2d):
        Converter(3).Input1.Link(Theta3.Value):
        Converter(3).Input2.Link(v2deg):
        Converter(3).Output.Link(Theta3d):
End Sub

Sub VertMode()
        if A > B,
        Theta1dot = 3/(UpArm*(sin(90-Theta1d)-sin(Theta1d)*sin(90-
Theta2d)/sin(Theta2d))):
        Theta2dot = 0-Theta1dot*(UpArm*sin(Theta1d))/(ForArm*sin(Theta2d)):
End Sub

Sub HorizMode()
        Theta1dot = 3/(UpArm*(sin(Theta1d) - sin(90-Theta1d)*sin(Theta2d)/sin(90-
Theta2d))):
        Theta2dot = 0-Theta1dot*(UpArm*sin(90-Theta1d))/(ForArm*sin(90-Theta2d)):
End Sub

Sub Manipulate()
'       Theta3dot = 20:
End Sub
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  This is for control mode selection, although it    ''
''  has not been tested either.                         ''
''   Rock on.                                           ''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim modeEvent1 as New oEvent
Dim modeEvent2 as New oEvent
Dim modeEvent3 as New oEvent
Dim mode_gate(3) as New oGate
Dim mode_btn(3) as New oDio1
Dim mode_led(3) as new oDio1
Dim cc as New oNibble:

Sub Main()
  call Startup:
  mode_gate(1).Input1.Link(mode_btn(1)):
  mode_gate(1).Output.Link(modeEvent1.Operate):
  mode_gate(2).Input1.Link(mode_btn(2)):
  mode_gate(2).Output.Link(modeEvent2.Operate):
  mode_gate(3).Input1.Link(mode_btn(3)):
  mode_gate(3).Output.Link(modeEvent3.Operate):
End Sub

Sub Startup()
          For cc = 1 to 3 step 1
            mode_btn(i).IOline = 14 + i:
            mode_btn(i).Direction = cvInput:
            mode_led(i).IOline = :
            mode_led(i).Direction = cvOutput:
            mode_led(i).Value = cvOff:
          Next cc
End Sub

Sub modeEvent1_CODE()
          mode_led(1).Value = cvOn:
          mode_led(2).Value = cvOff:
          mode_led(3).Value = cvOff:
End Sub

Sub modeEvent2_CODE()
          mode_led(1).Value = cvOff:
          mode_led(2).Value = cvOn:
          mode_led(3).Value = cvOff:
End Sub
```

```
Sub modeEvent3_CODE()
        mode_led(1).Value = cvOff:
        mode_led(2).Value = cvOff:
        mode_led(3).Value = cvOn:
End Sub
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  This program is a working, tested version using   ''
''  the joystick as a 4-way input switch.  It only       ''
''  on LEDs in response to the switching action, but  ''
''  can and probably will be used for actual control   ''
''  code actuation.                                            ''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim UpLED As New oDio1
Dim UpJoy As New oDio1
Dim DownLED As New oDio1
Dim DownJoy As New oDio1
Dim LeftLED As New oDio1
Dim LeftJoy As New oDio1
Dim RightLED As New oDio1
Dim RightJoy As New oDio1

Sub Main()
  UpLED.Ioline = 19
  UpLED.Direction = cvOutput
  UpJoy.Ioline = 17
  UpJoy.Direction = cvInput
  DownLED.Ioline = 13
  DownLED.Direction = cvOutput
  DownJoy.Ioline = 12
  DownJoy.Direction = cvInput
  LeftLED.Ioline = 11
  LeftLED.Direction = cvOutput
  LeftJoy.Ioline = 10
  LeftJoy.Direction = cvInput
  RightLED.Ioline = 9
  RightLED.Direction = cvOutput
  RightJoy.Ioline = 8
  RightJoy.Direction = cvInput
Do
  UpLED = UpJoy
  DownLED = DownJoy
  LeftLED = LeftJoy
  RightLED = RightJoy
Loop
End Sub
```

APPENDIX C – Power Calculation Code

This code serves the purpose of calculating power consumption over the arm's entire in-plane motion. Since both motors were to be actuated simultaneously, the total instantaneous current draw needed to be known. The code varies shoulder and elbow joint angles and calculates power consumption according to the restriction of moving the gripper in either the parallel or perpendicular to ground phases. While it was known, and obvious, that the maximum moment occurred when the arm was fully extended, it was thought that the peak power draw might occur at a slightly different location than the fully extended position. The output of the code is a number of plots and a text file, all of which serve to identify maximum current draw (for a specified battery voltage) at the shoulder and elbow motors. For the purposes of this project, the information was used to design appropriate control circuitry that could withstand the current drawn. It could be used for any general arm, providing that the movement may be simplified to actuation in only two joints. Battery voltages may also need to be altered within the program.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Developed by:   Fred Griesemer
% Title:          Senior M.E. Student
% Project:        1999-2000 Gateway Design Team
% Location:       Ohio State University
% Department:     Mechanical Engineering
%
% This program determines power consumption
% of the robotic arm.  Simulation is done over
% typical range of motion, using horizontal
% and vertical drive: replicating actual control algorithm.
%
% This is a (physical) 3 link member.  The upper arm is
% analogous to the humerus - upper arm bone in the human
% body (bone between bicep and tricep).  The forearm is
% self explanatory, while the gripper would be everything
% you see from the wrist to fingertips.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
clc;
%clf;
welcometext = ['Please be patient while this runs.  There are many\n'...
      'calculations to be performed and graphics to be\n' ...
      'created.  Sing a song to pass the time...\n\n' ...
      'This will create jpg files of each of the\n' ...
      'figure windows, for reference after running\n' ...
      'the program.  If you modify any of the\n' ...
      'dimensions or weights, be sure to change\n' ...
      'them in the m-file and let us Buckeyes\n' ...
      'know as well.\n\n'];
fprintf(1,welcometext);

% Link Lengths and other constants.  Modify these as necessary
% for your particular design.
in2m = 2.54/100;           % conversion, inches to meters
lb2N = (1/2.204623)*9.81;  % conversion, (lb to kg)*9.81 = N -- 2.204623 lb =
1 kg -- 1lb = .45359 kg
d2r = pi/180;              % conversion, degrees to radians
UpperArm.Length= 18*in2m;  % inches -> m
UpperArm.cg    = 10*in2m;  % inches -> m: from "back" of section (piece
closest to wheelchair)
UpperArm.Weight= 7*lb2N;   % lb -> N  original weight was 14
ForeArm.Length = 12*in2m;  % inches -> m
ForeArm.cg     = 6.5*in2m; % inches -> m
ForeArm.Weight = 3.3*lb2N; % lb -> N
Gripper.Length = 7*in2m;   % inches -> m
Gripper.cg     = 3.5*in2m; % inches -> m
Gripper.Weight = 3.1*lb2N; % lb -> N

% The following are used in order to simplify typing required
% for calculations in the upcoming loop.
r1 = UpperArm.Length;      % meters
r2 = ForeArm.Length + Gripper.Length;  % meters
r3dot = 2.67*in2m;   % Horizontal constant speed, 2.67 in/s -> m/s
r4dot = r3dot;       % Vertical constant speed
```

```
% Set position movement.
theta1d = -30:2:145; % degrees
theta2pd= 1:2:180;   % degrees (theta2 prime, angle between upper- and fore-
arms)

for i = 1:1:length(theta1d),
   for j = 1:1:length(theta2pd),
      theta1 = theta1d(i)*d2r;      % Computational Simplification (radians)
      theta2 = (theta1d(i) + theta2pd(j) + 180)*d2r;  % radians
      theta1pl(i,j) = theta1d(i);
      theta2pl(i,j) = theta2pd(j);
      r3(i,j) = UpperArm.Length*cos(theta1) + (ForeArm.Length +
Gripper.Length)*cos(theta2);  % Gripper horizontal distance from shoulder,
meters
      r4(i,j) = UpperArm.Length*sin(theta1) + (ForeArm.Length +
Gripper.Length)*sin(theta2); % Vertical same, meters
      % For horizontal movement only:
      theta1dot_h(i,j) =
r3dot/((r1*cos(theta1))/(r2*cos(theta2))*r2*sin(theta2) - r1*sin(theta1));
      theta2dot_h(i,j) = -theta1dot_h(i,j)*(r1*cos(theta1))/(r2*cos(theta2));
      theta2pdot_h(i,j) = theta2dot_h(i,j) - theta1dot_h(i,j);
      % For vertical movement only:
      theta1dot_v(i,j) = r4dot/(r1*cos(theta1) -
(r1*sin(theta1))/(r2*sin(theta2))*r2*cos(theta2));
      theta2dot_v(i,j) = -theta1dot_v(i,j)*(r1*sin(theta1))/(r2*sin(theta2));
      theta2pdot_v(i,j) = theta2dot_v(i,j) - theta1dot_v(i,j);
      % Calculation of Torques, assuming forearm/gripper as combined in-line
rigid body.
      % Results in N*m.
      Torque.Elbow(i,j) = (ForeArm.cg*ForeArm.Weight + (Gripper.cg +
ForeArm.Length)*Gripper.Weight)*cos(theta2);
      Torque.Shoulder(i,j) = UpperArm.cg*UpperArm.Weight*cos(theta1) + ...
         ((ForeArm.cg + UpperArm.Length*cos(theta1))*ForeArm.Weight + ...
         (Gripper.cg + ForeArm.Length +
UpperArm.Length*cos(theta1)*Gripper.Weight)*cos(theta2));
      Torque.Net(i,j) = 34*12 - Torque.Shoulder(i,j); % include "controlled
drop" braking force
      % Power Required at each joint, same assumptions as above.
      % Results in W: N*m/s.
      Power.Elbow_h(i,j)      = abs(Torque.Elbow(j)*theta2dot_h(i,j)/80);
      Power.Shoulder_h(i,j)   = abs(Torque.Shoulder(i,j)*theta1dot_h(i,j));
      Power.Total_h(i,j)      = abs(Power.Elbow_h(i,j)) +
abs(Power.Shoulder_h(i,j));
      Power.Elbow_v(i,j)      = abs(Torque.Elbow(j)*theta2dot_v(i,j)/80);
      Power.Shoulder_v(i,j)   = abs(Torque.Shoulder(i,j)*theta1dot_v(i,j));
      Power.Total_v(i,j)      = abs(Power.Elbow_v(i,j)) +
abs(Power.Shoulder_v(i,j));
   end
end

fignum = 1;    % Figure Indexer
Voltage = 24;  % Battery Voltage
figure(fignum);
plot(r3,r4,'LineStyle','none','Marker','o','Color','blue');
title('Planar Workspace of Wheelchair, dimensions in inches');
grid on;
```

```
figure(fignum + 1);
surf(theta1d,theta2pd,Power.Total_h);
xlabel('Theta1 (Deg)');
ylabel('Theta2p (Deg)');
zlabel('Power (W)');
title('Horizontal Mode Total Power Consumption for Arm Positions');
print -djpeg90 hpower.jpg

figure(fignum + 2);
surf(theta1d,theta2pd,Power.Total_v);
xlabel('Theta1 (Deg)');
ylabel('Theta2p (Deg)');
zlabel('Power (W)');
title('Vertical Mode Total Power Consumption for Arm Positions');
print -djpeg90 vpower.jpg

figure(fignum + 3);
surf(theta1d,theta2pd,Power.Total_h/Voltage);
xlabel('Theta1 (Deg)');
ylabel('Theta2p (Deg)');
hcurrent_max = max(max(abs(Power.Total_h/Voltage)));
zlabeltexth = sprintf('Current (A): Max %5.2f',hcurrent_max);
zlabel(zlabeltexth);
title('Horizontal Mode Current Requirement for Arm Positions (linear gripper
speed 2.67 inches/s)');
print -djpeg90 hcurrent.jpg

figure(fignum + 4);
surf(theta1d,theta2pd,Power.Total_v/Voltage);
xlabel('Theta1 (Deg)');
ylabel('Theta2p (Deg)');
vcurrent_max = max(max(abs(Power.Total_v/Voltage)));
zlabeltextv = sprintf('Current (A): Max %5.2f',vcurrent_max);
zlabel(zlabeltextv);
title('Vertical Mode Current Requirement for Arm Positions (linear gripper
speed 2.67 inches/s)');
print -djpeg90 vcurrent.jpg

figure(fignum + 5);
surf(theta1d,theta2pd,Torque.Shoulder);
xlabel('Theta1 (Deg)');
ylabel('Theta2p (Deg)');
storque_max = max(max(abs(Torque.Shoulder)));
zlabeltextst = sprintf('Torque (N*m): Max %5.2f',storque_max);
zlabel(zlabeltextst);
title('(Net) Shoulder Torque Requirement for Arm Positions');
print -djpeg90 storque.jpg

figure(fignum + 6);
surf(theta1d,theta2pd,Torque.Elbow);
xlabel('Theta1 (Deg)');
ylabel('Theta2p (Deg)');
etorque_max = max(max(abs(Torque.Elbow)));
zlabeltextet = sprintf('Torque (N*m): Max %5.2f',etorque_max);
zlabel(zlabeltextet);
title('Elbow Torque Requirement for Arm Positions');
print -djpeg90 etorque.jpg
```

```
fid = fopen('JointTorques.txt','wt');
fprintf(fid,'\nMaximum Joint Torques and Stuff\n\n');
fprintf(fid,'Max Elbow\t\tMax Shoulder (N-m)\n');
fprintf(fid,'%8.3f\t\t%8.3f\n\n',[etorque_max storque_max]);
fprintf(fid,'Max Elbow\t\tMax Shoulder (lb-in)\n');
fprintf(fid,'%8.3f\t\t%8.3f\n\n',[etorque_max storque_max]*1/in2m*1/lb2N);
fprintf(fid,'Max Power (W)\t\tMax Current (A)\n');
current_max = max([hcurrent_max vcurrent_max]);
fprintf(fid,'%8.3f\t\t%8.3f',[current_max*Voltage current_max]);
fclose(fid);

rads2rpm = 60/(2*pi);   % conversion rads/s to rpm
fprintf(1,'\nMaximum Joint Torques and Stuff\n\n');
fprintf(1,'Max Elbow\t\tMax Shoulder (N-m)\n');
fprintf(1,'%8.3f\t\t%8.3f\n\n',[etorque_max storque_max]);
fprintf(1,'Max Elbow\t\tMax Shoulder (lb-in)\n');
fprintf(1,'%8.3f\t\t%8.3f\n\n',[etorque_max storque_max]*1/in2m*1/lb2N);
fprintf(1,'Max Power (W)\t\tMax Current (A)\n');
fprintf(1,'%8.3f\t\t%8.3f\n\n',[current_max*Voltage current_max]);
fprintf(1,'Max Speedv (RPM)\tMax Speedh (RPM)\t THETA1dot\n');
fprintf(1,'%8.3f\t\t%8.3f\n\n',[max(max(abs(theta1dot_v)))*rads2rpm
max(max(abs(theta1dot_h)))*rads2rpm]);
fprintf(1,'Max Speedv (RPM)\tMax Speedh (RPM)\t THETA2dot\n');
fprintf(1,'%8.3f\t\t%8.3f\n\n',[max(max(abs(theta2pdot_v)))*rads2rpm
max(max(abs(theta2pdot_h)))*rads2rpm]);
```

APPENDIX D – Torque Calculations

This section is WSU's calculation for motor torque requirements. In the table included, the lengths of each link as well as that of their centers of gravity are listed. Figure D.1 is a simplified diagram of the arm, showing the lengths as displayed in the tables. Similar calculations are performed in the program included in Appendix C. This may be useful for quick hand calculations, substituting the appropriate link lengths in place of those used here.

