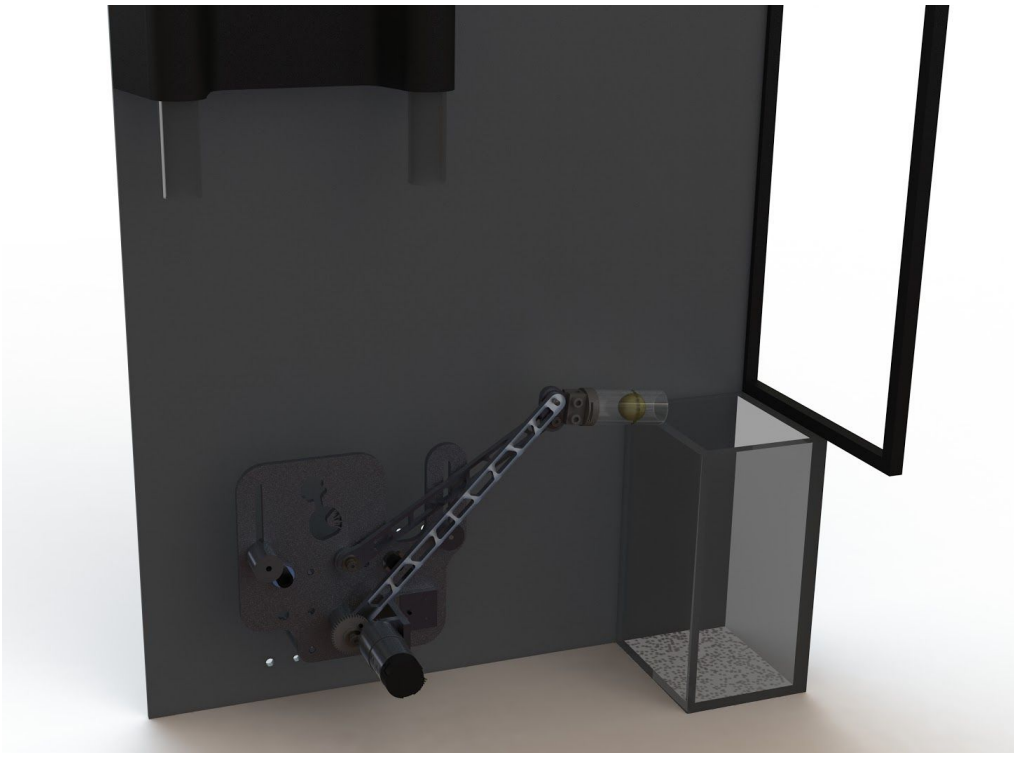
ME350
WN17 Semester
FINAL REPORT

Team 52

Austin Broda
Marcos Cavallin
Nikko Van Crey
David Van Dyke
Mitchell Williams

GSI: Kyle Kenkel



"We have fully abided by the University of Michigan College of Engineering Honor Code"

_____

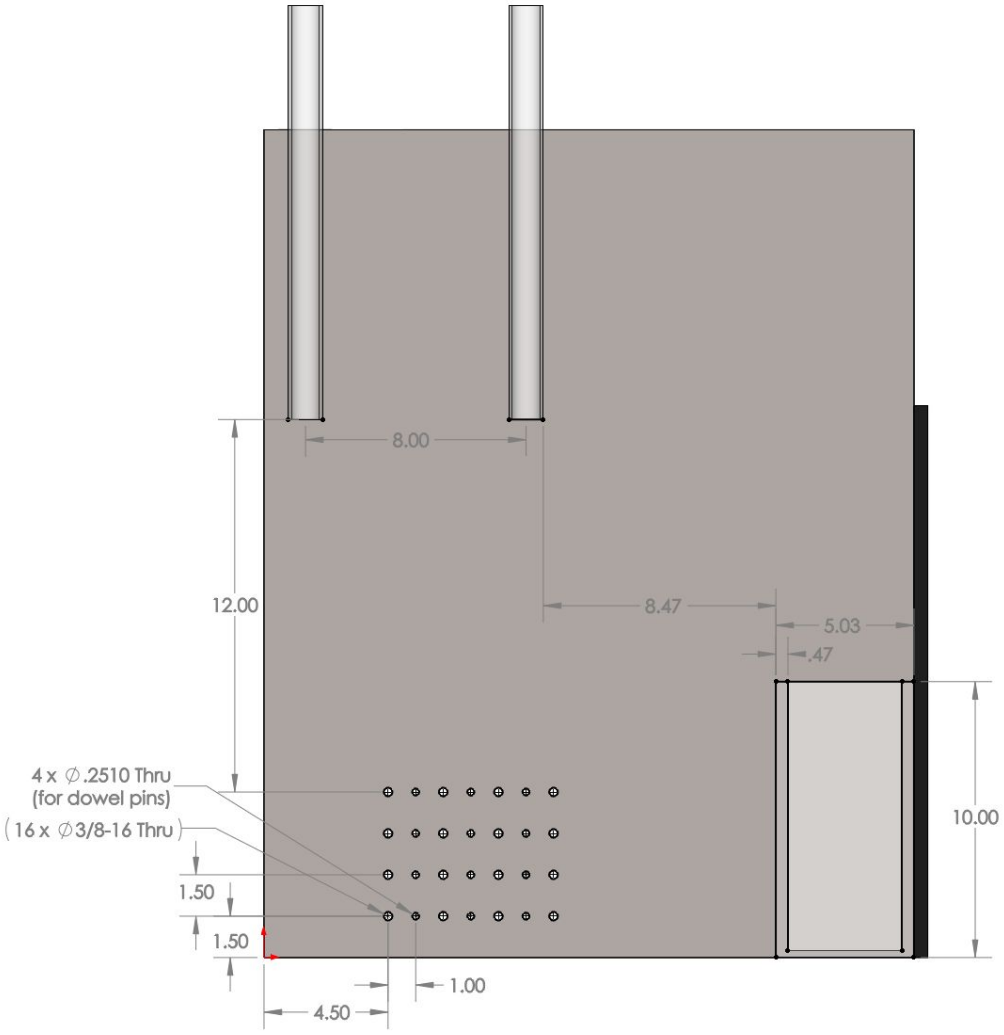
_____

_____

_____

Section 1: Project Introduction

The motivation for this project is to design, build, and test an automated mechanism that catches balls in a cup, and reliably deposits them into the correct basket. In doing so, we will learn methods in which to efficiently design mechanical linkages, how to program an arduino, and how to seamlessly integrate electrical and mechanical systems. This class will introduce the process of manufacturing parts based solely on the engineering prints drafted by other individuals, as well as creating universal drawings that can be read by any manufacturer. The environment that the mechanism must work in is shown in Figure 1.1, with the mounting points at the bottom left corner and ball capture and release points above and to the right, respectively.



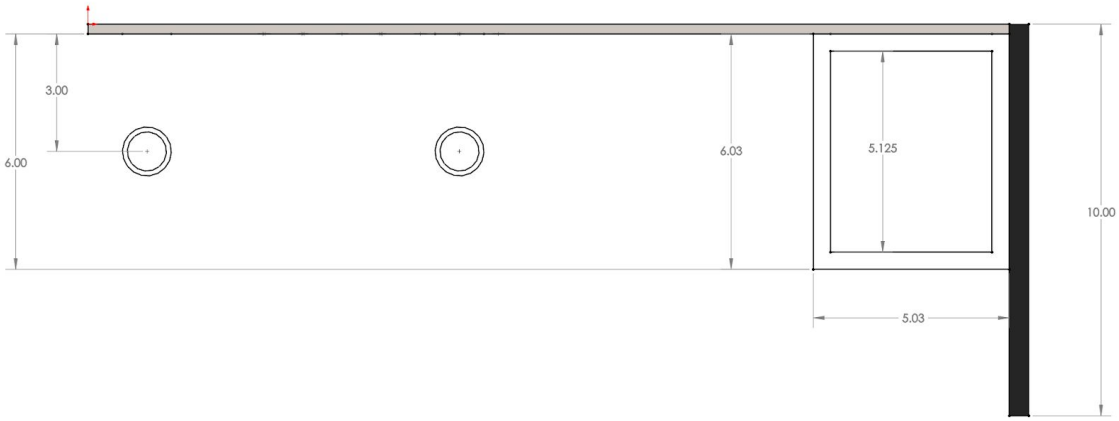


Figure 1.1: Front and Top View of the Playing Field

This project has many "hard metrics" that limit our design. The mechanism must be able to fit within the arena assembly and be manufactured in the winter 2017 semester using only machines available in the machine shop. Aside from the $100 allotted budget, the materials we use to make our mechanism such as metal alloys, motors, and fasteners, must be from those provided. In order for the design to meet the target pickup and drop off points, the transmission angle must remain between 30 and 150 degrees throughout the entirety of the mechanism's motion, so that the linkage does not buckle or bind. In practice the speed of the communication between sensors and motors will be limited by the processing speed of the Arduino microcontroller board.

Table 1.1: Hard Specifications for Project

Hard Specifications	Target Values
Arena Assembly (Volume, Mounting, Deposit/Goal Locations)	Dimensions: See Figure 1 & 2 Less than 15,000 in^3
Machines in X50 shop	Manual Mill Manual Lathe Arbor Press Drill Press etc.
Allotted Budget	$100
Time	Winter 2017 Semester
Transmission Angle	30° < angle < 150°
Arduino microcontroller board	16MHz CPU

Ball Size	~1”

There are also several soft project metrics that distinguish performance of mechanisms. One of these includes the manufacturing skills of those that are creating our parts. The skills of the engineers producing our parts is not known, thus asking for difficult machining processes is not possible. General craftsmanship can also affect the project's performance and safety of the machine itself. Another metric includes the number of parts our mechanism is composed of. We cannot create a machine with excessive components because another manufacturer must produce these parts. General safety precautions should also be met, such as limiting the speed of the machine so it is not dangerous or removing sharp edges from design.

Some other restrictions in design include the mechanism being able to catch the ball in a 1 or 2 inch cup from the chutes (figure 3), recognizing the color of the ball received, and dropping the ball in the correct bucket based on its color. The machine must be able to drop the maize and blue ball within the area specified in the bucket shown in figure 3 and toss the scarlet and grey balls in the net also shown in Figure 2.

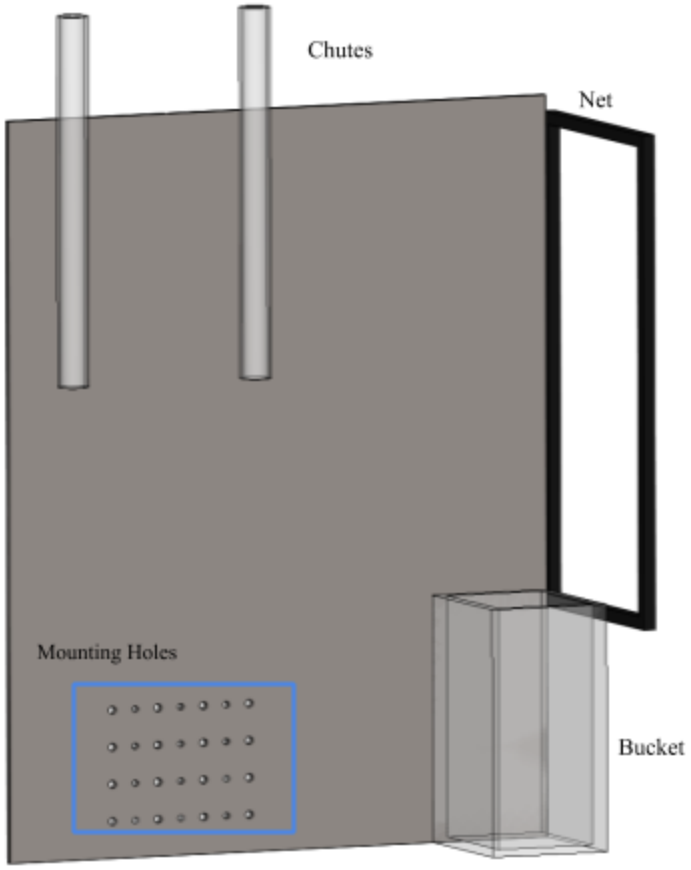


Figure 1.2: Isometric View of the Playing Field

Section 2: Design Processes

The goal of the design process is to create a mechanism capable of moving the cup to the desired positions and orientations with a single motor and single cup. Important things to consider in this design are the orientation of the cup with the ball droppers and goal and the transmission angle. It is ideal for the cup to be close to vertical with the droppers to make it easier for the ball to drop into the cup. The transmission angle, the acute angle between the coupler and the follower, should be between 30° and 150° in order the maintain a high mechanical advantage. Having an angle that exceeds these bounds makes the mechanism less efficient. A low moment of inertia is also important as this will allow the mechanism to move quickly and complete its objective in the allocated time.

Initial designs to find link lengths and positions were made is SolidWorks. Using sketch blocks, we created positions for the cups and found where the ground pivots would end up on the board. Sketch relations allowed us try many positions since the entire sketch would automatically update with each change.

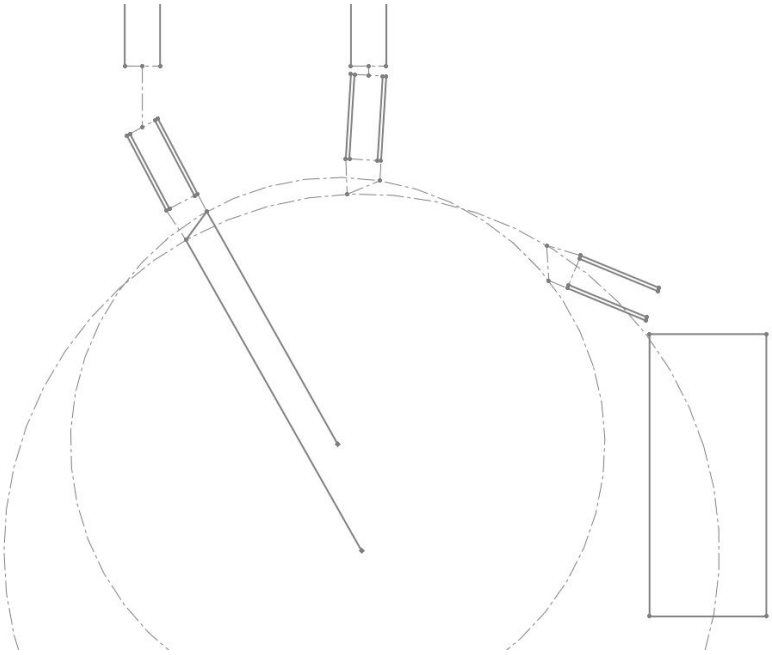


Figure 2.1: Initial SolidWorks Sketch Block

We created 3D models of our mechanisms in SolidWorks. We then created a dynamic model of the mechanism in Adams to see if the provided motor delivers enough power to meet the motion requirements of the design. We also analyzed the angular velocity and torque of our mechanism. Using this information from the Adams model, we compared each of our designs and selected the one that best met our specifications.

Section 3: Design Selection

The Pugh chart that we used to select a final design is depicted below. Our strategy is to use a 1 inch cup in order to gain the most amount of points possible. This is why the angle of the cup relative to the tube that drops the marbles is given a value of 2 because it is important that no marbles are dropped. Having the angle of the cup relative to the tube be low helps prevent this from happening as having an angled cup results in a greater chance that the marble will not fall in the cup. The angle relative to the basket at the final position is given a value of 1 because the speed of the mechanism is more important to get the marble in the basket or net. The power required is given the high value of 3 because if the mechanism uses more power than the motor can provide, it will not be able to move and we will be unable to move any marbles. The transmission angle deviation is give the highest weight of 4 because if it exceeds 60 degrees at any point in the motion, efficiency is greatly reduced. Speed is given the value of 2 because the mechanism must be able to move quickly to deposit marbles into the basket. The ADAMS analysis for each design that we used to determine values for the Pugh chart is depicted in table 3.2 below.

Table 3.1: Pugh Chart of Designs

Requirement	Weight	Mitch's Design	Nikko's Design	David's Design	Austin's Design	Marcos' Design
Position 1 angle compared to tube	2	1	0	0	0	0
Position 2 angle compare to tube	2	0	0	1	0	1
Final Position angle relative to basket	1	0	1	1	0	1
Power required	3	1	0	0	0	0
Transmission Angle Deviation	3	1	0	0	0	0
Speed	2	0	0	0	0	0
Total		8	1	3	0	3

After comparing each of the designs, we we decided Mitch's design was the best option for our strategy. Designs were very similar in terms of power required and speed, but had slightly different angles relative to each tube in the first two locations. Mitch's design had the polycarbonate tube directly underneath the marble drop tube and nearly vertical whereas Marcos' design was not as straight. The angle of the cup is very important for our strategy, since we are using the one inch cup. Being able to catch the cup was also seen as very important, simply because it assured a given point value of 10 (double of dumping the ball). We determined that Mitch's design would be able to catch the marbles much more reliably and effectively and therefore is the design we have chosen. Summary of these choices is shown in table 3.2 below.

Table 3.2: Analysis of Each Design

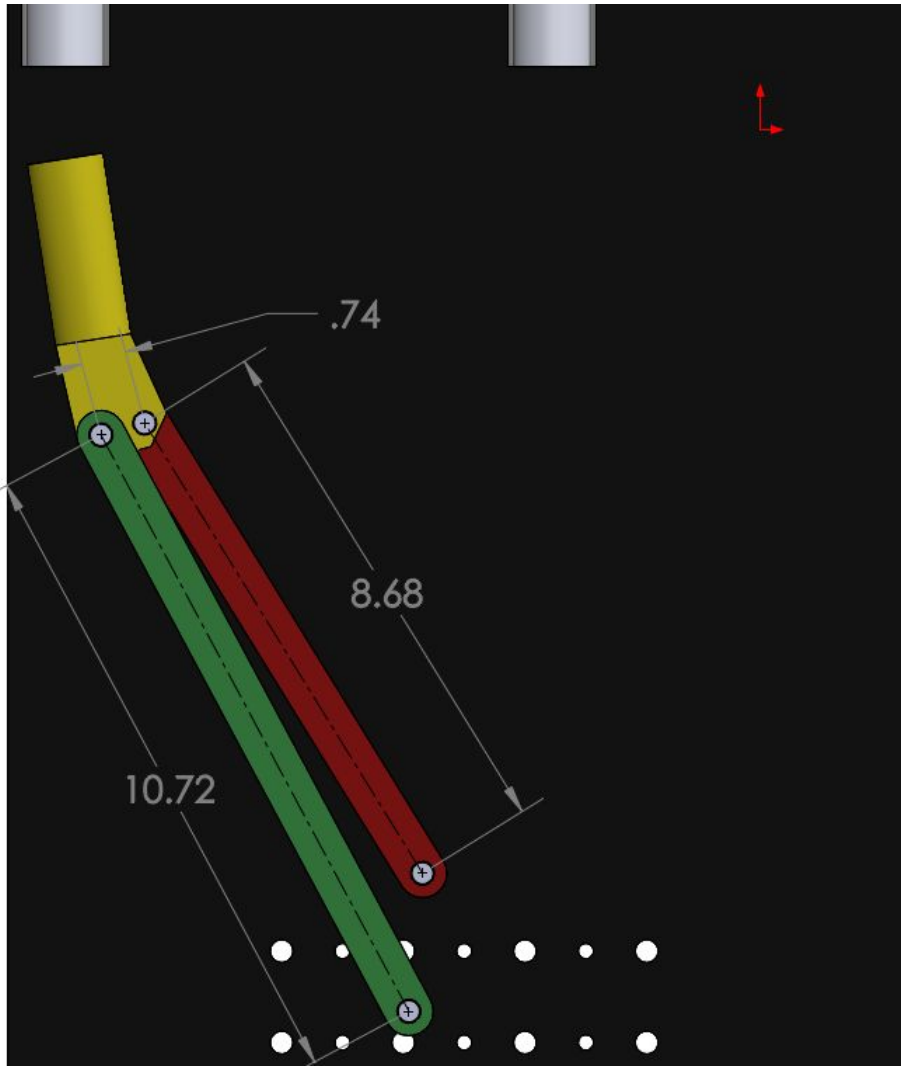
	Mitch's Design	Marcos' Design	Nikko's Design	David's Design	Austin's Design
Power Output [N-m]	.4	.5	.5	.6	.4
Volume Measurements [in^3]	6.85	13.82	8.68	15.67	7.55
Initial Transmission Angle Deviation [Degrees]	19.73	25.5	13.98	-24.41	8.78
Final Transmission Angle Deviation [Degrees]	37.24	11.3	19.4	-34.03	41.98

Section 4: Final Linkage Design

Table 4.1 and figure 4.1 show the 3 link lengths of the mechanism from the ground points to their connection to the coupler itself.

Table 4.1 & Figure 4.1: Length of Each Link

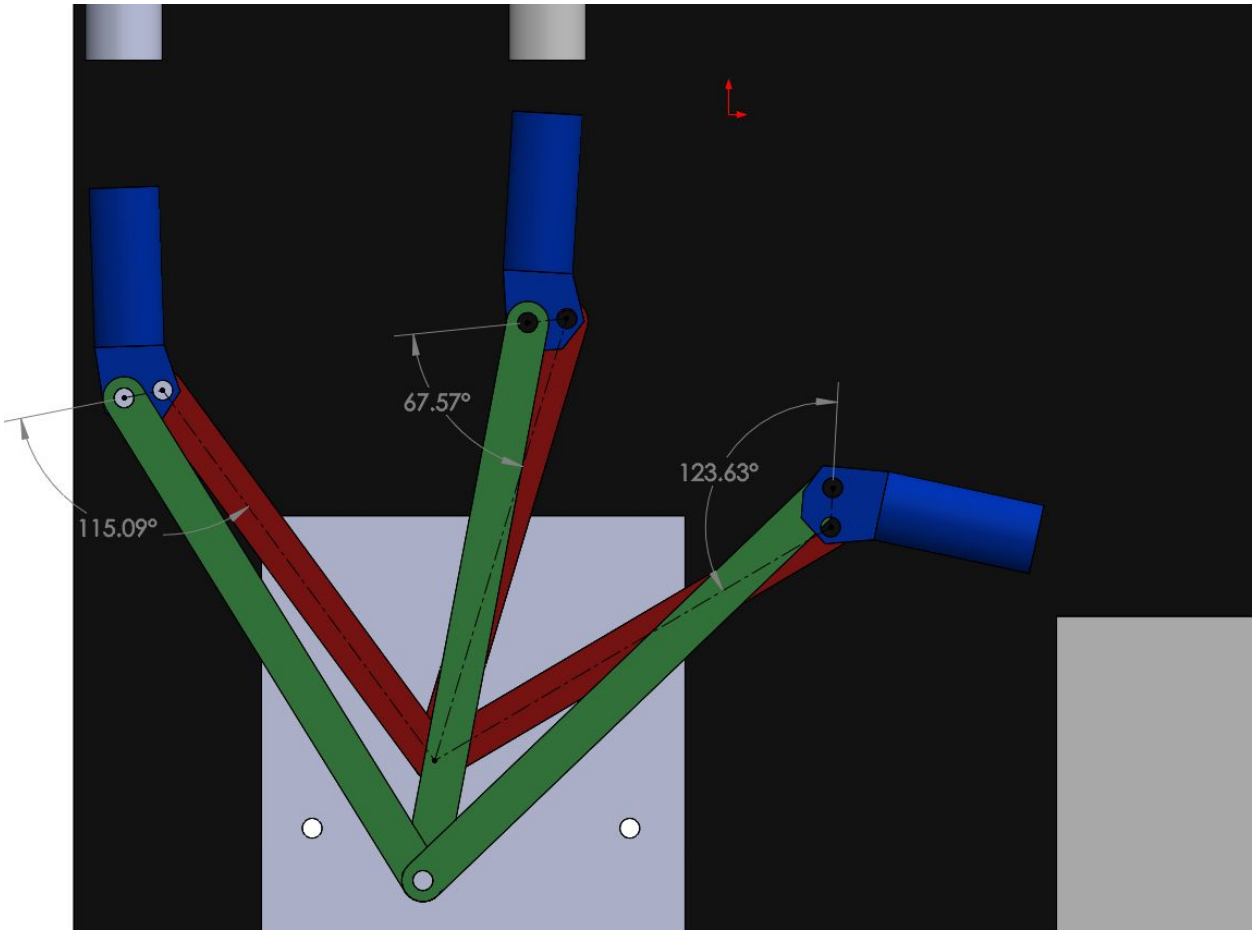
Link	Final Link Lengths [Inches]
Input	10.72
Follower	8.68
Coupler	.74



The table below shows the transmission angle at each of the 3 cup positions. It is important that these values do not exceed an angle of 150° or 30° so that the machine does not bind.

Table 4.2 & Figure 4.2: Transmission Angle for All Cup Positions

Cup Position	Transmission Angle [Degrees]
Under the left tube	115.09
Under the right tube	67.57
Over the basket	123.63



The following table shows the deviation between the measured transmission angles to 90°. Comparing our angles to those allowed, our design will stay within the allowed interval of plus or minus 60°. This is important to avoid binding in the mechanism.

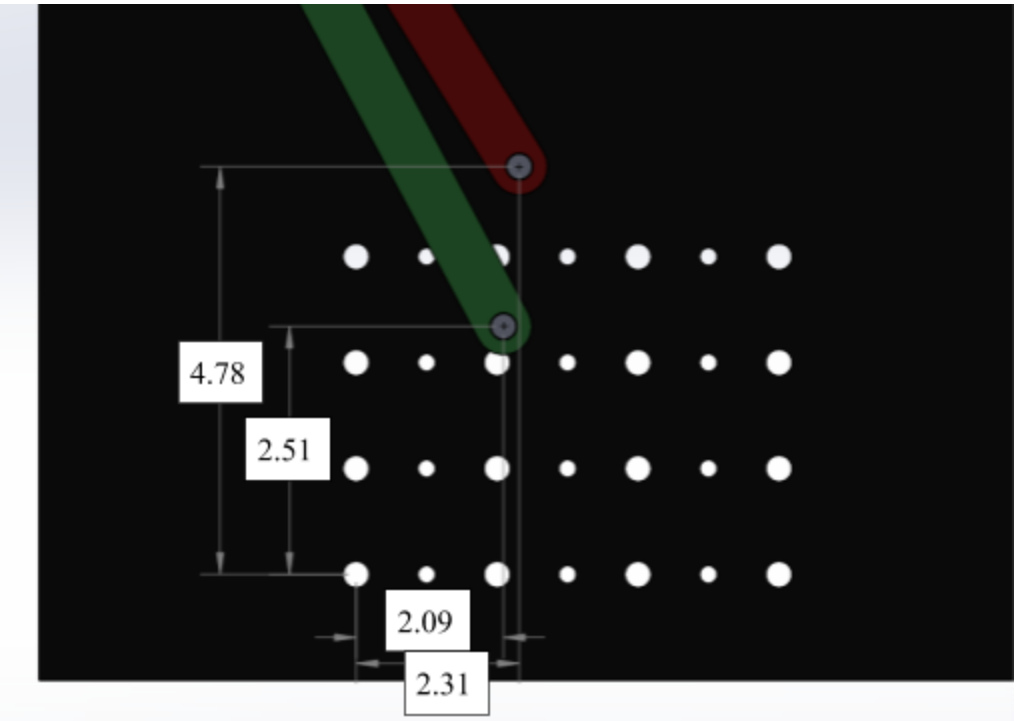
Table 4.3: Transmission angle deviation for all cup positions

Cup Position	Transmission Angle Deviation
Under the left tube	25.09
Under the right tube	22.43
Over the basket	33.63

Other important values include our distances to the bottom left mounting hole. This is to ensure that the ground pivots are in a relatively close location to the mounting area so that design of a base plate will be easier.

Table 4.4 & Figure 4.3: Location of the Ground Pivots Relative to Bottom Left Mounting Hole

Pivot	Locations <X Coordinate, Y Coordinate> [inches]
Input ground pivot	<2.09, 2.51>
Follower ground pivot	<2.32, 4.78>



For our project, we decided to choose the 1 inch cup. This decision was reached after deciding it was worth creating a more challenging scenario for more points.

Table 4.5: Size of Cup Inner Diameter

Size of Inner Diameter	1”

Section 5: Final Team CAD

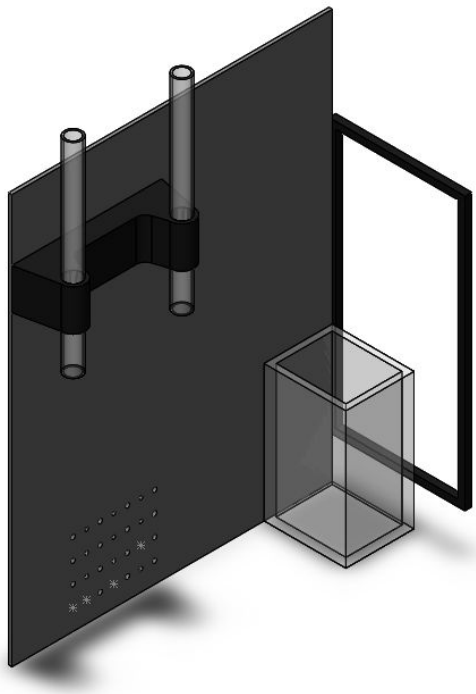


Figure 5.1: Model of Playing Field made with Lab Measurements

The mechanism is almost entirely made of 0.25 inch thick aluminum, with the exception of the 3-D printed coupler, and the cup 1" polycarbonate cup attached to the top of the coupler. Most of the components in this mechanism can be manufactured using the waterjet, or the 3-D printer. The mechanism can be seen below from the different views.

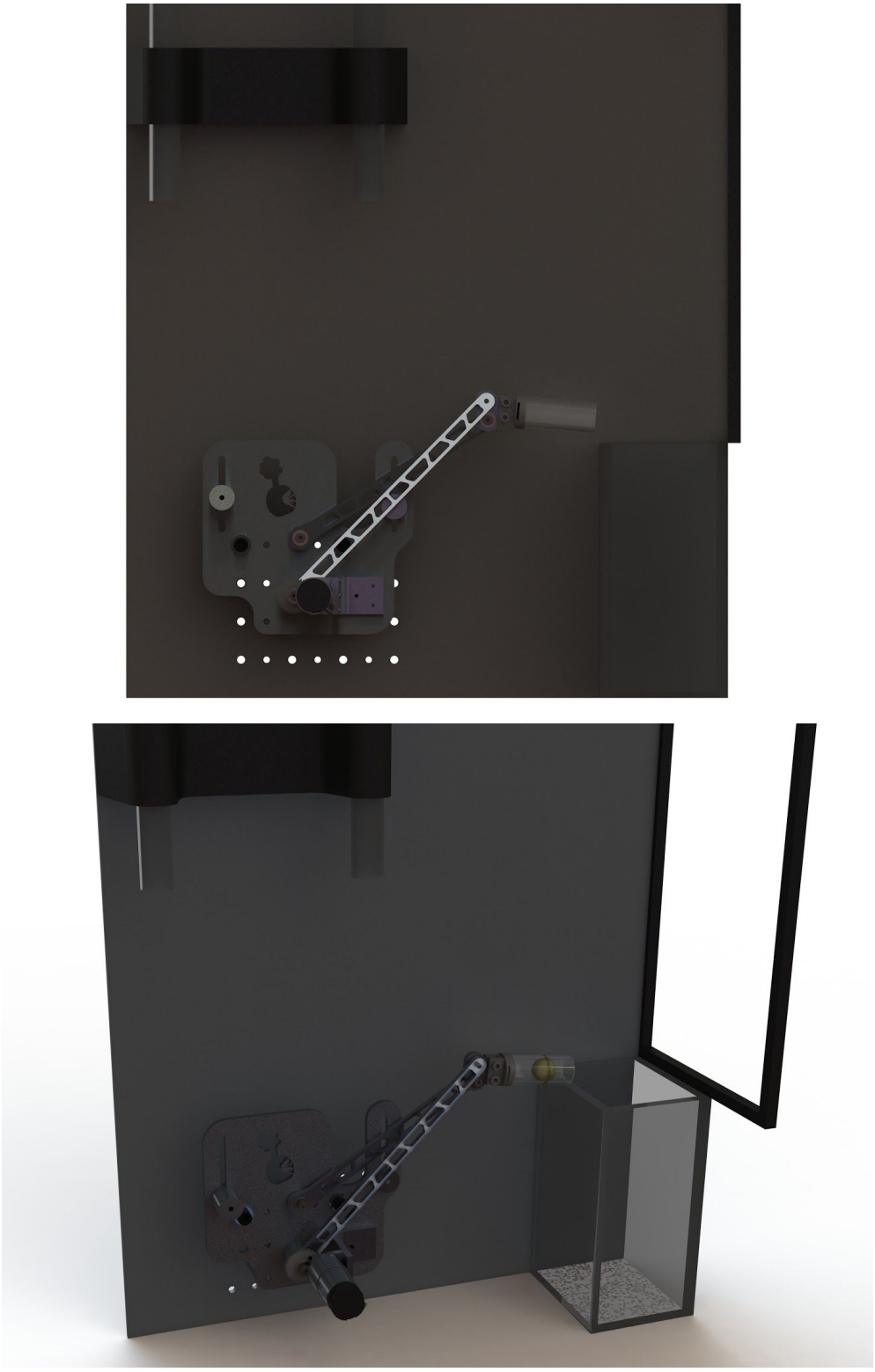


Figure 5.2: Mechanism and Its Full Range of Motion/Isometric View

The joint design used in this mechanism is consistent with the proper joint design that is shown in Figure 5.3, except for replacement of the bushings with bearings. This substitution was made to further reduce friction within the joints. The outer diameter of the bearing is approximately twice as large the outer diameter of the bushing, so the link design and joint design had to be adjusted to accommodate the larger size. The cross section of the coupler joint is shown below in Figure 5.4, as well as the cross section of the joints on the mounting board in Figure 5.5.

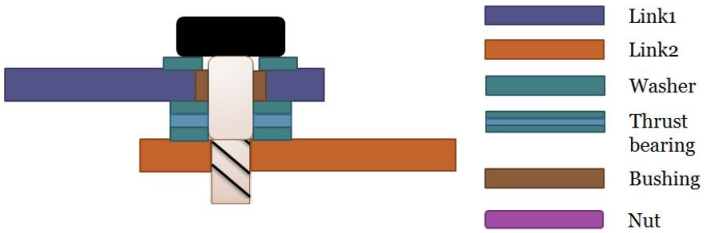


Figure 5.3: Cross Section Model of Joint

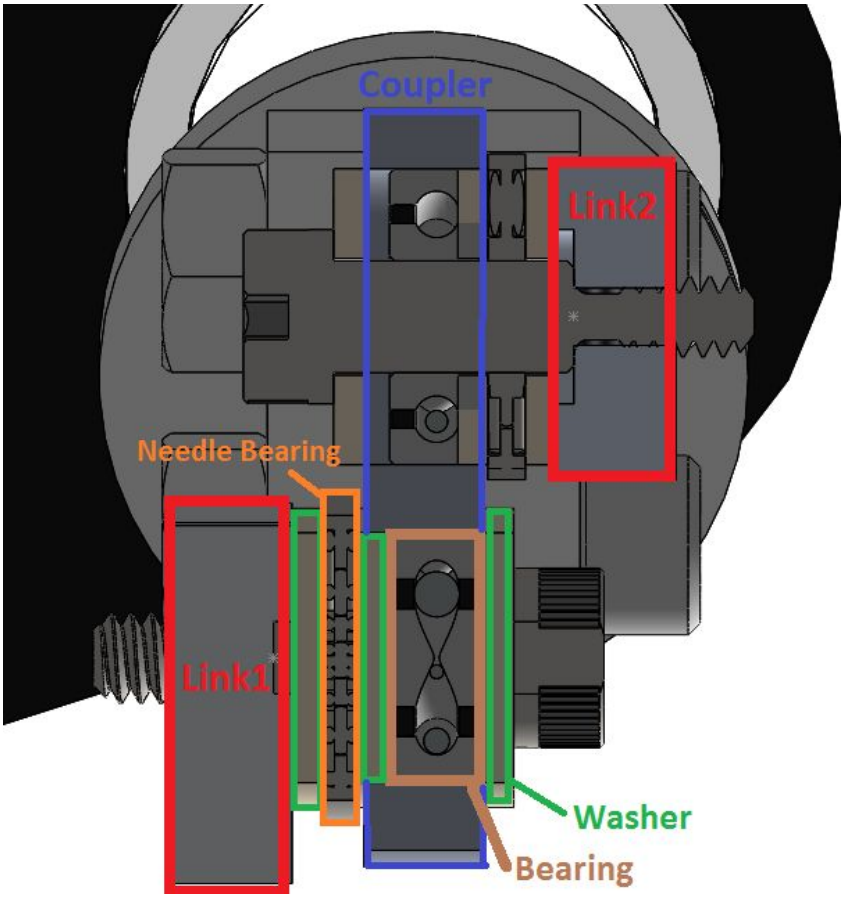


Figure 5.4: Cross Sectional Views of the Joints Connecting the Coupler to the Input and Follower

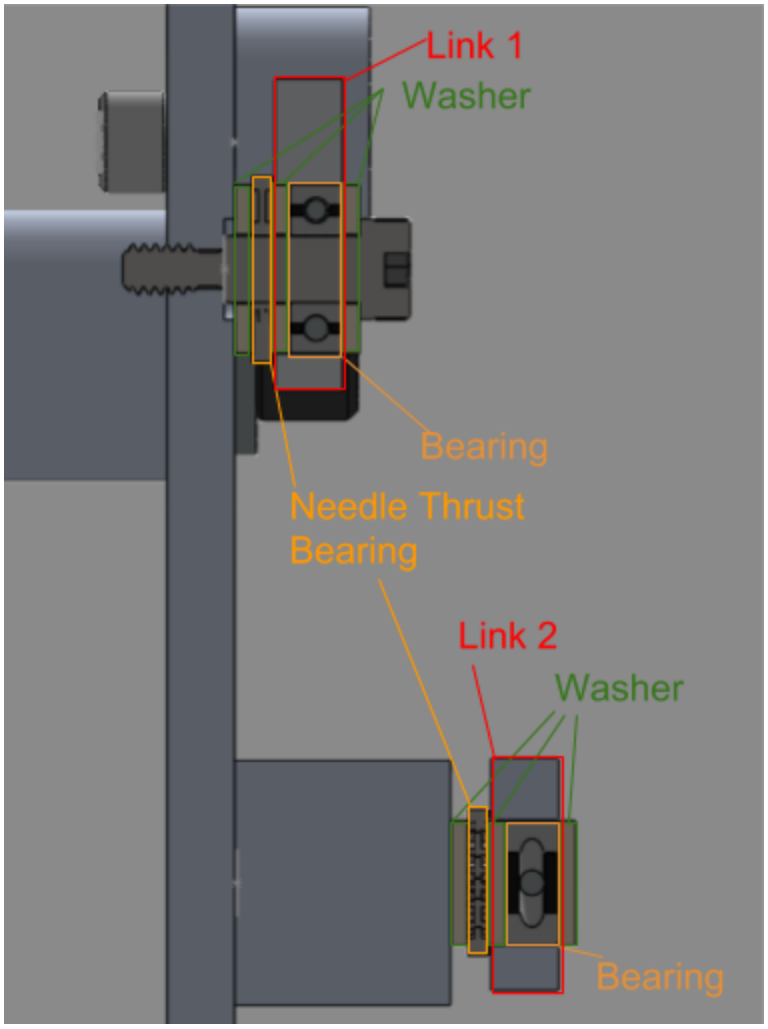


Figure 5.5: Cross Sectional Views of the Joints Connecting the Mounting Board to the Input and Follower

Below is a figure of the total volume the mechanism encompasses at its initial position. This is measured from the start of the backboard to the base plate. The height and length are from the top corner of the cup to the bottom right corner of the baseplate. The CAD volume approximation was 27.73 cubic Inches.

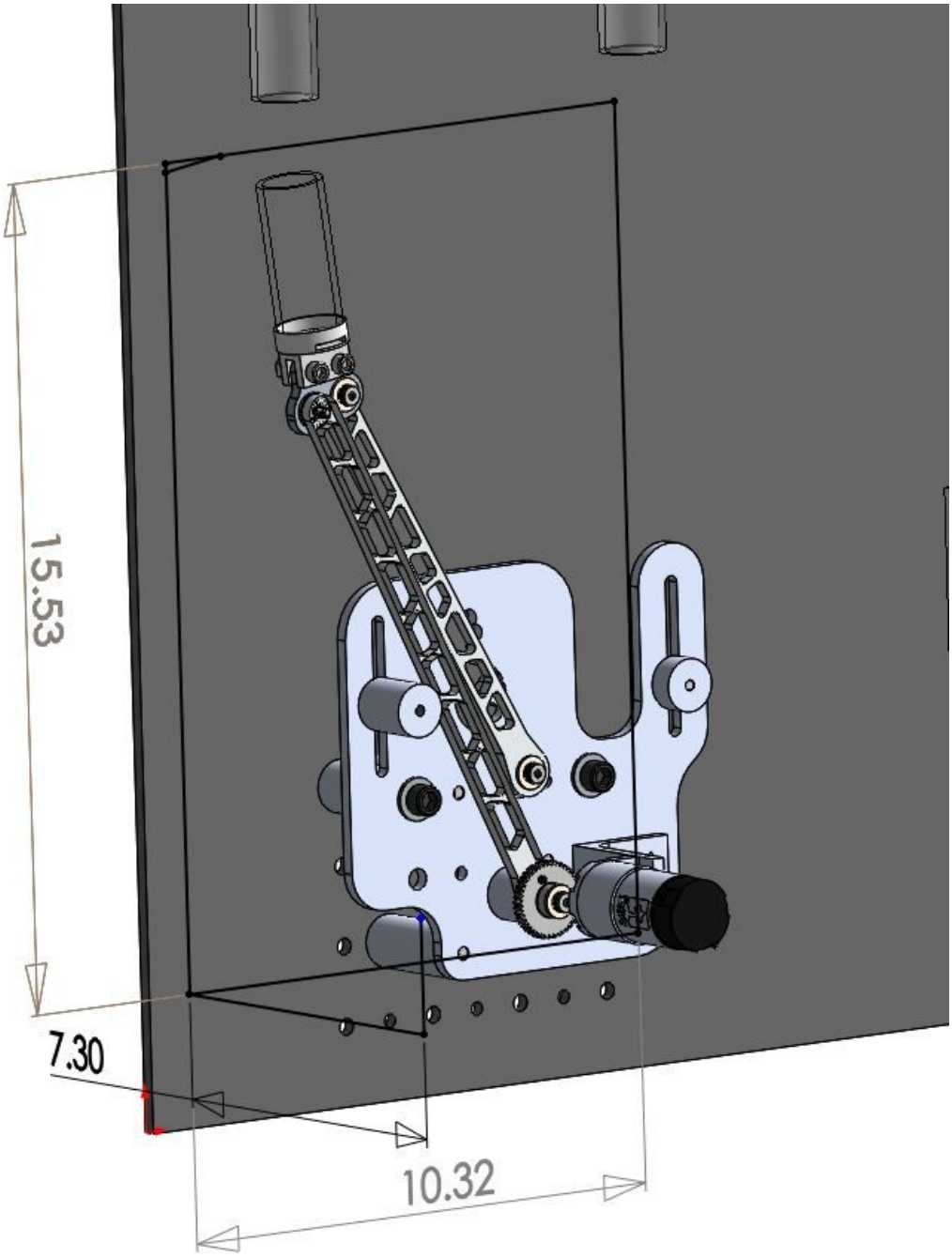


Figure 5.6: Mechanism with Volume Measurements Displayed

Our baseplate system will be mounted to the playing field area via Round Aluminum Spacers. These Spacers will be fixed with the playing field and baseplate via bolts and press fit dowel pins.

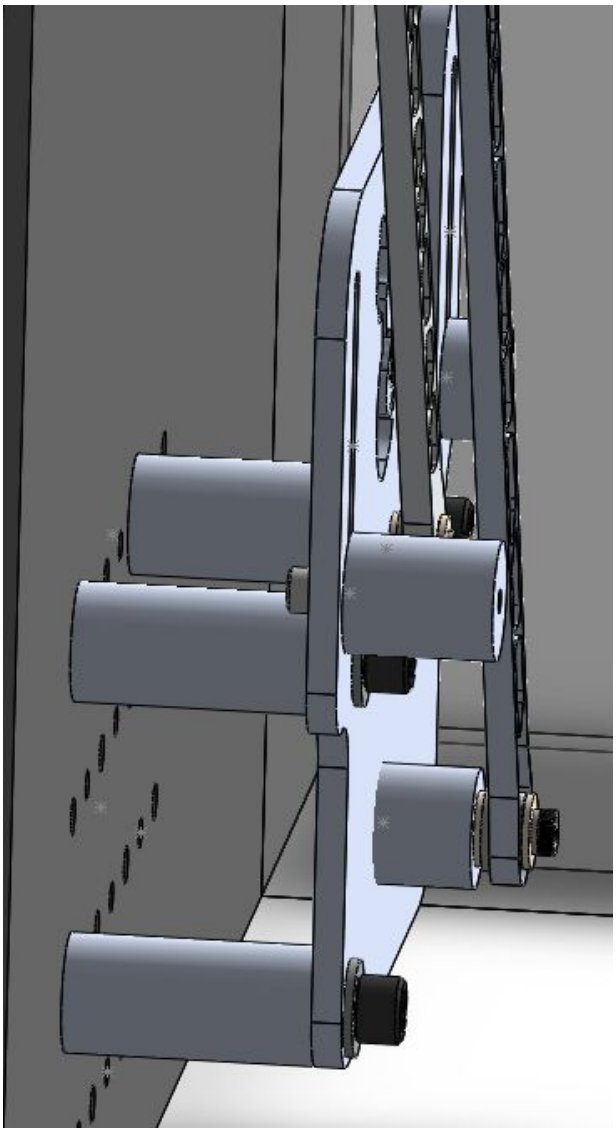


Figure 5.7: Board mounting system

The figure below shows the the adjustable hard stops on the mounting board. The hard stops are secured to back side of the mounting board by 3/8ths cap screws. The hard stops can be adjusted vertically to stop the linkage system at different angles.

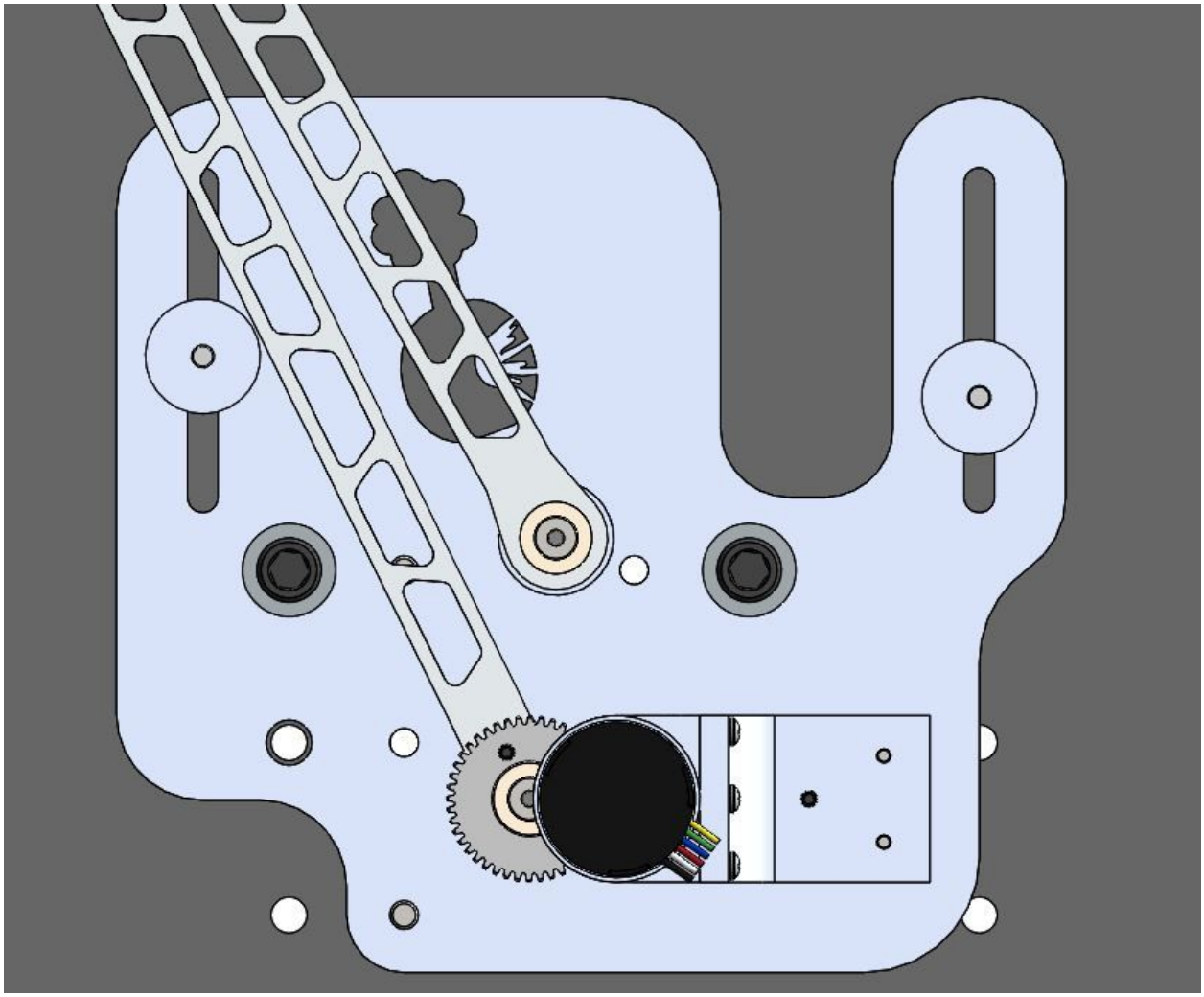


Figure 5.8: Adjustable hard stop system on the board

Our CLC cup mounting will be achieved by glueing the cup to the 3D Printed part. The 3D part is made to fit the cup, as seen in the figure below. The inner circle of the 3D part will slide inside the cup and has a curved shape to better fit the ball. At the bottom of the inner circle there is a hole to allow the light sensor to read the balls color. The inner curved part was designed to allow the ball to sit close to the light sensor to ensure accuracy.

