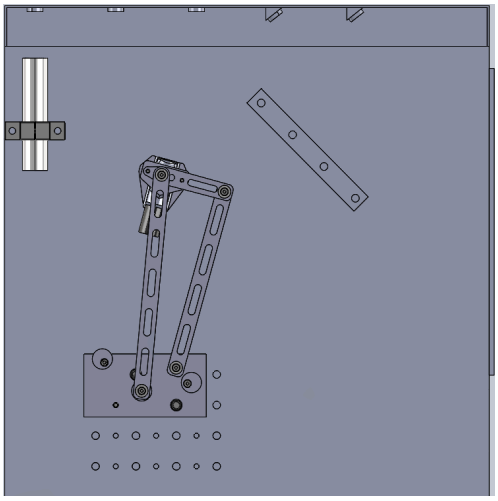
ME350
Winter 2024 Semester
FINAL REPORT

Team 64

Ethan Lei
Luis Barcenas
Mingyao Tao
Samuel Wasserfall

GSI: Revanth Damerla



"We have fully abided by the University of Michigan College of Engineering Honor Code"
Ethan Lei
Luis Barcenas
Mingyao Tao
Samuel Wasserfal

_____

Section 1: Design Goals, Requirements, and Specifications

Goals

The overarching goal of this project is to design, build, and test a powered mechanism that will automatically sense targets that pop up and shine light on their corresponding photosensors. The project playing field consists of five targets, each above a photosensor. Two of the photosensors are angled toward a mirror that is located along one wall of the playing field because there is an aluminum block that prevents light from directly hitting the sensors. Another obstacle is a tube located in front of the leftmost photosensor. This severely limits the angle at which you can successfully shine light on this photosensor. The mechanism we build is intended to sense when the targets pop up, and shine light on the corresponding photosensor as quickly as possible. To this end, the general design goals for this mechanism include minimizing the ideal transmission angle deviation, minimizing the weight of all moving parts, and maximizing the intensity of light/amount of time that light is able to reach the targets.

Requirements

Requirement	Specification	Connection to Goals
Transmission Deviation Angle	|Transmission Angle| - 90 < 30 [Minimize] and |Transmission Angle| - 90 = 60	This requirement is meant to prevent undesirable linkage behavior, which would negatively impact our mechanisms ability to meet the goal of accurately shining light at the targets
Manufacturing	Minimize time needed to manufacture parts 20 minutes [Minimize] 30 minutes [Max Waterjet Time]	Important to ensure that we can use the available waterjet.
Angle of Incidence	Maintain an angle of incidence close to directly perpendicular to each of the sensors 30 degrees from perpendicular to 45 degrees from perpendicular	Allows us to maximize our available light and its effectiveness on the photosensors.
Robustness	Design parts with ample yield	Important to the overall

	strength and stiffness No failure/broken parts	ability to perform other specifications.
Linkage Mounting/Hard Stops	Create a mounting component with hard stops capable of limiting the range of motion to the desired values	Useful to our goal of shining light on photosensors by allowing for movement calibration.
Flashlight Mounting System Weight	Design a mounting system for the flashlight which is lightweight and reliable 0.5 pounds 1 pound	Necessary to perform our objective of shining light on a photosensor as our only source of focused light.

Table 1.1 Summary of project requirements and specifications

Section 2: Design Selection

Results	Ethan	Luis	Mingyao	Samuel
1W Power Time (s)	0.57	0.66	0.87	0.81
Transmission Angle Deviation Average	14.02°	8.476°	23.9	18.56°
Length of Input Link	10.63”	10.68”	14.40”	14.75”
Length of Coupler	3.22”	4.25”	2.19”	2.47”
Length of Follower Link	8.95”	11.90”	15.40”	13.42”
Estimated Mass (g)	370	366	398	413

Table 2.1 Individual linkage design parameters are provided.

Pugh Chart

Requirement	Weight	Ethan	Luis	Mingyao	Samuel
Power	3	3	2	2	2
Mass	2	2	2	1	1
Transmission Angle Deviation	3	2	3	1	2
Angle of Impact of Laser on Photosensor	2	2	1	2	2
Total		9	8	6	7

Table 2.2 A Pugh chart comparing power, mass, transmission angle deviation, and angle of laser impact among individual linkage designs.

Pugh Chart Explanation
Weights are either 2 or 3. For categories with a weight of two; 1 represents less than average, 2 average or above average. For categories with a weight of three; 1 represents less than average, 2 represents average, and 3 represents above average. Averages are based on the sample of our team.

Ethan's design was selected due to the shortest 1W power time obtained and an overall adequate performance on mass, transmission angle deviation and angle of impact of laser on photosensor. While taking only 0.57 seconds to finish all the moves using less than 1W of power, the transmission angle deviation is much smaller than lab 1's requirement of 30 degrees from 90 (only 14.02 on average). Also the design's link's length is on average optimal not only for mass saving but also for manufacturing, since the longer the links, and the more complicated the geometry, the longer and harder to produce. The links are neither too short, decreasing light intensity by much, or too long to affect manufacturing. Overall Ethan's design trumped other team members' and going forward we are going to use his design and geometry to produce our team linkage's mechanical section.

While Luis' transmission angle deviation was smaller, the longer link lengths seemed to contribute to a slower 1W power time. The higher transmission angle deviations of Mingyao's and Samuel's linkages also seemed to lead to slower 1W power times. Mass was estimated through MSC ADAMS resulting in Ethan and Luis's design to have similar weight, with Mingyao and Samuel's design being slightly heavier. The long and wide links of the later 2 designs are the main reason for their heaviness. Angles of impact were roughly the same across all designs except Luis's which had inaccurate angles of impact.

If we could go back and change an aspect of our design selection, we would have made edits to our Pugh Chart. First off, we would take into account other factors that we overlooked, such as manufacturability and motion resolution. Additionally, we were missing a baseline column in our Pugh chart establishing the normal design that our designs were better or worse than. We also would not use arbitrary scoring in our Pugh chart considering the factors we chose were quantifiable. We would find the average of each value, make that the baseline, and grade designs based on that baseline.

Section 3: Final Linkage Design and Analysis

Input Link Length	10.63”
Coupler Length	3.22”
Follower Length	8.95”
Target 1 angle deviation	24.86º
Target 2 angle deviation	11.09º
Target 3 angle deviation	0.71º
Target 4 angle deviation	16.83º
Target 5 angle deviation	16.62º
Ground pivot 1 Location(x,y) relative to bottom left corner	(6.79, 5.2)
Ground pivot 2 Location(x,y) relative to bottom left corner	(8.49 , 6.3)

Table 3.1 Linkage parameters of final design

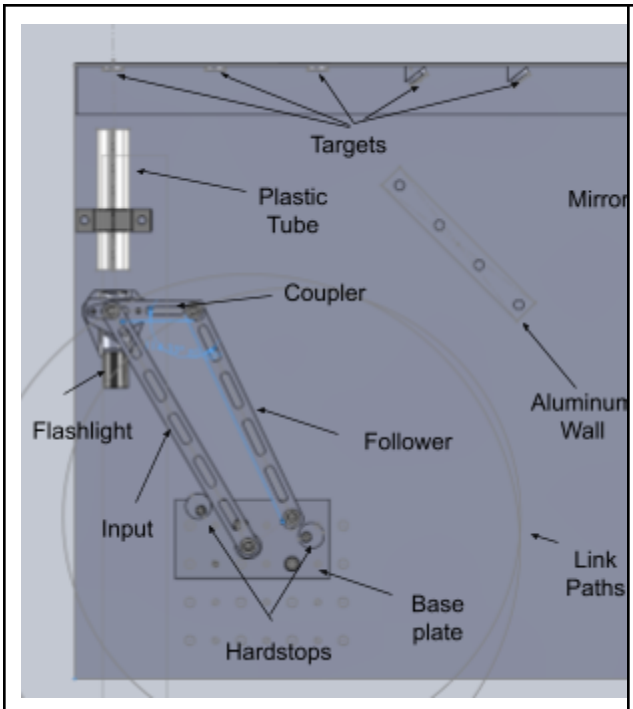


Fig 3.1a. Linkage over Target #1 (beginning

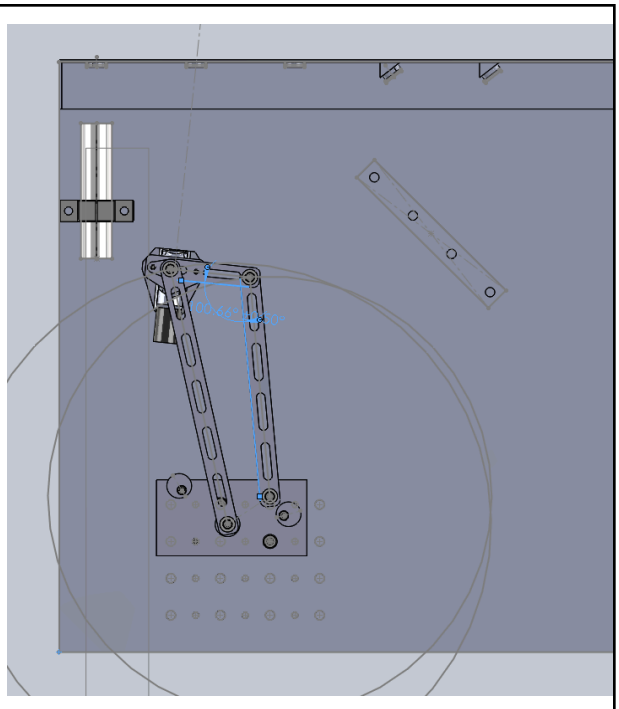


Fig 3.1b. Linkage over Target #2

of motion)	

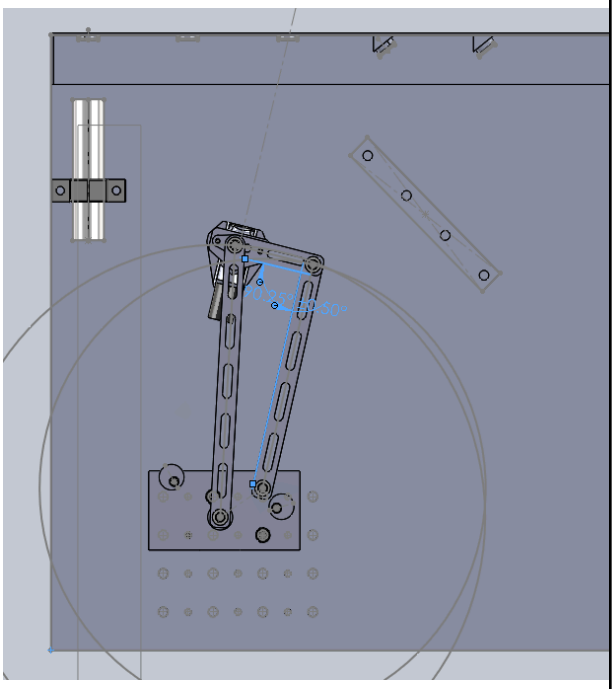
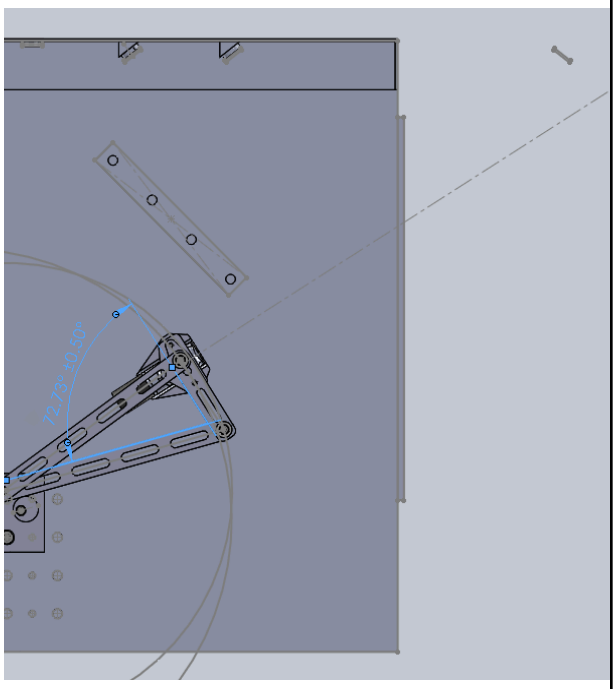

	

Fig 3.1c. Linkage over Target #3	**Fig 3.1d. Linkage over Target #4 (end of motion)**

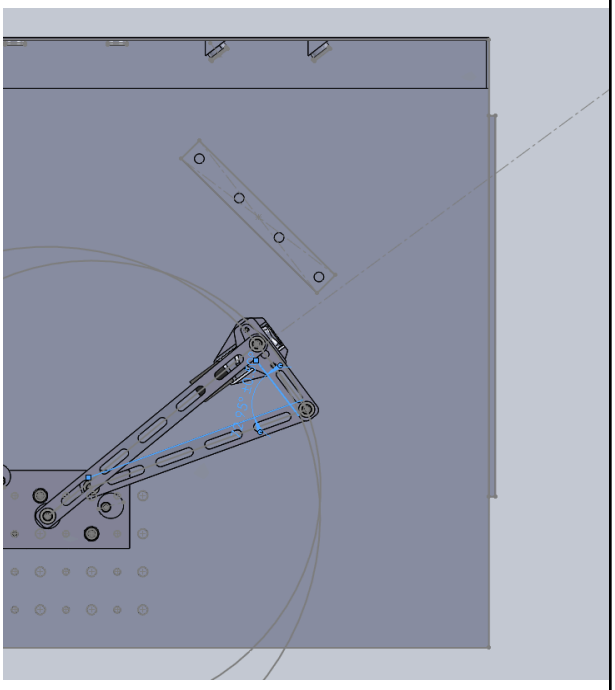
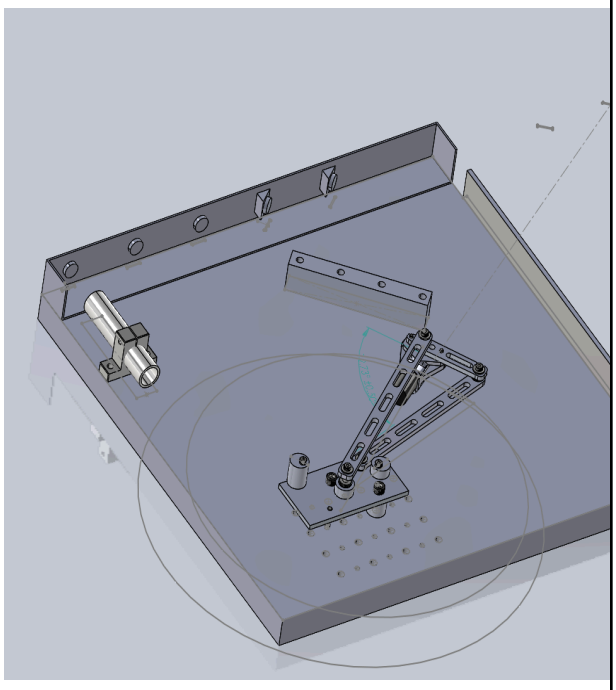

	

Fig 3.1e. Linkage over Target #5	**Fig 3.1f. Perspective View of Mechanism**

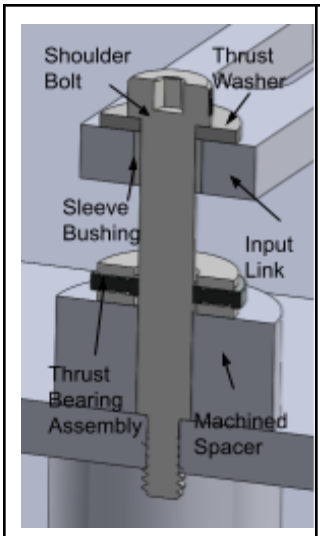


Fig 3.2a. Ground–Input Joint

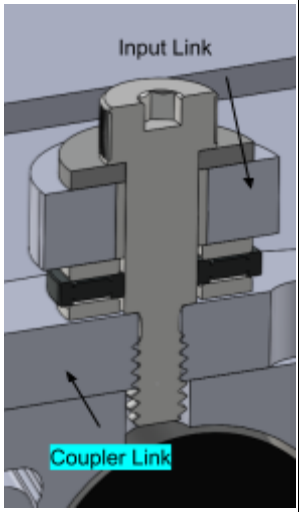


Fig 3.2b. Input–Coupler Joint

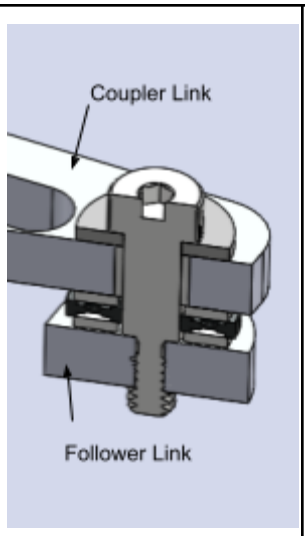


Fig 3.2c. Coupler–Follower Joint

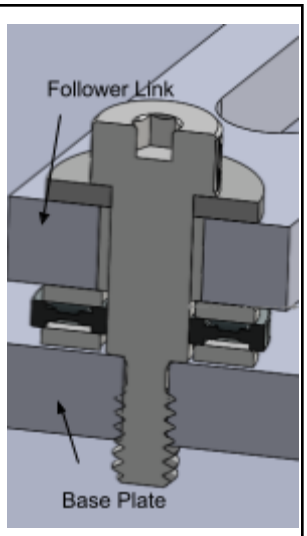


Fig 3.2d. Follower–Ground Joint

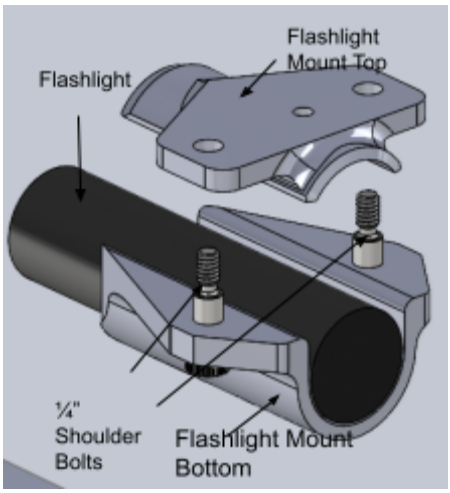


Fig 3.3a. Flashlight Mount

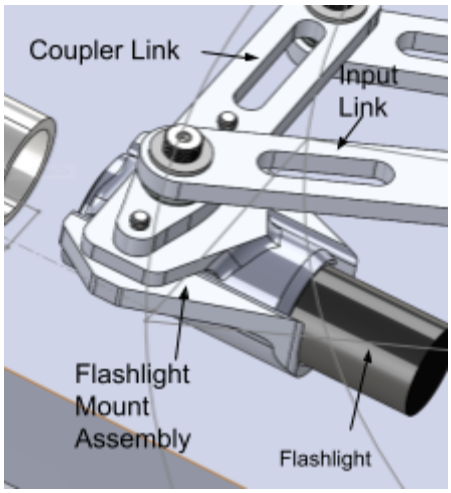


Fig 3.3b. Flashlight Mount to Coupler

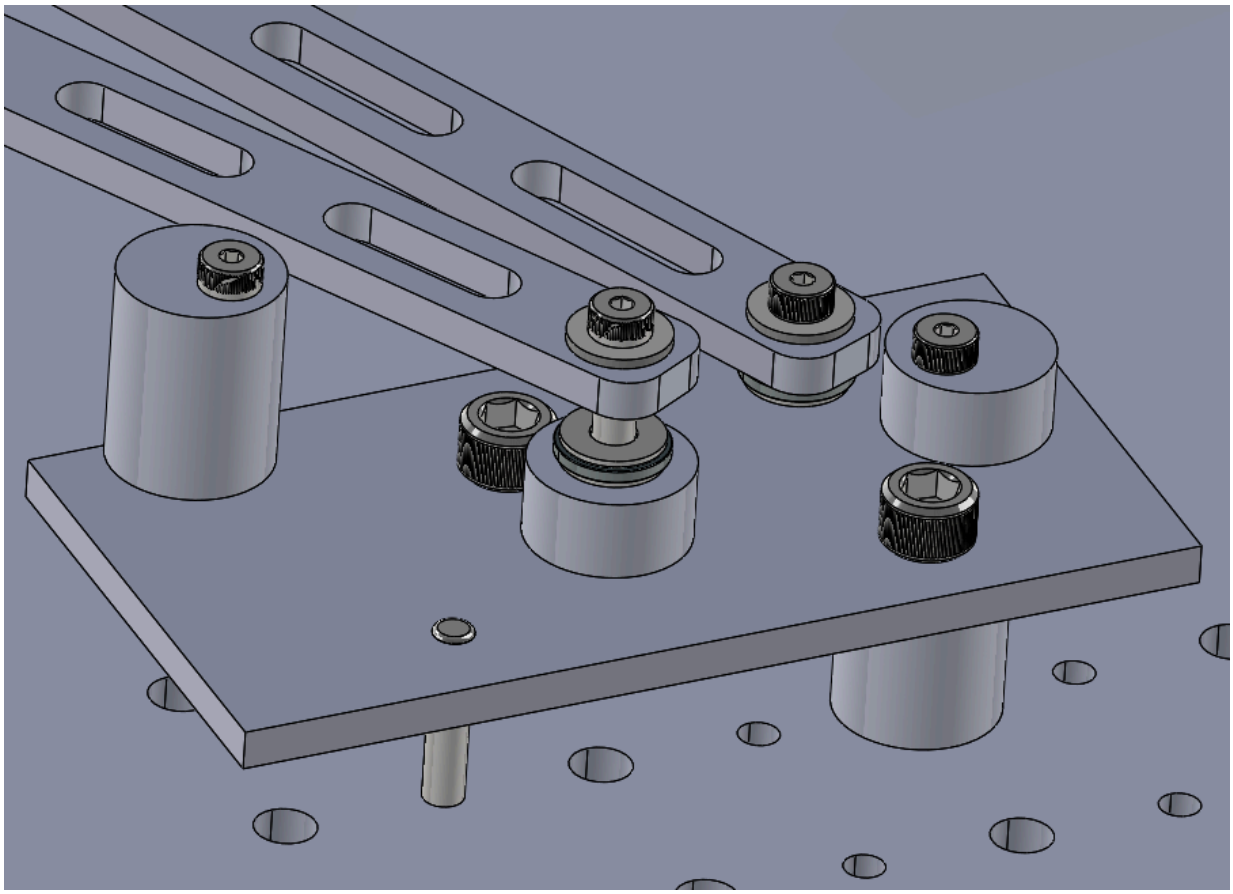


Fig 3.4. Base Plate, Spacers, and Hard Stops

Important Features

There are a number of important features included in our linkage design other than the links/joints themselves.