Weights (lbs)		Lengths (in)	
a	14.0	a	10.2
b	3.06	b	26.4
c	2.18	c	35.4
		x	20.4
		y	12

Weights (lbs)		Lengths (in)	
b	3.06	b-x	6
c	2.18	c-x	15

Shoulder	**Motor**	**451.1**	**lb-in**
Elbow	**Motor**	**76.6**	**lb-in**

Application Factor = 1.5

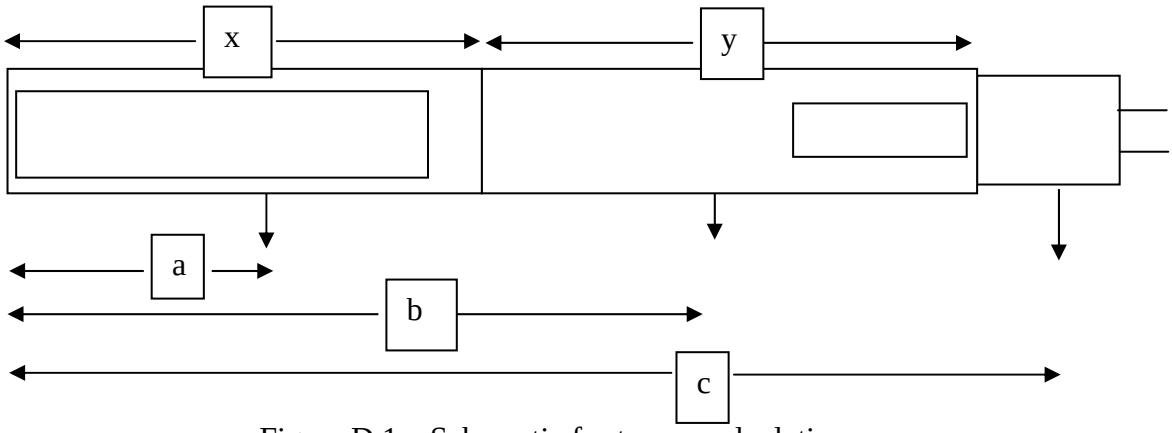
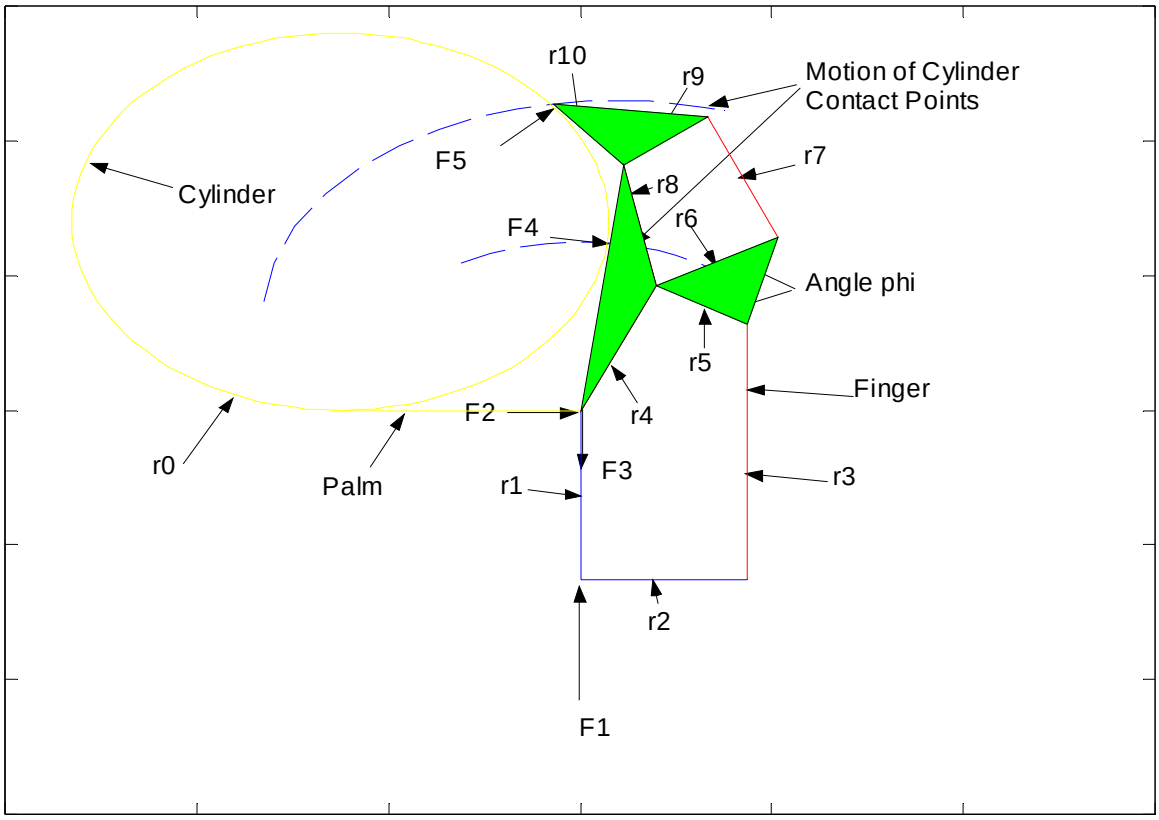


Figure D.1 – Schematic for torque calculations.

APPENDIX E – Finger Geometry Optimization Code

This section contains the Matlab code used to optimize the lengths of the finger links. The optimization was performed to maximize (within reason) gripping force, while minimizing actuation force for several size cylinders as grasping objects. This allowed a smaller, lighter motor to be used to drive the gripping. If another project required similar optimization, perhaps with a different set of constraints, the program could be modified to produce the desired link lengths according to the new restrictions.