The 3D printed part also has a slot built in on one side to allow access to the color sensor. The color sensor will be fixed into the 3D part by drilling a hole or glueing the sensor.

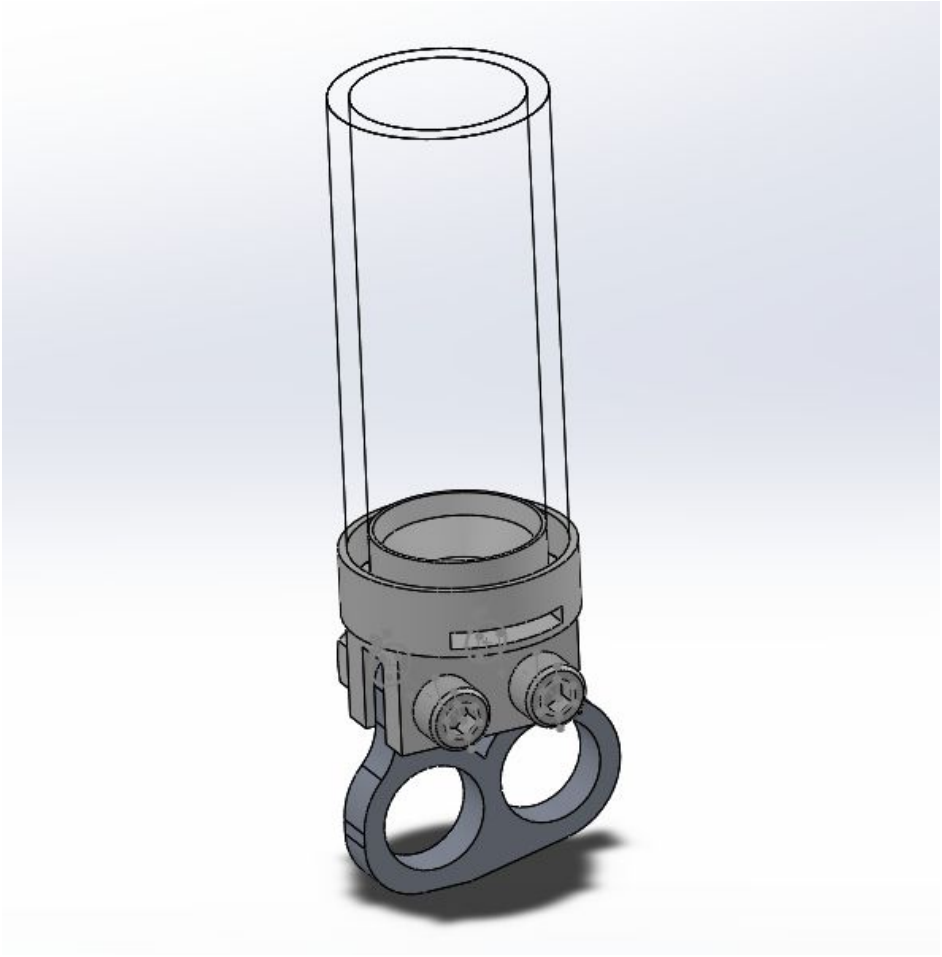


Figure 5.10: 3D Printed Coupler

Section 6: Preliminary Team ADAMS

Primary testing of the mechanism's final design was done using the ADAMS modeling program. The figure below shows the four-bar linkage in the ADAMS program without the many complex pieces of the joints included. Angular displacement, angular velocity, torque required, and power required to run the mechanism were calculated using the simulation and acceleration features in the program. The graphs below show the results of these simulations. Angular acceleration was calculated according to the Equation 6.1 for angular acceleration:

$$\alpha = \frac{4\theta}{t^2} \qquad \textbf{Eq. 6.1}$$

Our theta was determined by measuring the angle change of the input link during a full range of motion. Using CAD and ADAMS, this value was determined to be 71.2°. ADAMS simulation was utilized to determine a proper t value. This was done by tweaking the time until finding a satisfactory power consumption, which resulted in a minimum time of 0.45 seconds. The final calculated angular acceleration was thus 1406.41 deg/s^2

The Figure below shows the model that was made in ADAMS and then used in simulation. The mechanism was given an acceleration that was used to model the motor, and preliminary data was collected on the mechanism's function. The links and coupler were set as aluminum, and the cup and 3D printed part were set as plastic.

Figure 6.1: Mechanism in Adams

Figure 6.2 below shows the angular displacement of the input link as it travels from the left-most position under the left tube to the right-most position next to the bucket. The input link travels 71.2 degrees from right to left.

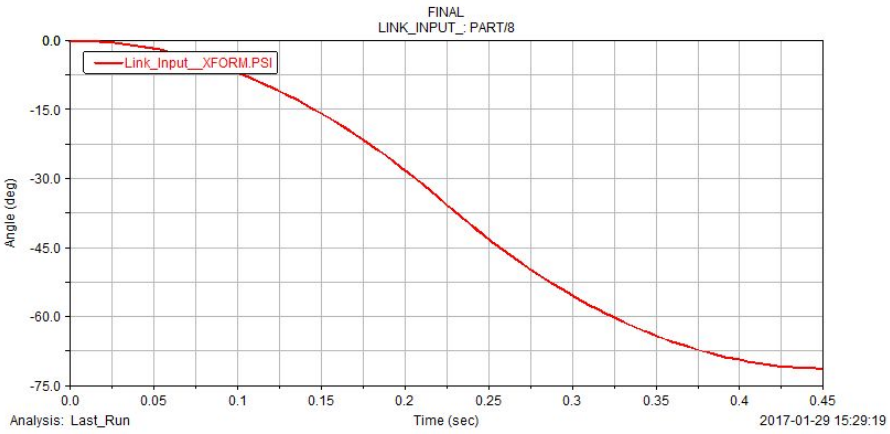


Figure 6.2: Angular Position of Input Link

The graph of our angular velocity (Figure 6.3**)** takes the shape of a 'V' with a rounded point at the bottom. Considering acceleration is the derivative of the velocity function, and the slope from 0 to 0.225 seconds is constant, this implies we have a constant acceleration with a linearly changing velocity. At 0.225 seconds the acceleration is reversed and the slope is now in the positive direction. The point where the accelerations change sign is the point of maximum angular velocity in the negative direction where we have about -320 degrees per second. The linkage then decelerates to zero angular velocity as it approaches the hardstop.

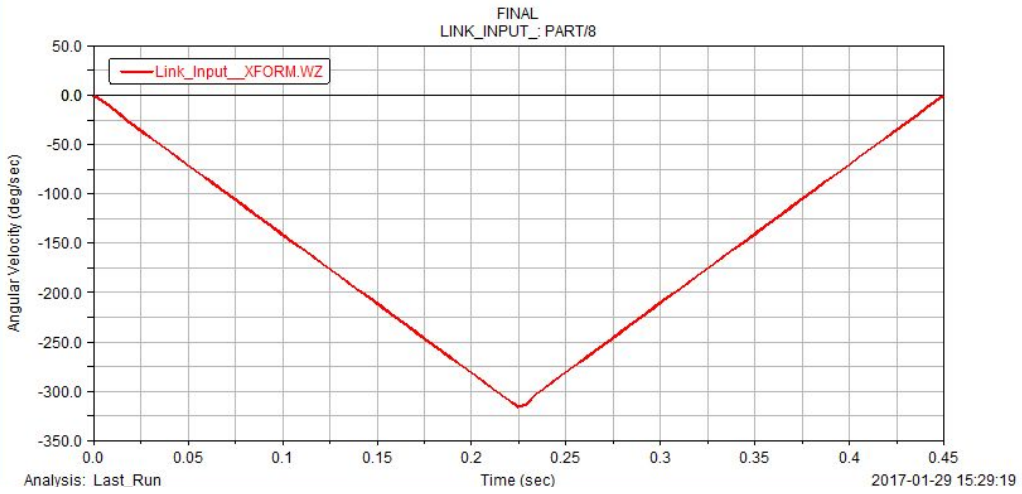


Figure 6.3: Angular Velocity of Input Link

The graph below shows the results of the ADAMS simulations for the torque on the input link due to the acceleration that is applied. The graph shows a group of spikes near the middle of the simulation, and also a small spike in the beginning of the simulation. These spikes are due to the application of the acceleration at intervals in time switching directions, and countering the inertia of the input link as it changes directions of acceleration.

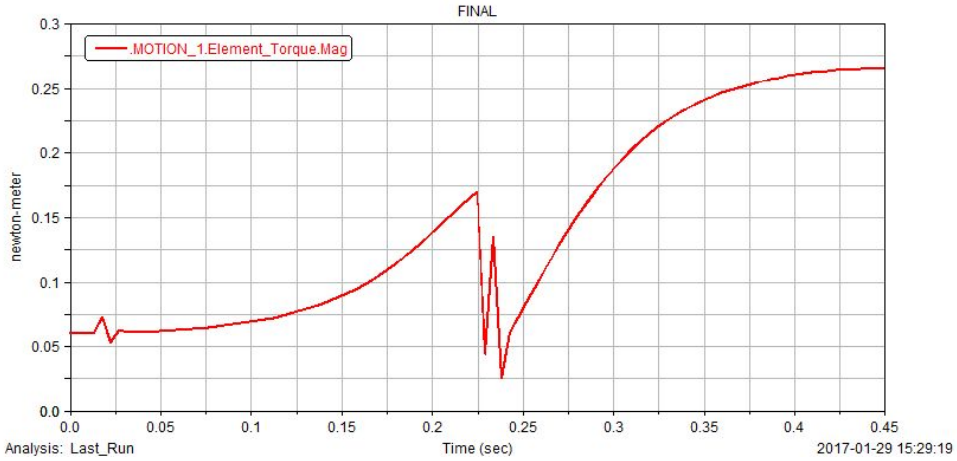


Figure 6.4: Torque on Input Link

Using equation 1 above, angular acceleration was calculated. This acceleration was applied to the input link in the ADAMS program to simulate the motor in the system. Using trial and error in ADAMS, it was determined that the mechanism could complete one trip from left to right in just under 0.5 seconds without exceeding a limit of 1 newton-meter/second, or 1 watt, power consumption. However, the graph generated from the ADAMS does not take into account the static friction in the system. This friction would cause the graph to have a spike near the beginning of its motion.

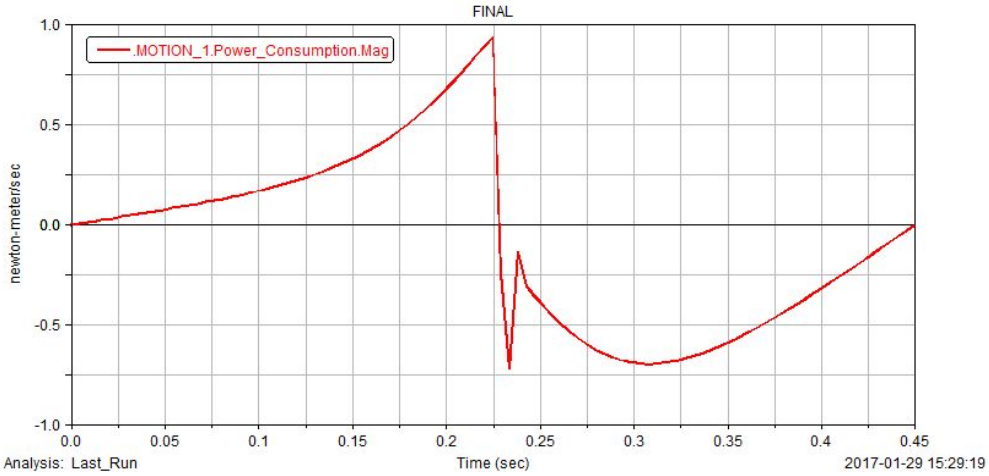


Figure 6.5: Power Consumption from Left to Right-most Position

Section 7: Motion Generator Revision

No changes were made to the motion generation since Gate 1 Review.

Section 8: Evaluation of Received Designs for Manufacturing

Coupler:

All of the holes on the coupler are to a three decimal place precision even though the reamers for the holes are not provided in the shop and the manufacturing plans says to use drill bits to achieve the hole diameters. I believe that the holes were meant to be two decimal place precision and drilled instead of reamed. Some of the holes appear both in the manufacturing plan and the DXF waterjet file so it is somewhat unclear what holes we are responsible for, because we do not have an ORD file. If they do plan on water jetting the holes, we would advise against it, because the waterjet does not create holes very precisely. In addition the 1.89" dimensions for the relative distance between 2 holes is located within the part and should be to a 3 decimal place precision. The 0.089" dimension is listed twice instead of using the "X2" notation, which would have been a little more efficient, and the manufacturing plan unnecessarily prompts us to clamp the part in the vise in steps 3-5. Even though it is obvious where the datum is in the drawing, we were not asked to use an edge finder to locate a datum anywhere in the manufacturing plan. All serious issues have been emailed to the other team.

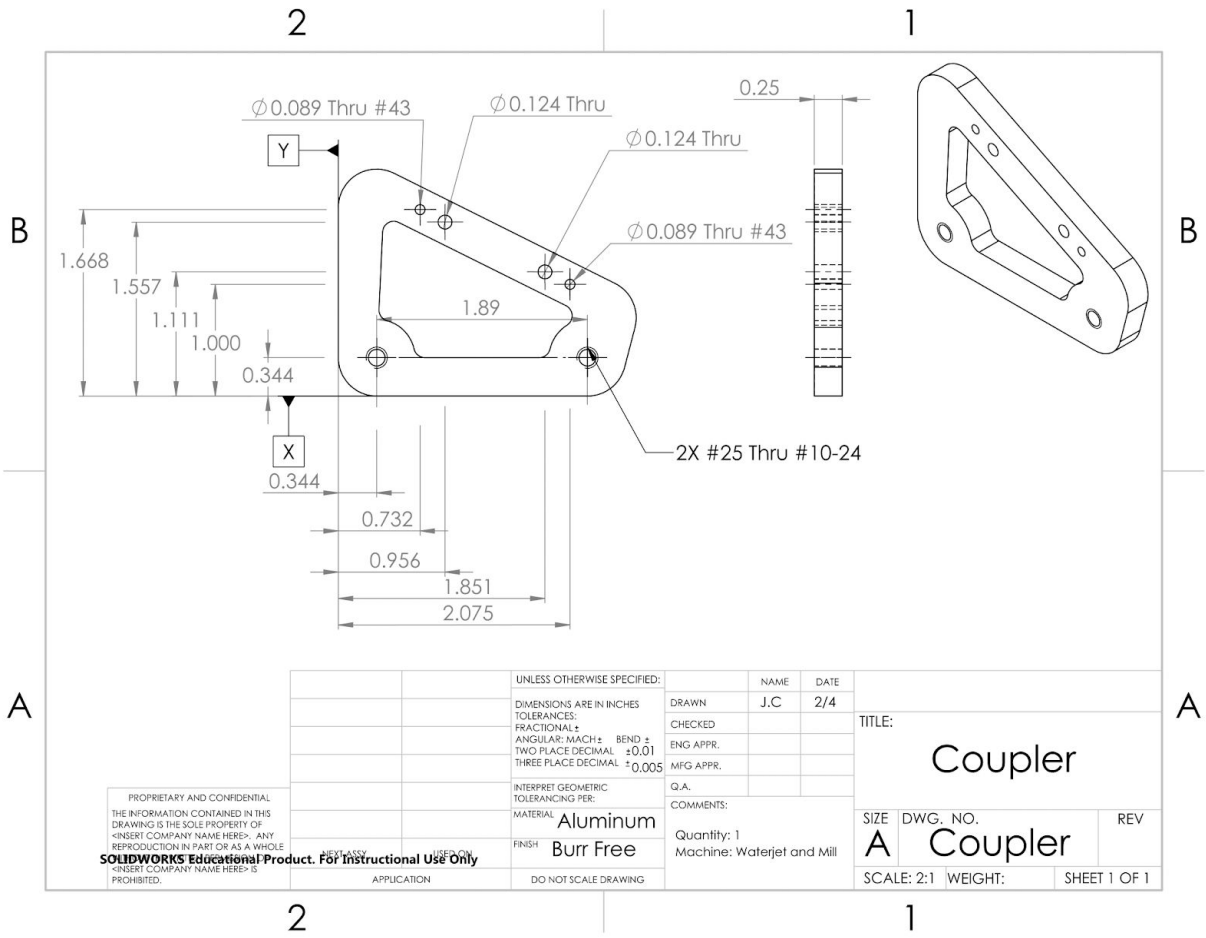


Figure 8.1: Coupler Drawing

Part Number: ME350-003 Revision Date: 1/28/2017
Part Name: Coupler
Team Name: Team 51

Raw Material Stock: Aluminum Plate, ¼" x 12" x 18"

Step #	Process Description	Machine	Fixtures	Tool(s)	Speed (RPM)
1	Waterjet rough part out of quarter-inch aluminum plate. With holes with of varying sizes.	Waterjet		Waterjet	
2	Break all edges with file and file all burrs.			File	
3	Clamp part in vise. Ream the smallest holes to a diameter of .0890". (2 holes)	Mill	vise	#43 Drill bit, Drill Chuck	1800
4	Clamp part in vise. Ream the medium sized holes to a diameter of .1160". (2 holes)	Mill	vise	#32 Drill bit, Drill Chuck	1600
5	Clamp part in vise. Ream the largest holes to a diameter of .1495". (2 holes)	Mill	vise	#25 Drill bit, Drill Chuck	1500
6	Remove part from vise and file and deburr all holes			File	
7	Insert part into a tap vise and tap both .1495" holes with the 10 - 24 tap.		vise	10 - 24 tap drill	
8	Remove part from vise and deburr the tapped holes.			Deburring Tool	

Figure 8.2: Coupler Manufacturing Plan

Plate:

The format of this drawing is not in the standard ANSI format. The manufacturing plan includes steps to drill and ream holes to 0.246” when there are no holes that match that description in the drawing. The plan also says to drill base plate attachment holes to 0.375”, while the drawing says to drill them to a W bit. The drawing also does not include the decimal hole size for the W thru holes. There were no major issues that needed to be included in the email to the design team.

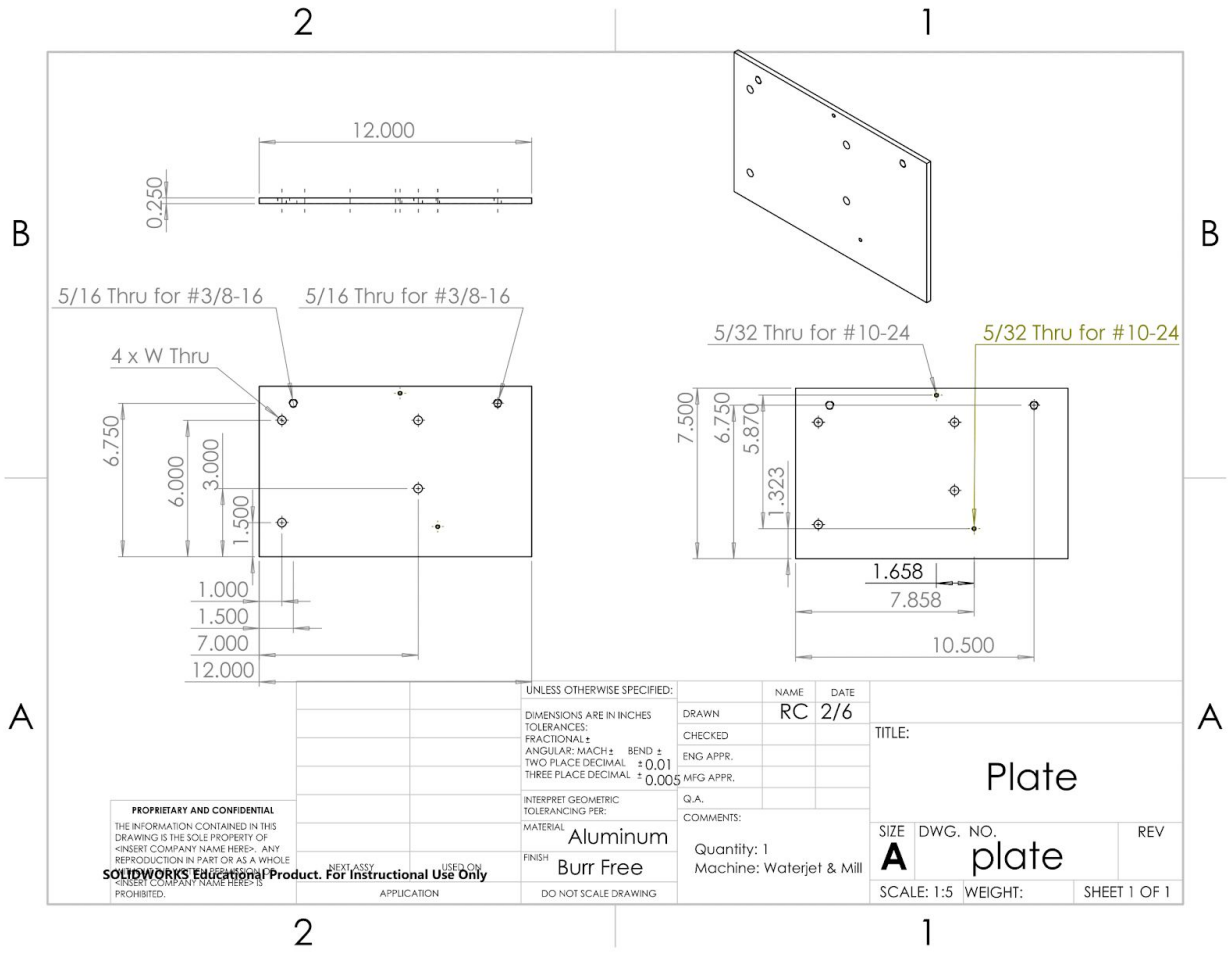


Figure 8.3: Plate Drawing

Part Number: ME350-004 Revision Date: 1/31/2017
Part Name: Plate
Team Name: Team 51

Raw Material Stock: Aluminum Plate, ¼" x 12" x 18"

Step #	Process Description	Machine	Fixtures	Tool(s)	Speed (RPM)
1	Waterjet rough part out of quarter-inch aluminum plate. With holes with holes of varying sizes.	Waterjet		Waterjet	
2	Break all edges with file and file all burrs.			File	
3	Clamp part in vise. Ream the dowel pin holes to a diameter of .246" (4 holes).	Mill	vise	D Drill bit, Drill Chuck	1200
4	Clamp part in vise. Ream the input and follower link holes to a diameter of .323". (2 holes)	Mill	vise	P Drill bit, Drill Chuck	1000
5	Clamp part in vise. Ream the base plate attachment holes to a diameter of .375". (5 holes)	Mill	vise	⅜" Drill bit, Drill Chuck	800
6	Drill the singular hard stop hole in the top right corner on the engineering drawing to a diameter of .3125". (1 hole)	Mill	vise	.3125" Drill Bit, Drill Chuck	1000
7	Remove part from vise and file and deburr all holes.			File	
8	Insert part into a tap vise and tap the .3125" hole with the ⅜" - 16 tap.		vise	⅜" -16 tap drill	
9	Remove part from vise and deburr tapped hole.			Deburring tool	

Figure 8.4: Plate Manufacturing Plan

Hardstop:

The outer diameter of the hardstop was dimensioned in the wrong view; it should have been dimensioned in the top view. Manufacturing the off-center hole through the part will be a challenge, as there is no good spot to use as a datum. There was not a manufacturing plan for this part, but the design team should have included how to zero the mill at the center of the outer circumference of the part using a dial indicator or with another method. Moreover, the plan should have included both the size of the drill bit to use for the off-center hole, and the whether the hole should be a through hole or a blind hole. For this part, the hidden lines in the top drawing suggest that the hole is a through hole that will have a ⅜" bolt that secures the part to the mounting plate. There were no major issues that needed to be included in the email to the design team.

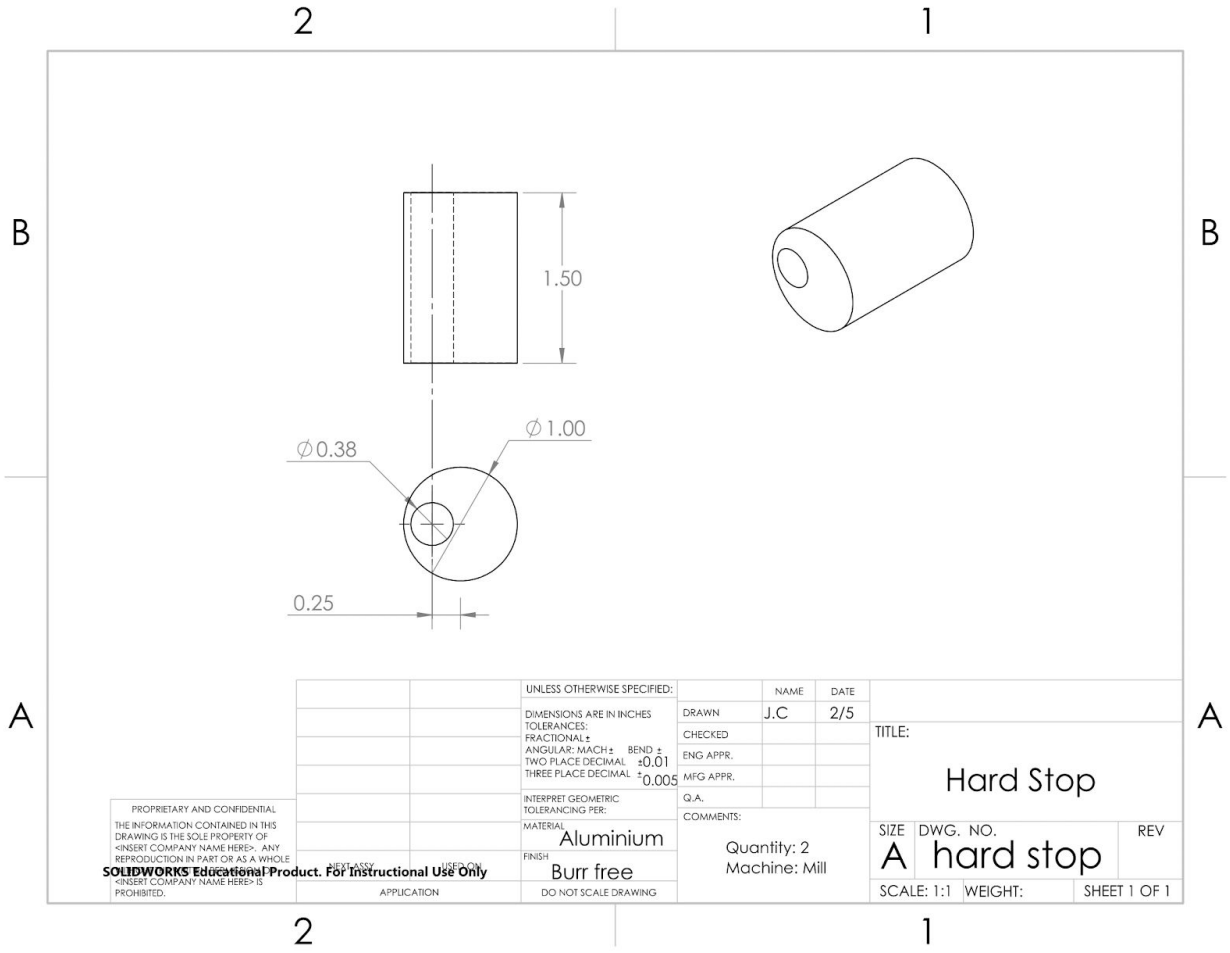


Figure 8.5: Hard Stop Drawing

Link 1

The link is shaved down on the sides with a three decimal place precision in the plunging diameter of 0.344”. The end mill that would be required to do this is not available in the shop and this would not be able to be done to a three decimal place precision. The depth of this shaving has to be fairly accurate, but there is no need for three decimal plate precision on the shape of the depth. There is also not a 0.344” reamer to make the holes on either end of the lever. In addition the manufacturing plan did not mention the measurement of bearings for these press fit holes or the pre drills necessary for these reams. The datum specified in the drawing is impossible to achieve, within reason, because I would have to use a dial indicator to find where that arcs on the end of the links ends. This impossible datum won’t be an issue for functionality, however, because we can get close to the datum and get the relative hole locations done precisely, but the holes won’t be exact in X relative to the waterjet profile. All serious issues have been emailed to the other team.

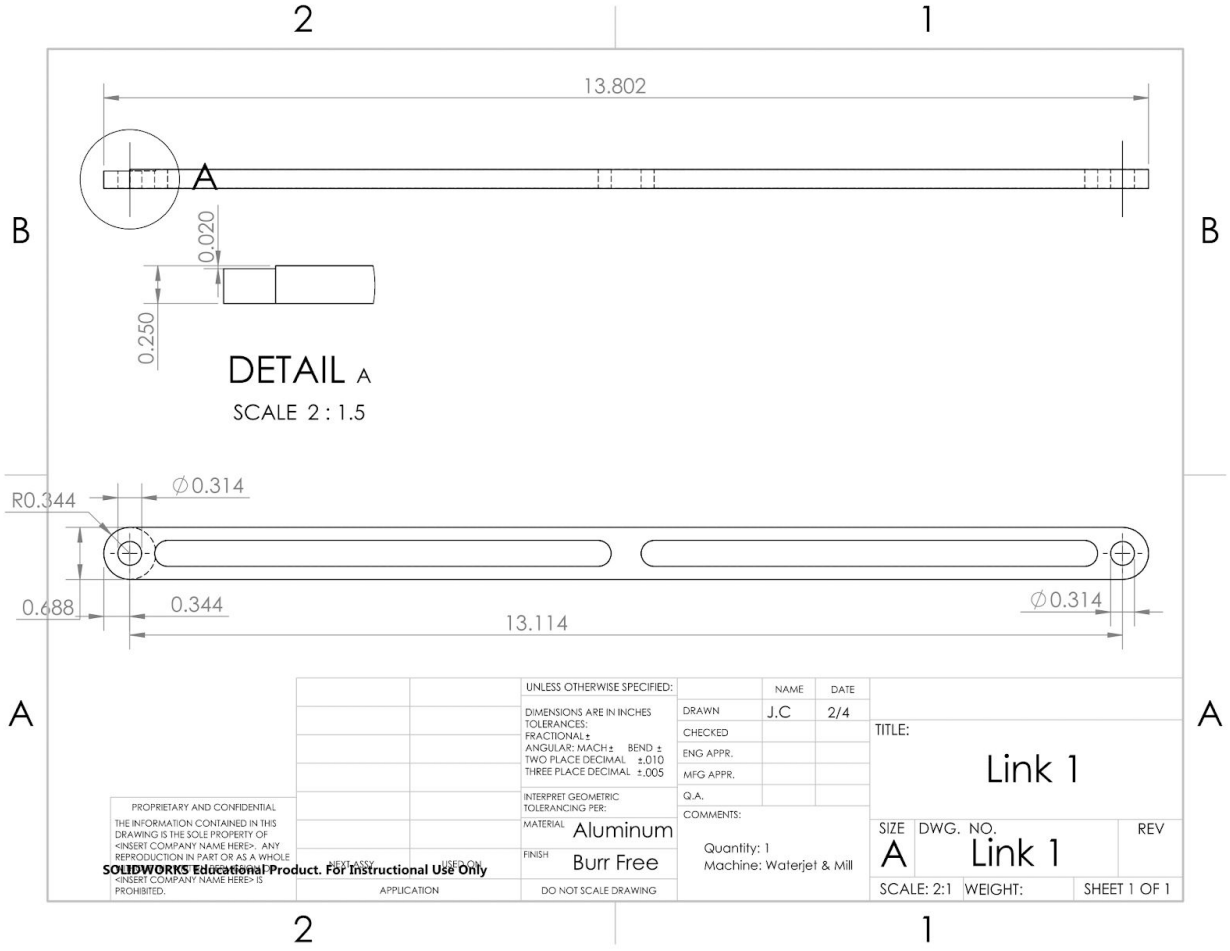


Figure 8.6: Link 1 Drawing

Part Number: ME350-001 Revision Date: 1/28/2017
Part Name: Link 1
Team Name: Team 51

Raw Material Stock: Aluminum Plate, ¼" x 12" x 18"

Step #	Process Description	Machine	Fixtures	Tool(s)	Speed (RPM)
1	Waterjet rough part out of quarter-inch aluminum plate. With holes with a diameter of .31".	Waterjet		Waterjet	
2	Break all edges with file and file all burrs.			File	
3	Clamp part in vise. Ream holes to a diameter of .3125".	Mill	vise	.3125" Diameter Drill bit, Drill Chuck	1000
4	Remove part from vise and file and deburr all holes.			File	

Figure 8.7: Link 1 Manufacturing Plans

Link 2

The link is shaved down on the sides with a three decimal place precision in the plunging diameter of 0.344”. The end mill that would be required to do this is not available in the shop and this would not be able to be done to a three decimal place precision. The depth of this shaving has to be fairly accurate, but there is no need for three decimal plate precision on the shape of the depth. There is also not a 0.344” reamer to make the holes on either end of the lever. In addition the manufacturing plan did not mention the measurement of bearings for these press fit holes or the pre drills necessary for these reams. The datum specified in the drawing is impossible to achieve, within reason, because I would have to use a dial indicator to find where that arcs on the end of the links ends. This impossible datum won’t be an issue for functionality, however, because we can get close to the datum and get the relative hole locations done precisely, but the holes won’t be exact in X relative to the waterjet profile. All serious issues have been emailed to the other team.

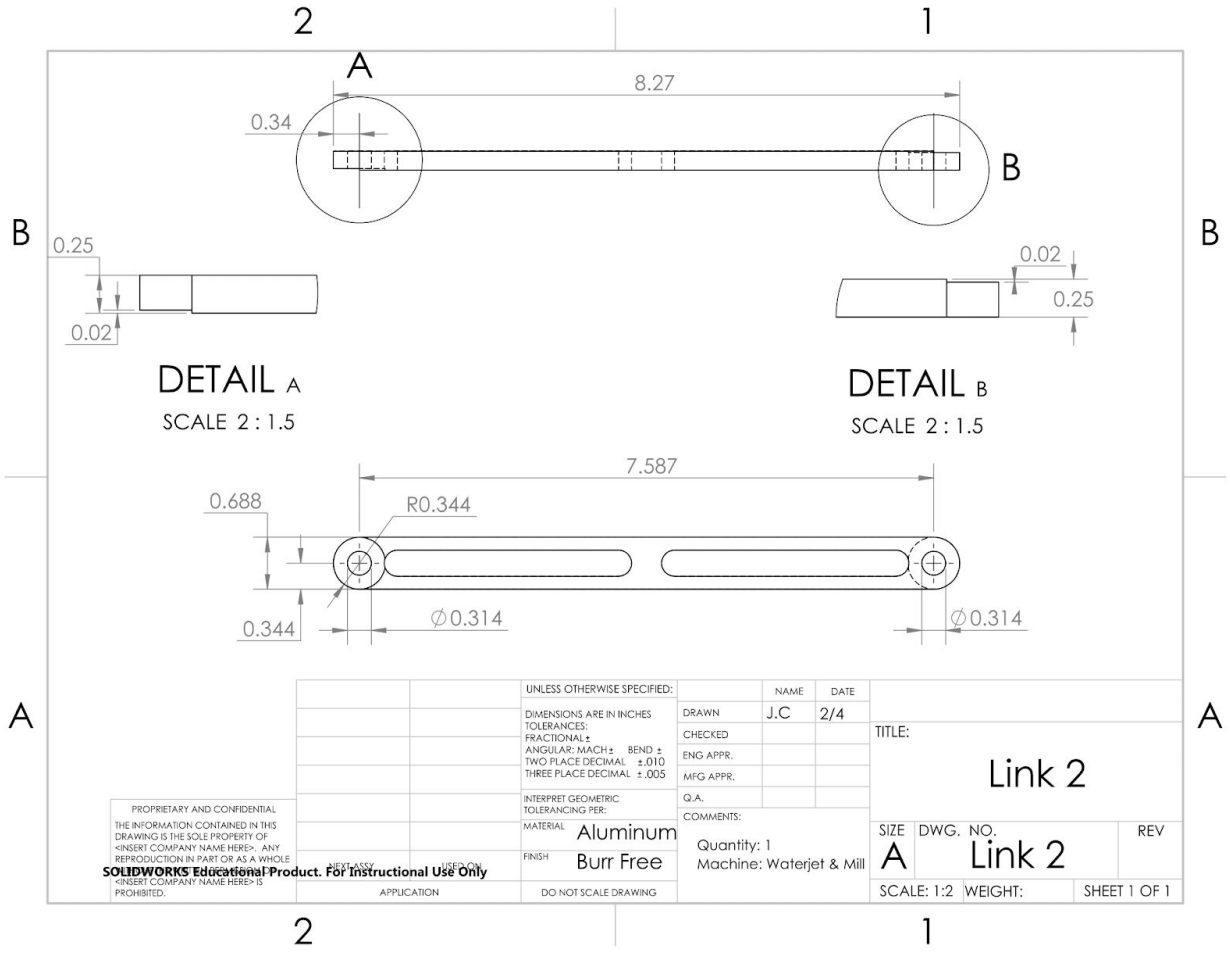


Figure 8.8: Link 2 Drawing

Part Number: ME350-002 Revision Date: 1/28/2017
Part Name: Link 2
Team Name: Team 51

Raw Material Stock: Aluminum Plate, ¼" x 12" x 18"

Step #	Process Description	Machine	Fixtures	Tool(s)	Speed (RPM)
1	Waterjet rough part out of quarter-inch aluminum plate. With holes with a diameter of .31".	Waterjet		Waterjet	
2	Break all edges with file and file all burrs.			File	
3	Clamp part in vise. Ream holes to a diameter of .3125".	Mill	vise	3125" Diameter Drill bit, Drill Chuck	1000
4	Remove part from vise and file and deburr all holes.			File	

Figure 8.9: Link 2 Manufacturing plan

Standoff

The standoff is listed as being made on a mill, but it would be far easier to make on a lathe. It is just a hole in a circular rod so it would be far faster to lathe this part. No manufacturing plan is provided, we do not know how we are intended to machine it. The 1.000" dimensions for the cylinder is also displayed in the wrong view. It should be on the side view of the part, since it is a cylinder and not a hole.

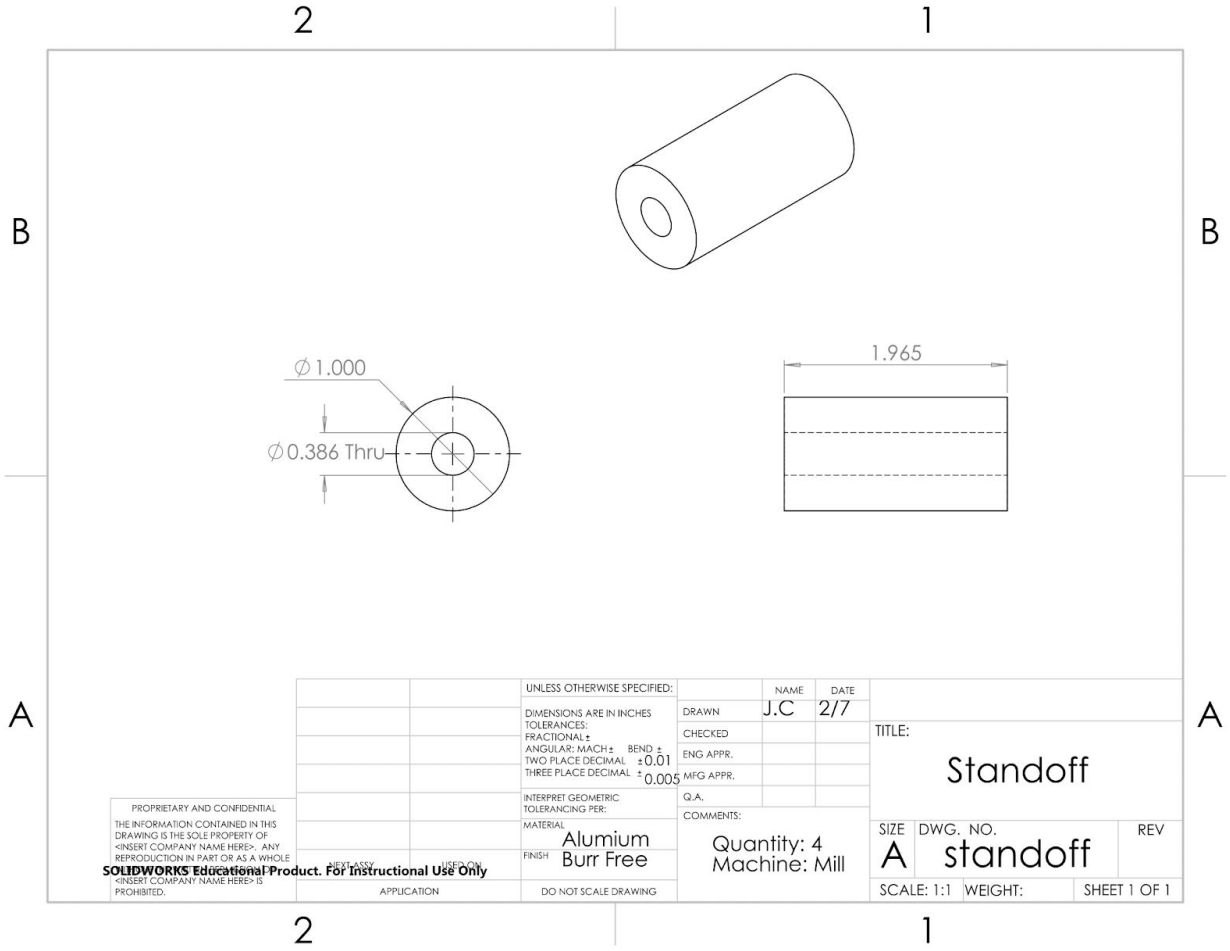


Figure 8.10: Standoff Drawing

Section 9: Evaluation of Received Manufactured Parts

A majority of the parts received met the specifications laid out in the drawings and were manufactured on time. All of the lathe parts were properly machined and deburred. The milled parts had correct dimensions but were not deburred.

Section 10: Energy Conversion Introduction

Transmissions are used to convert power from one form into another. A transmission is needed in order to convert different types of motion, such as rotary to linear motion. It can also convert low-torque high speed motion into high-torque low-speed motion. Transmissions also result in a slight loss of efficiency, so that must be considered for calculations and design. Some examples of transmissions are gears, belts, and chains. For this project, we will manipulate the torque/speed curve of the motor to meet the exact performance requirements of our mechanism. The design of a transmission is limited to a certain input voltage, rotational speed, and torque output in order to avoid failure or damage. If the input voltage is too large, the load on the motor may cause the motor to run too fast, or produce too much torque, which will cause the motor to fail. If the motor turns too fast, the motor will experience mechanical failure, but if the motor produces an excessive amount of torque, the current inside grow too large, and the motor will experience electrical failure. There is a limitation on the transmission ratio we can achieve due to the size constraints of the project. A larger transmission ratio typically requires a large amount of space. Exceptions include transmission systems like a harmonic drive, but this sort of system would exceed the budget of this project.

Section 11: Transmission Ratio and Type Determination

We had to determine the transmission ratio required to move between its starting and ending positions. These endpoints are depicted in Figure 11.1 below.

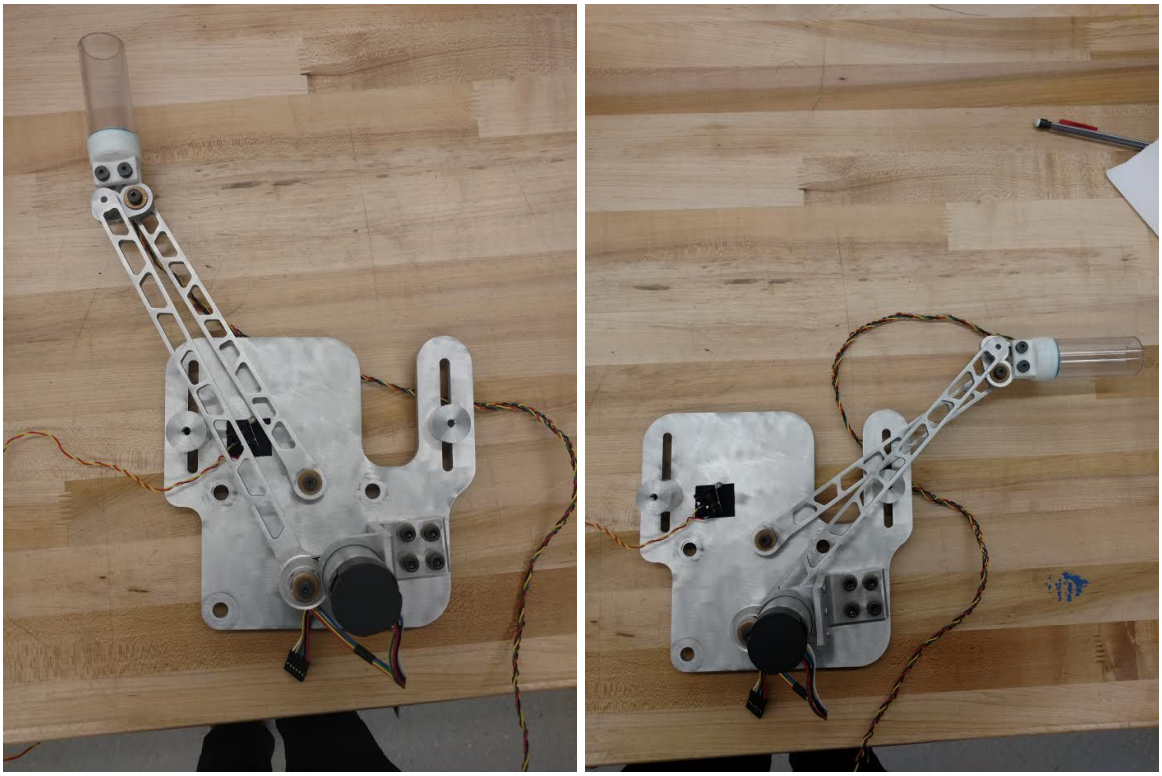


Figure 11.1: Starting and ending position of the mechanism

The first method we used to determine transmission ratio was inertia matching. As the linkage moves, its mass distribution changes and therefore inertia does as well. The torque required to move the mechanism also changes during this motion. We found the mechanisms total inertia and the motor inertia to find a transmission ratio.

In order to find the start up torque of the motor, we have to calculate the moment of inertia of the motor using Equation 11.1 below where I is the total moment of inertia as seen at the motor side, I_m is the moment of inertia of the motor, I_L is the inertial of the load, and N is the transmission ratio.

$$I = I_m + {}^{I_L}/_{N^2} \qquad \textbf{Eq. 11.1}$$

Next, we found the angular acceleration of the motor α_m using equation 11.2 where α_L is the angular acceleration of the load and N is the transmission ratio.

$$\alpha_m = N\,\alpha_L \qquad \textbf{Eq. 11.2}$$

The start up torque of the motor T is then given by equation 11.3 where I is the total moment of inertia seen by the motor and α_m is the angular acceleration of the motor.

$$T = I\,\alpha_m$$ **Eq. 11.3**

The purpose of inertia matching is to minimize the motor torque through the transmission ratio. The optimized transmission ratio is given by setting the derivative of the start up torque to zero. This simplifies down to equation 11.4.

$$N = \sqrt{\frac{I_L}{I_m}}$$ **Eq. 11.4**

In order to find the moment of inertia of the entire mechanism, we found the mass and moment of inertia of the input link, coupler, and follower link. The parallel axis theorem states that if a body is made to rotate about a new axis parallel to its old one displaced by a distance r, its moment of inertia I with respect to the new axis is related to I_{cm}, the moment of inertia about the center of mass, by equation 11.5.

$$I = I_{cm} + mr^2$$ **Eq. 11.5**

For the coupler link, we need to find its instant center between the ground and coupler link. To do this, we find measurements R1-R5 that measure the lengths of each link and the length of the line between the midpoint of the coupler and the intersection point between the lines of the two links. R1 is the length of the input, R4 is the length of the follower, R5 is the length of the coupler, R2 is the distance between the intersection of this input in coupler to the intersection of R2 and R4, and R3 is the distance between the intersection of the follower with the coupler and the intersection of R2 and R4. These lengths are labeled in figure 11.2 below.