One of these features is the flashlight mount (shown in Fig 3.3), which is meant to constrain the flashlight in place as the linkage completes its motion. It is a two part clamp design that has through-holes for shoulder bolts to attach the mount to the coupler via tapped holes. The mount is going to be 3D-printed and the current intention is to use the new SLA printer in the x50 Machine Shop. 3D printing allows us to create a lightweight and efficient part due to both the material properties or plastic or resin compared to metal and the smaller number of manufacturing constraints compared to conventional machining.

Another important feature is the spacers below the base plate which are designed with the linkage to set the flashlight at the correct height relative to the light sensors. This will be machined 1” aluminum round stock with a clearance hole for a bolt to go through the base plate.

The base plate itself is an important component of the linkage design as it serves as the mounting point to the playing field and determines the overall positioning of where the flashlight

will end up. A quarter inch thick aluminum plate will be used as the stock for this part. The holes will be machined and tapped, and the profile of the base plate will be cut with the water jet. A more optimized profile will be selected once the motor mounting is finalized.

There is also a spacer meant to properly locate the height of the input link, however, as our transmission has not been finalized, the spacing is not currently known. Similar to the other spacers, 1” aluminum round stock will be used to create this part and a clearance hole will be machined in the center.

The hardstops are the final important feature of our design and these provide a stopping point at the each end of our linkage's desired range of motion. These will be machined out of 1” aluminum round stock and an off-centered clearance hole will be machined in the part to allow for some adjustability of the linkage’s range of motion. The off-centered hole can act as a pivot point for the part, which will slightly change the contact point on the input/follower links depending on the orientation which the hard stops are bolted down in.

The base plate, spacers, and hardstops are all shown in Figure 3.4.

Joint Design

Each of the four joints in this design make use of a 1/4" shoulder bolt, a thrust washer, a thrust bearing, a sleeve bushing, and two “regular” washers. This can be seen clearly in Figure 2 for each of the four joints. In each two-part connection, one of the links contains a sleeve bushing while the other is tapped. The tapped hole eliminates the need for an external nut. Each of the other components of the joint is present to support the rotational motion between the links. The thrust washer’s purpose is to reduce the friction between the bolt head and the link, while the other washers are present to distribute the load of the links more evenly onto the needle rollers of the thrust bearing, and the thrust bearing itself has a large impact in enabling the links to glide past each other even with the significant axial load being applied from one link to the other. The length of the shoulder is designed to be equal to or very slightly greater than the distance from the top face of the lower link to the top face of the thrust bearing above the top link. This design choice means that it will be impossible to overtighten the linkage assembly, reducing the potential issues, such as binding, that could occur during testing. The actual spacing will vary based on the tolerances of the components involved in each joint, but the tolerance stack-up analysis shows that the joint is unlikely to have issues due to overtightening.

Weight estimate

The estimated mass of the assembly (excluding base plate, motor, and other control components) is 370g. Within the linkage, the input link and the follower are estimated to be 73.77g and 61.7g respectively. The coupler and the flashlight mount together are estimated to be 142.74g. The flashlight is estimated to be 70g. The mass data is based on the assumption that the links will be made out of 6061 Aluminum alloy and the flashlight mount is made out of PLA or other plastic with density close to 1.3g/cm^3. Since McMaster does not list the mass data for our fasteners and washers, we estimate that each washer, thrust bearing, bushing, and

nut would weigh around 1g, since they are relatively small. And each shoulder bolt should weigh approximately 2g to 4g. Therefore adding the mass of all components, plus a buffer (in order to make sure we accommodate for the many potential errors we introduced with our many assumptions, and give a conservative estimate for better motor performance), the final mass estimation arrives at around 370g for the linkage.

Suboptimal Design

It is difficult for any design to be truly optimal, but one concern that we have is related to the lengths of the links. It seems possible that the weight and large moment created by the linkage will cause the flashlight to be at a shorter height than intended, which may impact performance. A potential solution for this would be to create larger spacers for the base plate or use washers to shim the assembly higher to make up for any "slop." It's also possible that the links could have smaller wall thicknesses and still be effective, but for the sake of reliability, we chose to keep the thicknesses as they are.

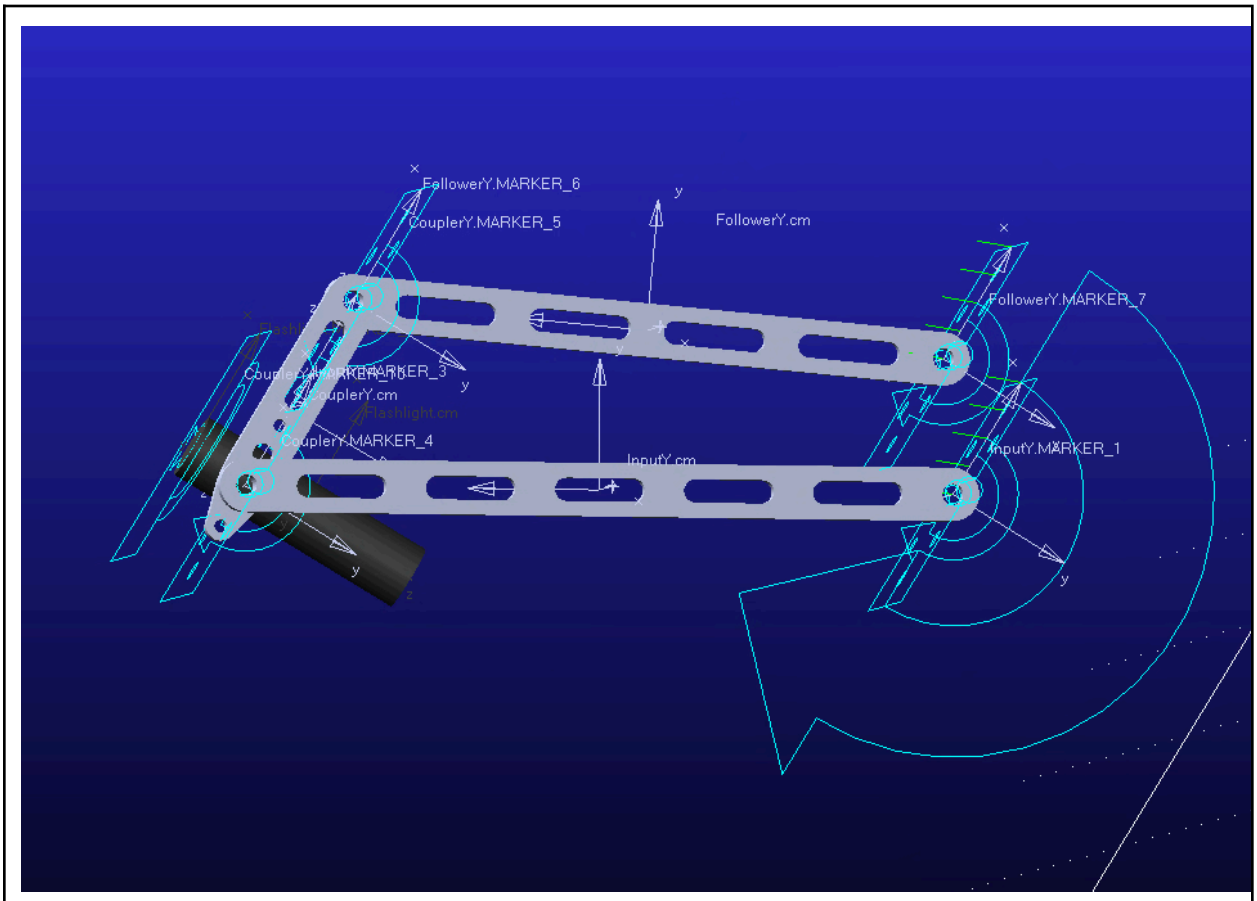


Fig 3.5. Final CAD Model in Adams

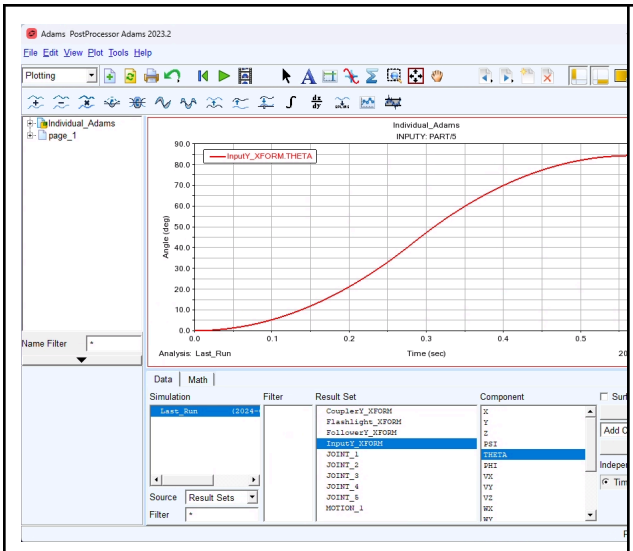


Fig 3.6a. Angular Position plotted against time, from 0 to 84 degrees

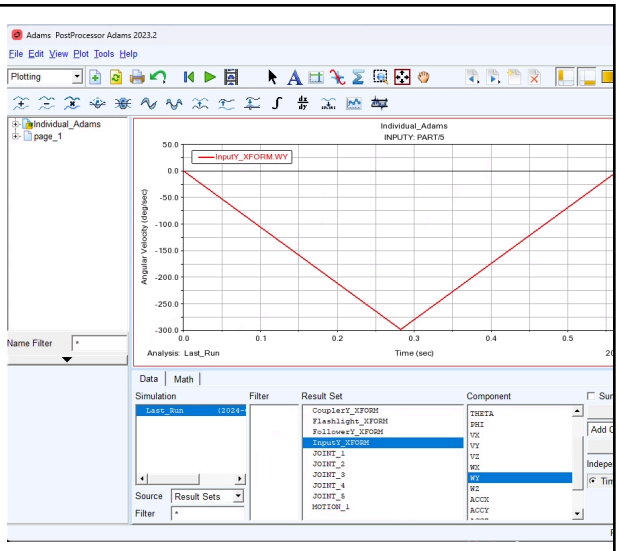


Fig 3.6b. Angular Velocity, plotted against time with a maximum magnitude halfway through the motion

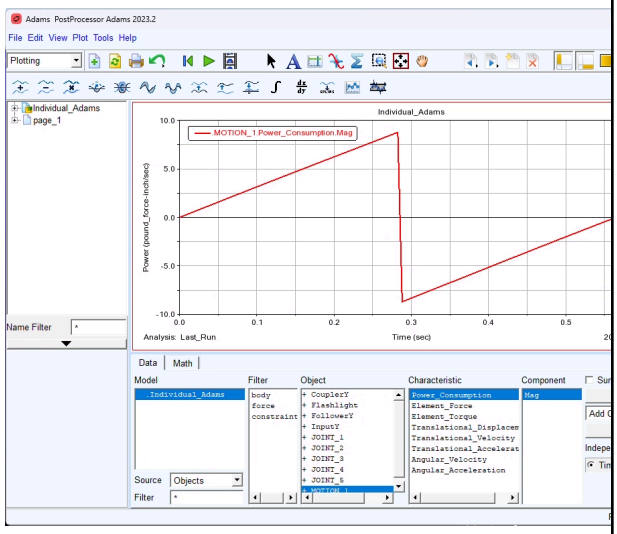


Fig 3.6c. Power Graph, maximum at 0.99 W

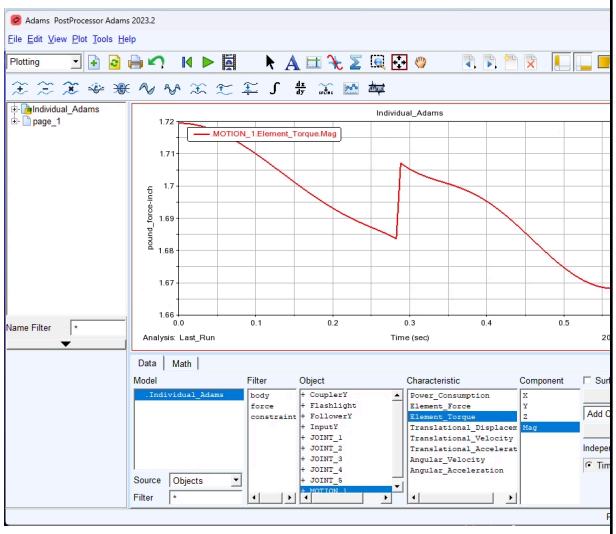


Fig 3.6d. Input Torque, steadily decreasing with a jump at the maximum angular velocity

Faults in ADAMS

Looking back on our analysis of the ADAMS models, we should have taken the values we got for our design and compared them to the specifications of the motor we are using for the project to decide whether our design meets the requirements. Knowing that now, our design does meet the requirements.

Acceleration Profile

The acceleration profile was chosen for the simulations by integrating a triangular velocity profile, as seen in Lab 3. An acceleration magnitude was found by inputting our linkage range of motion in degrees and an iteratively chosen total travel time value into the acceleration profile. The iteration was completed by running simulations in Adams which used varying acceleration magnitudes and travel times and comparing the maximum power out values necessary to achieve each acceleration magnitude. The travel time was optimized for a power out of 1 W. The maximum power out value can be seen as the highest peak in the power graph shown above.

$$\theta = 2\times\frac{1}{2}\alpha\left(\frac{t}{2}\right)^2$$

Equation 3.1 Angle traveled, where θ is the motion range, t is the total travel time, and α is the acceleration magnitude

$$\alpha = \frac{4\theta}{t^2} = \frac{4\times35}{1^2} = 140\ degrees/sec^2$$

Equation 3.2 Angular acceleration, derived from equation 3.1

The profiles shown in the graphs of Figure 3.6 are representative of an acceleration profile which consists of a phase of constant acceleration followed by a phase of constant deceleration where the two phases have the same magnitude, for the same duration of time, but in opposite directions.

Section 4: Motion Generator Revision

There were no major changes made to the motion generator portion of the system. The only change made at all was the moving of one of the base plate bolts into the playing ground, however, this does not impact the motion of the linkage. For context, the base plate bolt was moved because it would have been in the way of the transmission design.

Power Analysis

$$T_L = \frac{4 \cdot \theta \cdot J}{(T_{motion})^2}$$

$$\omega_{max,\,load} = \frac{2 \cdot \theta}{T_{motion}}$$

$$\theta = 84.23^{\circ} = 1.47\,rads,\; J = \frac{146814.46}{3.68^2} + 25000 = 35,841.1\,g * cm^2, n = 3.68$$

$$T_{nom} = 14\,kg \cdot cm,\; V_{nom} = 12\,V$$

$$K = \frac{T_{nom}}{\omega_{no-load}} = \frac{14\,kg{\cdot}cm}{330\,rpm} = 0.0424\,\frac{kg \cdot cm}{rpm} = 0.405\frac{kg \cdot cm}{rad/s}$$

$$V_S = 10\,V$$

$$\frac{T_L}{n} = \frac{T_{nom} \cdot V_S}{V_{nom}} - K \cdot n \cdot \omega_{max,\,load}$$

$$\frac{4 \cdot \theta \cdot J}{n \cdot (T_{motion})^2} = \frac{T_{nom} \cdot V_S}{V_{nom}} - K \cdot n \cdot \frac{2 \cdot \theta}{T_{motion}}$$

$$\frac{4 \cdot 1.47\,(rads) \cdot 35{,}841.1\,(g{\cdot}cm^2)}{3.68 \cdot (T_{motion})^2} = \frac{14\,(kg{\cdot}cm) \cdot 10\,V}{12\,V} - 0.405\,(\frac{kg \cdot cm}{rad/s}) \cdot 3.68 \cdot \frac{2 \cdot 1.47\,(rad)}{T_{motion}}$$

$$\frac{57.26784\,(kg{\cdot}cm^2)}{(T_{motion})^2} = 11.667\,(kg \cdot cm) - \frac{4.382\,(kg{\cdot}cm{\cdot}s)}{T_{motion}}$$

$$T_{motion} = 2.411\,s$$

This time value (2.411 seconds) is a decent bit higher than the 1W power time of 0.57 seconds derived from our original ADAMS model. There are a few likely reasons to explain the discrepancy. First, the calculation shown above assumes a constant total inertia (representative of the highest instance of inertia) of the mechanisms whereas the ADAMS model is repeatedly calculating and updating this value throughout the motion. This means 2.4 seconds is an overestimate. Additionally, the initial ADAMS model does not account for any losses directly from the motor or method of transmission, in this case gears. This indicates that the ADAMS model is an underestimate. The real value will likely be somewhere between our two calculated

times, but there are other conditions, for example the friction between the joints, which is not being taken into account.

We determined that our value for T_{motion} was so much of an overestimate because our T_{nom} value was incorrect. If we were to redo this analysis, we would double check our values to make sure the calculation was accurate. This would likely lead to a more reasonable value for T_{motion}.

The initial T_{nom} value was taken from the extrapolated stall torque listed on the Pololu website for the motor listed on the project kit. A screenshot is attached below.

Pololu Metal Gearmotors » 37D Metal Gearmotors » 12V 37D Metal Gearmotors »

30:1 Metal Gearmotor 37Dx68L mm 12V with 64 CPR Encoder (Helical Pinion)

Pololu item #: 4752 **128** in stock
Brand: Pololu supply outlook
Status: Active and Preferred
✓RoHS 3
Free shipping in USA over $100

Price break	Unit price (US$)
1	51.95
5	47.79
25	43.97
100	40.45

Quantity: 1 Add to cart
backorders allowed Add to wish list

This gearmotor is a powerful **12V** brushed DC motor with a **30:1** metal gearbox and an integrated quadrature encoder with a resolutic gearbox's output shaft. The gearbox is composed mainly of spur gears, but it features helical gears for the first stage for reduced noise shaped output shaft. This gearmotor is also available without an encoder.

Key specifications:

voltage	no-load performance	stall extrapolation
12 V	330 RPM, 200 mA	14 kg·cm (190 oz·in), 5.5 A

Section 5: Energy Conversion Introduction

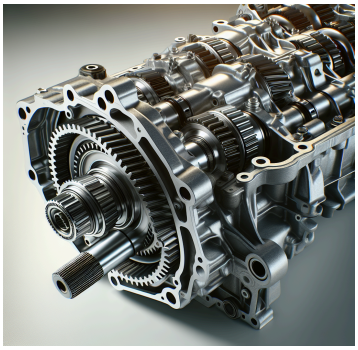


Figure 5.1 A transmission system

Transmission refers to the mechanism responsible for transferring power from a source (such as an engine or motor, one point) to a desired output (another point). Their primary purpose is to adjust torque, speed, and direction of motion to meet the requirements of the system. They can utilize various mechanisms including gears, belts, chains, and shafts to achieve this power transfer efficiently and reliably, making them fundamental components in a wide range of machinery and mechanical systems.

In ME350 a transmission is mostly used to change the slope of the torque-speed curve of a motor without changing the input voltage, so it could adapt to various load requirements.

A transmission can alter the output's speed, torque, and acceleration. If gear ratio is given in input/output, then for a constant input torque, output torque is negatively proportional to the gear ratio (the smaller the greater) while the speed and acceleration varies positively with gear ratio. In addition to these, transmission's changes to the resolution is also noted in the project. The larger the overall gear ratios become, the less resolution for the motor. This means if the smallest resolution of the motor is 2 deg (the smallest degree it can precisely move), a large gear ratio will make it even larger, making hitting the target harder. We need a transmission that enables the motor to hit targets accurately.

In general transmission design is limited by mechanical efficiency, limited gear choices, cost, heat dissipation, and serviceability. For this project we're limited by available gears, materials (especially gear materials, as some gears with certain teeth are only made in plastic), and cost.

Section 6: Transmission Ratio and Type Determination

For transmission ratio, we used inertia matching and light beam resolution methods to generate our transmission ratio. The inertia matching method will make sure the moment of inertia provided by the motor is matched with the load, so that our motor has enough torque to turn the linkage and flash light freely. (minimizing the motor torque) The light beam resolution method considers the maximum resolution of the motor. The smallest increment of the motor is finite (64*30 CPR), decided by motor encoder. And for each scoring position range, we wish there's at least 30 motor increments (the resolution would be 30). A higher desired count range is beneficial because it means that the controller has a wider range of values it needs to hit in order for the light beam to fall on the target. Both methods will generate an optimal transmission ratio, and the most conservative (biggest) will be selected, as it will satisfy both resolution requirement and inertia requirement.