Matlab Script used to optimize finger geometry:




```
% Matt Hunt Gateway coalition fin4.m
% kinematic analysis of the underactuated finger

clear all;
clf;
r2=.75;
r3=1.75;
r4=1.0;
r5=.7;
r6=.7;
r7=1.0;
r8=1.0;
r9=.60;
l=1.25;
phi=pi/3; % isolink angle
alpha=pi/2; % tip angle
ski=.29145679; % middle finger anglea1=3*pi/2;
a2=0;
F1=5;
px=0;
py=0;
width= 1;
height= 1;
```

```
 h1=axes('position', [0 0 width height],'box', 'on','xcolor', 'k','ycolor',
'k');   %axis location

rr1=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'b');
rr2=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'b');
rr3=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'r');
rr4=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'r');
rr5=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'g');
rr6=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'g');
rr7=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'r');
rr8=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'r');
rr9=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'g');
rr10=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'g');
rr11=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'y');
rr12=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'g');
rr13=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'y');
rr14=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'g');
base=line('xdata', [], 'ydata' ,[], 'linewidth' ,1, 'erasemode',
'xor','color', 'y');
cylin=patch('xdata', [], 'ydata', [], 'erasemode',
'xor','facecolor','none','edgecolor','y');
force4=line('xdata',[],'ydata',[],'linewidth',1,'linestyle', '--',
'erasemode','none','color','b');
force5=line('xdata',[],'ydata',[],'linewidth',1,'linestyle', '--',
'erasemode','none','color','b');
tip=patch('xdata', [], 'ydata', [], 'erasemode', 'xor','facecolor','g');
iso=patch('xdata', [], 'ydata', [], 'erasemode', 'xor','facecolor','g');
mid=patch('xdata', [], 'ydata', [], 'erasemode', 'xor','facecolor','g');
d0=input('Enter the Clyinder Diamater, d0(in)=');
rcyl=d0/2;

fprintf('Initial Lengths (in)\n');
fprintf('   r2     r3     r4     r5     r6     r7     r8     r9\n');
fprintf('%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f\n\n',...
      r2,r3,r4,r5,r6,r7,r8,r9);


x=[r3,r5,r6,r7,r9,rcyl,r4,r8,r2];
[x,fval,maxfval]=fminimax('funk',x);
r2=x(9);
r3=x(1);
r4=x(7);
r5=x(2);
```

```
r6=x(3);
r7=x(4);
r8=x(8);
r9=x(5);
rguess=x(6);
 fprintf('Optimized Lengths (in)\n');
fprintf('   r2     r3     r4     r5     r6     r7     r8     r9\n\n');
 fprintf('%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f\n\n',...
      r2,r3,r4,r5,r6,r7,r8,r9);
r0=[.5:.1:2.0];
mmm=size(r0);
mmmm=mmm(2);



for q=1:mmmm

  %draw the cylinder on the graph

t=[0:2*pi/100:2*pi];
tt=size(t);
ttt=tt(2);
for tttt=1:ttt
 xx(q,tttt)=-l+r0(q)*cos(t(tttt));
 yy(q,tttt)=r0(q)+r0(q)*sin(t(tttt));
end

a4(q)=pi-2*atan(r0(q)/l)-ski;
a8(q)=a4(q)+ski*2;
r16=sqrt(r4^2+r8^2+2*r4*r8*(sin(a4(q))*sin(a8(q))+cos(a4(q))*cos(a8(q))));
r10=r16-l;

a10(q)=pi+a4(q)+ski-2*atan(r0(q)/(r10));
a9(q)=a10(q)-alpha;

% find the length of r1.

A=(r8^2+r9^2+r6^2-
r7^2+2*r8*r9*(cos(a8(q))*cos(a9(q))+sin(a8(q))*sin(a9(q))))/(-2*r6);
B=(r8*cos(a8(q))+r9*cos(a9(q)));
C=(r8*sin(a8(q))+r9*sin(a9(q)));
r=[(A-B),2*C,(A+B)];
T=roots(r);
at=2*atan(T);
%for n=1:2
 %  if at(n) <= 2*pi/3
  %    if at(n) >= pi/3
          a6(q)=at(2);
   %  end
  % end
%end

a7(q)=atan2((r8*sin(a8(q))+r9*sin(a9(q))-
r6*sin(a6(q))),(r8*cos(a8(q))+r9*cos(a9(q))-r6*cos(a6(q))));

a5(q)=a6(q)-phi;
gama1(q)=pi-a4(q)-a5(q);
```

```
r13(q)=sqrt(r4^2+r5^2+2*r4*r5*(cos(a5(q))*cos(a4(q))+sin(a4(q))*sin(a5(q))));
a13(q)=atan2((r4*sin(a4(q))+r5*sin(a5(q))),(r4*cos(a4(q))+r5*cos(a5(q))));

A=r13(q)^2-r3^2+r2^2-2*r2*r13(q)*cos(a13(q));
r=[1,-2*r13(q)*sin(a13(q)),A];
T=roots(r);
for n=1:2
   if T(n)<0
      if T(n)>-1.7
         r1(q)=T(n);
      end
   end
end


a3(q)=atan2((r13(q)*sin(a13(q))-r1(q)),(r13(q)*cos(a13(q))-r2));

r11(q)=sqrt((r4+r8)^2+r10^2+2*(r8+r4)*r10*(sin(a4(q))*sin(a10(q))+cos(a4(q))*
cos(a10(q))));
a11(q)=atan2(((r4+r8)*sin(a4(q))+r10*sin(a10(q))),((r4+r8)*cos(a4(q))+r10*cos
(a10(q))));


% locations of points
bx(q)=r4*cos(a4(q));
by(q)=r4*sin(a4(q));
cx(q)=0;
cy(q)=r1(q);
dx(q)=r2;
dy(q)=cy(q);
ex(q)=bx(q)+r5*cos(a5(q));
ey(q)=by(q)+r5*sin(a5(q));
ffx(q)=bx(q)+r6*cos(a6(q));
fy(q)=by(q)+r6*sin(a6(q));
ax(q)=bx(q)+r8*cos(a8(q));
ay(q)=by(q)+r8*sin(a8(q));
gx(q)=ax(q)+r9*cos(a9(q));
gy(q)=ay(q)+r9*sin(a9(q));
ff5x(q)=ax(q)+r10*cos(a10(q));
ff5y(q)=ay(q)+r10*sin(a10(q));
ix(q)=ax(q)+r9*cos(a9(q));
iy(q)=ay(q)+r9*sin(a9(q));
ff4x(q)=l*cos(a4(q));
ff4y(q)=l*sin(a4(q));


% force analysis ( the torsion spring is negelcted )
beta=a10+a11;
aalpha=a7-a9;
T=.5;
big=[r10,0,-r9*sin(a7(q)-a9(q)),0,0,0,0,0,0,0,0;... % 1
      sin(a10(q)),0,-sin(a7(q)),1,0,0,0,0,0,0,0;... % 2
      cos(a10(q)),0,cos(a7(q)),0,-1,0,0,0,0,0,0;... % 3
      0,0,0,0,0,sin(a3(q)),0,0,0,0,0;... % 4
      0,0,0,0,0,0,-tan(a3(q)),0,0,0,0;... % 5
      0,0,sin(a7(q)-a6(q)),0,0,-sin(a3(q)-a5(q)),0,0,0,0,0;... % 6
      0,0,cos(a7(q)),0,0,cos(a3(q)),0,1,0,0,0;... % 7
```

```
      0,0,sin(a7(q)),0,0,-sin(a3(q)),0,0,1,0,0;... % 8
       0,l,0,-r16*sin(a4(q)+ski),r16*cos(a4(q)+ski),0,0,-r4*sin(a4(q)),-
r4*cos(a4(q)),0,0;... % 9
      0,0,0,0,1,0,0,0,1,1,0;... % 10
      0,-1,0,1,0,0,0,1,0,0,-1];  % 11




med=[0;0;0;F1;F1;-T;0;0;T;0;0];
s=(big^-1)*med;
f5(q)=s(1);
f4(q)=s(2);
fr7(q)=s(3);
Ax(q)=s(4);
Ay(q)=s(5);
fr3(q)=s(6);
fx(q)=s(7);
Bx(q)=s(8);
By(q)=s(9);
fpy(q)=s(10);
fpx(q)=s(11);
end

fprintf('Forces (lbf)\n\n');
fprintf('   r0      tip     middle  1.0 link   1.75 link    Ax       Ay
\n\n');
for q=1:mmmm

   fprintf('%6.2f   %6.2f   %6.2f   %6.2f     %6.2f    %6.2f   %6.2f \n',...
      r0(q), f5(q), f4(q), fr7(q), fr3(q), Ax(q), Ay(q));
end

fprintf('\n\n   r0       fx      Bx       By      pin y      pin x \n\n');
for q=1:mmmm
   fprintf('%6.2f   %6.2f   %6.2f   %6.2f   %6.2f   %6.2f    \n',...
      r0(q), fx(q),Bx(q), By(q), fpy(q), fpx(q));
end
fprintf('\nMagnitude of pin forces\n\n');
fprintf('   r0       a        b        fp       \n');
for q=1:mmmm
   a(q)=sqrt(Ax(q)^2+Ay(q)^2);
   b(q)=sqrt(Bx(q)^2+Ay(q)^2);
   fp(q)=sqrt(fpx(q)^2+fpy(q)^2);
   fprintf('%6.2f   %6.2f   %6.2f   %6.2f \n',...
      r0(q), a(q), b(q), fp(q));

  end


axis([(-3)*min(r4) (3)*max(r4) (-3)*min(r4) (3)*max(r4)]);


set(force4,'xdata', ff4x,'ydata',ff4y);
set(force5,'xdata', ff5x,'ydata',ff5y);
```

```
% Animate the results.
itotal=mmmm-1;
tt=2;
   for j=1:1:tt;
      for i=1:1:itotal;

         set(rr1,'xdata',[px cx(i)],'ydata',[py cy(i)]);
         set(rr2,'xdata',[cx(i) dx(i)],'ydata',[cy(i) dy(i)]);
         set(rr3,'xdata',[dx(i) ex(i)],'ydata',[dy(i) ey(i)]);
         set(rr4,'xdata',[px bx(i)],'ydata',[py by(i)]);
         set(rr5,'xdata',[bx(i) ex(i)],'ydata',[by(i) ey(i)]);
         set(rr6,'xdata',[bx(i) ffx(i)],'ydata',[by(i) fy(i)]);
         set(rr7,'xdata',[ffx(i) gx(i)],'ydata',[fy(i) gy(i)]);
         set(rr8,'xdata',[bx(i) ax(i)],'ydata',[by(i) ay(i)]);
         set(rr9,'xdata',[ax(i) gx(i)],'ydata',[ay(i) gy(i)]);
         set(rr10,'xdata', [ax(i) ff5x(i)],'ydata',[ay(i) ff5y(i)]);
         %set(rr11,'xdata', [px ff5x(i)],'ydata',[py ff5y(i)]);
         set(rr12,'xdata', [ex(i) ffx(i)],'ydata',[ey(i) fy(i)]);
         %set(rr13,'xdata', [px ex(i)],'ydata',[py ey(i)]);
         set(rr14,'xdata', [gx(i) ff5x(i)],'ydata',[gy(i) ff5y(i)]);
         set(cylin,'xdata', [xx(i,:)], 'ydata', [yy(i,:)]);
         set(base,'xdata', [px -l],'ydata',[py py]);
         set(tip,'xdata', [ff5x(i) ax(i) gx(i)],'ydata',[ff5y(i) ay(i)
gy(i)]);
         set(iso,'xdata', [bx(i) ex(i) ffx(i)],'ydata',[by(i) ey(i) fy(i)]);
         set(mid,'xdata', [bx(i) ax(i) px],'ydata',[by(i) ay(i) py]);



                  pause(.05);
         drawnow;   % Flush the draw buffer
      end
      i=i+1;   %increments for loop
   end      %ends while loop

% Draw in final position

i=10;
         set(rr1,'xdata',[px cx(i)],'ydata',[py cy(i)]);
         set(rr2,'xdata',[cx(i) dx(i)],'ydata',[cy(i) dy(i)]);
         set(rr3,'xdata',[dx(i) ex(i)],'ydata',[dy(i) ey(i)]);
         set(rr4,'xdata',[px bx(i)],'ydata',[py by(i)]);
         set(rr5,'xdata',[bx(i) ex(i)],'ydata',[by(i) ey(i)]);
         set(rr6,'xdata',[bx(i) ffx(i)],'ydata',[by(i) fy(i)]);
         set(rr7,'xdata',[ffx(i) gx(i)],'ydata',[fy(i) gy(i)]);
         set(rr8,'xdata',[bx(i) ax(i)],'ydata',[by(i) ay(i)]);
         set(rr9,'xdata',[ax(i) gx(i)],'ydata',[ay(i) gy(i)]);
         set(rr10,'xdata',[ax(i) ff5x(i)],'ydata',[ay(i) ff5y(i)]);
         %set(rr11,'xdata',[px ff5x(i)],'ydata',[py ff5y(i)]);
         set(rr12,'xdata',[ex(i) ffx(i)],'ydata',[ey(i) fy(i)]);
         %set(rr13,'xdata',[px ex(i)],'ydata',[py ey(i)]);
         set(rr14,'xdata',[gx(i) ff5x(i)],'ydata',[gy(i) ff5y(i)]);
         set(cylin,'xdata', [xx(i,:)], 'ydata', [yy(i,:)]);
         set(tip,'xdata', [ff5x(i) ax(i) gx(i)],'ydata',[ff5y(i) ay(i)
gy(i)]);
```

```
        set(iso,'xdata', [bx(i) ex(i) ffx(i)],'ydata',[by(i) ey(i) fy(i)]);
        set(mid,'xdata', [bx(i) ax(i) px],'ydata',[by(i) ay(i) py]);


%     Draw linkage first

        drawnow;
```

---

funk.m

```
function s = funk (x);

r2=x(9);
r3=x(1);
r4=x(4);
r5=x(2);
r6=x(3);
r8=x(8);

r7=x(4);
r9=x(5);

l=1.25;
phi=pi/3; % isolink angle
alpha=pi/2; % tip angle
ski=.29145679; % middle finger anglea1=3*pi/2;
a2=0;
F1=5;

r0(1)=x(6);
mmm=size(r0);
mmmm=mmm(2);

for q=1:mmmm

 a4(q)=pi-2*atan(r0(q)/l)-ski;
a8(q)=a4(q)+ski*2;
r16=sqrt(r4^2+r8^2+2*r4*r8*(sin(a4(q))*sin(a8(q))+cos(a4(q))*cos(a8(q))));
r10=r16-l;
a10(q)=pi+a4(q)+ski-2*atan(r0(q)/(r10));
a9(q)=a10(q)-alpha;

% find the length of r1.

A=(r8^2+r9^2+r6^2-
r7^2+2*r8*r9*(cos(a8(q))*cos(a9(q))+sin(a8(q))*sin(a9(q))))/(-2*r6);
B=(r8*cos(a8(q))+r9*cos(a9(q)));
C=(r8*sin(a8(q))+r9*sin(a9(q)));
r=[(A-B),2*C,(A+B)];
T=roots(r);
at=2*atan(T);
%for n=1:2
```

```
 %  if at(n) <= 2*pi/3
  %    if at(n) >= pi/3
         a6(q)=at(2);
  %    end
  % end
%end

a7(q)=atan2((r8*sin(a8(q))+r9*sin(a9(q))-
r6*sin(a6(q))),(r8*cos(a8(q))+r9*cos(a9(q))-r6*cos(a6(q))));

a5(q)=a6(q)-phi;
gama1(q)=pi-a4(q)-a5(q);
r13(q)=sqrt(r4^2+r5^2+2*r4*r5*(cos(a5(q))*cos(a4(q))+sin(a4(q))*sin(a5(q))));
a13(q)=atan2((r4*sin(a4(q))+r5*sin(a5(q))),(r4*cos(a4(q))+r5*cos(a5(q))));

A=r13(q)^2-r3^2+r2^2-2*r2*r13(q)*cos(a13(q));
r=[1,-2*r13(q)*sin(a13(q)),A];
T=roots(r);
for n=1:2
   if T(n)<0
      if T(n)>-1.7
         r1(q)=T(n);
      end
   end
end


a3(q)=atan2((r13(q)*sin(a13(q))-r1(q)),(r13(q)*cos(a13(q))-r2));

r11=sqrt((r4+r8)^2+r10^2+2*(r8+r4)*r10*(sin(a4(q))*sin(a10(q))+cos(a4(q))*cos
(a10(q))));
a11(q)=atan2(((r4+r8)*sin(a4(q))+r10*sin(a10(q))),((r4+r8)*cos(a4(q))+r10*cos
(a10(q))));



% force analysis ( the torsion spring is neglected )
T=.5;
big=[r10,0,-r9*sin(a7(q)-a9(q)),0,0,0,0,0,0,0,0;... % 1
      sin(a10(q)),0,-sin(a7(q)),1,0,0,0,0,0,0,0;... % 2
      cos(a10(q)),0,cos(a7(q)),0,-1,0,0,0,0,0,0;... % 3
      0,0,0,0,0,sin(a3(q)),0,0,0,0,0;... % 4
      0,0,0,0,0,0,-tan(a3(q)),0,0,0,0;... % 5
      0,0,sin(a7(q)-a6(q)),0,0,-sin(a3(q)-a5(q)),0,0,0,0,0;... % 6
      0,0,cos(a7(q)),0,0,cos(a3(q)),0,1,0,0,0;... % 7
      0,0,sin(a7(q)),0,0,-sin(a3(q)),0,0,1,0,0;... % 8
       0,l,0,-r16*sin(a4(q)+ski),r16*cos(a4(q)+ski),0,0,-r4*sin(a4(q)),-
r4*cos(a4(q)),0,0;... % 9
      0,0,0,0,1,0,0,0,1,1,0;... % 10
      0,-1,0,1,0,0,0,1,0,0,-1];  % 11



med=[0;0;0;F1;F1;-T;0;0;T;0;0];
```

```
s=(big^-1)*med;
f5(q)=s(1);
f4(q)=s(2);
fr7(q)=s(3);
Ax(q)=s(4);
Ay(q)=s(5);
fr3(q)=s(6);
fx(q)=s(7);
Bx(q)=s(8);
By(q)=s(9);
fpy(q)=s(10);
fpx(q)=s(11);

end
```

Finger Optimization Output:

» fin4
Enter the Clyinder Diamater, d0(in)=3.0
Initial Lengths (in)

r2	r3	r4	r5	r6	r7	r8	r9
0.75	1.75	1.00	0.70	0.70	1.00	1.00	0.60

Optimization terminated successfully:
Magnitude of directional derivative in search direction
less than 2*options.TolFun and maximum constraint violation
is less than options.TolCon
Active Constraints:
6

Optimized Lengths (in)

r2	r3	r4	r5	r6	r7	r8	r9
0.87	1.89	1.00	0.55	0.74	0.96	0.92	0.56

Forces (lbf)

r0	tip	middle	1.0 link	1.75 link	Ax	Ay
0.50	2.84	-0.20	5.84	6.02	2.73	-7.44
0.60	3.21	1.69	5.47	5.58	3.33	-7.83
0.70	3.49	3.02	5.21	5.32	3.40	-8.01
0.80	3.69	3.90	5.02	5.18	3.15	-7.90
0.90	3.82	4.41	4.85	5.09	2.74	-7.53
1.00	3.87	4.63	4.68	5.04	2.28	-6.94
1.10	3.86	4.61	4.50	5.02	1.83	-6.20
1.20	3.79	4.41	4.30	5.01	1.43	-5.37