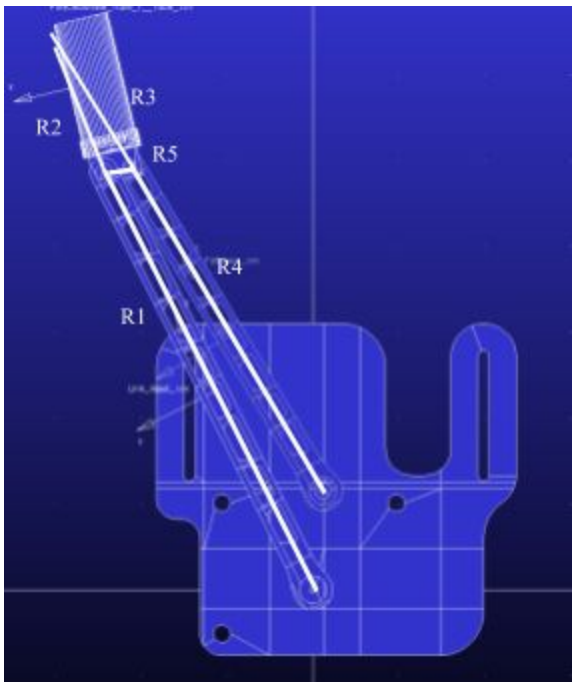


Figure 11.2: Measurements R1-R5 displayed the linkage in the starting position

By using these values R1-R5, and the moment of inertia of the linkages I_{input}, $I_{coupler}$, and I_{output},, we can find the total moment of inertia I_{total} using Equation 11.6 below.

$$I_{total} = I_{input} + I_{coupler}(\frac{R_1}{R_2})^2 + I_{output}(\frac{R_1 R_3}{R_2 R_4})^2 \quad \textbf{Eq. 11.6}$$

The motor with gearbox used for this project has a moment of inertia of 25000 (g*cm^2). Using Equation 11.4 with this I_{motor} and the I_{total} found by Equation 11.6, we can get the ideal transmission ratio for our mechanism. We found the required transmission ratio for three positions along the motion of our mechanism: the linkage in its initial position, directly below the second marble drop tube, and at the end of its motion. Table 11.1 below summarizes this information.

Table 11.1: Inertia of the mechanism at three different positions and the required transmission ratio

Position	R1 (cm)	R2 (cm)	R3 (cm)	R4 (cm)	R5 (cm)	Total Inertia (grams*cm^2)	Transmission Ratio
Initial	10.72	23.35	21.46	8.68	12.7	8735.79	0.59
Middle	10.72	6.61	6.39	8.68	6.49	19297.29	0.88
Final	10.72	2.4	2.84	8.68	2.6	57670.38	1.52

The major discrepancy between this model and the actual mechanism comes from that the mass we used for this calculation is different from the actual mass. It does not account for glue used to connect the coupler and color sensor, the color sensor itself, and some of the fasteners. This results in a calculated transmission ratio that is slightly smaller than what the mechanism actually needs to work properly.

The second method we used to find the optimal transmission ratio was finding the minimal torque our mechanism required to overcome gravity. Gravity will generate a torque on the linkage and we need to ensure that our motor is strong enough to overcome this torque. In order to generate a transmission ratio we created a simplified mechanism that only contained the coupler, cup, link, and mounting board. We imported this mechanism into ADAMS. The masses found in SOLIDWORKS were assigned to the components, joint relationships were established, and from this torque values were obtained. Pictures of the mechanism in ADAMS can be seen in Figure 11.

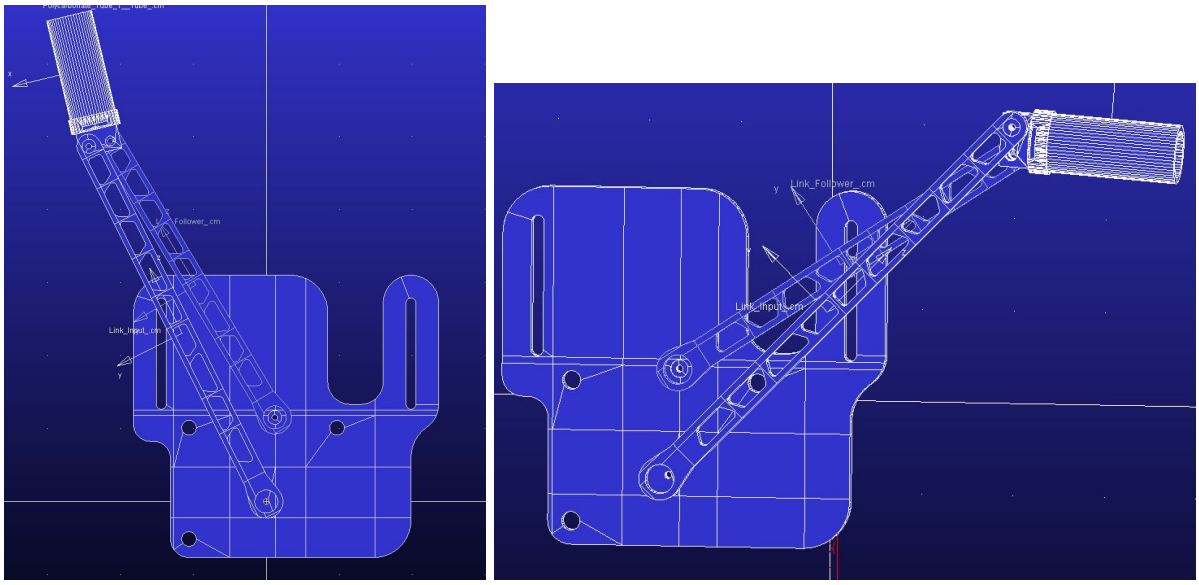


Figure 11.2: Initial and final positions of the mechanism in ADAMS

The max output torque of the gearbox motor at the shaft after a safety factor 5 is .13 N-m. By dividing the torque required by this output torque, we find the required transmission ratio to overcome gravity. Table 11.2 summarizes the torque required to lift the mechanism at its initial and final positions and the corresponding transmission ratio. This gravity compensation analysis is also susceptible to the same error as inertia matching; the mass of the mechanism in the analysis is less than the actual mechanism. The transmission ratios we find in this section will be smaller than the actual transmission ratio required.

Table 11.2: Max torque and transmission ratio for the beginning and end of the mechanisms range of motion

Position	Initial Angle (degrees)	Max Torque (N-m)	Transmission Ratio
Beginning	63.41	0.0696	0.535
End	134.65	0.2143	1.648

The final method we used to determine the transmission ratio was to examine the resolution of the encoder. We want a transmission that gives a positive resolution of the cup at 0.5 mm. We want a fine resolution because numerical derivatives are used to calculate the cups velocity. This calculation occurs 100 times a second resulting in a velocity resolution of 0.5mm/0.01s = 50 mm/s. The encoder on the motor has a base resolution of 64 counts per revolution so at a full revolution of 360 degrees, the gearbox yields 1920 counts. This is 360/1920 = .1875 degrees for every encoder count. This number divided by the transmission ratio N gives us the angular resolution of the input link as seen be seen by Equation 11.7 where r_1 is the length of the input link.

$$\Delta x = 0.1875 \deg \cdot \frac{\pi}{180\, deg} \cdot r_1 \cdot \frac{1}{N} < 0.5mm \qquad \textbf{Eq. 11.7}$$

By rearranging equation 11.7 to be in terms of transmission ratio N, we can compute the transmission ratio using Equation 11.8.

$$N_{encoder} > 0.1875 \deg \cdot \frac{\pi}{180\, deg} \cdot r_1 \cdot \frac{1}{0.5mm} \qquad \textbf{Eq. 11.8}$$

Since the input link has a length of 10.74 inches, we need a transmission ratio greater than 1.78. All of the transmission ratios for each method can be seen in Table 11.3 below.

Table 11.3: Max transmission ratio for each method of calculation

Method	Max Transmission Ratio
Inertia Matching	1.52
Gravity Analysis	1.648
Encoder Resolution	1.78

The max transmission required for each method were very similar in value. The greatest transmission ratio is 1.78, which is the ratio required in order to get the necessary encoder resolution. Therefore, the transmission ratio of our mechanism must be greater than 1.78. The transmission ratio we have chosen for our mechanism is 2.00.

Different options for transmission methods include gears, pulleys, timing belts, and chain and sprockets. The pugh chart that compares each method is in Table 11.4 below.

Table 11.4: Pugh chart comparing transmission types

Selection Criteria	Weight	Gears	Pulley	Timing Belt	Chain and Sprocket
Cost	2	0	3	2	-1
Reliability	5	0	-2	0	-1
Simplicity	2	0	3	2	1
Ease of Assembly	3	0	3	2	1
Aesthetics	1	0	-3	-2	1

Total	-	0	8	12	-1

Cost is given a weight of two because the transmission choice must fall within our $100 budget. It is not higher because a majority of the systems fall within this range. Gears are used as the base system and they cost about $30 each. Pulleys are the cheapest option and we can buy them for less than 10 dollars each which is why they have a value of three. The timing belt is slightly more expensive than the pulley but still far cheaper than the gears. A chain and sprocket system is more expensive than the gears, since we would have to buy a precision chain, which is why it has a negative one. Reliability is the most important factor because our transmission must reliable and accurately move to set positions in order to catch and throw marbles. This is why it has a weight of five. The pulley is much less reliable than the gear as it relies on friction to rotate and can slip. The timing belt has the same value as gears because is has teeth like the gear and will consistently return to the same location. The chain and sprocket has a negative one because while it has teeth to prevent slipping, it is more prone to slack and will be more difficult to tension accurately. Simplicity is given a weight of two because we have a limited amount of tools and materials to create this system so it cannot be overly complicated. Gears are the lowest because we have to worry about ensuring the gears are meshing, while with the other systems have a much more flexible mounting system. A pulley is the simplest system, shortly followed by a timing belt, followed by a chain and sprocket. Easy of assembly is given a three because like simplicity, we have a limited set of teels and materials to assemble this system. The weights are the same as they are for simplicity for the this reason. Aesthetics is given a weight of one because it is not important at all to the success of the mechanism and is purely for show. The chain and sprocket given the highest value of 1 since it is considered to be the most visually impressive. Based on this pugh chart, the transmission that we have chosen is the timing belt. It is very reliable, easy to make, and is cheap. In order to make a timing belt transmission system, we will need to buy a timing belt, input pulley, and output pulley. The components and manufacturer of each part are described in Table 5 below.

Table 11.5: Transmission components and sources

Component	Manufacturer	Part Number	Cost
¼” MXL Series Dust-Free Timing Belt	McMaster-Carr	1679K69	$2.55
Mxl Series Timing Belt Pulley for 1/4" Maximum Belt Width, 20 Tooth	McMaster-Carr	1375K42	$11.02
Mxl Series Timing Belt Pulley for 1/4" Maximum Belt Width, 40 Tooth	McMaster-Carr	1375K55	$14.24

Section 12: Final Transmission Design

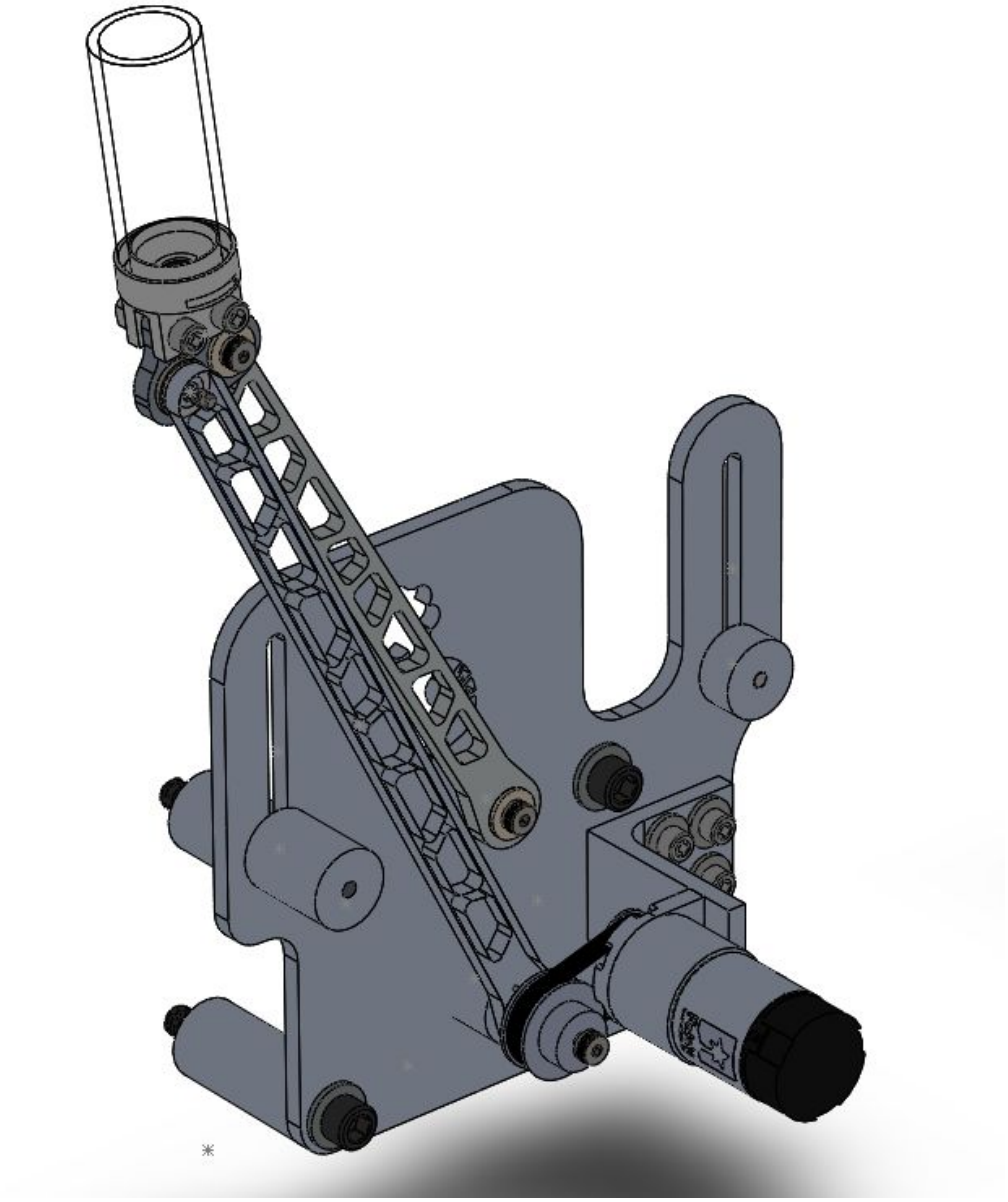


Figure 12.1: Full CAD Model with Transmission

The final transmission design for our mechanism (figure 12.1) utilizes a timing belt with a 2:1 transmission ratio and adjustable motor position along parallel slots in the mounting plate. The timing belt is cheap, has lower tolerances than meshing gears, is lightweight, and doesn't require lubrication. The parallel slot in the mounting plate shown in figure 12.2 will allow about 0.25" of linear tensioning which will allow our transmission design to be flexible and accommodate stretching of the timing belt.

The transmission will include a 0.685”OD, 20 tooth aluminum timing belt pulley on the motor shaft, input, a 1.21”OD, 40 tooth aluminum timing belt pulley on the output, and a 0.25” urethane timing belt coupling the two. The pulleys come with set screws that constrain them along the axis of the shafts and the output pulley will have a 1/16” hole in it with a spring pin for transferring torque to the input link of the mechanism.

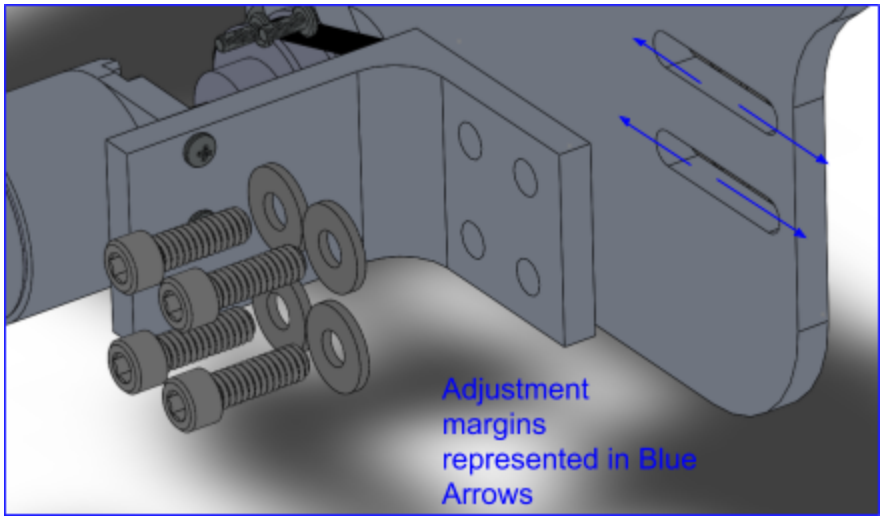


Figure 12.2: Horizontal Adjustment of Motor

The motor can be moved along the mounting plate through parallel slots and has four points of contact, two bolts with washers, along each slot, This allows the motor to have a lot of adjustability for tensioning, and enough friction to keep the motor from moving during operation of the mechanism (Figure 12.2). The motor can also be moved off the plate with spacers, to accommodate for any manufacturing error that may cause misalignment between the two pulleys (Figure 12.3).

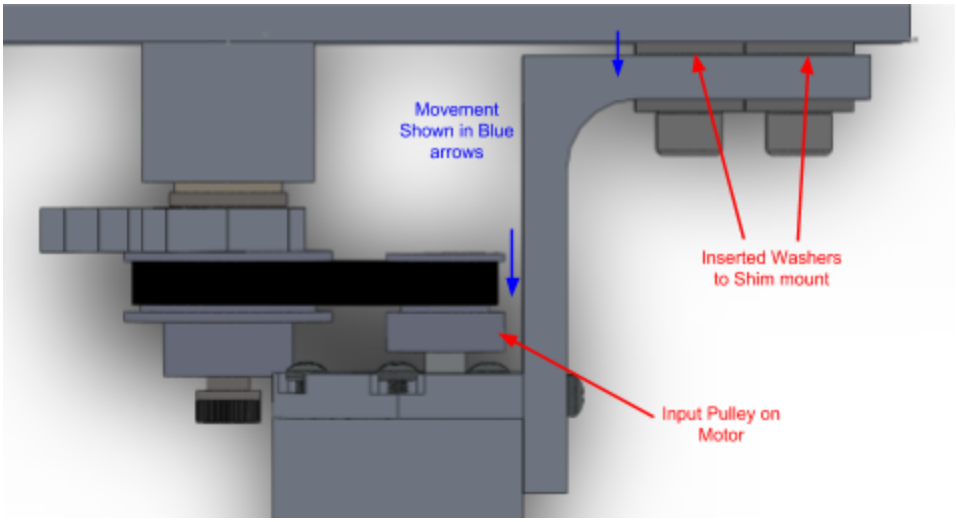


Figure 12.3: Vertical Adjustment of Motor

The volume of the mechanism is 506.45cm^3 and the volume of the transmission design, including motors, pulleys, belt, mounts, and fasteners, is 91.873cm^3.

Section 13: Gravity Compensation

Given that gravity will affect the acceleration and the deceleration of our linkage mechanism, we needed to compensate for its effects. The method used to compensate for gravity involved computing the center of mass and holding the linkages stationary against gravitational acceleration in ADAMS. Using a safety factor of 5, the available torque from our motor was 0.13N-m. The values gathered from ADAMS were from using a constant torque to hold the linkage system in one position at either extreme of its motion; this is where gravity would have the greatest effect. The transmission ratio was found by dividing the simulated torque by the motor torque with a safety factor of 5. From this data, the voltage that we wish to run the motor at can be found in each position respectively. For our specific mechanism, our compensation minimum gear ratios were 0.535 in the leftmost position and 1.648 in the rightmost position and our torque required was .2413 Nm and .0696 Nm respectively . Calculating these ratios and their respective voltages is necessary in this project because gravity will affect the angular velocity of the coupler and how the ball is released when depositing it into the box or flinging it into the net. Furthermore, the transmission ratios give us the minimum torque required to move the mass of the linkages; if the transmission ratio (N) is smaller than the recommended ratio, the linkages will not move and the motor would fry if it is ran at a voltage outside of the safe long-term range. Obviously, we want the minimum torque and voltage necessary to complete the deposit motion of the linkage system. This will also result in the least amount of power being consumed to perform the task. To calculate the torque required to hold the mechanism still at each position, we use Equation 13.1.

$$T_R = T_{Gravity} / N \qquad \textbf{Eq. 13.1}$$

Equation 13.1 shows that the torque required is the torque needed to counteract gravity divided by the transmission ratio on the belt and pulley transmission. Using the torque values calculated in the ADAMS simulation at the left and rightmost positions, we can get a torque required after applying our chosen ratio of 2. Table 13.1 shows the required torque after applying transmission ratio.

Table 13.1: Torques Required to overcome Gravity After Transmission Ratio

Position	**Torque required $T_{Gravity}$**	**Torque After Ratio T_r**
Position 1 (left-most)	0.2413 Nm	0.1201 Nm
Position 3 (right-most)	0.0696 Nm	0.0348 Nm

$$V_s = (T_R * R)/(K_t) \qquad \textbf{Eq. 13.2}$$

Equation 13.2 requires the use of constants not directly given in the Pololu motor specs. R and K_t are not listed on the spec sheet, but can be found using the specs that are shown. R was given by a matlab

program provided by the site, while K(slope) was found by using no load speed and stall torque at 10V. The following equation was then used to solve for K_t, assuming the motor is linear and conservative. All units were converted to metric scale, variables used are in Table 13.1.

$$K_t = T_s/I_s \qquad \textbf{Eq. 13.3}$$

Table 13.1: Variables Used to find Motor Constants

Variable	Value
R	2.4 Ohms
ω_0	27.49 Rad/s
T_s	0.583 Nm
I_s	3.75 A
K_t	0.155 Nm/A

After solving for Kt, we can now use equation 13.2 to find V_s at the two most extreme points in the mechanisms range of motion.

Table 13.2: Voltage required at Extreme positions

Position	Torque Required after Ratio T_r	Voltage required V_s
Position 1 (left-most)	0.1201 Nm	1.859 V
Position 3 (right-most)	0.0348 Nm	0.537 V

The maximum voltage does not exceed our limit of 10V at the two extreme cases to overcome gravity. Our current safety factor against shorting the motor is then 5.377 when trying to begin motion.

Section 14: Power Analysis

The power analysis being performed is supposed to show the maximum output our linkage mechanism is able to provide. This analysis will show the limitations our mechanism experiences due to lack of power and the maximum rate of our armature when supplied with 10V and transmitted through our gearing ratio. The table below shows the relevant values that were used to calculate the maximum torque required, the total moment of inertia of the system, and other operating values of the mechanism and the motor.

Table 14.1: Variables/Equations for Power Analysis

Variable	Value	Equation
I_M (Motor Inertia)	25,000 gcm^2	
I_L (Linkage Inertia)	57670.38 gcm^2	
I_T (Total inertia)	39417.595 gcm^2	$I_{Motor} + (I_{Load} / N^2)$
N (Gear ratio)	2	From Transmission selection
T_{max}	0.09676 Nm	$T_{max} = I_T \times \alpha_{required}$
ω_{max}	315 deg/s	From ADAMS
$\alpha_{required}$	1406.41 deg/s^2	From ADAMS
T_{motion}	0.45 s	From ADAMS
V_S	1.498 V	$V_s = (T_R * R)/(K_t)$

Total inertia of the system is calculated by adding the load inertia plus the reflected inertia of the motor. This is shown in Equation 14.1 below:

$$I_{Total} = I_{Motor} + I_{Load} / N^2 \qquad \textbf{Eq. 14.1}$$

$$I_{Total} = (25000) + (57670.38) / (2)^2$$

$$I_{Total} = 39417.595 \ gcm^2$$

After computing the total inertia, we assume a triangular velocity profile to compute the maximum speed ω_{max} and the required acceleration $\alpha_{required}$ to move the mechanism. These values are computed in ADAMS. Using the total inertia from Equation 14.1 and this $\alpha_{required}$, we can find the required torque T_{max} in Equation 14.2 below:

$$T_{max} = I_T \times \alpha_{required} \qquad \textbf{Eq. 14.2}$$

$$T_{max} = 39417.595 \ gcm^2 \times 1406.41 \times (2\pi /360) \ rad/s^2$$

$$T_{max} = 0.09676 \ Nm$$

Our transmission ratio of 2, the maximum speed ω_{max}, and the maximum torque T_{max} define the most demanding operating point for the motor. The supply voltage needed to operate the motor at this point is given by Equation 14.3 below:

$$V_s = (T_R * R)/(K_t) \qquad \textbf{Eq. 14.3}$$

$$V_s = 1.498 \text{ V}$$

Therefore, the T_{motion} of 0.45 seconds is achievable with the given limitations and a safety factor of 6.67. The max speed we can achieve with the limitations of our motor is .068 seconds.

Section 15: Torque Transfer Analysis

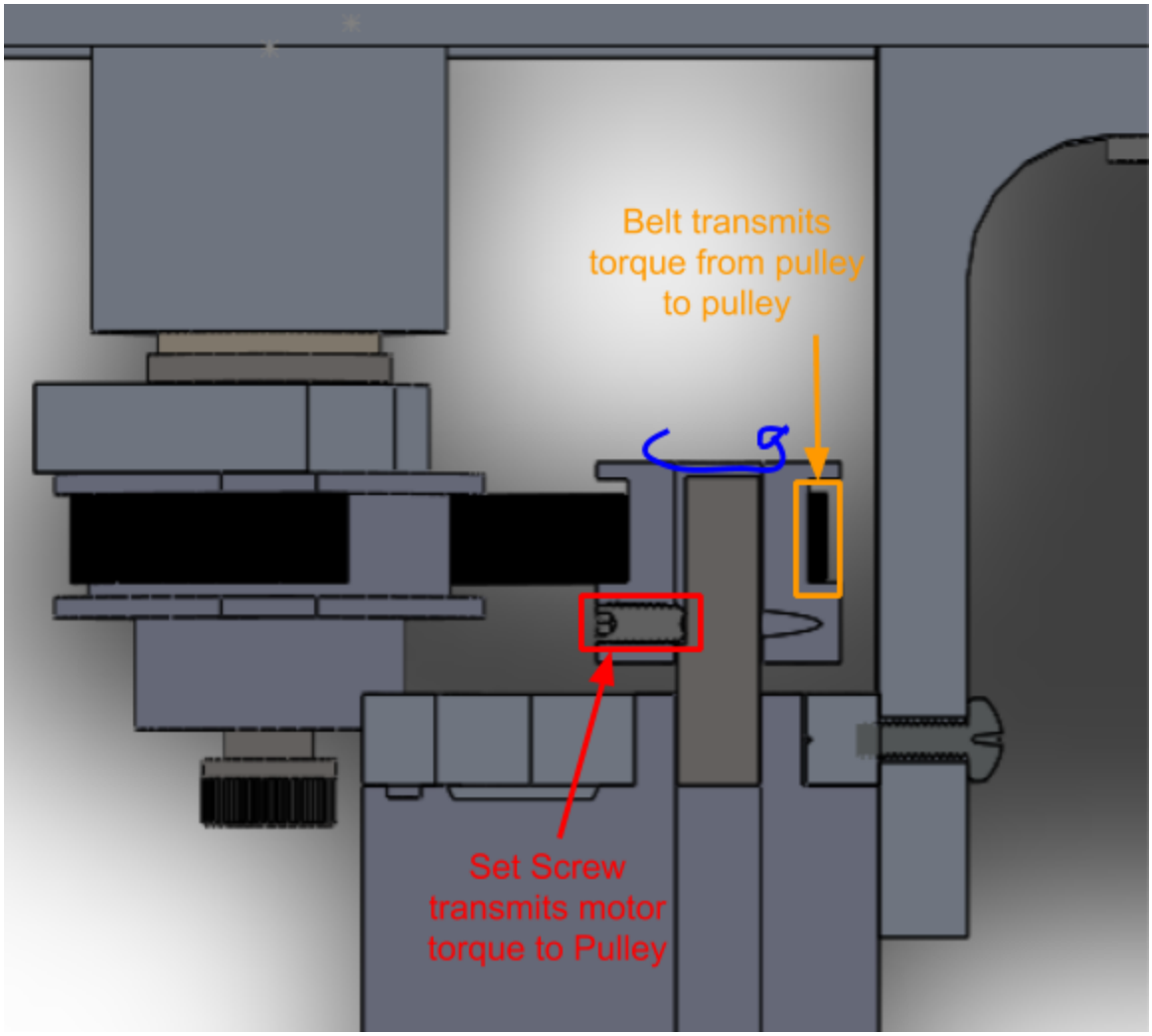


Figure 15.1: Cross Section of Torque Transfer to Pulley System

The torque is being transferred from the motor to the input timing belt pulley through a set screw (Figure 15.1), from the input pulley to the output pulley through the timing belt, and from the output pulley to the input lever of the mechanism through a spring pin (Figure 15.2).

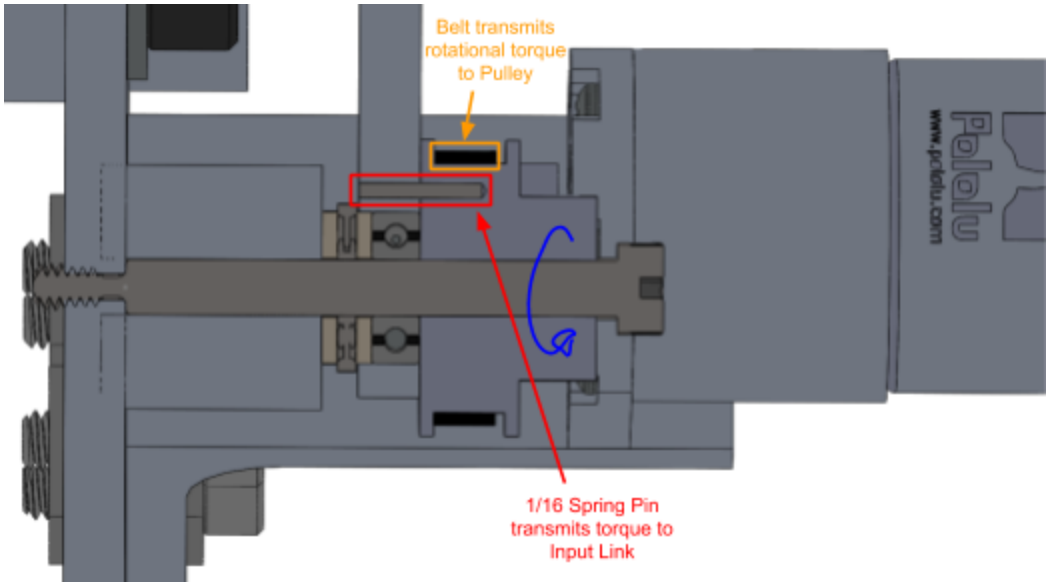


Figure 15.2: Cross Section of Torque Transfer to Input Link

The following Free Body Diagrams describes how the force from the DC motor is transferred to the Input Link. Below in figure 15.3 the Motor provides a Torque (T_m) which spins the mechanism. The Set screw inside of the Pulley is pressed down with force F providing a frictional force (F_f) that conjoins the Motor shaft and the Pulley. This frictional force is what limits how much normal force (F_p) can be placed on the pulley, and as a result, the amount moment/torque (M_p) that can be put into the system.

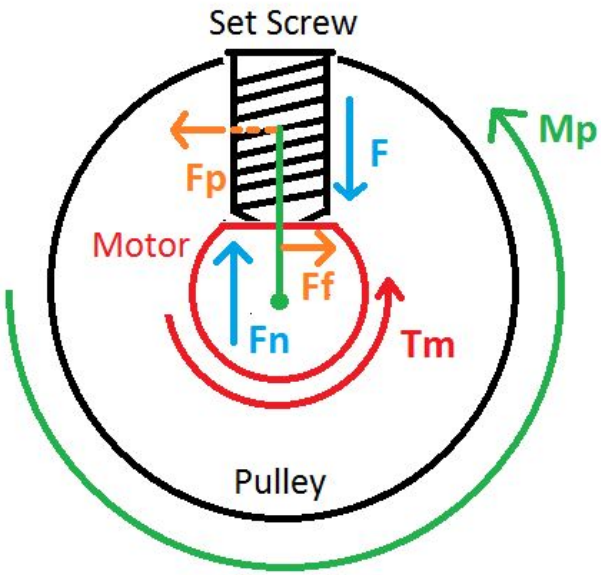


Figure 15.3: Free Body Diagram of Torque Transfer from Motor to Pulley

The moment in the pulley (M_p green) is then transferred into the attached belt. This Belt runs to another pulley and transmits the torque by using teeth and frictional force (F_p). As a result another Moment in the second pulley is transmitted (M_p red)

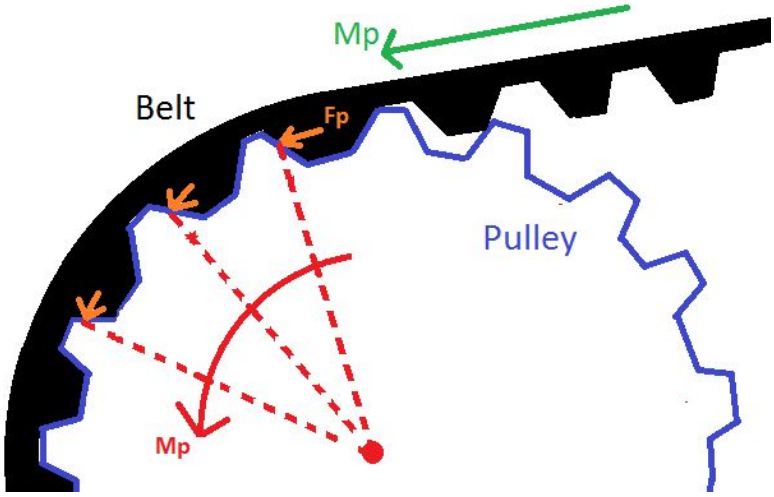


Figure 15.4: Free Body Diagram of Torque Transfer from Belt to Pulley

The moment in the second pulley is then transmitted into the Input link using a spring pin. The moment from the pulley (M_p) causes a force on the spring pin (F_p) which is directly transferred into the link by Newton's third law. F_p now causes a moment on the Input Link creating the final rotational motion (M_{link}).

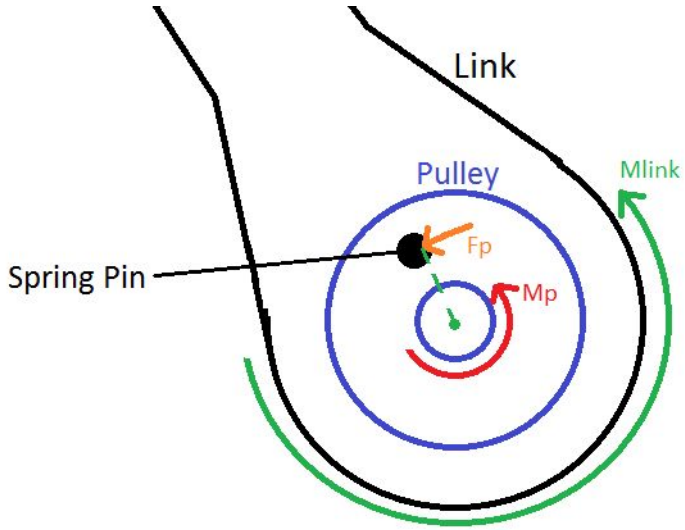


Figure 15.4: Free Body Diagram of Torque Transfer From Pulley to Input Link

Torque transfer analysis is being performed in order to verify if the current design can withstand the forces input into the system. It also provides understanding of where forcing are coming from and where they are being applied.

In order to calculate the torque transmitted, T_i, to the input lever of the mechanism, we first had to calculate the theoretical torque that would be applied without any energy loss, T. This was done by multiplying the max output torque of the Pololu 2823 gearbox motor of 0.268 Nm, from the ADAMS simulation shown in Figure 6.4, by the gear reduction ratio, GR, of the timing belt as shown in **Eq. #1 in Table 15.1**. This gave us a torque of 0.536 Nm without any energy loss. We then took into account the efficiency, e, of the motor and the that would both result in losses in the torque transmitted to the input lever.

In order to find the efficiency of the motor we first had to find the operating and no load speed. The no load speed was found by creating a torque speed curve for the Pololu motor at 6V from the data sheet provided on the manufacturer's website. The stall torque, 0.338Nm, and no load speed, 175 RPM, at 6V were then multiplied by a factor of 1.5 in order to create a torque speed curve at 9V. The transmission ratio of 2:1 was then applied to the torque speed curve by multiplying the stall torque by a factor 2 and dividing the no load speed by a factor of 2 as shown in **Eq. #2 & #3 in Table 15.1.** Next we used **Eq. #4 in Table 15.1** to calculate the torque speed gradient, K, and **Eq. #5 in Table 15.1** to calculate the operating speed, ω, at 0.536 Nm. This operating speed of 70.8637 RPM was then used in **Eq. #6 in Table 15.1** to calculate the efficiency of the motor, which came out to 54%. This efficiency was then applied to the conservative torque output of the motor to find the torque transmitted to the input lever, assuming 100% efficiency of the timing belt itself.

After applying the efficiency to the conservative torque, **Eq. #7 in Table 15.1,** we get a torque of 0.28944 Nm, T_i, transmitted to the input link. The radial distance of the spring pin that transfers this torque from the pulley to the input lever is 0.030734m, r. Using **Eq. #8 in Table 15.1** the force transmitted to the input lever by this torque, F, is 9.41758N. This force is applied by the spring pin to the mechanism across about half of the inner surface area of the 1/16" diameter, 0.25" depth, hole allocated to it in the input lever. This surface area, A, is found using **Eq. #9 in Table 15.1**. The stress on the lever, σ, can then be found using **Eq. #10 in Table 15.1.** The force also causes a shear stress, in the spring pin and set screw across the area perpendicular to the force, calculated using **Eq. #12 in Table 15.1**. This shear stress is found using **Eq. #13 in Table 15.1**.

The last step of the stress analysis is to determine if the mechanism will fail. This failure was defined as the aluminum in the input link of the mechanism, the spring pin, or the set screw yielding under stress. In order to determine if this was the case, we solved for the safety factor, SF, SF_2, and SF_3 against yielding using **Eq. #11 and #13 in Table 15.1**. This safety factor, with an assumed minimum yield stress of 240MPa for aluminum, σ_y, was found to be 403.5, which shows that it is very unlikely that our mechanism will yield. The safety factor, with an assumed minimum yield stress of 10787 MPa for the spring pin in shear, τ_y, was found to be 440.26 , which shows that it is very unlikely that our spring pin will yield. The safety factor, with an assumed minimum yield stress of 250 MPa for the steel set screw in shear, τ_y, was found to be 52.63, which shows that it is very unlikely that our set screw will yield.

Table 15.1: Variables and Equations for Torque Analysis

Variable	Value	Equation	Equation #
T	0.536Nm	T=GR (T_a)	1
T_s @9V	1.165Nm	T_s = (1.5)(2)(T_s @6V)	2
ω_0 @9V	131.25RPM	T_s, ω_0 = (1.5)(0.5)(ω_0 @6V)	3
K	0.008876 Nm/RPM	$K = T_s / \omega_0$	4
ω	70.8637RPM	ω = (T_s -T)/K	5
e	54%	$e = 100(\frac{\omega}{\omega_0})$	6
T_i	0.28944Nm	$T_i = 0.54\,(T)$	7
F	9.41758N	$F = T/r$	8
A	1.58346E-5m^2	A= π(1/16)(.5)(.25) (.0254)^2	9
σ	0.594748 MPa	$\sigma = F/A$	10
SF	403.5	$SF = \sigma_y/\sigma$	11
A_s	1.98E-6m^2	A= $\pi(1/32)^2$ (.0254^2)	12
τ	4.75MPa	$\tau = F/A$	13
SF_2	440.26	$SF_2 = \tau_y/\tau$	14
SF_3	52.63	$SF_3 = \tau_y/\tau$	14

Section 16: Safety & Motor Controls Introduction

Safety features and motor controls play a large part in the operation of the linkage mechanism when delivering balls to their respective destinations. Safety features generally prevent the operator or mechanism from being damaged under unusual operating conditions. One such feature implemented in our design is the set of hardstops at either extreme of the linkage motion. This is to prevent the linkage from damaging itself from swinging past its intended path or reaching a “toggle point” where the linkage may not be able to return to its intended position. An unintended safety mechanism included in the mechanism is the slipping feature the timing belt provides if the motor is overloaded with torque. Rather than burning out the motor, the belt will slip past the notches on the pulley and allow the belt to rotate more freely. The quality of the controls determines the outcomes of all measured quantities in the simulations we have run. Without properly reading in the position of the linkage mechanism, the motion of the system becomes unpredictable because the start point is incorrect. This could lead to a discrepancy in velocity of the input arm and, in turn, the exit velocity of the marble being sorted. Incorrect readings of the ball color are even more detrimental because they will be sorted into the wrong bin if all other system readings and calculations are correct. The sensors and electronics we are using to complete this include a breadboard, a relay terminal, an Arduino Uno, a Pololu motor with an encoder, a limit switch, a toggle switch, and a color sensor.

Section 17: Capabilities & Limitations of Sensors

We are using two sensors: a color sensor in the cup and an encoder on the motor. The color sensor is an Adafruit TCS34472. It measures the colored light reflected off of the ball and assigns it an integer value between 0 and 65535. The four colors in measures are red, blue, green, and clear light. Each integer is converted to 16-bit binary data and sent to the arduino where the signal is decoded. A stable input voltage and light source is needed to operate this sensor so it contains a voltage regulator and small LED. The values found using this sensor vary during testing based on the orientation and motion of the mechanism. In order to combat this, we calibrated the sensor while inside the cup, which is its final location, and glued it in place to keep its environment consistent. This reduces variance, but some error is still present.

The second sensor we are using is the Hall Effect encoder attached to the back of our motor. It has two channels and uses a magnetic field in order to measure the rotation and speed of the motor. It converts angular displacement into digital output which will be read by the arduino. Since there are two channels in this encoder, it can determine which direction the motor is turning by examining which square wave signal rises first in both A and B. An important consideration with the encoder is that it reads revolutions of the motor and not revolutions of the output shaft. Since the gearbox attached to the motor is 30:1 and a 2:1 transmission is connecting it to the linkage, the link only moves one rotation for every 60 rotations of the motor shaft. This is important in order to get a fine resolution of 0.5 mm needed to position the cup accurately. The encoder is not perfect, which can result in error in locating the mechanism. During testing, we found that the mechanism would be off by several encoder counts at known positions after testing. This is because small errors in the encoder occur which stack up the longer testing continues. In order to combat this, we have the limit switch attached at the left endstop of the mechanism. Whenever

the linkage returns to this position, it will zero itself thus eliminating any and all error built up by the encoder.

Sensor 18: Mounting Considerations & Methods

Color sensor placement was done within the cup that captures and contains the balls. The 3D printed part that holds the cup also was designed with the color sensor position in mind. A small slot was printed into the side that we slide the sensor in and out of without having to dismantle the cup. Below is a picture of the CAD model to further describe the cups features.

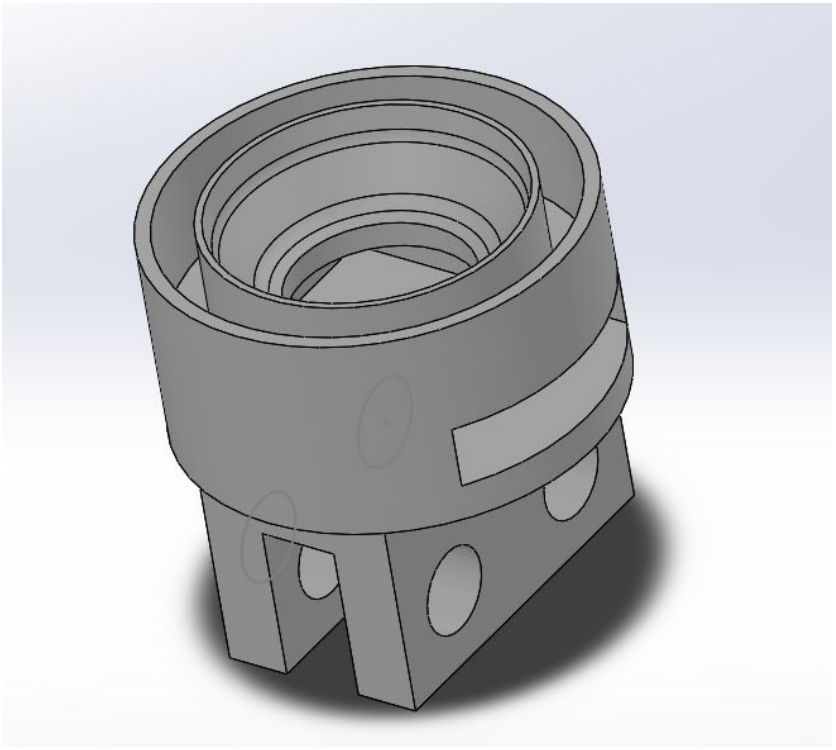


Figure 18.1: Color Sensor CAD

The design of the curved surface was calculated to allow the ball to be at a close distance to the LED sensor. Having closer proximity to the LED sensor provides better reliability in determining the type of ball color within the cup. The curved shape of the ball holder also provides stability and ensures the ball doesn't bounce around while the sensor is trying to make a reading.

In order to fix the color sensor within the 3D printed part without a permanent fixture, hot glue was used. This would allow enough rigidity for mechanism use, but also provide ease of removal should the sensor need to be taken out. A picture below shows the color sensor within the 3D part.

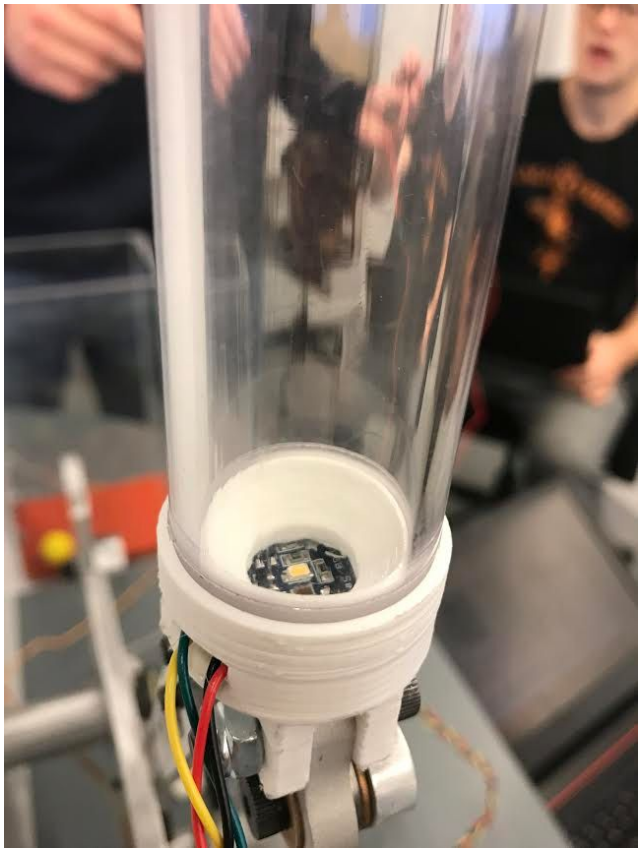


Figure 18.2: Color Sensor In Assembly

Our chosen method for securing the limit switch was mounting it with double sided tape for the preliminary testing. This method provided quick and easy adjustment to our design when trying to find the correct placement for the sensor. Being our preliminary method of attachment, this would not be an acceptable method in industry and we would require more stable mounting points. After a reasonable position is found for the limit switch, we plan to use bolts to mount it to the base board.

Section 19: Encoder Counts, Color Sensor Thresholds, and Controller Gains

Encoder Counts:

The encoder counts for the chute, wait, and put positions were determined by running our arduino code with the 0 encoder count position aligned with the left end stop. After moving the mechanism by hand qualitatively to the desired location, we recorded the encoder count displayed on the serial monitor. These encoder counts were used to define the location for each position. For this process we ensured that there wasn't slip in our mechanisms transmission and that the encoder circuit was correctly assembled We could have used a more theoretical approach, where we calculate the angle of the input link at each position and calculate the number of encoder counts by using the number of degrees the linkage turns

with each count. We decided to not use this approach because we know that there is some error in our mechanism and that we cannot start the mechanism in the exact correct position.

Color Sensor Thresholds:
The color sensor thresholds were determined through the process outlined in lab. The balls were placed in the one inch cup, and the red, blue, green, and clear color sensor values were displayed on the serial monitor. From these values, a range of values was set for each ball color that was large enough to encompass all variations found in experimental testing. Each color has a unique range of values that the code uses to determine the ball's color in the cup.

Controller Gains:
In order to calibrate the gains we first set the parameter “activeChutePosition” to a target position. We then set the derivative gain "KD" and integral gain “KI” to 0 and gradually increased the proportional gain "KP" starting from 0.01, until the system started to overshoot or oscillate rapidly. This value was divided by two and set as our KP. Next we set the proportional and integral gain to 0 and gradually increased the "KD" starting from 0.001, until the system started to oscillate. This value was divided by two and set as our KD. Next we set the proportional and derivative gains to the previously determined values and gradually increased the "KI" starting from 0.001, until the system came to rest at the target position and all steady state error was remedied. Lastly we made sure the PID controller worked at every position and that there wasn’t any significant overshoot.