For the 5 different scoring positions, the transmission angle, R values, total inertia of the mechanism, and transmission angles are as follows:

Position 1

Geometric Variables

Variable	Value	Units
R1 (input link)	27	cm
R2	86.84	cm
R3	83.15	cm
R4 (output link)	22.7584	cm
R5	84.92	cm

Total Inertias of a Link (via Parallel Axis)

Variable	Value	Units
Iinput	13486.9825	(gram*cm^2)
Ioutput	8025.118086	(gram*cm^2)
Icoupler	1040809.98	(gram*cm^2)

Scale Factors

R1/R2	0.3109166283	(unitless)
(R1*R3)/(R2*R4)	1.13596376	(unitless)

Total Inertia

Itotal	124456.9203	(gram*cm^2)

Position 2

Geometric Variables

Variable	Value	Units
R1	27	cm
R2	62.9	cm
R3	60.8	cm
R4	22.7584	cm
R5	61.72	cm

Total Inertias of a Link (via Parallel Axis)

Variable	Value	Units
Iinput	13486.9825	(gram*cm^2)
Ioutput	8025.118086	(gram*cm^2)
Icoupler	555201.648	(gram*cm^2)

Scale Factors

R1/R2	0.4292527822	(unitless)
(R1*R3)/(R2*R4)	1.146766432	(unitless)

Total Inertia

Itotal	126340.9187	(gram*cm^2)

Position 3

Geometric Variables

Variable	Value	Units
R1	27	cm
R2	47.72	cm
R3	46.94	cm
R4	22.7584	cm
R5	47.16	cm

Total Inertias of a Link (via Parallel Axis)

Variable	Value	Units
Iinput	13486.9825	(gram*cm^2)
Ioutput	8025.118086	(gram*cm^2)
Icoupler	328916.9537	(gram*cm^2)

Scale Factors

R1/R2	0.5658005029	(unitless)
(R1*R3)/(R2*R4)	1.166983426	(unitless)

Total Inertia

Itotal	129712.2453	(gram*cm^2)

Position 4

Geometric Variables

Variable	Value	Units
R1	27	cm
R2	23.05	cm
R3	24.04	cm
R4	22.7584	cm
R5	23.19	cm

Total Inertias of a Link (via Parallel Axis)

Variable	Value	Units
Iinput	13486.9825	(gram*cm^2)
Ioutput	8025.118086	(gram*cm^2)
Icoupler	88215.99051	(gram*cm^2)

Scale Factors

R1/R2	1.171366594	(unitless)
(R1*R3)/(R2*R4)	1.237330082	(unitless)

Total Inertia

Itotal	146814.4578	(gram*cm^2)

Position 5

Geometric Variables

Variable	Value	Units
R1	27	cm
R2	22.3266	cm
R3	23.2918	cm
R4	22.7584	cm
R5	22.2758	cm

Total Inertias of a Link (via Parallel Axis)

Variable	Value	Units
Iinput	13486.9825	(gram*cm^2)
Ioutput	8025.118086	(gram*cm^2)
Icoupler	82283.02606	(gram*cm^2)

Scale Factors

R1/R2	1.209319825	(unitless)
(R1*R3)/(R2*R4)	1.237663258	(unitless)

Total Inertia

Itotal	146115.118	(gram*cm^2)

Figure 6.1 Inertia matching calculation

The total inertia is calculated by summing the input link's inertia (which is directly attached to the motor), the output link's inertia times (R1/R2)^2 (compensate for the fact it's pulled by the linkage), and the coupler's inertia times ((R1*R3)/(R2*R4))^2. Each link's inertia is acquired in Solidworks after inputting the correct density.

We are able to sum these values using the parallel axis theorem.

The R values are as shown:

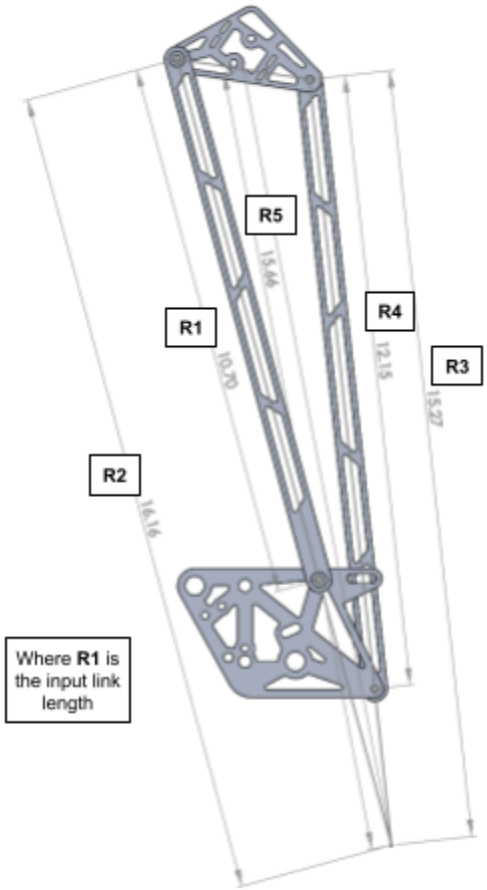


Figure 6.2 R values

R1 and R4 are input and output link lengths, respectively. Extending both the input and output links to where they intersect and the extended length of the input and output links, from the coupler joint to the intersection point, would be R2 and R3, respectively. R5 is the distance from the center of gravity of the coupler to the intersection point.

The resultant transmission ratio calculated is **2.423340321**

For the light resolution, we required that at least half of the sensor should be covered by the light beam. This will create a minimum angle and maximum angle, the former is “the first time” half of the sensor is covered with light, and the latter “the last time”. Between these 2 angles is the range where we consider the light to properly hit the target. And this range, we decided, should have at least 30 counts so we can fine-tune the motor to make the light shine more accurately onto the sensors.

Equations used:

$$Desired\ Count\ Range = \left(\frac{64\ counts}{1\ rev}\right) \cdot 30 \cdot \left(\frac{smallest\ difference\ [deg]}{\frac{360\ degrees}{1\ rev}}\right) \cdot N$$

Calculated angles:

Position	Minimum Angle [deg]	Maximum angle [deg]	Difference
1	59.01	62.25	**3.24**
2	75.75	79.6	**3.85**
3	91.7	94.97	**3.27**
4	139.45	141.2	**1.75**
5	143.19	144.72	**1.53**

Figure 6.3 measured target angles and ranges

The smallest range is 1.53 [deg]. From this difference we calculated that the transmission ratio should be **3.676470588**.

Different available transmission types:

1. Gear Transmission: Utilizes sets of gears with different tooth counts to transfer power and vary speed and torque. Pro: large torque transfer. Con: precise machining, limited choices.
2. Belt Transmission: Uses flexible belts, typically made of rubber or synthetic materials, to transfer power between pulleys. Pro: Smooth operation, quiet. Con: sliping, stretching, light load only.
3. Chain Transmission: Relies on roller chains to transmit power between sprockets. Pro: durable, capable of handling high loads. Con: heavy, low efficiency, precise machining.
4. Shaft Transmission: Involves the use of solid shafts to transmit power between components. Pro: robust and suitable for high-torque applications. Con: heavy, precise alignment.

Pugh chart for this specific project. Considering this is a working condition with limited space (base plate small), rapid changes in torque (sudden acceleration from one target to another), and limited torque output (small motor), efficiency, tolerance to sudden torque and space requirements are added to the chart.

	weight	Gear (reference)	Belt	Chain	Shaft
Cost	4	0	0	0	-1
Manufacturability	5	0	+1	0	-1
Efficiency	5	0	0	-1	-1.5
Sudden torque tolerant	3	0	-1	1	1
Packaging space	3	0	-1	-1	-1
Total		0	-1	–5	–16.5

Gears won by a substantial margin. A gear transmission with a ratio of [3.68] is selected.
Gear manufacture and part number is shown below.

Name	Quantity	Part number	Manufacturer	Price
Metal Gear	1	2664N472	McMaster-Carr	$22.26
Plastic Gear	1	2662N347	McMaster-Carr	$17.10

Section 7: Final Transmission Design

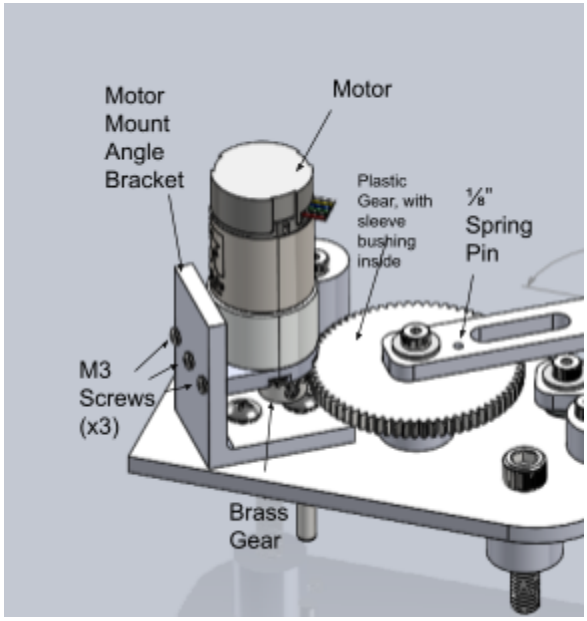


Fig 7.1a. Transmission Top View

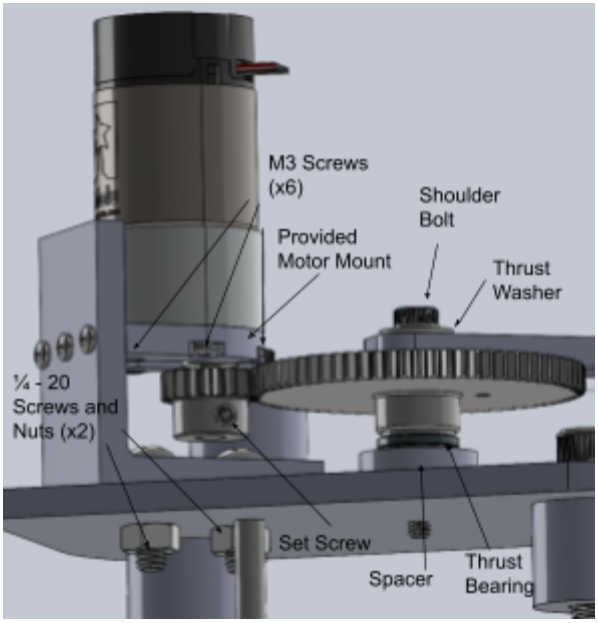


Fig 7.1b. Transmission Bottom View

Feature Discussion

The first feature to discuss is the method of motor mounting. The motor provided in the project kit is mounted to the base plate using a machined 90-degree angle bracket. There are two holes on the bottom face which allow ¼ - 20 screws to join the angle bracket and base plate. There are three holes on the side face, which allow M3 screws to join the angle bracket and motor mount (from the manufacturer). This motor mount contains six holes for M3 screws which connect to the shaft-side face of the motor.

The primary form of adjustability within this transmission design involves the base plate holes for the motor mount angle bracket. In initial design, these were slightly oversized to ensure that we can achieve a proper gear mesh, which may have to account for small manufacturing errors. Under the advice of our GSI, for the actual manufacturing we decided to keep the hole size nominal initially while being prepared to drill out the holes further if we were unsatisfied with the gear mesh.

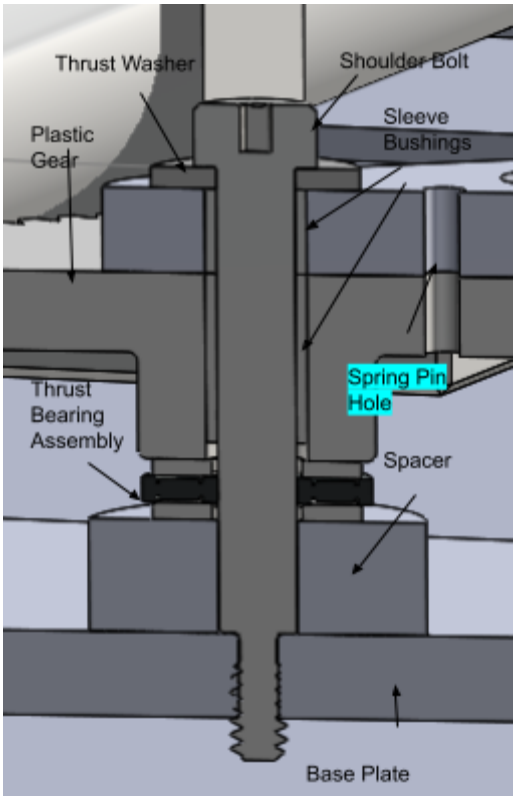


Fig 7.2. Cross Section of Input Link to Base Plate Joint

A 19-tooth brass gear with mod 1 is attached to the motor D-shaft using a set screw. If, during testing, we notice that the set screw is having difficulty staying in place, we are prepared to modify the motor and gear to make use of a spring pin. A matching mod 1, 70 tooth acetal plastic gear is attached to the input link and meshes with the brass gear. These gears were chosen to ensure an optimal transmission ratio, which was found by considering our desired encoder resolution and inertia matching, and then choosing the more conservative value. Our desired resolution ended up being the deciding factor, given the mechanism motion. The plastic gear transfers torque to the input link with an off-axis steel spring pin and a sleeve bushing was press-fit into the gear to ensure smooth rotation around the shoulder bolt.

Similar to the rest of the joints, as mentioned in earlier sections, the composition includes a ¼” shoulder bolt, a thrust washer, two sliding components with press-fit bushings, and a needle-roller thrust bearing enclosed by two washers. This specific joint also contains a spacer to maintain the input link’s proper height relative to the base plate.

Calculations are shown in the following section to support the material selections discussed above.

As detailed in Section 3, the mass estimate for the linkage design was approximately 370 g. This included the input link, follower link, coupler link, flashlight, flashlight mount, and all the connecting hardware components. The linkage design has not been touched, so the total estimate can be found by adding this weight with the two additional gears, an ⅛” spring pin, and a sleeve bushing for one of the gears. The other joint components between the input link and base plate should already be accounted for in the original estimate. The sleeve bushing and spring pin are both fairly small and each weight can be assumed to be around 1 g. The weight of the brass gear is roughly 31.2 g assuming a density of 8.73 g/cm^3 and a volume of roughly 3.57 cm^3. The weight of the acetal gear is approximately 27.42 g assuming a density of 1.41 g/cm^3 and a volume of roughly 19.45 cm^3. Volume estimates for both gears were found in SolidWorks. Thus, the additional weight due to the transmission is 60.6 grams, making the new total weight equal to roughly 430.6 g.

Section 8: Torque Transfer Analysis

Materials in Question

Acetal gear: https://www.mcmaster.com/2662N347/
Spring pin: https://www.mcmaster.com/98296A877/

Set Screw

Torque transfer analysis is being done for the set screw because it is critical for the screw to remain in place as the motor operates, otherwise the gear and the rest of the mechanism will not move as intended. The set screw will be considered to have “failed” once it has loosened enough that the motor shaft can spin freely relative to the brass gear. We are aware that the following set screw calculations may not be entirely representative of real life and we are prepared to convert this transfer method to use a spring pin instead if it becomes necessary.

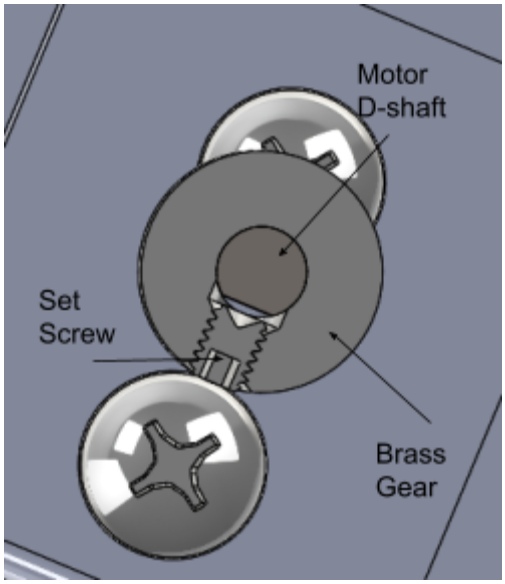


Fig. 8.1a. Labeled Cross-Sectional View of Set Screw Joint. Top-down view, with base plate to motor mount angle bracket screws in background.

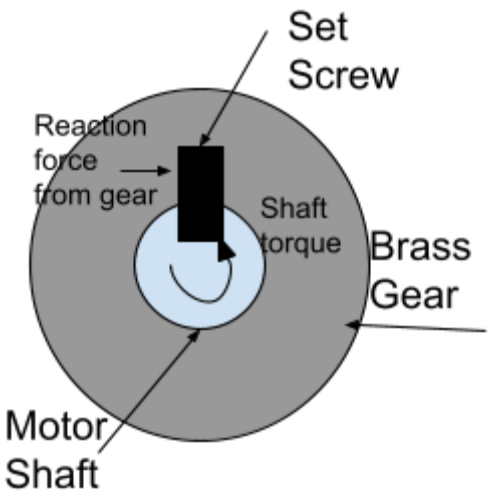


Fig. 8.1b. Free Body Diagram of Set Screw Joint. Torque from the shaft produces reaction forces in the screw.

Stall Torque Conversion

$$\frac{V_{nominal}}{T_{nominal}} = \frac{V_{actual}}{T_{actual}} \rightarrow \frac{12\,V}{14\,kg{\cdot}cm} = \frac{10\,V}{T_{actual}} \rightarrow T_{actual} = 11.67\,kg\cdot cm$$

$11.67\,kg\cdot cm = 1.144\,N\cdot m$
$1.144\,N\cdot m$ of axial holding power is required

Based on this chart the axial holding power of an m4 set screw on a 6 mm shaft is $5.1\,N\cdot m$ which provides a safety factor of greater than 4.

Thread Size	Seating Torque Nm	Axial Holding Power (kN)	Shaft diameter (shaft hardness Rc 15 to Rc 35) Torsional holding power Nm 1.4	1.6	1.8	2.0	3.0	4.0	5.0	6.0	8.0	10	12	14
M1.4	.10	.19	.13	.15	.17	.19	.29	.38	.48					
M1.6	.10	.22	.15	.18	.20	.22	.33	.44	.55	.66				
M1.8	.10	.25	.18	.20	.23	.25	.38	.50	.63	.75	1.0			
M2.0	.21	.29	.20	.23	.26	.29	.44	.58	.73	.87	1.2	1.5		
M2.5	.60	.53		.42	.48	.53	.80	1.10	1.30	1.60	2.1	2.7	3.2	
M2.6	.60	.56			.50	.56	.84	1.10	1.40	1.70	2.2	2.8	3.4	3.9
M3	.87	.71				.71	1.07	1.40	1.80	2.10	2.8	3.6	4.3	5.0
M4	2.20	1.70				1.70	2.60	3.40	4.30	5.10	6.8	8.5	10.0	12.0
M5	4.60	2.50					3.80	5.00	6.30	7.50	10.0	13.0	15.0	18.0
M6	7.80	4.20							11.00	13.00	17.0	21.0	25.0	29.0
M8	18.00	6.70								20.00	27.0	34.0	40.0	47.0
M10	36.00	9.30									37.0	47.0	56.0	65.0
M12	62.00	12.00										60.0	72.0	84.0
M14	62.00	15.00											90.0	105.0
M16	150.00	18.00												126.0

Spring Pin Strength

Analysis is being done to determine whether our method of transfer torque will be able to withstand the expected forces without failing. Failure can occur in two ways, one being the shearing of the spring pin and the other being bearing failure of either the plastic gear or aluminum link.

The breaking strength of a ⅛” diameter spring steel spring pin on McMaster is listed as 1,800 lbs. Given the torque calculated above and the distance of the spring pin from the shaft center, the safety factor for shearing in any of the materials can be calculated.

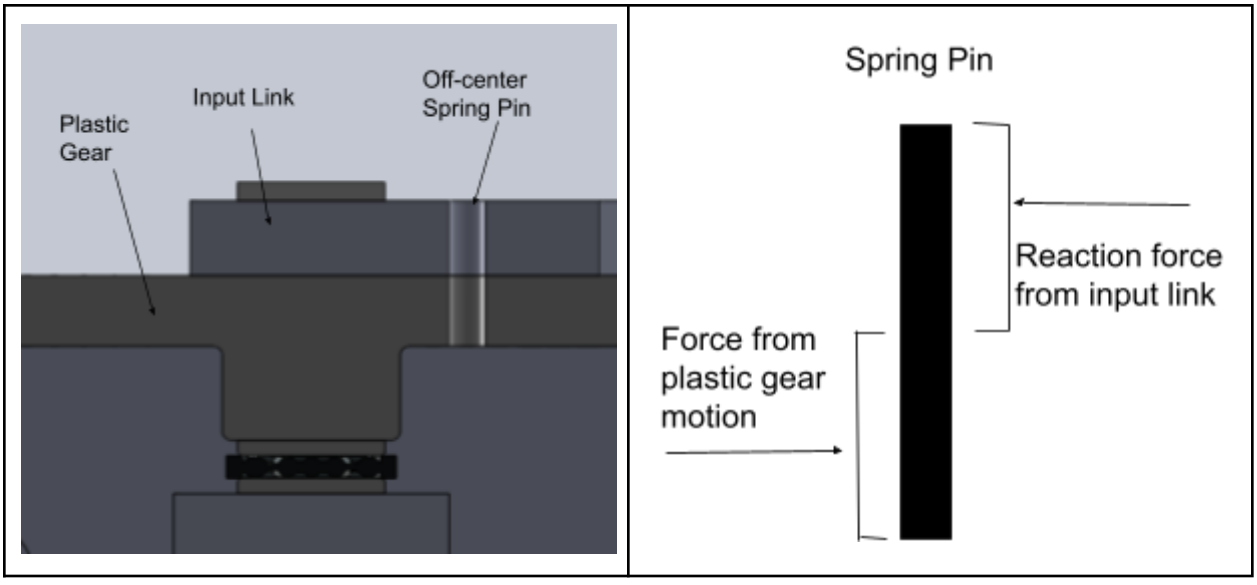


Fig. 8.2a. Labeled Cross-Sectional View of Spring Pin Joint. Spring Pin extends from top of input link to bottom of plastic gear.	Fig. 8.2b. Free Body Diagram of Spring Pin Joint. The forces shown in the diagram are tangential to the angular motion of the gear and they will be equal and opposite to the forces experienced in the gear and link.

$$11.67\ kg \cdot cm = 1.144\ N \cdot m = 10.1253\ in \cdot lbs$$

$$(10.1253\ in \cdot lbs) \cdot 3.7\ (gear\ ratio) = 37.46\ in \cdot lbs$$

$$F = T \div r = (37.46\ in \cdot lbs) \div 1.089\ in = 34.40\ lbs$$

34.40 lbs is far less than the given 1800 lbs breaking strength, so there is a high safety factor for failure.