1.30	3.68	4.11	4.08	5.00	1.09	-4.52
1.40	3.52	3.74	3.84	5.00	0.81	-3.69
1.50	3.33	3.37	3.59	5.00	0.60	-2.92
1.60	3.11	3.01	3.33	5.00	0.43	-2.22
1.70	2.88	2.71	3.07	5.00	0.31	-1.61
1.80	2.64	2.46	2.79	5.00	0.22	-1.10
1.90	2.39	2.29	2.52	5.01	0.16	-0.67
2.00	2.14	2.18	2.25	5.01	0.12	-0.33

r0	fx	Bx	By	pin y	pin x
0.50	3.35	9.17	4.61	2.83	12.10
0.60	2.47	7.76	3.62	4.22	9.40
0.70	1.83	6.59	2.88	5.13	6.97
0.80	1.34	5.58	2.31	5.59	4.83
0.90	0.96	4.69	1.90	5.63	3.02
1.00	0.66	3.89	1.61	5.33	1.54
1.10	0.43	3.17	1.44	4.76	0.40
1.20	0.25	2.53	1.36	4.01	-0.45
1.30	0.11	1.97	1.37	3.15	-1.05
1.40	-0.00	1.48	1.45	2.24	-1.45
1.50	-0.08	1.06	1.59	1.33	-1.71
1.60	-0.14	0.70	1.78	0.45	-1.88
1.70	-0.18	0.41	1.99	-0.38	-1.99
1.80	-0.21	0.17	2.23	-1.14	-2.06
1.90	-0.23	-0.01	2.49	-1.81	-2.13
2.00	-0.24	-0.15	2.75	-2.41	-2.21

Magnitude of pin forces (lbf)

r0	a	b	fp
0.50	7.92	11.81	12.43
0.60	8.51	11.03	10.30
0.70	8.70	10.37	8.65
0.80	8.51	9.67	7.38
0.90	8.01	8.87	6.38
1.00	7.31	7.95	5.55
1.10	6.46	6.96	4.78
1.20	5.56	5.94	4.04
1.30	4.65	4.93	3.32
1.40	3.78	3.98	2.67
1.50	2.98	3.11	2.17
1.60	2.26	2.33	1.93
1.70	1.64	1.66	2.02
1.80	1.12	1.11	2.35

1.90	0.69	0.67	2.80
2.00	0.36	0.37	3.27

References

1. Brayton, R.K., “A New Algorithm for Statistical Circuit Design Based on Quasi-Newton Methods and Functional Splitting” *IEEE Trans. Circuits and Systems*, Vol. CAS-26, pp. 784-794, Sept. 1975.