Table 19.1: Summary of Variables in Arduino Code

Variable Name in Arduino Sketch File	Purpose of Variable	Device	Calculated Value	Actual Value Used During Testing
ChutePosition1	Chute Position #1	Motor	0	0
ChutePosition2	Chute Position #2	Motor	NA	407
WAIT_POSITION	Wait Position	Motor	NA	200
PUT_POSITION	Put Position	Motor	NA	722
K_p	Proportional Gain	Motor	N/A	0 .15
K_i	Integral Gain	Motor	NA	0.01
K_d	Derivative Gain	Motor	NA	0.0085
Blue Color Values	Value ranges of the color sensor to determine a blue ball	Color Sensor	N/A	R: >1000, <2500 B: >4200, <5200 G: >2700, <3500

				C: >8000
Maize Color Values	Value ranges of the color sensor to determine a maize ball	Color Sensor	N/A	R: >8000, <10000 B: >4000, <6000 G: >8000, <10000 C: >20000
Red Color Values	Value ranges of the color sensor to determine a Red ball	Color Sensor	N/A	R: >2000, <3500 B: >1600, <2500 G: >1600, <2500 C: >5500
White Color Values	Value ranges of the color sensor to determine a White ball	Color Sensor	N/A	R: >10000, <20000 B: >10000, <20000 G: >10000, <20000 C: >15000
BASE_CMB	Voltage needed to overcome friction at the mechanisms balanced position	Motor	NA	2V
FF_VOLTAGE_LOWER_BOUND	Voltage needed to overcome gravity at the chute 1 position (first hardstop)	Motor	1.859 V	4.75
FF_VOLTAGE_UPPER_BOUND	Voltage needed to overcome gravity at the second hardstop	Motor	-0.537 V	-5

Section 20: Arduino Code Changes

Constants and Encoder Count Values

The first changes made to the code were constant values inputted for voltages and encoder positions. These values were found by qualitative testing. Voltages and encoder count values were inputted until a satisfactory value was reached. These values can be seen in code lines 63-103.

Separate Cases for Michigan and Ohio balls

In line 337- 358 of the code below, we had the code navigate to separate cases for depositing the Michigan and Ohio State balls. Instead of there being a generic case for depositing the ball, the color sensor feedback determines whether to proceed with case PUT_UMICH or case PUT_BALL. This allowed the mechanism to behave differently depending on which goal we wanted to score in. We also included an else statement that defaults to the PUT_BALL case, but this else statement will likely never be reached because of the nature of the Color Sensor code defaulting to MAIZE. We added the code as a safety precaution should the color sensor fail and return an error when called.

```
if(ballColor == MAIZE || ballColor == BLUE) {
//CalibrateCounter = CalibrateCounter + 1;
  Serial.println("State transition from WAIT_FOR_BALL to PUT_UMICH");
  state = PUT_UMICH;
  break;
}
if(ballColor == RED || ballColor == WHITE) {
//CalibrateCounter = CalibrateCounter + 1;
  Serial.println("State transition from WAIT_FOR_BALL to PUT_BALL");
  state = PUT_BALL;
  break;
}
else {
  Serial.println("State transition from WAIT_FOR_BALL to ELSE");
  state = PUT_BALL;
  break;
}
```

Figure 20.1: Code for Michigan and Ohio Balls

Case and Position Specific PID Values

After creating separate cases for Ohio and Michigan balls, chute specific throws were desired. These cases would provide a tailored speed to ensure the ball is deposited into the correct basket based off of which chute they were dropped from and their color.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// 6************************************************** PUT_UMICH ************************************************************6 //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// In this state, we move to the correct position for placing MAIZE and BLUE balls into the bucket
case PUT_UMICH:

  // ************************************************** CHUTE 2 ************************************************************
  if(motorPosition <= (CHUTE_2_POSITION + TARGET_BAND) && motorPosition >= (CHUTE_2_POSITION - TARGET_BAND) && motorVelocity == 0)
  // If the mechanism reached CHUTE 2 and stopped, proceed to placing ball
  // PID values are assigned to ensure better placement
  {
    KP             = 0.11;      // [Volt / encoder counts] P-Gain
    KI             = 0.01;      // [Volt / (encoder counts * seconds)] I-Gain
    KD             = 0.006;     // [Volt * seconds / encoder counts] D-Gain

    Serial.println("PUT_UMICH");
    targetPosition = UMICH_POSITION;
    // Since this is for the BLUE and MAIZE case, we move to the UMICH color encoder position
  }

  // ************************************************** CHUTE 1 ************************************************************
  if(motorPosition <= (CHUTE_1_POSITION + TARGET_BAND) && motorPosition >= (CHUTE_1_POSITION - TARGET_BAND) && motorVelocity == 0)
    // If the mechanism reached CHUTE 1 and stopped, proceed to placing ball
    // PID values are assigned to ensure better placement
  {
    KP             = 0.125;     // [Volt / encoder counts] P-Gain
    KI             = 0.01;      // [Volt / (encoder counts * seconds)] I-Gain
    KD             = 0.006;     // [Volt * seconds / encoder counts] D-Gain

    Serial.println("PUT_UMICH");
    targetPosition = UMICH_POSITION;
    // Since this is for the BLUE and MAIZE case, we move to the UMICH color encoder position
  }
```

Figure 20.2: Chute Specific PID Values

As seen in code lines 408-440 , the PID values are tailored to a specific value that was found qualitatively. Balls were thrown using different values until an optimal throw was achieved. In the Ohio color ball case from Chute 1, we discovered that moving to chute 2 before throwing was the most accurate option. Code lines 367-389 describe this situation. The loop will first evaluate the Chute 1 position case as true, move to the second case, then continue to throw.

Recalibration after every throw

Because of the belt drive there was small drift that would occur after smashing into the endstop. To alleviate this issue, a piece of code was written that would automatically calibrate the mechanism after depositing every ball.

```
// Decide what to do next:
if (motorPosition <= (PUT_POSITION + TARGET_BAND) && motorPosition >= (PUT_POSITION - TARGET_BAND) && motorVelocity == 0 ) {
  // We reached the basket and dropped the ball.
  // Transition into WAIT state to restart the cycle
  Serial.println("State transition from PUT_BALL to WAIT");
  targetPosition = WAIT_POSITION;
}
if (motorPosition <= (WAIT_POSITION + TARGET_BAND) && motorPosition >= (WAIT_POSITION - TARGET_BAND) && motorVelocity == 0) {
  // After reaching WAIT, we calibrate before picking up another ball
  state = CALIBRATE;
  Serial.println("State transition from PUT_BALL to CALIBRATE");
  break;
}
// Otherwise we continue moving towards the chute
break;
```

Figure 20.3: Re-Calibration Code

This piece of code appears at the end of each ball placement state. Because calibrating from the rightmost endstop caused too violent of a smash into the left end stop, the motor first ensures it reaches the wait position calmly, then proceeds to move into calibration state. The lines of code 391-405 execute this operation.

Reliable Color Evaluation

Color sensor code was first changed by reading values from the color sensor and creating bounds for specific color cases. This would allow the code to decide what color the ball was based on the number values of RGB and Clear. Lines 595-610 complete this.

Instead of evaluating the color once, the code was written in lines 612-669 so that the color of the ball is repeatedly assessed until the color reading is consistent twice in a row or the code times out. This was done by making an additional variable, ballType2, and only returning the ballType if both variables agree or the counter reaches 15. If the counter value is reached, the is ballType defaulted to MAIZE, sending the ball into the PUT_UMICH state after running through the Michigan/Ohio cases explained above. This allows for more precise assessments of the color and filters any incorrect readings that may have occurred. If the ball was not caught or couldn't be read, the counter system ensures that the mechanism will still throw after enough time has passed. The picture below also displays this code.

```
// ************************************* COLOR COMPARISON ********************************************************
// Code that compares the two ball type colors and will return their value if they are the same.
// NOTE: Two NONES will continue running the loop. This ensures the color sensor outputs an actual color.
if(ballType2 == ballType && ballType != 5)
{
  return ballType;
}

// If the ball color is still NONE, continue running the loop.
else

  if(COUNTERC >= 15)
  // If the colors are measured 15 times and there is still no value, assign it as a UMICH ball and break the loop.
  // This protects against balls being dropped or a ball outside of the color range (usually blues and yellows).
  {
    ballType = 1;
    return ballType;
  }