Alternatively, comparing stresses using the single shear stress equation:

$$Shear\ Strength = \frac{F}{A} = \frac{1800\ lbs}{\pi \cdot \frac{1}{16}^2} = 146677.2\ psi$$

$$Shear\ Stress = \frac{F}{A} = \frac{34.40\ lbs}{\pi \cdot \frac{1}{16}^2} = 2803.2\ psi$$

There is a similarly large margin of safety between the expected stress and the strength of the spring pin.

Another form of failure could be the bearing stress on the plastic gear, which can be found by using the following formula

$$\sigma = \frac{F}{t \cdot d}$$

Where, *F* is the force applied on the material, *t* is the thickness of the material, and *d* is the diameter of the hole. (6 mm = 0.236 in)

$$\sigma = \frac{F}{t \cdot d} = \frac{34.40\ lbs}{0.236\ in \cdot 0.125\ in} = 1166.16\ psi$$

According to Matweb the shear strength of acetal plastic is 52 MPa or 7500 psi which provides a safety factor of > 6.

Plastic Gear Tooth Strength

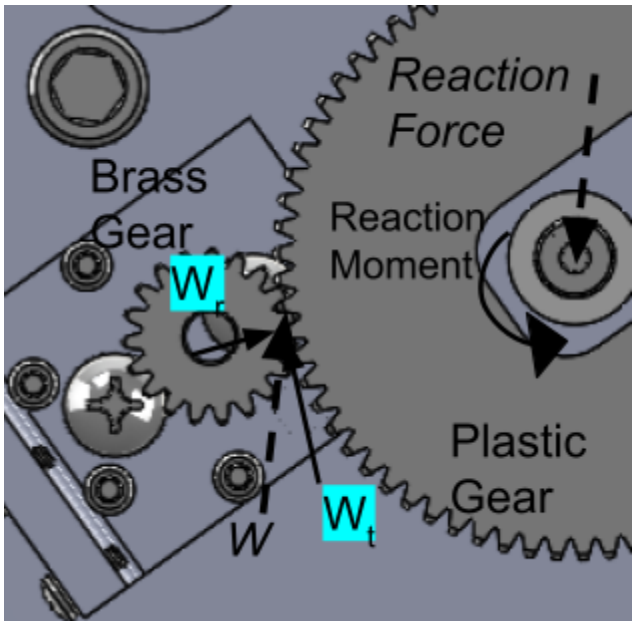


Fig 8.3. Free Body Diagram of Gear Mesh, with W_t being the tangential load experienced by the plastic gear

The equation for bending stress on a gear is shown below

$$\sigma = \frac{W_t \cdot P_d}{F \cdot Y} \cdot K_V$$

Where W_t is the tangential load, P_d is the diametral pitch, F is the face width, Y is the Lewis form factor, and K_V is the Barth Velocity Factor

Since the highest force will be applied at stall torque, we can assume there will be zero speed and therefore K_V is 1. From the Lewis Form Factor table, the factor Y can be assumed to be roughly 0.43 with 70 teeth. The face width F is 6 mm or 0.236 in. The diametral pitch is found by dividing the number of teeth, 70, in a gear by the pitch diameter, 70 mm or 2.76 in. So, P_d is equal to 25.36. The tangential load, W_t , is solved below.

$$W_t = T_S \cdot P_d \div 2 = (10.1253\ in \cdot lbs) \cdot (2.76\ in) \div 2 = 13.97\ lbs$$

Now, we can solve the bending stress on the plastic gear.

$$\sigma = \frac{W_t \cdot P_d}{F \cdot Y} \cdot K_V = \frac{(13.97\ lbs) \cdot (25.36\ teeth/in)}{(0.236\ in) \cdot (0.43)} \cdot 1 = 3491\ psi$$

According to Matweb, the average yield strength of Acetal plastic is around 55.5 MPa, which is ~8,000 psi. So, the safety factor for any plastic deformation in our plastic gear is > 2.

Regardless, in the unlikely case of failure, the first step taken would be to replace the gear with a metal gear of the same size and specifications. If a gear of the proper sizing cannot be found, changing the transmission ratio to something less ideal could be a consideration. Alternatively, finding a way to reinforce the plastic gear teeth, such as adhering a thin metal plate to either the top or bottom faces of the gear tooth could help reduce the stresses.

Conclusion

The calculations above demonstrate that our selected materials are strong enough to withstand our highest expected loading conditions.

Section 9: Motor Controls

Motor Controls
Controls are important for projects such as our linkage design because they allow us to incorporate sensing into an otherwise “blind” system. For example, if you were to just move the motor by a certain number of encoder counts, you could potentially have some error between your desired position and the actual position. This is analogous to walking in a dark room. In addition, some other problems that could come up are slow speed, encoder value drifting, and biases if our linkage design was moved from our desired start position.

Controls allow us to use sensor values such as those from limit switches and proximity sensors to update the position of the linkage through encoder values. Using our previous analogy, this is like using our hands to feel around the walls of a dark room.

In our linkage design, we use an encoder, proximity sensors, limit switches, toggle switches, Arduino board, breadboard, power supply, and H-Bridge.

The quality of our controls affects our ability to reach our desired position of shining the light onto our targets, the potential for error or harmful oscillations, and the speed of our linkage potentially resulting in low evaluation score.

PID and PWM
PID control is a method of control for a system response. The abbreviation PID refers to the proportional, integral, and derivative gains that are used to create a desired system response analogous to a mass, spring, and damper model. PWM is a method of control for a voltage response. Using varying frequencies, we can simulate a DC voltage which can range from 0 to the voltage given from our supply. In our linkage design we used PWM to change the speed of and overcome friction for our motor.
Our PID and PWM control are both done through the use of our Arduino microcontroller. The microcontroller takes signals from our sensors and creates output that our team programmed to do using duty cycles.

Proximity Sensors
Our proximity sensors use infrared light that is shined out and reflected back. At retrieval there is a photosensor that picks up the reflected light and converts it into a voltage output that is then discretized to allow us to evaluate the readings.

Proximity sensors can help us determine whether an object is close or far. This comes in useful in our project when determining which targets are up during our linkage design test. Some limitations include wide value ranges, noisy data, sensitivity, and often in need of calibration. The sensitivity of our sensors is both helpful and harmful as we need to determine which target

is up when there are targets a couple of inches away from each other. However, this is to our disadvantage as this sensitivity creates noise.

Section 10: Encoder Counts, Proximity Sensor Thresholds, and Controller Gains

Friction comp voltage

Friction compensation voltage was determined via testing and tuning because it is difficult to model joint friction. The linkage has 4 joints, a gearbox and motor with unknown inner resistance. The variables affecting friction are unknown and accurately modeling each bearing and each grind between washer and linkage is not feasible.

To find friction compensation voltage we first set the supply voltage to the motor 0, and gradually increment the voltage via a step of 0.1V. Until the motor stops winning and finally starts moving, we stopped increasing voltage and recorded the value to be 2.4V. Later with more fine tuning (increment by 0.005V) we found the more precise friction compensation voltage to be 2.42V.

Target Encoder values

Position values are calibrated by hand for each scoring trial. To calibrate encoder values we first calibrate the linkage to the hard stop and limit switch at the right most position. By tuning the hardstop we can make the flashlight shining accurately on target 5 sensor. This position is set to be the Target 5 position and on the motor count we set it to be 1600. Then we started calibrating each target's position by moving the linkage to the position (flashlight shining at the center of the target sensor) and reading off the encoder values. We did find play in the linkage, motor, and the gearbox: by approaching the target from a different direction we yielded a maximum of 50 encoder count difference. That is to say when pulling the linkage to the target from the left we may get 300, but approaching from the right after calibration we get 340. To solve this we first find the average number of the encoder counts from 2 directions, and let the linkage find the target from the 2 different directions for multiple times. We then get an average "error" or "miss" for the linkage. That is to say we know how much the linkage will miss the target every time it approaches the target. With this known error we can further fine tune the value, making sure no matter which direction we approach the target, the vast majority of the light beam shines on the sensor. This is usually a long process involving multiple trials and errors.

After this is done for one target, we repeat this for the other 3 targets, making sure the linkage always shine the light at the desired locations. The encoder values of the positions from 1 through 5 for playground 4 for our 28 point scoring trial is given below.

```
// Target positions:
const int CALIBRATION_VOLTAGE  = 5; // [Volt] Motor voltage used during the calibration process
const int TARGET_1_POSITION    = -2; // [encoder counts] Motor position corresponding to first target
const int TARGET_2_POSITION    = 320; // [encoder counts] Motor position corresponding to second target
const int TARGET_3_POSITION    = 635; // [encoder counts] Motor position corresponding to third target
const int TARGET_4_POSITION    = 900; // [encoder counts] Motor position corresponding to fourth target
const int TARGET_5_POSITION    = 1600; // [encoder counts] Motor position corresponding to fifth target
const int WAIT_POSITION        = TARGET_3_POSITION; // [encoder counts] Motor position corresponding to a wait position (when no targets are active)
const int LOWER_BOUND          = TARGET_1_POSITION; // [encoder counts] Position of the left end stop
const int UPPER_BOUND          = TARGET_5_POSITION; // [encoder counts] Position of the right end stop
const int TARGET_BAND          = 5; // [encoder counts] "Close enough" range when moving towards a target.
```

The wait position is determined to be the target 3 position as it is equally close to all other targets, so the linkage can move to any active target in a shorter time . The lower bond is target 1 position since it is the leftmost target and upper bond is the target 5 position, since we found that we do not need to reach target 4 via mirrors, but we can shine the light directly through the gap between the wall and the sensor.

Proximity target positions
Target positions off the proximity sensor readings are also decided by calibration and testing for each individual playgrounds. Once we wired the proximity sensors we used the “proxSensor” sketch file to gather the proximity sensor data. Since the proximity sensor numbers fluctuate and are very turbulent, we take 20 proximity sensor readings at a time to find the average of the actual proximity sensor reading. Once we have the 2 proximity sensor readings we can start calibrating the flags for different targets.

The proximity sensor reading for different target flags for playground 4 are given below:

```
const int TARGET1_PROXIMITYSENSE1_MIN = 350;     // [proximity sensor counts] Min value prox sensor 1 reads when target 1 is active
const int TARGET2_PROXIMITYSENSE1_MIN = 145;     // [proximity sensor counts] Min value prox sensor 1 reads when target 2 is active
const int TARGET2_PROXIMITYSENSE1_MAX = 180;     // [proximity sensor counts] Max value prox sensor 1 reads when target 2 is active
const int TARGET3_PROXIMITYSENSE1_MIN = 75;     // [proximity sensor counts] Min value prox sensor 1 reads when target 3 is active
const int TARGET3_PROXIMITYSENSE1_MAX = 90;     // [proximity sensor counts] Max value prox sensor 1 reads when target 3 is active
const int TARGET3_PROXIMITYSENSE2_MIN = 90;     // [proximity sensor counts] Min value prox sensor 2 reads when target 3 is active
const int TARGET3_PROXIMITYSENSE2_MAX = 105;     // [proximity sensor counts] Max value prox sensor 2 reads when target 3 is active
const int TARGET4_PROXIMITYSENSE2_MIN = 150;     // [proximity sensor counts] Min value prox sensor 2 reads when target 4 is active
const int TARGET4_PROXIMITYSENSE2_MAX = 170;     // [proximity sensor counts] Max value prox sensor 2 reads when target 4 is active
const int TARGET5_PROXIMITYSENSE2_MIN = 380;     // [proximity sensor counts] Min value prox sensor 2 reads when target 5 is active
const int TARGETNONE_PROXIMITYSENSE1_MAX = 30;  // [proximity sensor counts] Max value prox sensor 1 reads when no target is active
const int TARGETNONE_PROXIMITYSENSE2_MAX = 75;  // [proximity sensor counts] Max value prox sensor 2 reads when no target is active
```

To calibrate the readings we raise each target’s flag at a time, and read off the printed number in the serial monitor. For instance for the “TARGET1_PROXIMITYSENSE1_MIN” we raise flag 1 and read through the readings from the proximity sensor 1 on the serial monitor. To find min we find the minimum number of these readings. Thus the lower bound of this target can be determined.

Since the further away the flag is to the proximity sensor, the more likely the readings are off, we use sensor 1 for the left 3 targets (1 through 3) and 2 for the right 3 targets (3 through 5). Also when testing we found that flags may not come up one by one, and there are cases where more than 2 flags are raised. In these cases we need the proximity sensors to work individually. The leftmost sensor should be able to detect the leftmost flag, the right the rightmost, and mark those in the middle as “unsure” so that once the linkage clears the 2 targets on the outside it can start the search again. Thus we only use sensor 1 to detect the left targets, and 2 to detect right targets.
Finally we need the target none number. In this state the linkage will wait at the waiting position. This is decided by lowering all the flags and find the minimum reading from both proximity sensors respectively.

Gains

All the values for Kp, Ki and Kd are decided from iterations of tuning and testing because we decided that with our limited knowledge of controls (no ME360 experience) our calculations might not be accurate, or even correct for the linkage, and we risk the danger of damaging it at first try. Thus to be safe and manage the motion of the linkage much better, we decided to acquire the gain values also by testing and iterating.

We first set all gain values to 0, friction comp voltage to be 2.42 (as decided before) and let the linkage to approach a target location. The first gain we tune is Kp. We increment it by a step of 0.001 until the linkage can successfully move to the desired target. Then we incremented Kp by a step of 0.05, until we see oscillations of the linkage (going left and right instead of stopping at the target) at target. This number was around 0.24 and we then fine tuned the value by implementing smaller steps to get the largest Kp value that can accurately point the linkage with the smallest oscillation possible. This value is later determined to be 0.22. Using certain power supplies and playgrounds we used this value to be able to get 0.4, but with minimal speed difference. Thus to be conservative and stable across all playgrounds and equipment we decided to remain at 0.22.

After Kp is determined we started tuning Ki. As before we also increment the value with a step of 0.001. Incrementation stops when the linkage starts overshoot heavily. It is found that at Ki ≈ 0.05 the linkage can overshoot to a neighboring target and take a long time to stop oscillating. And at Ki ≈ 0.03 the oscillations are acceptable.

Finally Kd is determined by incrementing with a step of 0.001 until the linkage started to oscillate at high frequency when targeting. The final Kd value is 0.003.

After we found all the gains we further fine tuned the system. We decided that the linkage must be able to move not only quickly but efficiently. For us, the linkage being able to stop instantaneously at the target is more important than the shear speed. The time lost in waiting is disproportional to the time lost by speed difference. Therefore we required the linkage to have minimum overshoot, minimum oscillations, and we can trade speed for it. And with more iterations we determined

Kp = 0.22
Ki = 0.01
Kd = 0.003
Friction voltage = 1.4

Gain Values

```
const float KP                  = 0.22; // [Volt / encoder counts] P-Gain
const float KI                  = 0.01; // [Volt / (encoder counts * seconds)] I-Gain
const float KD                  = 0.003; // [Volt * seconds / encoder counts] D-Gain
const float SUPPLY_VOLTAGE      = 10; // [Volt] Supply voltage at the HBridge
const float FRICTION_COMP_VOLTAGE = 1.4; // [Volt] Voltage needed to overcome friction
```

Table 2: Summary of Variables in Arduino Code

Variable Name in Arduino Sketch File	Purpose of Variable	Device	Calculated Value	Actual Value Used During Testing
TARGET_1_POSITION	Position for Target #1	Motor	0	-2
TARGET_2_POSITION	Position for Target #2	Motor	N/A	320
TARGET_3_POSITION	Position for Target #2	Motor	N/A	635
TARGET_4_POSITION	Position for Target #2	Motor	N/A	900
TARGET_5_POSITION	Position for Target #2	Motor	1600	1600
K_p	Proportional Gain	Motor	N/A	.22
K_I	Proportional Gain	Motor	N/A	.01
K_D	Proportional Gain	Motor	N/A	.003
FRICTION_COMP_VOLTAGE	Friction Compensation Voltage	Motor	N/A	1.4
TARGET1_PROXIMITY SENSE1_MIN	Sensor Threshold for Target #1	Proximity Sensor	N/A	350
TARGET2_PROXIMITY SENSE2_MIN	Sensor Min Threshold for Target #2	Proximity Sensor	N/A	145
TARGET2_PROXIMITY SENSE2_MAX	Sensor Max Threshold for Target #2	Proximity Sensor	N/A	180
TARGET3_PROXIMITY SENSE3_MIN	Sensor Min Threshold for Target #3	Proximity Sensor	N/A	75
TARGET3_PROXIMITY SENSE3_MAX	Sensor Max Threshold for Target #3	Proximity Sensor	N/A	90
TARGET3_PROXIMITY SENSE3_MIN	Sensor Min Threshold for Target #3	Proximity Sensor	N/A	90

TARGET3_PROXIMITY SENSE3_MAX	Sensor Max Threshold for Target #3	Proximity Sensor	N/A	105
TARGET3_PROXIMITY SENSE4_MIN	Sensor Min Threshold for Target #4	Proximity Sensor	N/A	150
TARGET3_PROXIMITY SENSE4_MAX	Sensor Max Threshold for Target #4	Proximity Sensor	N/A	170
TARGET3_PROXIMITY SENSE5_MIN	Sensor Min Threshold for Target #5	Proximity Sensor	N/A	380
TARGETNONE_PROXI MITYSENSE1_MAX	Sensor Threshold for No Target	Proximity Sensor	N/A	30
TARGETNONE_PROXI MITYSENSE2_MAX	Sensor Threshold for No Target	Proximity Sensor	N/A	75

Section 11: Final Testing Results/Discussion

For our testing setup, we began by mounting our mechanism to whatever playing field was available. Due to slight differences between the different playing fields, we had to update the encoder values for each target position to make sure the flashlight beam would shine directly on each target. Once these values were correct, we adjusted the hard stops to make sure the beam was shining on target 1 while the mechanism was touching the left hard stop, and on target 5 when the mechanism was touching the right hard stop. Lastly we made sure that the friction compensation voltage was sufficient to overcome the friction caused by the joints and make the mechanism move smoothly.

Once we had our mechanism moving smoothly, we set it to a random position and plugged in the targets to see what score it hit. However, in our early tests, we ran into our first problem. When the mechanism made contact with the left hard stop, it would bounce off of the hard stop. This was caused by the speed at which it hit the hard stop and the slack between the gear teeth on our mechanism. When the mechanism bounced off the hard stop, the light would no longer shine on target 1, but because of the slack between the gears, the encoder count would be correct. Since the encoder counts were still correct, the mechanism would not correct itself, meaning the beam of light did not hit the targets for the rest of the trial run. To fix this problem, we adjusted the left hard stop so the mechanism did not make firm contact with it when in the target 1 position. Instead, the mechanism's contact with the hard stop was minimal. This ensured that the mechanism would not become uncalibrated during the test run and that it would hit every target. After solving this problem we were able to score a 25.

A problem that remained unresolved even after the hardstop issue was fixed was the slack between the two gears of our mechanism. The light would still hit the targets, but the clearance between the mating gear teeth made the beam only partially shine on the targets. This was an issue with the light intensity on the targets, especially the ones near the aluminum wall. The root of the problem was the speed of the mechanism combined with the change in direction and gear geometry. The momentum caused the beam of light to hit the targets less directly when the direction of movement changed. We were able to get a complete score, but we knew we could improve our score by improving the light intensity on the targets. There was nothing we could do to change the geometry of the gears and the way they meshed, so we fixed the problem in two other ways. We adjusted the flashlight so the beam had a wider radius. We also tightened the joints between the motor bracket and the motor mount to lessen the effect of the change of direction on the position of the linkage when it is calibrated. Fixing this issue improved our score from 25 to 27.

Our last approach to improving our score was simply making the mechanism faster and more efficient. To this end, we fine tuned our k_D, k_P, and k_I values to improve the movement of the mechanism. We also adjusted the proximity sensor tuning to make the movement of the mechanism more robust. Through multiple trials where we tested different PID values we were

able to improve our score from 27 to 28, which is the maximum score we reached. A summary of our trials and adjustments is shown in the table below.

Trial #	Score	Issue Recognized	Implementation for Improvement
1	DNF	Mechanism was bouncing off of the left hard stop and skewing the target positions to the right.	N/A
2	25	Clearance between mating gear teeth made the beam of light not hit the targets directly.	Adjusted the hard stop to make sure the mechanism did not bounce off the hard stop.
3	27	Mechanism could move more robustly and faster.	Increased the light beam radius and tightened the joint between the motor mount and the motor bracket.
4	28	N/A	Adjusted K_P, K_I, and K_P values and fine tuned the proximity sensors.

Section 12: Design Critique & Evaluation

The overall linkage motion seemed to work well and there were rarely (if any) issues which involved the linkage binding or slowing down significantly during certain ranges of its motion. This is primarily because the linkage design included very small transmission deviation angles (below 15 degrees) and was aided by the ADAMS analysis done at the beginning of the semester to determine the power needed to drive the linkage. The individual component design for the joints, spacers, and hardstops also worked well, which was impacted both by well thought out design and careful manufacturing. The design incorporated tolerance stackup considerations, adjustability, and ease of manufacturing. The careful manufacturing included selecting hardware which accommodated any tolerance issues we may have seen in machining to reduce the amount of play in our bolted joints. The biggest issue we faced was a calibration error which occurred every time the linkage hit the non-limit switch hardstop. The linkage would bounce off the hardstop, which, in addition to an imperfect gear mesh, would cause the motor to inaccurately measure its own shaft positioning. To fix this, we adjusted the hardstop orientation (which was a simple task due to design) so that it would not come into contact with the linkage at the desired target positioning.