APPENDIX F – Controls Simulation Code

The code presented in this appendix may be used to simulate intended arm motion. Future uses of the Matlab script could include modification to be driven by an actual joystick input, or simply a modification of the control algorithm. This is a tested program that should function on any version of Matlab 5.3 or higher. Older versions may have difficulty with some of the graphics statements.

```
function controlsim(action)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Developed by:   Fred Griesemer and Rob Siston
% Title:          Senior M.E. Students
% Project:        1999-2000 Gateway Design Team
% Location:       Ohio State University
% Department:     Mechanical Engineering
%
% This is a (physical) 3 link robotic arm.  The upper arm is
% analogous to the humerus - upper arm bone in the human
% body (bone between bicep and tricep).  The forearm is
% self explanatory, while the gripper would be everything
% you see from the wrist to fingertips.
%
% This is specifically intended for children, but will be
% of use to a wide range of people.  Our "consumer" is someone with
% limited use of their limbs, such as quadri- and paraplegics,
% or people with spinal cord injuries C5, C4, and above.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The Matlab editor 'guide' was used in part to create the GUI.
%
% Graphics Handles naming conventions:
% figure: CSFigHandle
% axes: AxesTopHandle
% uicontrol: ControlNameHandle (e.g. StartBtnHandle or Dim1Radio)
%
% Initialize general data carrying variables (Matlab Structure)
% "Data" - stored in the root object UserData
%                   and
% "GUIData" - stored in the figure object UserData
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     Check for inputs and take the proper action.  This functions
%     in some regard as an error trap and prevents multiple instances
%     of the simulation from being opened.  This is a memory intensive
%     and processory intensive program...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin < 1 & isempty(findobj(allchild(0),'Tag','ControlSimFig')),
   action = 'initialize';
elseif nargin < 1 & ~isempty(findobj(allchild(0),'Tag','ControlSimFig')),
   action = 'alreadyopen';
elseif nargin > 0 |~isempty(findobj(allchild(0),'Tag','ControlSimFig')),
   Data = get(0,'UserData');
   GUIData = Data.GUIData;
end

switch action,
case 'initialize',
   StartFigUp;
   StartPosn;
case 'info',
   CSInfo;
case 'close',
   CloseAns = questdlg('Are you sure you want to exit the
simulation?','Exit?','Yes','No','Yes');
```

```
   if strcmp(CloseAns,'Yes'),
      Data = get(0,'UserData');
      GUIData = Data.GUIData;
      close(GUIData.CSFig);
      return;
   elseif strcmp(CloseAns,'No'),
   else,
      msgbox('There has been an invalid entry and will be ignored','Invalid
Entry');
   end
case 'reset',
   StartPosn;
case 'dimension1',
   Data = get(0,'UserData');
   GUIData = Data.GUIData;
   Data.dimension = 1;
   set(GUIData.CSFig,'UserData',Data);
   set(GUIData.Dim2,'Value',0);
   set(GUIData.Dim3,'Value',0);
   set(0,'UserData',Data);
case 'dimension2',
   Data = get(0,'UserData');
   GUIData = Data.GUIData;
   Data.dimension = 2;
   set(GUIData.CSFig,'UserData',Data);
   set(GUIData.Dim1,'Value',0);
   set(GUIData.Dim3,'Value',0);
   set(0,'UserData',Data);
case 'dimension3',
   Data = get(0,'UserData');
   GUIData = Data.GUIData;
   Data.dimension = 3;
   set(GUIData.CSFig,'UserData',Data);
   set(GUIData.Dim1,'Value',0);
   set(GUIData.Dim2,'Value',0);
   set(0,'UserData',Data);
case 'grasp',
   Data = get(0,'UserData');
   GUIData = Data.GUIData;
   if Data.Grasp == 0,  % Check for open fingers.
      set(GUIData.Grasp,'String','Open Fingers', ...
         'BackgroundColor',[0 0 0],'ForegroundColor',[1 1 1]);
      Data.Grasp = 1;
   elseif Data.Grasp == 1, % Check for closed fingers.
      set(GUIData.Grasp,'String','Close Fingers', ...
         'BackgroundColor',[0.7529 0.7529 0.7529],'ForegroundColor',[0 0 0]);
      Data.Grasp = 0;
   end
   set(0,'UserData',Data);
case 'forward', %jfwd',
   TerminAnimator('forward');
case 'back', %'jbwd',
   TerminAnimator('backward');
case 'left', %'jlft',
   TerminAnimator('left');
case 'right', %jrt',
   TerminAnimator('right');
```

```
case 'alreadyopen',
   msgbox('An instance of the Control Simulation is already open.','Instance
Detected');
otherwise,
   msgbox('There has been an error: invalid input to Control
Simulation','Error!');
end

%%%%%%%%%%%%%%%%%
%%  StartPosn  %%
%%%%%%%%%%%%%%%%%
function StartPosn,
Data = get(0,'UserData');
Data.Angles.theta1 = 90*pi/180;  % |
Data.Angles.theta2 = 0;          % |
Data.Angles.theta3 = pi;         % |radians
Data.Angles.phi = 90*pi/180;     % |
Data.Coords = 0;
Data.dimension = 1;
Data.UALength = 18;
Data.FALength = 12;
Data.GRLength = 6;

% Set up axes and viewpoints:
axes(Data.GUIData.Axes3D);
cla;view(3);
axes(Data.GUIData.AxesTop);
cla;view([90 90]);
axes(Data.GUIData.AxesRear);
cla;view([270 0]);
axes(Data.GUIData.AxesSide);
cla;view([0 0]);
Data.GUIData = Data.GUIData;
set(0,'UserData',Data);
TubeCoords;
Data = get(0,'UserData');
GUIData = Data.GUIData;
PatchData = Data.PatchData;

% Create background colors to make understanding the animation
% a little bit easier.
Side1Data = [-30 -30 -10; 30 -30 -10; 30 30 -10;-30 30 -10];
Side2Data = [-30 30 -10; 30 30 -10; 30 30 30;-30 30 30];
Side3Data = [30 -30 -10; 30 30 -10;30 30 30;30 -30 30];
SideFaces = [1 2 3 4];
Side1 = patch('Vertices',Side1Data,'Faces',SideFaces,'FaceColor','flat',...
   'FaceVertexCData',[1 0
0],'CDataMapping','direct','Parent',GUIData.Axes3D);
Side2 = patch('Vertices',Side2Data,'Faces',SideFaces,'FaceColor','flat',...
   'FaceVertexCData',[0 1
0],'CDataMapping','direct','Parent',GUIData.Axes3D);
Side3 = patch('Vertices',Side3Data,'Faces',SideFaces,'FaceColor','flat',...
   'FaceVertexCData',[0 0
1],'CDataMapping','direct','Parent',GUIData.Axes3D);
Side4 = patch('Vertices',Side1Data,'Faces',SideFaces,'FaceColor','flat',...
   'FaceVertexCData',[1 0
0],'CDataMapping','direct','Parent',GUIData.AxesTop);
```

```
Side5 = patch('Vertices',Side2Data,'Faces',SideFaces,'FaceColor','flat',...
   'FaceVertexCData',[0 1
0],'CDataMapping','direct','Parent',GUIData.AxesSide);
Side6 = patch('Vertices',Side3Data,'Faces',SideFaces,'FaceColor','flat',...
   'FaceVertexCData',[0 0
1],'CDataMapping','direct','Parent',GUIData.AxesRear);

% Create an object to grasp, with a top.
[x, y, z] = cylinder;   % Generate a unit cylinder: radius 1", height 1"
z = z*4;                % Increase the height of the cylinder to 4 inches
x = x - 15;             % Shift object by 10" in x-direction
y = y - 10;             % Shift object by 10" in y-direction
Data.CylinderPosn = [-15.5 -10.5 2.0]; % inches
% This generates the "top" since the cylinder is thin-walled (invisible from
above/below)
circ = 0:2*pi/50:2*pi;
for cc = 1:1:length(circ),
   circx(cc) = cos(circ(cc));
   circy(cc) = sin(circ(cc));
end
circx = circx - 15;
circy = circy - 10;

% Draw the arm pieces/initialize the graphics objects:
Rear.UpperArm = patch('Vertices',PatchData.UAVerts,'Faces',PatchData.Faces,
...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesRear);
Rear.ForeArm = patch('Vertices',PatchData.FAVerts,'Faces',PatchData.Faces,
...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesRear);
Rear.Gripper = patch('Vertices',PatchData.GRVerts,'Faces',PatchData.Faces,
...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesRear);
Rear.GraspingObject = surface('XData',x,'YData',y,'ZData',z, ...
   'FaceColor','black','CDataMapping','direct', ...
   'EdgeColor','black','Parent',GUIData.AxesRear);
D3.UpperArm = patch('Vertices',PatchData.UAVerts,'Faces',PatchData.Faces, ...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.Axes3D);
D3.ForeArm = patch('Vertices',PatchData.FAVerts,'Faces',PatchData.Faces, ...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.Axes3D);
D3.Gripper = patch('Vertices',PatchData.GRVerts,'Faces',PatchData.Faces, ...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.Axes3D);
D3.GraspingObject = surface('XData',x,'YData',y,'ZData',z, ...
   'FaceColor','black','CDataMapping','direct', ...
   'EdgeColor','black','Parent',GUIData.Axes3D);
Top.UpperArm = patch('Vertices',PatchData.UAVerts,'Faces',PatchData.Faces,
...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesTop);
Top.ForeArm = patch('Vertices',PatchData.FAVerts,'Faces',PatchData.Faces, ...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
```

```
   'FaceColor','flat','Parent',GUIData.AxesTop);
Top.Gripper = patch('Vertices',PatchData.GRVerts,'Faces',PatchData.Faces, ...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesTop);
Top.GraspingObject = surface('XData',x,'YData',y,'ZData',z, ...
   'FaceColor','black','CDataMapping','direct', ...
   'EdgeColor','black','Parent',GUIData.AxesTop);
Top.GraspingObjectTop = patch('XData',circx,'YData',circy, ...
   'FaceVertexCData',[0 0 0],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesTop);
Side.UpperArm = patch('Vertices',PatchData.UAVerts,'Faces',PatchData.Faces,
...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesSide);
Side.ForeArm = patch('Vertices',PatchData.FAVerts,'Faces',PatchData.Faces,
...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesSide);
Side.Gripper = patch('Vertices',PatchData.GRVerts,'Faces',PatchData.Faces,
...
   'FaceVertexCData',[0.8 0.8 0.8],'CDataMapping','direct', ...
   'FaceColor','flat','Parent',GUIData.AxesSide);
Side.GraspingObject = surface('XData',x,'YData',y,'ZData',z, ...
   'FaceColor','black','CDataMapping','direct', ...
   'EdgeColor','black','Parent',GUIData.AxesSide);

Data.Rear = Rear;
Data.Side = Side;
Data.Top = Top;
Data.D3 = D3;
set(0,'UserData',Data);

%%%%%%%%%%%%%%%%%%%%
%% TerminAnimator %%
%%%%%%%%%%%%%%%%%%%%
function TerminAnimator(incommand),
Data = get(0,'UserData');
d2r = pi/180;                  % Degrees to Radians conversion.
theta1 = Data.Angles.theta1;
theta2 = Data.Angles.theta2;  % theta2 = Theta 2, prime.  This is the angle
subtended by links 1 and 2.
theta3 = Data.Angles.theta3;
phi = Data.Angles.phi;
alpha  =  theta1 + theta2 + pi;  % This is Theta 2: the (absolute) angle of
link 2 measured ccw from (+) x-axis.
r1 = Data.UALength;
r2 = Data.FALength + Data.GRLength;
r3 = r1*cos(theta1) + r2*cos(alpha);
r4 = r1*sin(theta1) + r2*sin(alpha);

switch incommand,
case 'forward',
   if Data.dimension == 1, % Gripper moving in the arm coordinate system x-
direction (only!)
      r3 = r3 + 1;         % increment horizontal pos'n by 1 inch
      if r3 == 0,
         r3 = 0.1;
```

```
      end
      a = r1^2 - r2^2 - r3^2 - r4^2;
      b = 2*r2*r3; c = 2*r2*r4;
      test = c^2 - (a - b)*(a + b);
      if test < 0,
         msgbox('The linkage cannot be assembled.  Choose a different
direction.','Invalid Assembly');
      else,
         alpha = 2*atan((-c + sqrt(c^2 - (a - b)*(a + b)))/(a - b));
         theta1 = acos((r3 - r2*cos(alpha))/r1);
         theta2 = alpha - theta1 - pi;
      end
   elseif Data.dimension == 2,   % Gripper moving in the wheelchair
coordinate system z-direction (only!)
      r4 = r4 + 1;               % increment vertical pos'n by 1 inch
      if r4 == 0,
         r4 = 0.1;
      end
      a = r1^2 - r2^2 - r3^2 - r4^2;
      b = 2*r2*r3; c = 2*r2*r4;
      test = c^2 - (a - b)*(a + b);
      if test < 0,
         msgbox('The linkage cannot be assembled.  Choose a different
direction.','Invalid Assembly');
      else,
         alpha = 2*atan((-c + sqrt(c^2 - (a - b)*(a + b)))/(a - b));
         theta1 = acos((r3 - r2*cos(alpha))/r1);
         theta2 = alpha - theta1 - pi;
      end
   elseif Data.dimension == 3,   % Gripper moving only!
      theta3 = theta3 + 4*d2r;
   end
case 'backward',
   if Data.dimension == 1, % Gripper moving in the arm coordinate system x-
direction (only!)
      r3 = r3 - 1;         % decrement horizontal pos'n by 1 inch
      if r3 == 0,
         r3 = -0.1;
      end
      a = r1^2 - r2^2 - r3^2 - r4^2;
      b = 2*r2*r3;  c = 2*r2*r4;
      test = c^2 - (a - b)*(a + b);
      if test < 0,
         msgbox('The linkage cannot be assembled.  Choose a different
direction.','Invalid Assembly');
      else,
         alpha = 2*atan((-c + sqrt(c^2 - (a - b)*(a + b)))/(a - b));
         theta1 = acos((r3 - r2*cos(alpha))/r1);
         theta2 = alpha - theta1 - pi;
      end
   elseif Data.dimension == 2,   % Gripper moving in the wheelchair
coordinate system z-direction/arm y-direction (only!)
      r4 = r4 - 1;               % decrement vertical pos'n by 1 inch
      if r4 == 0,
         r4 = -0.1;
      end
      a = r1^2 - r2^2 - r3^2 - r4^2;
```

```
      b = 2*r2*r3; c = 2*r2*r4;
      test = c^2 - (a - b)*(a + b);
      if test < 0,
         msgbox('The linkage cannot be assembled.  Move in a different
direction.','Invalid Assembly');
      else,
         alpha = 2*atan((-c + sqrt(c^2 - (a - b)*(a + b)))/(a - b));
         theta1 = acos((r3 - r2*cos(alpha))/r1);
         theta2 = alpha - theta1 - pi;
      end
   elseif Data.dimension == 3,
      theta3 = theta3 - 4*d2r;
   end
case 'left',
   if Data.dimension == 1 |Data.dimension == 2,
      phi = phi + 4*d2r;
   elseif Data.dimension == 3,
   end
case 'right',
   if Data.dimension == 1 |Data.dimension == 2,
      phi = phi - 4*d2r;
   elseif Data.dimension == 3,
   end
otherwise,
   msgbox('An error has occurred in The TerminAnimator.  Invalid
Input.','Wrong Input!');
end
Data.Angles.theta1 = theta1;
Data.Angles.theta2 = theta2;
Data.Angles.theta3 = theta3;
Data.Angles.phi = phi;
set(0,'UserData',Data);
TubeCoords;
Drawer;

%%%%%%%%%%%%%%%%%%%%%%%%
%%      Draw-er       %%
%%%%%%%%%%%%%%%%%%%%%%%%
function Drawer,
Data = get(0,'UserData');
GUIData = Data.GUIData;
PatchData = Data.PatchData;
ErrDist = ((Data.Coords(3,8,1) - Data.CylinderPosn(1))^2 + ...
   (Data.Coords(3,8,2) - Data.CylinderPosn(2))^2 + ...
   (Data.Coords(3,8,3) - Data.CylinderPosn(3))^2)^(1/2);
ErrText = sprintf('Distance from object: %4.2f inches.', ErrDist);
set(Data.GUIData.ErrInfo,'String',ErrText);
set(Data.Rear.UpperArm,'Vertices',PatchData.UAVerts,'Faces',PatchData.Faces);
set(Data.Rear.ForeArm,'Vertices',PatchData.FAVerts,'Faces',PatchData.Faces);
set(Data.Rear.Gripper,'Vertices',PatchData.GRVerts,'Faces',PatchData.Faces);
set(Data.Side.UpperArm,'Vertices',PatchData.UAVerts,'Faces',PatchData.Faces);
set(Data.Side.ForeArm,'Vertices',PatchData.FAVerts,'Faces',PatchData.Faces);
set(Data.Side.Gripper,'Vertices',PatchData.GRVerts,'Faces',PatchData.Faces);
set(Data.D3.UpperArm,'Vertices',PatchData.UAVerts,'Faces',PatchData.Faces);
set(Data.D3.ForeArm,'Vertices',PatchData.FAVerts,'Faces',PatchData.Faces);
set(Data.D3.Gripper,'Vertices',PatchData.GRVerts,'Faces',PatchData.Faces);
set(Data.Top.UpperArm,'Vertices',PatchData.UAVerts,'Faces',PatchData.Faces);
```

```
set(Data.Top.ForeArm,'Vertices',PatchData.FAVerts,'Faces',PatchData.Faces);
set(Data.Top.Gripper,'Vertices',PatchData.GRVerts,'Faces',PatchData.Faces);
drawnow;
Data.GUIData = GUIData;
set(0,'UserData',Data);

%%%%%%%%%%%%%%%%%
%%    CSInfo   %%
%%%%%%%%%%%%%%%%%
function CSInfo,
msgbox('Please refer to the Matlab command window.','Help');
helptxt = ['\n\nThis is a control simulation program meant to closely emulate
the circumstances\n' ...
      'incurred while actually operating the Robotic Assistive Manipulator
being designed\n'...
      'by the 1999 - 2000 Ohio State University Gateway Engineering Coalition
Team.\n' ...
      'As if that weren''t a mouthful enough, the following briefly explains
the\n' ...
      'functionality of the simulation window shown after typing
\"controlsim\"\n' ...
      'at the Matlab Command prompt.\n\n' ...
      'Buttons:\n=======\nInfo - you have already discovered what this is
for.\n' ...
      'Close - Self explanatory, although note that it has a safeguard.\n'
...
      '\tDon''t worry if you accidentally hit this button.\n' ...
      'Reset - clicking this will set all position values of the arm back\n'
...
      '\tto their original value.  This is the \"home\" position.'];
fprintf(1,helptxt);

%%%%%%%%%%%%%%%%%
%  StartFigUp   %
%%%%%%%%%%%%%%%%%
function StartFigUp,
GUIData.CSFig = 0;
load controlsim;

% Configure graphics to fit on any screen.  This prevents malfunction of the
figure
% and/or missing buttons, axes, etc.
set(0,'Units','pixels');
screenpix = get(0,'ScreenSize');
set(0,'Units','inches');
screeninches = get(0,'ScreenSize');
set(0,'Units','pixels');
% Figure:
GUIPos.Fig(1) = 0.15 * screenpix(3);
GUIPos.Fig(2) = 0.10 * screenpix(4);
GUIPos.Fig(3) = 0.75 * screenpix(3);
GUIPos.Fig(4) = 0.75 * screenpix(4);
% 3D Axis View:
GUIPos.Axes3D(1) = 0.45 * GUIPos.Fig(3);
GUIPos.Axes3D(2) = 0.07 * GUIPos.Fig(4);
GUIPos.Axes3D(3) = 0.35 * GUIPos.Fig(3);
GUIPos.Axes3D(4) = 0.40 * GUIPos.Fig(4);
```

```
% Top Axis View:
GUIPos.AxesTop(1) = 0.45 * GUIPos.Fig(3);
GUIPos.AxesTop(2) = 0.55 * GUIPos.Fig(4);
GUIPos.AxesTop(3) = 0.35 * GUIPos.Fig(3);
GUIPos.AxesTop(4) = 0.40 * GUIPos.Fig(4);
% Rear Axes View:
GUIPos.AxesRear(1) = 0.05 * GUIPos.Fig(3);
GUIPos.AxesRear(2) = 0.55 * GUIPos.Fig(4);
GUIPos.AxesRear(3) = 0.35 * GUIPos.Fig(3);
GUIPos.AxesRear(4) = 0.40 * GUIPos.Fig(4);
% Side Axes View:
GUIPos.AxesSide(1) = 0.05 * GUIPos.Fig(3);
GUIPos.AxesSide(2) = 0.07 * GUIPos.Fig(4);
GUIPos.AxesSide(3) = 0.35 * GUIPos.Fig(3);
GUIPos.AxesSide(4) = 0.40 * GUIPos.Fig(4);
% Info Button:
GUIPos.Info(1) = 0.85 * GUIPos.Fig(3);
GUIPos.Info(2) = 0.90 * GUIPos.Fig(4);
GUIPos.Info(3) = 0.10 * GUIPos.Fig(3);
GUIPos.Info(4) = 0.05 * GUIPos.Fig(4);
% Close Button:
GUIPos.Close(1) = 0.85 * GUIPos.Fig(3);
GUIPos.Close(2) = 0.83 * GUIPos.Fig(4);
GUIPos.Close(3) = 0.10 * GUIPos.Fig(3);
GUIPos.Close(4) = 0.05 * GUIPos.Fig(4);
% Reset Button:
GUIPos.Reset(1) = 0.85 * GUIPos.Fig(3);
GUIPos.Reset(2) = 0.76 * GUIPos.Fig(4);
GUIPos.Reset(3) = 0.10 * GUIPos.Fig(3);
GUIPos.Reset(4) = 0.05 * GUIPos.Fig(4);
% Dimension 1:
GUIPos.D1(1) = 0.85 * GUIPos.Fig(3);
GUIPos.D1(2) = 0.69 * GUIPos.Fig(4);
GUIPos.D1(3) = 0.10 * GUIPos.Fig(3);
GUIPos.D1(4) = 0.03 * GUIPos.Fig(4);
% Dimension 2:
GUIPos.D2(1) = 0.85 * GUIPos.Fig(3);
GUIPos.D2(2) = 0.66 * GUIPos.Fig(4);
GUIPos.D2(3) = 0.10 * GUIPos.Fig(3);
GUIPos.D2(4) = 0.03 * GUIPos.Fig(4);
% Dimension 3:
GUIPos.D3(1) = 0.85 * GUIPos.Fig(3);
GUIPos.D3(2) = 0.63 * GUIPos.Fig(4);
GUIPos.D3(3) = 0.10 * GUIPos.Fig(3);
GUIPos.D3(4) = 0.03 * GUIPos.Fig(4);
% Grasp Button/Box:
GUIPos.Grasp(1) = 0.85 * GUIPos.Fig(3);
GUIPos.Grasp(2) = 0.60 * GUIPos.Fig(4);
GUIPos.Grasp(3) = 0.10 * GUIPos.Fig(3);
GUIPos.Grasp(4) = 0.03 * GUIPos.Fig(4);
% Joystick Forward:
GUIPos.JFwd(1) = 0.88 * GUIPos.Fig(3);
GUIPos.JFwd(2) = 0.53 * GUIPos.Fig(4);
GUIPos.JFwd(3) = 0.05 * GUIPos.Fig(3);
GUIPos.JFwd(4) = 0.05 * GUIPos.Fig(4);
% Joystick Backward:
GUIPos.JBwd(1) = 0.88 * GUIPos.Fig(3);
```

```
GUIPos.JBwd(2) = 0.43 * GUIPos.Fig(4);
GUIPos.JBwd(3) = 0.05 * GUIPos.Fig(3);
GUIPos.JBwd(4) = 0.05 * GUIPos.Fig(4);
% Joystick Right:
GUIPos.JRt(1) = 0.93 * GUIPos.Fig(3);
GUIPos.JRt(2) = 0.48 * GUIPos.Fig(4);
GUIPos.JRt(3) = 0.05 * GUIPos.Fig(3);
GUIPos.JRt(4) = 0.05 * GUIPos.Fig(4);
% Joystick Lft:
GUIPos.JLft(1) = 0.83 * GUIPos.Fig(3);
GUIPos.JLft(2) = 0.48 * GUIPos.Fig(4);
GUIPos.JLft(3) = 0.05 * GUIPos.Fig(3);
GUIPos.JLft(4) = 0.05 * GUIPos.Fig(4);
% Error Info Box:
GUIPos.ErrInfo(1) = 0.83 * GUIPos.Fig(3);
GUIPos.ErrInfo(2) = 0.05 * GUIPos.Fig(4);