// Keep running the color sensor evaluation until above If statement is true
return evaluateColorSensor() ;
```

Figure 20.4: Color Comparison and Counter Timeout

Wait Time Reduction

Wait times were reduced in operations such as color sensing and also in the constant that appears in line 77. Reducing this number simply made the mechanism move quicker, thereby increasing our possible score.

Debugging

On several occasions, debugging was required after the mechanism wouldn't behave correctly. Several parts of the code thus have serial outputs that provide us information if that specific case is reached. An example of this code is lines 638-646 where the raw data sensor values are outputted as the code tries to evaluate them continuously. These values are outputted to serial monitor when the color sensor couldn't successfully determine a final ball color. From a testing standpoint, this helped us calibrate the color sensor values to ensure higher accuracy.

Section 21: Final Testing Results/Discussion

Testing setup was done according to the rules of the competition; we bolted our mechanism to the board and set up our wired connections in the five minute initial setup time. Our wiring diagram was used as a reference to ensure that the mechanism was connected to the power supply, breadboard, and the playing field. The wiring diagram and Arduino code can be seen in Appendix D. The results of our testing can be seen in Table 21.1 below.

Table 21.1: Testing data

Trial	Catching			Put in Basket			Put in Net			Cali-bration	Physical Contact	Score
	1st Ball Left	1st Ball Right	Any Ball	First Ball	Correct	Incorrect	First Ball	Correct	Incorrect			
1	1	1	0	0	0	0	0	0	0	1	2	75
2	1	1	8	1	4	2	1	3	1	1	1	495
3	1	1	23	1	12	1	1	12	0	1	10	690

Testing of our mechanism did not go nearly as well as our team expected. After extensive work on the mechanism and the Arduino code that controlled the automated function of the device, our team's mechanism was able to catch balls from both chutes, and sort all balls both accurately and quickly. Our team took a video of the mechanism the day before final testing, and the link is included here:

https://www.youtube.com/watch?v=l0G-hX9gch0

Following this video, the mechanism was further tuned, and the waiting time time for balls was reduced by a factor of three, and the mechanism again completed a perfect run.

During the final testing of our mechanism, our team experienced problems with encoder counts and wire connections that resulted in our team receiving a final score that was much lower than our expectations. During setup of our mechanism, our team did not properly attach the fuse to the wire from our power supply, and it resulted in a faulty connection. As a result of this faulty connection, the motor did not have any power for the first test, and the mechanism did not move at all for the first test. The connection was fixed during the two minute adjustment period between test. For the second test, our mechanism was unable to catch some of the balls from chute one, and was unable to catch any balls from chute 2. This was due to the left hardstop being slightly out of place. This caused our motor encoder counts to be slightly inaccurate, which led to our chute positions being slightly after the device calibrated, which caused the mechanism to miss properly catching balls. During the adjustment period after the second test, our team adjusted the left hardstop, and ran out of time while we were trying to calibrate the device to find the proper encoder count for the second chute. As a result, our mechanism was unable to catch balls from the right chute, and our team missed out on a significant number of points. To say the least, our team was very disappointed in our performance during final testing.

Section 22: Design Critique & Evaluation

1. What worked well? What didn't work well? Most importantly, explain why.

The overall design we chose worked as planned for the most part. The board was easily mountable, and the limit switch was readily accessible and adjustable. The timing belt worked well as a cheap transmission and packaging option, but was susceptible to slip at high speeds and impact. This was due to its small size and lack of surface area. We changed the code to reduce the speed of the linkage when running into the endstops which eliminated our issues with slippage. The code used in the arduino was well developed during testing and during the evaluation. Unfortunately the encoder count system used in the arduino was not nearly as precise as anticipated. This led to the mechanism misreading the position of the second chute and missing many of the balls that were dropped during the evaluation period. However, upon initial testing, the mechanism sorted all of the balls correctly in 1 minute 20 seconds. The PID controls were coded properly and were able to capture a ball, read in the color and then decide to throw or place the ball into its respective container. As a whole, the mechanism did what we wanted it to do, but the position was askew during the evaluation trials.

2. What was your scoring strategy? Did it work as planned? If not, what would you improve for better results?

The scoring strategy chosen was to wait between chutes and create four different commands depending on ball color that would either throw the ball or place it in the bucket and then to recalibrate after each deposit. Though this strategy is not as time efficient due to the recalibration time, it creates a foolproof reset for every ball deposit that ensures a repeatable cup position and motion even if the belt slips. A one inch cup was also used to create an opportunity for maximum points. Although during testing the cup worked as planned, during final evaluation the one inch cup became a burden. Its small size required the encoder count to be very accurate, which caused us to miss many balls in the right chute. Using a two inch cup would have solved this issue by creating a larger room for error.

3. How well did your design perform relative to your models? Explain what you think the difference was.

As compared to the model of our design, the actual design used a bit more voltage to operate and execute the throwing command and overcome friction. The difference was likely due to human errors in manufacturing, the additional weight of the balls and other materials such as screws, washers, bearings, etc., and friction. The Initial drawings of the 4 bar linkage were surprisingly accurate. The mechanism followed the circular path very well and its endstop positions were as expected.

4. How did friction influence your built device?

Friction was not a major influence on our device. Since we used ball bearings in the links and needle roller bearings on the surface, the links rotated easily and friction in the mechanism was not a major issue. The voltage required to drive the mechanism was only slightly larger than the calculated voltage.

5. How could the control algorithm (Arduino code) be improved to make better use of the available equipment? For example, could the position be more precisely controlled? Could you make use of a feed-forward signal? How?

If there were additional color sensors on the playing field we could use feedforward control for determining the balls in the chutes before they are dropped and catching multiple balls if desirable. In this code you could also have different PID controller gains depending on the number of balls you are trying to score. For more precise control, the arduino code could also be improved to provide a smaller target band.

6. Would there be a better way to use the sensors that were provided, or to use different sensors, to accomplish the same objectives?

A sensor that would have been helpful would have been an light and laser sensor to detect exactly where the 2nd marble chute is. The encoder did a very good job at getting the mechanism to the proper chute when calibrated correctly. The issue is that every board is slightly different which throws off the calibration. Having a sensor that would detect the location of the 2nd chute during initial calibration would help alleviate this issue and allow our mechanism to be used across a variety of boards without having to worry about manual calibration.

7. What other lessons or unique observations did you make about your device and the process that you follow to develop it? Having completed it, would you do anything different – what and why?

We learned how essential wearing a belt in is to a timing belt drive. After tensioning our transmission the belt would slowly stretch and eventually require additional tensioning. This could could drift the encoder counts if slip were to occur. We also learned that a violent proportional gain could cause the coupler to dip from whiplash and actually shorten the trajectory rather than lengthening it. In a belt drive, we also learned that violent shocks can cause slip, and that surface area is very important to transmit torque. If we could have done anything different, using a larger belt or a gear drive would have been ideal to reduce the amount of slip in the system.

8. On a scale of 1-5, how would you rate the safety of your mechanism? Were the provided sensors adequate or would additional sensors and guards be necessary?

The safety of our mechanism was a 4. The mechanism never posed a safety risk when operated correctly and was robustly assembled. Initial sensors or guards were not necessary. Limiting the input voltage to 10V and having the hard stops in place adequately contained our mechanism. The only safety issues that could occur are when a user reaches into the mechanism while it's running. There are no barriers to prevent someone from putting their finger in the mechanism while it is moving and it could cause minor injuries.

9. Would you recommend that we give your device to someone else right now to use and operate? Why or why not?

Another person could use and operate our device and we would recommend giving them the mechanism with a bit of proper instruction. The only thing that they need to do before they can operate it is calibrate it based on the board. This is easy, as all that needs to be done is move the left end stop slightly, adjust the limit switch, and change one encoder value until the mechanism stops directly underneath the 2nd ball chute, This is a simple process that anyone could complete given the proper amount of time.

10. What other parts and materials would have been useful to have in your design, if you had been given them?

We think that motors with a lot of controller feedback and a color sensor imbedded into the chutes would be a valuable investment for future semesters. If the torque and speed of the mechanism were to be plotted from the feedback of the motor, tuning the mechanism and the PID controller could be done more quantitatively and any unusual behavior could be isolated and analyzed. If there were additional color sensors on the playing field, and more variability in the size of the cup, we could have an additional coding element for catching multiple balls at once and different PID controller gains depending on the number of balls you are trying to score.

11. Do you think the current point distribution is fair? How could it be modified for future semesters?

The current point distribution is fair. The project and course material both account for equal percentages and the peer reviews are structured in a way that they can have a significant impact on your grade. However, the way in which the final testing of the mechanism was graded we feel was not fair. The testing policies from last year remained, even when the demand for playing field time was greatly reduced. This caused groups to lose points because of an inability to make minor tuning adjustments to the playing environment. Had we received just a few more minutes to properly adjust our mechanism, a large difference in points could have been scored, ultimately changing our grade.

12. What would you do for a project next year in ME350?

We feel that one aspect of this project that was lacking was teamwork between the groups in the section. To improve this, we should have a project that focuses on the different groups in the section trying to work together to move the balls. Each group could have a starting area with various challenges they have to overcome to move balls from one section, or zone, to another. Each group would make a robotic mechanism player to complete these challenges and move the balls across the zones, possibly over or through acrylic walls. Challenges could include obstacles such as a large hill to move over, a pit full of cubes, or a labyrinth of sorts. At the end of the semester there would be a competition between each section where the robotic mechanism players would compete to move the most balls. Having a collaborative project like this would increase each team's motivation to succeed, foster a collaborative environment, and create high quality parts for distributed manufacturing.

The goal of this project was to design, build, and test a powered mechanism that will automatically catch falling balls in a cup and deposit them into a basket. Its purpose was to reinforce the material learned in lecture and provide a realistic experience in manufacturing and model based system design. By creating our mechanism that successfully caught and moved the balls, we met the goals and fulfilled the purpose of this project. We used the lectures on 4-bar linkages to design the mechanism, the lessons on transmission to choose the ideal type for our mechanism, lessons in proper manufacturing plans to receive accurate parts, the ADAMS tutorials to determine the power usage, the motor lecture to determine the torque and speed output of our motor, and the controls lessons to make our code. By applying the knowledge we acquired in the classroom, we became more proficient with model based and system design and will be ready to apply this knowledge in future classes or real world scenarios.

Appendix A: Individual Sketch Blocks Design, 3D Solidworks, and ADAMS Analysis

Austin Broda:

My linkage mechanism was designed using the method that was learned in section. First, a basic coupler sketch was made in SolidWorks. Next, a simple 2-D sketch of the arena was made using measurements taken from the playing board in the X50 Assembly room. The coupler sketch block was then placed copied and placed in the three key positions on the board, under the two tubes, and over the basket. Using the three point circle sketch, the ground pivots and link lengths were found from the three point circle sketch. These lengths and angles were then used to construct the 3-D SolidWorks model from the 2-D sketch.

Following construction of both the playing board and the linkage mechanism in SolidWorks, Adams modeling program was used to do preliminary testing of the mechanism. The SolidWorks model was imported into Adams, joints were created, and a varying acceleration was added to the input link to simulate the torque of the motor. From this simulation, the built in Adams postprocessor recorded the angular displacement, angular velocity, torque, and power consumption. These values are shown in the following figures. Comparing these values, the best model among the members of our was chosen to be the rough draft for the final group model.

Table A.1: Austin's Mechanical Synthesis Values

Link Lengths(Input, Coupler, Follower)	12.53"	1.61"	9.42"
Transmission Angles(Position 1, 2, 3)	98.78°	90.00°	131.98°
Transmission Angle Deviation(Position 1,2, 3)	8.78°	0°	41.98°

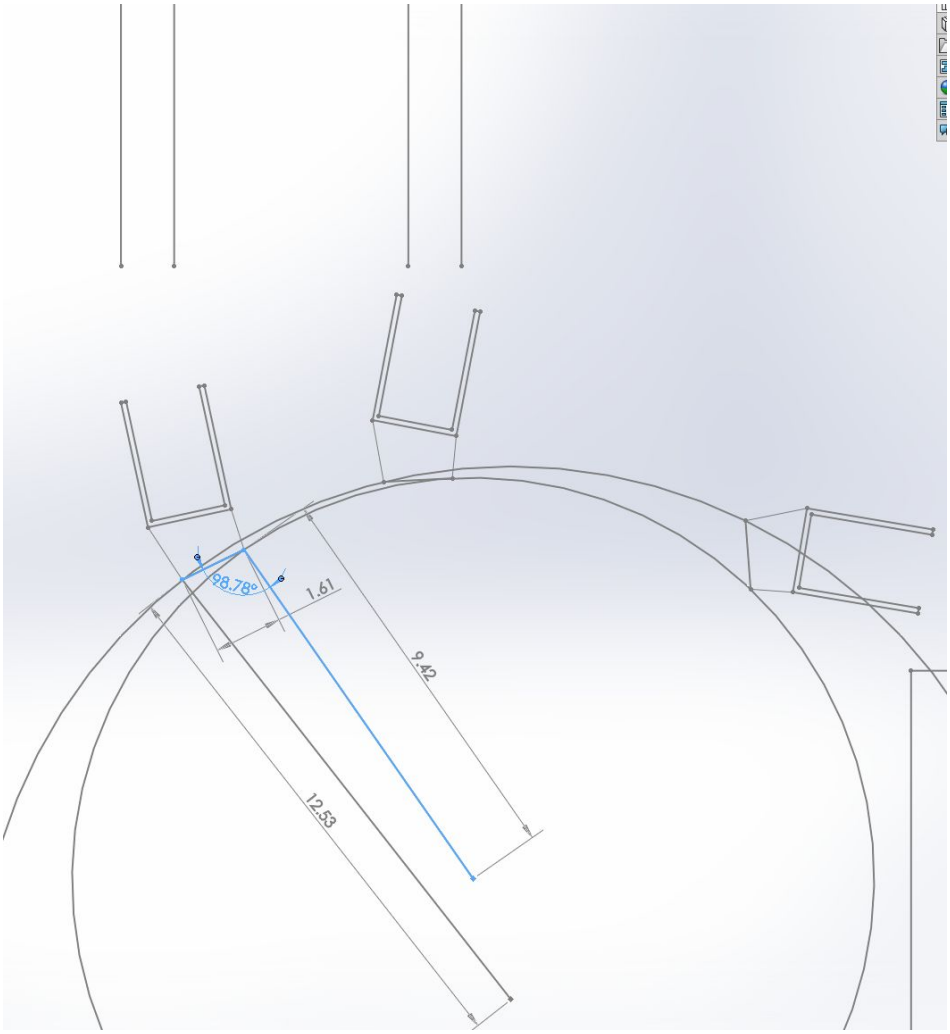


Figure A.1: Position 1 of Synthesis w/ Link Lengths and Transmission Angle

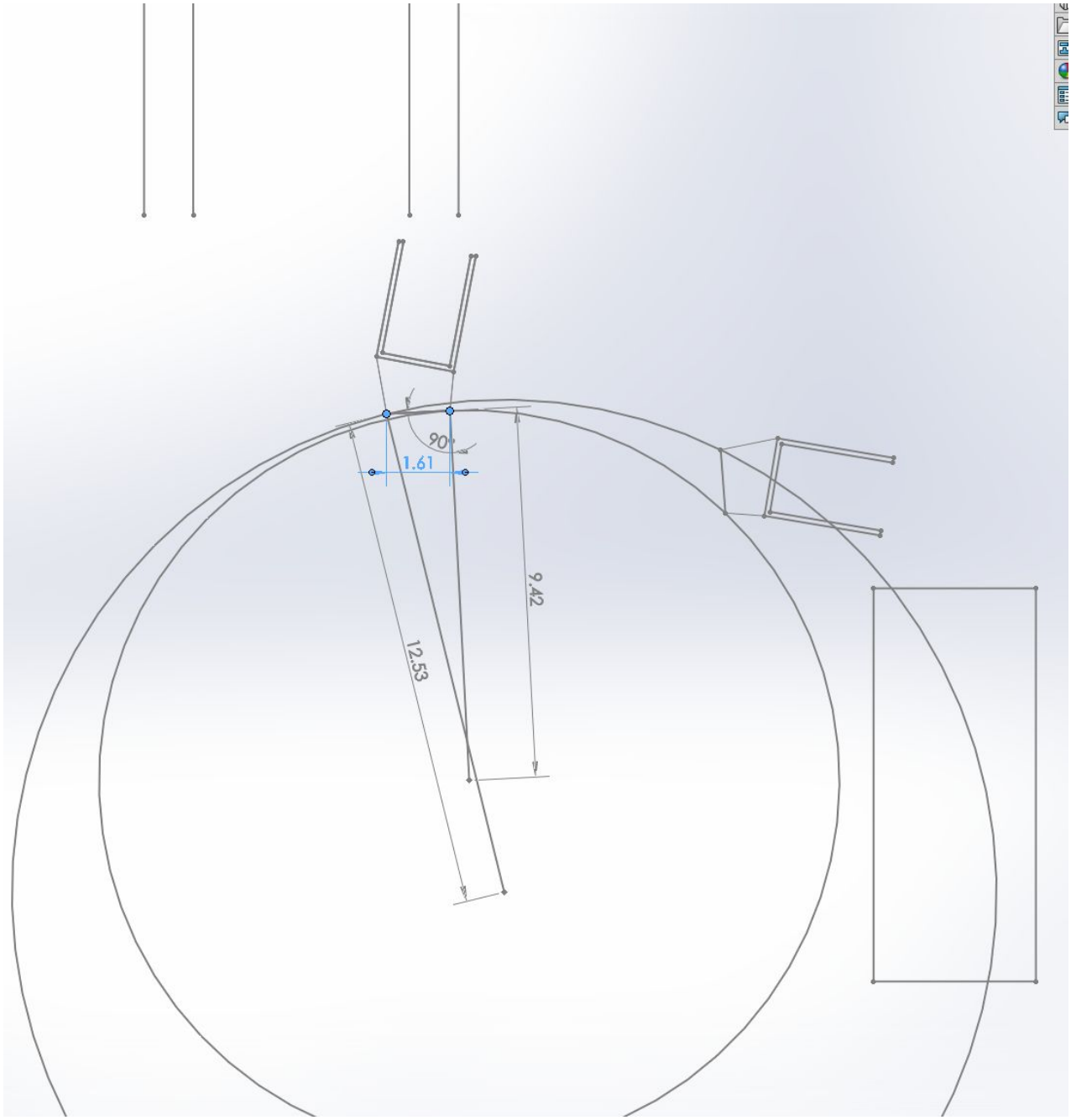


Figure A.2: Position 2 of Synthesis w/ Transmission Angle

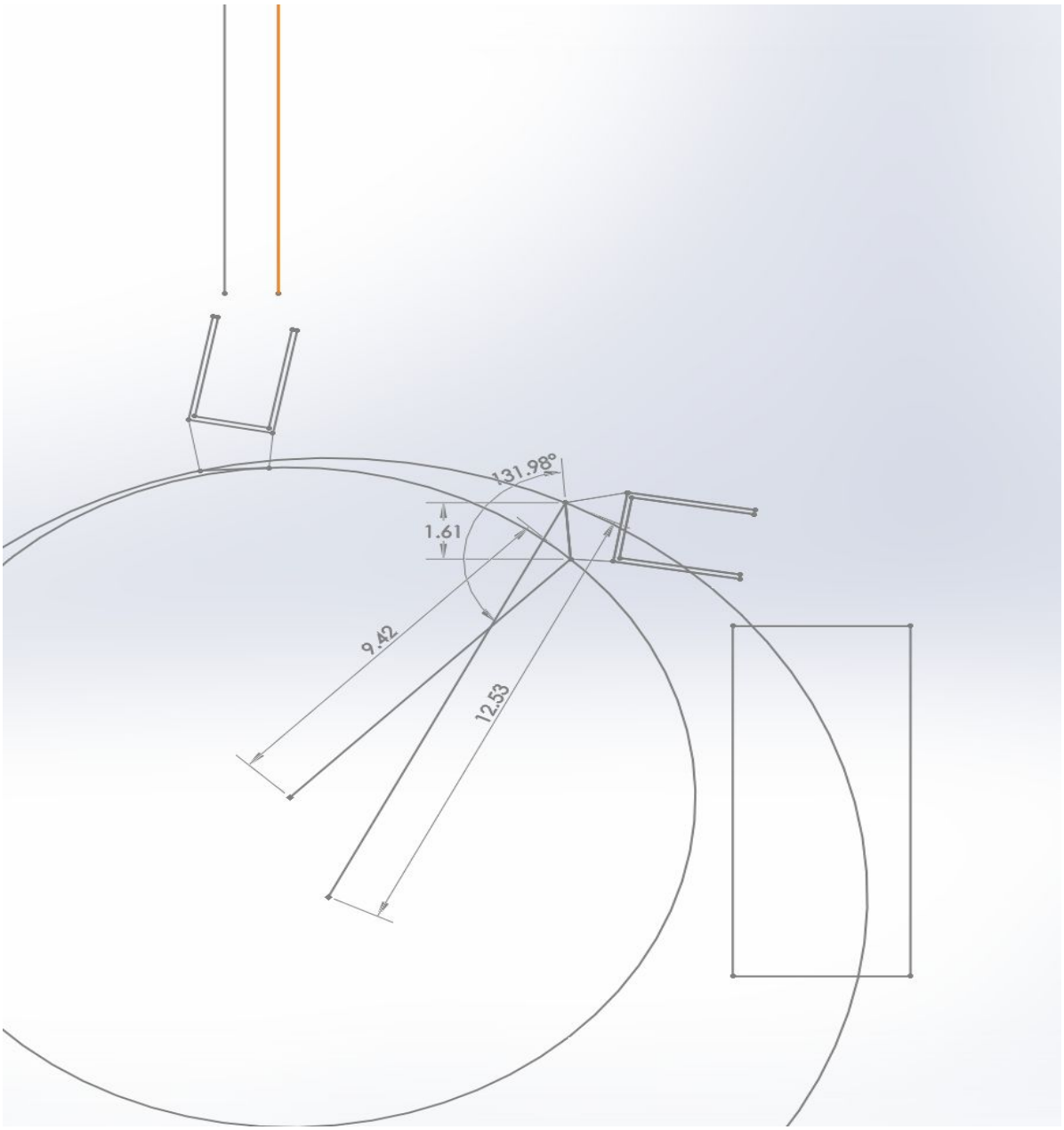


Figure A.3: Position 3 of Synthesis w/ Transmission Angle

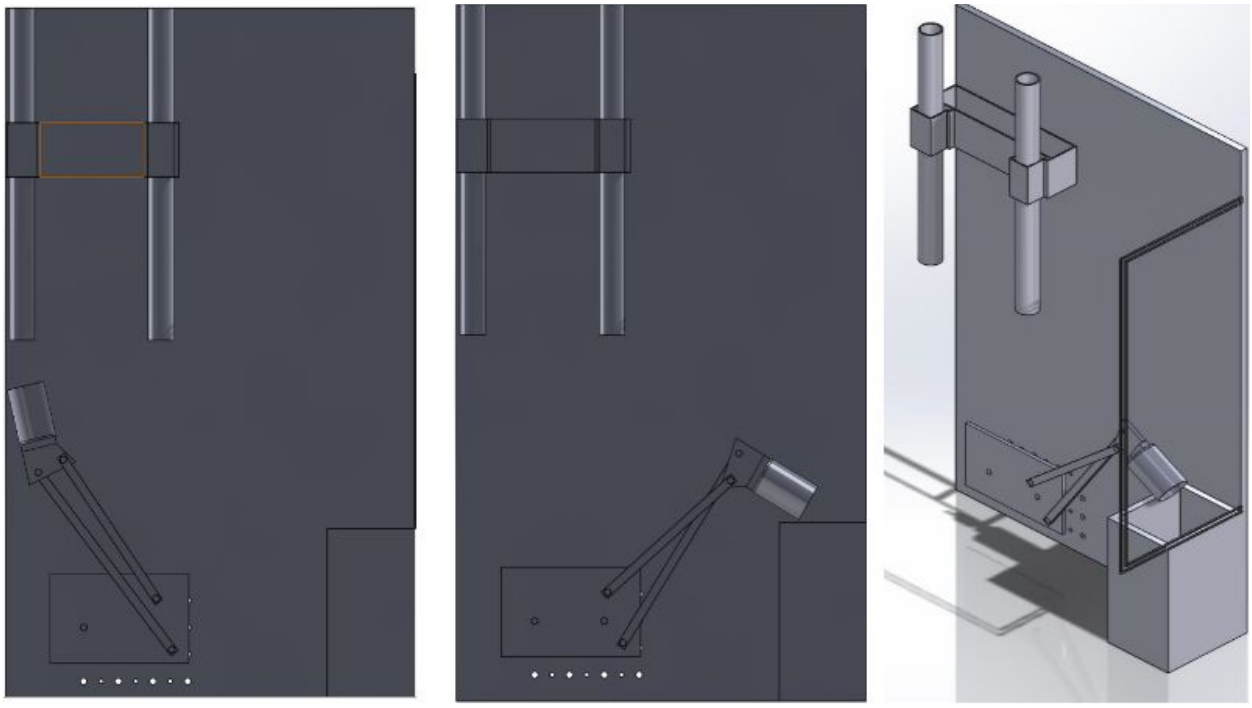


Figure A.4: Initial and End Position of 3D Solidworks and Isometric View

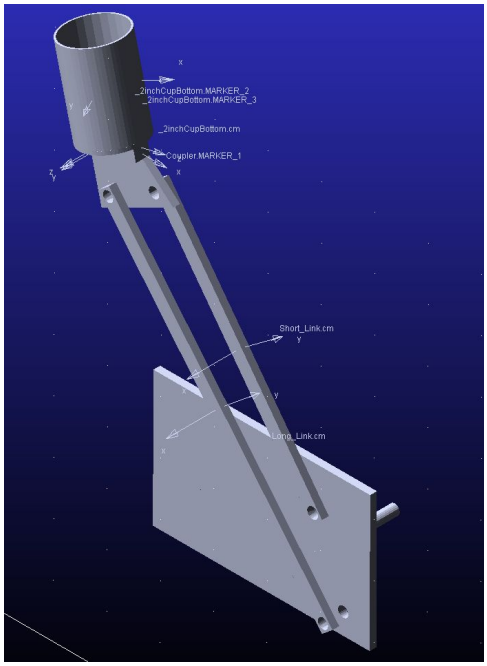


Figure A.5: Isometric View of ADAMS Model

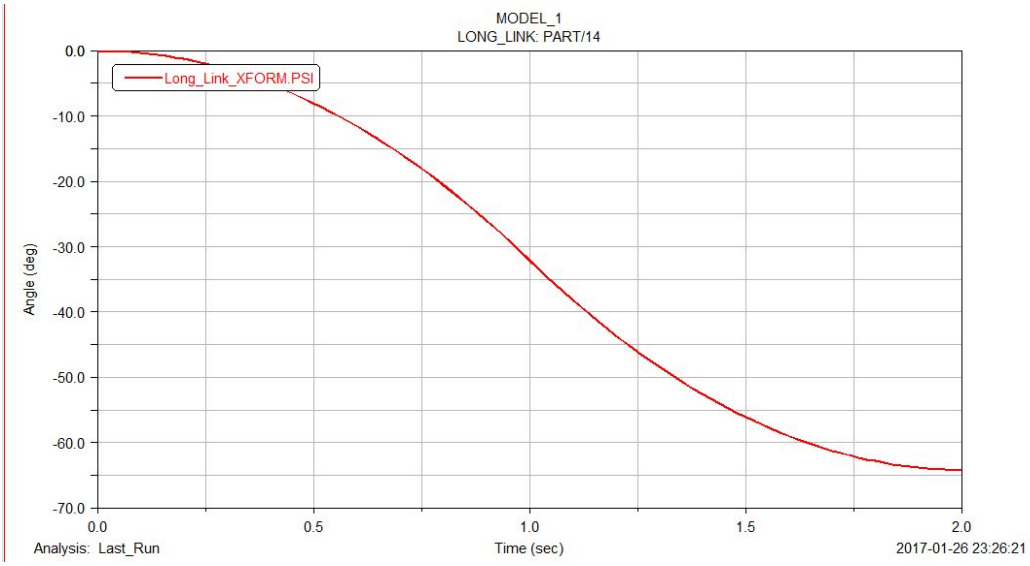


Figure A.6: Angular Displacement of ADAMS Simulation

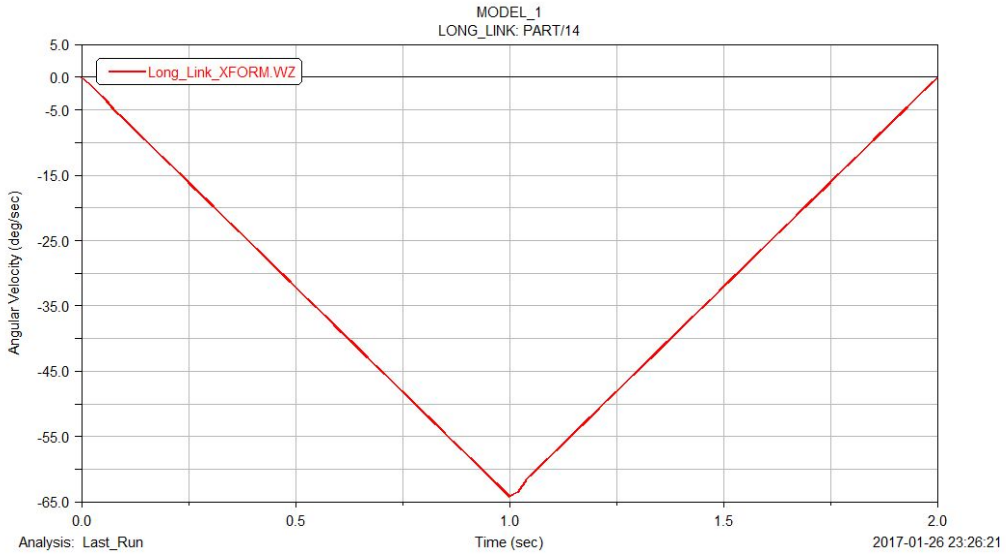


Figure A.7: Angular Velocity of ADAMS Simulation

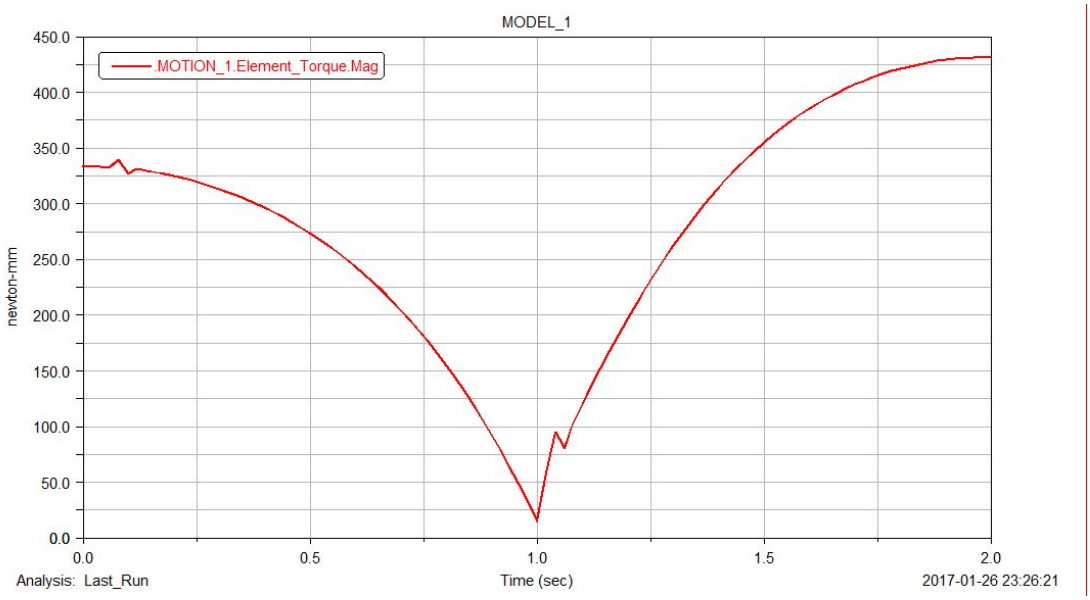


Figure A.8: Power Consumption of ADAMS Simulation

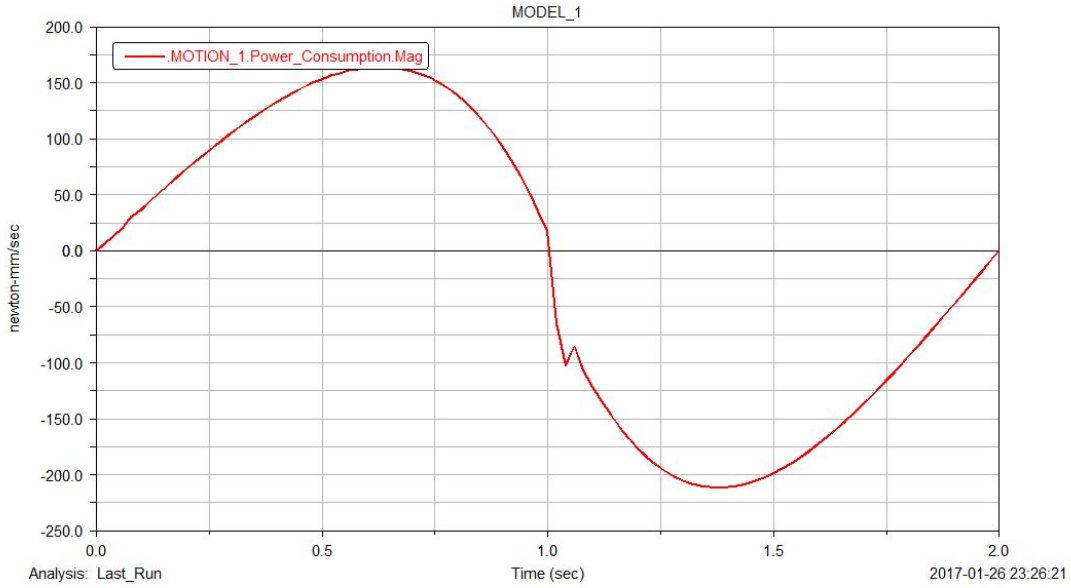


Figure A.9: Input Torque of ADAMS Simulation

Marcos Cavallin:

Using Solidworks and the rough dimensions of the playing field, I was able to create a sketch of the mechanism design. Using this sketch, I was able to easily tweak radii of the two circular paths and coupler length/angle to determine an optimal location for the cup in all 3 positions. This was done while also ensuring the transmission angle would not exceed its limits. I believed that the cup positioning was important when catching the ball, thus I made a point in my sketch to ensure that the cup at positions 1 and 2 were as parallel as possible to the chutes above (Figure A.3**)**. Because this was a 2D sketch, making changes to important values such as the link length and ground position was also quick and easy. This resulted in a bare bones design that could be used later in 3D modeling.

Once satisfied with an initial sketch, the values could then be used to design a full 3D solidworks model. This model could show the behavior of the design through its full range of motion. It also provided a more realistic perspective on how the design would catch and deposit balls into the bucket. Most importantly, this model could then be used in ADAMS to provide a rough simulation of the mechanism. This provided important data such as power consumption and angular velocity. After seeing these results, I realized the mechanism needed to be more lightweight in order to satisfy the energy requirements. The coupler was changed to a more lightweight material, and creating a more sleek design was under consideration for future use in the final design.

Table A.1: Marcos' Mechanical Synthesis Values

Link Lengths (Input, Coupler, Follower)	8.22 in	1.04 in	11.03 in
Transmission Angle (Position 1, 2, 3)	101.44°	42.80°	101.13°
Deviation (Position 1, 2, 3)	11.44°	-47.2°	11.3°

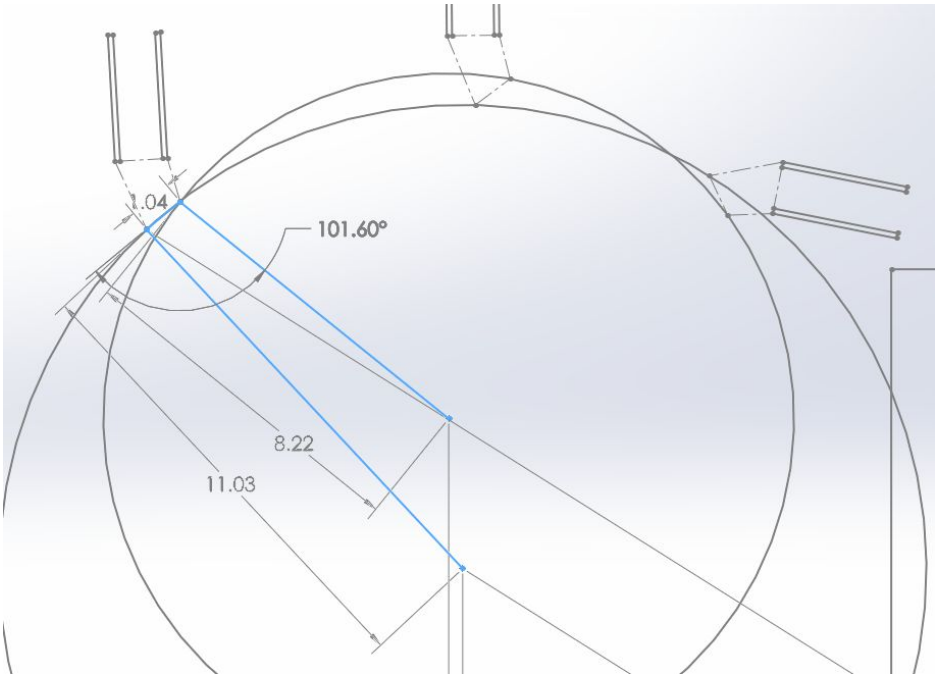


Figure A.1: Position 1 of Synthesis w/ Link lengths and Transmission Angle

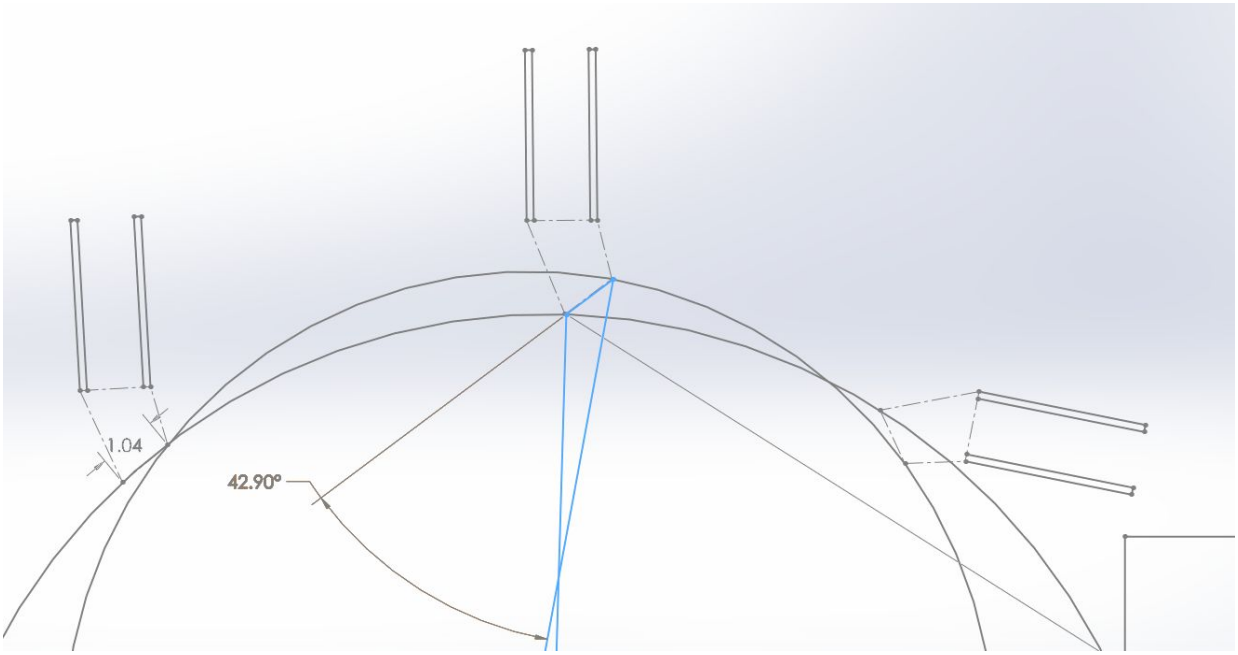