Our scoring strategy prioritized having a smooth linkage motion and having a high light intensity on the targets. To meet these goals, we created a kinematic model with a low transmission deviation angle and designed the lengths of the linkage to get the flashlight as close to the targets as possible. Having scored 28/31 this plan worked to a large extent, however, we believe the main thing holding us back was our speed moving between targets. Our speed was greatly impacted by the mass of our design and our transmission ratio, defined by our desired encoder count resolution. The lower priority we placed on the mass of our linkage, relative to robustness, meant that during the tuning of our programming, we weren't able to get very high k_D values before we saw significant oscillations. The primary speed reduction of our mechanism came from our 3.7 gear ratio, which was determined by looking at the smallest angular range at which our linkage might be able to move while leaving at least 50% of the flashlight beam on any given target. To address these two issues, the linkage could be improved with larger slots for weight saving with a smaller overall profile and the mounting locations of our links could be adjusted to provide a larger encoder count range at which the linkage can be positioned while providing ample light to the desired target.

From roughly analyzing the time steps in our recordings, our design performed close to as expected, falling between the simulated ADAMS performance (0.56s) and the hand calculated value (2.4s - which may have been inaccurate) for the time required to move between targets 1 and 5. The time required was much closer to the ADAMS model expectation than the hand calculated power analysis, but still a small amount higher. This discrepancy with the ADAMS model is very likely due to imperfections caused by friction and slop in our mechanism and, additionally, the small inefficiencies caused by our transmission ratio/gear mesh. The main reason that the hand calculated power analysis was so much higher than real life was because it did not account for the changing moment of inertia throughout the motion of the linkage and

always assumed the worst case scenario. While neither of the models precisely predicted the real world outcome of our part, it was useful to have an expected range of values within which we could expect our mechanism to perform. Additionally, the models/calculations we conducted which verified the strength of our components were representative of what we witnessed in real life, as we did not have issues with anything breaking or coming loose over time.

Friction primarily occurred between the joints of our linkage. It is a force that is not accounted for in our ADAMS model and this impacted both the speed and reliability with which our mechanism was able to perform. The joints were designed so that there would not be any friction induced by bolt clamping/preloading force, which meant that there would be some slop since motion in the vertical axis was not being entirely constrained by the bolt. So, in assembly, there was a tradeoff to be made in washer selection between potentially adding more friction, while limiting the play in the linkage, or ensuring no unnecessary friction but allowing more slop/uncontrolled motion in the linkage. Additionally, the force experienced in the joints was also affected by the cantilevered links, placing a moment on the joints, causing non-uniform levels of friction. To optimize this tradeoff, we made use of shims of varying thicknesses to tune the friction of our joints until we were satisfied with both the performance and reliability of our linkage, but again because friction is not accounted for in ADAMS, there is no way to achieve that idealized performance and our linkage did move a little more slowly.

One potential improvement to the control algorithm could be to include a calculation which determines which target is the closest before the linkage decides the next target to move to. An idea along the same lines is to include a deliberate delay or piece of code which will make sure that the mechanism will stay in a position until the flag goes down. Occasionally, it could be seen that the mechanism would jump between two active targets, and including a delay or statement requiring the mechanism to stay put could help reduce the amount of time wasted moving between targets. A feed-forward signal could be implemented if the sequence of the flags was known. In this way, we would be able to predefine the positioning and movement of the mechanism which would get rid of any potential errors or delays from the proximity sensor.

A sensor which could measure the amount of light being received by the targets could be used to determine the optimal positioning of the mechanism. This would have less reliance on having a stable calibration because the positioning could keep being adjusted until the light irradiance is maximized. Also, if the flags were made slightly wider from a front perspective, more proximity sensors could also be used to determine activation state of each target without the uncertainty caused by being unsure of the activation of flags being covered by an active flag. This might help with the back and forth mechanism motion which would sometimes occur after a flag went down.

One of the big things that we observed is the importance of adjustability and having a good understanding of a design space. We think prototyping and iterating designs off of real world data is a better way to go about creating this mechanism than we did. As mentioned in a

previous question, the links in our system are a fair bit heavier than they need to be, which would have been easier to recognize if we tried to maybe machine one singular part first or maybe had done some more strength analysis to determine how robust the links would need to be. So, we think one thing we would do differently is attempt to make a mockup of a designed part before finalizing it or maybe even machining a part to get a better idea of the scale of the assembly.

We would rate our mechanism in the 4-5 range, with 5 being the safest. There was minimal risk to our team and we didn't run into any particularly dangerous situations. The proximity sensors could be more helpful if they were more accurate, but they worked well enough on most of the playing fields. To make sure that everyone is staying safe, it could be a helpful reminder for teams to check the tightness of their bolts whenever they begin a session of testing, in the chance that they have been loosened.

At the moment, there is nothing stopping our device from being safely operated by anybody else. In terms of performance and hitting targets, there may be a few small adjustments that need to be made when it comes to ensuring a decent gear mesh and positioning the base plate. Particularly, the weight of the linkage causes the flashlight to droop slightly lower to the ground than anticipated. This can easily be accommodated by including a large washer beneath the base plate, and knowing this, we can't think of anything else that would prevent someone else from operating our device as designed.

It could have been useful to have various flashlight options so that more tradeoff analysis could be done between having a heavier linkage that allows a smaller flashlight to get close enough to the target to produce a high light intensity or having a lighter/smaller linkage with a bigger flashlight which can maybe produce similar light intensities from farther away. I think generally more plastic material options may also have been useful to give teams more design flexibility as opposed to just aluminum, although already the option to 3D-print has been very helpful.

We think the current point distribution is generally fair, although the percentage of points given to the weight of the linkage seems a little high. There is fairly little direct analysis which is suggested to be done to optimize weight other than using intuition and therefore it seems somewhat arbitrary for what a team's final mechanism weight comes out too. Additionally, the weight will already affect the performance score, so, to our team, it might make more sense to consider the reliability or ease of manufacturing of a design, and lessen the distribution of points towards the weight of the mechanism.

A project that could be fun and maybe a little too complicated is designing a linkage that mimics the gait of how a person walks. The goal could be to design a "robot," which is mainly just the linkage mechanism which allows it to move, and require it to try to navigate a very simple maze by making use of proximity sensors to see how close the nearest wall is. A motor/encoder could be used to adjust how much the robot turns its "legs" (the linkage) when it knows what direction it needs to move in. Extra points could be given to teams that can navigate the maze the quickest.

Appendix A: Assembly Manual

Part 1: Linkage and Mounting

Component	Base Plate	Input Link	Coupler Link	Follower Link	Hard Stops (Input, Follower)	Flashlight Mount Bottom/Top	Base Bolt Spacer	Input Link Spacer	Shoulder Bolt(½”)
Component Number	1	2	3	4	5 and 6	7 and 8	9	10	11
Component	Shoulder Bolt (1 ⅜”)	Socket Head bolt (⅜”)	Washer	Thrust Washer	Thrust Bearing	Dowel Pin	Sleeve Bushing	Flash-light	
Component Number	12	13	14	15	16	17	18	19	

Linkage Only Assembly

1. Press fit sleeve bushings into links. Use the arbor press to insert a sleeve bearing into the properly reamed out hole on each link. An example is shown in Figure A.1.

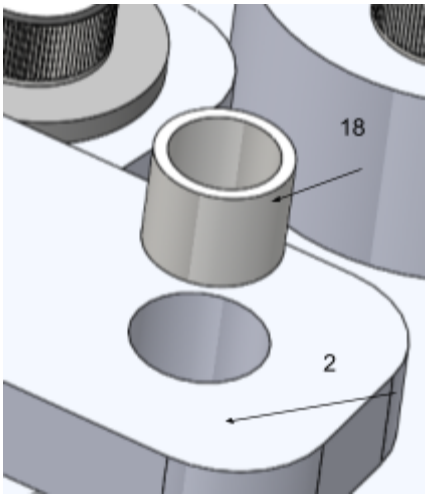


Figure A1.1

2. Attach the input link to the coupler link. Take a 1/2" long shoulder bolt and insert it into the input link hole containing a sleeve bushing, remembering to place a thrust washer between the head of the bolt and the link. Place the thrust bearing assembly (thrust bearing and two washers) onto the bolt on the opposite side of the link. Thread the bolt into the coupler link.

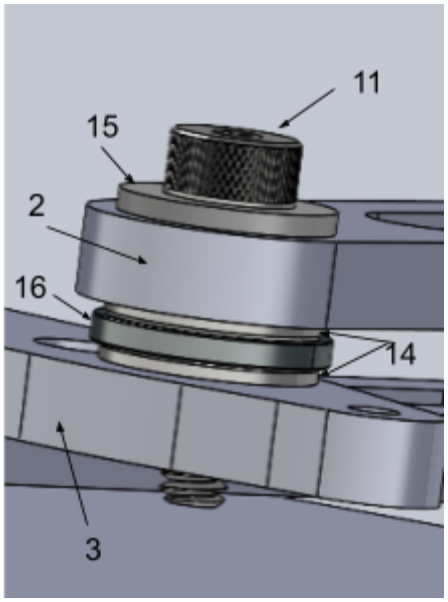


Figure A1.2

3. Attach the coupler link to the follower link. Take a 1/2" long shoulder bolt and insert it into the coupler link hole containing a sleeve bushing, remembering to place a thrust washer between the head of the bolt and the link. Place the thrust bearing assembly (thrust bearing and two washers) onto the bolt on the opposite side of the link. Thread the bolt into the follower link.

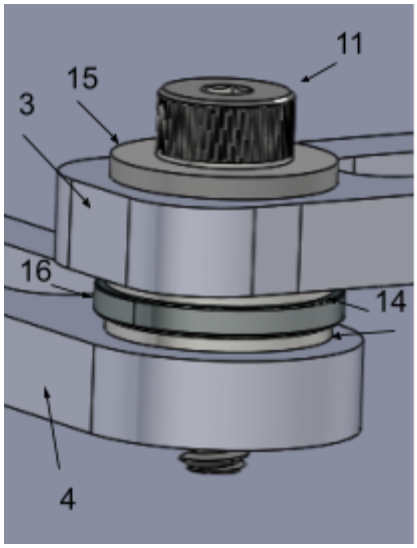


Figure A1.3

4. Attach the flashlight to the coupler. Place the flashlight inside the two piece flashlight mount. Insert bolts through the clearance holes of the mount and thread them into the tapped holes on the bottom face of the coupler.

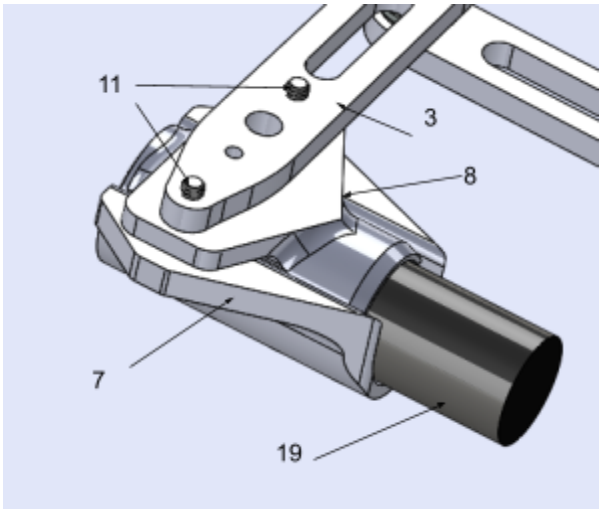


Figure A1.4

Base Plate Assembly

5. Align the spacers and dowel pin on the playing field for the base plate.

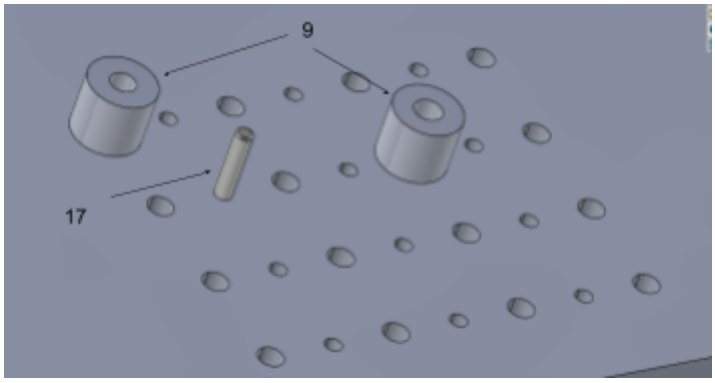


Figure A1.5

6. Connect the base plate to the playing field. Place the base plate on top of the spacers and insert the 3/8" bolts through the plate and into the playing field.

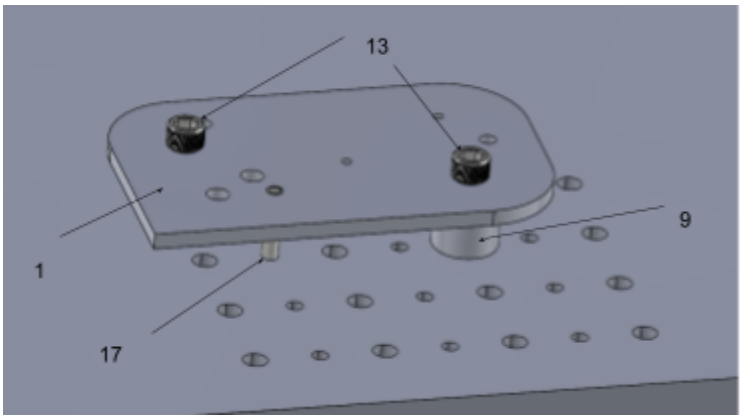


Figure A1.6

7. Attach the input link hard stop to the base plate. Use a shoulder bolt to connect the hard stop to the base plate in an orientation similar to that seen in CAD. The orientation will be adjusted as necessary during testing.

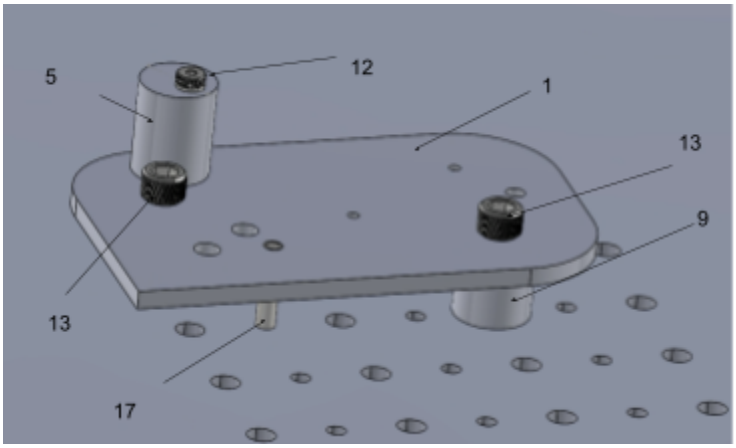


Figure A1.7

8. Attach the follower link hard stop to the base plate. Use a shoulder bolt to connect the hard stop to the base plate in an orientation similar to that seen in CAD. The orientation will be adjusted as necessary during testing.

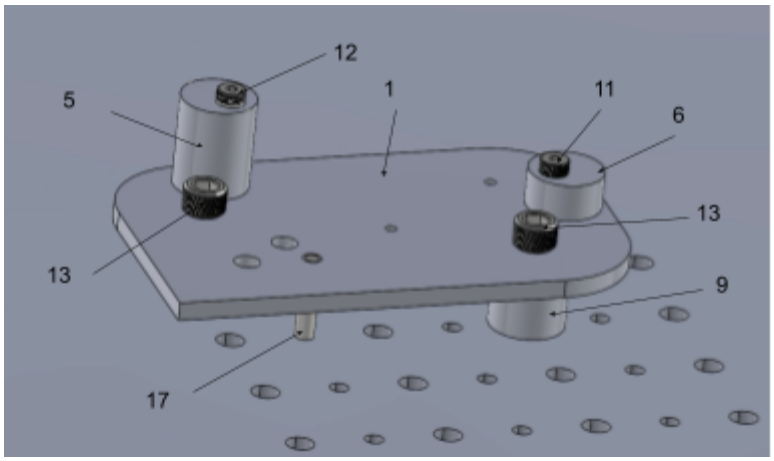


Figure A1.8

Connecting the linkage and base plate

9. Attach the follower link to the base plate. Take a shoulder bolt of proper length and insert it into the follower link hole containing a sleeve bushing, remembering to place a thrust washer between the head of the bolt and the link. Place the thrust bearing assembly (thrust bearing and two washers) onto the bolt on the opposite side of the link. Thread the bolt into the base plate.

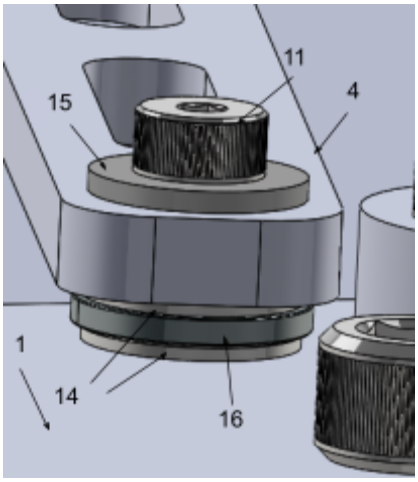


Figure A1.9

10. Place down the input link spacer over the input link hole of the base plate. Place a washer, thrust bearing, washer in that order above the input link spacer.

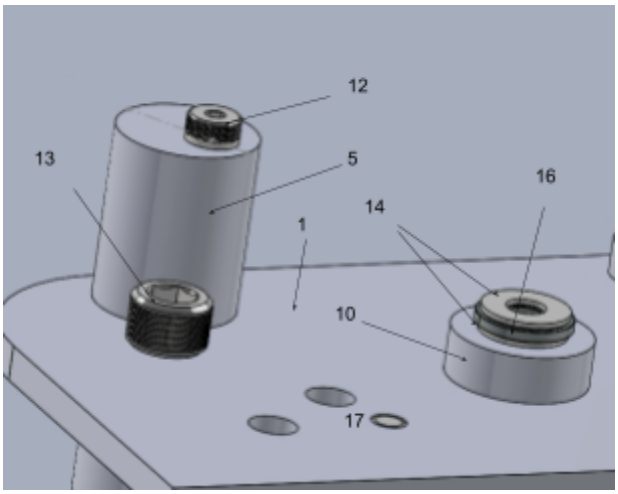


Figure A1.10

Part 2: Transmission

Component	Sleeve Bushing	Metal Gear	Plastic Gear	30:1 Metal Gearmotor	Bracket for 37D mm Metal Gearmotors	90 Degree Angle Motor Mount
Component Number	18	21	22	23	24	25
Component	Passivated 18-8 Stainless Steel Pan Head Phillips Screw (A118)	Passivated 18-8 Stainless Steel Pan Head Phillips Screw (A541)	18-8 Stainless Steel Hex Nut	Spring Pin		
Component Number	26	27	28	29		

Setting up the input link with gear.

1. Press fit an appropriate bushing into plastic gear.
2. Line up the plastic gear hole and input link hole to press fit a spring pin (not pictured) through both.
3. Align the input link joint components on the base plate in the following order. Place the machined spacer down, then the thrust washer, followed by the plastic gear, then input link, and the then thrust washer.
4. After setting up the stack, insert the appropriate shoulder bolt through all components and through the base plate.

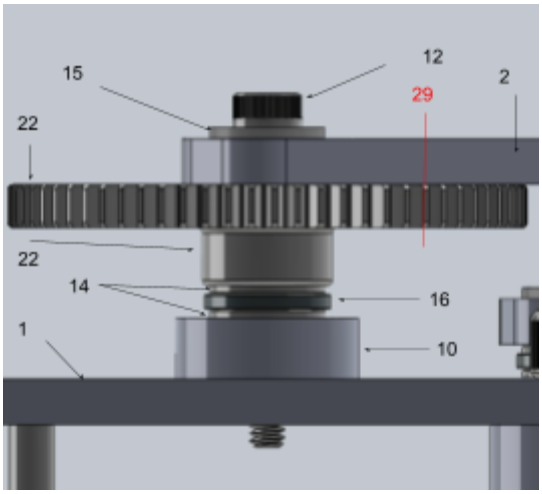


Figure A2.1

Mounting motor

5. Attach bracket to motor.

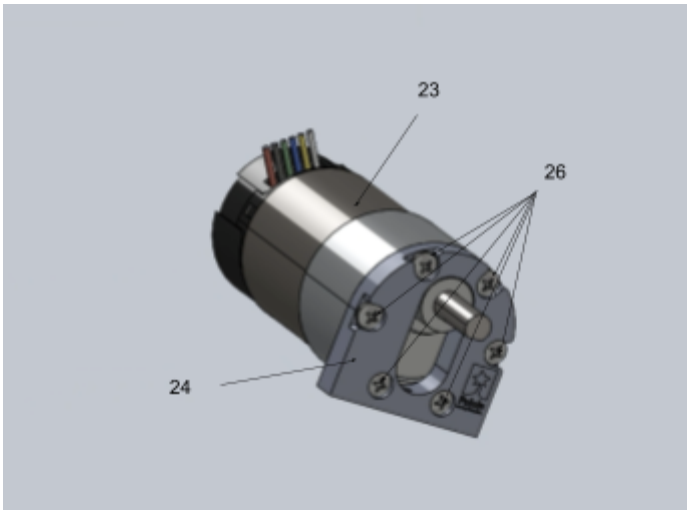


Figure A2.2

6. Attach the gear to the motor and properly torque the set screw.

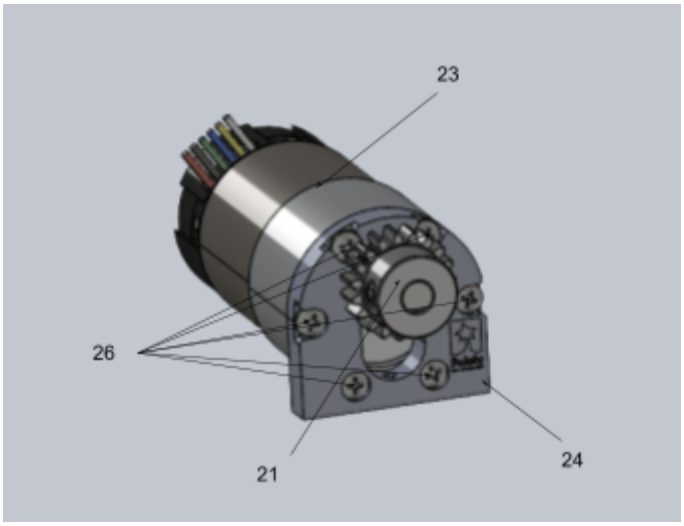


Figure A2.3

7. Attach the angle motor mount to the base plate using appropriate screws.

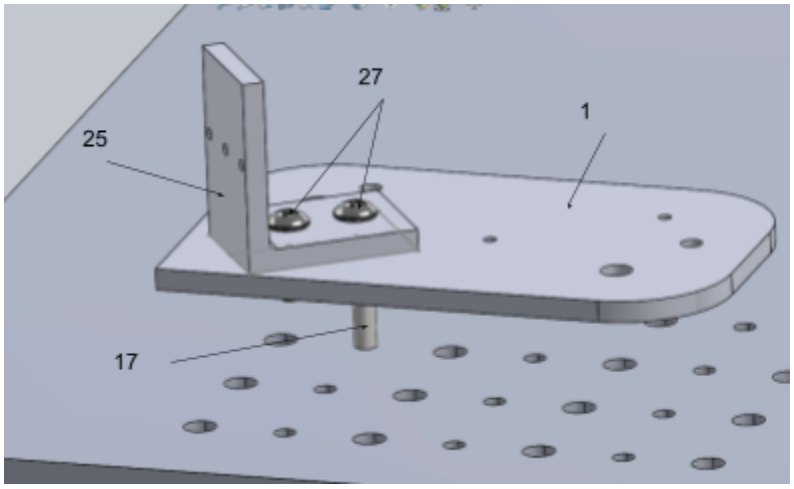


Figure A2.4

Some components of assembly are hidden for visibility.