GUIPos.ErrInfo(3) = 0.15 * GUIPos.Fig(3);
GUIPos.ErrInfo(4) = 0.05 * GUIPos.Fig(4);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  The heart of it all,   %%
%%  the Control Simulation %%
%%  Figure window.         %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CSFig = figure('Color',[0.8 0.8 0.8], ...
   'Colormap',mat0, ...
   'FileName','C:\MATLABR11\work\ControlSim\controlsim.m', ...
   'HandleVisibility','on', ...
   'Interruptible','on', ...
   'Name','Gateway Robotic Arm Control Simulation - The Ohio State University
2000 Team',...
   'PaperPosition',[18 180 576 432], ...
   'PaperUnits','points', ...
   'Position',[GUIPos.Fig], ...
   'ReSize','off', ...
   'Tag','ControlSimFig', ...
   'ToolBar','none', ...
   'Units','pixels');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Background Image Axes %%
%% and Background Image  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
AxesBack = axes('Parent',CSFig, ...
   'Units','pixels', ...
   'CameraUpVector',[0 1 0], ...
   'CameraUpVectorMode','manual', ...
   'Color',[1 1 1], ...
   'ColorOrder',mat1, ...
   'Position',[0 0 GUIPos.Fig(3) GUIPos.Fig(4)], ...
   'Tag','ViewRear', ...
   'XColor',[0 0 0], ...
   'YColor',[0 0 0], ...
   'ZColor',[0 0 0], ...
   'XTick',[],'YTick',[],'ZTick',[],...
   'XTickLabel',{},'YTickLabel',{},'ZTickLabel',{});
load CSFigBack;
backpic = image(figback,'Parent',AxesBack,'Tag','background');
```

```
axis off;

%%%%%%%%%%%%%%%%%%
%% Viewing Axes %%
%%%%%%%%%%%%%%%%%%
AxesRear = axes('Parent',CSFig, ...
   'Units','pixels', ...
   'CameraUpVector',[0 1 0], ...
   'CameraUpVectorMode','manual', ...
   'Color',[1 1 1], ...
   'ColorOrder',mat1, ...
   'Position',[GUIPos.AxesRear], ...
   'Tag','ViewRear', ...
   'XColor',[0 0 0], ...
   'YColor',[0 0 0], ...
   'ZColor',[0 0 0], ...
   'XLim',[-30 30],'YLim',[-30 30],'ZLim',[-10 30], ...
   'XLimMode','manual','YLimMode','manual','ZLimMode','manual', ...
   'XGrid','on','YGrid','on','ZGrid','on', ...
   'XTick',[],'YTick',[],'ZTick',[],...
   'XTickLabel',{},'YTickLabel',{},'ZTickLabel',{});
Rear4Text = text('Parent',AxesRear, ...
   'Color','white', ...
   'FontWeight','bold', ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','center', ...
   'Position',mat2, ...
   'Tag','Axes1Text4', ...
   'VerticalAlignment','cap');
set(get(Rear4Text,'Parent'),'XLabel',Rear4Text);
Rear3Text = text('Parent',AxesRear, ...
   'Color',[0 0 0], ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','center', ...
   'Position',mat3, ...
   'Rotation',90, ...
   'Tag','Axes1Text3', ...
   'VerticalAlignment','baseline');
set(get(Rear3Text,'Parent'),'YLabel',Rear3Text);
Rear2Text = text('Parent',AxesRear, ...
   'Color',[0 0 0], ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','right', ...
   'Position',[-0.1715481171548117 1.087866108786611 9.160254037844386], ...
   'Tag','Axes1Text2', ...
   'Visible','off');
set(get(Rear2Text,'Parent'),'ZLabel',Rear2Text);
Rear1Text = text('Parent',AxesRear, ...
   'Color',[1 1 1], ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','center', ...
   'Position',[0.497907949790795 1.02928870292887 9.160254037844386], ...
   'Tag','Axes1Text1', ...
   'VerticalAlignment','bottom');
set(get(Rear1Text,'Parent'),'Title',Rear1Text);
AxesSide = axes('Parent',CSFig, ...
   'Units','pixels', ...
```

```
    'CameraUpVector',[0 1 0], ...
    'CameraUpVectorMode','manual', ...
    'Color',[1 1 1], ...
    'ColorOrder',mat4, ...
    'Position',[GUIPos.AxesSide], ...
    'Tag','ViewSide', ...
    'XColor',[0 0 0], ...
    'YColor',[0 0 0], ...
    'ZColor',[0 0 0], ...
    'XLim',[-30 30],'YLim',[-30 30],'ZLim',[-10 30], ...
    'XLimMode','manual','YLimMode','manual','ZLimMode','manual', ...
    'XGrid','on','YGrid','on','ZGrid','on', ...
    'XTick',[],'YTick',[],'ZTick',[],...
    'XTickLabel',{},'YTickLabel',{},'ZTickLabel',{});
Side4Text = text('Parent',AxesSide, ...
    'Color',[0 0 0], ...
    'FontWeight','bold', ...
    'HandleVisibility','off', ...
    'HorizontalAlignment','center', ...
    'Position',mat5, ...
    'Tag','Axes2Text4', ...
    'VerticalAlignment','cap');
set(get(Side4Text,'Parent'),'XLabel',Side4Text);
Side3Text = text('Parent',AxesSide, ...
    'Color',[0 0 0], ...
    'HandleVisibility','off', ...
    'HorizontalAlignment','center', ...
    'Position',mat6, ...
    'Rotation',90, ...
    'Tag','Axes2Text3', ...
    'VerticalAlignment','baseline');
set(get(Side3Text,'Parent'),'YLabel',Side3Text);
Side2Text = text('Parent',AxesSide, ...
    'Color',[0 0 0], ...
    'HandleVisibility','off', ...
    'HorizontalAlignment','right', ...
    'Position',[-0.1715481171548117 2.313807531380753 9.160254037844386], ...
    'Tag','Axes2Text2', ...
    'Visible','off');
set(get(Side2Text,'Parent'),'ZLabel',Side2Text);
Side1Text = text('Parent',AxesSide, ...
    'Color',[0 0 0], ...
    'HandleVisibility','off', ...
    'HorizontalAlignment','center', ...
    'Position',[0.497907949790795 1.02928870292887 9.160254037844386], ...
    'Tag','Axes2Text1', ...
    'VerticalAlignment','bottom');
set(get(Side1Text,'Parent'),'Title',Side1Text);
AxesTop = axes('Parent',CSFig, ...
    'Units','pixels', ...
    'CameraUpVector',[0 1 0], ...
    'CameraUpVectorMode','manual', ...
    'Color',[1 1 1], ...
    'ColorOrder',mat7, ...
    'Position',[GUIPos.AxesTop], ...
    'Tag','ViewTop', ...
    'XColor',[0 0 0], ...
```

```
    'YColor',[0 0 0], ...
    'ZColor',[0 0 0], ...
    'XLim',[-30 30],'YLim',[-30 30],'ZLim',[-10 30], ...
    'XLimMode','manual','YLimMode','manual','ZLimMode','manual', ...
    'XGrid','on','YGrid','on','ZGrid','on', ...
    'XTick',[],'YTick',[],'ZTick',[],...
    'XTickLabel',{},'YTickLabel',{},'ZTickLabel',{});
Top4Text = text('Parent',AxesTop, ...
    'Color',[1 1 1], ...
    'FontWeight','bold', ...
    'HandleVisibility','off', ...
    'HorizontalAlignment','center', ...
    'Position',mat8, ...
    'Tag','Axes3Text4', ...
    'VerticalAlignment','cap');
set(get(Top4Text,'Parent'),'XLabel',Top4Text);
Top3Text = text('Parent',AxesTop, ...
    'Color',[0 0 0], ...
    'HandleVisibility','off', ...
    'HorizontalAlignment','center', ...
    'Position',mat9, ...
    'Rotation',90, ...
    'Tag','Axes3Text3', ...
    'VerticalAlignment','baseline');
set(get(Top3Text,'Parent'),'YLabel',Top3Text);
Top2Text = text('Parent',AxesTop, ...
    'Color',[0 0 0], ...
    'HandleVisibility','off', ...
    'HorizontalAlignment','right', ...
    'Position',[-1.414225941422594 1.083682008368201 9.160254037844386], ...
    'Tag','Axes3Text2', ...
    'Visible','off');
set(get(Top2Text,'Parent'),'ZLabel',Top2Text);
Top1Text = text('Parent',AxesTop, ...
    'Color',[0 0 0], ...
    'HandleVisibility','off', ...
    'HorizontalAlignment','center', ...
    'Position',[0.497907949790795 1.02928870292887 9.160254037844386], ...
    'Tag','Axes3Text1', ...
    'VerticalAlignment','bottom');
set(get(Top1Text,'Parent'),'Title',Top1Text);
Axes3D = axes('Parent',CSFig, ...
    'Units','pixels', ...
    'CameraUpVector',[0 1 0], ...
    'CameraUpVectorMode','manual', ...
    'Color',[1 1 1], ...
    'ColorOrder',mat10, ...
    'Position',[GUIPos.Axes3D], ...
    'Tag','View3D', ...
    'XColor',[0 0 0], ...
    'YColor',[0 0 0], ...
    'ZColor',[0 0 0], ...
    'XLim',[-30 30],'YLim',[-30 30],'ZLim',[-10 30], ...
    'XLimMode','manual','YLimMode','manual','ZLimMode','manual', ...
    'XGrid','on','YGrid','on','ZGrid','on', ...
    'XTick',[],'YTick',[],'ZTick',[],...
    'XTickLabel',{},'YTickLabel',{},'ZTickLabel',{});
```

```
D34Text = text('Parent',Axes3D, ...
   'Color',[1 1 1], ...
   'FontWeight','bold', ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','center', ...
   'Position',mat11, ...
   'Tag','Axes4Text4', ...
   'VerticalAlignment','cap');
set(get(D34Text,'Parent'),'XLabel',D34Text);
D33Text = text('Parent',Axes3D, ...
   'Color',[0 0 0], ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','center', ...
   'Position',mat12, ...
   'Rotation',90, ...
   'Tag','Axes4Text3', ...
   'VerticalAlignment','baseline');
set(get(D33Text,'Parent'),'YLabel',D33Text);
D32Text = text('Parent',Axes3D, ...
   'Color',[0 0 0], ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','right', ...
   'Position',[-1.414225941422594 2.313807531380753 9.160254037844386], ...
   'Tag','Axes4Text2', ...
   'Visible','off');
set(get(D32Text,'Parent'),'ZLabel',D32Text);
D31Text = text('Parent',Axes3D, ...
   'Color',[0 0 0], ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','center', ...
   'Position',[0.497907949790795 1.02928870292887 9.160254037844386], ...
   'Tag','Axes4Text1', ...
   'VerticalAlignment','bottom');
set(get(D31Text,'Parent'),'Title',D31Text);

%%%%%%%%%%%%%%%%%%%%%%
% Buttons & Controls %
%%%%%%%%%%%%%%%%%%%%%%
InfoBtn = uicontrol('Parent',CSFig, ...
   'Units','pixels', ...
   'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588],
...
   'ListboxTop',0, ...
   'Position',[GUIPos.Info], ...
   'String','Info', ...
   'Tag','Infobutton', ...
   'TooltipString','Brief information/help for Control Simulation', ...
   'Callback','controlsim info');

CloseBtn = uicontrol('Parent',CSFig, ...
   'Units','pixels', ...
   'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588],
...
   'ListboxTop',0, ...
   'Position',[GUIPos.Close], ...
   'String','Close', ...
   'Tag','Closebutton', ...
```

```
   'TooltipString','Exit the simulation.',...
   'Callback','controlsim close');

ResetBtn = uicontrol('Parent',CSFig, ...
   'Units','pixels', ...
   'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588],
...
   'ListboxTop',0, ...
   'Position',[GUIPos.Reset], ...
   'String','Reset', ...
   'Tag','Resetbutton', ...
   'TooltipString','Click here to restore default position.', ...
   'Callback','controlsim reset');

Dim2Radio = uicontrol('Parent',CSFig, ...
   'Units','pixels', ...
   'BackgroundColor',[0 0 0], ...
   'ForegroundColor',[1 1 1], ...
   'ListboxTop',0, ...
   'Position',[GUIPos.D2], ...
   'String','Dimension 2', ...
   'Style','radiobutton', ...
   'Tag','Dimension2', ...
   'TooltipString','Select the dimension you wish to control.', ...
   'Value',0, ...
   'Callback','controlsim dimension2');

Dim3Radio = uicontrol('Parent',CSFig, ...
   'Units','pixels', ...
   'BackgroundColor',[0 0 0], ...
   'ForegroundColor',[1 1 1], ...
   'ListboxTop',0, ...
   'Position',[GUIPos.D3], ...
   'String','Dimension 3', ...
   'Style','radiobutton', ...
   'Tag','Dimension3', ...
   'TooltipString','Select the dimension you wish to control.', ...
   'Value',0, ...
   'Callback','controlsim dimension3');

Dim1Radio = uicontrol('Parent',CSFig, ...
   'Units','pixels', ...
   'BackgroundColor',[0 0 0], ...
   'ForegroundColor',[1 1 1], ...
   'ListboxTop',0, ...
   'Position',[GUIPos.D1], ...
   'String','Dimension 1', ...
   'Style','radiobutton', ...
   'Tag','Dimension1', ...
   'TooltipString','Select the dimension you wish to control.', ...
   'Value',1, ...
   'Callback','controlsim dimension1');

GraspBtn = uicontrol('Parent',CSFig, ...
   'Units','pixels', ...
   'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588],
...
```

```
    'ListboxTop',0, ...
    'Position',[GUIPos.Grasp], ...
    'String','Close Fingers', ...
    'Tag','Graspbutton', ...
    'TooltipString','Click to open/close fingers.', ...
    'Callback','controlsim grasp');

JoyFwd = uicontrol('Parent',CSFig, ...
    'Units','pixels', ...
    'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588],
...
    'Interruptible','on', ...
    'ListboxTop',0, ...
    'Position',[GUIPos.JFwd], ...
    'Tag','JoystickForward', ...
    'String','Fwd', ...
    'Callback','controlsim forward');

JoyRt = uicontrol('Parent',CSFig, ...
    'Units','pixels', ...
    'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588],
...
    'ListboxTop',0, ...
    'Position',[GUIPos.JRt], ...
    'Tag','JoystickRight', ...
    'String','Right', ...
    'Callback','controlsim right');

JoyLft = uicontrol('Parent',CSFig, ...
    'Units','pixels', ...
    'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588],
...
    'ListboxTop',0, ...
    'Position',[GUIPos.JLft], ...
    'Tag','JoystickLeft', ...
    'String','Left', ...
    'Callback','controlsim left');

JoyBwd = uicontrol('Parent',CSFig, ...
    'Units','pixels', ...
    'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588],
...
    'ListboxTop',0, ...
    'Position',[GUIPos.JBwd], ...
    'Tag','JoystickBackward', ...
    'String','Bwd', ...
    'Callback','controlsim back');

ErrInfoText = ['Distance from object: 20 inches.'];
ErrInfoBox = uicontrol('Parent',CSFig, ...
    'Units','pixels', ...
    'ListboxTop',0, ...
    'Position',[GUIPos.ErrInfo], ...
    'Style','text', ...
    'String',ErrInfoText, ...
    'Tag','ErrorInfoBox');
```

```
if nargout > 0, fig = CSFig; end

GUIData.CSFig = CSFig;
GUIData.AxesTop = AxesTop;
GUIData.AxesRear = AxesRear;
GUIData.AxesSide = AxesSide;
GUIData.Axes3D = Axes3D;
GUIData.Dim1 = Dim1Radio;
GUIData.Dim2 = Dim2Radio;
GUIData.Dim3 = Dim3Radio;
GUIData.Grasp = GraspBtn;
GUIData.JFwd = JoyFwd;
GUIData.JBwd = JoyBwd;
GUIData.JLft = JoyLft;
GUIData.JRt = JoyRt;
GUIData.Info = InfoBtn;
GUIData.Close = CloseBtn;
GUIData.Reset = ResetBtn;
GUIData.ErrInfo = ErrInfoBox;
Data.GUIData = GUIData;
Data.Grasp = 0;         % 0 = Open, 1 = Closed.
set(0,'UserData',Data);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Kinematic Calculation     %%
%%  of joint angles, and tube %%
%%  coordinates for animation %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Coordinate Transformations %%
%% and kinematic programming  %%
%% done by Rob Siston.        %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function TubeCoords,

% The variable c used in this function is
% representative of the coordinate locations
% c = corner, the index represents the link
% (index 1) and corner number (index 2)
%
% Define constant variables
% (all dimensions in inches)
Data  =  get(0,'UserData');
link1 = Data.UALength;  % Upper Arm Length (inches)
link2 = Data.FALength;  % ForeArm Length (inches)
link3 = Data.GRLength;  % Gripper/Wrist Assembly Length (inches)
c  =  zeros(3,8);
HalfTubeDiam = 2.5/2; % Square Tube, Outside "diameter"

% Angular Inputs
theta1  =  Data.Angles.theta1;
theta2  =  Data.Angles.theta2;
theta3  =  Data.Angles.theta3;
phi  =  Data.Angles.phi;
alpha  =  theta1 + theta2 + pi;
beta  =  theta1 + theta2 + theta3;

% Position Kinematics
```

```
xprime = link1*(cos(theta1)) + link2*(cos(alpha)) + link3*(cos(beta));
zprime = link1*(sin(theta1)) + link2*(sin(alpha)) + link3*(sin(beta));

transphi =  [cos(phi) -sin(phi) 0; sin(phi) cos(phi) 0; 0 0 1];
transtheta = [ cos(theta1) 0  -sin(theta1); 0 1 0; sin(theta1) 0
cos(theta1)];
posxz = [ xprime 0 zprime]';
pos = transphi*posxz;

% The following section locates the corners of the tubes
% This data is then used to draw the arm elements for animation.
%
% Upper Arm Coords
x1prime = link1*(cos(theta1));
z1prime = link1*(sin(theta1));

posxz1 = [ x1prime 0 z1prime]';
pos1 = transphi*posxz1;

UpperArm = pos1;  % this will give the xyz for end of lower arm in the lower
arm coordinates

% General Corner Coordinate Transformations
c1 = [0, HalfTubeDiam  ,-HalfTubeDiam  ]';
c2 = [0, HalfTubeDiam  , HalfTubeDiam  ]';
c3 = [0,-HalfTubeDiam  , HalfTubeDiam  ]';
c4 = [0,-HalfTubeDiam  ,-HalfTubeDiam  ]';
c5 = [0, HalfTubeDiam  ,-HalfTubeDiam  ]';
c6 = transphi*[0, HalfTubeDiam  , HalfTubeDiam  ]';
c7 = transphi*[0,-HalfTubeDiam  , HalfTubeDiam  ]';
c8 = transphi*[0,-HalfTubeDiam  ,-HalfTubeDiam  ]';

% Shoulder (never translates).  Movement is restricted to rotation about the
origin vertical (z) axis
c(1,1,1:3) = transphi*[transtheta*c1];
c(1,2,1:3) = transphi*[transtheta*c2];
c(1,3,1:3) = transphi*[transtheta*c3];
c(1,4,1:3) = transphi*[transtheta*c4];

% Upper Arm, by elbow
c(1,5,1:3) = transphi*[posxz1+transtheta*c1];
c(1,6,1:3) = transphi*[posxz1+transtheta*c2];
c(1,7,1:3) = transphi*[posxz1+transtheta*c3];
c(1,8,1:3) = transphi*[posxz1+transtheta*c4];

% Forearm
x2prime = link1*(cos(theta1)) + link2*(cos(alpha));
z2prime = link1*(sin(theta1)) + link2*(sin(alpha));
posxz2 = [ x2prime 0 z2prime]';
pos2 = transphi*posxz2;
transalpha = [ cos(alpha) 0  -sin(alpha); 0 1 0; sin(alpha) 0 cos(alpha)];
Lowerarm = pos2;

% Near the elbow
c(2,1,1:3) = transphi*[posxz1+transalpha*c1];
c(2,2,1:3) = transphi*[posxz1+transalpha*c2];
c(2,3,1:3) = transphi*[posxz1+transalpha*c3];
```

```
c(2,4,1:3) = transphi*[posxz1+transalpha*c4];

% Near the wrist
c(2,5,1:3) = transphi*[posxz2+transalpha*c1];
c(2,6,1:3) = transphi*[posxz2+transalpha*c2];
c(2,7,1:3) = transphi*[posxz2+transalpha*c3];
c(2,8,1:3) = transphi*[posxz2+transalpha*c4];

% For the gripper
transbeta = [ cos(beta) 0  -sin(beta); 0 1 0; sin(beta) 0 cos(beta)];
Gripper = pos;

% Near the wrist
c(3,1,1:3) = transphi*[posxz2+transbeta*c1];
c(3,2,1:3) = transphi*[posxz2+transbeta*c2];
c(3,3,1:3) = transphi*[posxz2+transbeta*c3];
c(3,4,1:3) = transphi*[posxz2+transbeta*c4];

% Near the fingers
c(3,5,1:3) = transphi*[posxz+transbeta*c1];
c(3,6,1:3) = transphi*[posxz+transbeta*c2];
c(3,7,1:3) = transphi*[posxz+transbeta*c3];
c(3,8,1:3) = transphi*[posxz+transbeta*c4];
Data.Coords = c;

% Write the gripper's grasping centerpoint position to the following
% variable.  This will be used to update the onscreen information
% box at the lower right of the window.
Data.GripPosn = Gripper;

% UpperArm Vertices stored in Data.Coords(1,:,:)
% ForeArm vertices stored in Data.Coords(2,:,:)
% Gripper vertices stored in Data.Coords(3,:,:)
% UA = UpperArm, FA = ForeArm, GR = Gripper
PatchData.Faces = [1 2 3 4; 1 2 6 5; 1 5 8 4; 3 4 8 7; 2 3 7 6; 5 6 7 8];

PatchData.UAVerts = [Data.Coords(1,1,1) Data.Coords(1,1,2)
Data.Coords(1,1,3); ...
      Data.Coords(1,2,1) Data.Coords(1,2,2) Data.Coords(1,2,3); ...
      Data.Coords(1,3,1) Data.Coords(1,3,2) Data.Coords(1,3,3); ...
      Data.Coords(1,4,1) Data.Coords(1,4,2) Data.Coords(1,4,3); ...
      Data.Coords(1,5,1) Data.Coords(1,5,2) Data.Coords(1,5,3); ...
      Data.Coords(1,6,1) Data.Coords(1,6,2) Data.Coords(1,6,3); ...
      Data.Coords(1,7,1) Data.Coords(1,7,2) Data.Coords(1,7,3); ...
      Data.Coords(1,8,1) Data.Coords(1,8,2) Data.Coords(1,8,3)];

PatchData.FAVerts = [Data.Coords(2,1,1) Data.Coords(2,1,2)
Data.Coords(2,1,3); ...
      Data.Coords(2,2,1) Data.Coords(2,2,2) Data.Coords(2,2,3); ...
      Data.Coords(2,3,1) Data.Coords(2,3,2) Data.Coords(2,3,3); ...
      Data.Coords(2,4,1) Data.Coords(2,4,2) Data.Coords(2,4,3); ...
      Data.Coords(2,5,1) Data.Coords(2,5,2) Data.Coords(2,5,3); ...
      Data.Coords(2,6,1) Data.Coords(2,6,2) Data.Coords(2,6,3); ...
      Data.Coords(2,7,1) Data.Coords(2,7,2) Data.Coords(2,7,3); ...
      Data.Coords(2,8,1) Data.Coords(2,8,2) Data.Coords(2,8,3)];
```

```
PatchData.GRVerts = [Data.Coords(3,1,1) Data.Coords(3,1,2)
Data.Coords(3,1,3); ...
      Data.Coords(3,2,1) Data.Coords(3,2,2) Data.Coords(3,2,3); ...
      Data.Coords(3,3,1) Data.Coords(3,3,2) Data.Coords(3,3,3); ...
      Data.Coords(3,4,1) Data.Coords(3,4,2) Data.Coords(3,4,3); ...
      Data.Coords(3,5,1) Data.Coords(3,5,2) Data.Coords(3,5,3); ...
      Data.Coords(3,6,1) Data.Coords(3,6,2) Data.Coords(3,6,3); ...
      Data.Coords(3,7,1) Data.Coords(3,7,2) Data.Coords(3,7,3); ...
      Data.Coords(3,8,1) Data.Coords(3,8,2) Data.Coords(3,8,3)];

Data.PatchData = PatchData;
set(0,'UserData',Data);
```