Figure A.2: Position 2 of Synthesis w/ Transmission Angle

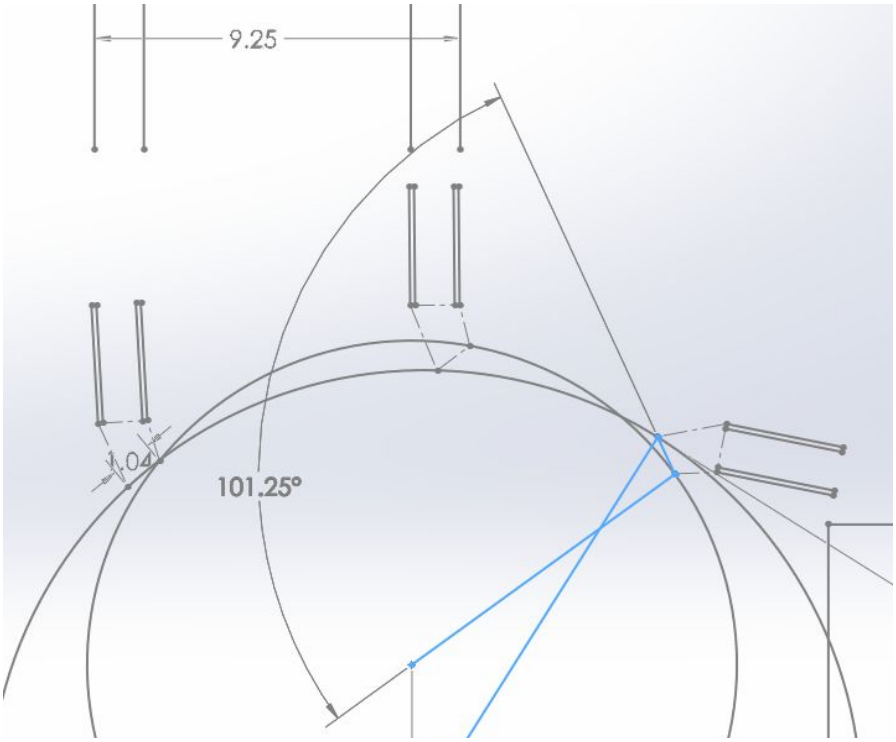


Figure A.3: Position 3 of Synthesis w/ Transmission Angle

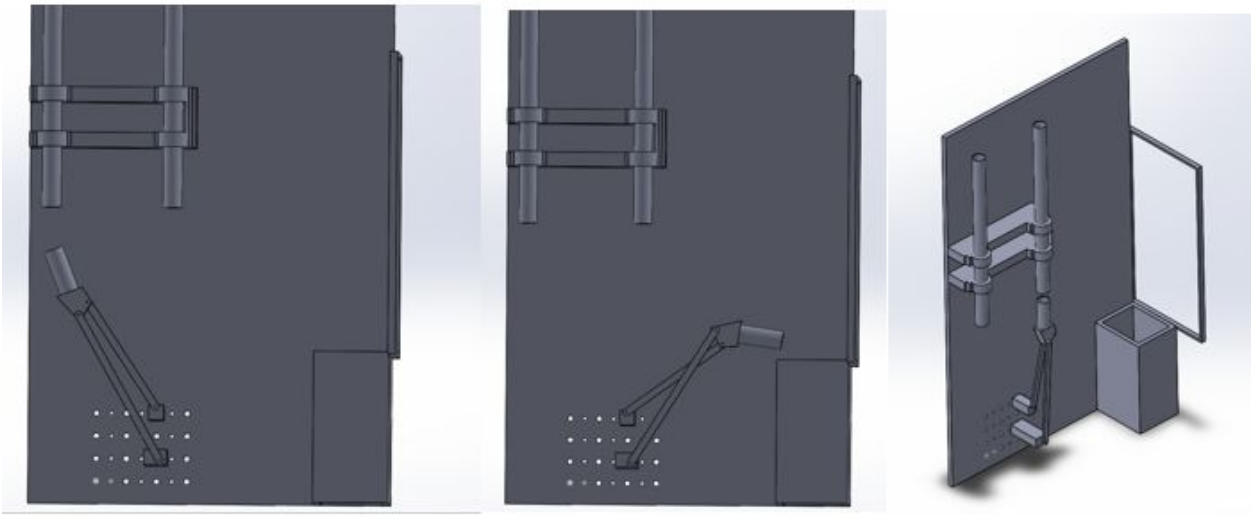


Figure A.4: Initial and End Positions of 3D Solidworks Model and Isometric View

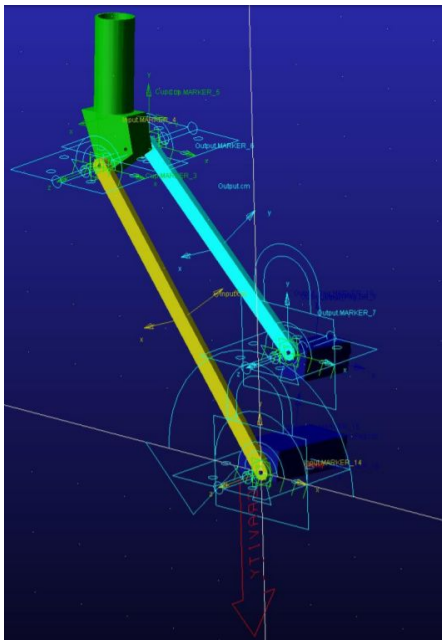


Figure A.5: Isometric View of ADAMS model

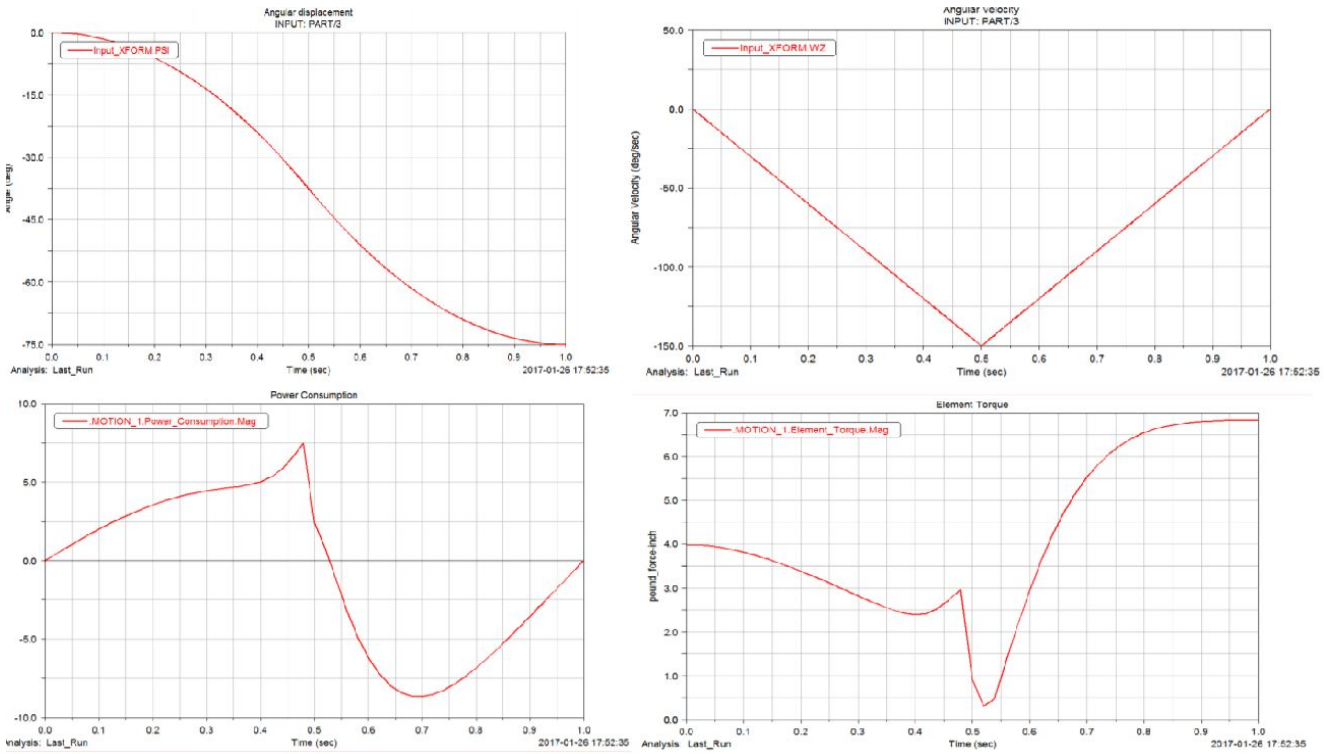


Figure A.6: ADAMS Simulation Graphs (Angular Displacement, Angular Velocity, Power, and Input Torque)

Mitchell Williams:

The focus of Lab 1 was centered around our individual mechanism synthesis using the graphical method of creating four-bar linkages. Once the playing field was represented in a two dimensional model, the objective was to create a coupler that aligned well with the chutes that would drop the multicolor balls in the competition. To do this, a two dimensional figure of the 1-inch cup was aligned with the chutes and a perimeter circle was drawn between all A linkage points and a separate circle was drawn between all B linkage points. Radii were then extended from the circles to the joints and represented the ground pivots of the mechanism. This simple but effective design could then be made into three-dimensional linkages with aluminum material properties. Once the three dimensional entities were finished, ADAMS simulations were run to ensure the torque required to accelerate the mechanism was sufficient and that the power usage at a given time was within boundaries.

The two dimensional Solidworks portion made the graphical method extremely simple without having to use a compass and ruler. Furthermore, it facilitated the structure behind the three-dimensional model that was used in Solidworks. Overall, the design was much more easily made using the applications available. The most useful of the programs was probably ADAMS. This took care of maximum torque and power calculations without needing to build and test the model or use a series of complicated equations to approximate the motion of the four-bar linkage system.

Table A.1: Mitchell's Mechanical Synthesis Values

Link Lengths(Input, Coupler, Follower)	8.68"	.74"	9.42"
Transmission Angles(Position 1, 2, 3)	87.83°	105.86°	47.61°
Transmission Angle Deviation(Position 1,2, 3)	2.17°	15.86°	42.39°

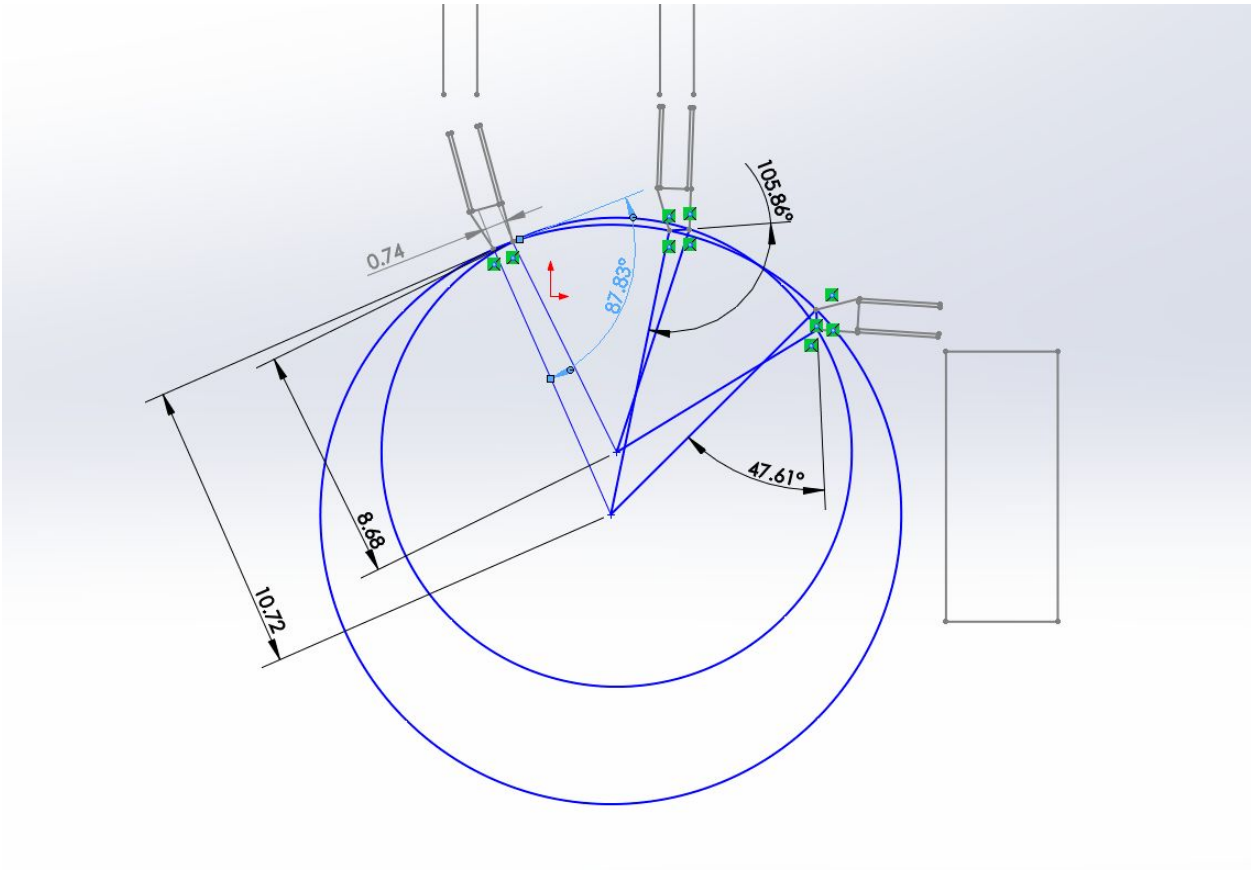


Figure A.1: All 3 Cup Positions w/ Transmission Angle and Link Lengths

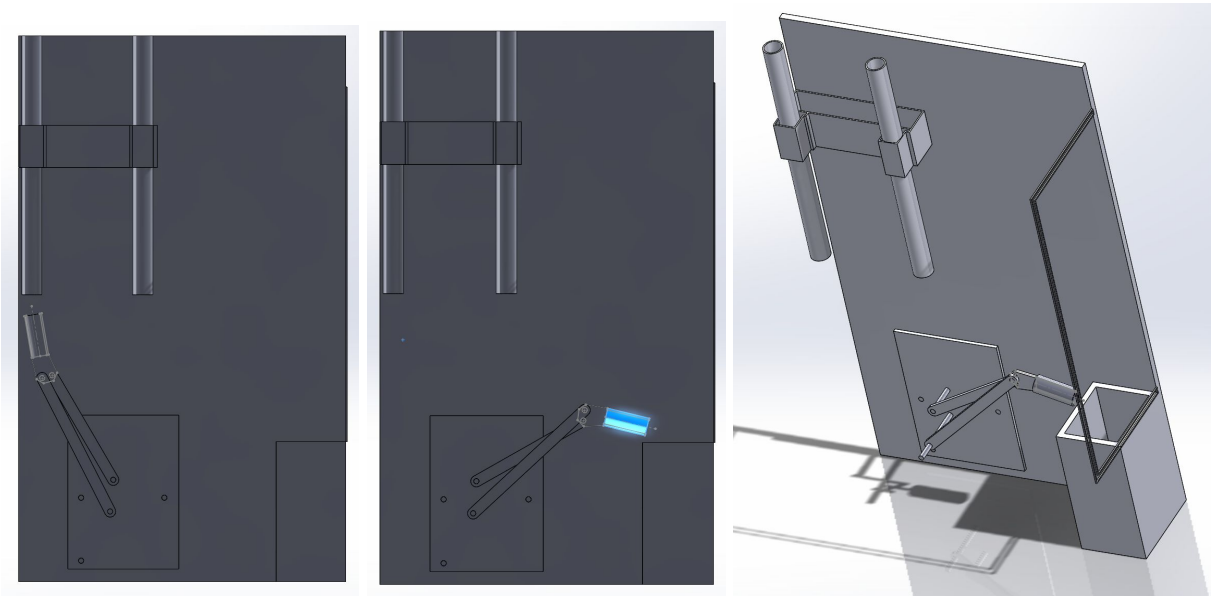


Figure A.2: Initial and End Positions of 3D Solidworks Model and Isometric View

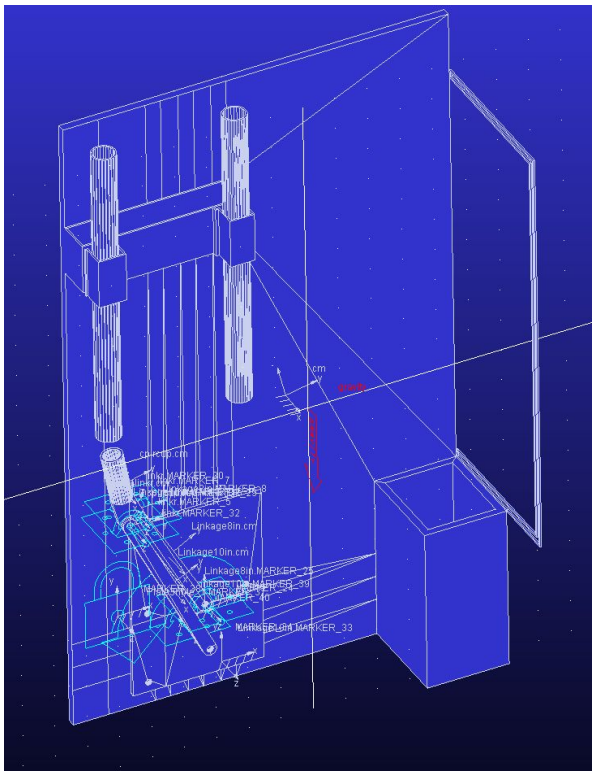


Figure A.3: Isometric View of ADAMS

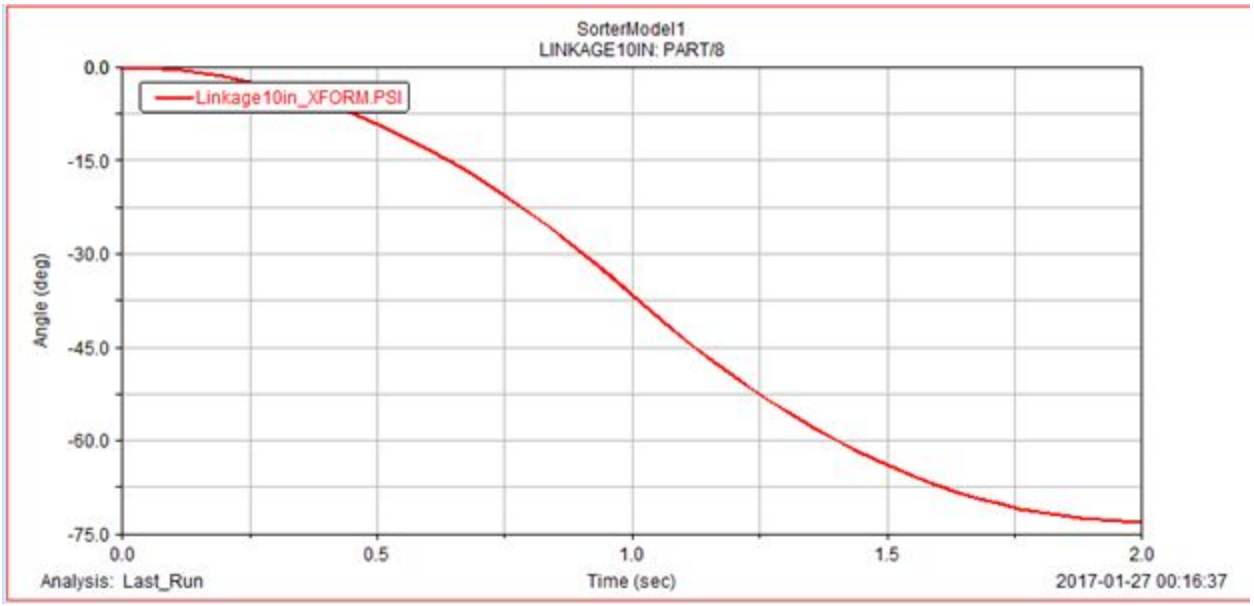


Figure A.4: Angular Displacement of ADAMS Simulation

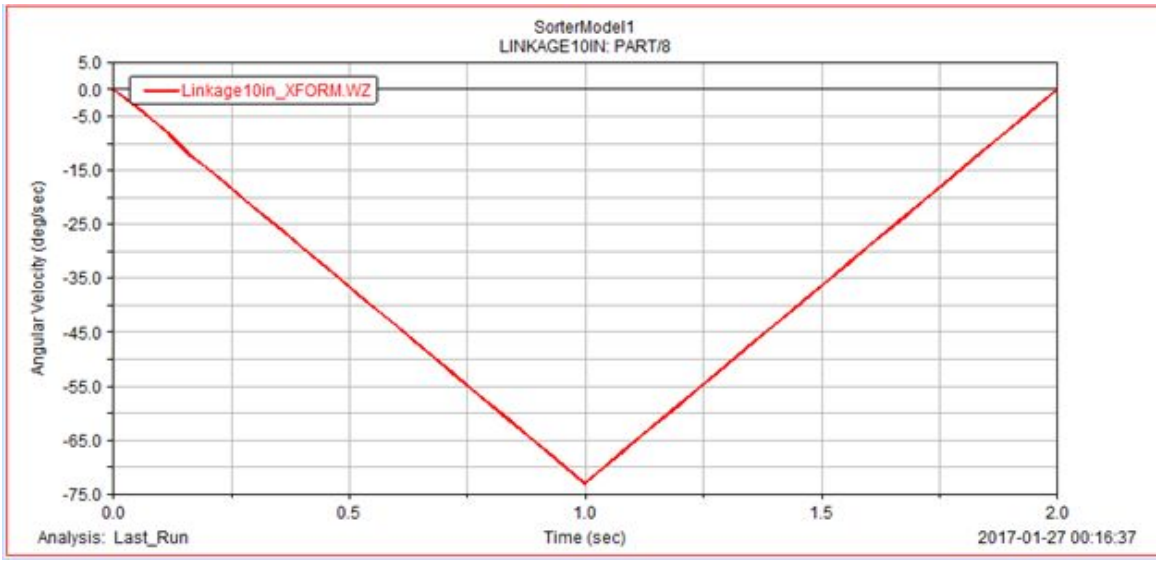


Figure A.5: Angular Velocity of ADAMS Simulation

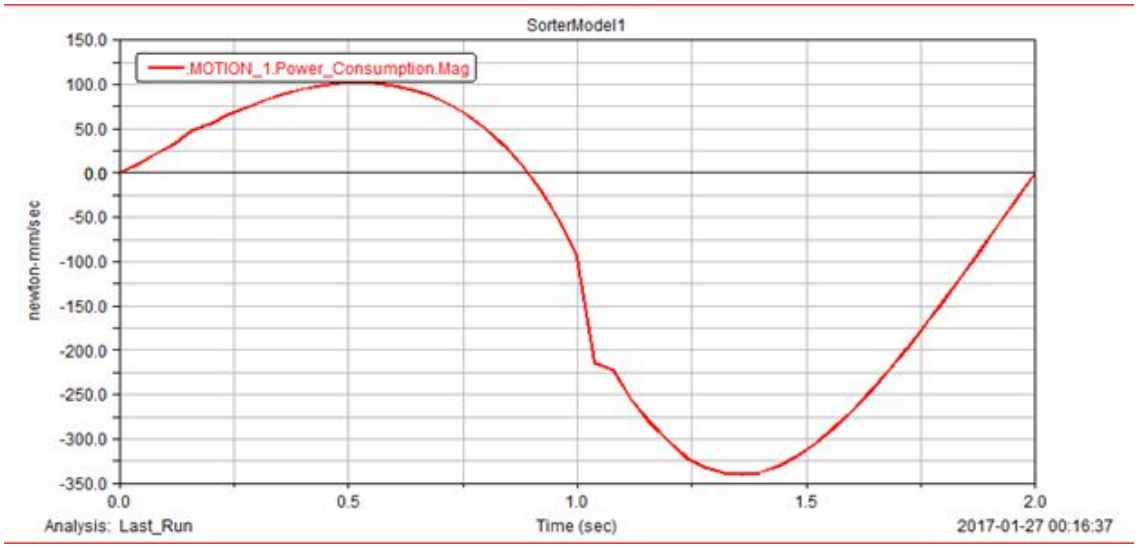


Figure A.6: Power Consumption of ADAMS Simulation

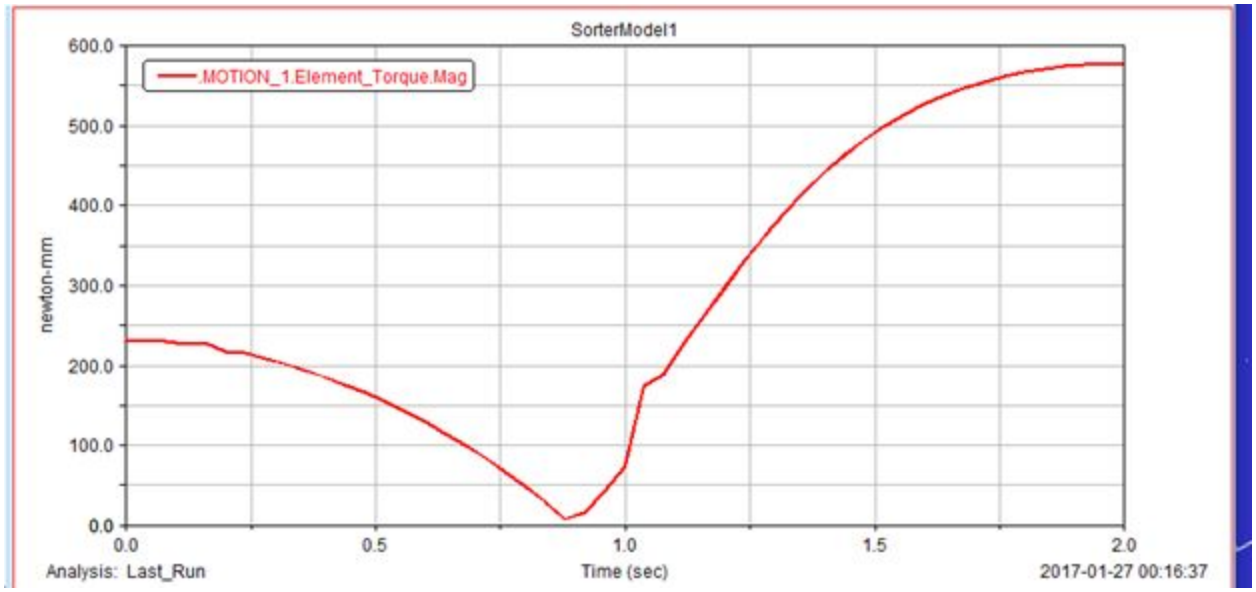


Figure A.7: Input Torque of ADAMS Simulation

David Van Dyke

The purpose of mechanism synthesis was to create a mechanism that could move between three positions while having mounting points that were in a reasonable location on the board. A field based on the actual field was created in a SolidWorks sketch and a sketch block was created to represent the cup. This allowed us to estimate ideal locations for the cup to be to catch and move the marbles. We could also use the transmission angles to estimate the efficiency of our system, as a lower transmission angle deviation will result in a more powerful system. I used the lengths of each link as well as initial positions in order to create a more sophisticated SolidWorks model. This allowed us to better visualize the mechanism in the playing field. We also used this model to create an ADAMS simulation to estimate power usage.

We exported the model we made in SolidWorks into ADAMS so we could estimate the power it would take to run our mechanism in a set amount of time. This is important since we need to ensure that our mechanism will be able to meet the minimum requirements and move. We do not want to choose a linkage design that requires a power level we cannot provide. We also used the power requirements of each design to help us finalize our design and find the best option. In order to find power, we put motion on the input link and ran the mechanism such that it moved over its complete range of motion in 1 second. We recorded angular displacement, velocity, power output., and input torque of the mechanism.

Table A.1: David's Mechanical Synthesis Values

Link Lengths (Input, Coupler, Follower)	12.65 in	1.25 in	9.45 in
Transmission Angle (Position 1, 2, 3)	65.59°	58.44°	124.27°
Deviation (Position 1, 2, 3)	-24.41°	-34.03°	34.27°

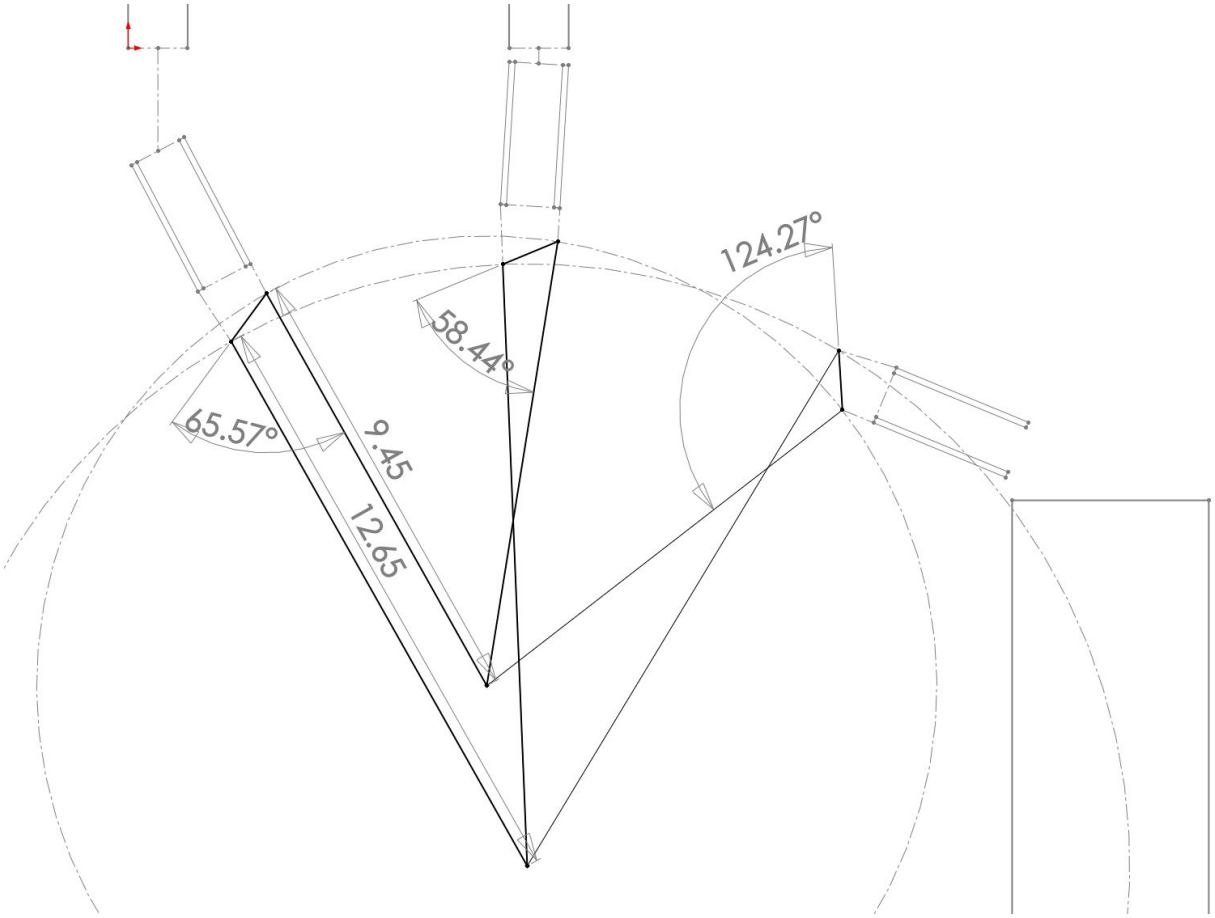


Figure A.1: David's Mechanical Synthesis model

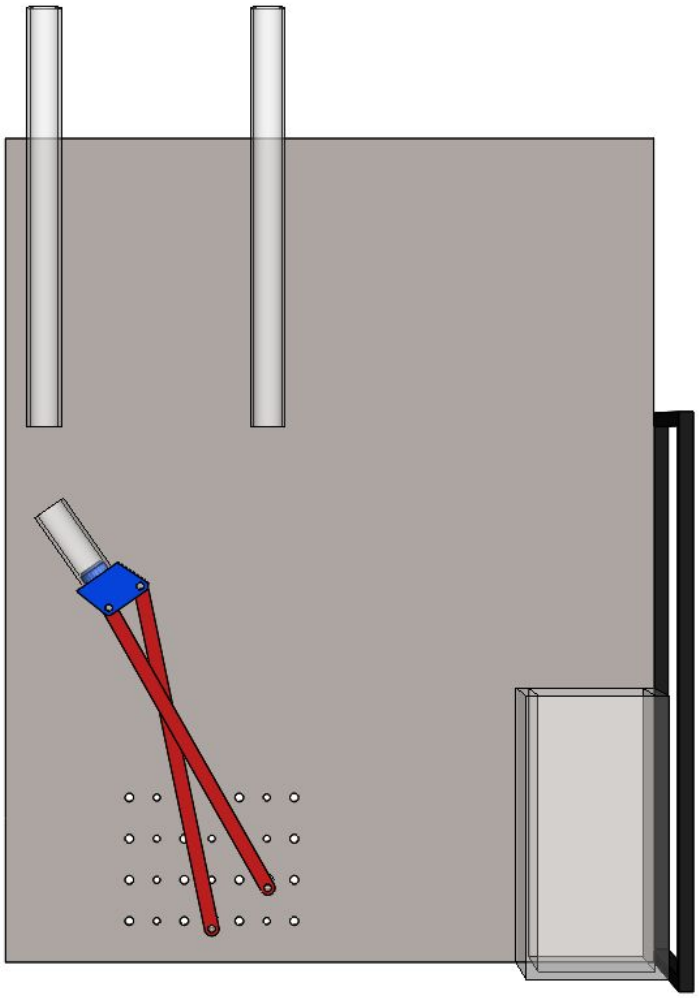


Figure A.2: Initial positions of David's 3D Solidworks Model

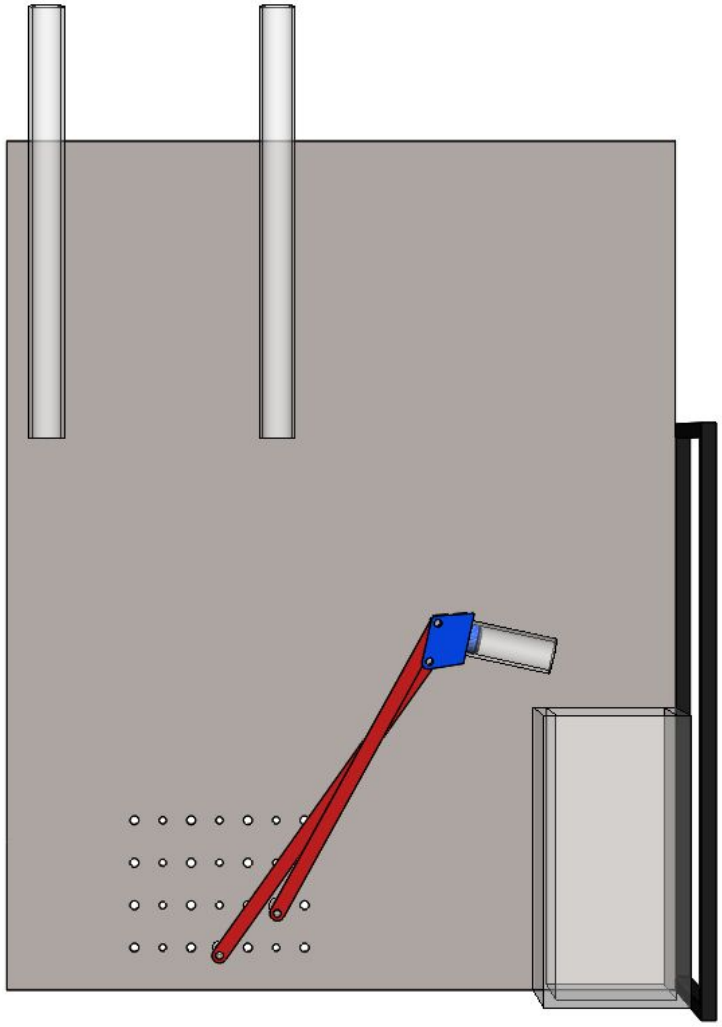


Figure A.3: Final Position of David's SolidWorks Model

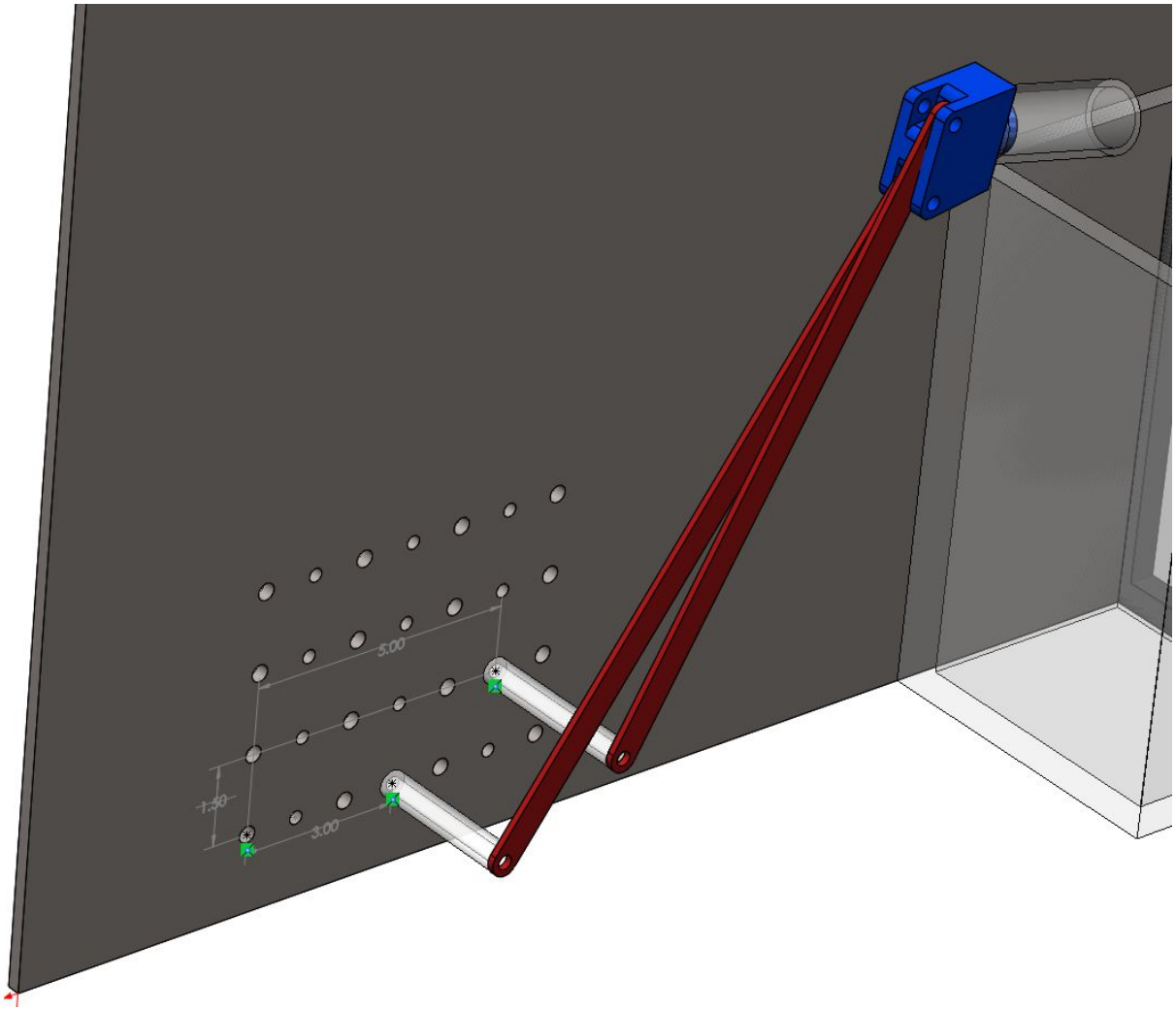


Figure A.4: Isometric View of David's Mechanism on the Playing Field

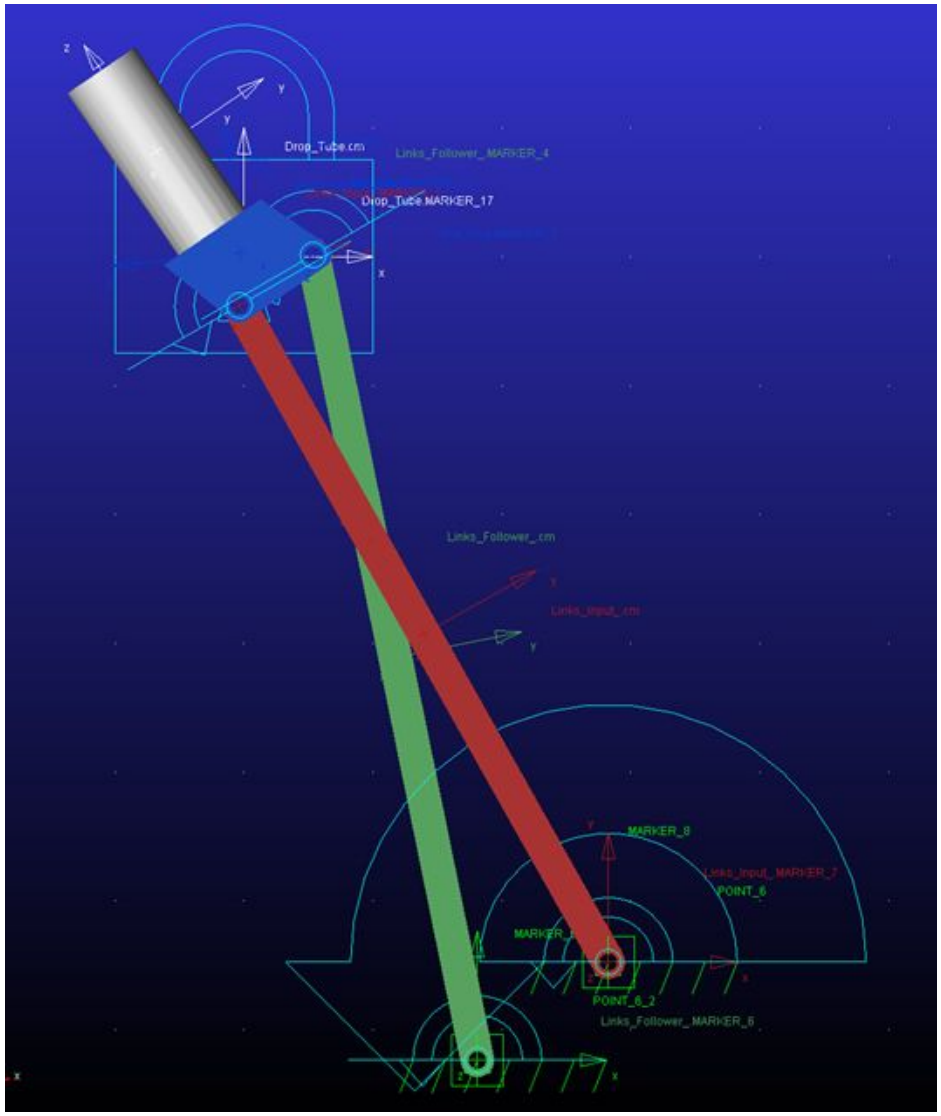


Figure A.5: David's Model in ADAMS

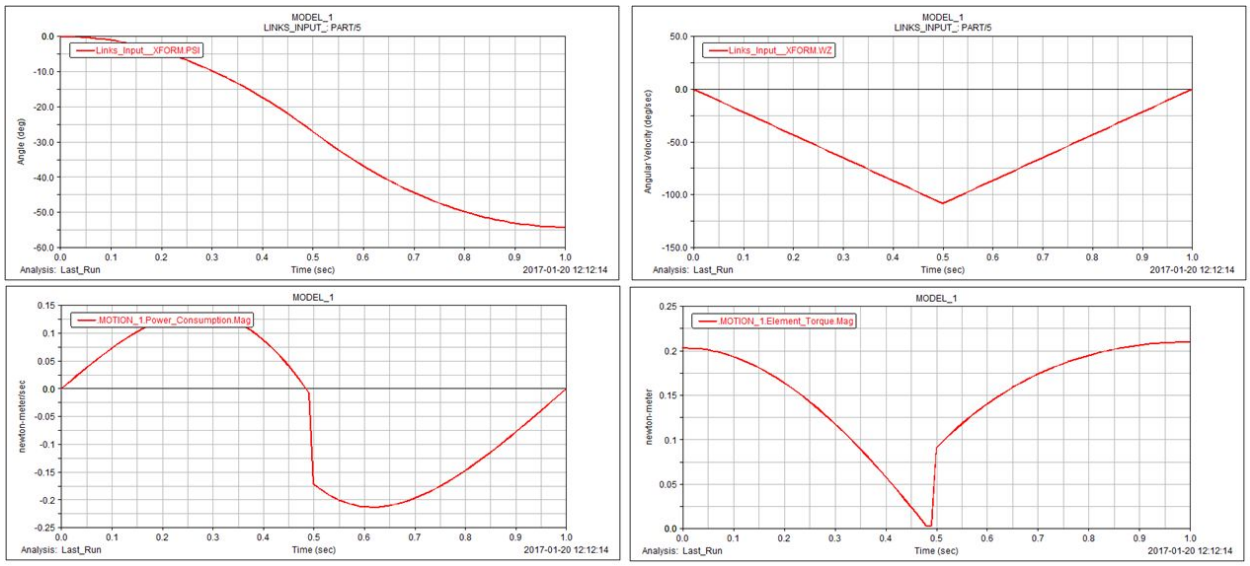


Figure A.6: David's ADAMS Simulation Graphs (Angular Displacement, Angular Velocity, Power, and Input Torque)

Nikko Van Crey

Each individual design should be introduced with a two-paragraph summary of the main ideas from (1) mechanism synthesis, i.e. Sketch Blocks (2) Solidworks, and (3) ADAMS.

The lab was structured in a way that allowed the linkage designs to be easily and quickly modified. The first step in designing my individual linkage was to use the graphical method of four bar linkage design. This method first required the playing field for the competition to be converted to a 2D sketch that would simply serve to visually choose favorable cup positions at different points in the linkages motion. Two different circles were made by forming arcs with the 3 different locations for each of the coupler pivots. The centers of these circles could then serve as the locations for each end of the ground pivot. This method allowed us to ensure that these centers were at a reasonable location for mounting and that the transmissions angles for the linkage were all within the 30-150 degree allowable range before even starting to take the design to the third dimension.

Now that the 2D mechanism synthesis is complete, the links can be given any shape and depth that I want as long as the pivots remain the same distance relative to one another as they were in the 2D analysis. All of the critical design work was done in the 2D analysis. After completing the 3D design of the mechanism and verifying that the assembly in Solidworks still hits the target cup locations across the linkage's motion, the last step was to test feasibility through an ADAMS simulation that would give data for angular displacement, velocity, power output, and input torque. This data is critical because we need to verify that we can physically move the mechanism with the motors and materials provided in the desired amount of time. If the mechanism has an unreasonable power output this would tell us that some pocket, lightweighting, or mechanism redesign is necessary.

Table A.1: Nikko's Mechanical Synthesis Values

Link Lengths (Input, Coupler, Follower)	9.69 in	1.45 in	6.93 in
Transmission Angle (Position 1, 2, 3)	103.98°	60.88°	109.4°
Deviation (Position 1, 2, 3)	13.98°	29.12°	19.4°

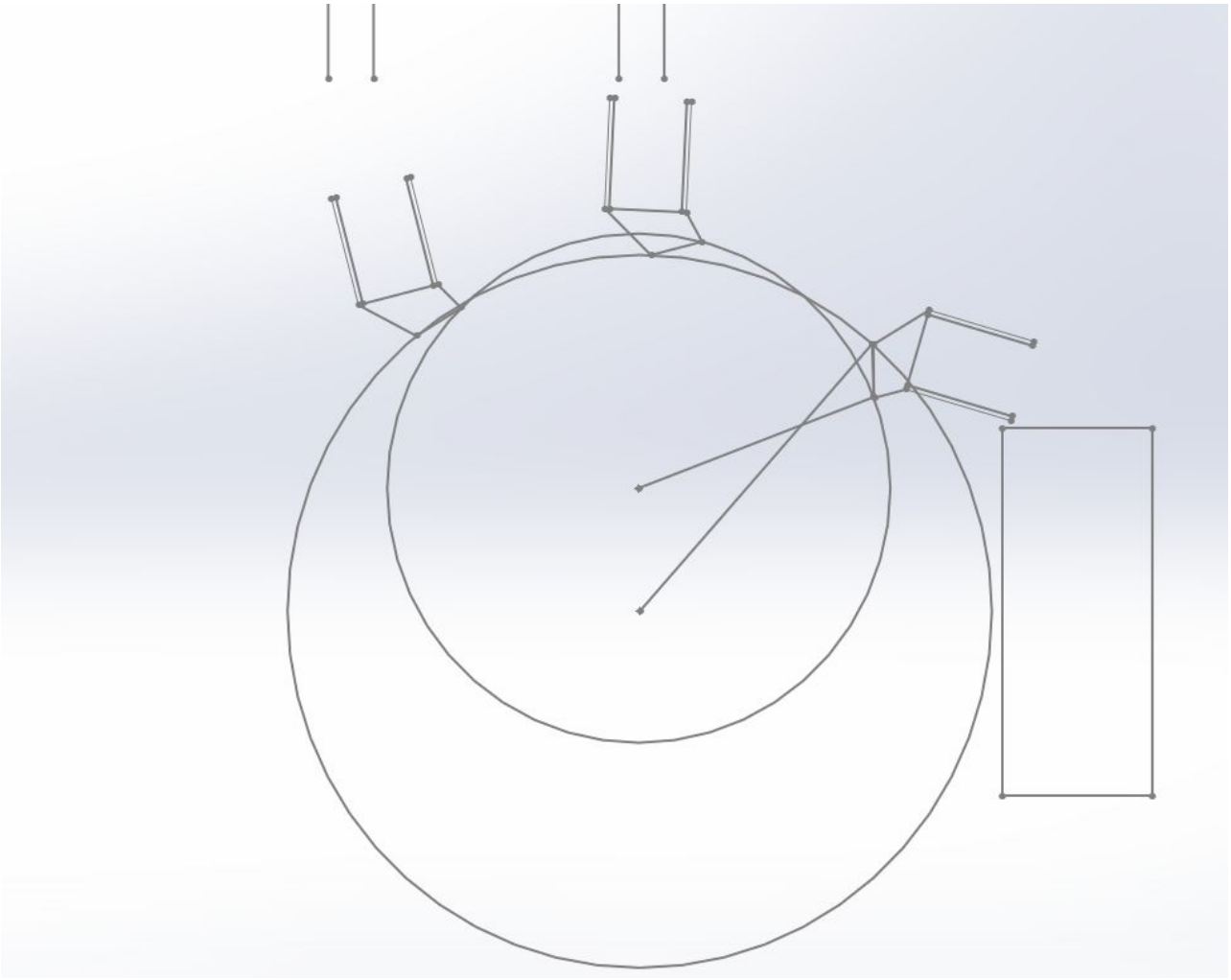


Figure A.1: Nikko's Mechanical Synthesis model

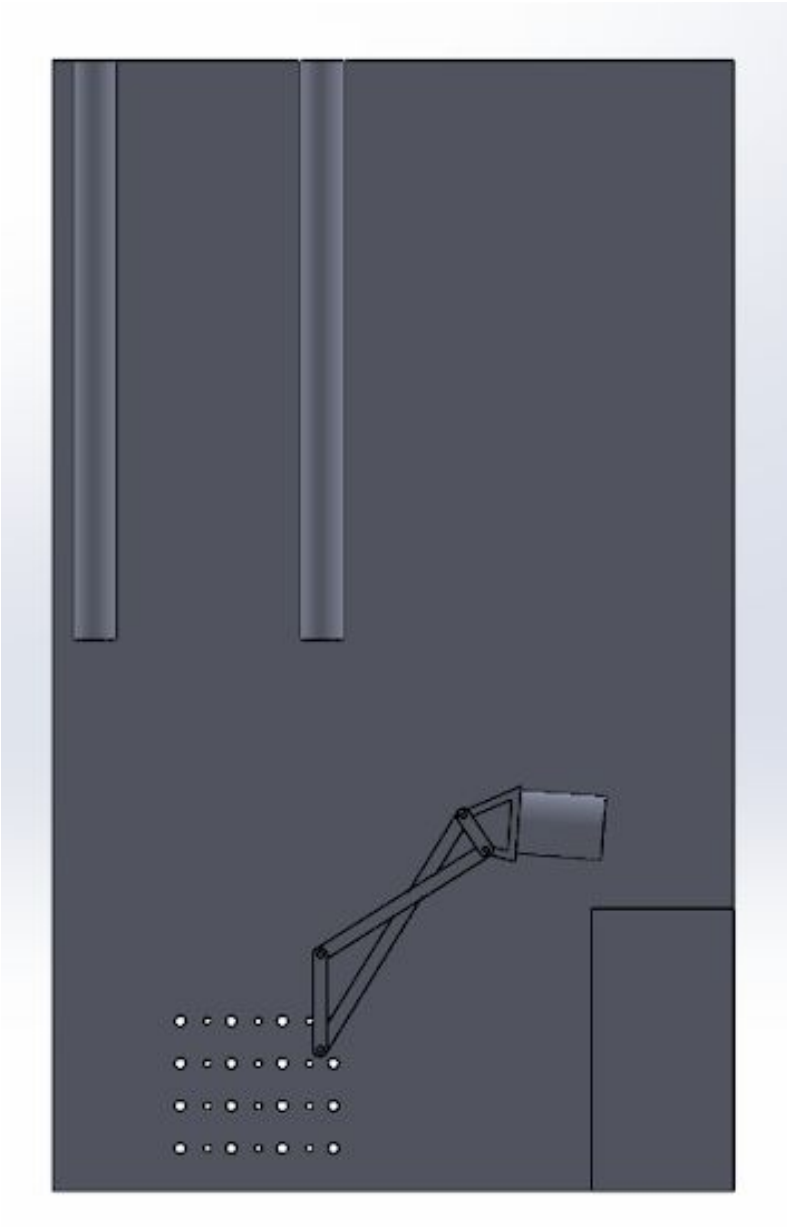


Figure A.2: Initial positions of Nikko's 3D Solidworks Model

Figure A.3: Final Position of Nikko's SolidWorks Model

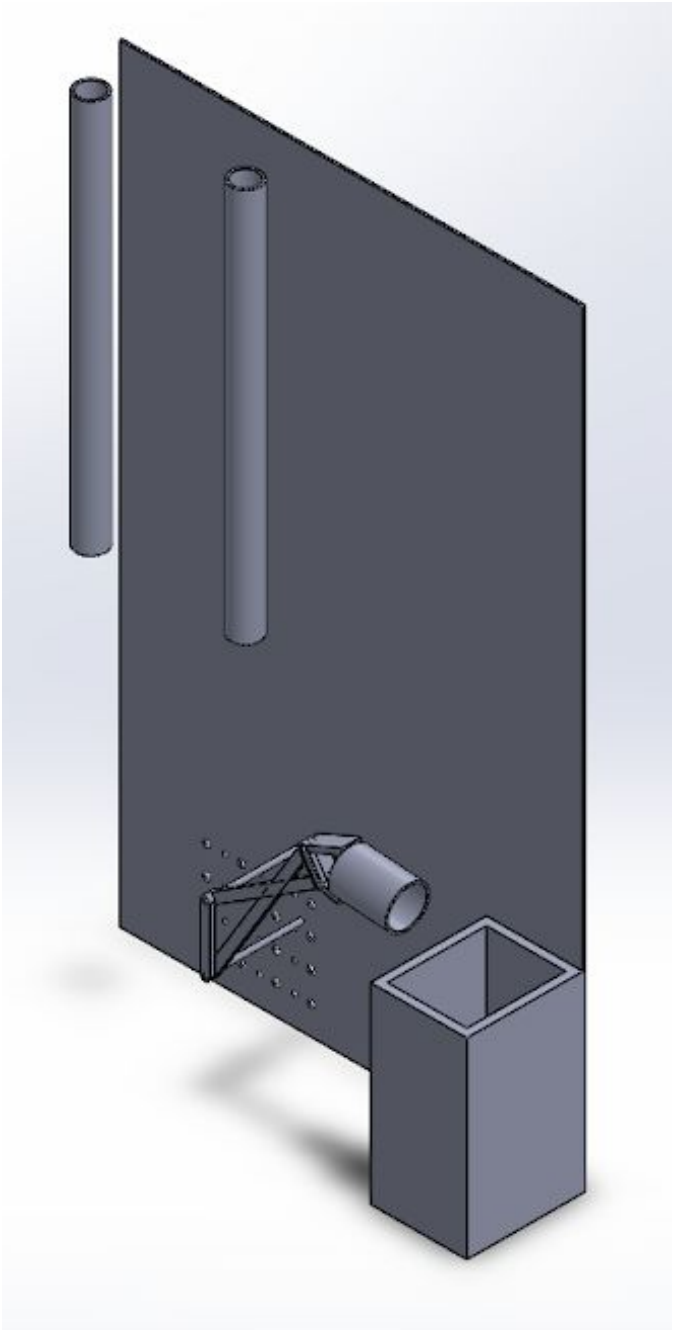


Figure A.4: Isometric View of Nikko's Mechanism on the Playing Field

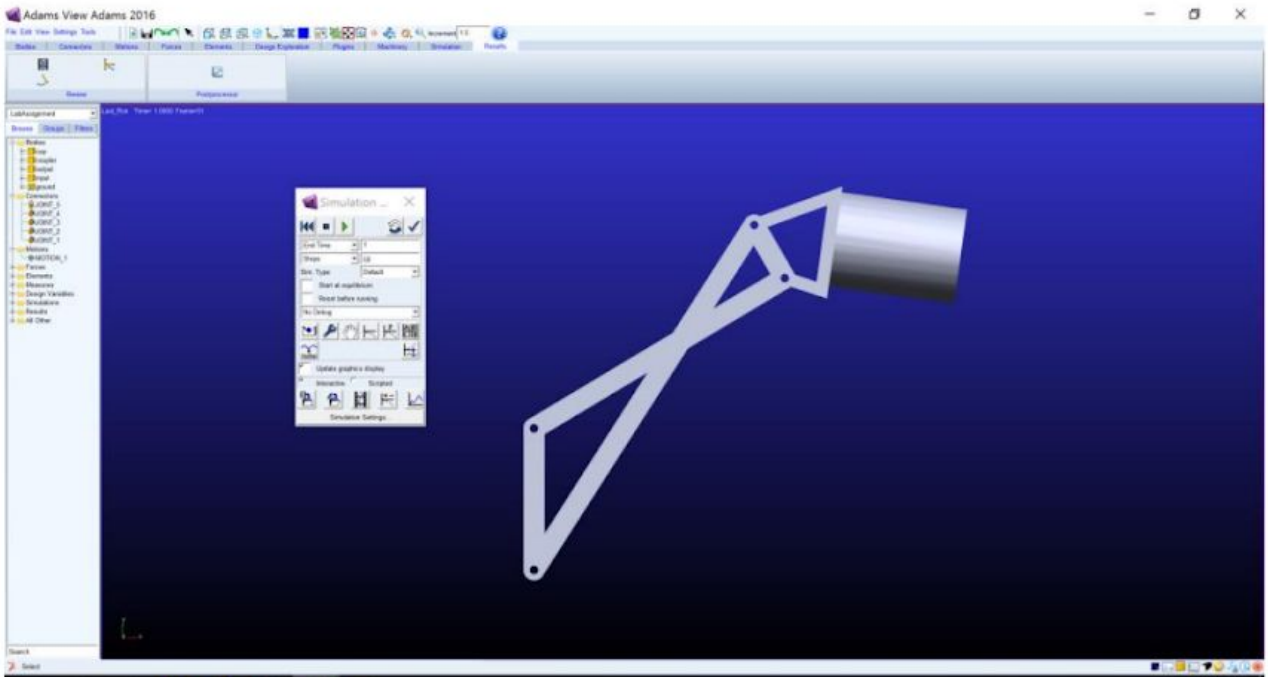


Figure A.5: Nikko's Model in ADAMS

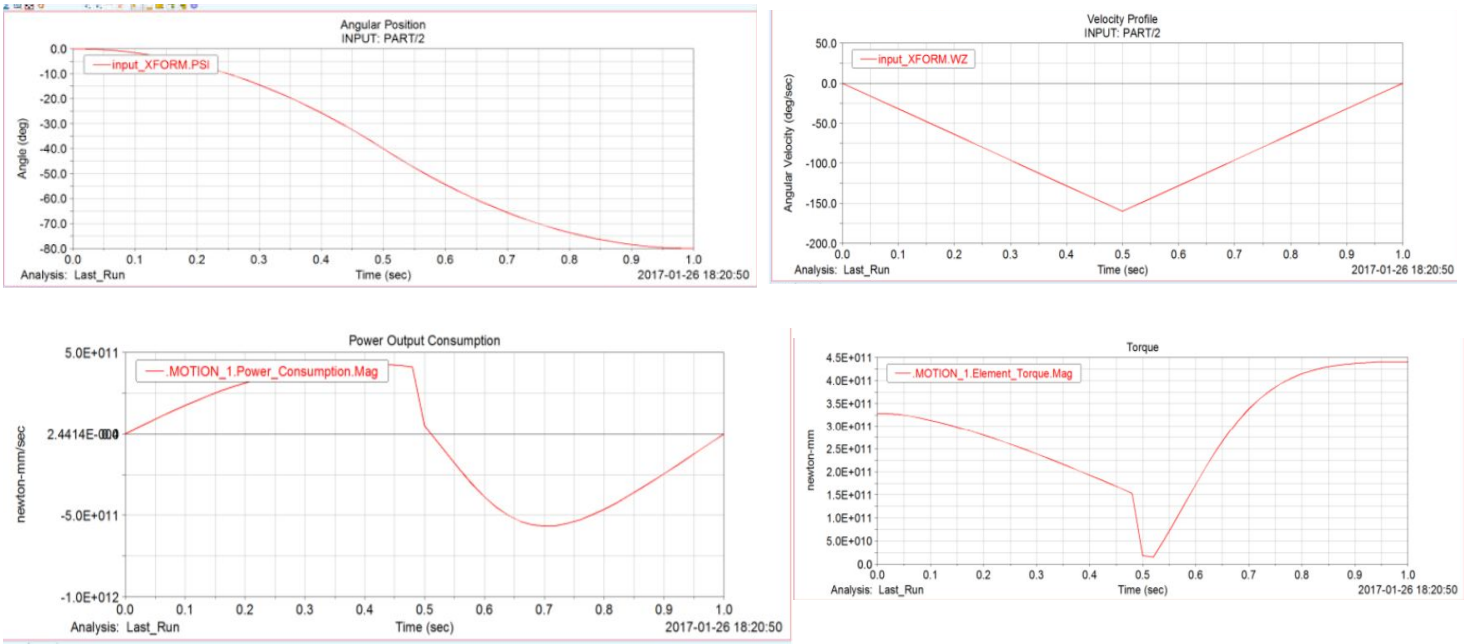
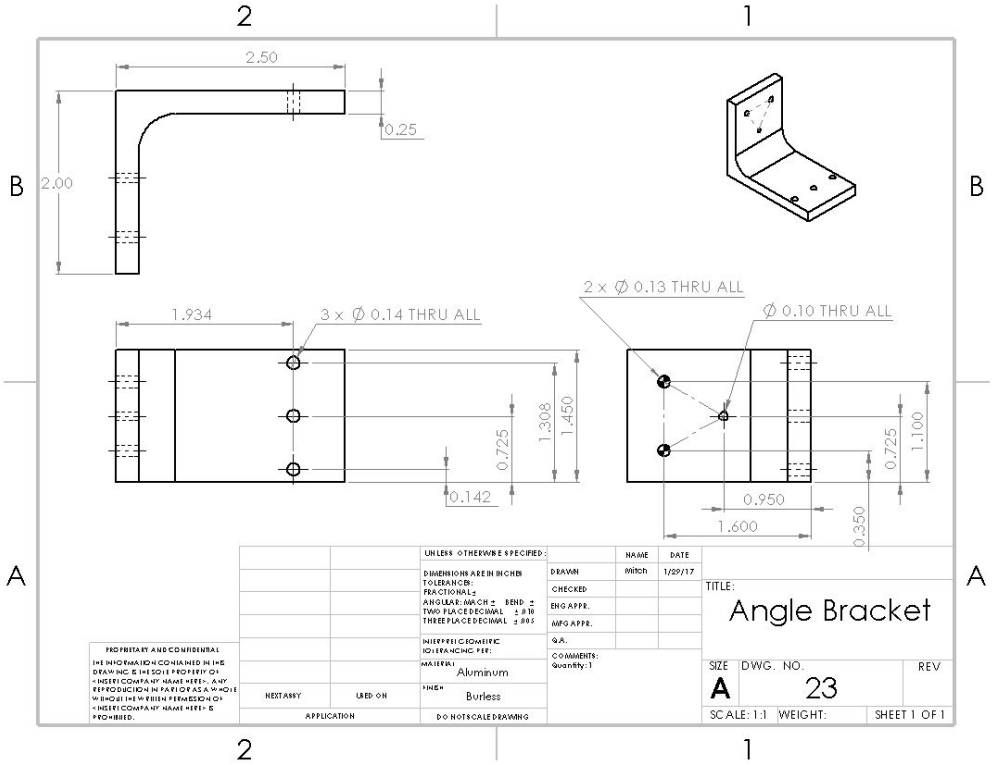


Figure A.6: Nikko's ADAMS Simulation Graphs (Angular Displacement, Angular Velocity, Power, and Input Torque)

Appendix B: Drawings, Manufacturing Plans, Bill of Materials, and Assembly Plan for Final Design

Drawings and Manufacturing Plans:

Angle Bracket



Part Number: ME350-001 Revision Date: 31/1/2017

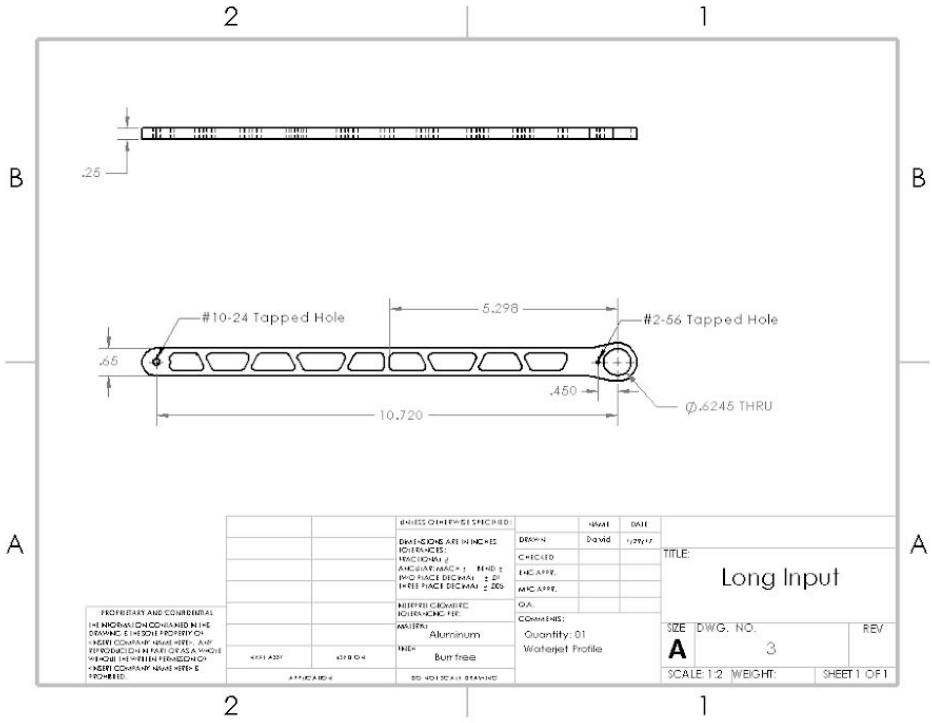
Part Name: Angle Bracket

Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum Angle, 2.5" x 2" x 1/4" thick, 6" long

Step #	*Process Description*	*Machine*	*Fixtures*	*Tool(s)*	*Speed (RPM)*
1	Cut 1.45" aluminum angle stock >.125 of finish length and deburr.	Band Saw		File	300 ft/min
2	Hold part in vise on top of parallels with >.125" material sticking out and 2.5" side flat.	Mill	vise	1.375 parallels	
3	Clamp one of the x dimensioin stoppers on the vice	Mill	vise	X dimension Stopper	
4	Install edge finder into drill chuck.			Drill chuck, edgefinder	
5	Find datum lines for X and Y. (no need to find datum again if part is repositioned because you can zero y to the vice and x to the stopper	Mill	vise	Edge finder, drill chuck	1000
6	Remove edge finder and install cutter and collet	Mill	vise	Drill chuck, edgefinder, 3/4 inch 2-flute endmill, collet	
7	Mill the band saw cut sides with a conventional cut in 10 thou increments until 5 thou above the desired 1.45" length	Mill	Vise	3/4 inch 2-flute endmill, collet	840
8	Mill a 5 thou finish pass slowly with a climb cut	Mill	Vise	3/4 inch 2-flute endmill, collet	840
9	Remove cutter and collet and install drill chuck	Mill	vise	Drill chuck, 3/4 inch 2-flute endmill, collet	
10	Center drill and drill the 3X0.14" holes	Mill	Vise	#3 Center drill, #6 drill bit, drill chuck, and cutting fluid	1500
11	Remove part from vise deburr all			File, deburring tool	
12	Reposion the part in the vise on top of parallels with the 2" side flat	Mill	vise	1.375 parallels	
13	Center drill and drill the 2X0.13" holes	Mill	Vise	#3 Center drill, #5 drill bit, drill chuck, and cutting fluid	1500
14	Center drill and drill the 1X0.10" hole	Mill	Vise	#3 Center drill, #3 drill bit, drill chuck, and cutting fluid	1500
15	Remove part from vise deburr all			File, deburring tool	

Long Input


2
1
B
A
.25
#10-24 Tapped Hole
5.298
#2-56 Tapped Hole
.65
.450
Ø.6245 THRU
10.720
Aluminum
Burr free
Quantity: 01
Waterjet Profile
Long Input
A
3
SCALE: 1:2
SHEET 1 OF 1

Part Number: ME350-002 Revision Date: 31/1/2017

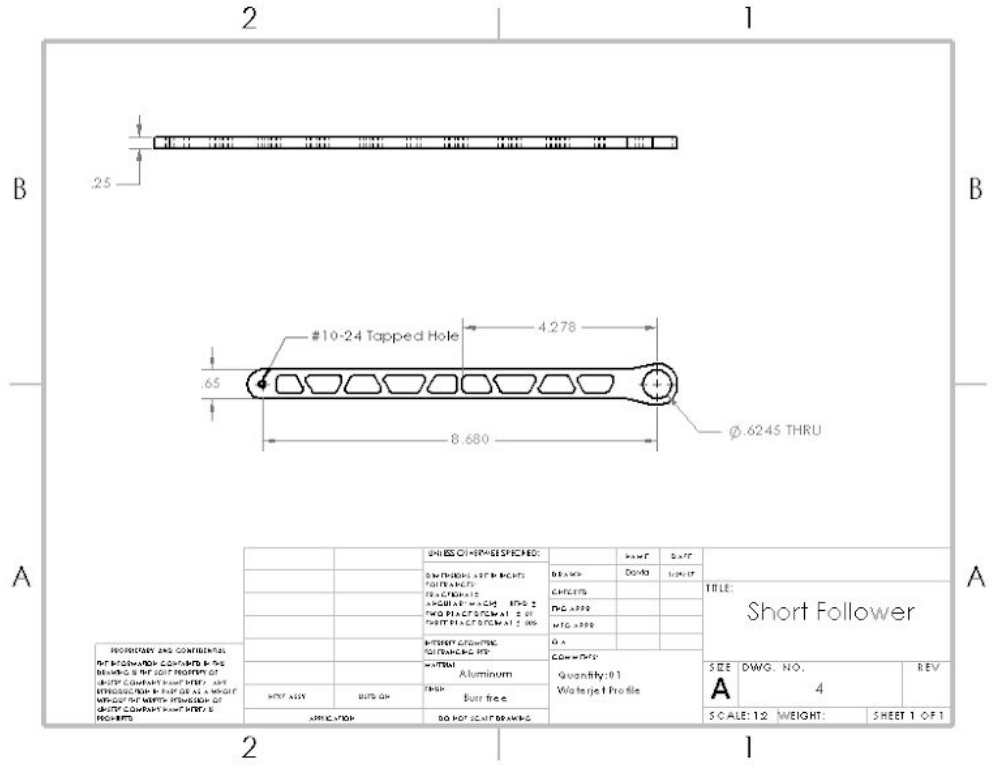
Part Name: Long Input

Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum Plate, 1/4" x 12" x 18"

Step #	*Process Description*	*Machine*	*Fixtures*	*Tool(s)*	*Speed (RPM)*
1	Make the machining blank	Waterjet			
2	Hold part in vise on top of parallels, make sure parallels aren't located under holes locations	Mill	vise	1.375 parallels	
3	Install edge finder into drill chuck.			Drill chuck, edgefinder	
4	Find datum lines for X and Y. (use the vice for y and flat part of pocket for x)	Mill	vise	Edge finder, drill chuck	1000
5	Centerdrill and drill a .25" pre-drill for the large press fit hole	Mill	vise	#3 Center drill, 1/4 drill bit, drill chuck	1100
6	Centerdrill and drill a .5" pre-drill for the large press fit hole	Mill	vise	#3 Center drill, 1/2 drill bit, drill chuck	500
6	Measure component to be press fit (micrometer) and choose reamer that is 0.001" undersized of .625" OD Bushing	Mill	Vise	Micrometer	
7	Center drill and drill a Pre-drill undersized by 0.015" for the required reamer	Mill	Vise	#3 Center drill, drill bit->(Depends on bushing measurement), drill chuck, and cutting fluid	(Depends on bushing measurement)
8	Ream hole with reamer undersized 0.001" for Bushing press fit	Mill	Vise	(size of reamer depends on OD of Bushing), drill chuck, and cutting fluid	(Depends on bushing measurement)
9	Centerdrill and drill the #10-24 hole.	Mill	vise	#3 Center drill, #25 drill bit, drill chuck	1400
10	Centerdrill and drill the #2-56 hole.	Mill	vise	#3 Center drill, #50 drill bit, drill chuck	1600
11	Remove part from vise deburr all			File, deburring tool	
12	Tap the 10-24 hole. Remove part from machine.	Tapping machine	vise	Drill chuck, #10-24 tap	
13	Tap the #2-56 hole. Remove part from machine.	Tapping machine	vise	Drill chuck, #2-56 tap	

Short Follower

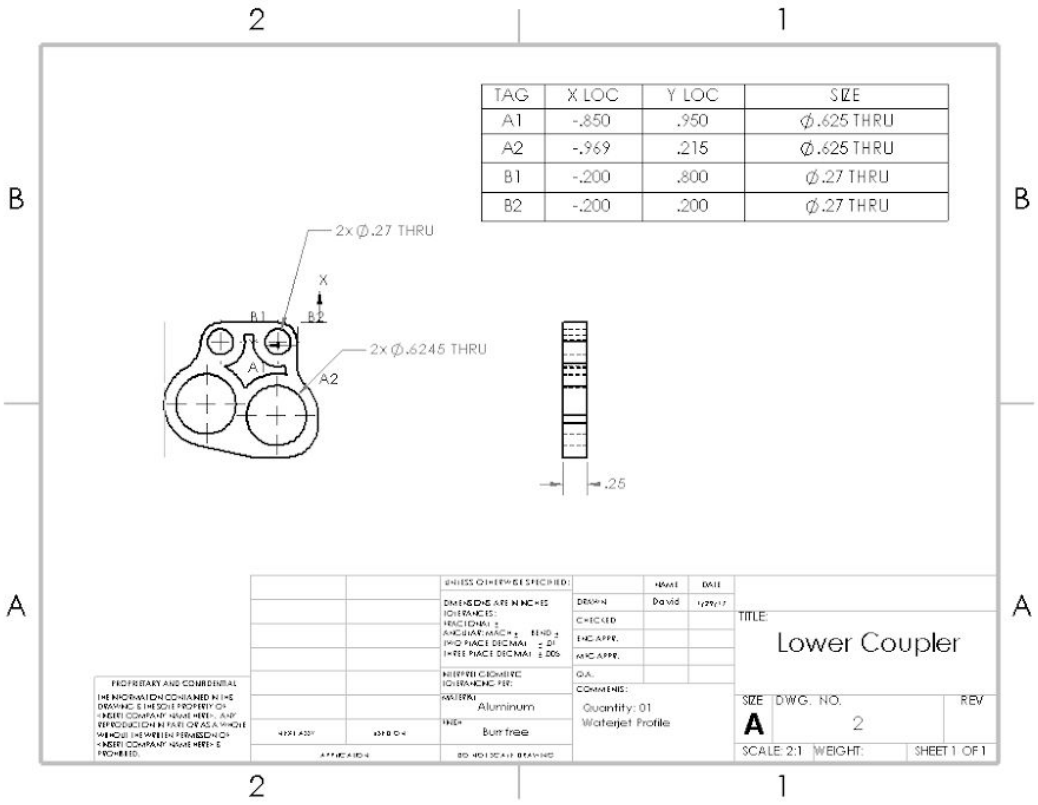


Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum Plate, 1/4" x 12" x 18"

Step #	Process Description	Machine	Fixtures	Tool(s)	Speed (RPM)
1	Make the machining blank	Waterjet			
2	Hold part in vise on top of parallels, make sure parallels aren't located under holes locations	Mill	vise	1.375 parallels	
3	Install edge finder into drill chuck.			Drill chuck, edgefinder	
4	Find datum lines for X and Y. (use the vice for y and flat part of pocket for x)	Mill	vise	Edge finder, drill chuck	1000
5	Centerdrill and drill a .25" pre-drill for the large press fit hole	Mill	vise	#3 Center drill, 1/4 drill bit, drill chuck	1100
6	Centerdrill and drill a .5" pre-drill for the large press fit hole	Mill	vise	#3 Center drill, 1/2 drill bit, drill chuck	500
6	Measure component to be press fit (micrometer) and choose reamer that is 0.001" undersized of .625" OD Bushing	Mill	Vise	Micrometer	
7	Center drill and drill a Pre-drill undersized by 0.015" for the required reamer	Mill	Vise	#3 Center drill, drill bit->(Depends on bushing measurement), drill chuck, and cutting fluid	(Depends on bushing measurement)
8	Ream hole with reamer undersized 0.001" for Bushing press fit	Mill	Vise	(size of reamer depends on OD of Bushing), drill chuck, and cutting fluid	(Depends on bushing measurement)
9	Centerdrill and drill the #10-24 hole.	Mill	vise	#3 Center drill, #25 drill bit, drill chuck	1400
10	Remove part from vise deburr all			File, deburring tool	
11	Tap the 10-24 hole. Remove part from machine.	Tapping machine	vise	Drill chuck, #10-24 tap	

Lower Coupler



Part Number: ME350-004 Revision Date: 31/1/2017

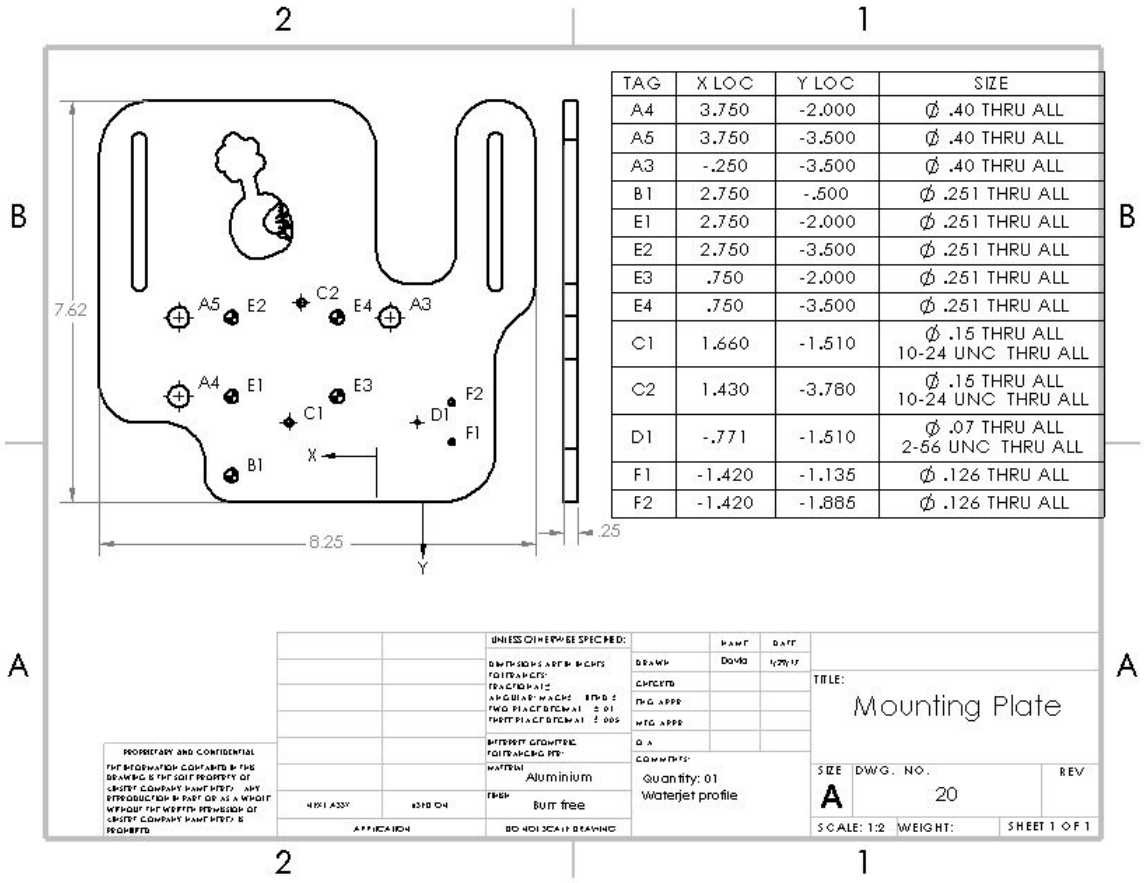
Part Name: Lower Coupler

Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum Plate, 1/4" x 12" x 18"

Step #	*Process Description*	*Machine*	*Fixtures*	*Tool(s)*	*Speed (RPM)*
1	Make the machining blank	Waterjet			
2	Hold part in vise on top of parallels, make sure parallels aren't located under holes locations during operations	Mill	vise	1.375 parallels	
3	Install edge finder into drill chuck.			Drill chuck, edgefinder	
4	Find datum lines for X and Y. (use the vice for y and flat part in the upper right of the drawing for x)	Mill	vise	Edge finder, drill chuck	1000
5	Centerdrill and drill a 2X0.25" pre-drill for the large press fit holes	Mill	vise	#3 Center drill, 1/4 drill bit, drill chuck	1100
6	Centerdrill and drill a 2X0.5" pre-drill for the large press fit holes	Mill	vise	#3 Center drill, 1/2 drill bit, drill chuck	500
7	Measure component to be press fit (Micrometer) for one of the bearing holes and choose reamer that is 0.001" undersized of .625" OD Bushing	Mill	Vise	Micrometer	
8	Center drill and drill a Pre-drill undersized by 0.015" for the required reamer	Mill	Vise	#3 Center drill, drill bit->(Depends on bushing measurement), drill chuck, and cutting fluid	(Depends on bushing measurement)
9	Ream hole with reamer undersized 0.001" for Bushing press fit	Mill	Vise	(size of reamer depends on OD of Bushing), drill chuck, and cutting fluid	(Depends on bushing measurement)
10	Repeat steps 7-9 for the second bearing hole				
11	Centerdrill and drill the 2X0.27" holes.	Mill	vise	#3 Center drill, #H drill bit, drill chuck	1000
12	Remove part from vise deburr all			File, deburring tool	

Mounting Plate



TAG	X LOC	Y LOC	SIZE
A4	3.750	-2.000	Ø .40 THRU ALL
A5	3.750	-3.500	Ø .40 THRU ALL
A3	-.250	-3.500	Ø .40 THRU ALL
B1	2.750	-.500	Ø .251 THRU ALL
E1	2.750	-2.000	Ø .251 THRU ALL
E2	2.750	-3.500	Ø .251 THRU ALL
E3	.750	-2.000	Ø .251 THRU ALL
E4	.750	-3.500	Ø .251 THRU ALL
C1	1.660	-1.510	Ø .15 THRU ALL 10-24 UNC THRU ALL
C2	1.430	-3.780	Ø .15 THRU ALL 10-24 UNC THRU ALL
D1	-.771	-1.510	Ø .07 THRU ALL 2-56 UNC THRU ALL
F1	-1.420	-1.135	Ø .126 THRU ALL
F2	-1.420	-1.885	Ø .126 THRU ALL

UNLESS OTHERWISE SPECIFIED:
DRAWN David 1/29/15
CHECKED
ENG APPR
MFG APPR
Q A
COMMENTS:
Quantity: 01
Waterjet profile
Aluminium
Burr free
TITLE: Mounting Plate
SIZE A
DWG. NO. 20
REV
SCALE: 1:2
WEIGHT:
SHEET 1 OF 1

Part Number: ME350-005 Revision Date: 31/1/2017

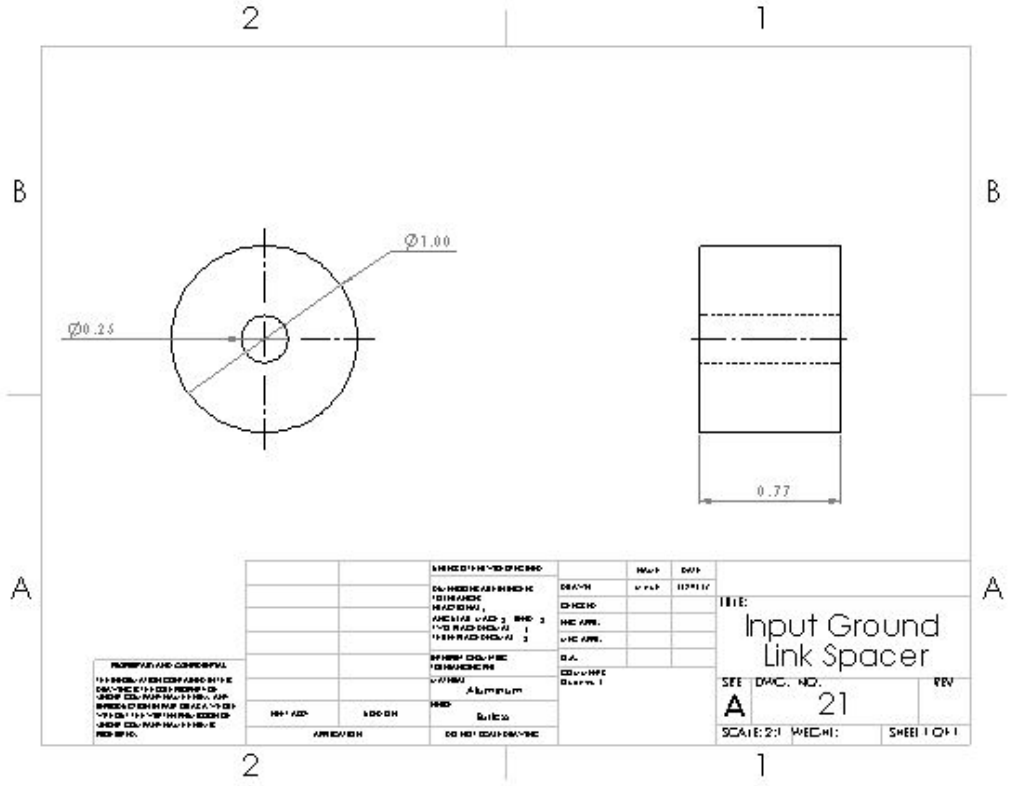
Part Name: Mounting Plate

Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum Plate, 1/4" x 12" x 18"

Step #	*Process Description*	*Machine*	*Fixtures*	*Tool(s)*	*ω (RPM)*
1	Make the machining blank. Only do the the outside profile and pockets. Holes and hole locations need to be done on the manual mill	Waterjet			
2	Position piece Mill using toe clamps, 1:2:3 blocks, and the pins that keep the edge parallel with the mill table. Make sure the blocks are placed so that they won't interfere with any holes. (If it doesn't fit in vise) If it does fit in vise, hold part in vise on top of parallels, make sure parallels aren't located under holes locations during operations	Mill	Toe Clamps, 1:2:3 blocks	1.375 parallels, toe claps, 2X1:2:3 blocks, 2 positioning cylinders	
3	Install edge finder into drill chuck.			Drill chuck, edgefinder	
4	Find datum lines for X and Y as shown in the drawing	Mill	vise	Edge finder, drill chuck	1000
5	Centerdrill and drill a 0.25" pre-drill for the large .4" hole	Mill	vise	#3 Center drill, 1/4 drill bit, drill chuck	1100
6	Center drill and drill the 3X0.40" holes	Mill	Vise	#3 Center drill, X drill bit, drill chuck, and cutting fluid	600
7	Measure component for clearance fit (micrometer) and choose reamer that is 0.001" oversized of .25" OD Dowel	Mill	Vise	Micrometer	
8	Center drill and drill a Pre-drill oversized by 0.015" for the required reamer	Mill	Vise	#3 Center drill, drill bit->(Depends on dowel measurement), drill chuck, and cutting fluid	(Depends on dowel measurement)
9	Ream hole with reamer oversized 0.001" for Dowel clearance fit	Mill	Vise	(size of reamer depends on OD of dowel), drill chuck, and cutting fluid	(Depends on dowel measure
10	Repeat Steps 6-8 for the other .251" clearance holes				
11	Measure component for clearance fit (micrometer) and choose reamer that is 0.001" oversized of .125" OD Bolt	Mill	Vise	Micrometer	
12	Center drill and drill a Pre-drill oversized by 0.015" for the required reamer	Mill	Vise	#3 Center drill, drill bit->(Depends on bolt measurement), drill chuck, and cutting fluid	(Depends on bolt measurement)