8. Ensure good contact of plastic and metal gears.
9. Attach bracket to motor mount using appropriate screws.

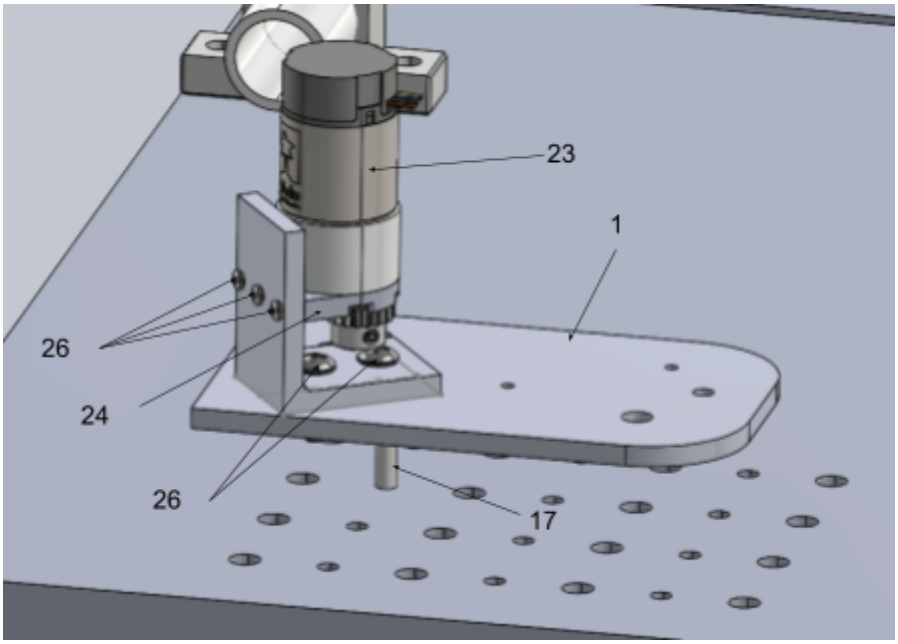
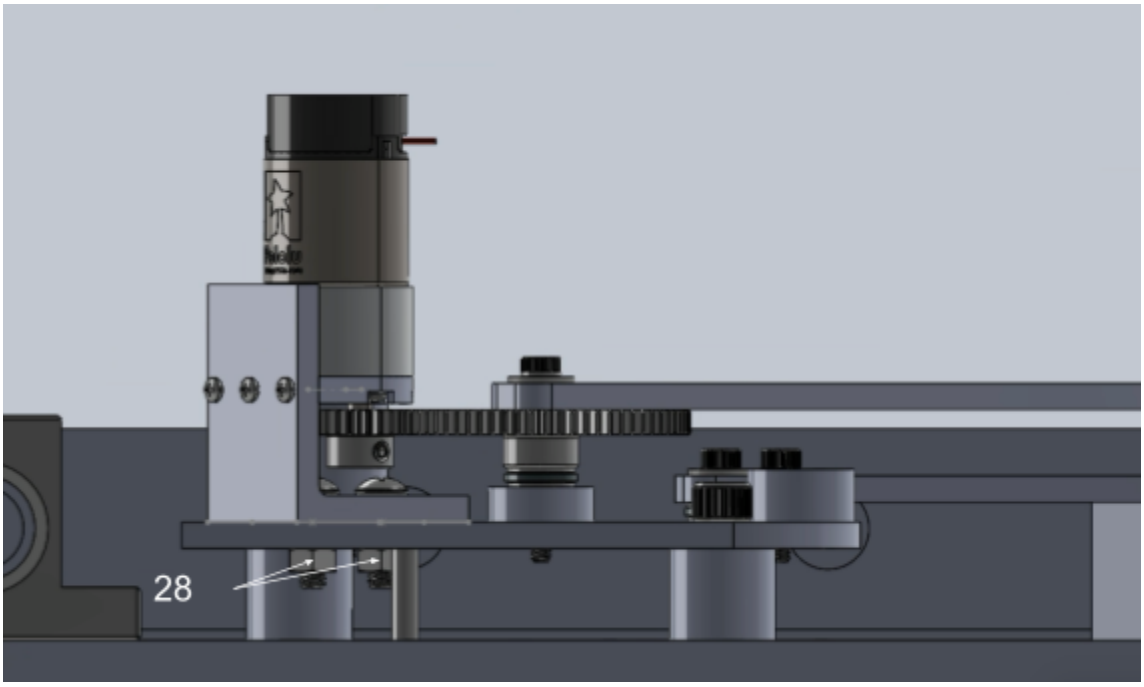


Figure A2.5

10. Fix the motor mount with nuts.



FigureA2.6

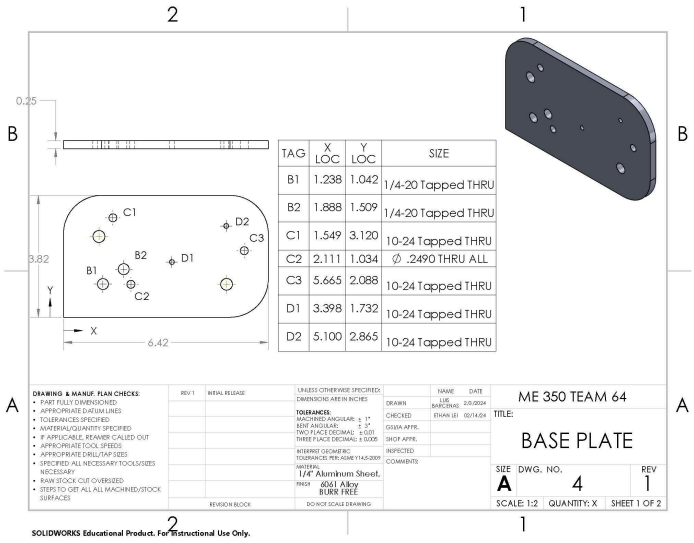
Appendix B: Bill of Materials for the Final Design

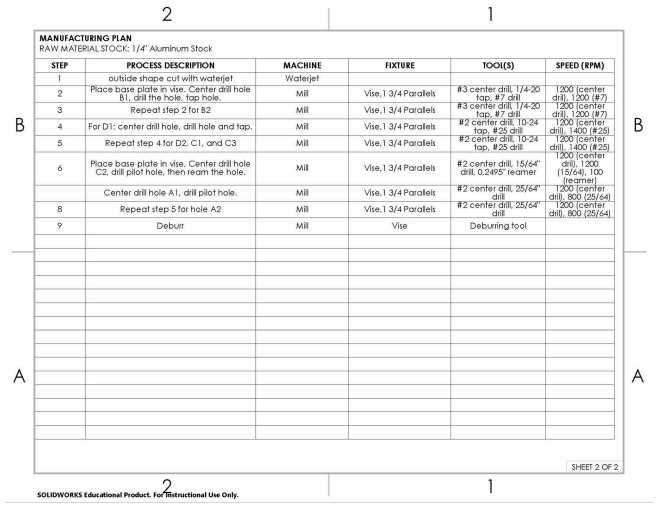
Part #	Title	Quantity	Supplier ID	Supplier	Price($)
1	Base Plate	1	n/a	Project Kit	0
2	Input Link	1	n/a	Project Kit	0
3	Coupler Link	1	n/a	Project Kit	0
4	Follower Link	1	n/a	Project Kit	0
5	Hard Stop 1	1	n/a	Project Kit	0
6	Hard Stop 2	1	n/a	Project Kit	0
7	Flashlight Mount 1	1	n/a	Project Kit	0
8	Flashlight Mount 2	1	n/a	Project Kit	0
9	Base Bolt Spacer	2	n/a	Project Kit	0
10	Input Link Spacer	1	n/a	Project Kit	0
11	Shoulder Bolt (½”)	6	91259A537	Project Kit	$1.64
12	Shoulder Bolt (1 ⅜”)	2	91259A471	McMaster	$8.30
13	Socket Head Bolt(⅜”)	2	91251A663	Project Kit	$10.98/10 pack
14	Washer	8	91950A047	Project Kit	$17/50 pack
15	Thrust Washer	4	5906K531	Project Kit	$1.57

16	Thrust Bearing	4	5909K23	Project Kit	$3.02
17	Dowel Pin	1	98381A546	Project Kit	$8.65/25 pack
18	Sleeve Bushing	1	60695K1	Project Kit	$1.86
19	Flashlight	1	n/a	Project Kit	0
20	Sleeve Bushing	2	6391K127	McMaster-Carr	$1.30
21	Metal Gear	1	2664N472	McMaster-Carr	$22.26
22	Plastic Gear	1	2662N347	McMaster-Carr	$17.10
23	30:1 Metal Gearmotor	1	n/a	Project Kit	0
24	Gear Bracket	1	n/a	Project Kit	0
25	Angle Motor Mount	1	n/a	X50 Shop	0
26	Steel Screws(m3)	9	92095A179	Assembly Room	$5.83/100 pack
27	Steel Screws(¼”-20)	2	91772A542	X50 Shop	$12.62/25 pack
28	Hex Nuts(¼”-20)	2	95462A029	X50 Shop	$8.95/100 pack
29	Spring pin	1	98296A877	Assembly Room	$18.35/500 pack
Shipping					$16.59
Tax					$3.02
Total					$69.87

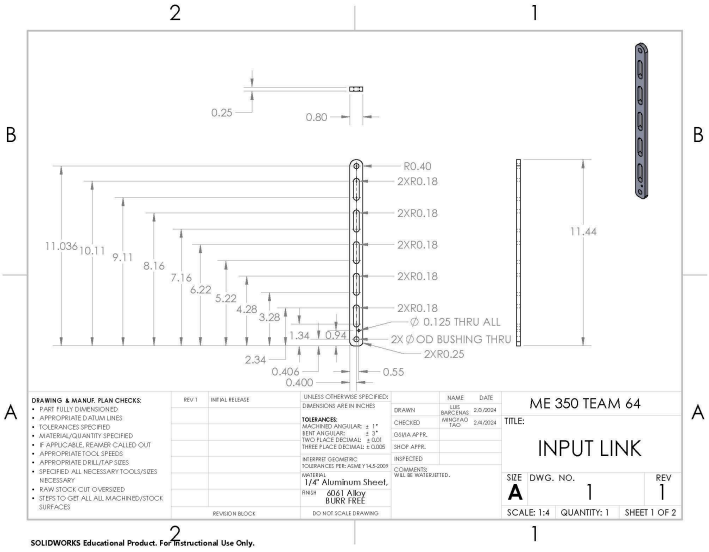
Appendix C: Approval Packages

Base Plate





Input Link



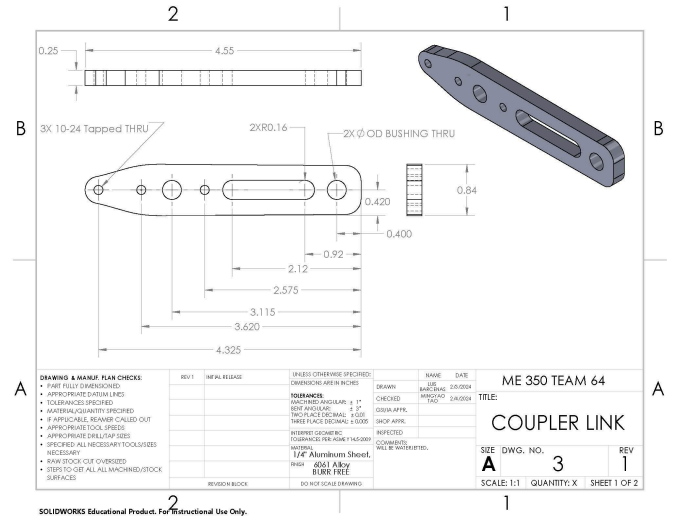
MANUFACTURING PLAN
RAW MATERIAL STOCK: XXXX

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Waterjet	Waterjet			
2	Clamp the water jetted part on the vise with one end extending out	Mill	Toe clamps		
3	Set x datum on left most point. Set y datum on bottom most point.	Mill	Toe clamps	Edge finder, drill chuck	1000
4	Center drill both large holes.	Mill	Toe clamps	#5 center drill, drill chuck	1200
5	Drill hole, change to low gear, then ream	Mill	Toe clamps	19/64" drill bit, drill chuck, appropriate reamer	1000, 100
6	Repeat for other hole.	Mill	Toe clamps	19/64" drill bit, drill chuck, appropriate reamer	1000, 100
7	Drill 1/8" hole	Mill	Toe clamps	1/8" drill bit, drill chuck	1600
8	Deburr			Deburring tools	

SHEET 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Coupler Link



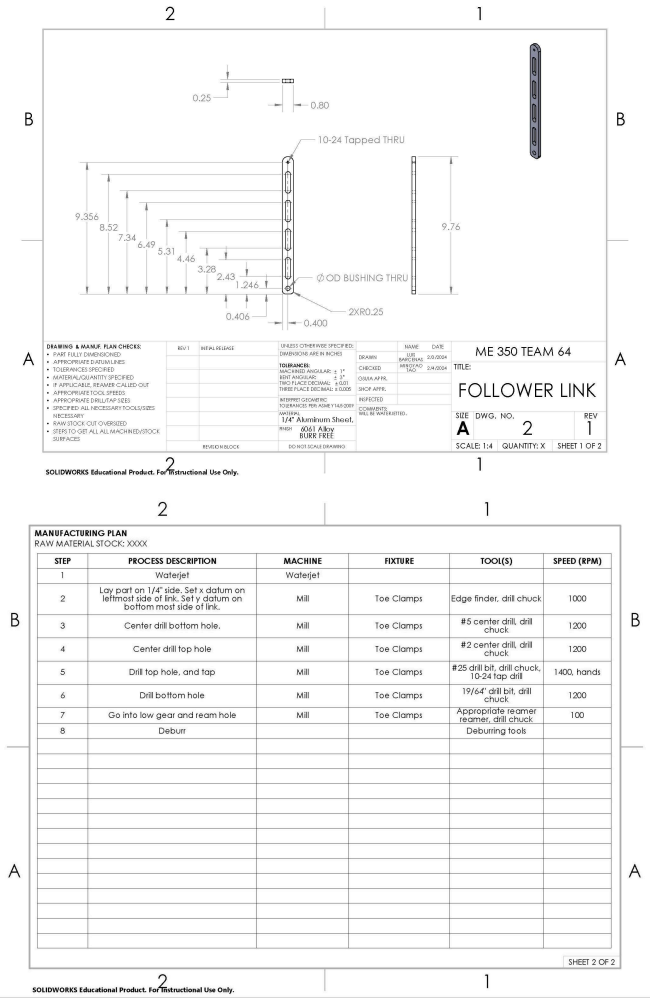
MANUFACTURING PLAN
RAW MATERIAL STOCK: XXXX

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Waterjet	Waterjet			
2	Set y datum on bottom most edge. Set x datum on leftmost side of part.	Mill	Vise, Toe Clamp Fixture	Edge finder, drill chuck	1000
3	Center drill the bushing holes	Mill	Vise, Toe Clamp Fixture	#5 center drill, drill chuck	1200
4	Drill the bushing hole, switch to low gear, ream hole.	Mill	Vise, Toe Clamp Fixture	19/64" drill bit, appropriate reamer drill chuck	1000,100
5	Repeat for other bushing hole.	Mill	Vise, Toe Clamp Fixture	19/64" drill bit, appropriate reamer drill chuck	1000,100
6	Center drill the tapped holes	Mill	Vise, Toe Clamp Fixture	#2 Center drill, drill chuck	1200
7	Drill the tapped holes	Mill	Vise, Toe Clamp Fixture	#25 drill bit, drill chuck	1400
8	Tap the tapped holes	Mill	Vise, Toe Clamp Fixture	10-24 tap, drill chuck	Hand
9	Deburr			Deburring tools	

SHEET 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Follower Link



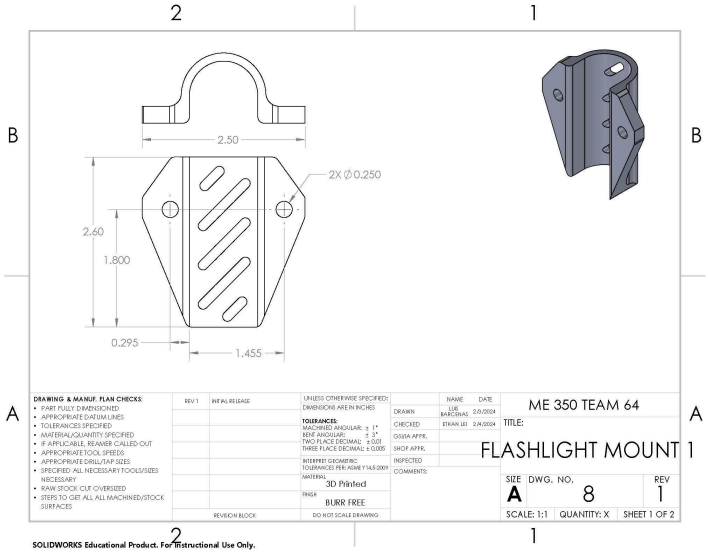
MANUFACTURING PLAN
RAW MATERIAL STOCK: XXXX

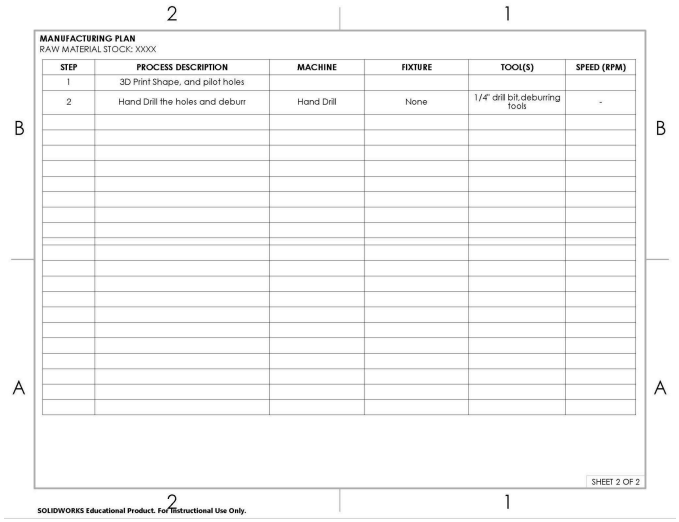
STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Waterjet	Waterjet			
2	Lay part on 1/4" side. Set x datum on leftmost side of link. Set y datum on bottom most side of link.	Mill	Toe Clamps	Edge finder, drill chuck	1000
3	Center drill bottom hole.	Mill	Toe Clamps	#5 center drill, drill chuck	1200
4	Center drill top hole	Mill	Toe Clamps	#2 center drill, drill chuck	1200
5	Drill top hole, and tap	Mill	Toe Clamps	#25 drill bit, drill chuck, 10-24 tap drill	1400, hands
6	Drill bottom hole	Mill	Toe Clamps	19/64" drill bit, drill chuck	1200
7	Go into low gear and ream hole	Mill	Toe Clamps	Appropriate reamer, reamer, drill chuck	100
8	Deburr			Deburring tools	

SHEET 2 OF 2

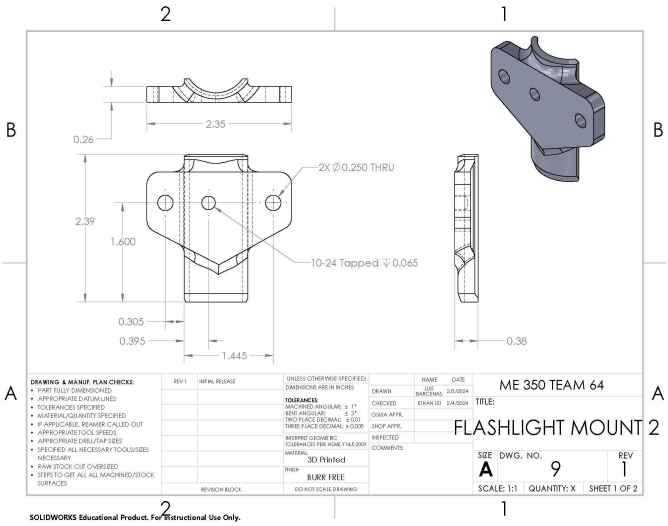
SOLIDWORKS Educational Product. For Instructional Use Only.