APPENDIX G – Final Design Revisions

A good deal of work was performed on the arm during the final few weeks of the project. Some of this was to improve the design of some parts, while some was to redo some parts that were not properly machined the first time.

The first major change was the replacement of the C33-I-200E08 K & D Magmotor at the shoulder shaft. After consulting members of the schools, it was found that a drill motor could provide the needed torque to drive the shoulder while significantly saving on the weight and cost of the entire system. Specifically, the motor from a Porter Cable, 19.2V, Model 884, 1/2" Cordless Drill/Driver was used at the shoulder joint. This motor provided 390 in-lb of torque, which was more than sufficient for the needs of this mechanism.

The procedure to implement this type of change was fairly simple to perform. First, the drill motor, gear box, and clutch were removed from its casing, and the leads to the trigger were also severed. A box was made from a 0.060 aluminum sheet to surround the gearbox and transmit the torque from the motor to the motor mount. The motor mount for the drill motor adapted the bolt pattern on the gearbox to the existing bolt pattern of the K&D Magmotor, and mounted the motor and gearbox to the base. Some of the features of the drill motor needed to be modified before it could be used for our application. First, the clutch was locked

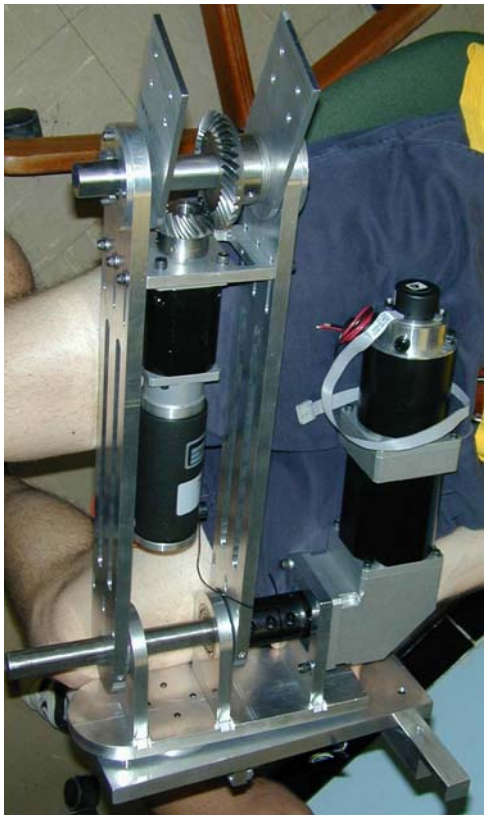


Figure G.1 - Arm Assembly with C33-I-200E08 K & D Magmotor at the Shoulder Shaft

in place to prevent slipping. Six bolts were pressed into the clutch plate and fixed to the motor mounting plate. The gearbox was then locked in its highest output torque setting with a hose-clamp. It was necessary to machine the end of the shoulder shaft to mate with the gearbox. With these modifications in place the drill motor could power the arm.