13	Ream hole with reamer oversized 0.001" for Bolt clearance fit	Mill	Vise	(size of reamer depends on OD of bolt), drill chuck, and cutting fluid	(Depends on bolt measurement)
14	Repeat Steps 10-12 for the other .126" clearance holes				
15	Centerdrill and drill the 2X.15" holes	Mill	vise	#3 Center drill, #25 drill bit, drill chuck	1400
16	Centerdrill and drill the .07" hole	Mill	vise	#3 Center drill, #50 drill bit, drill chuck	1600
17	Remove part from vise deburr			File, deburring tool	
18	Tap the 10-24 holes	Tapping machine	vise	Drill chuck, 10-24 tap	
19	Tap the 2-56 hole	Tapping machine	vise	Drill chuck, 2-56 tap	

Input Ground Link Spacer



Part Number: ME350-007 Revision Date: 31/1/2017

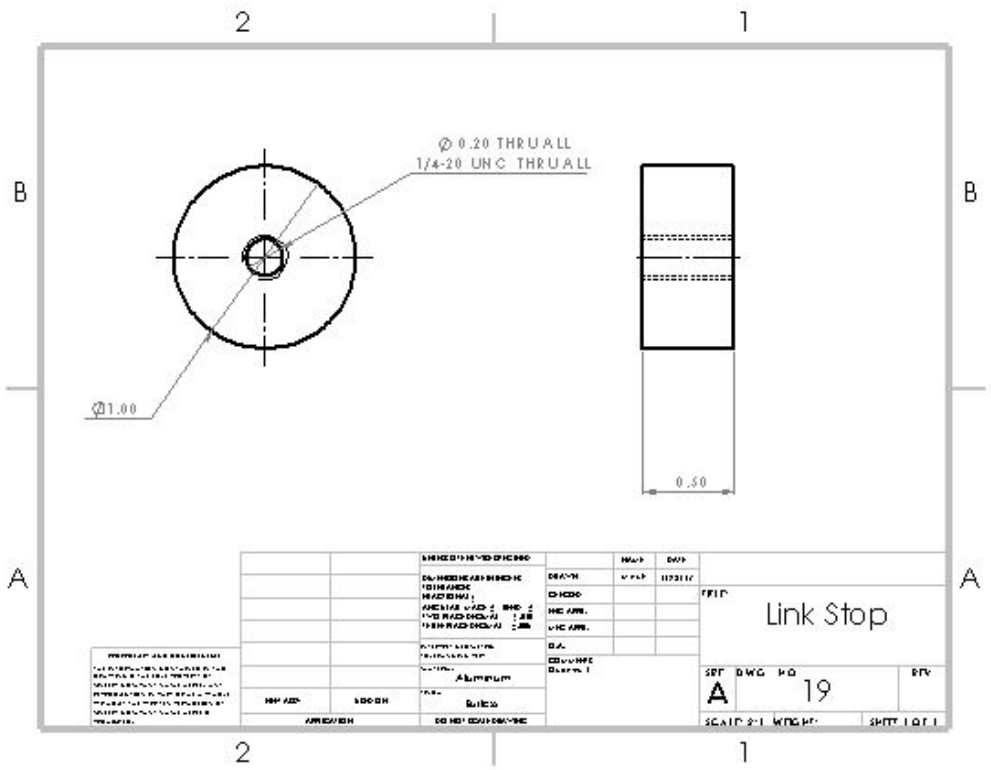
Part Name: Input Ground Link Spacer

Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum round stock, 1" Diameter, 12" Long

Step #	Process Description	Machine	Fixtures	Tool(s)	Speed (RPM)
1	Put the stock in the chuck and ensure there sufficient material for part thickness	Lathe	Chuck		
2	Install Drill chuck and Center Drill into Quill	Lathe	Chuck, Quill	Drill Chuck, #3 Center Drill	
3	Center Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, #3 Center Drill	1600
4	Remove Center Drill and Install .125" Drill Bit	Lathe	Chuck, Quill	Drill Chuck, .125" Drill Bit	
5	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, .125" Drill Bit	1600
6	Remove .125" Drill Bit and Install .25" Drill Bit	Lathe	Chuck, Quill	Drill Chuck, .25" Drill Bit	
6	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, .25" Drill Bit	1200
7	Slide Quill away and remove Chuck and Bit. Install Parting tool	Lathe	Chuck, Quill	Parting Tool	
8	Zero axis by pressing flat surface of parting tool against the flat end of circular stock	Lathe	Chuck	Parting Tool	
9	Once zeroed, move parting tool -0.125 and zero again to account for tool thickness	Lathe	Chuck	Parting Tool	
10	Adjust parting tool to -0.77	Lathe	Chuck	Parting Tool	
11	Part the stock	Lathe	Chuck	Parting Tool	300
12	Retrieve Part, Remove Stock, Deburr Part and Clean workstation	Lathe	Chuck		

Link Stop


2
1
B
A
⌀ 0.20 THRU ALL
1/4-20 UNC THRU ALL
⌀1.00
0.50
Link Stop
A
19
Aluminum

Part Number: ME350-008 Revision Date: 31/1/2017

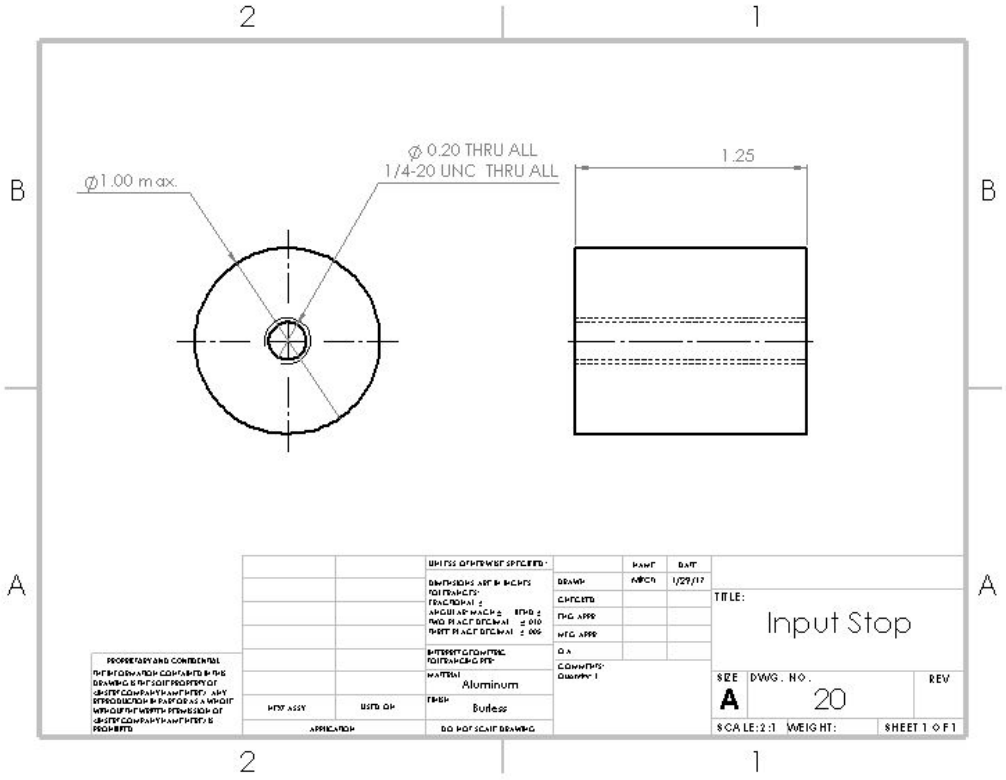
Part Name: Link Stop

Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum round stock, 1" Diameter, 12" Long

Step #	***Process Description***	***Machine***	***Fixtures***	***Tool(s)***	***Speed (RPM)***
1	Put the stock in the chuck and ensure there sufficient material for part thickness	Lathe	Chuck		
2	Install Drill chuck and Center Drill into Quill	Lathe	Chuck, Quill	Drill Chuck, #3 Center Drill	
3	Center Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, #3 Center Drill	1600
4	Remove Center Drill and Install .125" Drill Bit	Lathe	Chuck, Quill	Drill Chuck, .125" Drill Bit	
5	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, .125" Drill Bit	1600
6	Remove .125" Drill Bit and Install #7 Drill Bit	Lathe	Chuck, Quill	Drill Chuck, #7 Drill Bit	
6	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, #7 Drill Bit	1200
7	Slide Quill away and remove Chuck and Bit. Install Parting tool	Lathe	Chuck, Quill	Parting Tool	
8	Zero axis by pressing flat surface of parting tool against the flat end of circular stock	Lathe	Chuck	Parting Tool	
9	Once zeroed, move parting tool -0.125 and zero again to account for tool thickness	Lathe	Chuck	Parting Tool	
10	Adjust parting tool to -0.50	Lathe	Chuck	Parting Tool	
11	Part the stock	Lathe	Chuck	Parting Tool	300
12	Retrieve Part, Remove Stock, Deburr Part and Clean workstation	Lathe	Chuck		
13	Tap the 1/4-20 hole	Tapping machine	vise	Drill chuck, 1/4-20 tap	

Input Stop


2
1
B
A
Ø1.00 max.
Ø 0.20 THRU ALL
1/4-20 UNC THRU ALL
1.25
NAME
DATE
DRAWN
ARPON
1/29/17
CHECKED
ENG APPR
MFG APPR
Q.A
COMMENTS:
TITLE:
Input Stop
SIZE
A
DWG. NO.
20
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
Aluminum
Buffess
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING

Part Number: ME350-009 Revision Date: 31/1/2017

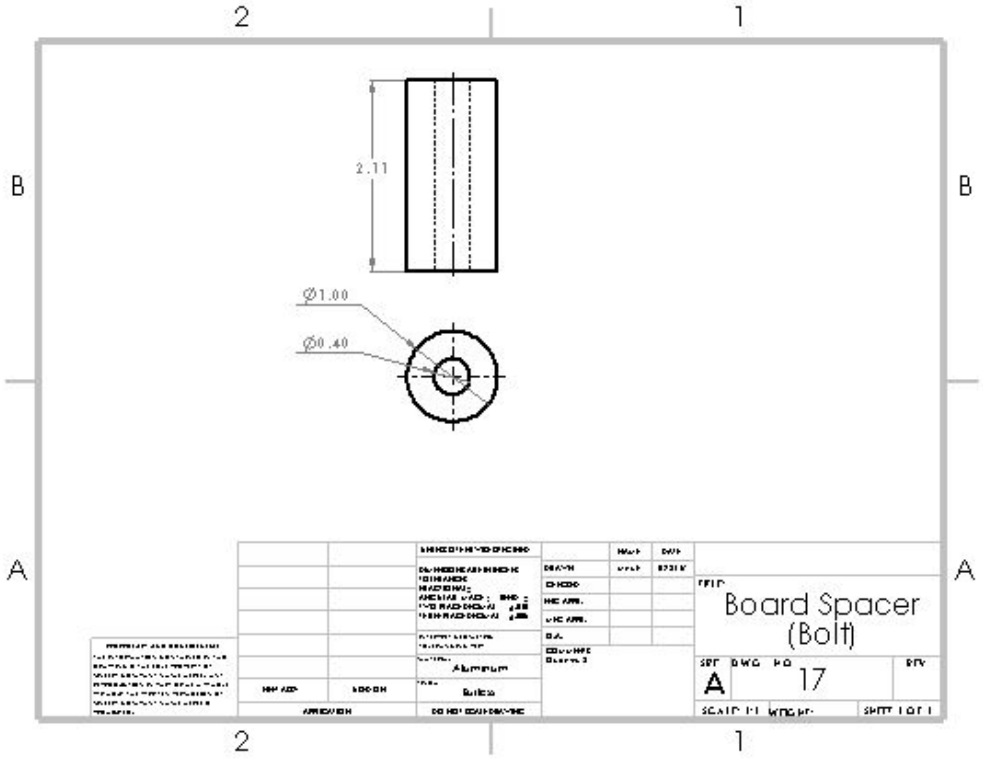
Part Name: Input Stop

Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum round stock, 1" Diameter, 12" Long

Step #	Process Description	Machine	Fixtures	Tool(s)	Speed (RPM)
1	Put the stock in the chuck and ensure there sufficient material for part thickness	Lathe	Chuck		
2	Install Drill chuck and Center Drill into Quill	Lathe	Chuck, Quill	Drill Chuck, #3 Center Drill	
3	Center Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, #3 Center Drill	1600
4	Remove Center Drill and Install .125" Drill Bit	Lathe	Chuck, Quill	Drill Chuck, .125" Drill Bit	
5	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, .125" Drill Bit	1600
6	Remove .125" Drill Bit and Install #7 Drill Bit	Lathe	Chuck, Quill	Drill Chuck, #7 Drill Bit	
6	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, #7 Drill Bit	1200
7	Slide Quill away and remove Chuck and Bit. Install Parting tool	Lathe	Chuck, Quill	Parting Tool	
8	Zero axis by pressing flat surface of parting tool against the flat end of circular stock	Lathe	Chuck	Parting Tool	
9	Once zeroed, move parting tool -0.125 and zero again to account for tool thickness	Lathe	Chuck	Parting Tool	
10	Adjust parting tool to -1.25	Lathe	Chuck	Parting Tool	
11	Part the stock	Lathe	Chuck	Parting Tool	300
12	Retrieve Part, Remove Stock, Deburr Part and Clean workstation	Lathe	Chuck		
13	Tap the 1/4-20 hole	Tapping machine	vise	Drill chuck, 1/4-20 tap	

Board Spacer (Bolt)



Part Number: ME350-011 Revision Date: 31/1/2017

Part Name: Board Spacer (Bolt)

Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum round stock, 1" Diameter, 12" Long

Step #	***Process Description***	***Machine***	***Fixtures***	***Tool(s)***	***Speed (RPM)***
1	Put the stock in the chuck and ensure there sufficient material for part thickness	Lathe	Chuck		
2	Install Drill chuck and Center Drill into Quill	Lathe	Chuck, Quill	Drill Chuck, #3 Center Drill	
3	Center Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, #3 Center Drill	1600
4	Remove Center Drill and Install .125" Drill Bit	Lathe	Chuck, Quill	Drill Chuck, .125" Drill Bit	
5	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, .125" Drill Bit	1600
6	Remove .125" Drill Bit and Install .25" Drill Bit	Lathe	Chuck, Quill	Drill Chuck, .25" Drill Bit	
6	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, .25" Drill Bit	1200
7	Remove .25" Drill Bit and Install X Drill Bit	Lathe	Chuck, Quill	Drill Chuck, X Drill Bit	
8	Drill Into Stock	Lathe	Chuck, Quill	Drill Chuck, X Drill Bit	600
9	Slide Quill away and remove Chuck and Bit. Install Parting tool	Lathe	Chuck, Quill	Parting Tool	
10	Zero axis by pressing flat surface of parting tool against the flat end of circular stock	Lathe	Chuck	Parting Tool	
11	Once zeroed, move parting tool -0.125 and zero again to account for tool thickness	Lathe	Chuck	Parting Tool	
12	Adjust parting tool to -2.11	Lathe	Chuck	Parting Tool	
13	Part the stock	Lathe	Chuck	Parting Tool	300
14	Retrieve Part, Remove Stock, Clean Machine and Deburr Part	Lathe	Chuck		

Bill of Materials

Part Number	Part Name	Material	Dimension(s)	Supplier	Quantity	Price (per item)	Notes
1	Upper Coupler	Plastic			1	----	3D Print
2	Lower Coupler	Aluminum	1/4"	Kit	1	----	
3	Long Input	Aluminum	10.71" x .65" x 1/4"	Kit	1	----	
4	Short Follower	Aluminum	8.68" x .65" x 1/4"	Kit	1	----	
5	Color Sensor	N/A	15.25mm x 2.16mm	Kit	1	----	
6	Sleeve Bearing	Oil Impregnated Brass	ID .25" OD .313" x .25"	Kit	4	----	
7	Washer	Oil Impregnated Brass	ID .26" OD .625" x .063"	Kit	9	----	
8	Cup	Polycarbonate	ID 1" x 3" Long x 1/8" Thick	Kit	1	----	
9	Metal Gearmotor	Steel	37Dx52L mm	Kit	1	----	
10	Needle Thrust Bearing	Steel	.250" Bore x .687" OD x .078" Height	Kit	4	----	
11	Motor Bracket	Steel	37D mm	Kit	1	----	
12	Mounting Plate	Aluminum	.25” Thick x 8.25” Long x 7.62” Height	Kit	1	----	
13	Ball Bearing	Steel	ID 1/4" OD 5/8" x .196" Height	McMaster-Carr	4	$6.56	
Omitted Gear Choice							
Omitted Gear							

Choice							
17	Board Spacer	Aluminum	ID .40" OD 1.00" x 2.11" Height	Kit / Purchase	3		Need to purchase stock
19	Link Stop	Aluminum	ID 1/4-20 UNC OD 1.00" x .5" Height	Kit / Purchase	1		Need to purchase stock
20	Input Stop	Aluminum	ID 1/4-20 UNC OD 1.00" x 1.25" Height	Kit / Purchase	1		Need to purchase stock
21	Input Ground Link Spacer	Aluminum	ID .25" OD 1.00" x .84" Height	Kit / Purchase	1		Need to purchase stock

1) Glue the 1 inch cup into the 3D printed coupler

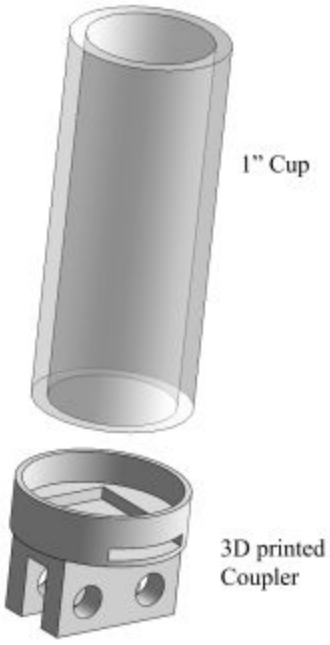


Figure B.1: Cup and Coupler Exploded Assembly

2) Press fit the two bearings into the aluminum half of the coupler

Figure B.2: Aluminum Coupler + Bearings Exploded Assembly

3) Attach the coupler parts using two 1-4/20 bolts and locknuts

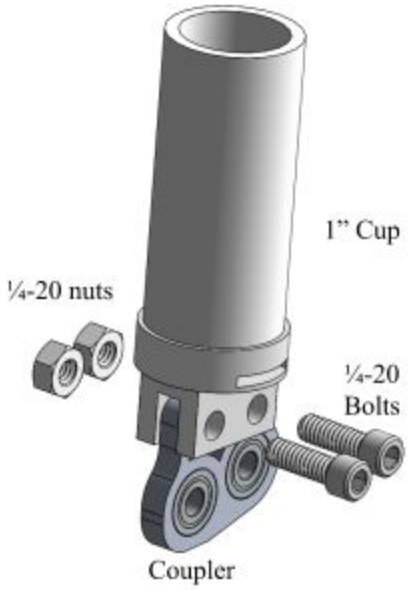


Figure B.3: Full Coupler Exploded Assembly

4) Connect the input link to the coupler. A ⅜ inch long shoulder screw with a oil infused thrust washer goes through bearing in the coupler. It goes through the bearing that is angled further down and further away from the cup. It goes through a needle bearing and is screwed into the input link.

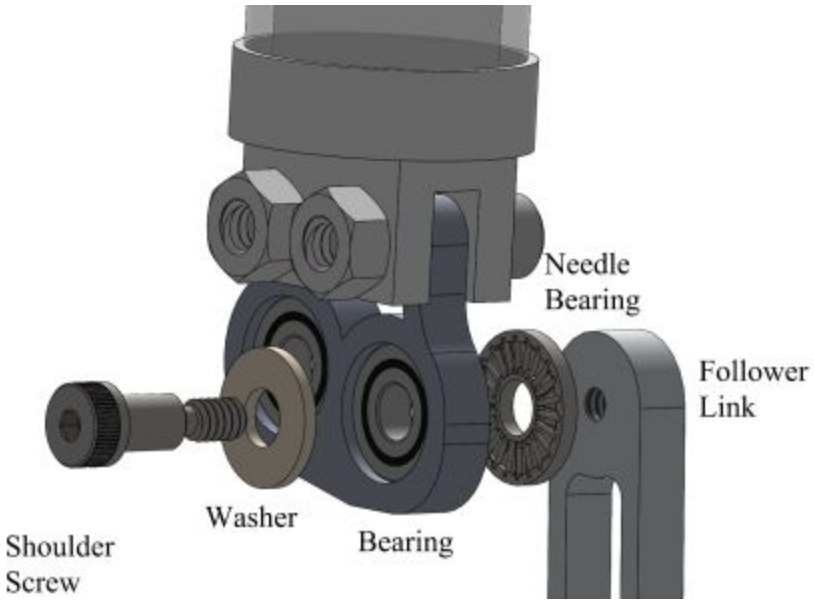


Figure B.4: Coupler to Follower Exploded Assembly

5) Press Fit the 1/16th Spring Pin into the Pulley and also the Input Link

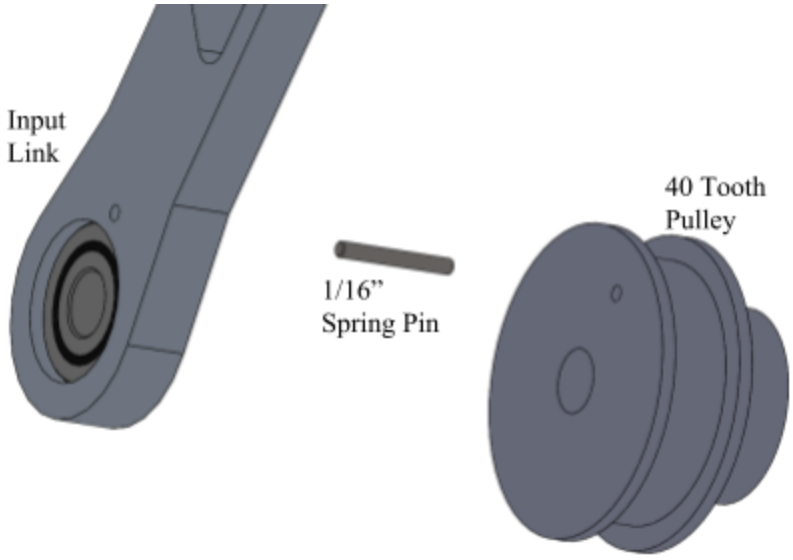


Figure B.5: Spring Pin Assembly

6) Attach the follower link to the coupler. The mounting system is exactly the same as the input, except the link is mounted on the opposite side of the coupler

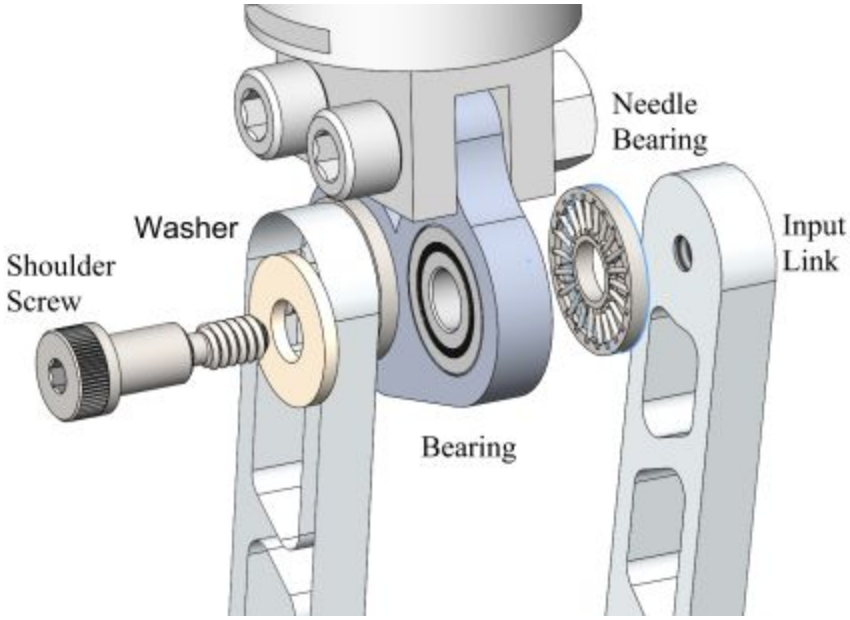


Figure B.6: Coupler to Input Exploded Assembly

7) Mount the Input link to the base board. Put a 2 ½" long shoulder screw through a oil embedded thrust washer, the bearing in the follower, a needle bearing, the follower ground spacer, and the pulley and screw it into the base board.

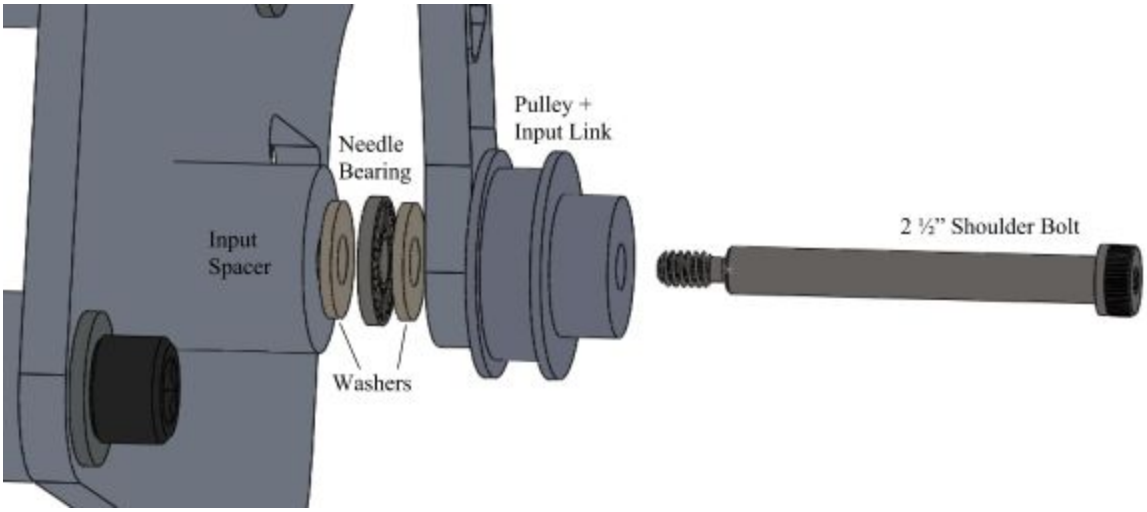


Figure B.7: Input to Base Plate Assembly

8) Attach the follower link to the board. Put a ½" long shoulder screw through a oil embedded thrust washer, bearing in the link, and a needle bearing between two more washers and screw it into the board.

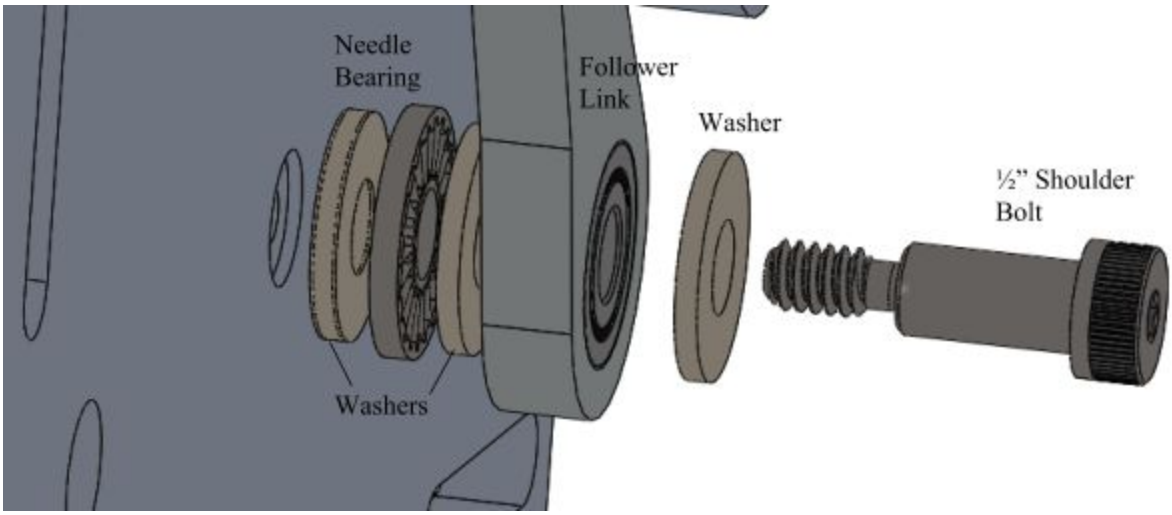


Figure B.8: Follower to Base Plate Assembly

9) Connect the link stops to the board with ½” long ¼-20 screws. The short one goes in the right slot and the longer one goes in the left slot. Exact placement in the slot is not important.

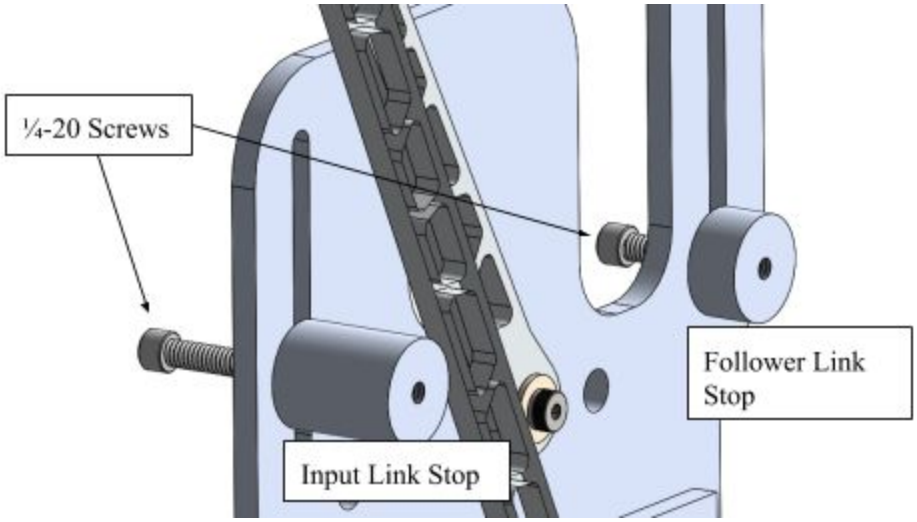


Figure B.9: Hard Stops Assembly to Baseplate

10) Attach the board to the field. Use 3 ⅜-16 bolts with 3 ⅜ washers and the 3 spacers to bolt the baseplate to the playing field. The upper left bolt screws into the upper leftmost hole in the playing field.

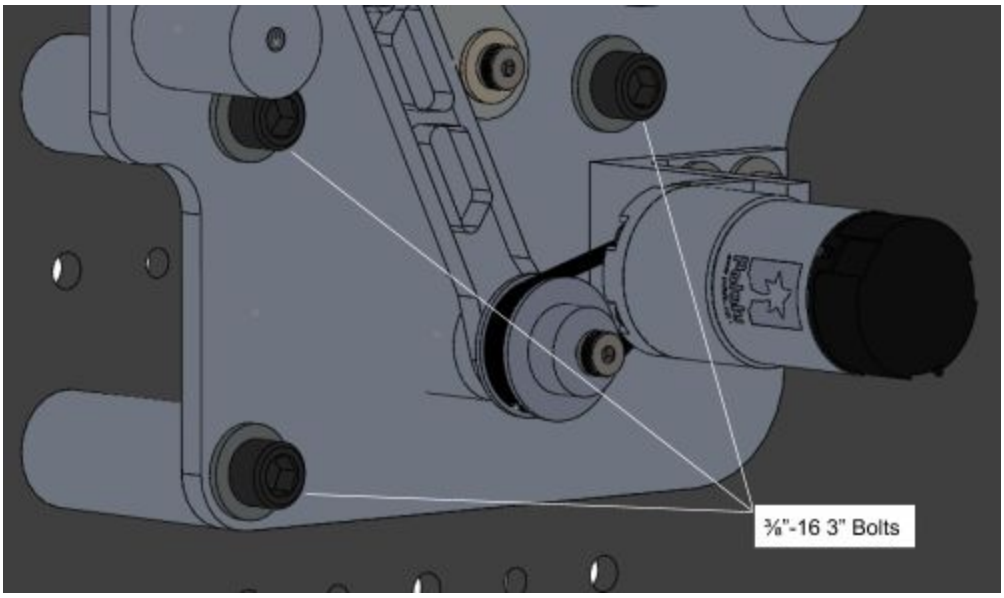


Figure B.10: Spacer + Baseplate Assembly

11) Assembly is complete

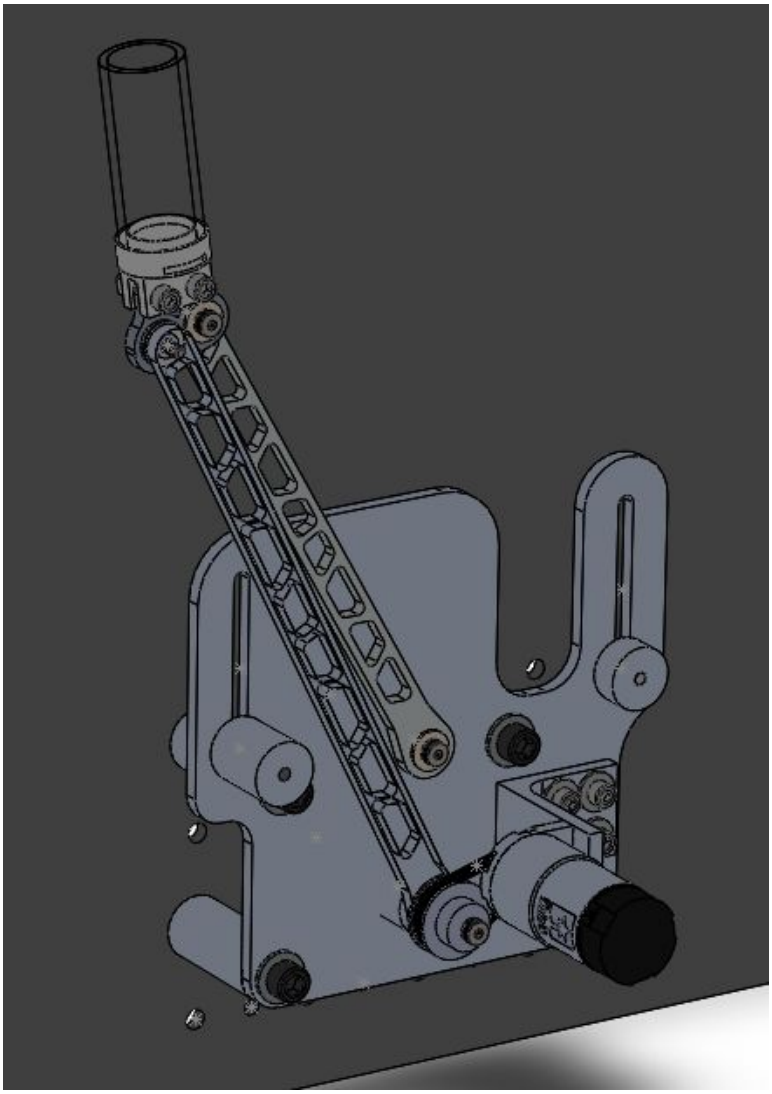
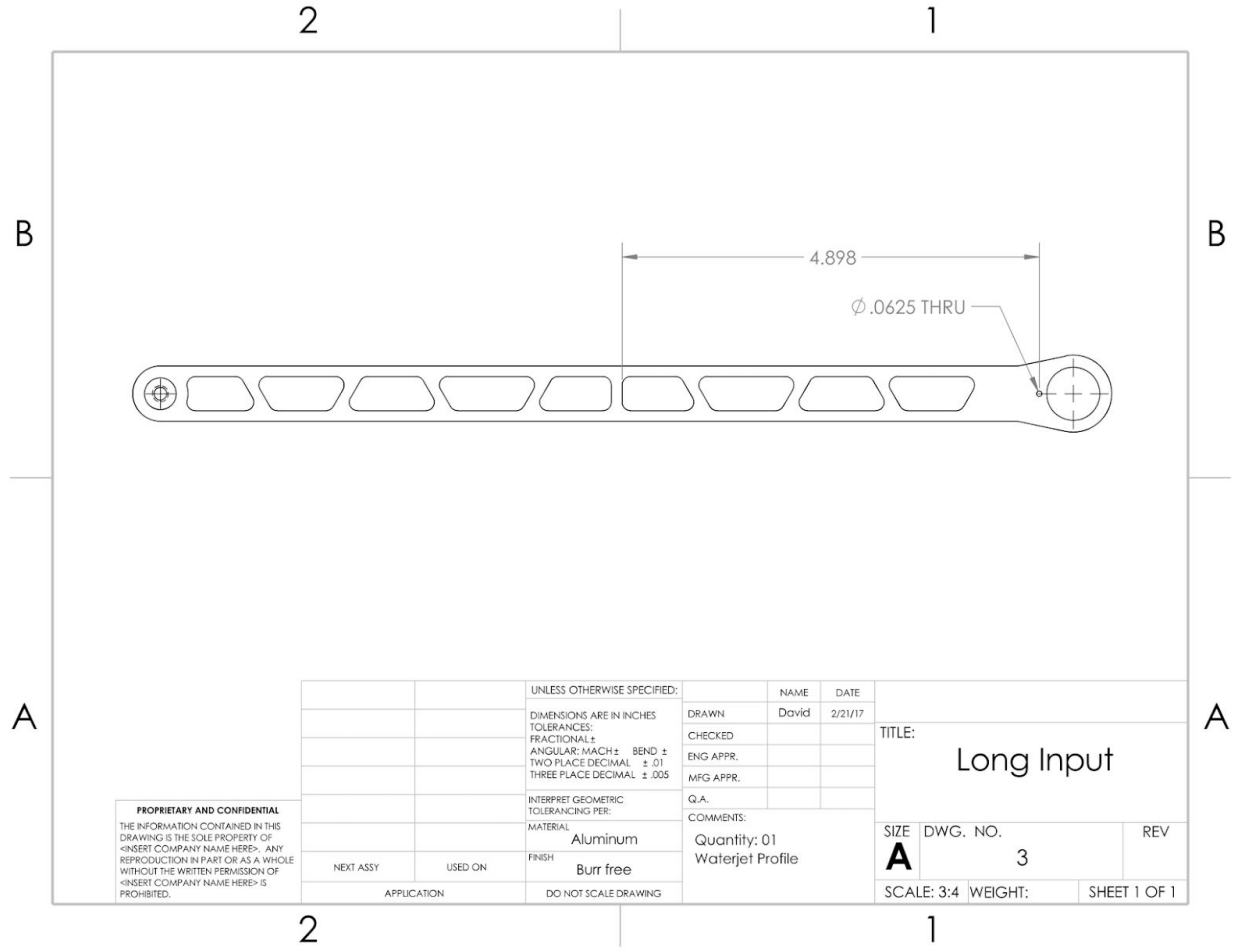


Figure B.11: Full Mechanism Assembly

Appendix C: Approval Packages, Bill of Materials, and Assembly Plan for Transmission Design

Approval Packages:

Long Input

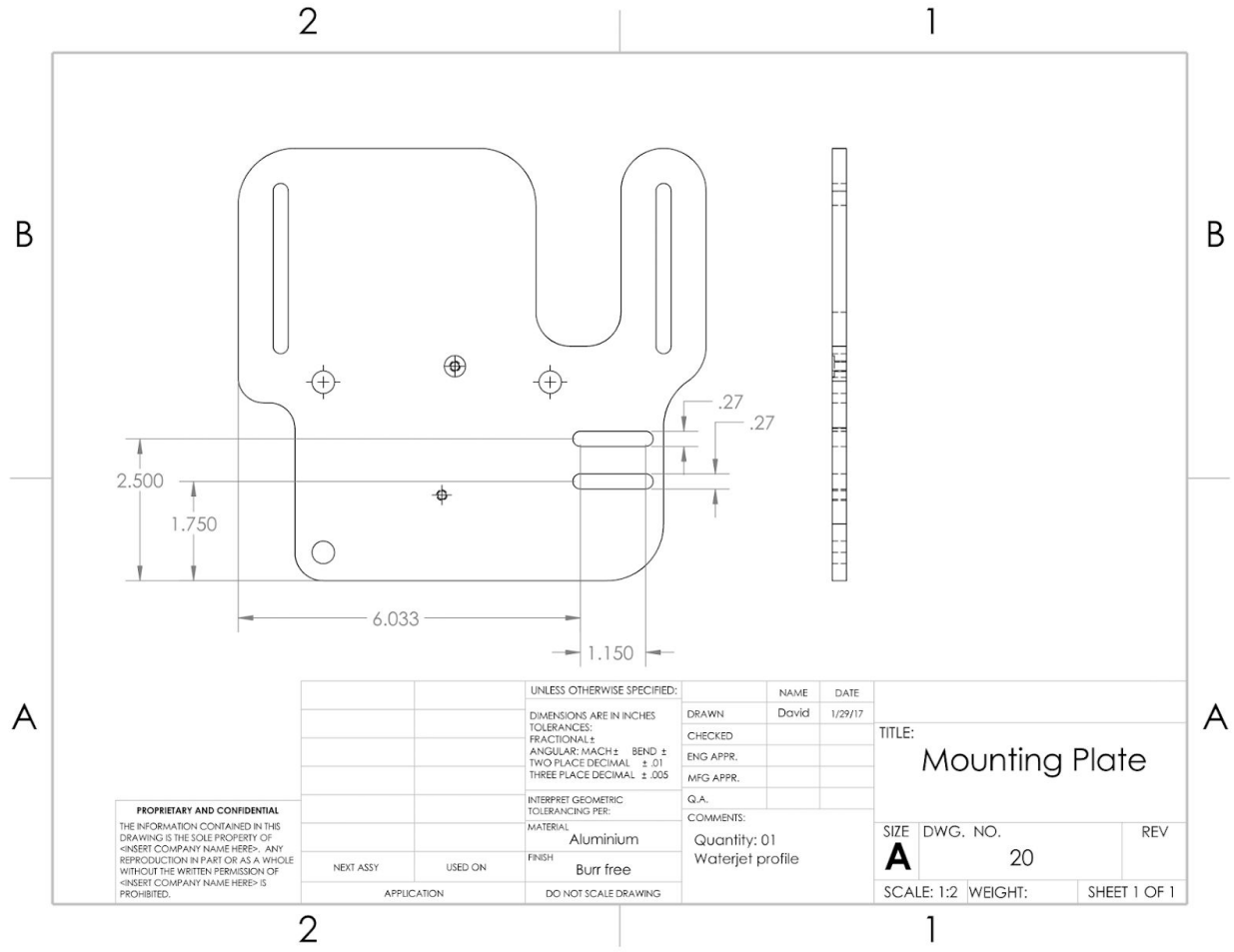


Part Number: ME350-002 Revision Date: 21/2/2017
Part Name: Long Input
Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum Plate, 1/4" x 12" x 18"

Step #	***Process Description***	***Machine***	***Fixtures***	***Tool(s)***	***Speed (RPM)***
1	Waterjet Profile	Waterjet			
2	Hold part in vise on top of parallels, make sure parallels aren't located under holes locations during operations	Mill	vise	1.375 parallels	
3	Install edge finder into drill chuck.			Drill chuck, edgefinder	
4	Find datum lines for X and Y. (use the vice for y and flat part of pocket for x)	Mill	vise	Edge finder, drill chuck	1000
5	Drill a Pre-drill undersized by 0.015" for the 1/16" reamed spring pin hole	Mill	Vise	#3 Center drill, #53 drill bit	1500
6	Ream hole with 1/16" Reamer	Mill	Vise	1/16" Reamer, collet or drill chuck if there isn't the right size collet available and cutting fluid	100
7	Remove part from vise deburr all			File, deburring tool	

Mounting Plate

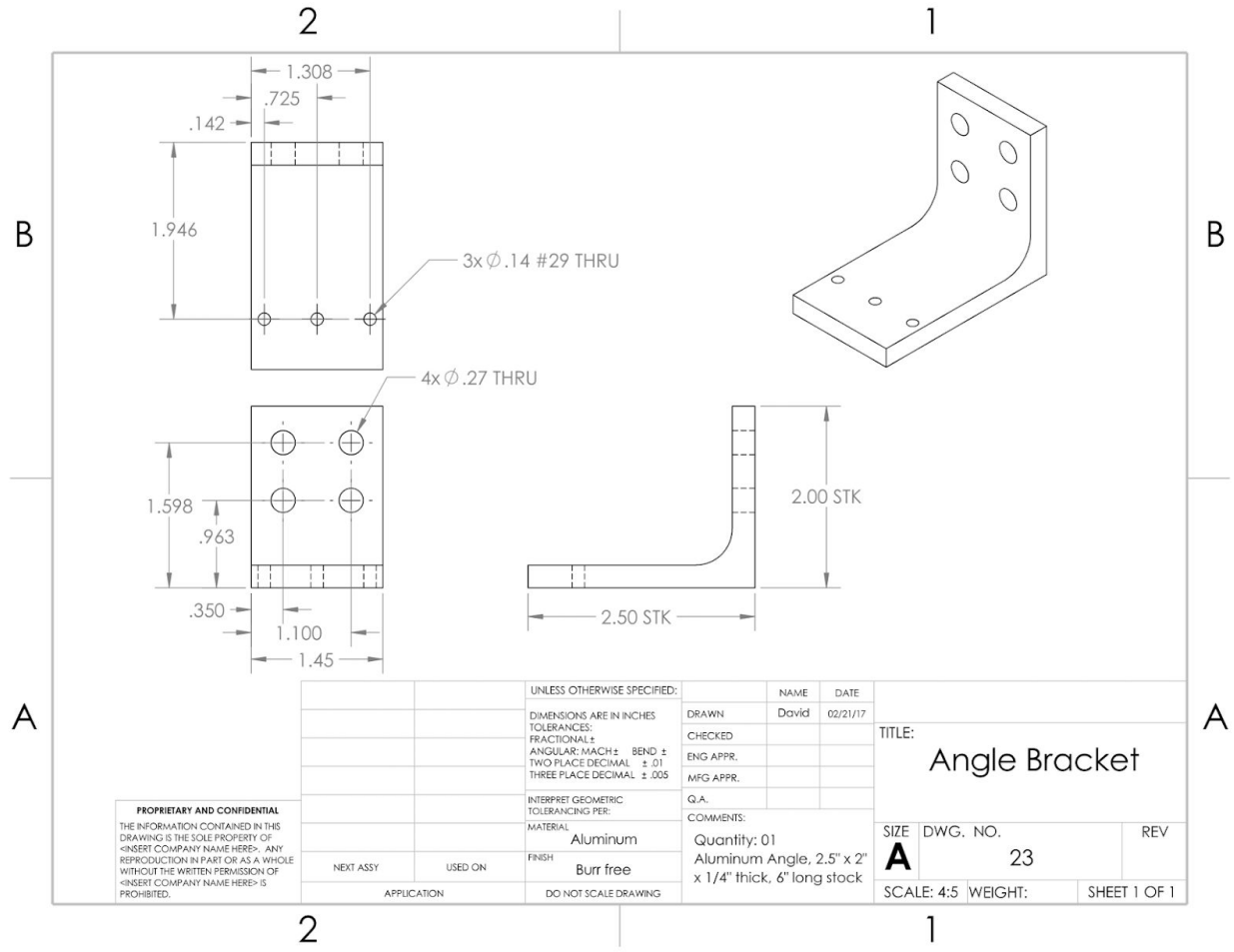


Part Number: ME350-002 Revision Date: 21/2/2017
Part Name: Long Input
Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum Plate, 1/4" x 12" x 18"

Step #	***Process Description***	***Machine***	***Fixtures***	***Tool(s)***	***Speed (RPM)***
1	Waterjet Profile	Waterjet			
2	Hold part in vise on top of parallels, make sure parallels aren't located under holes locations during operations	Mill	vise	1.375 parallels	
3	Install edge finder into drill chuck.			Drill chuck, edgefinder	
4	Find datum lines for X and Y. (use the vice for y and flat part of pocket for x)	Mill	vise	Edge finder, drill chuck	1000
5	Drill a Pre-drill undersized by 0.015" for the 1/16" reamed spring pin hole	Mill	Vise	#3 Center drill, #53 drill bit	1500
6	Ream hole with 1/16" Reamer	Mill	Vise	1/16" Reamer, collet or drill chuck if there isn't the right size collet available and cutting fluid	100
7	Remove part from vise deburr all			File, deburring tool	

Angle Bracket

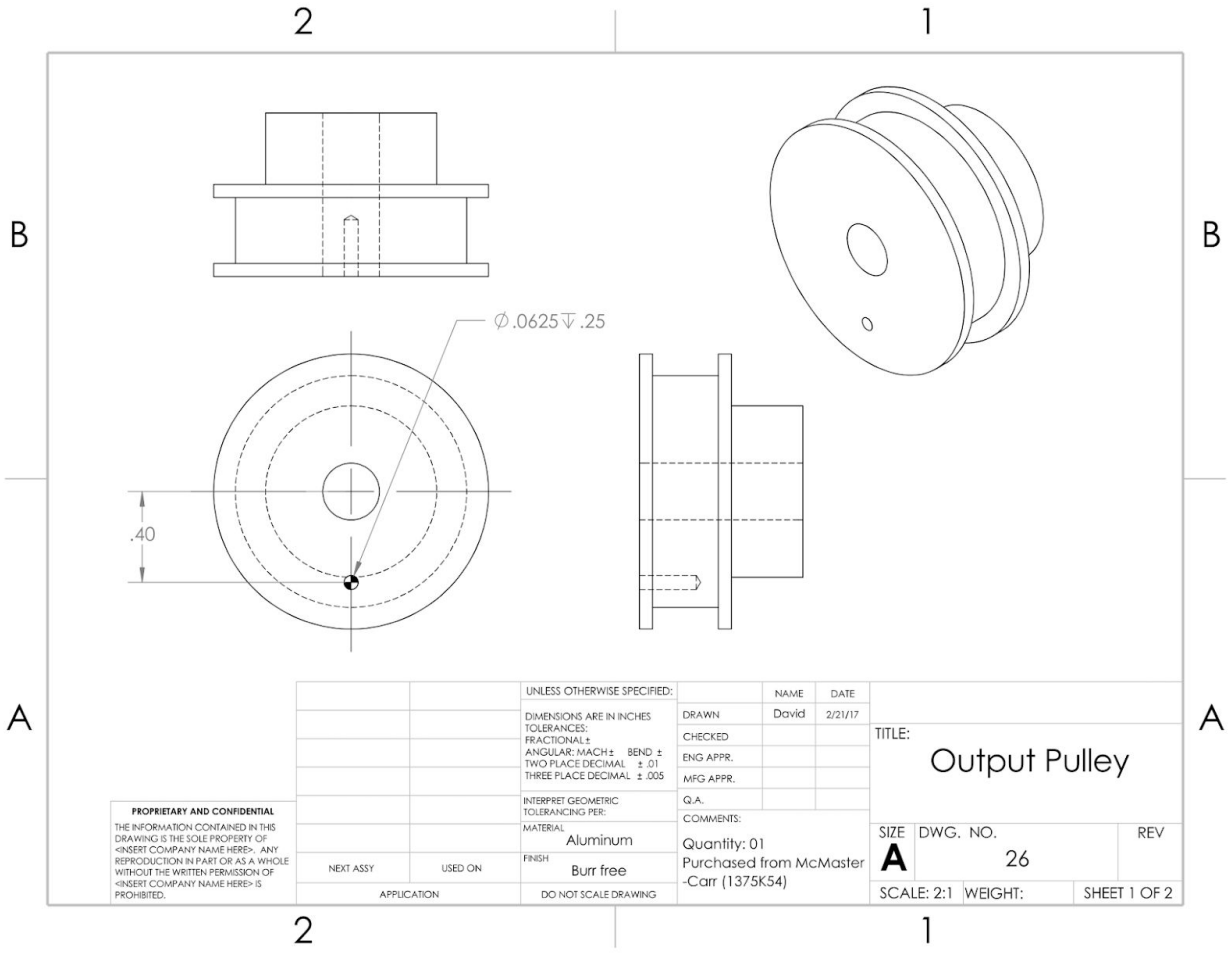


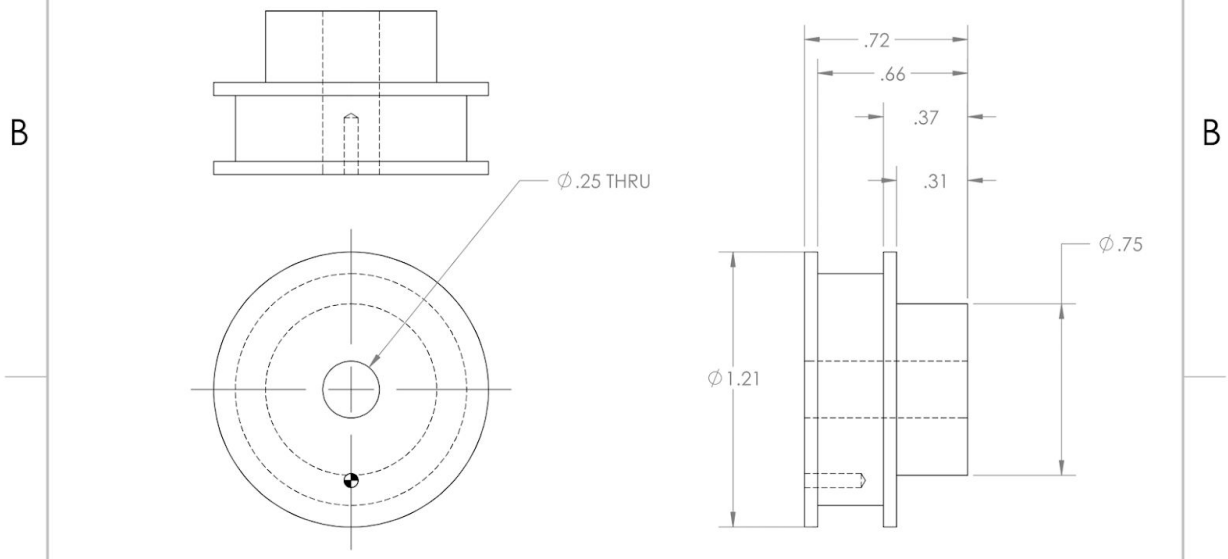
Part Number: ME350-001 Revision Date: 21/2/2017
Part Name: Angle Bracket
Team Name: Plumbus

Raw Material Stock: 6061-T6 Aluminum Angle, 2.5" x 2" x 1/4" thick, 6" long

Step #	*Process Description*	*Machine*	*Fixtures*	*Tool(s)*	*Speed (RPM)*
1	Cut 1.45" aluminum angle stock >.125 of finish length and deburr.	Band Saw		File	300 ft/min
2	Hold part in vise on top of parallels with >.125" material sticking out and 2.5" side flat.	Mill	vise	1.375 parallels	
3	Clamp one of the x dimensioin stoppers on the vice	Mill	vise	X dimension Stopper	
4	Install edge finder into drill chuck.			Drill chuck, edgefinder	
5	Find datum lines for X and Y. (no need to find datum again if part is repositioned because you can zero y to the vice and x to the stopper	Mill	vise	Edge finder, drill chuck	1000
6	Remove edge finder and install cutter and collet	Mill	vise	Drill chuck, edgefinder, 3/4 inch 2-flute endmill, collet	
7	Mill the band saw cut sides with a conventional cut in 10 thou increments until 5 thou above the desired 1.45” length	Mill	Vise	3/4 inch 2-flute endmill, collet	840
8	Mill a 5 thou finish pass slowly with a climb cut	Mill	Vise	3/4 inch 2-flute endmill, collet	840
9	Remove cutter and collet and install drill chuck	Mill	vise	Drill chuck, 3/4 inch 2-flute endmill, collet	
10	Center drill and drill the 3X0.14" holes	Mill	Vise	#3 Center drill, #6 drill bit, drill chuck, and cutting fluid	1500
11	Remove part from vise deburr all			File, deburring tool	
12	Reposion the part in the vise on top of parallels with the 2" side flat	Mill	vise	1.375 parallels	
13	Center drill and drill the 4X0.27" holes	Mill	Vise	#3 Center drill, H drill bit, drill chuck, and cutting fluid	1000
14	Remove part from vise deburr all			File, deburring tool	

Output Pulley





		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± .01 THREE PLACE DECIMAL ± .005	DRAWN	David	2/21/17
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL Aluminum	COMMENTS: Quantity: 01 Purchased from McMaster -Carr (1375K54)		
NEXT ASSY	USED ON	FINISH Burr free			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:
Output Pulley

SIZE	DWG. NO.	REV
A	26	
SCALE: 2:1	WEIGHT:	SHEET 2 OF 2

Part Number: ME350-003 Revision Date: 21/2/2017
Part Name: Pulley
Team Name: Plumbus

Raw Material Stock: MXL Series Corrosion-Resistant Timing Belt Pulley

Step #	***Process Description***	***Machine***	***Fixtures***	***Tool(s)***	***Speed (RPM)***
1	Waterjet Profile	Waterjet			
2	Hold part in circular jawed vise	Mill	vise	circular jawed vise	
3	Install edge finder into drill chuck.			Drill chuck, edgefinder	
4	Find datum lines for X and Y. (use dial indicator)	Mill	vise	Edge finder, drill chuck, dial indicator	NA
5	Drill a Pre-drill undersized by 0.015" for the 1/16" reamed spring pin hole	Mill	Vise	#3 Center drill, #53 drill bit	1500
6	Ream hole with 1/16" Reamer	Mill	Vise	1/16" Reamer, collet or drill chuck if there isn't the right size	100
7	Remove part from vise deburr all			File, deburring tool	

Bill of Materials:

Part Number	Part Name	Material	Dimension(s)	Supplier	Quantity	Price (per item)	Notes
22	Spring Pin	Steel	1/16”	Assembly Room	1	----	
23	Angle Bracket	Aluminum	.25” x 2.50” x 2.00”	Kit	1	----	
24	Timing Belt	Urethane	.080” Pitch ¼” Width 5.2” OD	McMaster-Carr	1	2.55	1679K69
25	Input Pulley	Aluminum	0.685" OD 0.080" Pitch 0.208" W	McMaster-Carr	1	11.02	1375K42
26	Output Pulley	Output Pulley	1.21" OD 0.080" Pitch 0.276" W	McMaster-Carr	1	14.24	1375K55

Assembly Manual:

1) Connect the gearbox to the bracket using size 8mm long M3 screws

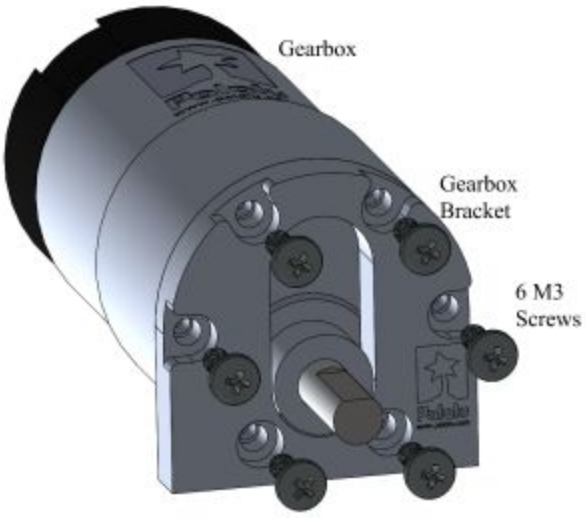


Figure C.1: DC Motor Assembly

2) Place the 20 tooth Pulley onto the Motor Shaft and screw in Set Screw to conjoin the two.

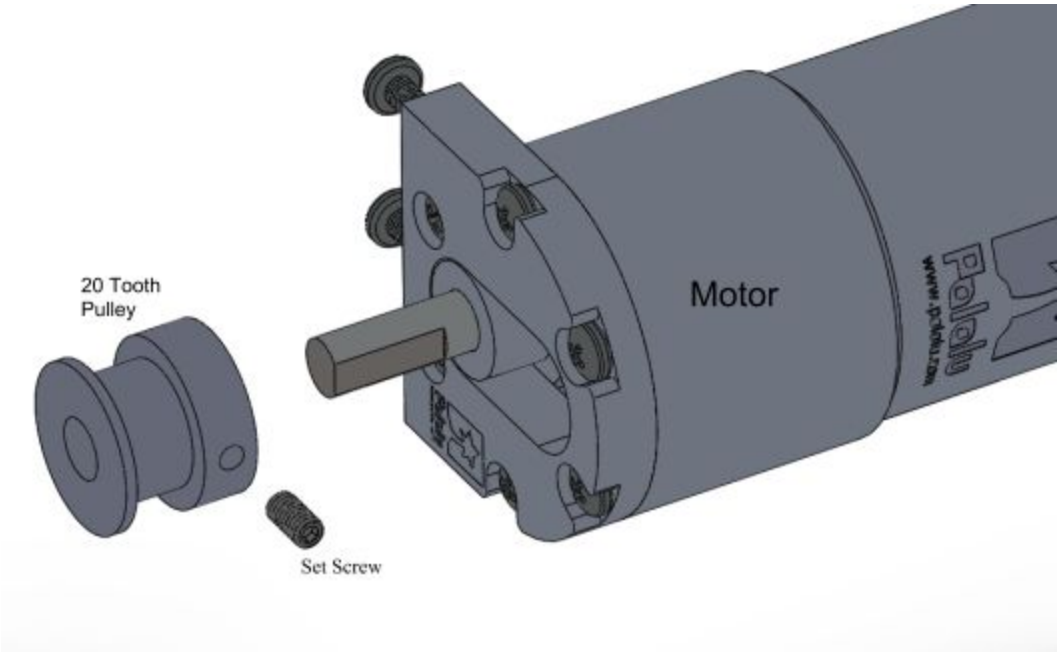


Figure C.2: 20 Tooth Pulley Assembly

3) Be sure to assemble the pulley on the input linkage. As pictured in Figure C.3, the transmission is assembled using an aluminum mount, an oil impregnated washer, a needle thrust bearing, another oil impregnated washer, a press fit ball bearing, a spring pin, the 40 tooth pulley, and a shoulder screw. See section B for assembly.

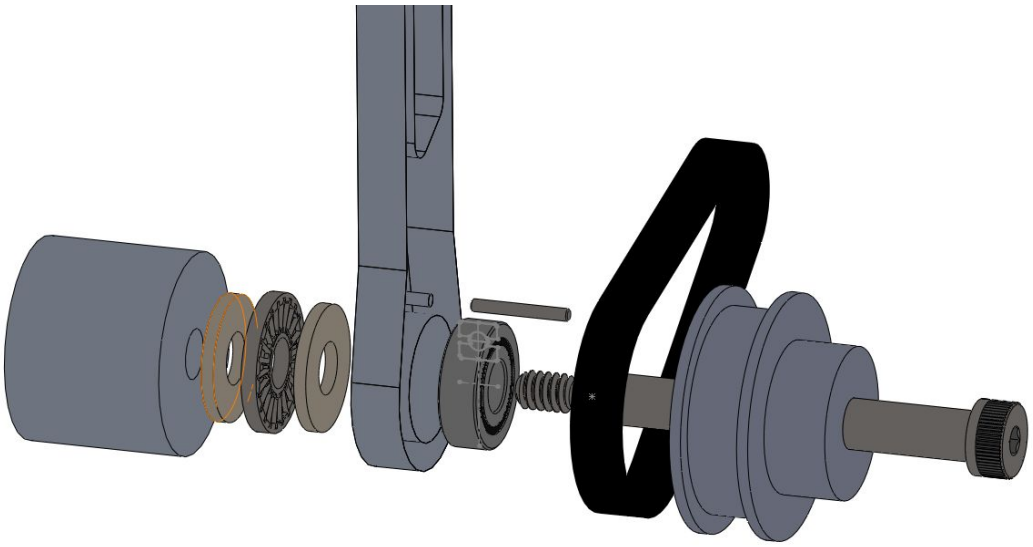


Figure C.3: 40 Tooth Input Pulley Assembly

4) Connect the gearbox bracket to the angled aluminum piece using 3 8mm M3 screws. Connect the Angled Bracket to the baseplate using 4 ¼-20 1" Bolts, 8 Washers (4 in the front of the plate and 4 in the back),, and 4 ¼-20 Locknuts placed in the rear of the assembly. Attaching the mechanism further left in the baseplate slots is recommended as it will provide ease of assembly when putting the belt on.

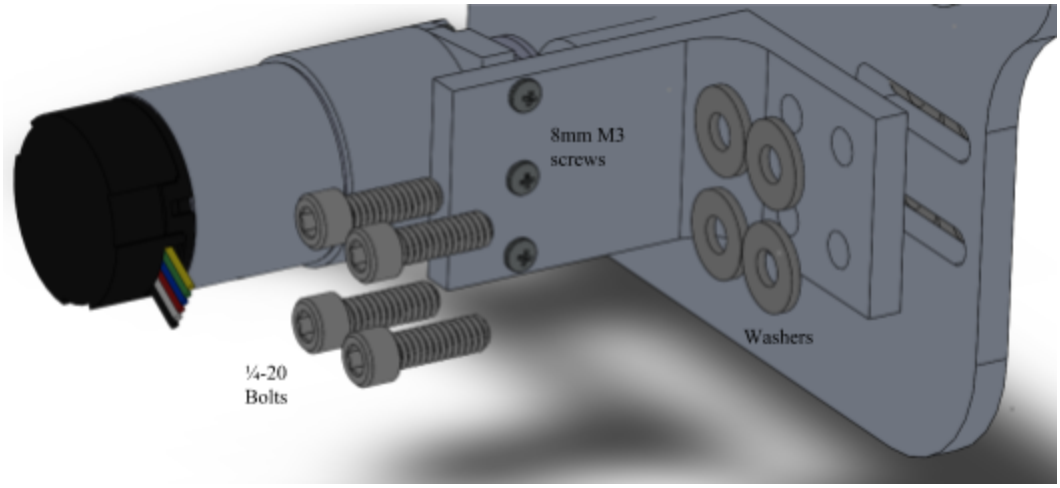


Figure C.4: Gearbox + Bracket Exploded Assembly

5) Attach the Belt to the input linkage pulley first, then slide the belt over the motor pulley. Tension the belt as needed by shifting the angle bracket further right in the slots on the baseplate and finally tightening the 4 bolts that hold the bracket in place. If vertical alignment is off, shim the angle bracket by adding washers between the baseplate and the bracket. Refer to figures 12.2 and 12.3 for visual aide.

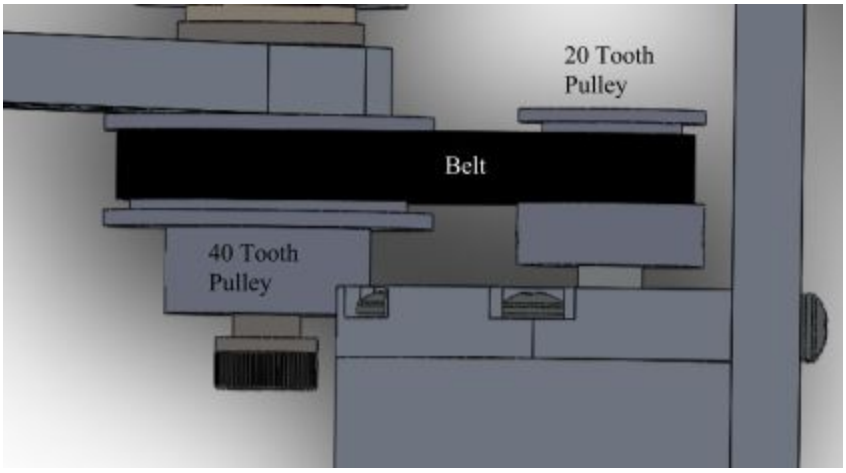
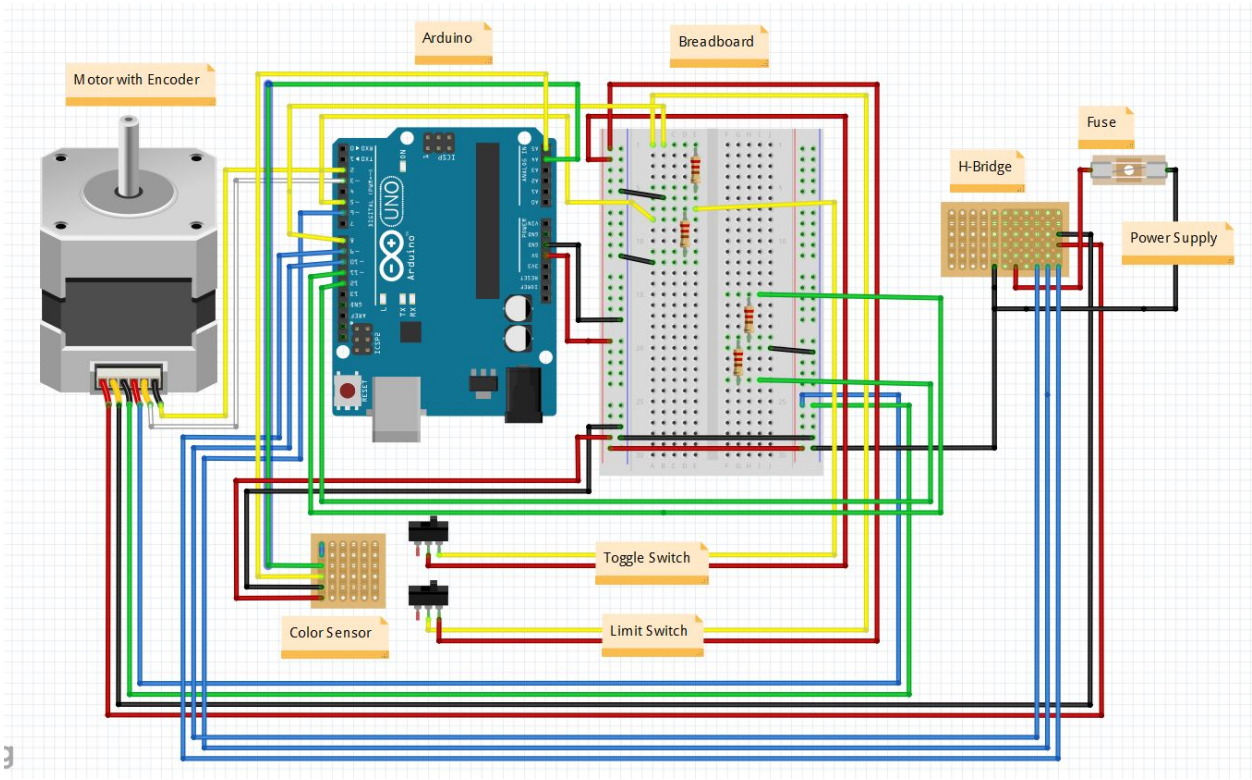


Figure C.5: Belt Assembly

Appendix D: Wiring Diagram, Arduino Code, Calculations, and Bill of Materials for Safety & Motor Controls

Wiring Diagram:



Arduino Code:

```
// ME350 Ball Handling
// updated 04/13/2017
//
// Team 52

////////////////////////////////////////////////
// DEFINE CONSTANTS AND GLOBAL VARIABLES:   //
////////////////////////////////////////////////

//** State Machine: **//
```

```
// CONSTANTS:
// Definition of states in the state machine
const int CALIBRATE     = 1;
const int WAIT          = 2;
const int MOVE_TO_CHUTE = 3;
const int WAIT_FOR_BALL = 4;
const int PUT_BALL      = 5;
const int PUT_UMICH     = 6;
int  state = CALIBRATE;
// VARIABLES:
// Global variable that keeps track of the state:
// Start the state machine in calibration state:


//** Color Sensor: **//
// Include the necessary code headers:
#include "Adafruit_TCS34725.h"
#include <Wire.h>
// CONSTANTS:
// Definition of ball types:
const int MAIZE = 1;
const int BLUE  = 2;
const int RED   = 3;
const int WHITE = 4;
const int NONE  = 5;
int ballColor = 5;
int COUNTERC = 0;         //Counter used in color sensor to time out
//int CalibrateCounter = 0;
// VARIABLES:
// Create a variable that allows us to access the color sensor:
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);
// Return values from the sensor
uint16_t red;
uint16_t green;
uint16_t blue;
uint16_t clear;

//** Computation of position and velocity: **//
// CONSTANTS:
// Settings for velocity computation:
const int MIN_VEL_COMP_COUNT = 2;     // [encoder counts] Minimal change in motor
position that must happen between two velocity measurements
```

```
52. const long MIN_VEL_COMP_TIME = 10000; // [microseconds] Minimal time that must pass
    between two velocity measurements
53. // VARIABLES:
54. volatile int motorPosition = 0; // [encoder counts] Current motor position (Declared 'volatile',
    since it is updated in a function called by interrupts)
55. volatile int encoderStatus = 0; // [binary] Past and Current A&B values of the encoder  (Declared
    'volatile', since it is updated in a function called by interrupts)
56. // The rightmost two bits of encoderStatus will store the encoder values from the current iteration
    (A and B).
57. // The two bits to the left of those will store the encoder values from the previous iteration (A_old
    and B_old).
58. float motorVelocity        = 0; // [encoder counts / seconds] Current motor velocity
59. int previousMotorPosition  = 0; // [encoder counts] Motor position the last time a velocity was
    computed
60. long previousVelCompTime   = 0; // [microseconds] System clock value the last time a velocity
    was computed
61.
62. //** High-level behavior of the controller:  **//
63. // CONSTANTS:
64. // Target positions:
65. int CalibrateCount = 0;
66. const int PRE_CALIBRATE_POSITION = -250;           // [Volt] Motor position used during
    pre-calibration
67. const int CALIBRATION_VOLTAGE = -6;               // [Volt] Motor voltage used during the
    calibration process
68. const int WAIT_POSITION       = +245;             // [encoder counts] Motor position
    corresponding to a wait position near the two chutes
69. const int CHUTE_1_POSITION    = +0;               // [encoder counts] Motor position
    corresponding to first chute
70. const int CHUTE_2_POSITION    = +412;             // [encoder counts] Motor position
    corresponding to second chute
71. const int PUT_POSITION        = +720;             // [encoder counts] Motor position corresponding
    to basket lane
72. const int UMICH_POSITION      = +700;             // [encoder counts] Motor position
    corresponding to UMICH balls
73. const int LOWER_BOUND         = CHUTE_1_POSITION;   // [encoder counts] Position of the
    left end stop
74. const int UPPER_BOUND         = PUT_POSITION;      // [encoder counts] Position of the right
    end stop
75. const int TARGET_BAND         = 20;               // [encoder counts] "Close enough" range when
    moving towards a target.
76. // Timing:
```

```
77. const long  WAIT_TIME        = 250000;           // [microseconds] Time waiting for the ball to drop.
78. // VARIABLES:
79. int activeChutePosition;           // [encoder counts] position of the currently active chute
80. unsigned long startWaitTime;        // [microseconds] System clock value at the moment the WAIT_FOR_BALL state started
81.
82. //** PID Controller  **//
83. // CONSTANTS:
84. float KP            = 0.12; // [Volt / encoder counts] P-Gain
85. float KI            = 0.01; // [Volt / (encoder counts * seconds)] I-Gain
86. float KD            = 0.005; // [Volt * seconds / encoder counts] D-Gain
87. const float SUPPLY_VOLTAGE = 10; // [Volt] Supply voltage at the HBridge
88. const float BASE_CMD       = 2; // [Volt] Voltage needed to overcome friction
89. // VARIABLES:
90. int  targetPosition  = 0; // [encoder counts] desired motor position
91. float positionError  = 0; // [encoder counts] Position error
92. float integralError  = 0; // [encoder counts * seconds] Integrated position error
93. float velocityError  = 0; // [encoder counts / seconds] Velocity error
94. float desiredVoltage = 0; // [Volt] Desired motor voltage
95. int   motorCommand   = 0; // [0-255] PWM signal sent to the motor
96. unsigned long executionDuration = 0; // [microseconds] Time between this and the previous loop execution.  Variable used for integrals and derivatives
97. unsigned long lastExecutionTime = 0; // [microseconds] System clock value at the moment the loop was started the last time
98.
99. //** Gravity Compensation Lookup Table: **//
100.     // CONSTANTS:
101.     const float FF_BALANCED_POSITION   = 200; // [encoder counts] Position at which the device is fully balanced.
102.     const float FF_VOLTAGE_LOWER_BOUND = 4.75; // [Volt] Voltage to be applied at the left endstop
103.     const float FF_VOLTAGE_UPPER_BOUND = -5; // [Volt] Voltage to be applied at the right endstop
104.
105.     //** Pin assignment: **//
106.     // CONSTANTS:
107.     const int PIN_NR_ENCODER_A       = 2;  // Never change these, since the interrupts are attached to pin 2 and 3
108.     const int PIN_NR_ENCODER_B       = 3;  // Never change these, since the interrupts are attached to pin 2 and 3
109.     const int PIN_NR_DROP_REQ        = 13;
110.     const int PIN_NR_ON_OFF_SWITCH    = 5;
```

```
111.    const int PIN_NR_CHUTE_1_READY    = 12;
112.    const int PIN_NR_CHUTE_2_READY    = 11;
113.    const int PIN_NRL_LIMIT_SWITCH    = 8;
114.    const int PIN_NR_PWM_OUTPUT       = 9;
115.    const int PIN_NR_PWM_DIRECTION_1  = 10;
116.    const int PIN_NR_PWM_DIRECTION_2  = 6;
117.    // End of CONSTANTS AND GLOBAL VARIABLES
118.
119.
120.    ////////////////////////////////////////////////////////////////////////////////
121.    // The setup() function is called when a sketch starts. Use it to initialize variables, //
122.    // pin modes, start using libraries, etc. The setup function will only run once, after  //
123.    // each powerup or reset of the Arduino board:                                          //
124.    ////////////////////////////////////////////////////////////////////////////////
125.    void setup() {
126.      // Declare which digital pins are inputs and which are outputs:
127.      pinMode(PIN_NR_ENCODER_A,       INPUT_PULLUP);
128.      pinMode(PIN_NR_ENCODER_B,       INPUT_PULLUP);
129.      pinMode(PIN_NR_CHUTE_1_READY,   INPUT);
130.      pinMode(PIN_NR_CHUTE_2_READY,   INPUT);
131.      pinMode(PIN_NR_ON_OFF_SWITCH,   INPUT);
132.      pinMode(PIN_NRL_LIMIT_SWITCH,   INPUT);
133.      pinMode(PIN_NR_DROP_REQ,        OUTPUT);
134.      pinMode(PIN_NR_PWM_OUTPUT,      OUTPUT);
135.      pinMode(PIN_NR_PWM_DIRECTION_1, OUTPUT);
136.      pinMode(PIN_NR_PWM_DIRECTION_2, OUTPUT);
137.
138.      // Turn on the pullup resistors on the encoder channels
139.      digitalWrite(PIN_NR_ENCODER_A, HIGH);
140.      digitalWrite(PIN_NR_ENCODER_B, HIGH);
141.
142.      // Activate interrupt for encoder pins.
143.      // If either of the two pins changes, the function 'updateMotorPosition' is called:
144.      attachInterrupt(0, updateMotorPosition, CHANGE);  // Interrupt 0 is always attached to
    digital pin 2
145.      attachInterrupt(1, updateMotorPosition, CHANGE);  // Interrupt 1 is always attached to
    digital pin 3
146.
147.      // Begin serial communication for monitoring.
148.      Serial.begin(115200);
149.      Serial.println("Start Executing Program.");
150.
151.      // Begin the operation of the color sensor and check if it works.
```

```
152.      if (tcs.begin()) {
153.        Serial.println("Color sensor found");
154.      } else {
155.        Serial.println("Color sensor not found.  Please check your connections");
156.        while (1); // infinite loop to halt the program
157.      }
158.
159.      // Initialize outputs:
160.      // Set the dropRequestSignal to low:
161.      digitalWrite(PIN_NR_DROP_REQ, LOW);
162.      // Set initial output to the motor to 0
163.      analogWrite(PIN_NR_PWM_OUTPUT, 0);
164.    }
165.    // End of function setup()
166.
167.
168.    ////////////////////////////////////////////////////////////////////////////////////////
169.    // After going through the setup() function, which initializes and sets the initial values,   //
170.    // the loop() function does precisely what its name suggests, and loops consecutively,       //
171.    // allowing your program to sense and respond. Use it to actively control the Arduino board.
    //
172.    ////////////////////////////////////////////////////////////////////////////////////////
173.    void loop() {
174.      // Determine the duration it took to execute the last loop. This time is used
175.      // for integration and for monitoring the loop time via the serial monitor.
176.      executionDuration = micros() - lastExecutionTime;
177.      lastExecutionTime = micros();
178.
179.      // Speed Computation:
180.      if ((abs(motorPosition - previousMotorPosition) > MIN_VEL_COMP_COUNT) || (micros()
    - previousVelCompTime) > MIN_VEL_COMP_TIME){
181.        // If at least a minimum time interval has elapsed or
182.        // the motor has travelled through at least a minimum angle ...
183.        // .. compute a new value for speed:
184.        // (speed = delta angle [encoder counts] divided by delta time [seconds])
185.        motorVelocity = (double)(motorPosition - previousMotorPosition) * 1000000 /
186.                              (micros() - previousVelCompTime);
187.        // Remember this encoder count and time for the next iteration:
188.        previousMotorPosition = motorPosition;
189.        previousVelCompTime   = micros();
190.      }
191.      if (digitalRead(PIN_NRL_LIMIT_SWITCH)==HIGH && motorVelocity==0) {
192.          // We reached the endstop.  Update the motor position to the limit:
```

```
193.          // (NOTE: If the limit switch is on the right, this must be UPPER_BOUND)
194.          motorPosition = LOWER_BOUND;
195.          // Reset the error integrator:
196.          integralError = 0;
197.      }
198.      //Reset to original values after each loop
199.
200.        Serial.print(" KP: ");
201.        Serial.print(KP);
202.        Serial.print(" KI: ");
203.        Serial.print(KI);
204.        Serial.print(" KD: ");
205.        Serial.print(KD);
206.
    //*************************************************************************
    ************************************************//
207.      // The state machine:
208.      switch (state) {
209.
210.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
211.       // 1********************************************** CALIBRATE
    **********************************************************1 //
212.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
213.       // In the CALIBRATE state, we move the mechanism to a position outside of the
214.       // work space (towards the limit switch).  Once the limit switch is on and
215.       // the motor stopped turning, we know that we are against the end stop
216.       case CALIBRATE:
217.
218.         // We don't have to do anything here since this state is only used to set
219.         // a fixed output voltage.  This happens further below.
220.         KP           = 0.12; // [Volt / encoder counts] P-Gain
221.         KI           = 0.01; // [Volt / (encoder counts * seconds)] I-Gain
222.         KD           = 0.009; // [Volt * seconds / encoder counts] D-Gain
223.         // Decide what to do next:
224.         if (digitalRead(PIN_NRL_LIMIT_SWITCH)==HIGH && motorVelocity==0) {
225.           // We reached the endstop.  Update the motor position to the limit:
226.           // (NOTE: If the limit switch is on the right, this must be UPPER_BOUND)
227.           motorPosition = LOWER_BOUND;
228.           // Reset the error integrator:
229.           integralError = 0;
230.           // Calibration is finalized. Transition into WAIT state
```

```
231.            Serial.println("State transition from CALIBRATE to WAIT");
232.            state = WAIT;
233.          }
234.
235.         // Otherwise we continue calibrating
236.         break;
237.
238.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
239.         // 2********************************************************* WAIT
    ***********************************************************2 //
240.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
241.         // In the WAIT state, we move the cup to a neutral position (close to the
242.         // chutes) while we wait for one of them to become active.
243.         case WAIT:
244.
245.           COUNTERC = 0;              //Counter used in color sensor to time out
246.           KP           = 0.125;     // [Volt / encoder counts] P-Gain
247.           KI           = 0.01;      // [Volt / (encoder counts * seconds)] I-Gain
248.           KD           = 0.003;     // [Volt * seconds / encoder counts] D-Gain
249.
250.           // Set the target position to a neutral position near the chutes:
251.           targetPosition = WAIT_POSITION;
252.
253.           // Decide what to do next:
254.           if (digitalRead(PIN_NR_CHUTE_1_READY) == HIGH) {
255.             // Chute 1 signaled to be ready to drop a ball.  Set the position of chute
256.             // 1 as the position of the active chute:
257.             activeChutePosition = CHUTE_1_POSITION;
258.             // Transit into MOVE_TO_CHUTE state:
259.             Serial.println("State transition from WAIT to MOVE_TO_CHUTE. Active chute = 1!");
260.             state = MOVE_TO_CHUTE;
261.           }
262.           if (digitalRead(PIN_NR_CHUTE_2_READY) == HIGH) {
263.             // Chute 2 signaled to be ready to drop a ball.  Set the position of chute
264.             // 2 as the position of the active chute:
265.             activeChutePosition = CHUTE_2_POSITION;
266.             // Transit into MOVE_TO_CHUTE state:
267.             Serial.println("State transition from WAIT to MOVE_TO_CHUTE. Active chute = 2!");
268.             state = MOVE_TO_CHUTE;
269.           }
270.           // Otherwise we continue waiting
```

```
271.
272.         break;
273.
274.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
275.       // 3*****************************************************
    MOVE_TO_CHUTE ******************************************************3 //
276.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
277.       // In the MOVE_TO_CHUTE state, we move the cup under one of the two chutes
278.       // (indicated by the variable active_chute). Once the position was reached
279.       // (with some error) and the motor stopped turning, we know that we are under
280.       // the chute.
281.       case MOVE_TO_CHUTE:
282.
283.         KP          = 0.1;      // [Volt / encoder counts] P-Gain
284.         KI          = 0.01;     // [Volt / (encoder counts * seconds)] I-Gain
285.         KD          = 0.005;    // [Volt * seconds / encoder counts] D-Gain
286.
287.         // Set the target position to chute 1 or 2:
288.         targetPosition = activeChutePosition;
289.
290.         // Decide what to do next:
291.         if (motorPosition <= (activeChutePosition + TARGET_BAND) && motorPosition >=
    (activeChutePosition - TARGET_BAND) && motorVelocity == 0) {
292.           // We reached the chute.  Ask the playing field to drop a ball by
293.           // setting chuteActivateSignal to HIGH:
294.           if (digitalRead(PIN_NRL_LIMIT_SWITCH)==HIGH && motorVelocity==0) {
295.             motorPosition = LOWER_BOUND;
296.             integralError = 0;
297.           }
298.           digitalWrite(PIN_NR_DROP_REQ, HIGH);
299.           // Start waiting timer:
300.           startWaitTime = micros();
301.           // Transition into WAIT_FOR_BALL state
302.           Serial.println("State transition from MOVE_TO_CHUTE to WAIT_FOR_BALL");
303.           state = WAIT_FOR_BALL;
304.
305.         }
306.         // Otherwise we continue moving towards the chute
307.
308.         break;
309.
```

```
310.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
311.        // 4*************************************************
    WAIT_FOR_BALL ************************************************************4 //
312.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
313.        // In this state, we stay at the chute and wait until the ball was dropped:
314.        case WAIT_FOR_BALL:
315.
316.          // The target position remains at either chute 1 or 2:
317.          targetPosition = activeChutePosition;
318.
319.          // Decide what to do next:
320.          if (micros()-startWaitTime>WAIT_TIME) {
321.            // We have waited long enough for the ball to drop. By now we should have
322.            // recieved it.
323.            // Set the chuteActivateSignal back to LOW.
324.            digitalWrite(PIN_NR_DROP_REQ, LOW);
325.
326.            // call ball color function to give ball color from sensor
327.            ballColor = evaluateColorSensor();
328.
329.            Serial.print("Ball color is: ");
330.            switch (ballColor) {
331.              case MAIZE: Serial.println("MAIZE."); break;
332.              case BLUE:  Serial.println("BLUE.");  break;
333.              case RED:   Serial.println("RED.");   break;
334.              case WHITE: Serial.println("WHITE."); break;
335.              case NONE:  Serial.println("NONE."); break;
336.            }
337.            // Transition into PUT_BALL state
338.            if(ballColor == MAIZE || ballColor == BLUE) {
339.            //CalibrateCounter = CalibrateCounter + 1;
340.              Serial.println("State transition from WAIT_FOR_BALL to PUT_UMICH");
341.              state = PUT_UMICH;
342.              break;
343.            }
344.            if(ballColor == RED || ballColor == WHITE) {
345.            //CalibrateCounter = CalibrateCounter + 1;
346.              Serial.println("State transition from WAIT_FOR_BALL to PUT_BALL");
347.              state = PUT_BALL;
348.              break;
349.            }
```

```
            else {
              Serial.println("State transition from WAIT_FOR_BALL to ELSE");
              state = PUT_BALL;
              break;
            }
          }
          // Otherwise we continue waiting for the ball

          break;


    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
          // 5****************************************************** PUT_BALL ******************************************************5 //
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
          // In this state, we move the cup to launch RED and WHITE balls into the NET
          case PUT_BALL:

            // ****************************************************** CHUTE 2 ************************************************************
          if (motorPosition <= (CHUTE_2_POSITION + TARGET_BAND) && motorPosition >= (CHUTE_2_POSITION - TARGET_BAND) && motorVelocity == 0) {
            // If the mechanism reached chute 2 and stopped, proceed to throwing ball
            // PID values are assigned to ensure better launch
            KP  = 0.14;   // [Volt / encoder counts] P-Gain
            KI  = 0.01;   // [Volt / (encoder counts * seconds)] I-Gain
            KD  = 0.004;  // [Volt * seconds / encoder counts] D-Gain

            Serial.println("PUT_POSITION");
            targetPosition = PUT_POSITION;
            //Since this is for the WHITE and RED case, we move to the OHIO color encoder position
          }

          // ****************************************************** CHUTE 1 ************************************************************
          if (motorPosition <= (CHUTE_1_POSITION + TARGET_BAND) && motorPosition >= (CHUTE_1_POSITION - TARGET_BAND) && motorVelocity == 0) {
            // If the mechanism reached chute 1 and stopped, proceed to move to chute 2 (this is for better accuracy)
            // PID values are assigned to ensure smooth transition to chute 2 position
            KP  = 0.14;   // [Volt / encoder counts] P-Gain
```

```
384.            KI           = 0.01;      // [Volt / (encoder counts * seconds)] I-Gain
385.            KD            = 0.006;     // [Volt * seconds / encoder counts] D-Gain
386.
387.            targetPosition = CHUTE_2_POSITION;
388.            //To ensure better accuracy throwing OHIO color balls, we move to chute 2 encoder
    position and wait for above loop to be true
389.          }
390.
391.          // Decide what to do next:
392.          if (motorPosition <= (PUT_POSITION + TARGET_BAND) && motorPosition >=
    (PUT_POSITION - TARGET_BAND) && motorVelocity == 0 ) {
393.            // We reached the basket and dropped the ball.
394.            // Transition into WAIT state to restart the cycle
395.            Serial.println("State transition from PUT_BALL to WAIT");
396.            targetPosition = WAIT_POSITION;
397.          }
398.          if (motorPosition <= (WAIT_POSITION + TARGET_BAND) && motorPosition >=
    (WAIT_POSITION - TARGET_BAND) && motorVelocity == 0) {
399.            // After reaching WAIT, we calibrate before picking up another ball
400.            state = CALIBRATE;
401.            Serial.println("State transition from PUT_BALL to CALIBRATE");
402.            break;
403.          }
404.          // Otherwise we continue moving towards the chute
405.          break;
406.
407.
408.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
409.        // 6****************************************************** PUT_UMICH
    ********************************************************6 //
410.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
411.        // In this state, we move to the correct position for placing MAIZE and BLUE balls into the
    bucket
412.        case PUT_UMICH:
413.
414.          // ****************************************************** CHUTE 2
    ************************************************************
415.          if(motorPosition <= (CHUTE_2_POSITION + TARGET_BAND) && motorPosition >=
    (CHUTE_2_POSITION - TARGET_BAND) && motorVelocity == 0)
416.          // If the mechanism reached CHUTE 2 and stopped, proceed to placing ball
417.          // PID values are assigned to ensure better placement
```

```
418.          {
419.           KP           = 0.11;      // [Volt / encoder counts] P-Gain
420.           KI           = 0.01;      // [Volt / (encoder counts * seconds)] I-Gain
421.           KD           = 0.006;     // [Volt * seconds / encoder counts] D-Gain
422.
423.           Serial.println("PUT_UMICH");
424.           targetPosition = UMICH_POSITION;
425.           // Since this is for the BLUE and MAIZE case, we move to the UMICH color encoder
    position
426.          }
427.
428.         // ********************************************************* CHUTE 1
    ************************************************************
429.          if(motorPosition <= (CHUTE_1_POSITION + TARGET_BAND) && motorPosition >=
    (CHUTE_1_POSITION - TARGET_BAND) && motorVelocity == 0)
430.           // If the mechanism reached CHUTE 1 and stopped, proceed to placing ball
431.           // PID values are assigned to ensure better placement
432.          {
433.           KP           = 0.125;     // [Volt / encoder counts] P-Gain
434.           KI           = 0.01;      // [Volt / (encoder counts * seconds)] I-Gain
435.           KD           = 0.006;     // [Volt * seconds / encoder counts] D-Gain
436.
437.           Serial.println("PUT_UMICH");
438.           targetPosition = UMICH_POSITION;
439.           // Since this is for the BLUE and MAIZE case, we move to the UMICH color encoder
    position
440.          }
441.
442.
443.         // Decide what to do next:
444.          if (motorPosition <= (UMICH_POSITION + TARGET_BAND) && motorPosition >=
    (UMICH_POSITION - TARGET_BAND) && motorVelocity == 0 )
445.          {
446.           // We reached the basket and dropped the ball.
447.           // Transition into WAIT state to restart the cycle
448.           Serial.println("State transition from PUT_BALL to WAIT");
449.           targetPosition = WAIT_POSITION;
450.          }
451.           if (motorPosition <= (WAIT_POSITION + TARGET_BAND) && motorPosition >=
    (WAIT_POSITION - TARGET_BAND) && motorVelocity == 0)
452.          {
453.           // After reaching WAIT, we calibrate before picking up another ball
454.           state = CALIBRATE;
```

```
455.            Serial.println("State transition from PUT_BALL to CALIBRATE");
456.            break;
457.          }
458.          // Otherwise we continue moving towards the chute
459.
460.        break;
461.
462.
    //*************************************************************************//
463.        // We should never reach the next bit of code, which would mean that the state
464.        // we are currently in doesn't exist.  So if it happens, throw an error and
465.        // stop the program:
466.        default:
467.          Serial.println("Statemachine reached at state that it cannot handle.  ABORT!!!!");
468.          Serial.print("Found the following unknown state: ");
469.          Serial.println(state);
470.          while (1); // infinite loop to halt the program
471.        break;
472.
473.      }
474.      // End of the state machine.
475.
    //*************************************************************************
    ************************************************//
476.
477.
478.
    //*************************************************************************
    ************************************************//
479.      // Position Controller
480.      if (digitalRead(PIN_NR_ON_OFF_SWITCH)==HIGH) {
481.        // If the toggle switch is on, run the controller:
482.
483.        //** PID control: **//
484.        // Compute the position error [encoder counts]
485.        positionError = targetPosition - motorPosition;
486.        // Compute the integral of the position error  [encoder counts * seconds]
487.        integralError = integralError + positionError * (float)(executionDuration) / 1000000;
488.        // Compute the velocity error (desired velocity is 0) [encoder counts / seconds]
489.        velocityError = 0 - motorVelocity;
490.        // This is the actual controller function that uses the error in
491.        // position and velocity and the integrated error and computes a
492.        // desired voltage that should be sent to the motor:
```

```
493.        desiredVoltage = KP * positionError +
494.                         KI * integralError +
495.                         KD * velocityError;
496.
497.        //** Feedforward terms: **//
498.        // Compensate for friction.  That is, if we now the direction of
499.        // desired motion, add a base command that helps with moving in this
500.        // direction:
501.        if (positionError < -5) {
502.          desiredVoltage = desiredVoltage - BASE_CMD;
503.        }
504.        if (positionError > +5) {
505.          desiredVoltage = desiredVoltage + BASE_CMD;
506.        }
507.        // Gravity compensation lookup.  Here we record which voltage we need
508.        // to keep the device balanced at the left and at the right, and note
509.        // where it is balanced passively.  The feedforward value is determined
510.        // by linear interpolation between these three points.
511.        if (motorPosition<FF_BALANCED_POSITION) {
512.           desiredVoltage = desiredVoltage +
    (FF_BALANCED_POSITION-motorPosition)/(FF_BALANCED_POSITION-LOWER_BOUN
    D)*FF_VOLTAGE_LOWER_BOUND;
513.        }
514.        if (motorPosition>FF_BALANCED_POSITION) {
515.           desiredVoltage = desiredVoltage +
    (motorPosition-FF_BALANCED_POSITION)/(UPPER_BOUND-FF_BALANCED_POSITION
    )*FF_VOLTAGE_UPPER_BOUND;
516.        }
517.
518.        // Anti-Wind-Up
519.        if (abs(desiredVoltage)>SUPPLY_VOLTAGE) {
520.          // If we are already saturating our output voltage, it does not make
521.          // sense to keep integrating the error (and thus ask for even higher
522.          // and higher output voltages).  Instead, stop the integrator if the
523.          // output saturates. We do this by reversing the summation at the
524.          // beginning of this function block:
525.          integralError = integralError - positionError * (float)(executionDuration) / 1000000;
526.        }
527.        // End of 'if(onOffSwitch==HIGH)'
528.
529.        // Override the computed voltage during calibration.  In this state, we simply apply a
530.        // fixed voltage to move against one of the end-stops.
531.        if (state==CALIBRATE) {
```

```
      desiredVoltage = CALIBRATION_VOLTAGE; // add calibration code here
    }
  } else {
    // Otherwise, the toggle switch is off, so do not run the controller,
    // stop the motor...
    desiredVoltage = 0;
    // .. and reset the integrator of the error:
    integralError = 0;
    // Produce some debugging output:
    Serial.println("The toggle switch is off.  Motor Stopped.");
  }
  // End of  else onOffSwitch==HIGH

  //** Send signal to motor **//
  // Convert from voltage to PWM cycle:
  motorCommand = int(abs(desiredVoltage * 255 / SUPPLY_VOLTAGE));
  // Clip values larger than 255
  if (motorCommand > 255) {
    motorCommand = 255;
  }
  // Send motor signals out
  analogWrite(PIN_NR_PWM_OUTPUT, motorCommand);
  // Determine rotation direction
  if (desiredVoltage >= 0) {
    // If voltage is positive ...
    // ... turn forward
    digitalWrite(PIN_NR_PWM_DIRECTION_1,LOW);  // rotate forward
    digitalWrite(PIN_NR_PWM_DIRECTION_2,HIGH); // rotate forward
  } else {
    // ... otherwise turn backward:
    digitalWrite(PIN_NR_PWM_DIRECTION_1,HIGH); // rotate backward
    digitalWrite(PIN_NR_PWM_DIRECTION_2,LOW);  // rotate backward
  }
  // End of Position Controller
  //*****************************************************************//

  // Print out current controller state to Serial Monitor.
  printStateToSerial();
}
// End of main loop
//*********************************************************************//
```

```
575.    ////////////////////////////////////////////////////////////////////
576.    // This is a function that returns the type of ball found in the    //
577.    // cup.  It is called from the loop()-routine.  It returns one of   //
578.    // the following values:                                            //
579.    // 'MAIZE', 'BLUE', 'RED', 'WHITE', 'NONE'.                         //
580.    ////////////////////////////////////////////////////////////////////
581.
582.    int evaluateColorSensor() {
583.    // The ball sensor evaluation will read two values, then compare them to make a more
       accurate color assessment
584.    // It will also continue to read until it times out, then will guess the ball as a UMICH color
       and proceed
585.
586.      // initialize ball type with 'NONE'.  Override later if a ball color was detected.
587.      int ballType = NONE;
588.
589.      // ********************************************** 1ST COLOR READING
       ********************************************** //
590.      tcs.setInterrupt(false);      // turn on LED
591.      delay(50);                    // Takes 0.05s to turn on the LED and stablize it
592.      tcs.getRawData(&red, &green, &blue, &clear);
593.      tcs.setInterrupt(true);       // turn off LED
594.
595.      // Check if the ball is MAIZE
596.      if ((red>7500) && (red <12000) && (green>7500) && (green <13000)&& (blue>3500)
       && (blue <7000) && (clear>18000)){
597.        ballType = MAIZE;
598.      }
599.      // Check if the ball is BLUE
600.      if ((red>1000) && (red <2500) && (green>2500) && (green < 3800)&& (blue>4200) &&
       (blue <6500) && (clear>8000)){
601.        ballType = BLUE;
602.      }
603.      // Check if the ball is RED
604.      if ((red>1950) && (red <4200) && (green>1600) && (green <3000)&& (blue>1600) &&
       (blue <3000) && (clear>5500)){
605.        ballType = RED;
606.      }
607.      // Check if the ball is WHITE
608.      if ((red>8000) && (red <20000) && (green>10000) && (green <20000)&& (blue>10000)
       && (blue <20000) && (clear>15000) ){
609.        ballType = WHITE;
610.      }
```

```
611.
612.       // ********************************************** 2ND COLOR READING
    ********************************************** //
613.       tcs.setInterrupt(false);          // turn on LED
614.       delay(10);                        // Set a small delay between reading the first and second value
615.       tcs.getRawData(&red, &green, &blue, &clear);
616.       tcs.setInterrupt(true);           // turn off LED
617.       int ballType2 = NONE;             // Initialize the second ballType to compare to the first
618.
619.       // Check if the ball is MAIZE
620.       if ((red>7500) && (red <12000) && (green>7500) && (green <13000)&& (blue>3500)
    && (blue <7000) && (clear>18000)){
621.         ballType2 = MAIZE;
622.       }
623.       // Check if the ball is BLUE
624.       if ((red>1000) && (red <2500) && (green>2500) && (green < 3800)&& (blue>4200) &&
    (blue <6500) && (clear>8000)){
625.         ballType2 = BLUE;
626.       }
627.       // Check if the ball is RED
628.       if ((red>1950) && (red <4200) && (green>1600) && (green <3000)&& (blue>1600) &&
    (blue <3000) && (clear>5300)){
629.         ballType2 = RED;
630.       }
631.       // Check if the ball is WHITE
632.       if ((red>8000) && (red <20000) && (green>10000) && (green <20000)&& (blue>10000)
    && (blue <20000) && (clear>15000) ){
633.         ballType2 = WHITE;
634.
635.       }
636.
637.       // If the program struggles to read a value, It will begin to spit out the values it reads onto
    serial monitor
638.       // This is for debugging purposes to adjust ball color values
639.       Serial.print(F("Raw R:"));
640.       Serial.print(red);
641.       Serial.print(F(" G:"));
642.       Serial.print(green);
643.       Serial.print(F(" B:"));
644.       Serial.print(blue);
645.       Serial.print(F(" C:"));
646.       Serial.println(clear);
647.
```

```
648.      // A counter that was initialized earlier, every time the loop reads another color value it adds
      to this value
649.      COUNTERC = COUNTERC + 1;
650.      Serial.print(COUNTERC);
651.
652.      // ************************************** COLOR COMPARISON
      ************************************************** //
653.      // Code that compares the two ball type colors and will return their value if they are the
      same.
654.      // NOTE: Two NONES will continue running the loop. This ensures the color sensor outputs
      an actual color.
655.      if(ballType2 == ballType && ballType != 5)
656.      {
657.        return ballType;
658.      }
659.
660.      // If the ball color is still NONE, continue running the loop.
661.      else
662.
663.        if(COUNTERC >= 15)
664.        // If the colors are measured 15 times and there is still no value, assign it as a UMICH ball
      and break the loop.
665.        // This protects against balls being dropped or a ball outside of the color range (usually
      blues and yellows).
666.        {
667.          ballType = 1;
668.          return ballType;
669.        }
670.
671.      // Keep running the color sensor evaluation until above If statement is true
672.      return evaluateColorSensor() ;
673.    }
674.
675.    // End of function evaluateColorSensor()
676.
677.
678.    ////////////////////////////////////////////////////////////////////
679.    // This is a function to update the encoder count in the Arduino.   //
680.    // It is called via an interrupt whenever the value on encoder      //
681.    // channel A or B changes.                                          //
682.    ////////////////////////////////////////////////////////////////////
683.    void updateMotorPosition() {
684.      // Bitwise shift left by one bit, to make room for a bit of new data:
```

```
685.      encoderStatus <<= 1;
686.      // Use a compound bitwise OR operator (|=) to read the A channel of the encoder (pin 2)
687.      // and put that value into the rightmost bit of encoderStatus:
688.      encoderStatus |= digitalRead(2);
689.      // Bitwise shift left by one bit, to make room for a bit of new data:
690.      encoderStatus <<= 1;
691.      // Use a compound bitwise OR operator  (|=) to read the B channel of the encoder (pin 3)
692.      // and put that value into the rightmost bit of encoderStatus:
693.      encoderStatus |= digitalRead(3);
694.      // encoderStatus is truncated to only contain the rightmost 4 bits by  using a
695.      // bitwise AND operator on mstatus and 15(=1111):
696.      encoderStatus &= 15;
697.      if (encoderStatus==2 || encoderStatus==4 || encoderStatus==11 || encoderStatus==13) {
698.        // the encoder status matches a bit pattern that requires counting up by one
699.        motorPosition++;         // increase the encoder count by one
700.      }
701.      else if (encoderStatus == 1 || encoderStatus == 7 || encoderStatus == 8 || encoderStatus ==
    14) {
702.        // the encoder status does not match a bit pattern that requires counting up by one.
703.        // Since this function is only called if something has changed, we have to count downwards
704.        motorPosition--;         // decrease the encoder count by one
705.      }
706.    }
707.    // End of function updateMotorPosition()
708.
709.
710.    ////////////////////////////////////////////////////////////////////
711.    // This function sends a status of the controller to the serial     //
712.    // monitor.  Each character will take 85 microseconds to send, so   //
713.    // be selective in what you write out:                              //
714.    ////////////////////////////////////////////////////////////////////
715.    void printStateToSerial() {
716.      //*****************************************************************//
717.      // Send a status of the controller to the serial monitor.
718.      // Each character will take 85 microseconds to send, so be selective
719.      // in what you write out:
720.
721.      //Serial.print("State Number:  [CALIBRATE = 1; WAIT = 2; MOVE_TO_CHUTE = 3;
    WAIT_FOR_BALL = 4; PUT_BALL = 5]: ");
722.      Serial.print("State#: ");
723.      Serial.print(state);
724.
725.      //Serial.print("Power switch [on/off]: ");
```

```
726.      //Serial.print("  PWR: ");
727.      //Serial.print(digitalRead(PIN_NR_ON_OFF_SWITCH));
728.
729.      //Serial.print("      Motor Position [encoder counts]: ");
730.      Serial.print("  MP: ");
731.      Serial.print(motorPosition);
732.
733.      //Serial.print("      Motor Velocity [encoder counts / seconds]: ");
734.      Serial.print("  MV: ");
735.      Serial.print(motorVelocity);
736.
737.      //Serial.print("      Encoder Status [4 bit value]: ");
738.      //Serial.print("  ES: ");
739.      //Serial.print(encoderStatus);
740.
741.      //Serial.print("      Target Position [encoder counts]: ");
742.      Serial.print("  TP: ");
743.      Serial.print(targetPosition);
744.
745.      //Serial.print("      Position Error [encoder counts]: ");
746.      Serial.print("  PE: ");
747.      Serial.print(positionError);
748.
749.      //Serial.print("      Integrated Error [encoder counts * seconds]: ");
750.      Serial.print("  IE: ");
751.      Serial.print(integralError);
752.
753.      //Serial.print("      Velocity Error [encoder counts / seconds]: ");
754.      Serial.print("  VE: ");
755.      Serial.print(velocityError);
756.
757.      //Serial.print("      Desired Output Voltage [Volt]: ");
758.      Serial.print("  DV: ");
759.      Serial.print(desiredVoltage);
760.
761.
762.
763.      //Serial.print("      Motor Command [0-255]: ");
764.      //Serial.print("  MC: ");
765.      //Serial.print(motorCommand);
766.
767.      //Serial.print("      Execution Duration [microseconds]: ");
768.      //Serial.print("  ED: ");
```

```
769.       //Serial.print(executionDuration);
770.
771.       //Serial.print("      Raw signals from the color sensor: ");
772.       //Serial.print("  R: ");
773.       //Serial.print(red);
774.       //Serial.print("  G: ");
775.       //Serial.print(green);
776.       //Serial.print("  B: ");
777.       //Serial.print(blue);
778.       //Serial.print("  C: ");
779.       //Serial.print(clear);
780.
781.      // ALWAYS END WITH A NEWLINE.  SERIAL MONITOR WILL CRASH IF NOT
782.       Serial.println(); // new line
783.     }
784.     // End of Serial Out
```

Calculations:

We did not do any calculations to determine the encoder count that led to our desired position. This is because each of the test boards is slightly different and requires a different number of encoder counts to reach desired positions. Calculating the ideal count is pointless since the actual count needed varies greatly between the testing environments and the final testing board.

Bill of Materials:

Part Number	Part Name	Material	Dimension(s)	Supplier	Quantity	Price (per item)	Notes
1	Motor Gearmotor	Steel	37Dx52L mm	Kit	1	----	
2	Arduino	N/A	4” x 2.1”	Kit	1	----	
3	Breadboard	Plastic and metal	2.2” x 3.5”	Kit	1	----	
4	H-Bridge	N/A	1.5” x 1.5”	Kit	1	----	
5	Fuse	N/A	6” x 1”	Kit	1	----	
6	Toggle Switch	N/A	1” x .5”	Kit	1	----	
7	Limit Switch	N/S	1” x .75”	Kit	1	----	