Bottom Flashlight Mount





Flashlight Mount



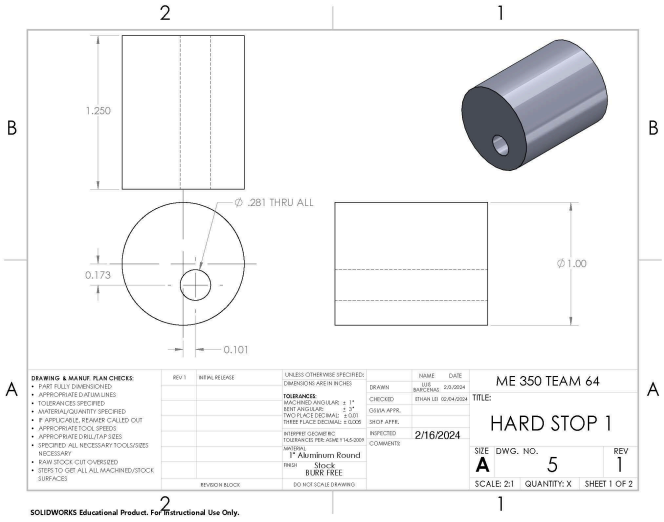
MANUFACTURING PLAN
RAW MATERIAL STOCK: XXXX

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	3D Print				
2	Hand drill 2 matching holes	Hand Drill	None	1/4" drill bit	-
3	Hand drill tapped hole	Hand Drill	None	#25 drill bit	
4	Tap hole	Drill Press	None	10-24 Tap drill	

SHEET 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Hard Stop 1



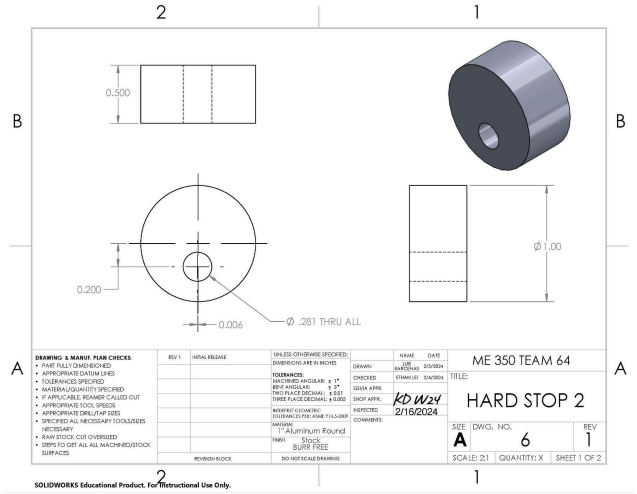
MANUFACTURING PLAN
RAW MATERIAL STOCK: XXXX

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Cut stock > 1/8" over final length then deburr	Horizontal Bandsaw	Vise	Deburring Tool	300 ft/min
2	Mount with 0.5" exposed and face one end to create flat surface, flip the part to face other end, measure then turn to length	Lathe	1" collet	Turning Tool, Caliper	750
3	Mount in mill vertically with square collet. Zero the mill on sides of square collet.	Mill	Vise	Square collet, edge finder	1000
4	Center drill hole on top face.	Mill	Vise	#5 center drill	1200
5	Drill thru hole	Mill	Vise	9/32 drill bit	1000
6	Deburr			Deburring tool	

SHEET 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Hard Stop 2

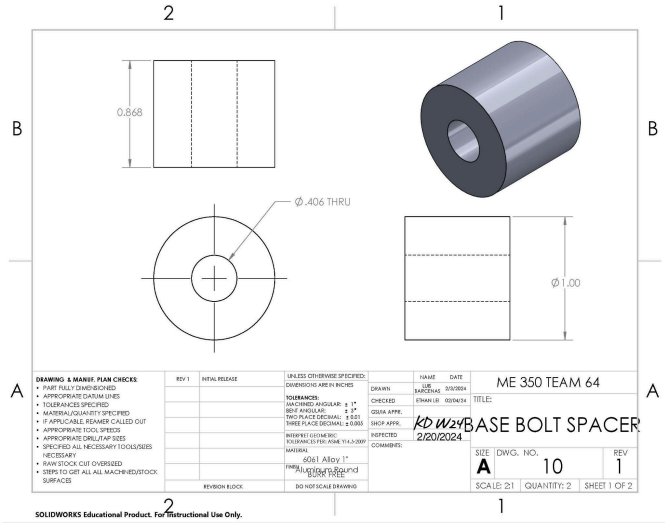


MANUFACTURING PLAN
RAW MATERIAL STOCK: XXXX

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Cut stock > 1/8" over final length then deburr	Horizontal Bandsaw	Vise	deburring tool	300 ft/s
2	Mount with 0.5" exposed and face one end to create flat surface, flip part to face other end, then turn to correct length	Lathe	1" collet	Turning tool, calipers	750
3	Mount in mill vertically with square collet. Zero the mill on sides of square collet.	Mill	Vise, square collet	edge finder	1000
4	Center drill on top face	Mill	Vise, square collet	#5 center drill	1200
5	Drill thru hole	Mill	Vise, square collet	9/32 Drill bit	1000
6	Deburr			Deburring tool	

SHEET 2 OF 2

Base Bolt Spacer



MANUFACTURING PLAN

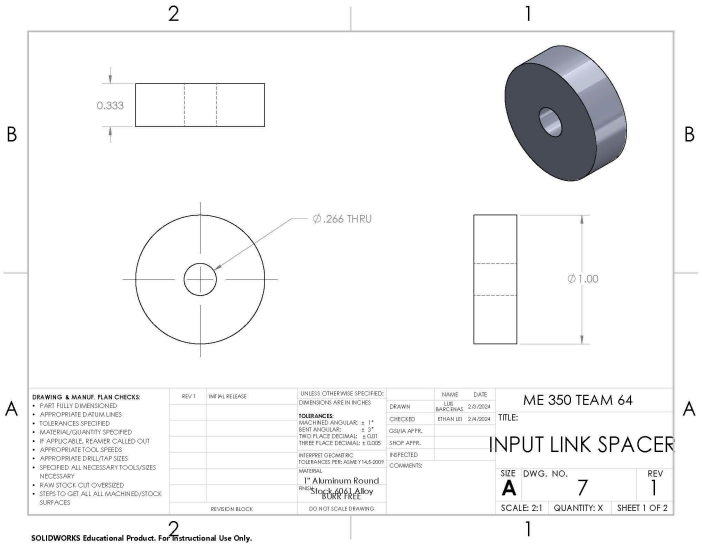
RAW MATERIAL STOCK: 1" Aluminum Round Stock

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Cut stock > 1/8" over final length then deburr	Horizontal Bandsaw	Vise	Deburring tool	300 ft/min
2	Mount with 0.5" exposed and face one end to create flat surface	Lathe	1" collet	Turning tool	750
3	Flip the stock and face the other end, measure and face to correct length	Lathe	1" collet	Turning tool, calipers	750
4	Center drill the hole in the top face	Lathe	1" collet	#5 center drill	1200
5	Drill the thru hole	Lathe	1" collet	13/32" drill bit, drill chuck	600
6	Deburr			deburring tool	

SHEET 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Input Link Spacer



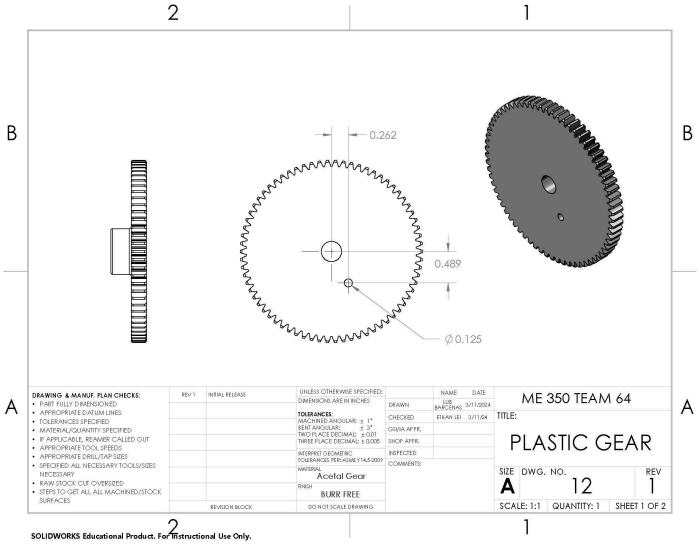
MANUFACTURING PLAN
RAW MATERIAL STOCK: XXXX

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Cut stock > 1/8" over final length then deburr	Horizontal Bandsaw	vise	Deburring tool	300 ft/min
2	Mount with 0.5" exposed and face one end to create flat surface	Lathe	1" collet	Turning tool	750
3	Flip stock and face other end.	Lathe	1" collet	Turning tool	750
4	Center drill the top face.	Lathe	1" collet	#3 Center drill, drill chuck	1200
5	Drill the hole. Deburr after.	Lathe	1" collet	17/64" Drill bit, drill chuck	1000 (17/64")

SHEET 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Plastic Gear



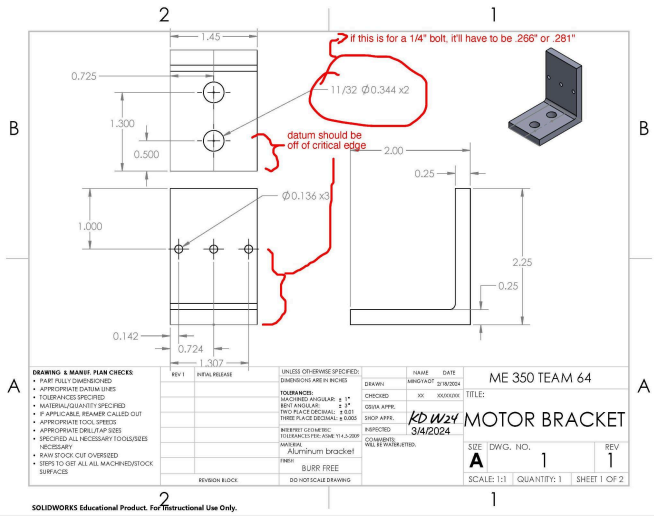
MANUFACTURING PLAN
RAW MATERIAL STOCK: Acetal Gear

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Clamp gear in place using provided gear fixture. Locate hole by datuming using fixture. Drill hole.	Mill	Gear fixture, vise	Edge finder, #3 center drill, 1/8" drill bit	1000,1200,1600

SHEET 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Motor Bracket



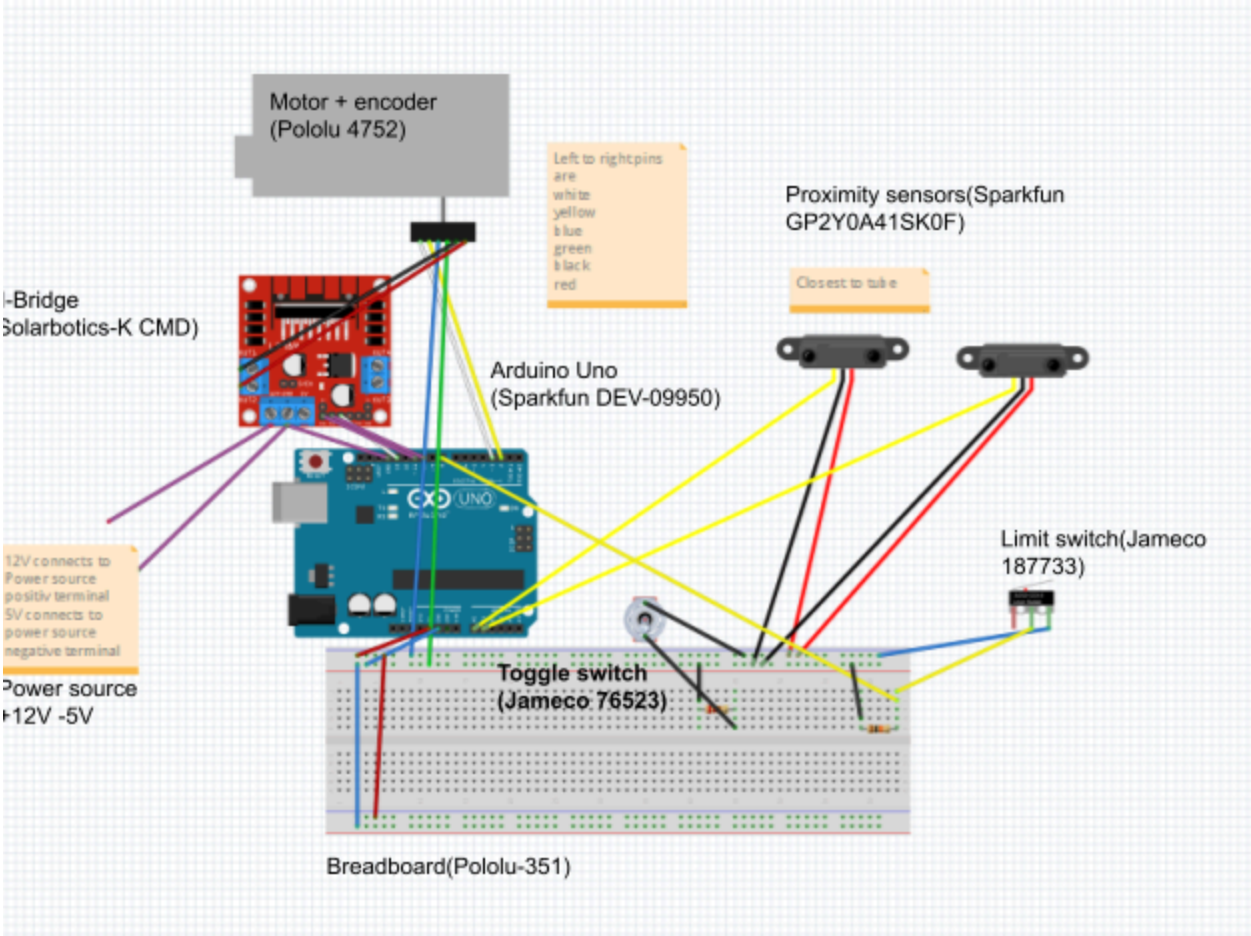
MANUFACTURING PLAN
RAW MATERIAL STOCK: XXXX

STEP	PROCESS DESCRIPTION	MACHINE	FIXTURE	TOOL(S)	SPEED (RPM)
1	Cut the angle to 0.2" over the width	Bandsaw			
2	Clamp the part on the vise, with one end extruding out. 2.25" side facing down	Mill	Vise		
3	Face the extruded side down to create a flat surface	Mill	Vise	extended 3/4" endmill ~~Face mill~~	~~500~~ 800
4	Measure the width of the part. Find the edge of the left most edge of the part and top most, zero x and y	Mill	Vise	1/2" collet, edgefinder, parallels, Calipers	1000
5	Mill the width of the angled bracket to the width required	Mill	Vise	1/2" collet, ~~3/8" Endmill~~ mill, parallels	~~1000~~
6	Remove ~~endmill and~~ collet and install drill chuck	Mill	Vise	#3 center drill, Drill Chuck, parallels	
7	Center drill and drill the 3 0.136" hole on the front view	Mill	Vise	#29 Drill bit, #3 center drill, Drill Chuck, parallels	1200, 1600
8	Flip the part so the other side faces down	Mill	Vise	Parallels	
9	Find the edge of the left most edge of the part and top most, zero x and y	Mill	Vise	1/2" collet, edgefinder	1000
10	Remove edge finder and collet and install drill chuck	Mill	Vise	#3 center drill, Drill Chuck, parallels	
11	Center drill and drill the 2 11/32 holes on the top view	Mill	Vise	11/32 Drill bit, Drill Chuck, parallels	1000
12	Remove the part and parallel. Debur.			Deburring tool	
	again, check your hole sizes				

SHEET 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Appendix D: Wiring Diagram, Arduino Code Changes



Arduino Code

```
// ME350 Target Dropping Sketch - Version 04.2 by Eli Parham
// updated 8-Apr-2024
//
//

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

```
//** State Machine: **//
// CONSTANTS:
// Definition of states in the state machine
const int CALIBRATE              = 1;
const int DETERMINE_ACTIVE_TARGETS = 2;
const int MOVE_TO_TARGET         = 3;

// VARIABLES:
// Global variable that keeps track of the state:
// Start the state machine in calibration state:
int state = CALIBRATE;
```

Changed to match the playground's proximity sensors' readings for each targets' flag. These are the values for playground 4, where we scored 28. They were calibrated to read only the target that is active at the time by manually lifting each target and reading and inputting the corresponding range of values.

```
//** Proximity Sensors or Potentiometer: **//
// CONSTANTS:
// Definition of proximity sensor thresholds for each target:
const int TARGET1_PROXIMITYSENSE1_MIN = 350;     // [proximity sensor counts] Min value prox sensor 1 reads when target 1 is active
const int TARGET2_PROXIMITYSENSE1_MIN = 145;     // [proximity sensor counts] Min value prox sensor 1 reads when target 2 is active
const int TARGET2_PROXIMITYSENSE1_MAX = 180;     // [proximity sensor counts] Max value prox sensor 1 reads when target 2 is active
const int TARGET3_PROXIMITYSENSE1_MIN = 75;     // [proximity sensor counts] Min value prox sensor 1 reads when target 3 is active
const int TARGET3_PROXIMITYSENSE1_MAX = 100;     // [proximity sensor counts] Max value prox sensor 1 reads when target 3 is active
const int TARGET3_PROXIMITYSENSE2_MIN = 90;     // [proximity sensor counts] Min value prox sensor 2 reads when target 3 is active
const int TARGET3_PROXIMITYSENSE2_MAX = 105;     // [proximity sensor counts] Max value prox sensor 2 reads when target 3 is active
const int TARGET4_PROXIMITYSENSE2_MIN = 150;     // [proximity sensor counts] Min value prox sensor 2 reads when target 4 is active
const int TARGET4_PROXIMITYSENSE2_MAX = 170;     // [proximity sensor counts] Max value prox sensor 2 reads when target 4 is active
const int TARGET5_PROXIMITYSENSE2_MIN = 380;     // [proximity sensor counts] Min value prox sensor 2 reads when target 5 is active
const int TARGETNONE_PROXIMITYSENSE1_MAX = 30;  // [proximity sensor counts] Max value prox sensor 1 reads when no target is active
const int TARGETNONE_PROXIMITYSENSE2_MAX = 75;  // [proximity sensor counts] Max value prox sensor 2 reads when no target is active
```

```
// Definition of target variables for array indexing
const int TARGET1  = 0;   // For indexing into activeTargets array, activeTargets[0] corresponds to Target 1
const int TARGET2  = 1;   // For indexing into activeTargets array, activeTargets[1] corresponds to Target 2
const int TARGET3  = 2;   // For indexing into activeTargets array, activeTargets[2] corresponds to Target 3
const int TARGET4  = 3;   // For indexing into activeTargets array, activeTargets[3] corresponds to Target 4
const int TARGET5  = 4;   // For indexing into activeTargets array, activeTargets[4] corresponds to Target 5
const int ACTIVE   = 1;   // In activeTargets array, a 1 means the corresponding target is active
const int INACTIVE = 0;   // In activeTargets array, a 0 means the corresponding target is inactive
const int UNSURE   = -1;  // In activeTargets array, a -1 means the corresponding target is not visible
// VARIABLES:
// Create variables that read and average proximity sensor or potentiometer readings:
int proximitySense1Values[20]; // Holds 20 proximity sensor readings for averaging
int proximitySense2Values[20]; // Holds 20 proximity sensor readings for averaging
float proximitySense1Avg = 0;  // Average of all values in proximitySense1Values
float proximitySense2Avg = 0;  // Average of all values in proximitySense2Values
int potValue = 0;              // Used to hold the reading of the potentiometer
// Variable to keep track of active targets:
int activeTargets[5] = {-1, -1, -1, -1, -1}; // Shows whether or not each of the five targets is active. -1 means the target is not visible.
                                             // 0 means the target is inactive. 1 means the target is active.
int activeTargetPosition = -1; // [encoder counts] Encoder count of the active target the linkage is moving toward

//** 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
const long MIN_VEL_COMP_TIME = 10000; // [microseconds] Minimal time that must pass between two velocity measurements
// VARIABLES:
volatile int motorPosition = 0; // [encoder counts] Current motor position (Declared 'volatile', since it is updated in a function called by interrupts)
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)
```

```
// The rightmost two bits of encoderStatus will store the encoder values from the current iteration (A and B).
// The two bits to the left of those will store the encoder values from the previous iteration (A_old and B_old).
float motorVelocity       = 0; // [encoder counts / seconds] Current motor velocity
int previousMotorPosition  = 0; // [encoder counts] Motor position the last time a velocity was computed
long previousVelCompTime   = 0; // [microseconds] System clock value the last time a velocity was computed
```

Changed to fit each trail and each individual playground. This is calibrated for playground 4. This was done by manually moving our linkage to each position after having clicked the limit switch to determine a reference point. Our wait position was the only arbitrary value that we decided should be target 3, as that is roughly the middle of the playground and would allow us to get to the other targets quicker.