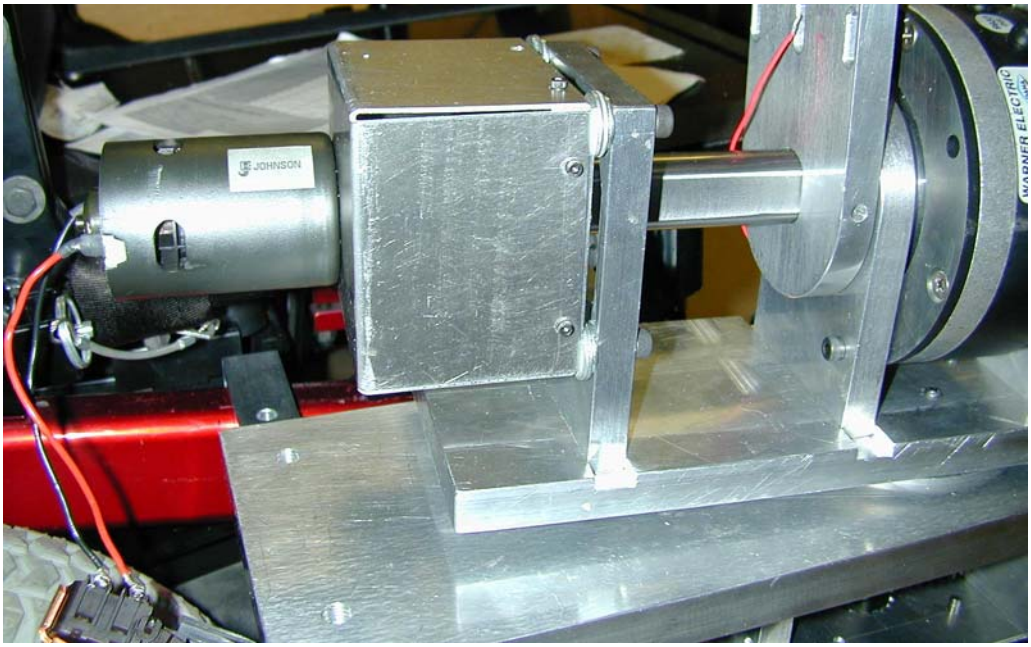


Figure G.2 - Drill Motor at the Shoulder Shaft

Significant gains were obtained from the replacement of the K&D Magmotor with the drill motor. First, a tremendous weight savings was obtained. The previous motor and corresponding right angle gearhead weighed 20 lb. The drill motor weighed a total of approximately 2 lb. As the base and the upper arm weighed around 50 lb and 18 lb a 36% reduction in weight could be enormously beneficial to arm longevity, wheelchair battery life, and general aesthetics. Secondly, the cost of this portion of the arm was significantly reduced. The drill package including: batteries, chargers, carrying case, plus all the parts of the actual drill (chuck, trigger, handle, etc), cost $250 for the shoulder. At a cost of $25 per hour, the machining bill for the conversion was $125. The sheet metal needed to mount the motor was also an extremely minor cost. Even if one didn't estimate a cost reduction for all the parts not needed (batteries and chargers, carrying case, etc), the cost savings was more than $1200, which is approximately a 71% reduction in cost.

The next significant change involved the mounting of the arm to the wheelchair. The original mounting device did not accommodate for the incline of the frame on the chair, thus the entire robotic arm was slanted forward when mounted on the wheelchair. This was a problem for two reasons and could not be neglected. First, and most obvious, there existed a great need to have the arm level to make its usage as easy and natural as possible. However, the declined mount was also unstable. If the center of mass of the shoulder motor would rotate to the outside of the swiveling axis (as was encountered frequently) the entire arm would swing completely forward, as the swivel motor did not have sufficient back-driving torque to counter this moment. It would have been impossible to

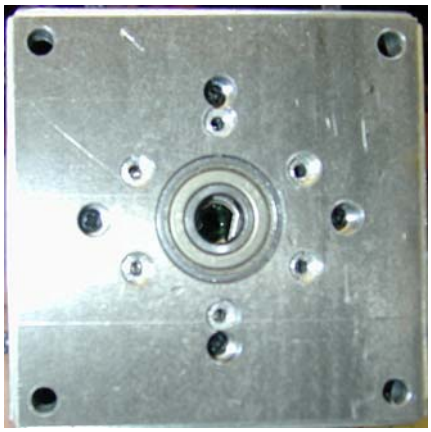


Figure G.3 - Drill Motor Mounting Plate

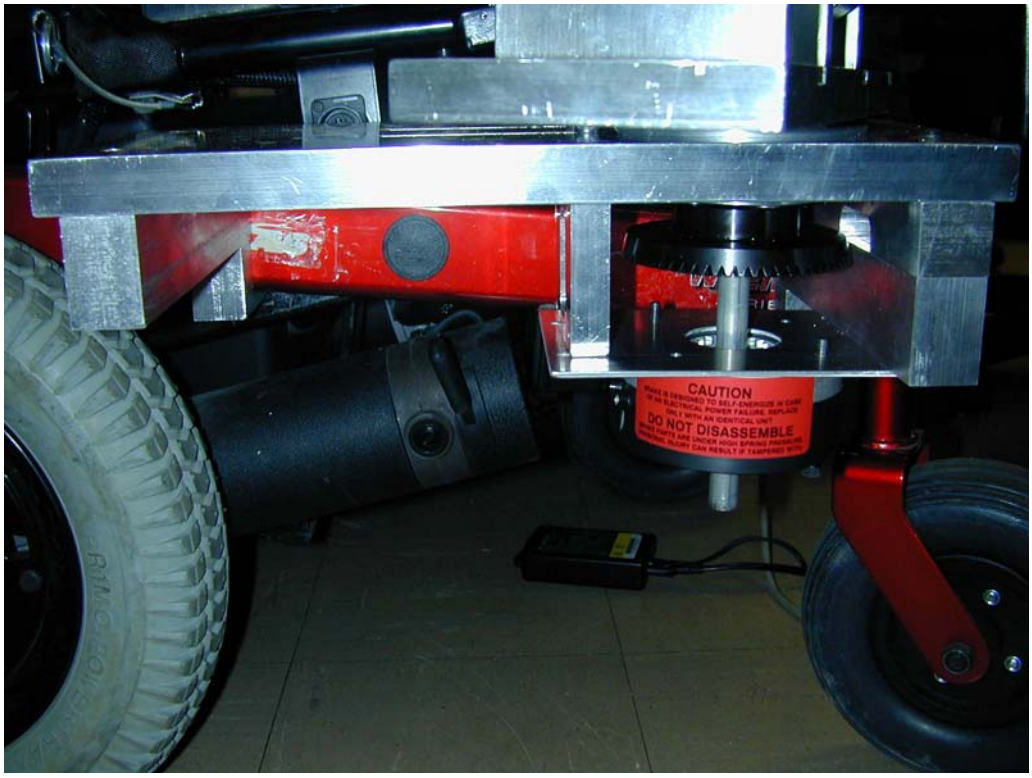
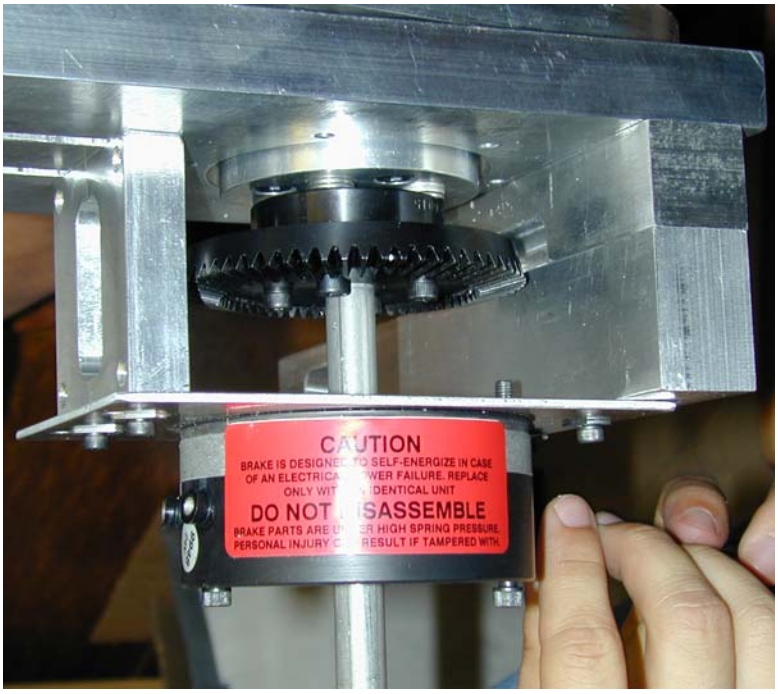


Figure G.4 - New Mounting Brackets (Attached to Wheelchair)

control the arm on a level surface under this condition. As a result of this, the mounted brackets were redesigned. First, an angle was machined into the clamping portion of each bracket to account for the incline of the wheelchair frame. Next, an extra spacer was added to the mounting bracket closer to the front of the wheelchair. These modifications ensured that the base structure of the arm was mounted parallel to the ground.

However, there was a concern that the arm would still freewheel while traveling on inclined terrain. For that reason, another brake was used to stop the swivel action. This provided added control while manipulating the arm and guaranteed that it would remain in the home position when not in use. To accomplish this, a 6 inch section of ½ inch 6061 AL rod was machined and press fit into the swivel gear. A simple mounting plate then held the brake in place, as it spanned the swivel motor mount and the newly machined mounting bracket. The brake, the same model used at the elbow, was fastened to the mounting plate, a 0.050" thick sheet metal, with #10 machine screws.



Other minor design changes were needed to mount the other two brakes. The elbow brake had the easiest mount. Two holes were drilled into the side of the upper arm and then tapped for machine screws. The shoulder brake had a slightly more

complicated mounting scheme. First, a single hole was drilled into one of the shoulder pillow blocks. To provide further support and prevent rotation, a new mounting plate was added in the slot next to this piece and had a hole drilled and tapped into it. Screws

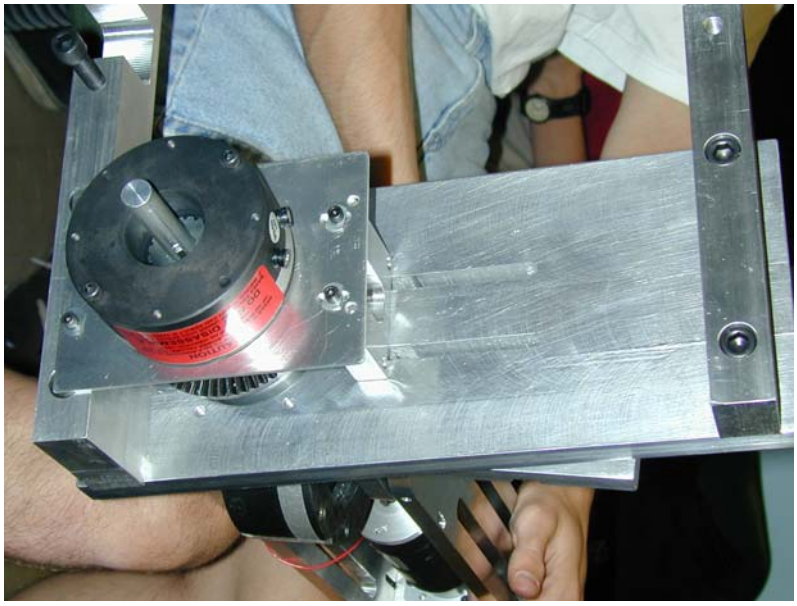


Figure G.5 and G.6 – Mounting the Swivel Brake

through these two holes in the two plates mounted the brake. This held the brake in place during operation, since when the brake is disengaged the disk is free to move (play of about 0.25-0.5 inches) inside the brake housing. This is actually beneficial since it accommodates radial shaft play and does not sacrifice holding torque.

Modifications were needed to both the elbow and shoulder shafts as well. The elbow shaft had a keyway cut into the end to mount the brake. Also, an appropriately sized hole was drilled into its center, about the long axis, to mount the potentiometer. The shoulder shaft required much more machining. As the initial design was milled flat across its length and prevented proper mounting of the brake, a new shaft was made. This one was fabricated from ground steel. Only the ends of the shaft were machined with a flat

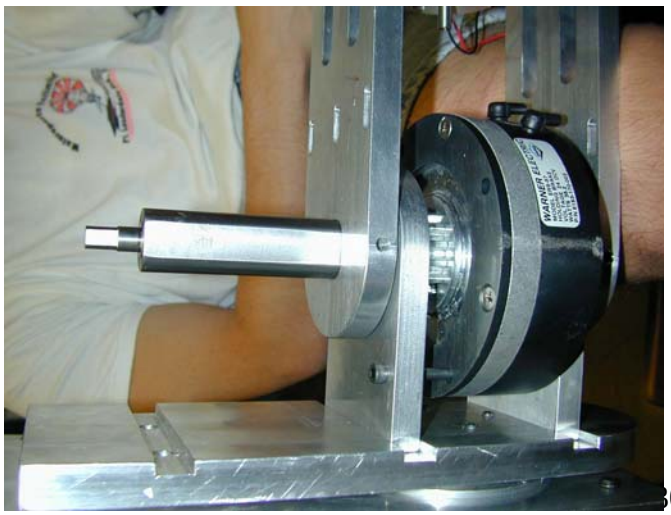


Figure AG.7 – Modified shoulder shaft

across them, as the middle of the shaft was left circular to ensure a press fit with the bearings in the shoulder. A keyway was also machined into this section to mount the shoulder brake. One end of the shaft was then turned down from the 1" diameter to engage the shoulder motor. Machining the end of the shaft to fit the motor eliminated the need for a coupling.

In a minor change, elbow shaft bearing retainers (resembling sturdier custom-made washers) were removed from both sides of the arm. Their original intent was to hold the elbow bearings in place, however the bearings had a sufficient press fit as to keep them in place. The retaining ring was also not needed as a result. This way, the brake and the potentiometer mounted easier than with them in place.

Lastly, spacers were needed to join the upper arm to the forearm sections, as over a ½" gap existed between the square tubing of the forearm and the sides of the upper arm. The spacers were turned from a ¾" aluminum rod and parted to the appropriate length.