```
//** High-level behavior of the controller:  **//
// CONSTANTS:
// Target positions:
const int CALIBRATION_VOLTAGE  = 5; // [Volt] Motor voltage used during the calibration process
const int TARGET_1_POSITION    = -2; // [encoder counts] Motor position corresponding to first target
const int TARGET_2_POSITION    = 320; // [encoder counts] Motor position corresponding to second target
const int TARGET_3_POSITION    = 635; // [encoder counts] Motor position corresponding to third target
const int TARGET_4_POSITION    = 900; // [encoder counts] Motor position corresponding to fourth target
const int TARGET_5_POSITION    = 1600; // [encoder counts] Motor position corresponding to fifth target
const int WAIT_POSITION        = TARGET_3_POSITION; // [encoder counts] Motor position corresponding to a wait position (when no targets are active)
const int LOWER_BOUND          = TARGET_1_POSITION; // [encoder counts] Position of the left end stop
const int UPPER_BOUND          = TARGET_5_POSITION; // [encoder counts] Position of the right end stop
const int TARGET_BAND          = 5; // [encoder counts] "Close enough" range when moving towards a target.
// Timing:
//const long  WAIT_TIME         = 0; // [microseconds] Time waiting for the target to drop.
                                  // TBD - implement a timer so if the target doesn't drop, the linkage moves to a different target
```

```
// VARIABLES:
//unsigned long startWaitTime; // [microseconds] System clock value
```

Changed gain and comp voltage values to ensure the linkage can point at the target accurately and quickly without delays or errors. These values were calibrated arbitrarily through trial and error and general knowledge of PID controllers and gains. Friction voltage was determined empirically by simply increasing the voltage until our linkage responded with minimal movement.

```
//** PID Controller  **//
// CONSTANTS:
const float KP                = 0.22; // [Volt / encoder counts] P-Gain
const float KI                = 0.01; // [Volt / (encoder counts * seconds)] I-Gain
const float KD                = 0.003; // [Volt * seconds / encoder counts] D-Gain
const float SUPPLY_VOLTAGE    = 10; // [Volt] Supply voltage at the HBridge
const float FRICTION_COMP_VOLTAGE = 1.4; // [Volt] Voltage needed to overcome friction
// VARIABLES:
int desiredPosition  = 0; // [encoder counts] desired motor position
float positionError  = 0; // [encoder counts] Position error
float integralError  = 0; // [encoder counts * seconds] Integrated position error
float velocityError  = 0; // [encoder counts / seconds] Velocity error
float desiredVoltage = 0; // [Volt] Desired motor voltage
int   motorCommand   = 0; // [0-255] PWM signal sent to the motor
unsigned long executionDuration = 0; // [microseconds] Time between this and the previous loop execution.  Variable used for integrals and derivatives
unsigned long lastExecutionTime = 0; // [microseconds] System clock value at the moment the loop was started the last time
```

Added

```
//** Pin assignment: **//
// CONSTANTS:
const int PIN_NR_ENCODER_A      = 2;  // Never change these, since the interrupts are attached to pins 2 and 3
const int PIN_NR_ENCODER_B      = 3;  // Never change these, since the interrupts are attached to pins 2 and 3
const int PIN_NR_ON_OFF_SWITCH   = 5;  // Connected to toggle switch (turns mechanism on and off)
const int PIN_NRL_LIMIT_SWITCH   = 8;  // Connected to limit switch (mechanism calibration)
const int PIN_NR_PWM_OUTPUT     = 11; // Connected to H Bridge (controls motor speed)
const int PIN_NR_PWM_DIRECTION_1  = 10; // Connected to H Bridge (controls motor direction)
const int PIN_NR_PWM_DIRECTION_2  = 13;  // Connected to H Bridge (controls motor direction)
const int PIN_PROXSENSE1        = A0; // Connected to proximity sensor 1
const int PIN_PROXSENSE2        = A1; // Connected to proximity sensor 2
```

```
const int PIN_POTENTIOMETER       = A2; // Connected to potentiometer used to test target
positions


// Add this line of code if you want to use two limit switches; const int
PIN_NRL_LIMIT_SWITCH_2  = 11
// ^KEEP IN MIND THAT YOU HAVE TO ADD CODE DOWNSTREAM (FOR EXAMPLE YOU NEED
TO ADD THIS VARIABLE IN THE DECLARATION SECTION

// End of CONSTANTS AND GLOBAL VARIABLES


////////////////////////////////////////////////////////////////////////////////
// The setup() function is called when a sketch starts. Use it to initialize variables, //
// pin modes, start using libraries, etc. The setup function will only run once, after  //
// each powerup or reset of the Arduino board:                                          //
////////////////////////////////////////////////////////////////////////////////
void setup() {
  // Declare which digital pins are inputs and which are outputs:
  pinMode(PIN_NR_ENCODER_A,       INPUT_PULLUP);
  pinMode(PIN_NR_ENCODER_B,       INPUT_PULLUP);
  pinMode(PIN_NR_ON_OFF_SWITCH,    INPUT);
  pinMode(PIN_NRL_LIMIT_SWITCH,    INPUT);
  pinMode(PIN_PROXSENSE1,         INPUT);
  pinMode(PIN_PROXSENSE2,         INPUT);
  pinMode(PIN_NR_PWM_OUTPUT,      OUTPUT);
  pinMode(PIN_NR_PWM_DIRECTION_1,  OUTPUT);
  pinMode(PIN_NR_PWM_DIRECTION_2,  OUTPUT);
  pinMode(PIN_POTENTIOMETER,      INPUT);

  // Turn on the pullup resistors on the encoder channels
  digitalWrite(PIN_NR_ENCODER_A, HIGH);
  digitalWrite(PIN_NR_ENCODER_B, HIGH);

  // Activate interrupt for encoder pins.
  // If either of the two pins changes, the function 'updateMotorPosition' is called:
  attachInterrupt(0, updateMotorPosition, CHANGE);  // Interrupt 0 is always attached to digital
pin 2
  attachInterrupt(1, updateMotorPosition, CHANGE);  // Interrupt 1 is always attached to digital
pin 3

  // Begin serial communication for monitoring.
  Serial.begin(115200);
```

```
  Serial.println("Start Executing Program.");

  // Set initial output to the motor to 0
  analogWrite(PIN_NR_PWM_OUTPUT, 0);
}
// End of function setup()


////////////////////////////////////////////////////////////////////////////////////////
// After going through the setup() function, which initializes and sets the initial values,   //
// the loop() function does precisely what its name suggests, and loops consecutively,       //
// allowing your program to sense and respond. Use it to actively control the Arduino board.  //
////////////////////////////////////////////////////////////////////////////////////////
void loop() {
  // Determine the duration it took to execute the last loop. This time is used
  // for integration and for monitoring the loop time via the serial monitor.
  executionDuration = micros() - lastExecutionTime;
  lastExecutionTime = micros();

  // Speed Computation:
  if ((abs(motorPosition - previousMotorPosition) > MIN_VEL_COMP_COUNT) || (micros() -
previousVelCompTime) > MIN_VEL_COMP_TIME){
    // If at least a minimum time interval has elapsed or
    // the motor has travelled through at least a minimum angle ...
    // .. compute a new value for speed:
    // (speed = delta angle [encoder counts] divided by delta time [seconds])
    motorVelocity = (double)(motorPosition - previousMotorPosition) * 1000000 /
                    (micros() - previousVelCompTime);
    // Remember this encoder count and time for the next iteration:
    previousMotorPosition = motorPosition;
    previousVelCompTime   = micros();
  }

  //*************************************************************************//
  // The state machine:
  switch (state) {
    //*************************************************************************//
    // In the CALIBRATE state, we move the mechanism to a position outside of the
    // work space (towards the limit switch).  Once the limit switch is on and
    // the motor has stopped turning, we know that we are against the end stop
    case CALIBRATE:
      // We don't have to do anything here since this state is only used to set
      // a fixed output voltage.  This happens further below.
```

```
      // Decide what to do next:
      if (digitalRead(PIN_NRL_LIMIT_SWITCH)==HIGH && motorVelocity==0) {
        // We reached the endstop.  Update the motor position to the limit:
        // (NOTE: If the limit switch is on the right, this must be UPPER_BOUND)
        motorPosition = UPPER_BOUND;  Changed since we have 1 limit switch and is mounted at the rightmost hard stop.
        // Reset the error integrator:
        integralError = 0;
        // Calibration is finalized. Transition into DETERMINE_ACTIVE_TARGETS state
        Serial.println("State transition from CALIBRATE to DETERMINE_ACTIVE_TARGETS");
        state = DETERMINE_ACTIVE_TARGETS;
      }
      // Otherwise we continue calibrating
      break;

    //*************************************************************************//
    // In the DETERMINE_ACTIVE_TARGETS state, we read the proximity sensors 10 times each, average
    // values to reduce noise, then use them to determine which targets are active.
    // We then select an active target to move toward based on which targets are active.
    // We default to TARGET_POSITION_3 if no targets are active.
    case DETERMINE_ACTIVE_TARGETS:
      // Reset active targets array to all UNSURE
      for (int i = 0; i < sizeof(activeTargets)/sizeof(int); i++) {
        activeTargets[i] = UNSURE;
      }
      // Reset activeTargetPosition to UNSURE
      activeTargetPosition = UNSURE;

      // Read and average 20 photosensor values
      for (int i = 0; i < sizeof(proximitySense1Values)/sizeof(int); i++) {
//        Serial.println(analogRead(PIN_PROXSENSE1));
        proximitySense1Values[i] = analogRead(PIN_PROXSENSE1);
      }
      for (int i = 0; i < sizeof(proximitySense2Values)/sizeof(int); i++) {
        proximitySense2Values[i] = analogRead(PIN_PROXSENSE2);
      }
      proximitySense1Avg = average(proximitySense1Values, sizeof(proximitySense1Values)/sizeof(int));
      proximitySense2Avg = average(proximitySense2Values, sizeof(proximitySense2Values)/sizeof(int));
```

```
    // Compare proximitySense1Avg and proximitySense2Avg to thresholds to determine which
targets are active
    if (proximitySense1Avg > TARGET1_PROXIMITYSENSE1_MIN) {
      activeTargets[TARGET1] = ACTIVE;    // Target 1 is active
      activeTargetPosition = TARGET_1_POSITION;
    }
    else if (proximitySense1Avg > TARGET2_PROXIMITYSENSE1_MIN && proximitySense1Avg <
TARGET2_PROXIMITYSENSE1_MAX) {
      activeTargets[TARGET1] = INACTIVE;  // Target 1 is inactive
      activeTargets[TARGET2] = ACTIVE;    // Target 2 is active
      activeTargetPosition = TARGET_2_POSITION;
    }
    else if (proximitySense1Avg > TARGET3_PROXIMITYSENSE1_MIN && proximitySense1Avg <
TARGET3_PROXIMITYSENSE1_MAX) {
      activeTargets[TARGET1] = INACTIVE;  // Target 1 is inactive
      activeTargets[TARGET2] = INACTIVE;  // Target 2 is inactive
      activeTargets[TARGET3] = ACTIVE;    // Target 3 is active
      activeTargetPosition = TARGET_3_POSITION;
    }
    else if (proximitySense1Avg < TARGET3_PROXIMITYSENSE1_MIN) {
      activeTargets[TARGET1] = INACTIVE;  // Target 1 is inactive
      activeTargets[TARGET2] = INACTIVE;  // Target 2 is inactive
      activeTargets[TARGET3] = INACTIVE;  // Target 3 is inactive
    }
    if (proximitySense2Avg > TARGET5_PROXIMITYSENSE2_MIN) {
      activeTargets[TARGET5] = ACTIVE;    // Target 5 is active
      // If activeTargetPosition has not already been changed, update it
      if (activeTargetPosition == -1) {
        activeTargetPosition = TARGET_5_POSITION;
      }
    }
    else if (proximitySense2Avg > TARGET4_PROXIMITYSENSE2_MIN && proximitySense2Avg <
TARGET4_PROXIMITYSENSE2_MAX) {
      activeTargets[TARGET5] = INACTIVE;  // Target 5 is inactive
      activeTargets[TARGET4] = ACTIVE;    // Target 4 is active
      // If activeTargetPosition has not already been changed, update it
      if (activeTargetPosition == -1) {
        activeTargetPosition = TARGET_4_POSITION;
      }
    }
    else if (proximitySense2Avg > TARGET3_PROXIMITYSENSE2_MIN && proximitySense2Avg <
TARGET3_PROXIMITYSENSE2_MAX) {
```

```
      activeTargets[TARGET5] = INACTIVE;  // Target 5 is inactive
      activeTargets[TARGET4] = INACTIVE;  // Target 4 is inactive
      activeTargets[TARGET3] = ACTIVE;    // Target 3 is active
    }
    else if (proximitySense2Avg < TARGET3_PROXIMITYSENSE2_MIN) {
      activeTargets[TARGET5] = INACTIVE;  // Target 5 is inactive
      activeTargets[TARGET4] = INACTIVE;  // Target 4 is inactive
      activeTargets[TARGET3] = INACTIVE;  // Target 3 is inactive
    }
    if (proximitySense1Avg < TARGETNONE_PROXIMITYSENSE1_MAX && proximitySense2Avg <
TARGETNONE_PROXIMITYSENSE2_MAX) {
      for (int i = 0; i < sizeof(activeTargets)/sizeof(int); i++) {
        activeTargets[i] = INACTIVE;  // All targets are inactive
      }
      activeTargetPosition = WAIT_POSITION; // Make linkage go to wait position if no targets are
active
    }

    // Read potentiometer to determine which target is active USED FOR TESTING PURPOSES
ONLY
//     potValue = analogRead(PIN_POTENTIOMETER);
//     Serial.print("potValue: ");
//     Serial.println(potValue);
//     if (potValue > 100 && potValue < 200) {
//       activeTargetPosition = TARGET_1_POSITION;
//     }
//     else if (potValue > 300 && potValue < 400) {
//       activeTargetPosition = TARGET_2_POSITION;
//     }
//     else if (potValue > 500 && potValue < 600) {
//       activeTargetPosition = TARGET_3_POSITION;
//     }
//     else if (potValue > 700 && potValue < 800) {
//       activeTargetPosition = TARGET_4_POSITION;
//     }
//     else if (potValue > 900 && potValue < 1000) {
//       activeTargetPosition = TARGET_5_POSITION;
//     }

    // Proceed to MOVE_TO_TARGET if we have a valid activeTargetPosition:
    if (activeTargetPosition != UNSURE) {
      state = MOVE_TO_TARGET;
    }
```

```
    // Otherwise, we stay in DETERMINE_ACTIVE_TARGETS
    break;

  //*********************************************************************************//
  // In the MOVE_TO_TARGET state, we select an active target and move toward it, or
  // move toward Target 3 (a default position) if there is no active target
  case MOVE_TO_TARGET:
    desiredPosition = activeTargetPosition;
    if (motorPosition <= activeTargetPosition + TARGET_BAND && motorPosition >=
activeTargetPosition - TARGET_BAND) {
      state = DETERMINE_ACTIVE_TARGETS;
    }
    break;

  //*********************************************************************************//
  // We should never reach the next bit of code, which would mean that the state
  // we are currently in doesn't exist.  So if it happens, throw an error and
  // stop the program:
  default:
    Serial.println("Statemachine reached at state that it cannot handle.  ABORT!!!!");
    Serial.print("Found the following unknown state: ");
    Serial.println(state);
    while (1); // infinite loop to halt the program
  break;
 }
 // End of the state machine.
 //*********************************************************************************//

 //*********************************************************************************//
 // Recalibrate if we are in the leftmost position
 if (digitalRead(PIN_NRL_LIMIT_SWITCH)==HIGH && motorVelocity==0) {
     // We reached the endstop.  Update the motor position to the limit:
     // (NOTE: If the limit switch is on the right, this must be UPPER_BOUND)
     motorPosition = UPPER_BOUND;
     // Reset the error integrator:
     integralError = 0;
 }


 //*********************************************************************************//
 // Position Controller
 if (digitalRead(PIN_NR_ON_OFF_SWITCH)==HIGH) {
  // If the toggle switch is on, run the controller:
```

```
//** PID control: **//
// Compute the position error [encoder counts]
positionError = desiredPosition - motorPosition;
// Compute the integral of the position error  [encoder counts * seconds]
integralError = integralError + positionError * (float)(executionDuration) / 1000000;
// Compute the velocity error (desired velocity is 0) [encoder counts / seconds]
velocityError = 0 - motorVelocity;
// This is the actual controller function that uses the error in
// position and velocity and the integrated error and computes a
// desired voltage that should be sent to the motor:
desiredVoltage = KP * positionError +
                 KI * integralError +
                 KD * velocityError;

//** Feedforward terms: **//
// Compensate for friction.  That is, if we now the direction of
// desired motion, add a base command that helps with moving in this
// direction:
if (positionError < -5) {
  desiredVoltage = desiredVoltage - FRICTION_COMP_VOLTAGE;
}
if (positionError > +5) {
  desiredVoltage = desiredVoltage + FRICTION_COMP_VOLTAGE;
}

// Anti-Wind-Up
if (abs(desiredVoltage)>SUPPLY_VOLTAGE) {
  // If we are already saturating our output voltage, it does not make
  // sense to keep integrating the error (and thus ask for even higher
  // and higher output voltages).  Instead, stop the integrator if the
  // output saturates. We do this by reversing the summation at the
  // beginning of this function block:
  integralError = integralError - positionError * (float)(executionDuration) / 1000000;
}
// End of 'if(onOffSwitch==HIGH)'

// Override the computed voltage during calibration.  In this state, we simply apply a
// fixed voltage to move against one of the end-stops.
if (state==CALIBRATE) {
  // add calibration code here
  desiredVoltage = CALIBRATION_VOLTAGE;
}
```

```
  } 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
```

These 2 lines are added to monitor sensor data, so we can solve the issue of linkage not detecting the correct target.

```
Serial.print(proximitySense1Avg);
Serial.println(proximitySense2Avg);

  //** 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
```

```
//*********************************************************************//


/////////////////////////////////////////////////////////////////////
// This is a function to update the encoder count in the Arduino.   //
// It is called via an interrupt whenever the value on encoder      //
// channel A or B changes.                                          //
/////////////////////////////////////////////////////////////////////
void updateMotorPosition() {
  // Bitwise shift left by one bit, to make room for a bit of new data:
  encoderStatus <<= 1;
  // Use a compound bitwise OR operator (|=) to read the A channel of the encoder (pin 2)
  // and put that value into the rightmost bit of encoderStatus:
  encoderStatus |= digitalRead(2);
  // Bitwise shift left by one bit, to make room for a bit of new data:
  encoderStatus <<= 1;
  // Use a compound bitwise OR operator  (|=) to read the B channel of the encoder (pin 3)
  // and put that value into the rightmost bit of encoderStatus:
  encoderStatus |= digitalRead(3);
  // encoderStatus is truncated to only contain the rightmost 4 bits by  using a
  // bitwise AND operator on mstatus and 15(=1111):
  encoderStatus &= 15;
  if (encoderStatus==2 || encoderStatus==4 || encoderStatus==11 || encoderStatus==13) {
    // the encoder status matches a bit pattern that requires counting up by one
    motorPosition++;        // increase the encoder count by one
  }
  else if (encoderStatus == 1 || encoderStatus == 7 || encoderStatus == 8 || encoderStatus == 14)
{
    // the encoder status does not match a bit pattern that requires counting up by one.
    // Since this function is only called if something has changed, we have to count downwards
    motorPosition--;        // decrease the encoder count by one
  }
}
// End of function updateMotorPosition()


/////////////////////////////////////////////////////////////////////
// This function sends a status of the controller to the serial     //
// monitor.  Each character will take 85 microseconds to send, so   //
// be selective in what you write out:                              //
/////////////////////////////////////////////////////////////////////
void printStateToSerial() {
  //*********************************************************************//
```

```
  // Send a status of the controller to the serial monitor.
  // Each character will take 85 microseconds to send, so be selective
  // in what you write out:

  //Serial.print("State Number:  [CALIBRATE = 1; DETERMINE_ACTIVE_TARGETS = 2;
MOVE_TO_TARGET = 3]: ");
  Serial.print("State#: ");
  Serial.print(state);

  //Serial.print("Power switch [on/off]: ");
  //Serial.print("  PWR: ");
  //Serial.print(digitalRead(PIN_NR_ON_OFF_SWITCH));

  //Serial.print("      Motor Position [encoder counts]: ");
  Serial.print("  MP: ");
  Serial.print(motorPosition);

  //Serial.print("      Motor Velocity [encoder counts / seconds]: ");
  Serial.print("  MV: ");
  Serial.print(motorVelocity);

  //Serial.print("      Encoder Status [4 bit value]: ");
  //Serial.print("  ES: ");
  //Serial.print(encoderStatus);

  //Serial.print("      Target Position [encoder counts]: ");
  Serial.print("  DP: ");
  Serial.print(desiredPosition);

  //Serial.print("      Position Error [encoder counts]: ");
  Serial.print("  PE: ");
  Serial.print(positionError);

  //Serial.print("      Integrated Error [encoder counts * seconds]: ");
  Serial.print("  IE: ");
  Serial.print(integralError);

  //Serial.print("      Velocity Error [encoder counts / seconds]: ");
  Serial.print("  VE: ");
  Serial.print(velocityError);

  //Serial.print("      Desired Output Voltage [Volt]: ");
  Serial.print("  DV: ");
```

```
  Serial.print(desiredVoltage);

  //Serial.print("      Motor Command [0-255]: ");
  //Serial.print("  MC: ");
  //Serial.print(motorCommand);

  //Serial.print("      Execution Duration [microseconds]: ");
  //Serial.print("  ED: ");
  //Serial.print(executionDuration);

  //Serial.print("      Active Targets [-1,0,1]: ");
  Serial.print("  AT: ");
  for (int i = 0; i < sizeof(activeTargets)/sizeof(int); i++) {
    Serial.print(activeTargets[i]);
    Serial.print(" ");
  }

//  Serial.print("  PS2: ");
//  Serial.print(proximitySense2Avg);

  // ALWAYS END WITH A NEWLINE.  SERIAL MONITOR WILL CRASH IF NOT
  Serial.println(); // new line
}
// End of Serial Out

////////////////////////////////////////////////////////////////////
// This function returns the average of the integer array         //
// pointed to by array_ptr and of length len.                     //
////////////////////////////////////////////////////////////////////
float average (int * array_ptr, int len) {
  long sum = 0L;
  for (int i = 0; i < len; i++) {
    sum += array_ptr[i];
  }
  return ((float) sum)/len;
}
// End of average
```